Verifying Library Code for Concurrent Access

Bernhard Beckert
Vladimir Klebanov
vladimir@uni-koblenz.de

June 15, 2007

What?

Verifying concurrent Java programs

In KeY

Calculus Properties

Full reasoning about data

Beyond just safety or race detection

No abstractions

java.lang.StringBuffer

```
private char value[];
private int count;

public synchronized StringBuffer
                        append(char c) {
    int newcount = count + 1;
    if (newcount > value.length)
        expandCapacity(newcount);
    value[count++] = c;
    return this;
}
```

Verify That...

$$\begin{gathered}
\texttt{strb.<lockcount>} = 0 \land \neg \texttt{strb} = \texttt{null} \land \texttt{strb.count} = 0 \rightarrow \\
\forall n.\ n > 0 \rightarrow \\
\langle^{\{n\}} \texttt{strb.append(c);}^{\{0\}} \rangle \texttt{strb.count} = n \land \\
\forall k.\ 0 \leq k < n \rightarrow \texttt{strb.value}[k] = \texttt{c}(p_1(k+1))
\end{gathered}$$

Three-Step Programme

❶ Unfold

❷ Prove atomicity invariant

❸ Symbolic execution + induction

Statistics

- Proof steps: 14622
- Branches: 238 (3 relevant)
- Interactions: 2
- Runtime: $\sim$1 minute
- Result:

Statistics

- Proof steps: 14622
- Branches: 238 (3 relevant)
- Interactions: 2
- Runtime: $\sim$1 minute
- Result: conjecture false

Concurrency Verification Problems

- Number of threads
  ↪ symmetry reduction (this work)

- Number of interference points
  ↪ exploit locking, data confinement

- Java Memory Model
  ↪ ?

Alas. . .

No thread identities **in** programs

No dynamic thread creation (but unbounded concurrency)

Only atomic loops

Symbolic Execution (Sequential)

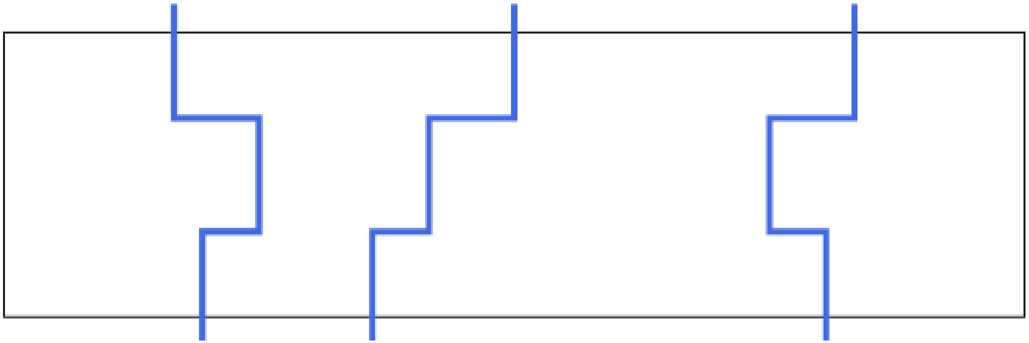
IN:

Assertion about program

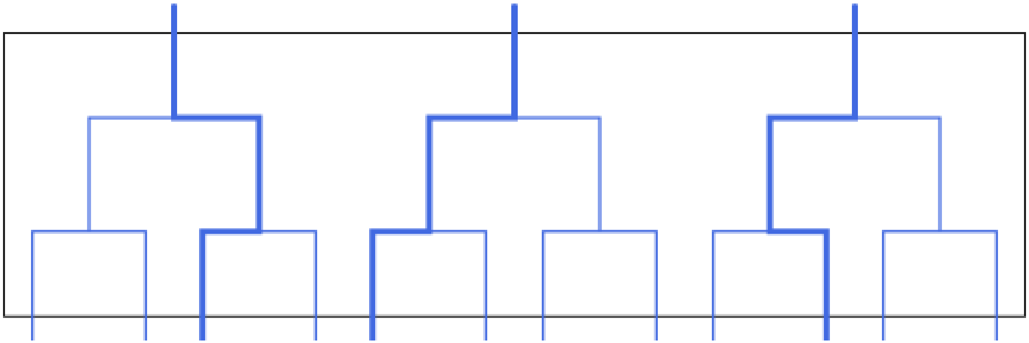
OUT:

FOL over $\mathbb{Z}$

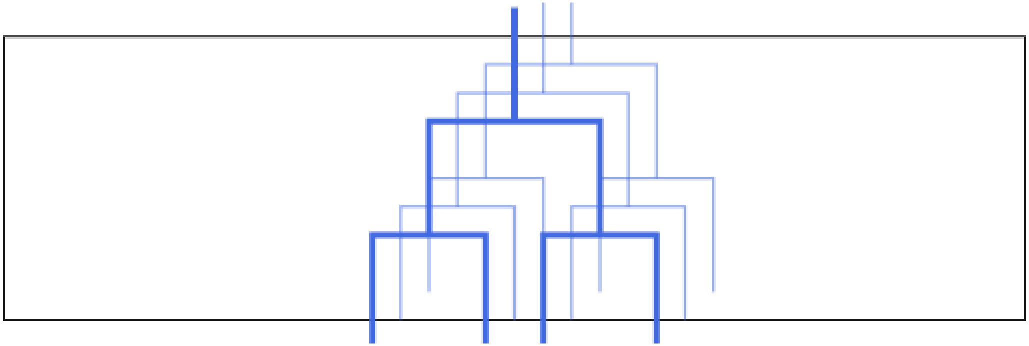
Concurrent Programs



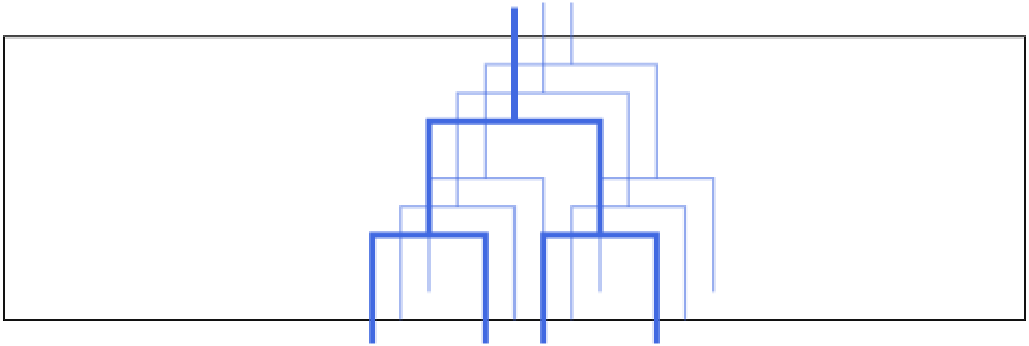
Concurrent Programs



Concurrent Programs



Enter the Scheduler



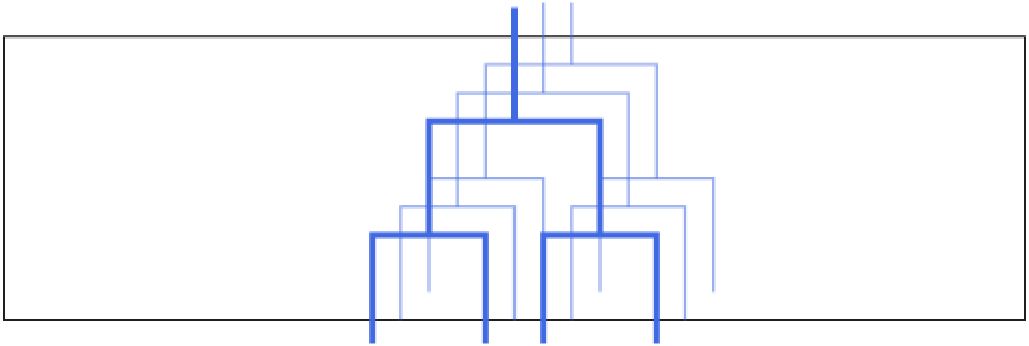
p(n)

. . .

p(2)

p(1)

Enter the Scheduler



$$\$(p(1)) + \$(p(2)) + \ldots + \$(p(n))$$

Concurrent Symbolic Execution

IN:

Assertion about program

OUT:

FOL over $\mathbb{Z}$ with
scheduler function

Concurrent Symbolic Execution

OUT:

FOL over $\mathbb{Z}$ with
scheduler function

$$\sum_{i=1}^{n} \$(\mathrm{i}) = \sum_{i=1}^{n} \$(\mathrm{p}(\mathrm{i}))$$

So What Does It Mean?

$$\text{step}\ \frac{\begin{array}{r}
\Longrightarrow P(r, c) = pos \\
path(pos, p) \Longrightarrow \langle\!\![S^{*(pos)}]\!\!\rangle \langle\!\![r|\ \pi^{\{p_{pos:n-1}\}}\, S^{\{p_{pos+1:k+1}\}}\ \omega]\!\!\rangle \phi \\
\neg path(pos, p) \Longrightarrow \langle\!\![r|\ \pi^{\{p_{pos:n-1}\}}\, S^{\{p_{pos+1:k+1}\}}\ \omega]\!\!\rangle \phi
\end{array}}{\Longrightarrow \langle\!\![r|\ \underbrace{\pi^{\{p_{pos:n}\}}\, S^{\{p_{pos+1:k}\}}\ \omega}_{=p}]\!\!\rangle \phi}$$

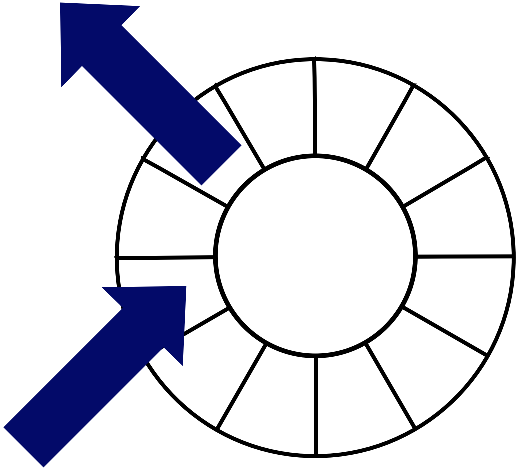
and where pos is the position of S in p

So What Does It Mean?

Proofs have fewer cases than programs inputs

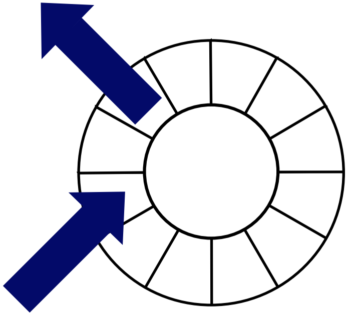
Good scheduler formalization takes you far

Next Proof



Blocking concurrent queue

Next Proof



$$\mathtt{q.<lockcount>} = 0 \land \neg \mathtt{q} = \mathtt{null} \land \mathtt{q.list.size} = 0 \rightarrow$$

$$\forall n.\ n > 0 \rightarrow \langle^{\{n\}}\mathtt{q.put(in);}^{\{0\}}\mathtt{||}^{\{n\}}\mathtt{out=q.get();}^{\{0\}}\rangle$$

$$\forall k.\ 1 \leq k \leq n \rightarrow \mathtt{out}(p_r(k)) = \mathtt{in}(p_a(k))$$

Conclusion

First deductive proof
of full functional correctness
of production Java code
in concurrent setting.

Thanks

Questions?

TOC