Static Analysis for Java Concurrency with 
*ThreadSafe*

Robert Atkey

http://bentnib.org

*Advances in Programming Languages*

11th November 2014
Static Analysis Tools

what? why?

Contemplate readSafe

a tool for finding Java concurrency bugs

hopes, dreams, tradeoffs
Static Analysis Tools

*what? why?*

Contemplate ThreadSafe

*a tool for finding Java concurrency bugs*
Static Analysis Tools
  what? why?

Contemplate ThreadSafe
  a tool for finding Java concurrency bugs

How ThreadSafe works
  hopes, dreams, tradeoffs
Static Analysis Tools
In general, a Static Analysis Tool:

1. Receives some software code, often source code, sometimes compiled bytecode or machine code.
2. Analyses the code, under some assumptions, using algorithms of varying sophistication.
3. Outputs "something interesting" about the input:
   - usually a list of possible bugs;
   - possible improvements;
   - interesting information (e.g., $10 \times 10$ is always true).
In general, a Static Analysis Tool:

1. Receives some software code
   
   * often source code,
   * sometimes compiled bytecode or machine code

   outputs "something interesting" about the input
   
   * usually a list of possible bugs;
   * possible improvements;
   * interesting information (e.g., $0 \leq 10$ is always true)
In general, a Static Analysis Tool:

1. Receives some software code
   - *often source code,*
   - *sometimes compiled bytecode or machine code*

2. Analyses the code, under some assumptions
   - *using algorithms of varying sophistication*
In general, a Static Analysis Tool:

1. Receives some software code
   * often source code,
   * sometimes compiled bytecode or machine code

2. Analyses the code, under some assumptions
   * using algorithms of varying sophistication

3. Outputs “something interesting” about the input
   * usually a list of possible bugs; or
   * possible improvements;
   * interesting information (e.g., $0 \leq x \leq 10$ is always true)
Static Analysis tools for defect finding...

- Static Analysis tools are a kind of developer tool like: IDEs, doc generators, code search, refactoring...
- They find problems with code without executing it. "Static" analysis.
- Static/dynamic hybrids exist,
  · sometimes without compilation.
- They are intended to improve software quality
  · however this is measured:
    · fewer defects in production?
    · easier to maintain?
    · performs to spec?
Static Analysis tools for defect finding...

- ... are a kind of developer tool
  
  *like: IDEs, doc generators, code search, refactoring...*
Static Analysis tools for defect finding...

- ... are a kind of developer tool
  *like: IDEs, doc generators, code search, refactoring...*

- ... find problems with code without executing it
  *“static” analysis*
  - static/dynamic hybrids exist
  - sometimes without compilation

*however this is measured*
- fewer defects in production?
- easier to maintain?
- performs to spec?
Static Analysis tools for defect finding...

- are a kind of developer tool
  
  like: IDEs, doc generators, code search, refactoring...

- find problems with code without executing it
  
  “static” analysis
  
  · static/dynamic hybrids exist
  
  · sometimes without compilation

- are intended to improve “software quality”

  however this is measured
  
  · fewer defects in production?
  
  · easier to maintain?
  
  · performs to spec?
Static Analysis Tools are good for:

- bugs that are difficult to find with testing
  - security bugs
  - concurrency bugs
  - distributed systems bugs
  - memory corruption bugs

- properties that are boring to check
  - coding style
  - documentation coverage
  - complexity metrics

- ensuring the absence of very serious bugs
  - exception freedom for safety critical systems
  - responsiveness (e.g., < 1 ms)
Static Analysis Tools are good for...

- bugs that are difficult to find with testing
  - security bugs
  - concurrency bugs
  - distributed systems bugs
  - memory corruption bugs
Static Analysis Tools are good for...

- bugs that are difficult to find with testing
  - security bugs
  - concurrency bugs
  - distributed systems bugs
  - memory corruption bugs

- properties that are boring to check
  - coding style
  - documentation coverage
  - complexity metrics
Static Analysis Tools are good for...

▷ ... bugs that are difficult to find with testing
  - security bugs
  - concurrency bugs
  - distributed systems bugs
  - memory corruption bugs

▷ ... properties that are boring to check
  - coding style
  - documentation coverage
  - complexity metrics

▷ ... ensuring the absence of very serious bugs
  - exception freedom for safety critical systems
  - responsiveness (e.g., $< 1\text{ms}$)
Examples of Static Analysis tools...

▶ Compilers

Type systems; Checked exceptions; Definite assignment

Compiler plugins (e.g., javac, clang)
Examples of Static Analysis tools...

▶ Compilers

Type systems; Checked exceptions; Definite assignment
Compiler plugins (e.g., javac, clang)

▶ IDEs

Eclipse and IntelliJ IDEA have built-in Java analyses
Analysis supports code completion, code search
Examples of Static Analysis tools...

▶ Compilers
Type systems; Checked exceptions; Definite assignment
Compiler plugins (e.g., javac, clang)

▶ IDEs
Eclipse and IntelliJ IDEA have built-in Java analyses
Analysis supports code completion, code search

▶ Commercial/Open Source tools
ThreadSafe, Coverity, Klocwork, Fortify, Microsoft SDV, ...
FindBugs, PMD, Checkstyle, ...
Examples of Static Analysis tools...

- **Compilers**
  - Type systems; Checked exceptions; Definite assignment
  - Compiler plugins (e.g., javac, clang)

- **IDEs**
  - Eclipse and IntelliJ IDEA have built-in Java analyses
  - Analysis supports code completion, code search

- **Commercial/Open Source tools**
  - ThreadSafe, Coverity, Klocwork, Fortify, Microsoft SDV, ...
  - FindBugs, PMD, Checkstyle, ...

- **Academic prototypes**
  - Dafny, Chord, Smallfoot, ESC/Java2, SLAM, ...
Static Analysis Tools differ in...

- The language(s) they analyse
  
  C, C++, Java, C#, Javascript, SQL, config languages, ...

- The properties they analyse for
  
  Coding style, concurrency bugs, security bugs, ...

- Scope
  
  Whole program vs. modules/classes/methods/lines

- Soundness
  
  If the tool says "no bugs", then there are no bugs

- Completeness
  
  If there are no bugs, then the tool says "no bugs"
Static Analysis Tools differ in...

- The language(s) they analyse
  
  *C, C++, Java, C#, Javascript, SQL, config languages, ...*

- The properties they analyse for

  *Coding style, concurrency bugs, security bugs, ...*
Static Analysis Tools differ in...

- The language(s) they analyse
  
  C, C++, Java, C#, Javascript, SQL, config languages, ...

- The properties they analyse for
  
  Coding style, concurrency bugs, security bugs, ...

- Scope
  
  Whole program vs. modules/classes/methods/lines
Static Analysis Tools differ in...

- **The language(s) they analyse**
  - C, C++, Java, C#, Javascript, SQL, config languages, ...

- **The properties they analyse for**
  - Coding style, concurrency bugs, security bugs, ...

- **Scope**
  - Whole program vs. modules/classes/methods/lines

- **Soundness**
  - *If the tool says “no bugs”, then there are no bugs*
Static Analysis Tools differ in...

- The language(s) they analyse
  
  C, C++, Java, C#, Javascript, SQL, config languages, ...

- The properties they analyse for
  
  Coding style, concurrency bugs, security bugs, ...

- Scope
  
  Whole program vs. modules/classes/methods/lines

- Soundness
  
  If the tool says “no bugs”, then there are no bugs

- Completeness
  
  If there are no bugs, then the tool says “no bugs”
A fully sound static analysis:

10 PRINT "THERE MIGHT BE BUGS"
A fully sound static analysis:

10 PRINT "THERE MIGHT BE BUGS"

A fully complete static analysis:

10 PRINT "LOOKS FINE TO ME"
A *fully sound* static analysis:

```
10 PRINT "THERE MIGHT BE BUGS"
```

A *fully complete* static analysis:

```
10 PRINT "LOOKS FINE TO ME"
```

But... for “non trivial” properties, a fully sound *and* complete analysis is impossible, by reduction to the Halting Problem.
Soundness vs. Completeness...

Traditional academic answer:

- Retain soundness
  *allow no false negatives*

- Drop full completeness, but aim for “complete enough”
  *admit some false positives*

- “Better safe than sorry”
Soundness vs. Completeness...

Traditional academic answer:

▶ Retain soundness
  
  *allow no false negatives*

▶ Drop full completeness, but aim for “complete enough”
  
  *admit some false positives*

▶ “Better safe than sorry”

However:

▶ Retrofitting a sound analysis to language is very hard

▶ Almost always need to do one of:
  
  ▶ Rearrange the program to suit the analysis
  ▶ Annotate the program to help the analysis
  ▶ Deal with large numbers of false positives
  ▶ Only handle very small programs
Soundiness

“Yet, in practice, soundness is commonly eschewed: we are not aware of a single realistic whole-program analysis tool (e.g., tools widely used for bug detection, refactoring assistance, programming automation, etc.) that does not purposely make unsound choices. Similarly, virtually all published whole-program analyses are unsound and omit conservative handling of common language features when applied to real programming languages.”

http://soundiness.org

- Full soundness is generally unachievable
  Usually for reasons of scalability
- Should make clear exactly what is not handled
  e.g., pointer arithmetic, reflection, runtime code gen
- “Rigidly defined areas of doubt and uncertainty”
In practice, even “soundiness” is not achieved.

In order to reliably analyse “big” code bases:

- Make heuristic assumptions
  - assume non-aliasing of references
  - assume some exceptions never happen
  - assume that data is never mutated, sometimes
  - many vaguely stated assumptions

- Many of which are obviously false
  - very difficult to predict the interactions

- Evaluating such tools can be very hard
  - don't know what the “real” bug count is
  - maintain a test suite of desired true/false +ves/-ves
  - continuously test on real-world code
Contemplate ThreadSafe
ThreadSafe is:

- A tool to find concurrency bugs in Java programs
  - Race conditions
  - Atomicity violations
  - Deadlocks
  - Performance issues
  - Bad style

- Not sound, not complete, but often useful

- Has Eclipse, SonarQube and CLI/HTML interfaces

- Developed by Contemplate Ltd, an Informatics spin-out

- Free trials at:
  http://www.contemplateltd.com/threadsafe
Demo Pt. I...
Inside ThreadSafe
Analysing Java is surprisingly challenging:

- Many bugs in C or C++ code are disallowed in Java:
  
  *buffer overflows, double frees, undefined behaviour, ...*
Analysing Java is surprisingly challenging:

▶ Many bugs in C or C++ code are disallowed in Java: buffer overflows, double frees, undefined behaviour, ...

▶ Many classes of error remain: Unsafe by default concurrency, NPEs, memory leaks
Analysing Java is surprisingly challenging:

▶ Many bugs in C or C++ code are disallowed in Java:  
  *buffer overflows, double frees, undefined behaviour, ...*

▶ Many classes of error remain:  
  *Unsafe by default concurrency, NPEs, memory leaks*

▶ Java's interprocedural control flow is highly dynamic:  
  *x.method(...);*  
  What this does depends on what x points to...
To find out what \( x \) points to, look at all the places \( x \) is assigned:

\[
x = y.f;
\]
To find out what $x$ points to, look at all the places $x$ is assigned:

$$x = y.f;$$

Now need to know what object $o$ that $y$ points to, and what $o.f$ points to,
To find out what x points to, look at all the places x is assigned:

\[ x = y.f; \]

Now need to know what object o that y points to, and what o.f points to, so look for:

\[ z.f = w; \]

where z points to o,
To find out what $x$ points to, look at all the places $x$ is assigned:

$$x = y.f;$$

Now need to know what object $o$ that $y$ points to, and what $o.f$ points to, so look for:

$$z.f = w;$$

where $z$ points to $o$, and $w$ might be a method parameter,
To find out what \( x \) points to, look at all the places \( x \) is assigned:

\[
x = y.f;
\]

Now need to know what object \( o \) that \( y \) points to, and what \( o.f \) points to, so look for:

\[
z.f = w;
\]

where \( z \) points to \( o \), and \( w \) might be a method parameter, so look for calls:

\[
u.m(v);
\]
To find out what \( x \) points to, look at all the places \( x \) is assigned:

\[
x = y.f;
\]

Now need to know what object \( o \) that \( y \) points to, and what \( o.f \) points to, so look for:

\[
z.f = w;
\]

where \( z \) points to \( o \), and \( w \) might be a method parameter, so look for calls:

\[
u.m(v);
\]

where we have to find out what \( u \) and \( v \) point to...
Andersen's algorithm is a (class of) algorithms that compute the “closure” of the effect of all the variable/field assignments, method calls and object creation statements in a whole program.
Andersen's algorithm is a (class of) algorithms that compute the “closure” of the effect of all the variable/field assignments, method calls and object creation statements in a whole program.

Able to answer queries like “what classes of object can x point to”, or “can x and y ever point to the same object?”
Andersen's algorithm is a (class of) algorithms that compute the “closure” of the effect of all the variable/field assignments, method calls and object creation statements in a whole program.

Able to answer queries like “what classes of object can x point to”, or “can x and y ever point to the same object?”

Refined versions of Andersen's algorithm exist that are context sensitive: “can the field f of object o1 that was created by object o2 ever point to o3?”; or “can x point to o in thread t?”.
Andersen's algorithm is a (class of) algorithms that compute the “closure” of the effect of all the variable/field assignments, method calls and object creation statements in a whole program.

Able to answer queries like “what classes of object can x point to”, or “can x and y ever point to the same object?”

Refined versions of Andersen's algorithm exist that are context sensitive: “can the field f of object o1 that was created by object o2 ever point to o3?”, or “can x point to o in thread t?”. 

But:

- Very difficult to reliably scale to large programs
- Needs the whole program, including bits generated at runtime
- Often imprecise in baffling ways
  
  *usually as a result of code reuse*
So *ThreadSafe* doesn't use whole-program points-to analysis.
So *ThreadSafe* doesn't use whole-program points-to analysis.

Instead:

1. Perform a
   - flow sensitive
   - path sensitive
   - context sensitive

   points-to and lock analysis on *each class individually*

   *Generates an “exploded control flow graph”*
So *ThreadSafe* doesn't use whole-program points-to analysis.

Instead:

1. Perform a
   - flow sensitive
   - path sensitive
   - context sensitive
   points-to and lock analysis on each class individually
   *Generates an “exploded control flow graph”*

2. Method calls that leave the class are over-approximated
   *(mostly... see above about unsound heuristics)*
So *ThreadSafe* doesn't use whole-program points-to analysis.

Instead:

1. Perform a
   - flow sensitive
   - path sensitive
   - context sensitive
   points-to and lock analysis on *each class individually*
   
   *Generates an “exploded control flow graph”*

2. Method calls that leave the class are over-approximated
   
   *(mostly... see above about unsound heuristics)*

3. Aggregate the results into a gigantic graph
So *ThreadSafe* doesn't use whole-program points-to analysis.

Instead:

1. Perform a
   - flow sensitive
   - path sensitive
   - context sensitive
   points-to and lock analysis on *each class individually*  
     *Generates an “exploded control flow graph”*

2. Method calls that leave the class are over-approximated  
   *(mostly... see above about unsound heuristics)*

3. Aggregate the results into a gigantic graph

4. Search graph looking for potential concurrency bugs
An example:

If:

1. There is a graph node where field $f$ is assigned a concurrent collection; and
2. All graph nodes where the object read from the field $f$ has a method invoked on it are in a state with a common (abstract) lock $l$ is held,

Then:

- Report “ThreadSafe collection consistently guarded”

In practice, the combination of the two steps of the analysis approximate what Andersen's analysis does, up to a fixed depth.
Demo Pt. II...
Extra-linguistic challenges
Extra-linguistic challenges

- Developers differ widely in their attitude to static analysis
  - If it is never called, is it a bug?
  - If it doesn't go wrong on Intel chips, is it a bug?
  - If a user cannot trigger it, is it a bug?
  - If a user cannot easily trigger it, is it a bug?
  - If I don't understand the report, is it a bug?
  - Tolerance to false positives
  - Tolerance to adding annotations
Extra-linguistic challenges

- Developers differ widely in their attitude to static analysis
  - If it is never called, is it a bug?
  - If it doesn't go wrong on Intel chips, is it a bug?
  - If a user cannot trigger it, is it a bug?
  - If a user cannot easily trigger it, is it a bug?
  - If I don't understand the report, is it a bug?
  - Tolerance to false positives
  - Tolerance to adding annotations

- The Java development world is massively diverse
  - Many IDEs
  - Many frameworks: JEE, Spring, Guava,
  - Many coding styles
  - (growing) many languages that compile to the JVM
Conclusions
Static Analysis:

- Can be very useful

- Often requires very careful evaluation to determine usefulness for your circumstances
Static Analysis:

- Can be very useful
- Often requires very careful evaluation to determine usefulness for your circumstances

Contemplate ThreadSafe

- A useful tool for finding dangerous concurrency bugs in Java programs
Static Analysis:

- Can be very useful
- Often requires very careful evaluation to determine usefulness for your circumstances

Contemplate ThreadSafe

- A useful tool for finding dangerous concurrency bugs in Java programs

Pointers

- http://soundiness.org
- http://www.contemplatet ltd.com/threadsafe