Preventing Errors Before They Happen
The Checker Framework

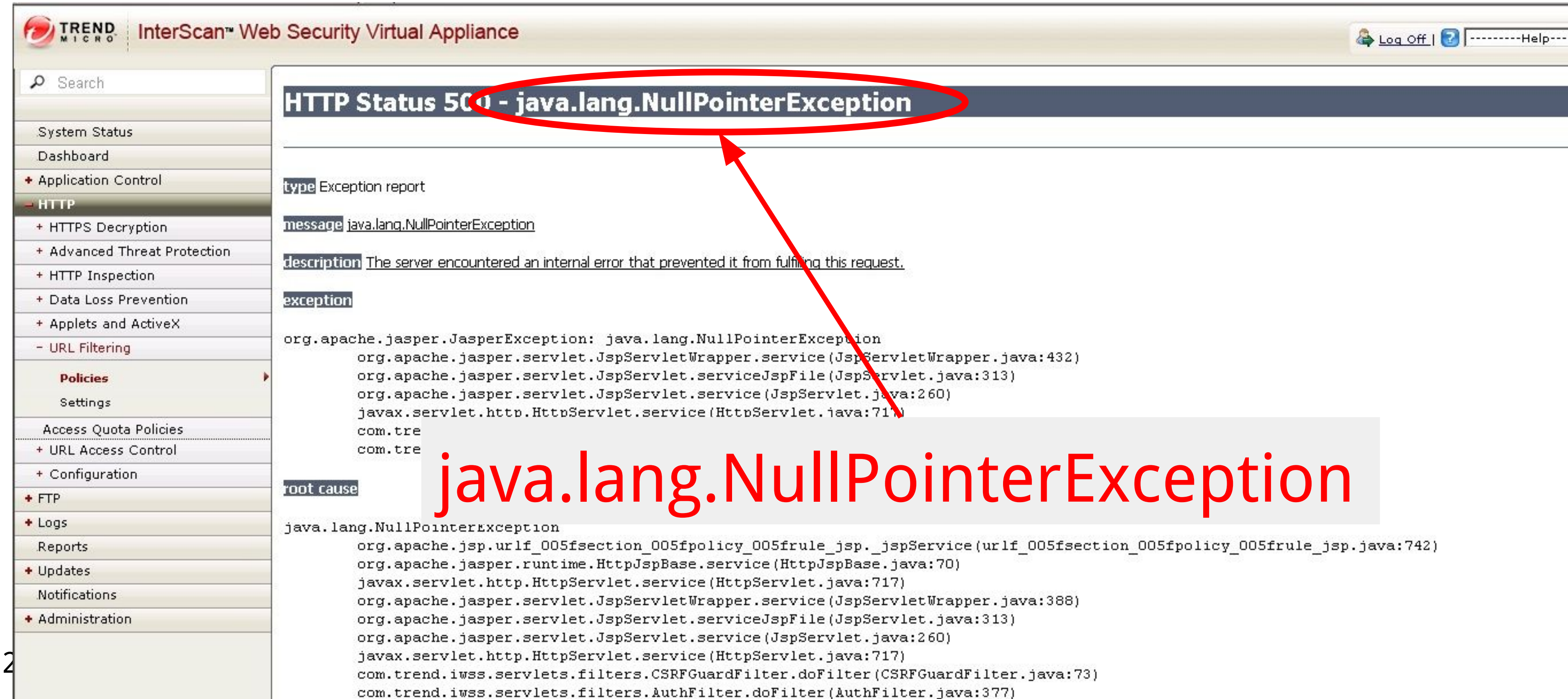
`http://CheckerFramework.org/`

Twitter: @CheckerFrmwrk

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

Werner Dietl, University of Waterloo

Motivation



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
```

UnsupportedOperationException

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

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 uid) {
  return "SELECT * FROM User WHERE userId=’" + uid + "’";
}
```

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

Where is the defect?

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

Vulnerable code

Where is the defect?

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

Vulnerable code

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

Where is the defect?

Can’t decide without specification!

Specification 1: tainted parameter

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

✓

Specification 1: tainted parameter

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

Specification 1: tainted parameter

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

Specification 2: untainted parameter

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

✓

Specification 2: untainted parameter

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

✓

Preventing SQL injection

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

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

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

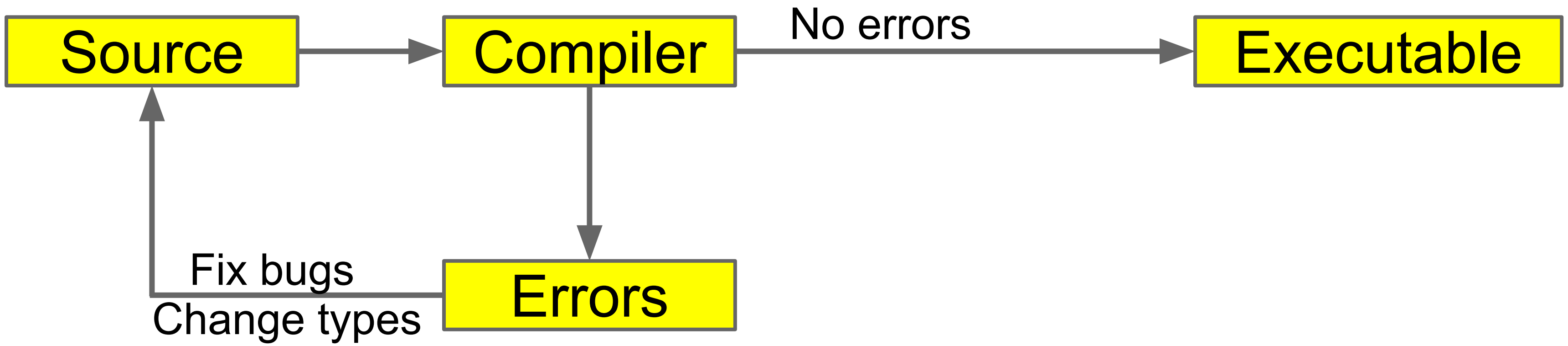
1. Design a type system to solve a specific problem
2. Write type qualifiers in code (or, use type inference)

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

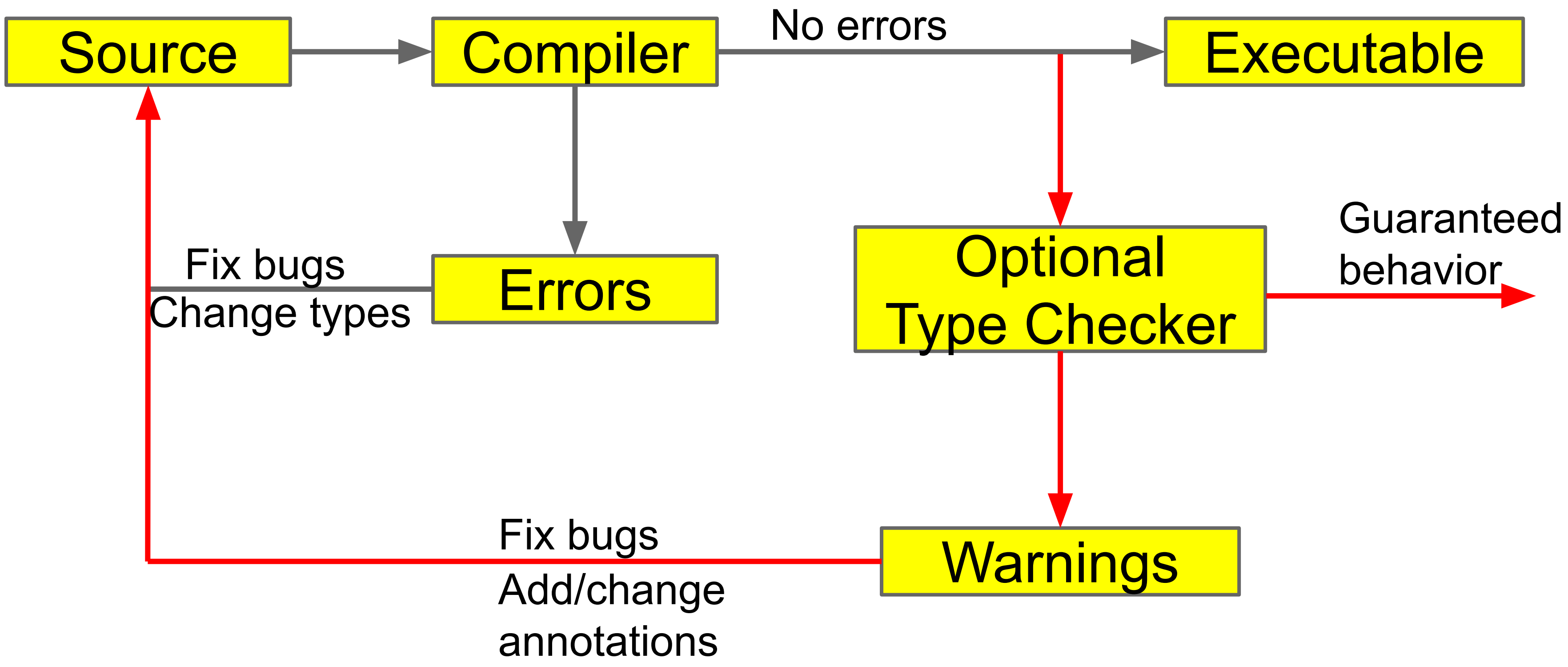
3. Type checker warns about violations (bugs)

```
% javac -processor NullnessChecker MyFile.java

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

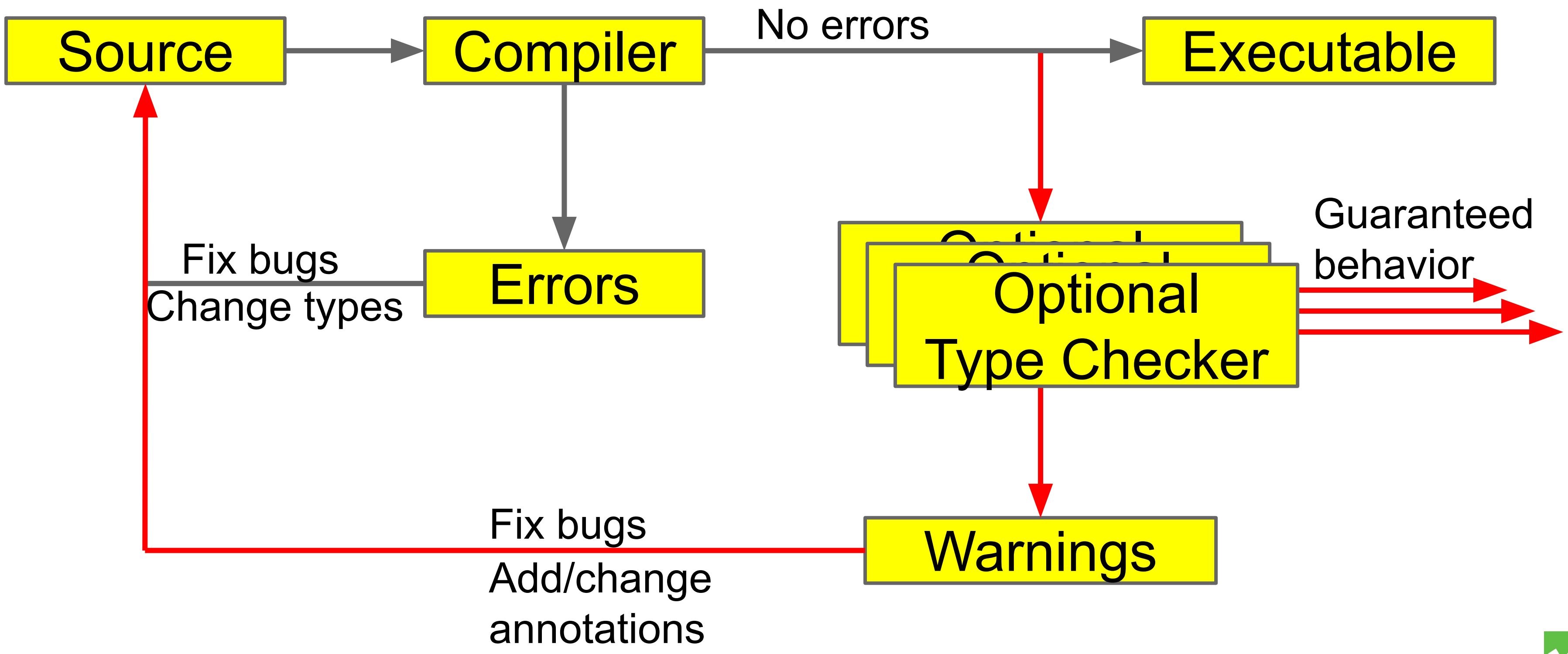
Type Checking



Optional Type Checking



Optional Type Checking



Static type system

Plug-in to the compiler

Doesn’t impact:

- method binding
- memory consumption
- execution

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);
```

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

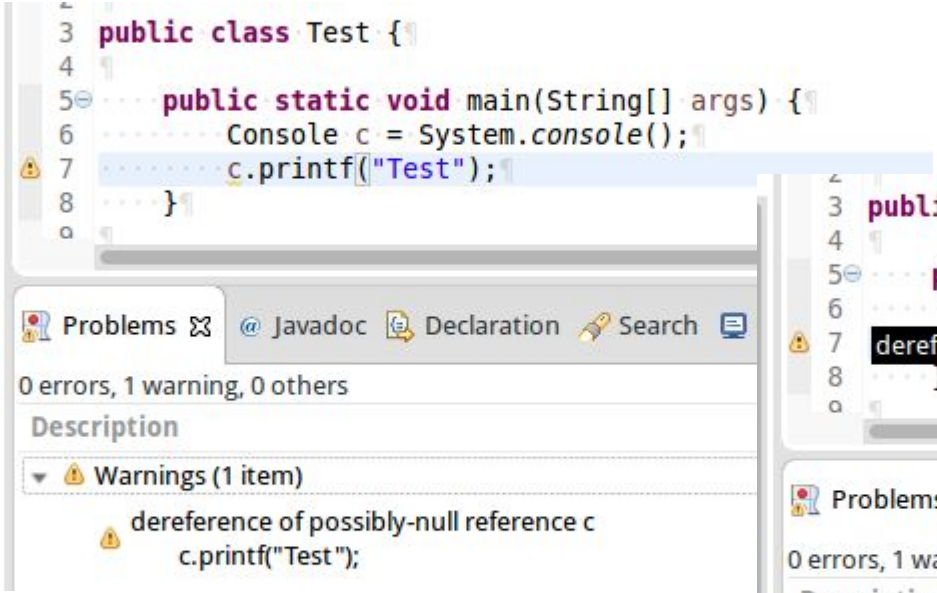
✓

Ant, Gradle, Maven, etc. integration

```
<presetdef name="cfJavac">
  <javac fork="yes"
    executable="${checkerframework}/checker/bin/${cfJavac}" >
    <!-- 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>3.6.1</version>
    </dependency>
</dependencies>
```

Eclipse, IntelliJ, NetBeans integration

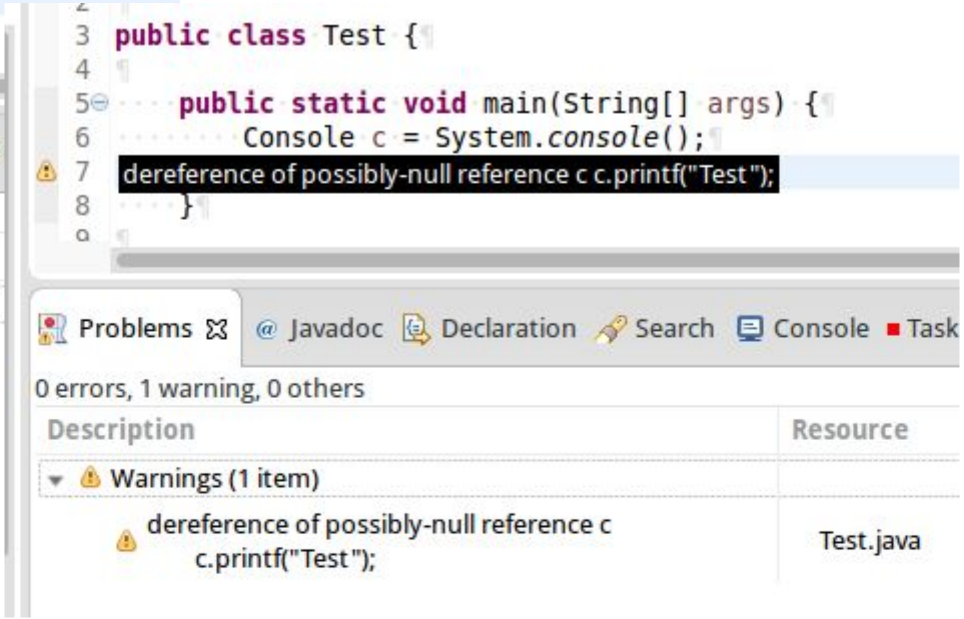


```java
public class Test {

    public static void main(String[] args) {
        Console c = System.console();
        c.printf("Test");
    }
```

Problems | Javadoc | Declaration | Search

0 errors, 1 warning, 0 others

Description
Warnings (1 item)
dereference of possibly-null reference c c.printf("Test");



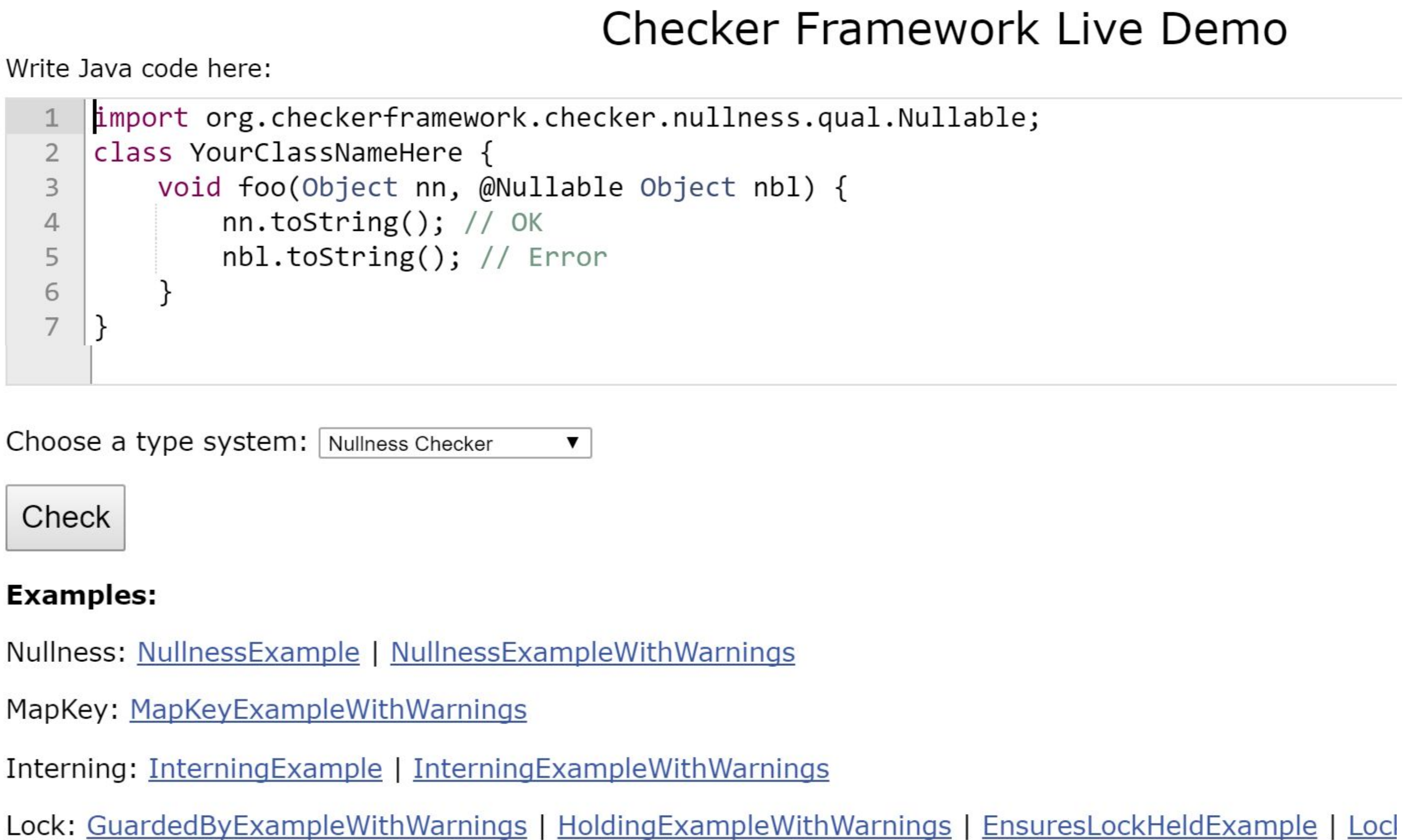
```java
public class Test {

    public static void main(String[] args) {
        Console c = System.console();
dereference of possibly-null reference c c.printf("Test");
    }
```

Problems | Javadoc | Declaration | Search | Console | Task

0 errors, 1 warning, 0 others

Description	Resource
Warnings (1 item)	
dereference of possibly-null reference c c.printf("Test");	Test.java

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



Checker Framework Live Demo

Write Java code here:

```java
import org.checkerframework.checker.nullness.qual.Nullable;
class YourClassNameHere {
    void foo(Object nn, @Nullable Object nbl) {
        nn.toString(); // OK
        nbl.toString(); // Error
    }
}
```

Choose a type system: Nullness Checker

Examples:

Nullness: NullnessExample | NullnessExampleWithWarnings

MapKey: MapKeyExampleWithWarnings

Interning: InterningExample | InterningExampleWithWarnings

Lock: GuardedByExampleWithWarnings | HoldingExampleWithWarnings | EnsuresLockHeldExample | Locl

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)

✓

Example type systems

Null dereferences (`@Nullable`)

>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
  - **`@Nullable`**: 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

$\mathtt{h}$	$\in$	Heap	$=$	$\mathtt{Addr} \rightarrow \mathtt{Obj}$
ι	$\in$	Addr	$=$	Set of Addresses $\cup \{\mathtt{null}_a\}$
$\mathtt{o}$	$\in$	Obj	$=$	${}^{r}\mathtt{Type}$, Fields
${}^{r}\mathtt{T}$	$\in$	^{r}Type	$=$	$\mathtt{OwnerAddr\ ClassId}\mathtt{<}\overline{{}^{r}\mathtt{Type}}\mathtt{>}$
Fs	$\in$	Fields	$=$	$\mathtt{FieldId} \rightarrow \mathtt{Addr}$
ι	$\in$	OwnerAddr	$=$	$\mathtt{Addr} \cup \{\mathtt{any}_a\}$
${}^{r}\Gamma$	$\in$	^{r}Env	$=$	$\overline{\mathtt{TVarId}\ {}^{r}\mathtt{Type}};\ \overline{\mathtt{ParId\ Addr}}$

P	$\in$	Program	::=	$\overline{\mathtt{Class}}$, ClassId, Expr
Cls	$\in$	Class	::=	class ClassId<$\overline{\mathtt{TVarI}}$
				extends ClassId<$\overline{{}^{s}\mathtt{Ty}}$
				{ $\overline{\mathtt{FieldId}\ {}^{s}\mathtt{Type};}$ $\overline{\mathtt{Met}}$
${}^{s}\mathtt{T}$	$\in$	^{s}Type	::=	^{s}NType | TVarId
${}^{s}\mathtt{N}$	$\in$	^{s}NType	::=	OM ClassId<$\overline{{}^{s}\mathtt{Type}}$>
u	$\in$	OM	::=	
mt	$\in$	Meth	::=	
		MethSig	::=	
w	$\in$	Purity	::=	
e	$\in$	Expr	::=	
				Expr.MethId<^{s}Type>(Expr) |
				new ^{s}Type | (^{s}Type) Expr
${}^{s}\Gamma$	$\in$	^{s}Env	::=	$\overline{\mathtt{TVarId}\ {}^{s}\mathtt{NType};}$ $\overline{\mathtt{ParId}\ {}^{s}\mathtt{Type}}$

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

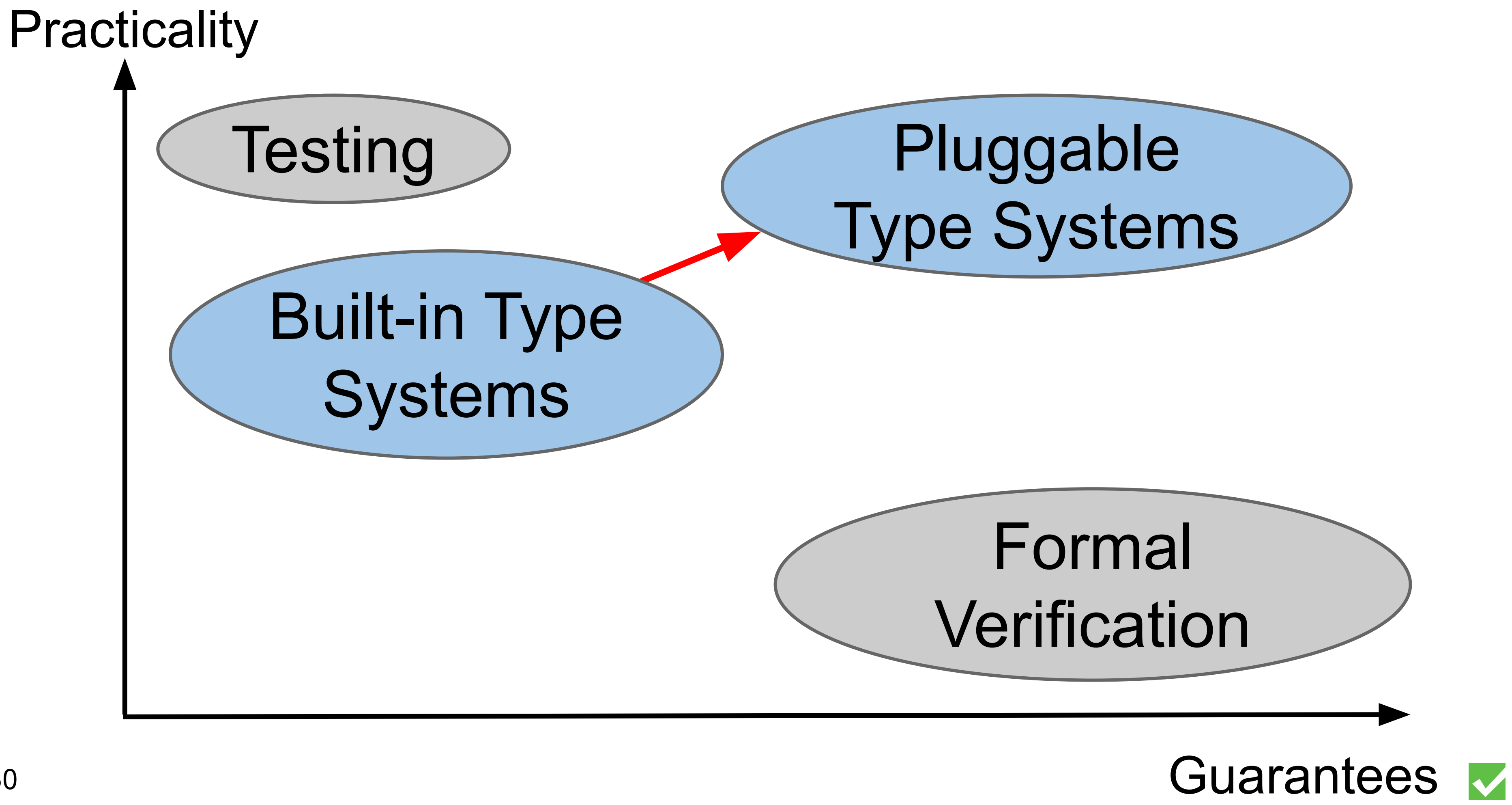
$$\text{OS-Upd}\ \frac{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \qquad \iota_0 \neq \mathtt{null}_a \qquad \mathtt{h}_0, {}^{r}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \qquad \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]}{\mathtt{h}, {}^{r}\Gamma,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

$$\text{GT-Upd}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\mathtt{<\_>} \qquad \mathtt{T}_1 = \mathit{fType}(\mathtt{C}_0, \mathtt{f}) \qquad \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1 \qquad \mathtt{u}_0 \neq \mathtt{any} \qquad \mathit{rp}(\mathtt{u}_0, \mathtt{T}_1)}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1}$$

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

$$\left.\begin{array}{l} \mathtt{h} \vdash {}^{r}\Gamma : {}^{s}\Gamma \\ \mathtt{h} \vdash \iota_1 : \mathit{dyn}({}^{s}\mathtt{N}, \mathtt{h}, {}^{r}\Gamma) \\ \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{s}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\mathtt{<\_>} \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{r}\Gamma(\mathtt{this}) \\ \mathit{free}({}^{s}\mathtt{T}) \subseteq \mathit{dom}(\mathtt{C}_N) \end{array}\right\} \Longrightarrow \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{N} \triangleright {}^{s}\mathtt{T}, \mathtt{h}, {}^{r}\Gamma)$$

$$\text{DYN}\ \frac{{}^{r}\mathtt{T} = \iota'\ \mathtt{\_<\_>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\mathtt{<:}\ \iota'\ \mathtt{C}\mathtt{<}\overline{{}^{r}\mathtt{T}}\mathtt{>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\mathtt{<:}\ \iota'\ \mathtt{C}\mathtt{<}\overline{{}^{r}\mathtt{T}_a}\mathtt{>} \Rightarrow \iota \vdash \overline{{}^{r}\mathtt{T}}\ {}^{r}\mathtt{<:}\ \overline{{}^{r}\mathtt{T}_a} \qquad \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathrm{free}({}^{s}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}}{\mathit{dyn}({}^{s}\mathtt{T}, \iota, {}^{r}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{r}\mathtt{T}'}; \_)) = {}^{s}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{r}\mathtt{T}/\mathtt{X}}, \overline{{}^{r}\mathtt{T}'/\mathtt{X}'}]}$$


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 code with bugs

```
// Takes a regex and a string to match against
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 code with bugs

```
// Takes a regex and a string to match against
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 <u>non-null</u> reference to my data

An <u>interned</u> string

A <u>non-null</u> List of <u>English</u> strings

A <u>non-empty</u> array of <u>English</u> 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.

Two syntaxes:

- As separate text file (“stub file”)
- Within source code (compiler puts in .jar file)

Checker Framework facilities

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

Developing 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 reference is 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 reference is 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,

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 {}
```

✓

Defining a type system

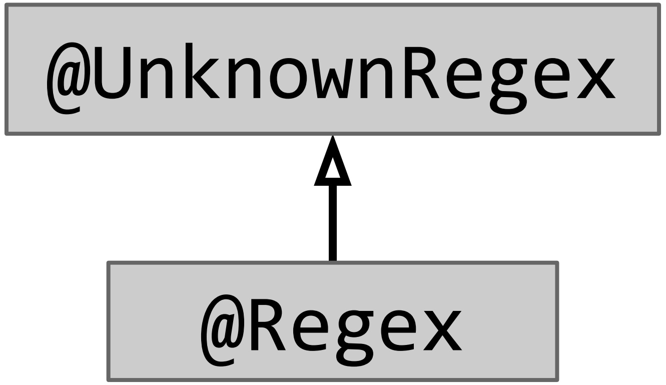
1. Qualifier hierarchy
    - defines subtyping
2. Type introduction rules
    - types for expressions
3. Type rules
    - checker-specific errors
4. Dataflow refinement
    - better types than the programmer wrote

✓

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(myVar) {
}
```

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. Dataflow refinement
    - better types than the programmer wrote

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

✓

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");
```

✓

Dataflow Framework

Goal: Compute properties about expressions

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

To define a new analysis:

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

Dataflow Framework does all the work!

More at JAX 2020

Implement your own Type System today!

Mittwoch, 9. September 2020

17:00 - 18:00

Raum: Westfoyer

Tips for using the Checker Framework

- Use subclasses (not type qualifiers) if possible

- Start by type-checking part of your code
- Only type-check properties that matter to you
- 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
- >15,700 commits, 105 authors
- Welcoming & responsive community

Checker Framework plans

More type systems:

- Immutability
- Determinism
- Signed vs. unsigned numbers

Type inference

Combined static & dynamic enforcement

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/`

Acknowledgement: Slides developed with Michael Ernst, University of Washington. ✓