Quis Custodiet Ipsos Custodes
The Java memory model
Andreas Lochbihler

theorem drf:
  assumes sync: "correctly_synchronized P E"
  and legal: "legal_execution P E (E, ws)"
  shows "sequentially_consistent P (E, ws)"
using legal_wf_execD[OF legal] legal_ED[OF legal] sync
proof(rule drf lemma)
  fix r
  assume "r ∈ read_actions E"

  from legal obtain J where E: "E ∈ E"
  and wf_exec: "P ⊢ (E, ws) √"
  and J: "P ⊢ (E, ws) justified_by J"
Quis custodiet ipsos custodes?

Joana: information flow control for Java

leak

Ifc
Quis custodiet ipsos custodes?

non-interference
program slicing
control flow graph
formal semantics for Java

Joana: information flow control for Java

Quis custodiet: verify IFC algorithm

λ → ∀ = Isabelle
β α

Joana
IFC

leak

How can we include Java concurrency?
Quis custodiet ipsos custodes?

Joana: information flow control for Java

Quis custodiet: verify IFC algorithm

How can we include Java concurrency?
The Java memory model (JMM): beyond interleaving semantics

initially: $x = y = 0$;

$x = 1$; \hspace{1cm} $y = 2$;

$j = y$; \hspace{1cm} $i = x$;
initially: $x = y = 0$;

$x = 1$; $y = 2$

$j = y$; $i = x$

interleaving semantics

<table>
<thead>
<tr>
<th>$j == 0$</th>
<th>$j == 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i == 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$i == 1$</td>
</tr>
</tbody>
</table>
The Java memory model (JMM): beyond interleaving semantics

initially: $x = y = 0$;

- $x = 1$
- $j = y$
- $j = y$
- $i = x$

interleaving semantics

- $j = 0$
- $j = 2$

- $i = 0$
- $i = 1$
- $\sqrt{}$
The Java memory model (JMM): beyond interleaving semantics

initially: \( x = y = 0; \)
\[
\begin{align*}
x &= 1; \\
j &= y; & y &= 2; \\
i &= x;
\end{align*}
\]

interleaving semantics
\[
\begin{array}{c|c|c}
\text{j} & \text{j} \equiv 0 & \text{j} \equiv 2 \\
\text{i} & \text{i} \equiv 0 & \checkmark \\
\text{i} & \text{i} \equiv 1 & \checkmark \\
\end{array}
\]
The Java memory model (JMM): beyond interleaving semantics

initially: \(x = y = 0\);

\[
\begin{array}{c}
x = 1; \quad y = 2; \\
j = y; \quad i = x;
\end{array}
\]

interleaving semantics

<table>
<thead>
<tr>
<th>(j == 0)</th>
<th>(j == 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i == 0)</td>
<td>(\checkmark)</td>
</tr>
<tr>
<td>(i == 1)</td>
<td>(\checkmark)</td>
</tr>
</tbody>
</table>
The Java memory model (JMM): beyond interleaving semantics

initially: \( x = y = 0 \);

\[
x = 1;\quad y = 2;\quad i = x;
j = y;\quad i = x;
\]

interleaving semantics

\[
\begin{array}{c|cc}
                      & j == 0 & j == 2 \\
\hline
i == 0          & \times & \checkmark \\
i == 1          & \checkmark & \checkmark \\
\end{array}
\]
The Java memory model (JMM): beyond interleaving semantics

initially: $x = y = 0$;

$\begin{array}{c}
\text{x = 1;} \\
\text{j = y;} \\
\text{i = x;}
\end{array}$

interleaving semantics

<table>
<thead>
<tr>
<th></th>
<th>$j = 0$</th>
<th>$j = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$</td>
<td>$X$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$i = 1$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

compiler and hardware reorder statements

$\begin{array}{c}
\text{j = y;} \\
\text{x = 1;} \\
\text{i = x;} \\
\text{y = 2;}
\end{array}$

$\begin{array}{c}
\text{j = 0} \\
\text{i = 0} \\
\text{i = 1}
\end{array}$
The Java memory model (JMM): beyond interleaving semantics

initially: \( x = y = 0 \);

\[
\begin{align*}
&x = 1; &y = 2; \\
j = y; &i = x;
\end{align*}
\]

Java memory model

\[
\begin{array}{c|cc}
& j == 0 & j == 2 \\
\hline
i == 0 & \checkmark & \checkmark \\
i == 1 & \checkmark & \checkmark \\
\end{array}
\]

compiler and hardware reorder statements

\[
\begin{align*}
&j = y; &i = x; \\
x = 1; &y = 2;
\end{align*}
\]

\[
\begin{array}{c|cc}
& j == 0 & j == 2 \\
\hline
i == 0 & \checkmark \\
i == 1 & \\
\end{array}
\]
The Java memory model (JMM): beyond interleaving semantics

data races

initially: \( x = y = 0; \)
\begin{align*}
x &= 1; \\
j &= y; \\
y &= 2; \\
i &= x;
\end{align*}

compiler and hardware reorder statements

\begin{align*}
j &= y; \\
x &= 1; \\
i &= x; \\
y &= 2;
\end{align*}

Java memory model

\begin{align*}
\begin{array}{c|cc}
\text{j} & \text{j} = 0 & \text{j} = 2 \\
\hline
\text{i} & \text{i} = 0 & \text{i} = 2 \\
\text{i} & \text{i} = 1 & \text{i} = 2
\end{array}
\end{align*}
Problems with data races under the JMM

1. Data races allow time travel.

   - r1 may contain `input()`.
   - Time-sensitive analyses do not cover that.

```
public = x;  x = y;

y = secret;
```

```java
if (b) r2 = new A(); else r3 = new A();
y = r3;
b = true;
```

Initially, `x = y = null; b = false;`
Problems with data races under the JMM

1. Data races allow time travel.

- \( r_1 \) may contain \texttt{input()}.
- Time-sensitive analyses do not cover that.
Problems with data races under the JMM

1. Data races allow time travel.

- r1 may contain `input()`.  
- Time-sensitive analyses do not cover that.

2. Values appear out of thin air.

- r2 and r3 may alias,  
- but typical points-to analyses compute: They never alias.
Results

**Unified model** of Java and the JMM
First formal link between Java and the Java memory model

**Proofs** about the JMM

**DRF guarantee:**
- Programs without data races *behave* as if executed under interleaving semantics
- Most program analyses assume interleaving semantics
  ⇒ Sound for DRF programs

**Type safety:**
- Java’s type safety extends to the JMM, even if there are data races.
- Crucial for CFG construction (no undefined behaviour)
What is a data race?

Data race:
- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$
What is a data race?

Data race:
- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_h b$

Synchronisation via by spawning threads:

initially: $x = \text{new Thread()}; \ y = 0;$

\[
\begin{align*}
\quad y &= 1; \\
\quad x.start(); \\
\quad \text{try} \ {\ x.start();} \\
\quad \text{try} \ {\ x.start();} \\
\quad \text{catch} \ \{ \ \text{IllegalThreadStateException} \ _\} \ \{ \ r = y; \ \}
\end{align*}
\]
What is a data race?

Data race:
- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:

```
initially: x = new Thread(); y = 0;

y = 1; try { x.start(); x.start(); } catch (IllegalThreadStateException _) { r = y; }
```

data race?
What is a data race?

Data race:
- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:

```
initially: x = new Thread(); y = 0;

y = 1;
x.start();
try { x.start(); } catch (IllegalThreadStateException _) { r = y; }
```

data race?

intuition: no  JMM: yes
What is a data race?

Data race:
- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:

```java
initially: x = new Thread(); y = 0;

y = 1; try { x.start(); x.start(); } catch (IllegalThreadStateException _) { r = y; }
```

disallowed synchronisation

data race?