Data races in Java and static analysis

Frédéric Dabrowski

\(^1\)INRIA, LANDE
1. Concurrency in Java

2. Static Detection of dataraces
   - lock-based typing
     - Flanagan, Abadi & Freund
     - Boyapati, Lee & Rinard
   - Points-to analysis: Naik & Aiken
     (Points-to analysis + Type and effect system)

3. Conclusion and ongoing work
Concurrency in Java
Concurrency model

- *Thread-based* concurrency: shared memory (fields of shared objects)
- Lexically scoped *locking* construct: `synchronized(x){...}
- *Preemptive* scheduling (Interleaving semantics)
Interleaving semantics

(small step) sequential semantics

\[ t, \text{Mem} \rightarrow_{\text{seq}} t', \text{Mem}' \]

interleaving semantics

\[ t_i, \text{Mem} \rightarrow_{\text{seq}} t'_i, \text{Mem}' \]

\[ \{\ldots, t_i, \ldots\}, \text{Mem} \rightarrow_{\text{inter}} \{\ldots, t'_i, \ldots\}, \text{Mem}' \]

Problem: This semantics is incomplete with respect to the Java Memory Model, **unless you write well-synchronized programs**
Concurrenty in Java

Natural hypothesis: sequential consistency

Intuivively, **sequential consistency** means that all executions respect the program order.

```java
void mn()
{
    a // a should not observe b
    ...
    b
}
```

**Problem:** enforcing sequential consistency for all Java programs makes many of the compiler/processor optimizations illegal.

**Why?** some optimizations assume well-synchronized programs!
Concurrent in Java

Example: code reordering (cache mechanisms, ...)

<table>
<thead>
<tr>
<th>Original code</th>
<th>Optimized code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C.f = C.g = 0$</td>
<td>$C.f = C.g = 0$</td>
</tr>
<tr>
<td>$1: x = C.g$; $3: y = C.f$</td>
<td>$2: C.f = 1$; $4: C.g = 1$</td>
</tr>
<tr>
<td>$2: C.f = 1$; $4: C.g = 1$</td>
<td>$1: x = C.g$; $3: y = C.f$</td>
</tr>
<tr>
<td>${Perm(1, 2, 3, 4) \mid 1 &lt; 2, 3 &lt; 4}$</td>
<td>${Perm(1, 2, 3, 4) \mid 2 &lt; 1, 4 &lt; 3}$</td>
</tr>
</tbody>
</table>

Frédéric Dabrowski (INRIA, LANDE)
Concurrency in Java

Java’s memory model is weak memory model

All executions of well-synchronized programs are sequentially consistent \(^a\).

\(^a\)Manson, Pugh & Adve : The Java Memory Model (Special Popl issue)

Programs **must** be well-synchronized

several static analysis depend on it
Concurrency in Java

Well-synchronized programs

\((P_1)\) : For all execution (w.r.t the interleaving semantics), every conflicting actions \(a\) and \(b\) are synchronized

\[
\text{synchronized}(C)\{ \\
(1 : x = C.g); (2 : C.f = 1) \}
\text{synchronized}(C)\{ \\
(3 : y = C.f); (4 : C.g = 1) \}
\]

compiler/jvm/jit : \((P_1) \Rightarrow \) every exec. is captured by the inter. sem.
Happens-before relation

≺_{hb} is the transitive closure of the following rules:

- **sequentiality** \( a^t \prec_{hb} 1 b^t \)
- **start/join synchronisation**
  \[
  \begin{align*}
  t.\text{start}() & \prec_{hb} 1 t.\text{join}() \\
  a^t & \prec_{hb} 1 t.\text{join}()
  \end{align*}
  \]
- **lock-based synchronisation**
  \[
  \begin{align*}
  \text{unlock}(m) & \prec_{hb} 1 \text{lock}(m) \\
  \text{write}(x.f) & \prec_{hb} 1 \text{read}(x.f) \quad (f \text{ volatile})
  \end{align*}
  \]
Data races

**definition (JMM)**: a program has a data race if there exists an execution with two conflicting actions not ordered by $\prec_{hb}$.

**alternative definition**: a program has a data race if there exists an execution such that, at some point, there is a non-deterministic choice (interleaving semantics) among two conflicting actions.

**Problem (undecidable)**: Given a program, are all executions of that program race free?
Static detection of data races
RCC Java

[PLDI’00] Type-based race detection for Java (Flanagan and Freund)

- supports classes parameterized by locks of given types (Dependent types)
- introduce a notion of thread local classes
- Fields protected by locks (static fields)
- Encapsulation: self-protected class
- Extend previous work based on a simple thread calculus
Example

class A<ghost Object x>{
  Object y = new Object() guarded_by x;
  void set(Object z) requires x{
    this.y = z;
  }
}

class B{
  final Object z = new Object();
  A<this.z> x = new A<this.z>();
  void f(){
    synchronized(this.z){set(new Object())}
  }
}
Ownership types

[OOPSLA’02] Ownership types for safe programming : preventing data races and deadlocks (Boyapati, Lee and Rinard)

Basic idea

Write generic code and create different instance with different protection mechanisms.
Ownership types

Each object is owned by

\[
\begin{cases}
\text{another object (final field)} \\
\text{itself (self)} \\
\text{a thread (thisThread)}
\end{cases}
\]

\[
\text{class } C\langle \text{Owner}_0, \text{Owner}_1, \ldots \rangle \quad \{ \ldots \text{new } D\langle \text{Owner}_1 \rangle() \ldots \}
\]
Ownership types

Static analysis

1. The ownership relation builds a forest of trees
2. The fields of an object must be protected by the ancestor of this object (a root, i.e. an object protected by itself or a thread)

Extensions

- support for read-only/single pointer objects
Ownership types

Example

class Account⟨thisOwner⟩{
    int balance = 0;
    void deposit(int x) requires thisOwner{
        this.balance = this.balance + x;
    }
}

Account ⟨thisThread⟩ a1 = new Account⟨thisThread⟩
a1.deposit(10);

final Account⟨self⟩ a2 = new Account⟨self⟩;
fork{ synchronized(a2){ a2.deposit(10); } }

Account⟨a2⟩ a3 = new Account⟨a2⟩;
Type inference

[VMCAI’04] Type Inference for Parameterized Race-Free Java (Agarwal and Stoller)

idea:

- Perform a set of execution
- Extract types from this set
- Check types

problem: incomplete

[SAS’04, SCP’07] Type inference against races (Flanagan and Freund)

- consider parameterization of classes as introduced by Boyapati, Lee and Rinard
- by reduction of the problem of finding a satisfying assignment for a boolean formula (NP-complete)
Bytecode analysis

[Sigplan Not.] A type system for preventing data races and deadlocks in the java virtual machine language
(Permandla, Roberson, Boyapati)

**problem**: monitorenter/monitorexit replace synchronized blocks

**solution**: use indexed types to recover structured locking

Indexed types: [TCS’03] A type system for JVM threads (Laneve)

very simple alias analysis

\[ i : \text{Load } n, \{ \ldots n \mapsto \tau \ldots \}, \text{stack} \rightarrow \{ \ldots n \mapsto \tau_i \ldots \}, \tau_i :: \text{stack} \]
Limitations of type-based approaches

- very strict lock-based discipline
- can’t handle other synchronization patterns
Naik and Aiken & al
\[ a = \text{new } h_1[N]; \]
\[
\text{for}(i = 0; i < N; i ++)\{
    a[i] = \text{new } h_2;
    a[i].f = \text{new } h_3;
\}
\]

\[
\text{while }(*)\{
    x = a[*];
    \text{fork}\{
        \text{sync}(x)\{x.f.g = *; \}
    \}
\}
\]
Language

\[
\begin{align*}
s & ::= \quad | \ x = \text{null} \ | \ x = \text{new} \ h \\
 & \quad | \ x = y \ | \ x = y.f \ | \ x.f = y \\
 & \quad | \ s_1; s_2 \ | \ \text{if} \ (\ast) \ \text{then} \ s_1 \ \text{else} \ s_2 \ | \ \text{while}^w (\ast) \ \text{do} \ s
\end{align*}
\]

\(h\) allocation site

\(w \in W\) loop counter

Dynamic semantics

\[
\begin{align*}
\text{Obj} & ::= \langle h, \pi \rangle \quad \pi : W \to N \\
\text{C} & ::= \{ \ldots \text{Obj} \triangleright \text{Obj}' \ldots \}
\end{align*}
\]
Conditional Must Not Aliasing

\[
\text{synchronized}(x)\{ \begin{array}{c}
x.f.g = *; \\
\end{array} \}
\quad \text{synchronized}(y)\{ \begin{array}{c}
y.f.g = *; \\
\end{array} \}
\]

Conditional Must Not Aliasing

\[\text{must_not_alias}(x, y) \Rightarrow \text{must_not_alias}(x.f, y.f)\]
Disjoint Reachability

Disjoint reachability

\[ h \in DR_C(H) \iff \begin{cases} 
\overline{o_1}.h \in H \land (\overline{o_1} \triangleright \overline{o}) \in C^+ \land \\
\overline{o_2}.h \in H \land (\overline{o_2} \triangleright \overline{o}) \in C^+ \land \\
\overline{o}.h = h
\end{cases} \Rightarrow \overline{o_1} = \overline{o_2} \]

Abstraction

\[
\begin{align*}
Obj & ::= \langle \hat{h}, \Pi \rangle \\
\hat{h} & ::= h \mid \top \\
\Pi & : \mathcal{W} \leftrightarrow \mathbb{N}_\top \\
\mathbb{N}_\top & = \{0, 1, \top\}
\end{align*}
\]

Judgments : \( \mathcal{W}, \Pi, \Gamma \vdash s : \Gamma', K \)

\[ h \in DR_K(H) \quad DR_K(H) \text{ is a safe appr. of } DR_C(H) \]
Disjoint reachability

Examples

\[ h_2 \in DR_K(\{h_1\})? \]

\[
\begin{array}{lll}
\text{while}^1(*)\{ \\
\text{while}^2(*)\{ \\
\text{while}^1(*)\{ \\
\}
\end{array}
\]

\[
\begin{array}{l}
\text{YES : } \{\langle h_1, (1) \rangle \triangleright \langle h_2, (1) \rangle \}\}
\end{array}
\]

\[
\begin{array}{l}
\text{YES : } \{\langle h_1, (1, 0) \rangle \triangleright \langle h_2, (1, 1) \rangle \}\}
\end{array}
\]

\[
\begin{array}{l}
\text{NO : } \{\langle h_1, (1, 1) \rangle \triangleright \langle h_2, (1, 0) \rangle \}\}
\end{array}
\]

Frédéric Dabrowski (INRIA, LANDE)
Disjoint reachability

\[ W, \Pi, \Gamma \vdash x = \text{new } h : \Gamma[x \mapsto \langle \Pi', h \rangle], \emptyset \]

\[ W, \Pi, \Gamma \vdash x = y : \Gamma[x \mapsto \Gamma(y)] \quad W, \Pi, \Gamma \vdash x.f = y, \Gamma, K \]

\[ W, \Pi, \Gamma \vdash x = y.f : \Gamma[x \mapsto \langle \lambda w. \top, \top \rangle], \emptyset \]

\[ W, \Pi, \Gamma \vdash s_1 : \Gamma', K_1 \quad W, \Pi, \Gamma' \vdash s_1 : \Gamma'', K_2 \]

\[ W, \Pi, \Gamma \vdash s_1; s_2 : \Gamma'', K_1 \cup K_2 \]

\[ W, \Pi, \Gamma \vdash s_1 : \Gamma_1, K_1 \quad W, \Pi, \Gamma \vdash s_1 : \Gamma_2, K_2 \]

\[ W, \Pi, \Gamma \vdash \text{if } (\ast) \text{ then } s_1 \text{ else } s_2 : \Gamma_1 \sqcup \Gamma_2, K_1 \cup K_2 \]

\[ W \cup \{w\}, \Pi, \Gamma^w \vdash s : \Gamma, K \quad \Pi(w) \neq 0 \]

\[ W, \Pi, \Gamma \vdash \text{while}^w (\ast) \text{ do } s : \Gamma, K \]
Conflicting pairs elimination

Static analysis
- Call graph construction and context-sensitive points-to analysis
- Type and effect system
Conflicting pairs elimination

- Reachable pairs computation
- Aliasing pairs computation
- Escaping pairs computation
- Possible in parallel pairs computation
- Unlocked pairs computation
- May happen in parallel pairs computation
- Call-graph construction
- Thread-escape analysis
- Alias analysis
- Conditional must not alias analysis
- Static race detection

Frédéric Dabrowski (INRIA, LANDE)
Soundness

- reflection
- dynamic loading
- native methods
- libraries ? ? ?
Conclusion and ongoing work

- Data races detection is important
- Objects offer a good framework
- Type systems can handle strict lock-based discipline but lack more elaborated synchronisation patterns
- Points-to analysis can give very precise results but is much more complex
Conclusion and ongoing work

Ongoing work

Certification of a static analysis for data race detection in Coq

1. Context-sensitive points-to analysis
   - Certification of a result checker in Coq

2. Static analysis for data race detection
   - Formalisation of aiken’s type and effect system for Java Bytecode
   - Formalisation of successive stages
   - Certification in Coq