Concurrency Assurance in Fluid

Related reading: ***Assuring and Evolving Concurrent Programs: Annotations and Policy***

15-413:
Introduction to Software Engineering
Jonathan Aldrich

Find the Concurrency Bug!

```
public class Logger {
    private Filter filter;

    public void setFilter(Filter newFilter)
            throws SecurityException {
        if (!anonymous) manager.checkAccess();
        filter = newFilter;
    }

    public void log (LogRecord record) { …
        synchronized (this) {
            if (filter != null && !filter.isLoggable(record))
                return;
        }
    }
}
```

PREfix: Language-Level Errors

- Error defined by language
  - Precise characterization of error
  - Any program that manifests that error is incorrect
  - Easy to define fully automated analysis
- Example: null pointer dereference
  - Occurs when *p is executed and p == null
  - Can be found by tracking which pointers may be null

Concurrency Errors

- Example: data race condition
  - (Definition from Savage et al., *Eraser: A Dynamic Data Race Detector for Multithreaded Programs*)
  - Two threads access the same variable v
  - At least one access is a write
  - No explicit mechanism prevents the accesses from being simultaneous

Concurrency Errors

- Example: data race condition
  - (Definition from Savage et al., *Eraser: A Dynamic Data Race Detector for Multithreaded Programs*)
  - Two threads access the same variable v
  - At least one access is a write
  - No explicit mechanism prevents the accesses from being simultaneous
- Challenges
  - Difficult to check statically
    - How to tell if accesses can be simultaneous?
    - How to tell what synchronization mechanism is used?
  - Not always an error
    - Race may not affect correctness
- PREfix approach will not work
  - Too many possibilities to explore, too many false positives

Would Testing/Inspections Work?

Would Testing/Inspections Work?

- Testing
  - Difficult because concurrency errors are non-deterministic
- Inspections
  - Concurrency errors are often non-local
    - Like errors that PREfix finds
  - Require knowledge of programmer intent

Fluid: Models are missing

- **Programmer design intent** is missing
  - Not explicit in Java, C, C++, *etc*
    - What lock protects this object?
      - ***This lock** protects **that state***
    - What is the actual extent of shared state of this object?
      - ***This object** is "part of" that object*
- Adoptability
  - Programmers: "Too difficult to express this stuff."
  - Fluid: Minimal **effort** — concise expression
    - Capture what programmers are **already thinking about**
    - No full specification
- Incrementality
  - Programmers: "I'm too busy; maybe after the deadline."
  - Fluid: **Payoffs** early and often
    - Direct programmer utility – *negative marginal cost*
    - Increments of payoff for increments of effort

Reporting Code–Model Consistency

Tool analyzes model/code consistency

- No model ⇒ no assurance
- Identify likely model sites

Three classes of results

- Code–model consistency

- Code–model inconsistency

- Informative — Request for annotation

BoundedFIFO

```
public class BoundedFIF0 {
//@aggregate [] into Instance
//@unshared
LoggingEvent[] buf;

//@ lock BufLock is this protects Instance

int numElts = 0, first = 0, next = 0, size;

public BoundedFIF0(int size) { ... }

//@ requires BufLock
public LoggingEvent get() {
    if(numElts == 0) return null;
    LoggingEvent r = buf[first];
    if(++first == size) first = 0;
    numElts--;
    return r;
}

//@ requires BufLock
public void put(LoggingEvent o) {
    if(numElts != size) {
        buf[next] = o;
        if(++next = = size) next = 0;
            numElts++;
    }
}

//@ requires BufLock
public int getMaxSize() {
    return size;
}

// no annotation required
public synchronized void resize(int
    newSize) { ... }
...
}
```

BoundedFIFO Client

```
public class FIF0Client {
    private final BoundedFIFO fifo = ...;
    ...
    public void putter(LoggingEvent e) {
      synchronized(fifo) {
          while(fifo.isFullO) {
              try { fifo.wait(); }
              catch(InterruptedExn ie) {}
          }
          fifo.put(e);
          if(fifo.wasEmptyO) fifo.notify();
      }
    }
```

```
public LoggingEvent getter() {
    synchronized(fifo) {
      LoggingEvent e;
      while(fifo.length() == 0) {
          try { fifo.wait(); }
          catch(InterruptedExn ie) { }
      }
      e = fifo.get();
      if(fifo.wasFull()) fifo.notify();
      return e ;
    }
}

public int length() {
    synchronized(fifo) { return fifo.length();
    }
}
```

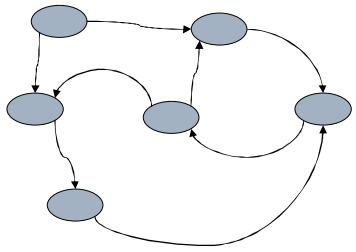

Logger Revisited

```
/** @lock FilterLock is this protects filter */
public class Logger {
    private Filter filter;

    public void setFilter(Filter newFilter)
            throws SecurityException {
        if (!anonymous) manager.checkAccess();
        filter = newFilter;
    }

    public void log (LogRecord record) { ...
        synchronized (this) {
            if (filter != null && !filter.isLoggable(record))
                return;
        }
    }
}
```

How Incrementality Works

- Incrementality
  - When you annotate a portion of the program, you'll get immediate feedback on concurrency correctness
  - Incremental benefit for each unit of cost
- How can one provide incremental benefit with mutual dependencies?

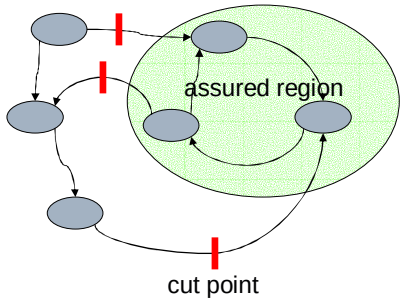
Call Graph of Program



How Incrementality Works

- Incrementality
  - When you annotate a portion of the program, you'll get immediate feedback on concurrency correctness
  - Incremental benefit for each unit of cost
- How can one provide incremental benefit with mutual dependencies?
- Cut points
  - Method annotations partition call graph
  - Can assure property of a subgraph
  - Assurance is *contingent* on accuracy of trusted cut point method annotations

Call Graph of Program



Questions for Evaluating Tools

- What class of errors does the tool find?
  - And can that class be found with other techniques?
- Can the tool miss errors?
- How many false errors does it report?
- Can I run it on part of a system?
- How much manual effort is required?
- Does it find errors across procedure boundaries?
- Does it scale to large systems?