COMP31212: Concurrency

Topics 4.2: Modelling and Analyzing Interference
Topic 4.2: Modelling and Analyzing Interference in FSP

Outline

- Interference
- A GUI Example
- FSP Modelling of the Garden
- Modelling Locks in FSP
Outline

Topic 4.2: Modelling and Analyzing Interference in FSP

Interference
A GUI Example
FSP Modelling of the Garden
Modelling Locks in FSP
Interference - summary

- When 2 or more threads have access to the same object, there is potential for interference between the actions on the object.
- Destructive update caused by arbitrary interleaving of read and write actions is interference.
- Interference leads to unexpected and undesirable behaviour.
- The Ornamental Garden (from Magee and Kramer) example is a good demonstration of the general problem.
- We will model and analyze the problem in FSP.
Outline

Topic 4.2: Modelling and Analyzing Interference in FSP

- Interference
- A GUI Example
- FSP Modelling of the Garden
- Modelling Locks in FSP
GUI Example: Ornamental Garden

How Many Visitors are in The Garden??
Desired Behaviour

- Again, two processes reading and updating a common object
- Complexity due to GUI . . .
- . . . and persuading the code/scheduler to Do The Wrong Thing!!!
Oh Dear!! ... It’s Gone Wrong
Ornamental Garden: Class Structure

Applet

Garden
init()
go()

Thread

Turnstile
run()
east, west

counterD

NumberCanvas
setvalue()
eastD,
westD,

counterD
display

Counter
increment()
display

garden

people
public class Garden extends Applet {

    private void go() {
        counter = new Counter(counterD);
        west = new Turnstile(westD, counter);
        east = new Turnstile(eastD, counter);
        west.start();
        east.start();
    }
}
The Counter Class

class Counter {

    int value=0;
    NumberCanvas display;

    Counter(NumberCanvas n) {
        display=n;
        display.setvalue(value);
    }

    void increment() {
        int temp = value; //read[v]
        Simulate.HWinterrupt();
        value=temp+1; //write[v+1]
        display.setvalue(value);
    }
}

The Turnstile Class

class Turnstile extends Thread {
    NumberCanvas display; Counter people;

    Turnstile(NumberCanvas n, Counter c) {
        display = n; people = c;
    }

    public void run() {
        try{
            display.setvalue(0);
            for (int i=1; i<=Garden.MAX; i++){
                Thread.sleep(500); // 0.5 second
                display.setvalue(i);
                people.increment();
            }
        } catch (InterruptedException e) {} 
    }
}
Simulation of Hardware Interruption

class Simulate {
    public static void HWinterrupt() {
        if (Math.random()<0.5)
            try{
                Thread.sleep(200);
            } catch(InterruptedException e){};
    }
}
Outline

Topic 4.2: Modelling and Analyzing Interference in FSP

- Interference
- A GUI Example
- FSP Modelling of the Garden
- Modelling Locks in FSP
Ornamental Garden: FSP Model

\[
\begin{align*}
\text{const } N &= 4 \quad \text{range } T = 0..N \\
\text{set } \text{VarAlpha} &= \{\text{value.}\{\text{read}[T], \text{write}[T]\}\}
\end{align*}
\]

\[
\begin{align*}
\text{VAR} &= \text{VAR}[0], \\
\text{VAR}[u: T] &= (\text{read}[u] \rightarrow \text{VAR}[u] \mid \text{write}[v: T] \rightarrow \text{VAR}[v]).
\end{align*}
\]

\[
\begin{align*}
\text{TURNSTILE} &= (\text{go} \rightarrow \text{RUN}), \\
\text{RUN} &= (\text{arrive} \rightarrow \text{INCREMENT} \mid \text{end} \rightarrow \text{TURNSTILE}), \\
\text{INCREMENT} &= (\text{value.read}[x: T] \rightarrow \text{value.write}[x + 1] \rightarrow \text{RUN} \mid \text{VarAlpha}).
\end{align*}
\]

\[
\begin{align*}
\|\text{GARDEN} &= (\text{east : TURNSTILE} \| \text{west : TURNSTILE} \\
&\| \{\text{east, west, display}\} :: \text{value : VAR}) \\
&/\{\text{go/\{east, west\}.go}, \text{end/\{east, west\}.end}\}.
\end{align*}
\]
Exhaustive Analysis

\[
\begin{align*}
\text{TEST} & = \text{TEST}[0], \\
\text{TEST}[v:T] & = (\text{when } (v<N)\{\text{east.arrive, west.arrive}\} \rightarrow \text{TEST}[v+1] \\
& \quad | \text{end}\rightarrow \text{CHECK}[v] \\
& \big), \\
\text{CHECK}[v:T] & = (\text{display.value.read}[u:T] \rightarrow \\
& \quad (\text{when } (u==v) \right \rightarrow \text{TEST}[v] \\
& \quad | \text{when } (u!=v) \right \rightarrow \text{ERROR}) \\
& \big)+\{\text{display.}\{\text{VarAlpha}\}\}.
\end{align*}
\]

Now compose TEST in parallel with the GARDEN process

\[
||\text{TESTGARDEN} = (\text{GARDEN} || \text{TEST}).
\]

The act of building the above composition will check all possible runs of the GARDEN.
And we see it fails.
Outline

Topic 4.2: Modelling and Analyzing Interference in FSP

- Interference
- A GUI Example
- FSP Modelling of the Garden
- Modelling Locks in FSP
An FSP Model for Locks

\[ \text{LOCK} = (\text{acquire} \rightarrow \text{release} \rightarrow \text{LOCK}). \]

\[ ||\text{LOCKVAR} = (\text{LOCK} || \text{VAR}). \]

\[ \text{set VarAlpha} = \{\text{value} \{\text{read}[T], \text{write}[T], \text{acquire}, \text{release}\}\} \]

\[ \text{INCREMENT} = (\text{value} \cdot \text{acquire} \rightarrow \text{value} \cdot \text{read}[x:T] \rightarrow \text{value} \cdot \text{write}[x+1] \rightarrow \text{value} \cdot \text{release} \rightarrow \text{RUN}) + \text{VarAlpha}. \]
Include in the FSP GARDEN model and ..

||GARDEN = (east:TURNSTILE || west:TURNSTILE
   || east,west,display::value:LOCKVAR)
   /{go /{east,west}.go,
   end/{east,west}.end}.

and re-test

||TESTGARDEN = (GARDEN || TEST).

And the composition occurs without ERRORs.
Synchronized Counter

For the Java version, create an extension of the Counter that has a synchronised increment() method.
Synchronized Counter

For the Java version, create an extension of the Counter that has a synchronised increment() method.

class SynchronizedCounter extends Counter {

    SynchronizedCounter(NumberCanvas n) {
        super(n);

        synchronized void increment() {
            super.increment();
        }
    }
}
Synchronized Counter

For the Java version, create an extension of the Counter that has a synchronised increment() method.

class SynchronizedCounter extends Counter {

    SynchronizedCounter(NumberCanvas n) {
        super(n);
    }

    synchronized void increment() {
        super.increment();
    }
}

and use SynchronizedCounter in place of Counter.
Synchronized Counter

For the Java version, create an extension of the Counter that has a synchronised increment() method.

```java
class SynchronizedCounter extends Counter {

    SynchronizedCounter(NumberCanvas n) {
        super(n);

        synchronized void increment() {
            super.increment();
        }
    }
}
```

and use SynchronizedCounter in place of Counter.
(This is used in the applet when the “fix it” checkbox is checked)