Preventing Errors Before They Happen
The Checker Framework

`http://CheckerFramework.org/`

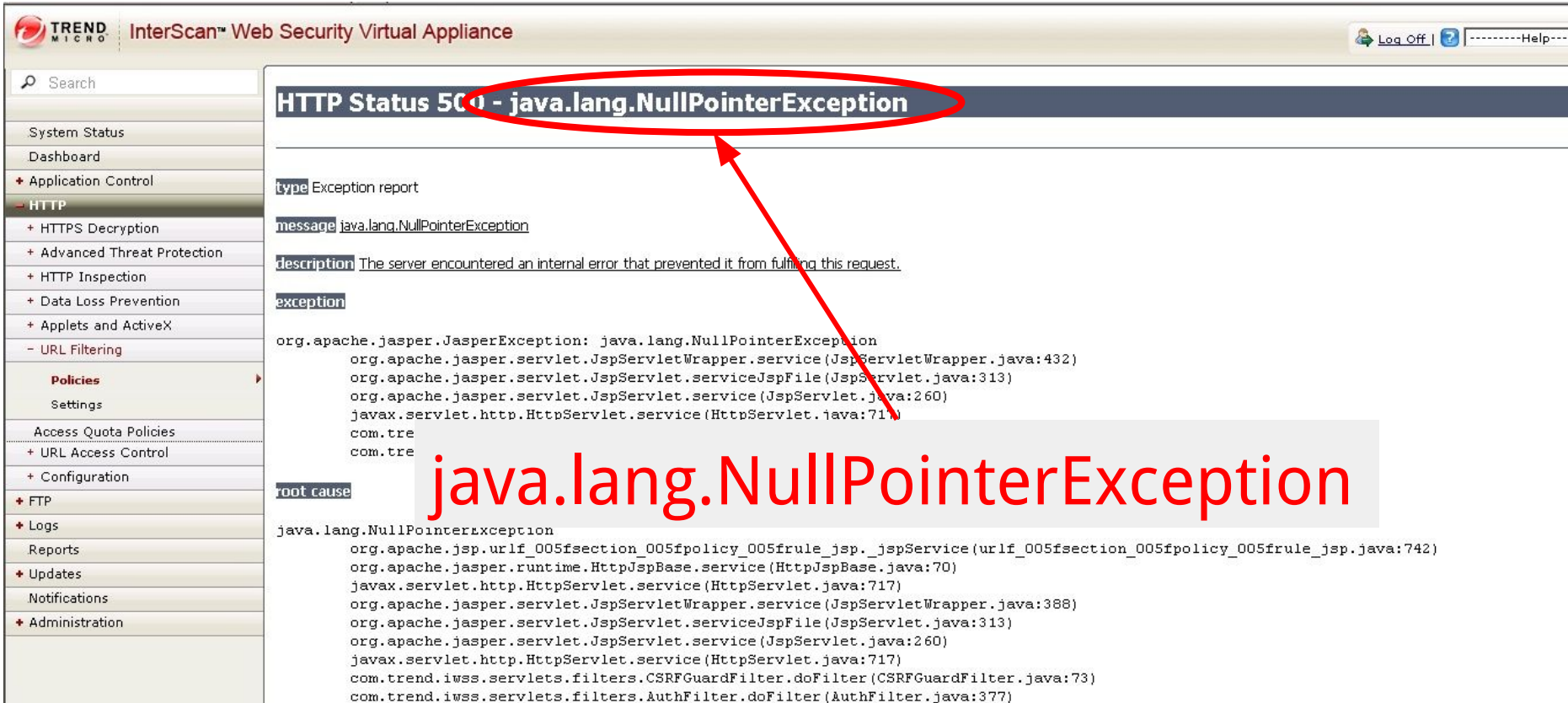
Twitter: @CheckerFrmwrk

Live demo: `http://CheckerFramework.org/live/`

Werner Dietl, University of Waterloo
Michael Ernst, University of Washington

UNIVERSITY OF WATERLOO

Motivation


TREND MICRO
InterScan™ Web Security Virtual Appliance
Log Off |
Help
Search
System Status
Dashboard
Application Control
HTTP
HTTPS Decryption
Advanced Threat Protection
HTTP Inspection
Data Loss Prevention
Applets and ActiveX
URL Filtering
Policies
Settings
Access Quota Policies
URL Access Control
Configuration
FTP
Logs
Reports
Updates
Notifications
Administration
HTTP Status 500 - java.lang.NullPointerException
type Exception report
message java.lang.NullPointerException
description The server encountered an internal error that prevented it from fulfilling this request.
exception
org.apache.jasper.JasperException: java.lang.NullPointerException
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.tre
com.tre
root cause
java.lang.NullPointerException
org.apache.jsp.urlf_005fsection_005fpolicy_005frule_jsp._jspService(urlf_005fsection_005fpolicy_005frule_jsp.java:742)
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:388)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.trend.iwss.servlets.filters.CSRFGuardFilter.doFilter(CSRFGuardFilter.java:73)
com.trend.iwss.servlets.filters.AuthFilter.doFilter(AuthFilter.java:377)
java.lang.NullPointerException

Cost of software failures

$312 billion per year global cost of software bugs (2013)

$300 billion dealing with the Y2K problem

$440 million loss by Knight Capital Group Inc. in 30 minutes in August 2012

$650 million loss by NASA Mars missions in 1999; unit conversion bug

$500 million Ariane 5 maiden flight in 1996; 64-bit to 16-bit conversion bug

Software bugs can cost lives

1985-2000: **>8 deaths**: Radiation therapy
1991: **28 deaths**: Patriot missile guidance system
1997: **225 deaths**: jet crash caused by radar software
2003: **11 deaths**: blackout

2011: Software caused 25% of all medical device recalls

Outline

- Verification approach: Pluggable type-checking
- Tool: Checker Framework
- How to use it
- Creating a custom type system

Java's type system is too weak

Type checking prevents many errors

```
int i = "hello";
```

Type checking doesn't prevent enough errors

```
System.console().readLine();
```

```
Collections.emptyList().add("one");
```

Java's type system is too weak

Type checking prevents many errors

```
int i = "hello";
```

Type checking doesn't prevent enough errors

NullPointerException

```
System.console().readLine();
```

```
Collections.emptyList().add("one");
```

Java's type system is too weak

Type checking prevents many errors

```
int i = "hello";
```

Type checking doesn't prevent enough errors

```
System
Collections.emptyList().add("one");
```

UnsupportedOperationException

Some errors are silent

```
Date date = new Date();
myMap.put(date, "now");
date.setSeconds(0);   // round to minute
myMap.get(date);
```

Some errors are silent

```
Date date = new Date();
myMap.put(date, "now");
date.setSeconds(0);   // round to minute
myMap.get(date);
```

Corrupted map

Some errors are silent

```
dbStatement.executeQuery(userInput);
```

Some errors are silent

```
dbStatement.executeQuery(userInput);
```

SQL injection attack

Initialization, data formatting, equality tests, …

SQL injection attack

Goal: don't execute user input as a SQL command

```
private String wrapQuery(String s) {
  return "SELECT * FROM User WHERE userId='" + s + "'";
}
```

If a user inputs their name as: `' or 'x'='x`

the SQL query is: `... WHERE userID='' or 'x'='x'`

To prevent errors: sanitize user data before use

Vulnerable code

```
void op(String in) {
   …
   executeQuery(in);
}
…
op(userInput);
```

Vulnerable code

```
void op(String in) {
   …
   executeQuery(in);
}
…
op(userInput);
```

Where is the defect?

Vulnerable code

```
void op(String in) {
   …
   executeQuery(in);
}
…
op(userInput);
```

Where is the defect?

Vulnerable code

```
void op(String in) {
   …
   executeQuery(in);
}
…
op(userInput);
```

Where is the defect?

Can’t decide without specification!

Specification 1: untainted parameter

```
void op(@Untainted String in) {
  …
  executeQuery(in);
}
…
op(userInput);
```

Specification 1: untainted parameter

```
void op(@Untainted String in) {
  …
  executeQuery(in);
}
…
op(userInput);  // error
```

Specification 2: tainted parameter

```
void op(@Tainted String in) {
   …
   executeQuery(in);
}
…
op(userInput);
```

Specification 2: tainted parameter

```
void op(@Tainted String in) {
   …
   executeQuery(in);                  // error
}
…
op(userInput);
```

Demo: Preventing SQL injection

Goal: don't execute user input as a SQL command

```
private String wrapQuery(String s) {
  return "SELECT * FROM User WHERE userId='" + s + "'";
}
```

If a user inputs their name as: `' or 'x'='x`
the SQL query is: `... WHERE userID='' or 'x'='x'`

@Tainted = might be untrusted user input

@Untainted = sanitized, safe to use

Verification approach: Pluggable Type Checking

1. Design a type system to solve a specific problem
2. Write type qualifiers in code (or, use type inference)

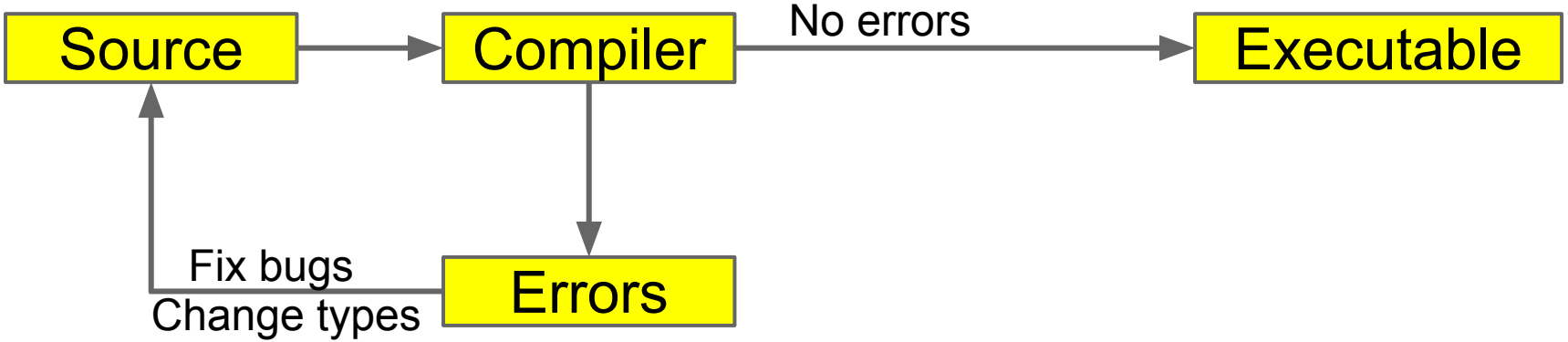
```
@Immutable Date date = new Date();
date.setSeconds(0);  // compile-time error
```

3. Type checker warns about violations (bugs)

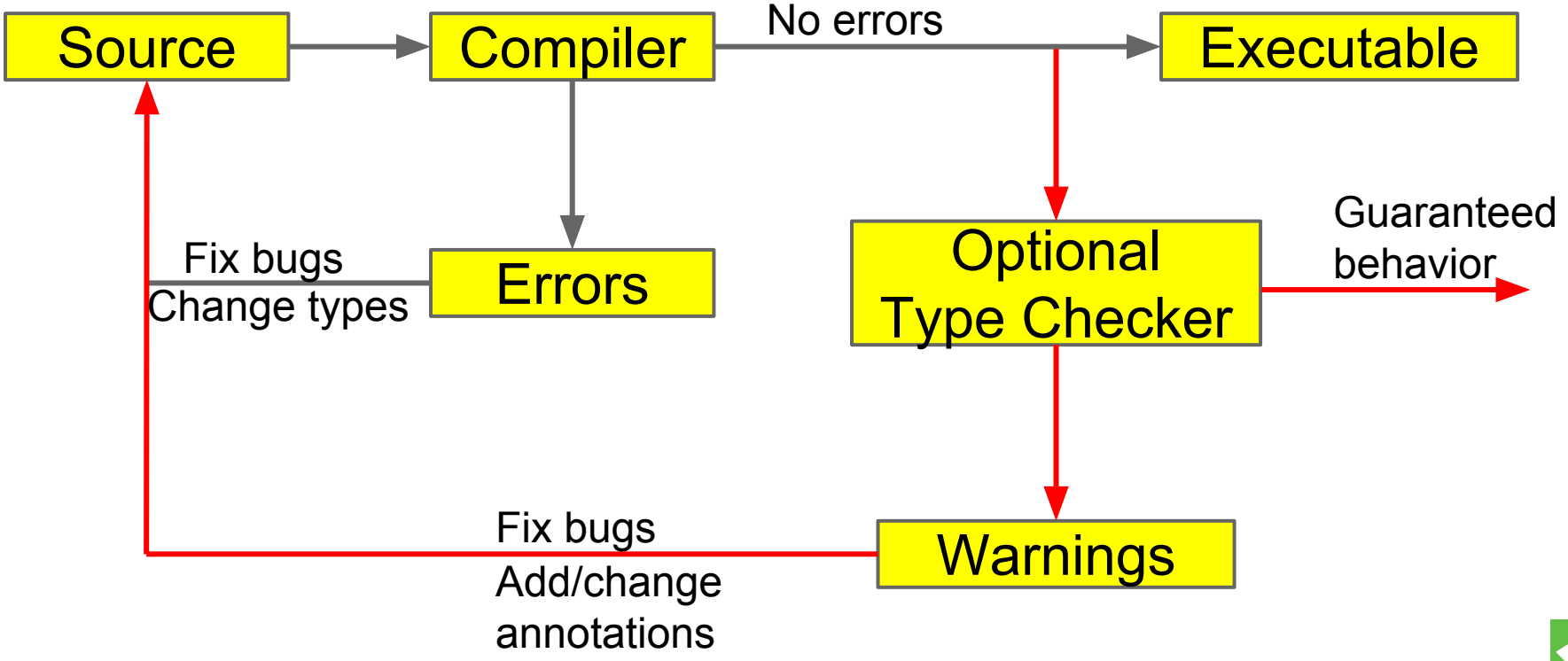
```
% javac -processor NullnessChecker MyFile.java

MyFile.java:149: dereference of possibly-null reference bb2
       allVars = bb2.vars;
                 ^
```

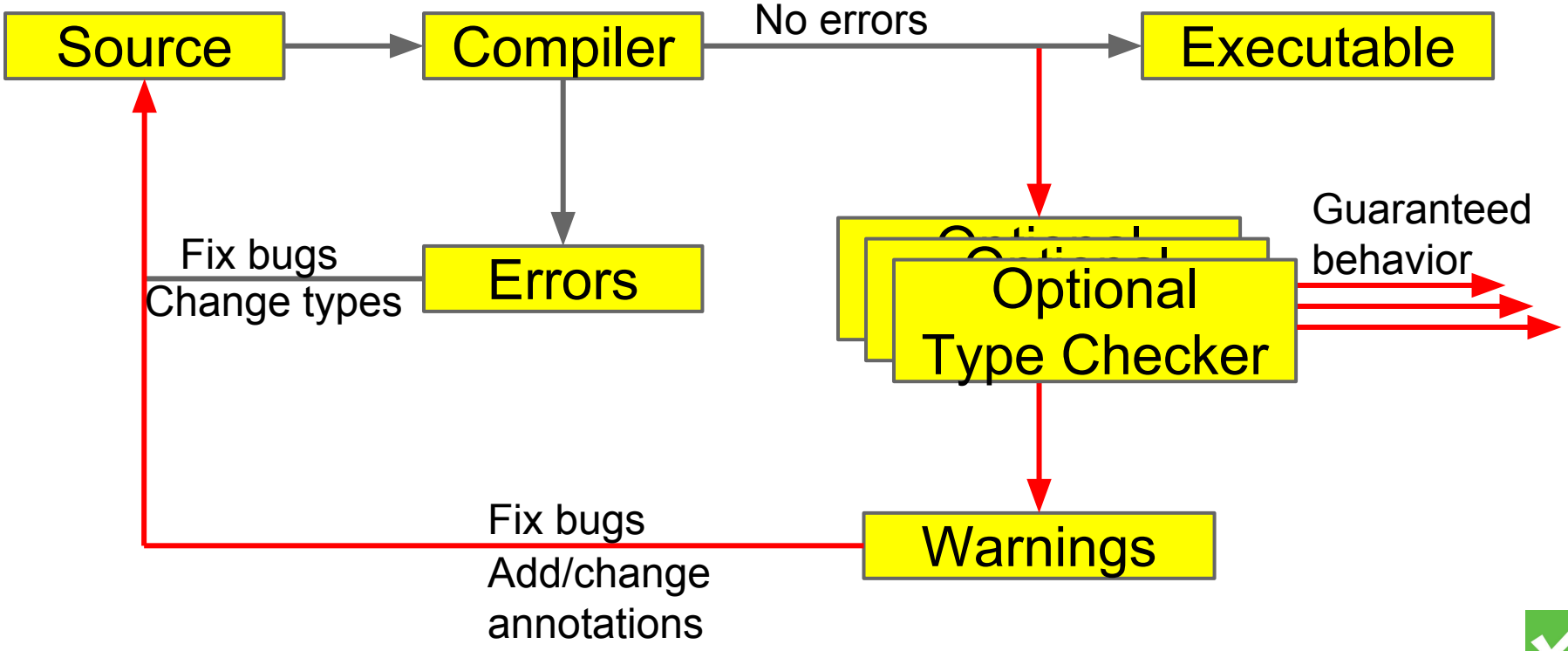
Type Checking



Optional Type Checking



Optional Type Checking



Static type system

Plug-in to the compiler

Doesn’t impact:

- method binding
- memory consumption
- execution

A future tool might affect run-time behavior

Prevent null pointer exceptions

Type system that statically guarantees that:

the program only dereferences known non-null references

Types of data:

`@NonNull`	reference is never null
`@Nullable`	reference may be null

Null pointer exception

```
String op(Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Can’t decide without specification!

Specification 1: non-null parameter

```
String op(@NonNull Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Specification 1: non-null parameter

```
String op(@NonNull Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);      // error
```

Specification 2: nullable parameter

```
String op(@Nullable Data in) {
  return "transform: " + in.getF();
}
…
String s = op(null);
```

Specification 2: nullable parameter

```
String op(@Nullable Data in) {
  return "transform: " + in.getF();
}                                  // error
…
String s = op(null);
```

Nullness demo

- Detect errors
- Guarantee the absence of errors
- Verify the correctness of optimizations

Benefits of type systems

- **Find bugs** in programs
  - Guarantee the **absence of errors**
- **Improve documentation**
  - Improve code structure & maintainability
- Aid compilers, optimizers, and analysis tools
  - E.g., could reduce number of run-time checks

- Possible negatives:
  - Must write the types (or use type inference)
  - False positives are possible (can be suppressed)

The Checker Framework

A framework for pluggable type checkers

“Plugs” into the OpenJDK or OracleJDK compiler

```
javac -processor MyChecker ...
```

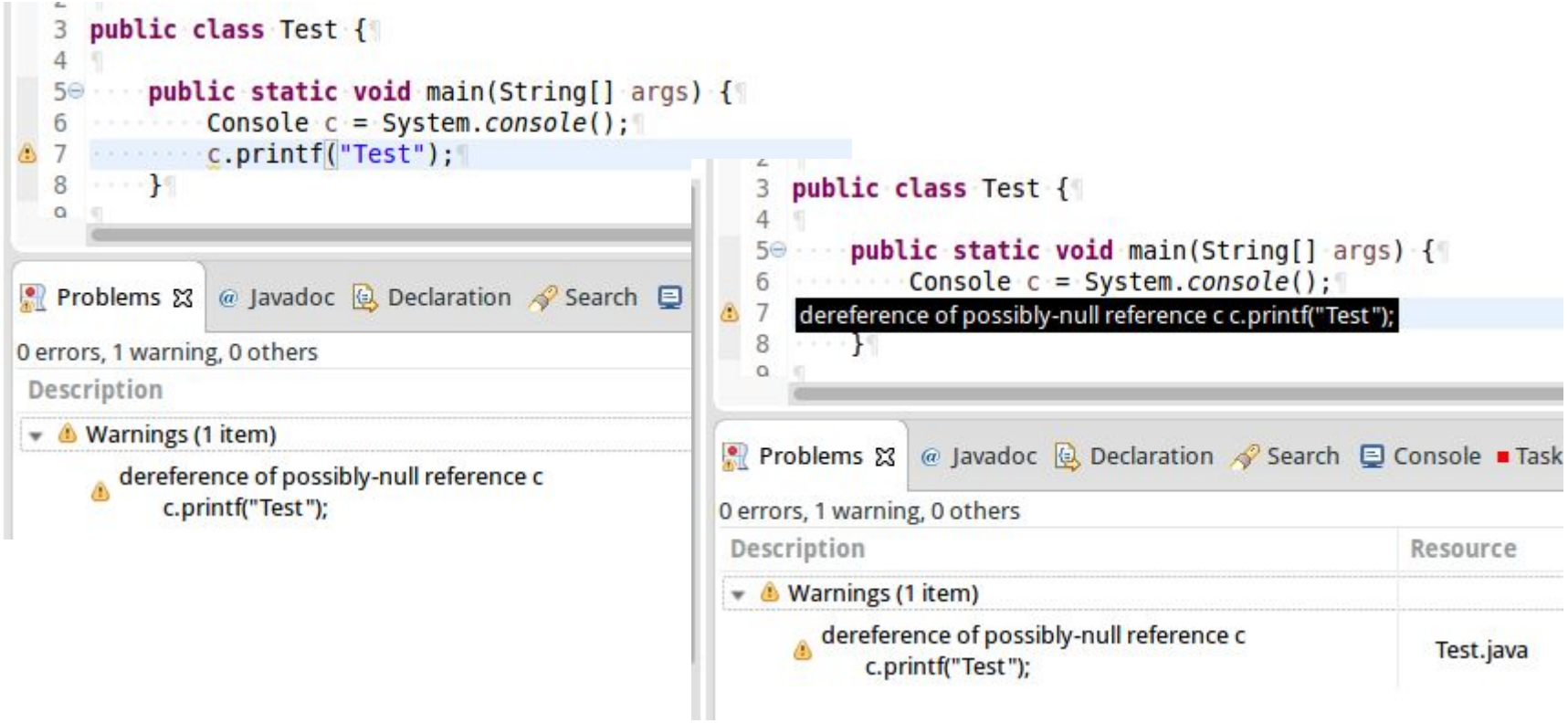
Standard error format allows tool integration

Ant, Maven, Gradle integration

```
<presetdef name="jsr308.javac">
  <javac fork="yes"
    executable="${checkerframework}/checker/bin/${cfJavac}" >
    <!-- JSR-308-related compiler arguments -->
    <compilerarg value="-version"/>
    <compilerarg value="-implicit:class"/>
  </javac>
</presetdef>
```

```
<dependencies>
  ... existing <dependency> items ...
  <!-- annotations from the Checker Framework:
          nullness, interning, locking, ... -->
    <dependency>
      <groupId>org.checkerframework</groupId>
      <artifactId>checker-qual</artifactId>
      <version>1.9.7</version>
    </dependency>
</dependencies>
```

Eclipse, IntelliJ, NetBeans integration

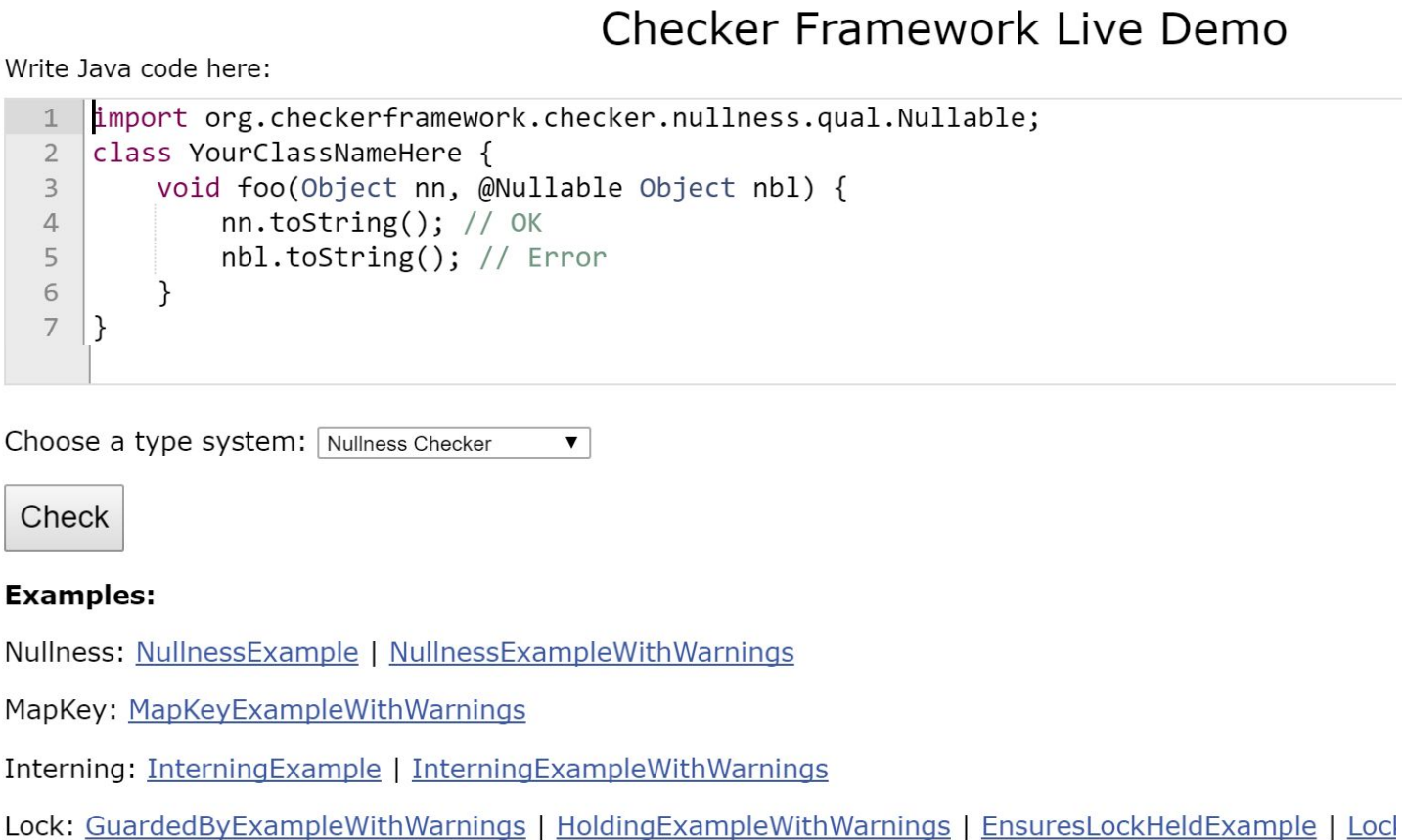


Comparison: other nullness tools

	Null pointer errors		False warnings	Annotations written
	Found	Missed		
Checker Framework	9	0	4	35
FindBugs	0	9	1	0
Jlint	0	9	8	0
PMD	0	9	0	0
Eclipse, in 2017	0	9	8	0
Intellij (@NotNull default), in 2017	0	9	1	0
	3	6	1	925 + 8

Checking the Lookup program for file system searching (4kLOC)

Live demo: http://CheckerFramework.org/live/



Example type systems

Null dereferences (`@NonNull`)

>200 errors in Google Collections, javac, ...

Equality tests (`@Interned`)

>200 problems in Xerces, Lucene, ...

Concurrency / locking (`@GuardedBy`)

>500 errors in BitcoinJ, Derby, Guava, Tomcat, ...

Fake enumerations / typedefs (`@Fenum`)

problems in Swing, JabRef

String type systems

Regular expression syntax (`@Regex`)

56 errors in Apache, etc.; 200 annos required

printf format strings (`@Format`)

104 errors, only 107 annotations required

Method signature format (`@FullyQualified`)

28 errors in OpenJDK, ASM, AFU

Compiler messages (`@CompilerMessageKey`)

8 wrong keys in Checker Framework

Security type systems

Command injection vulnerabilities (`@OsTrusted`)

5 missing validations in Hadoop

Information flow privacy (`@Source`)

SPARTA detected malware in Android apps

It’s easy to write your own type system!

Checkers are usable

- Type-checking is familiar to programmers
- Modular: fast, incremental, partial programs
- Annotations are not too verbose
  - **`@NonNull`**: 1 per 75 lines
  - **`@Interned`**: 124 annotations in 220 KLOC revealed 11 bugs
  - **`@Format`**: 107 annotations in 2.8 MLOC revealed 104 bugs
  - Possible to annotate part of program
  - Fewer annotations in new code
- Few false positives
- First-year CS majors preferred using checkers to not
- **Practical**: in use in Silicon Valley, on Wall Street, etc.

What a checker guarantees

The program satisfies the type property. There are:

- no bugs (of particular varieties)
- no wrong annotations

- Caveat 1: only for code that is checked
  - Native methods (handles reflection!)
  - Code compiled without the pluggable type checker
  - Suppressed warnings
    - Indicates what code a human should analyze

  Checking part of a program is still useful
- Caveat 2: The checker itself might contain an error

Formalizations

$$
\begin{array}{rcll}
\mathtt{P} & \in & \text{Program} & ::= \ \overline{\mathtt{Class}},\ \mathtt{ClassId},\ \mathtt{Expr} \\
\mathtt{Cls} & \in & \text{Class} & ::= \ \mathtt{class\ ClassId}\langle\overline{\mathtt{TVarId}}\rangle \\
 & & & \quad \mathtt{extends\ ClassId}\langle\overline{{}^{\mathtt{s}}\mathtt{Type}}\rangle \\
 & & & \quad \{\ \overline{\mathtt{FieldId}\ {}^{\mathtt{s}}\mathtt{Type};}\ \overline{\mathtt{Met}} \\
{}^{\mathtt{s}}\mathtt{T} & \in & {}^{\mathtt{s}}\text{Type} & ::= \ {}^{\mathtt{s}}\mathtt{NType} \mid \mathtt{TVarId} \\
{}^{\mathtt{s}}\mathtt{N} & \in & {}^{\mathtt{s}}\text{NType} & ::= \ \mathtt{OM\ ClassId}\langle\overline{{}^{\mathtt{s}}\mathtt{Type}}\rangle \\
\mathtt{u} & \in & \text{OM} & ::= \\
\mathtt{mt} & \in & \text{Meth} & ::= \\
 & & \mathtt{MethSig} & ::= \\
\mathtt{w} & \in & \text{Purity} & ::= \\
\mathtt{e} & \in & \text{Expr} & ::= \\
 & & & \quad \mathtt{Expr.MethId}\langle{}^{\mathtt{s}}\mathtt{Type}\rangle\mathtt{(Expr)} \mid \\
 & & & \quad \mathtt{new}\ {}^{\mathtt{s}}\mathtt{Type} \mid \mathtt{(}{}^{\mathtt{s}}\mathtt{Type}\mathtt{)\ Expr} \\
{}^{\mathtt{s}}\Gamma & \in & {}^{\mathtt{s}}\text{Env} & ::= \ \overline{\mathtt{TVarId}\ {}^{\mathtt{s}}\mathtt{NType};}\ \overline{\mathtt{ParId}\ {}^{\mathtt{s}}\mathtt{Type}}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{h} & \in & \text{Heap} & = \ \mathtt{Addr} \to \mathtt{Obj} \\
\iota & \in & \text{Addr} & = \ \text{Set of Addresses} \cup \{\mathtt{null}_a\} \\
\mathtt{o} & \in & \text{Obj} & = \ {}^{\mathtt{r}}\mathtt{Type},\ \mathtt{Fields} \\
{}^{\mathtt{r}}\mathtt{T} & \in & {}^{\mathtt{r}}\text{Type} & = \ \mathtt{OwnerAddr\ ClassId}\langle\overline{{}^{\mathtt{r}}\mathtt{Type}}\rangle \\
\mathtt{Fs} & \in & \text{Fields} & = \ \mathtt{FieldId} \to \mathtt{Addr} \\
\iota & \in & \text{OwnerAddr} & = \ \mathtt{Addr} \cup \{\mathtt{any}_a\} \\
{}^{\mathtt{r}}\Gamma & \in & {}^{\mathtt{r}}\text{Env} & = \ \overline{\mathtt{TVarId}\ {}^{\mathtt{r}}\mathtt{Type};}\ \overline{\mathtt{ParId\ Addr}}
\end{array}
$$

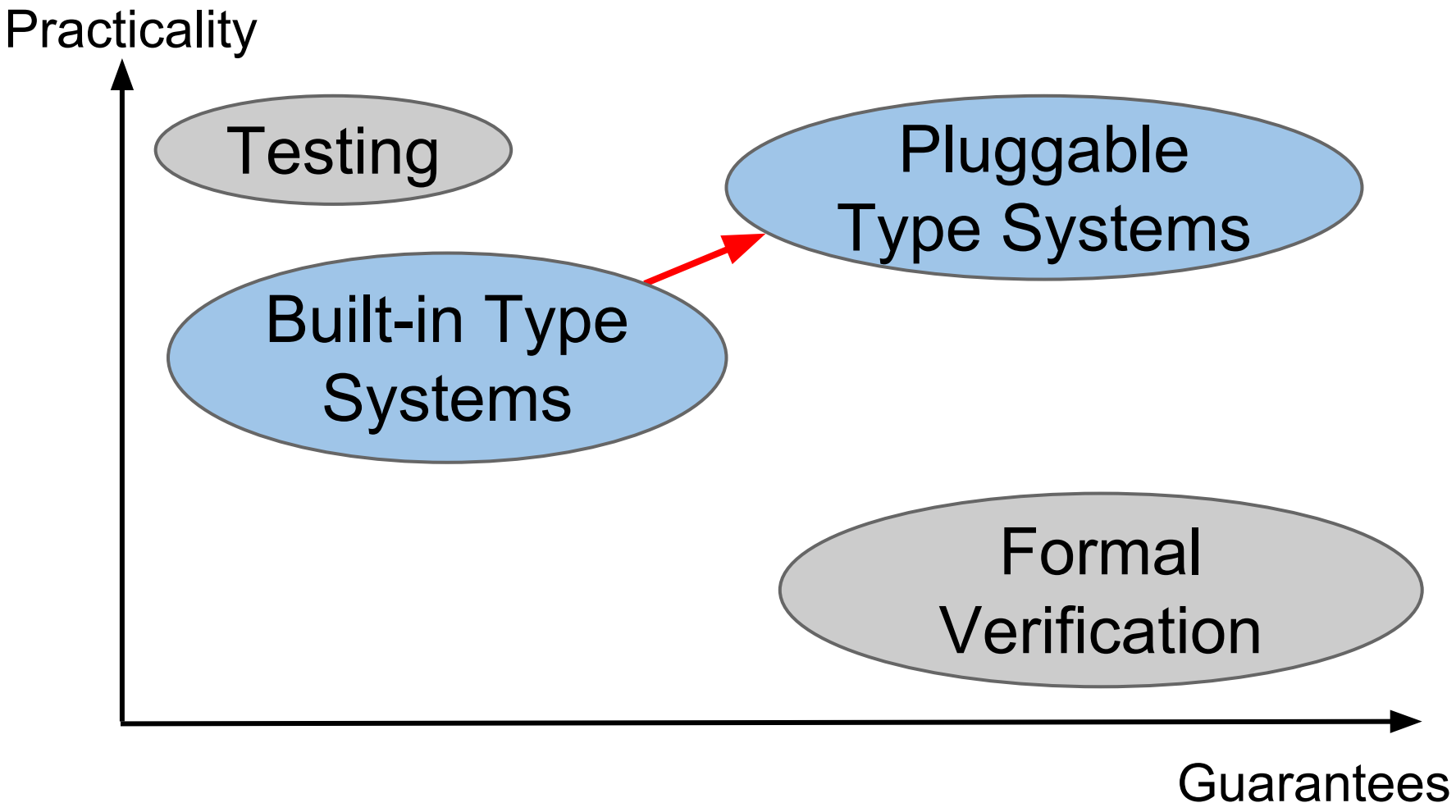
$$
\text{OS-Upd}\ \frac{\begin{array}{c}\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \mathtt{h}_0, {}^{\mathtt{r}}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \\ \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]\end{array}}{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 \rightsquigarrow \mathtt{h}',}
$$

$$
\text{OS-Read}\ \frac{\begin{array}{c}\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})\end{array}}{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathtt{h}', \iota}
$$

$$
\text{GT-Read}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \_\ \langle}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{N}_0 \rhd fType(\mathtt{C}_0, \mathtt{f})}
$$

$$
\text{GT-Upd}\ \frac{\begin{array}{c}\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\langle\_\rangle \\ \mathtt{T}_1 = fType(\mathtt{C}_0, \mathtt{f}) \\ \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1 \\ \mathtt{u}_0 \neq \mathtt{any} \qquad rp(\mathtt{u}_0, \mathtt{T}_1)\end{array}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1}
$$

$$
\text{DYN}\ \frac{\begin{array}{l}\mathtt{h} \vdash {}^{\mathtt{r}}\Gamma : {}^{\mathtt{s}}\Gamma \\ \mathtt{h} \vdash \iota_1 : dyn({}^{\mathtt{s}}\mathtt{N}, \mathtt{h}, {}^{\mathtt{r}}\iota_1) \\ \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{\mathtt{s}}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\langle\_\rangle \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{\mathtt{r}}\Gamma(\mathtt{this}) \\ free({}^{\mathtt{s}}\mathtt{T}) \subseteq dom(\mathtt{C}_N)\end{array}}{\left.\begin{array}{l}\\ \\ \\ \\ \\ \\ {}^{\mathtt{r}}\mathtt{T} = \iota'\ \_\langle\_\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}{<:}\ \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}}\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}{<:}\ \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}_a}\rangle \Rightarrow \iota \vdash \overline{{}^{\mathtt{r}}\mathtt{T}}\ {}^{\mathtt{r}}{<:}\ \overline{{}^{\mathtt{r}}\mathtt{T}_a} \\ \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathrm{free}({}^{\mathtt{s}}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}\right\} \implies \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{N} \rhd {}^{\mathtt{s}}\mathtt{T}, \mathtt{h}, {}^{\mathtt{r}}\Gamma)}{dyn({}^{\mathtt{s}}\mathtt{T}, \iota, {}^{\mathtt{r}}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{\mathtt{r}}\mathtt{T}'}; \_)) = {}^{\mathtt{s}}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{\mathtt{r}}\mathtt{T}/\mathtt{X}}, \overline{{}^{\mathtt{r}}\mathtt{T}'/\mathtt{X}'}]}
$$


Practicality
Testing
Pluggable
Type Systems
Built-in Type
Systems
Formal
Verification
Guarantees

Regular expression errors

`@Regex` = valid regular expression

OK: “`colou?r`”

NOT: “`1) first point`”

`@Regex(2)` = has 2+ capturing groups

OK: “`((Linked)?Hash)?Map`”

OK: “`(http|ftp)://([^/]+)(/.*)?`”

NOT: “`(brown|beige)`”

Regular Expression Example

```
public static void main(String[] args) {
  String regex = args[0];
  String content = args[1];
  Pattern pat = Pattern.compile(regex);
  Matcher mat = pat.matcher(content);
  if (mat.matches()) {
    System.out.println("Group: " + mat.group(1));
  }
}
```

Regular Expression Example

```
public static void main(String[] args) {
   String regex
   String conten
   Pattern pat = Pattern.compile(regex);
   Matcher mat =
   if (mat.match
      System.out.println("Group: " + mat.group(1));
   }
}
```

PatternSyntaxException

IndexOutOfBoundsExceptionon

Demo: Fixing the Errors

`Pattern.compile` only on valid regex
`Matcher.group(i)` only if > i groups

```
...
if (!RegexUtil.isRegex(regex, 1)) {
  System.out.println("Invalid: " + regex);
  System.exit(1);
}
...
```

Since Java 5: declaration annotations

Only for declaration locations:

```
@Deprecated
class Foo {
   @Getter @Setter private String query;
   @SuppressWarnings("unchecked")
   void foo() { … }
}
```

But we couldn’t express

A non-null reference to my data

An interned string

A non-null List of English strings

A non-empty array of English strings

Since Java 8: Type Annotations

A non-null reference to my data

```
@NonNull Data mydata;
```

An interned String

```
@Interned String query;
```

A non-null List of English Strings

```
@NonNull List<@English String> msgs;
```

A non-empty array of English strings

```
@English String @NonEmpty [] a;
```

Type annotation syntax

Annotations on all occurrences of types:

```
@Untainted String query;
List<@NonNull String> strings;
myGraph = (@Immutable Graph) tmp;
class UnmodifiableList<T>
  implements @Readonly List<T> {}
```

Stored in classfile

Handled by javac, javap, javadoc, …

Annotating external libraries

When type-checking clients, need library spec.

```
class System {
            Console console() { … }
}

... System.console().readLine() ...
```

Annotating external libraries

When type-checking clients, need library spec.

```
class System {
  @Nullable Console console() { … }
}
```

Compile-time warning

```
... System.console().readLine() ...
```

Annotating external libraries

When type-checking clients, need library spec.

Can write manually or automatically infer

Two syntaxes:

- As separate text file (stub file)
- Within its .jar file (from annotated partial source code)

Checker Framework facilities

- Full type systems: inheritance, overriding, ...
- Generics (type polymorphism)
  - Also qualifier polymorphism
- Qualifier defaults
- Pre-/post-conditions
- Warning suppression

Brainstorming new type checkers

What runtime exceptions to prevent?

What properties of data should always hold?

What operations are legal and illegal?

Type-system checkable properties:

- Dependency on values
- Not on program structure, timing, ...

Example: Nullness Checker

What runtime exceptions to prevent?

What properties of data should always hold?

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

@NonNull references always non-null

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

@NonNull references always non-null

What operations are legal and illegal?

Dereferences only on @NonNull references

Example: Regex Checker

What runtime exceptions to prevent?

What properties of data should always hold?

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException,
IndexOutOfBoundsException

What properties of data should always hold?

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException,
IndexOutOfBoundsException

What properties of data should always hold?

Whether a string is a regex and number of groups

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException,
IndexOutOfBoundsException

What properties of data should always hold?

Whether a string is a regex and number of groups

What operations are legal and illegal?

Pattern.compile with non-@Regexp, etc,

New type system

What runtime exceptions to prevent?

1. IndexOutOfBoundException

What properties of data should always hold?

2. Array & Index properties

What operations are legal and illegal?

3. Index into array valid

New type system

What runtime exceptions to prevent?

1 Side Effects & purity of methods/lambdas

What properties of data should always hold?

2 Mutability of data

What operations are legal and illegal?

3 Modifications in pure methods

New type system

What runtime exceptions to prevent?

1 Determinism of code

What properties of data should always hold?

2 Deterministic vs. Ordered vs. Non-Det.

What operations are legal and illegal?

3 Don’t iterate over non-det

New type system

What runtime exceptions to prevent?

1 Types for values in properties files

What properties of data should always hold?

2

What operations are legal and illegal?

3

New type system

What runtime exceptions to prevent?

1 Object initialization

What properties of data should always hold?

2 Fully vs. partially initialized objects

What operations are legal and illegal?

3 Nullness assumptions

Building a checker is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {...}
@Encrypted String msg1 = ...;
send(msg1);    // OK
String msg2 = ....;
send(msg2);    // Warning!
```

Building a checker is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {…}
@Encrypted String msg1 = ...;
send(msg1);   // OK
String msg2 = ....;
send(msg2);   // Warning!
```

The complete checker:

```
@Target(ElementType.TYPE_USE)
@SubtypeOf(Unqualified.class)
public @interface Encrypted {}
```

Encrypted Checker Demo

Let’s build it!

Testing infrastructure

jtreg-based testing as in OpenJDK

Lightweight tests with in-line expected errors:

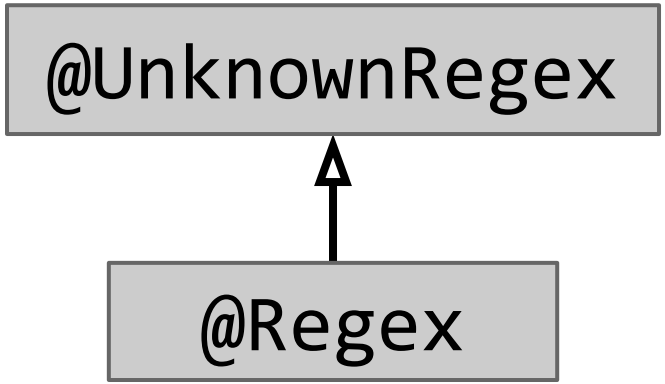
```
String s = "%+s%";
//:: error: (format.string.invalid)
f.format(s, "illegal");
```

Defining a type system

1. Qualifier hierarchy
   - defines subtyping
2. Type introduction rules
   - types for expressions
3. Type rules
   - checker-specific errors
4. Flow-refinement
   - better types than the programmer wrote

Defining a type system

1. Qualifier hierarchy
   - subtyping, assignments




```
@SubtypeOf(UnknownRegex.class)
public @interface Regex {
```

Defining a type system

```
Data d = new Data();
```

2. Type introduction rules
   - types for expressions

```
@ImplicitFor( trees = {
      Tree.Kind.NEW_CLASS,
      Tree.Kind.NEW_ARRAY, ... })
@DefaultQualifierInHierarchy
@DefaultForUnannotatedCode({
      DL.PARAMETERS, DL.LOWER_BOUNDS })
```

Defining a type system

```
synchronized(xxx) {
}
```

3. Type rules
   - checker-specific errors

```
void visitSynchronized(SynchronizedTree node) {
   ExpressionTree expr = node.getExpression();
   AnnotatedTypeMirror type =
         getAnnotatedType(expr);
   if (!type.hasAnnotation(NONNULL))
      checker.report(Result.failure(...), expr);
}
```

Defining a type system

```
if (x != null) {
  x.f = …; // valid
```

4. Flow-refinement
   - better types than the programmer wrote

```
if (ElementUtils.matchesElement(method,
        IS_REGEX_METHOD_NAME,
        String.class, int.class)) {
   …
}
```

Dataflow Framework

Goal: Compute properties about expressions

- More accurate types than the user wrote
- Foundation for other static analyses
  - e.g. by Google error-prone and Uber NullAway

Dataflow Framework user provides

- What are we tracking?
- What do operations do?
- What are intermediate results?

Dataflow Framework does all the work!

Tips

- Start by type-checking part of your code
- Only type-check properties that matter to you
- Use subclasses (not type qualifiers) if possible
- Write the spec first (and think of it as a spec)
- Avoid warning suppressions when possible
- Avoid raw types such as `List`; use `List<String>`

Verification

- **Goal**:
  prove that no bug exists
- **Specifications**:
  user provides
- **False negatives**:
  none
- **False positives**:
  user suppresses warnings
- **Downside**: user burden

Bug-finding

- **Goal**:
  find some bugs at low cost
- **Specifications**:
  infer likely specs
- **False negatives:**
  acceptable
- **False positives**:
  heuristics focus on most important bugs
- **Downside**: missed bugs

Neither is “better”; each is appropriate in certain circumstances.

Checker Framework Community

Open source project:

`https://github.com/typetools/checker-framework`

- Monthly release cycle
- >13,800 commits, 75 authors
- Welcoming & responsive community

Checker Framework plans

More type systems:

- Immutability
- Determinism
- Signed vs. unsigned numbers

Type inference

Combined static & dynamic enforcement

More at CodeOne 2018

Using Type Annotations to Improve Your Code

BOF4992, Tue Oct 23, 19:30 – 20:15

Moscone West - Room 2009

Pluggable type-checking improves code

Checker Framework for creating type checkers

- Featureful, effective, easy to use, scalable

Prevent bugs at compile time

Create custom type-checkers

Improve your code!

`http://CheckerFramework.org/`