Towards Deductive Verification of Concurrent Java Software

Vladimir Klebanov
vladimir@uni-koblenz.de

June 9, 2005

Microcalculus

- Captures the execution semantics
- Good things will happen

Extending DL For Concurrency

1. Concurrent modalities
2. Causal tokens and places
3. New implicit object fields
   - `int <lockCount>;`
   - `int <waiting>;`
4. New rules

A Concurrent Diamond

An example:

$$\langle \{\bullet\}\hat{}t_1; \parallel \{\bullet\bullet\}\hat{}sta_1;\hat{}\{\ \}\hat{}sta_2;\hat{}\{\ \}\hat{}sta_3;\rangle\phi$$

may evolve to:

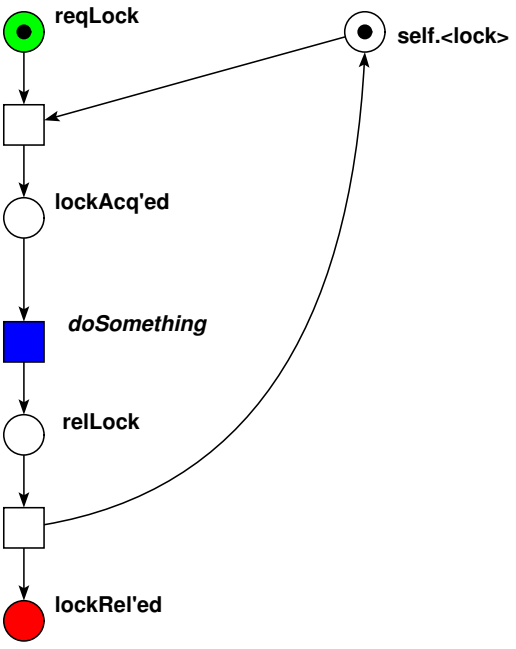
$$\langle \{\bullet\}\hat{}t_1; \parallel \{\ \bullet\ \}\hat{}sta_1;\hat{}\{\bullet\}\hat{}sta_2;\hat{}\{\ \}\hat{}sta_3;\rangle\phi$$

or to...

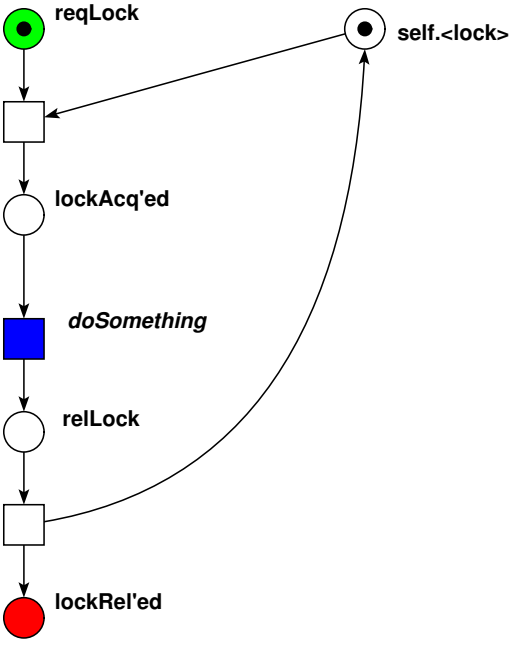
Thread Attributes

- Identity
  - `t.interrupt()`
  - `t.join()`
- “Program counter”
- Locks held
- Working memory contents

Synchronized Methods

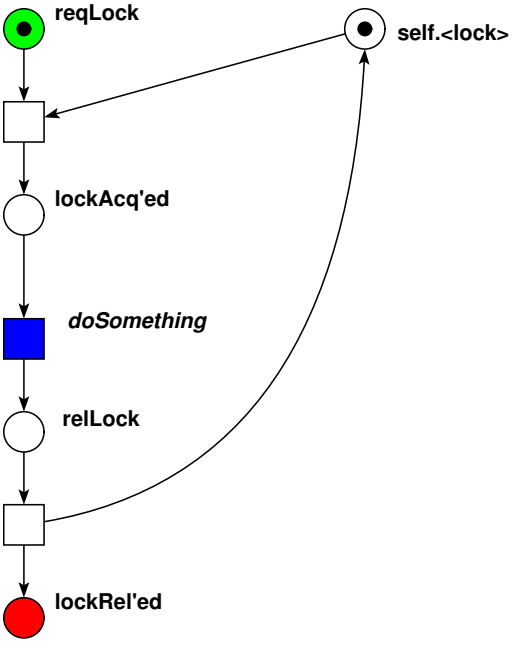


Synchronized Methods




```
if (<lockCount>==0) {
  {self.<lock>}^ ;
}
<lockCount>++;
```

Synchronized Methods




```
if (<lockCount>==0) {
  {self.<lock>}^ ;
}
<lockCount>++;
```

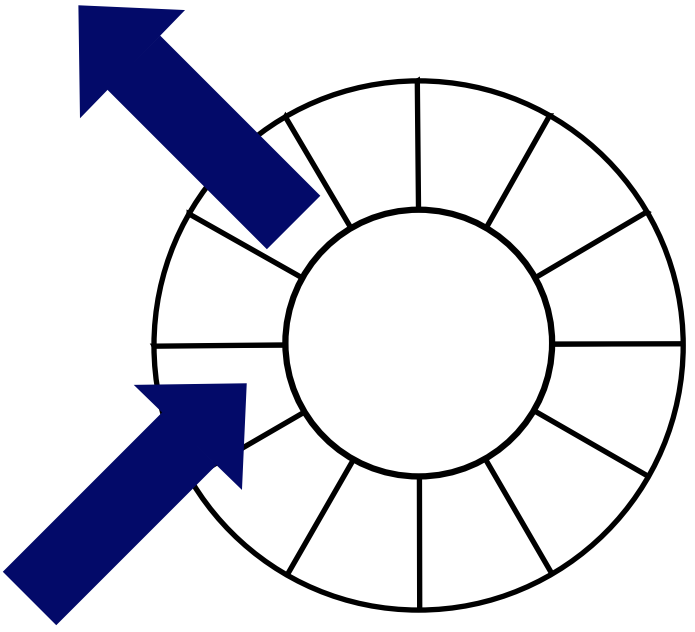
```
if (<lockCount>==1) {
  ; ^{self.<lock>}
}
<lockCount>--;
```

Waiting and Notification

```
public void wait() {
  int l = lockCount();
  unlock(l);

  <waiting>++;
  {notif-thisref} ^ ;
  <waiting>--;

  lock(l);
}
```

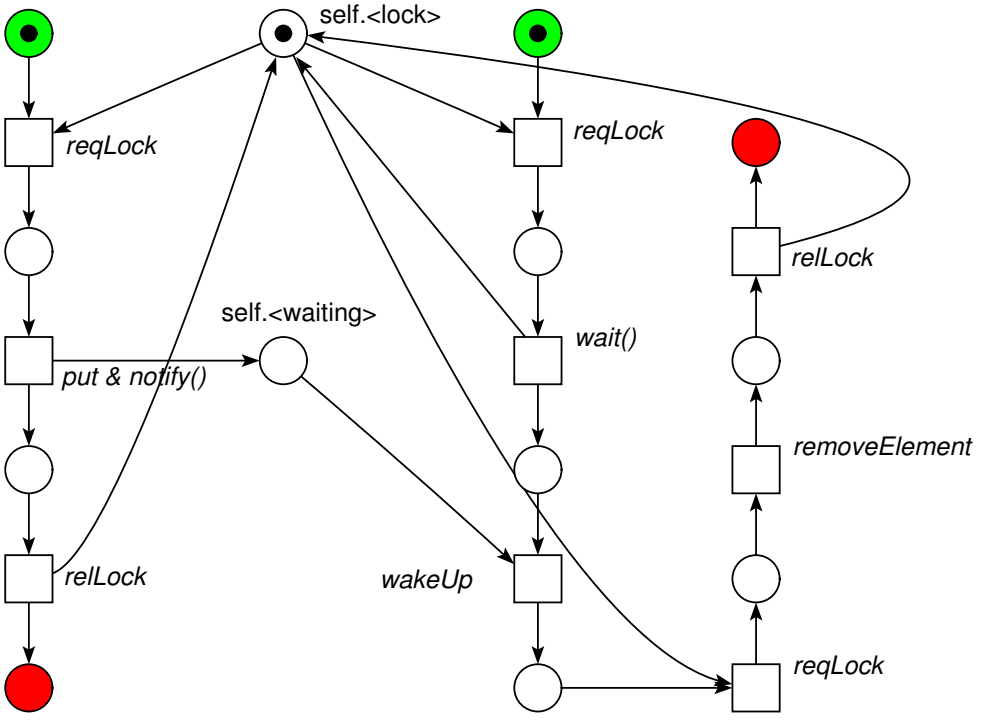
```
public void notify() {



  if (<waiting> > 0) {
    ; ^{notif-thisref}
  }
  // else do nothing

}
```

Example: Blocking Concurrent Queue

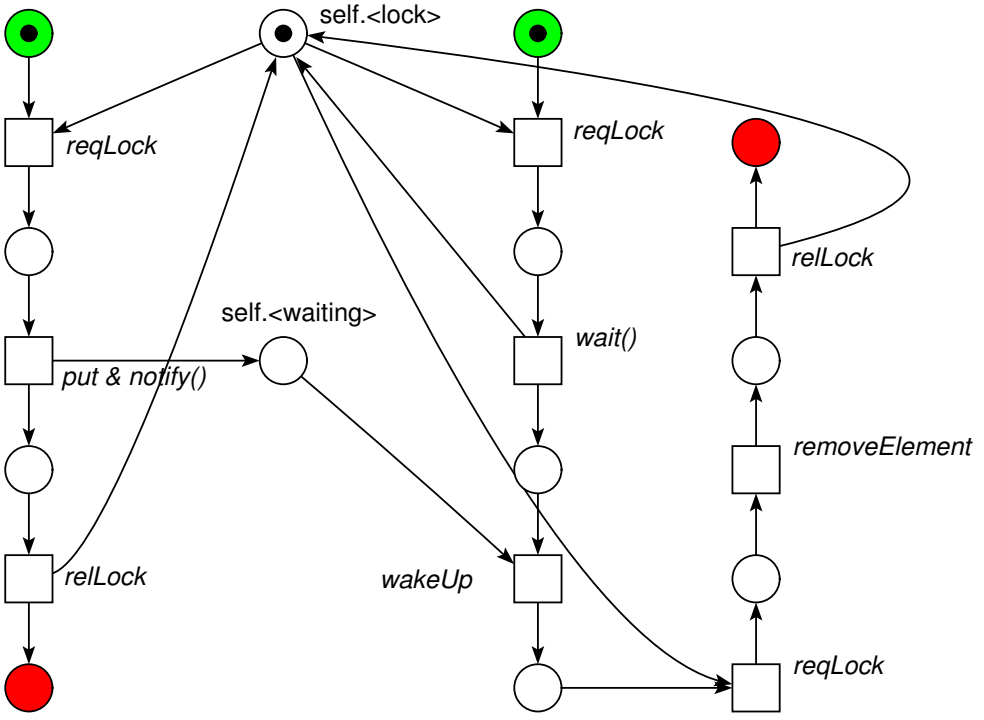


Property: “1 consumer thread, 1 producer thread; whatever is put, is eventually retrieved”

Example: Concurrent Queue



Example: Concurrent Queue



$\langle \{\bullet\}\texttt{\^{}q.put(x);} \parallel \{\bullet\}\texttt{\^{}y=q.get();} \rangle x = y$

Concurrent Queue In Context

$\langle$`{●}^q.put(x);` $\|$ `{●}^y=q.get();`$\rangle x = y$

- "Denial of service" possible?
- Implementation correct for several consumers?
- Livelock possible?

Accomodating the Java Memory Model

Coming soon...

Macrocalculus

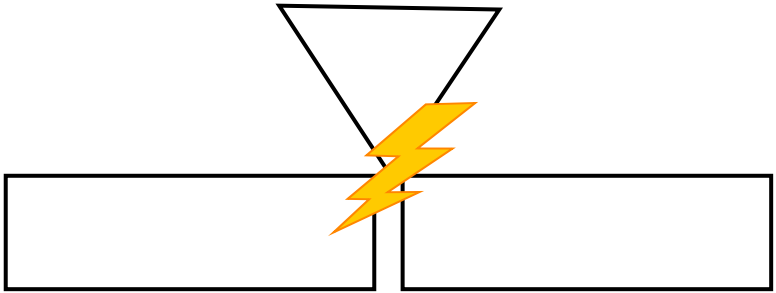
- Modularity
- Design by Contract
- Assumptions about the world
- No bad things will happen

A Quest for Modularity

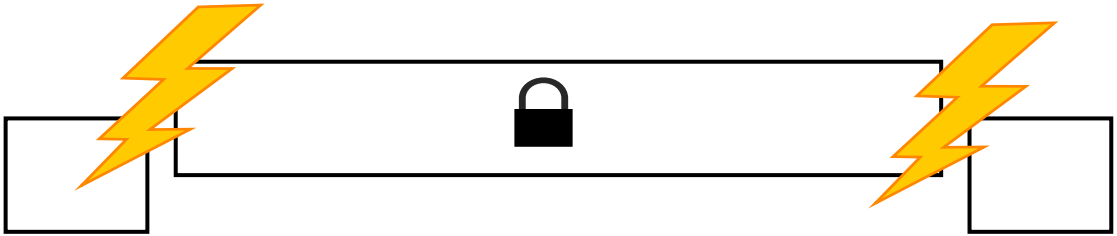
Conventional specifications cannot serve as behavioral abstraction in presence of concurrency.

Contract Fulfillment Endangered

Internal interference



External interference



Good News (At Last)

Developers think in terms of **serialization**

Concurrent execution $\cong$ serial execution

[Greenhouse et al.][Robby et al.]

Serializability

...is achieved by

- locking
- data confinement

State Protection (Locking) Specification

- We have: The **modifies** clause (of a method)
- We add: A **reads** clause
- Sugar: Annotations to delineate/aggregate state regions inside and across objects
- Finally add:
  - Mapping locks $\leftrightarrow$ regions of state
  - Locking policy (locks acquired or required)

Then method specifications can be used for verification.

(almost...)

State Protection Example

```
/**@ lock BufLock is this protects Instance */
public class BoundedFIFO {

    /** @aggregate [] into Instance
     *  @unshared */
    Object[] buf;

    /** @requiresLock BufLock
        @pre ...
        @post ...  */
    public void put(Object o) {...}
}
```

Data Confinement

Objects can be declared `\thread_local`. Thread-local objects are not subject to interference.

How do we check this?

Serializability

One basic flavor (stated as a meta-rule):

$$\frac{\langle p_1 \rangle \phi_1 \qquad \langle p_2 \rangle \phi_2 \qquad \Phi_1 \ldots \Phi_n}{\langle p_1 \parallel p_2 \rangle \phi_1 \wedge \phi_2} \text{ par\_comp}$$

[Vladimir Klebanov: *A JMM-Faithful Non-Interference Calculus for Java.* FIDJI 2004 Workshop on Scientific Engineering of Distributed Java App's.]

To Be Done

Develop and implement further criteria of **serializability**.

Thank You!

Questions?

TOC