Preventing Null Pointer Exceptions at Compile Time

The Checker Framework

`http://CheckerFramework.org/`

Twitter: @CheckerFrmwrk

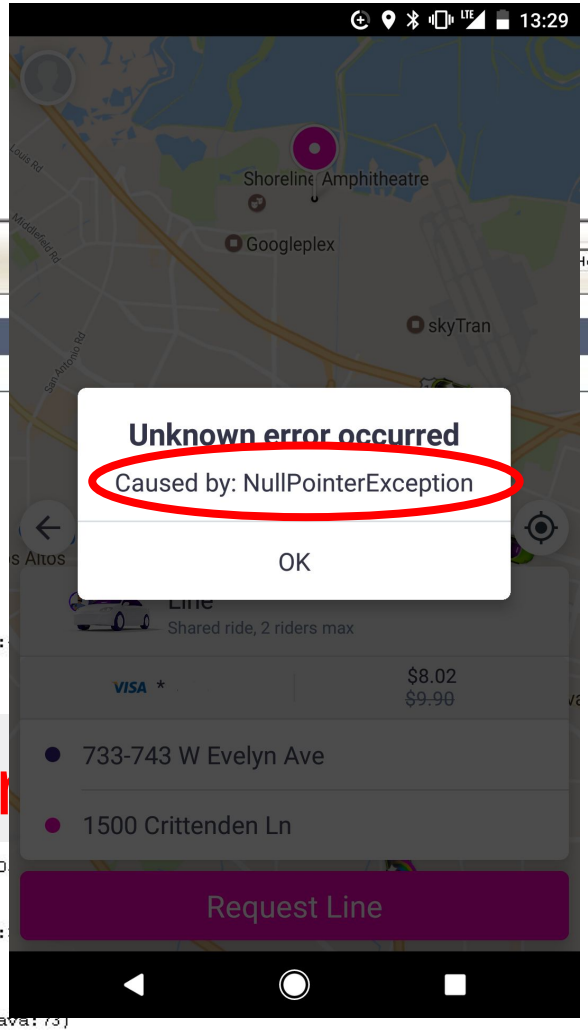
Live demo: `http://CheckerFramework.org/live/`

Werner Dietl, University of Waterloo

UNIVERSITY OF WATERLOO

Motivation


TREND MICRO
InterScan™ Web Security Virtual Appliance
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:
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)
root cause
java.lang.NullPointerException
org.apache.jsp.urlf_005fsection_005fpolicy_005frule_jsp._jspService(urlf_00
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:
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.NullPointer


13:29
Shoreline Amphitheatre
Googleplex
skyTran
Unknown error occurred
Caused by: NullPointerException
OK
Shared ride, 2 riders max
VISA *
$8.02
$9.90
733-743 W Evelyn Ave
1500 Crittenden Ln
Request Line

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

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

2011: Software caused 25% of all medical device recalls

Outline

- Verification approach: Pluggable type-checking
- Tool: Checker Framework
- How to use it
- Advanced features

This talk focuses on nullness.

Talk today at 15:30-16:30 expands on other properties and the Checker Framework in general.

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();
```


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();
```


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();
```

Nullness Checker demo Fixed

Prevent null pointer exceptions

Goal: the program only dereferences 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);
```


Solution: Pluggable Type Checking

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

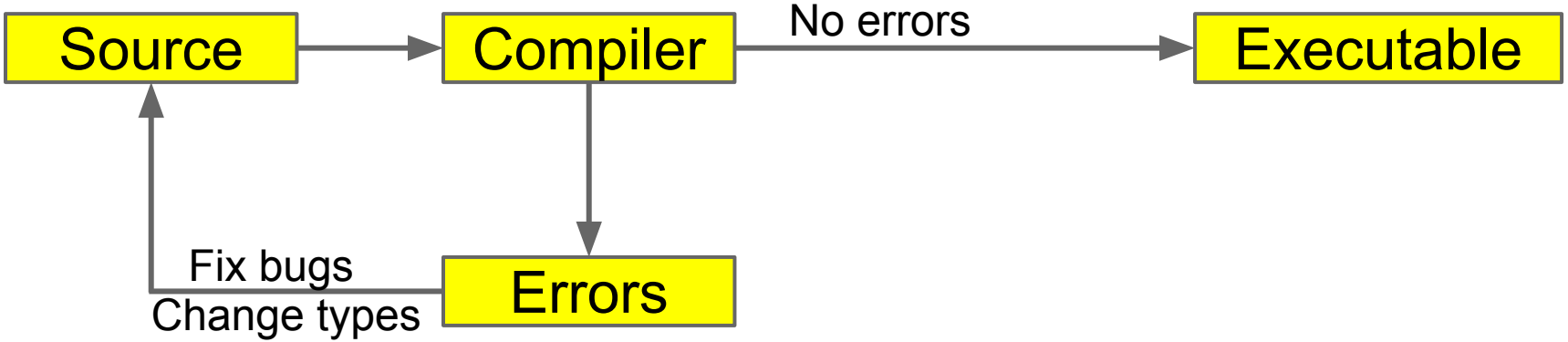
```
void foo (@Nullable Date date) {
  date.setSeconds(0); // compile-time error
```

3. Type checker warns about violations (bugs)

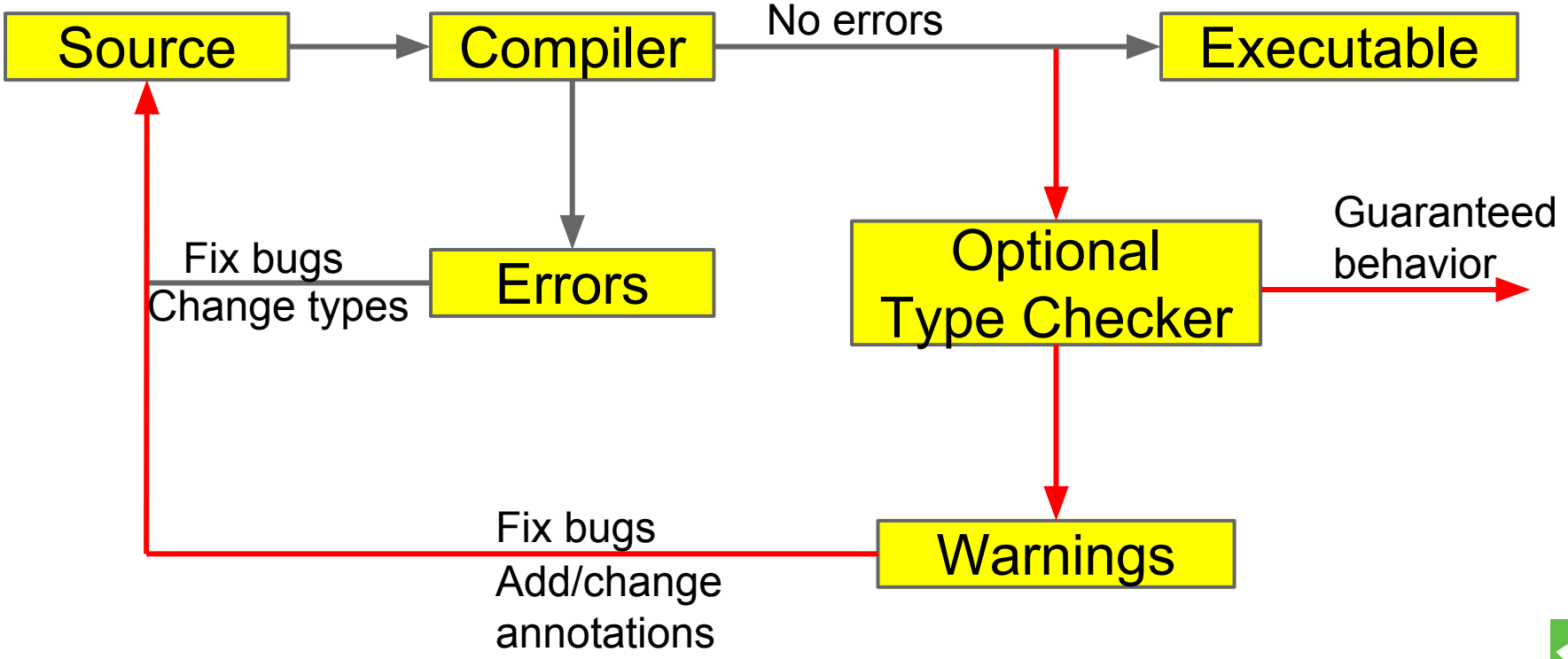
```
% javac -processor NullnessChecker MyFile.java

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

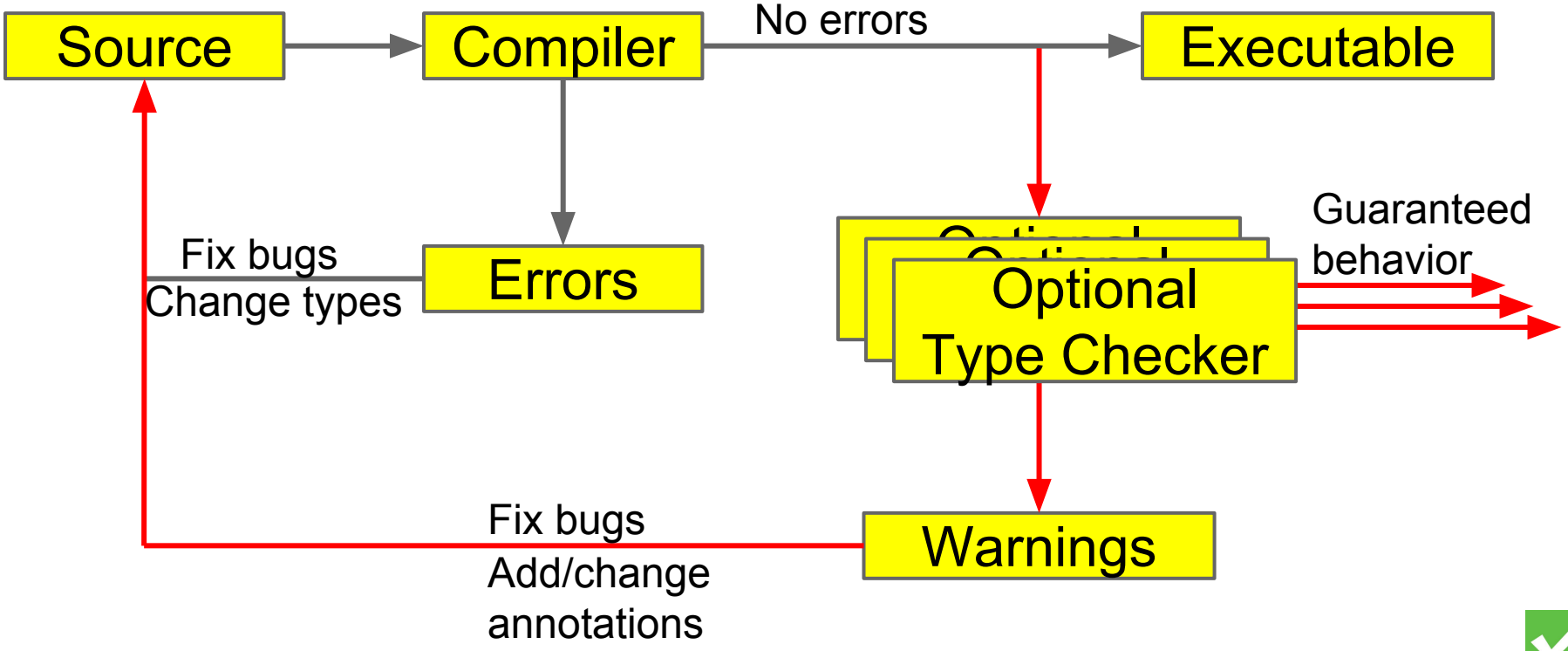

Type Checking



Optional Type Checking



Optional Type Checking



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 ...
```

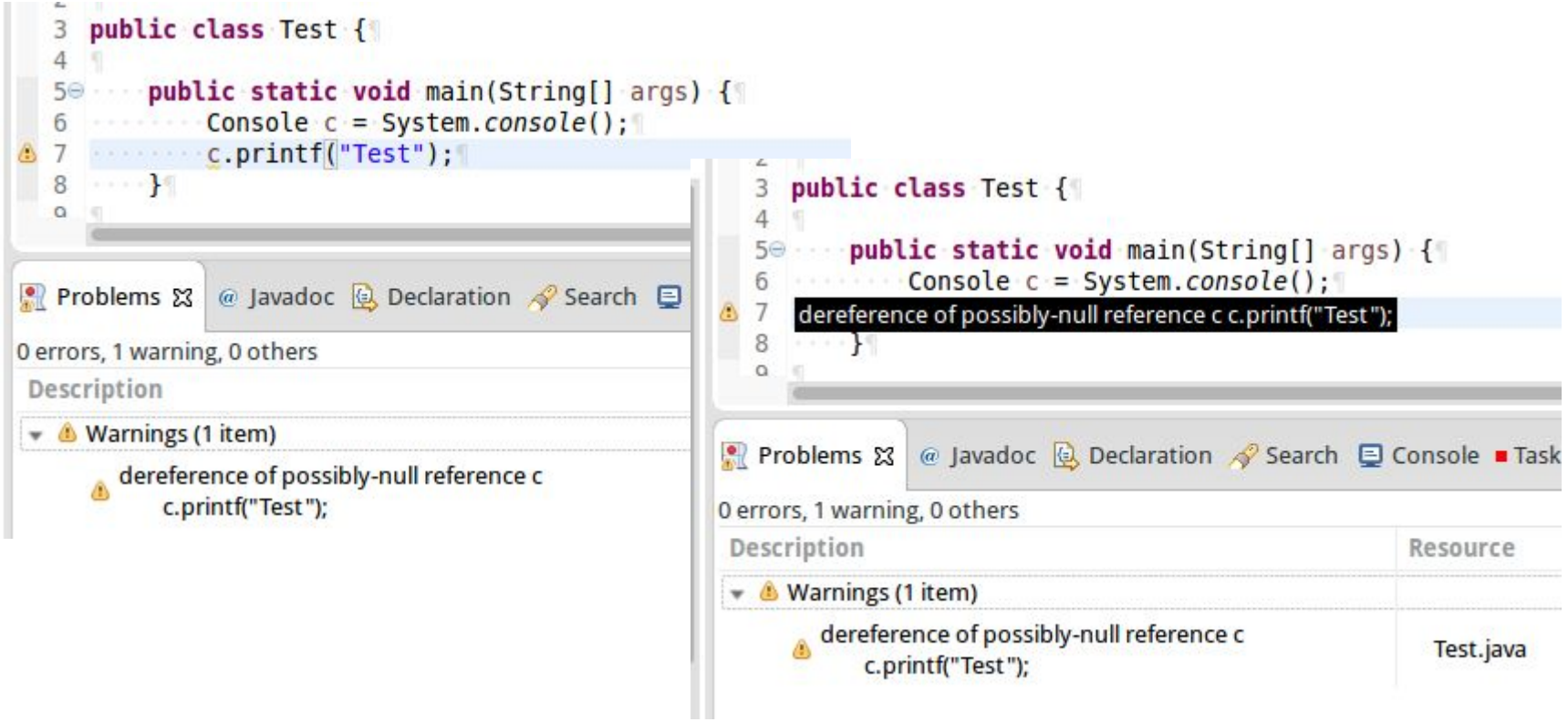
Standard error format allows tool integration

Ant, Maven, Gradle integration

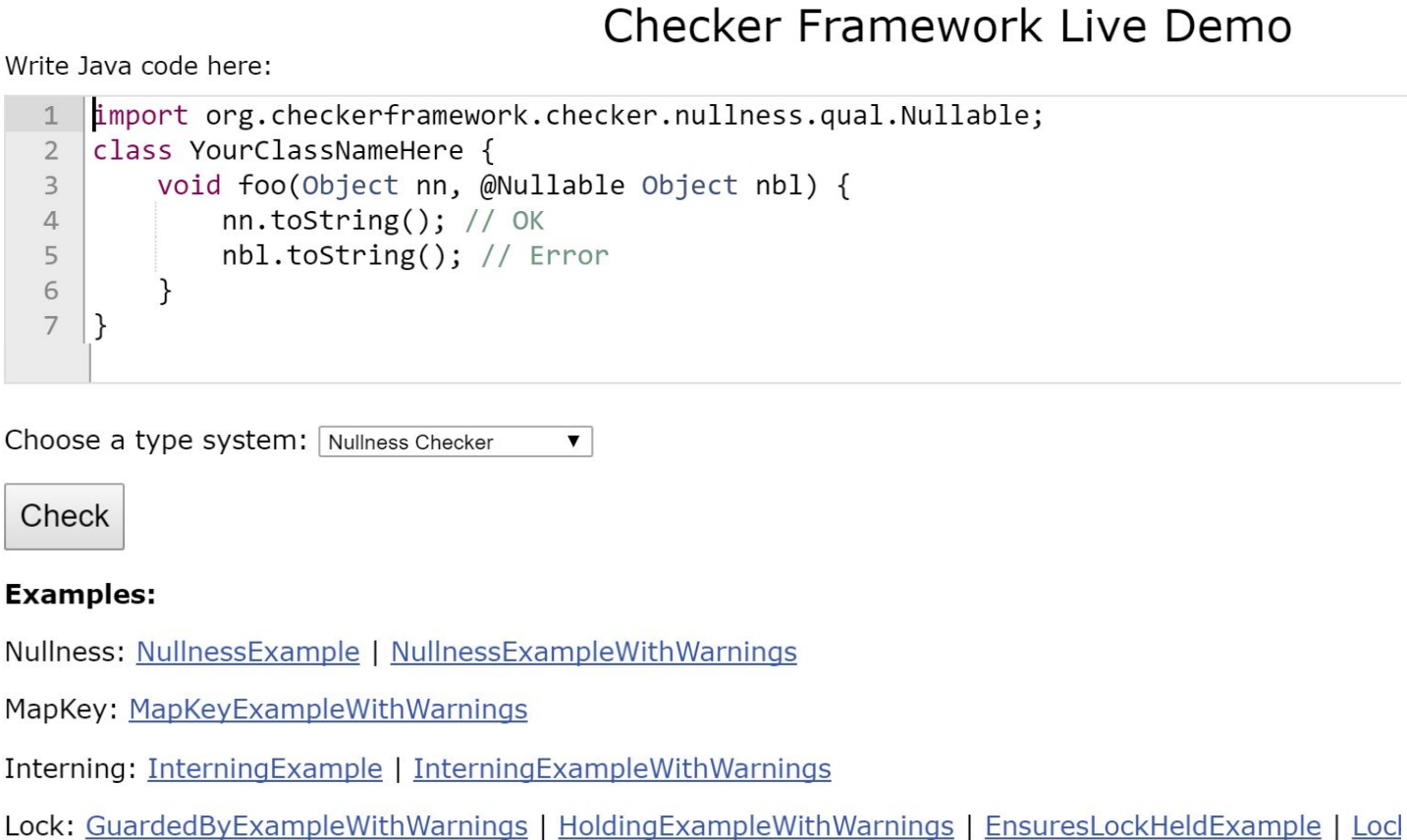
```
<presetdef name="jsr308.javac">
  <javac fork="yes"
    executable="${checkerframework}/checker/bin/${cfJavac}" >
    <!-- JSR-308-related compiler arguments -->
    <compilerarg value="-version"/>
    <compilerarg value="-implicit:class"/>
  </javac>
</presetdef>
```

```
<dependencies>
  ... existing <dependency> items ...
  <!-- annotations from the Checker Framework:
          nullness, interning, locking, ... -->
    <dependency>
      <groupId>org.checkerframework</groupId>
      <artifactId>checker-qual</artifactId>
      <version>2.5.0</version>
    </dependency>
</dependencies>
```

Eclipse, IntelliJ, NetBeans integration



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



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

With Java 8 Type Annotations we can!

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;
```


Java 8 extends 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, …

Java 6 & 7 compatibility
(or avoid dependency on Checker Framework)

Annotations in comments:

```
List</*@NonNull*/ String> strings;
```

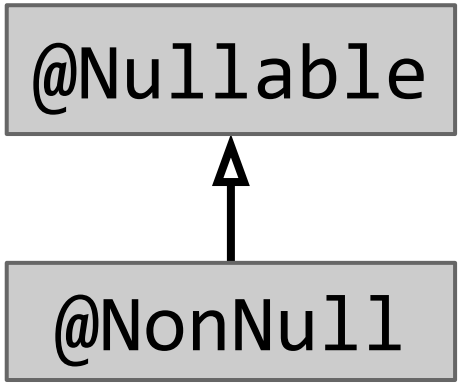
(Requires use of jsr308-langtools compiler.)

Preventing null-pointer exceptions

Basic type system:

`@Nullable` might be null

`@NonNull` definitely not null



Default is `@NonNull` (opposite of Java's default)

- Requires fewer annotations
- Makes the dangerous case explicit

(Nearly) no annotations in method bodies!

Flow-sensitive type refinement

```
if (myField != null) {

  myField.hashCode();
}
```

No need to declare a new local variable

One check for `null` is not enough

```
if (myField != null) {
  method1();
  myField.hashCode();
}
```

3 ways to express persistence across side effects:

```
@SideEffectFree void method1() { … }
@MonotonicNonNull myField;
@EnsuresNonNull(“myField”) method1() {…}
```


Side effects

`@SideEffectFree`

Does not modify externally-visible state

`@Deterministic`

If called with == args again, gives == result

`@Pure`

Both side-effect-free and deterministic

The side-effect annotations are trusted, not checked

Lazy initialization and persistence across side effects

`@MonotonicNonNull`

Might be null or non-null

May only be (re-)assigned a non-null value

Purpose: avoid re-checking

Once non-null, always non-null

Method pre- and post-conditions

Preconditions:

```
@RequiresNonNull
```

Postconditions:

```
@EnsuresNonNull
@EnsuresNonNullIf
    @EnsuresNonNullIf(expression="#1", result=true)
    public boolean equals(@Nullable Object obj) { ... }
```


Polymorphism over qualifiers

```
/** Interns a String, and handles null. */
@PolyNull String intern(@PolyNull String a) {
  if (a == null) {
    return null;
  }
  return a.intern();
}
```

Like defining two methods:

```
@NonNull  String intern(@NonNull  String a) {…}
@Nullable String intern(@Nullable String a) {…}
```


A non-null field might contain `null`

```
@NonNull String name;
MyClass() {  // constructor
 … this.name.hashCode() …
}
```

Initialization

`@Initialized` (constructor has completed)

`@UnderInitialization(Frame.class)`

Its constructor is currently executing

`@UnknownInitialization(Frame.class)`

Might be initialized or under initialization

Map keys and `Map.get`

```
Map<String, @NonNull Integer> gifts;
... gifts.get("pipers piping").intValue() ...
```

`Map.get` can return null! ... unless

- value type is non-null, **and**
- argument key appears in the map

`@KeyFor` [rarely written, usually inferred]

Map key example

```
/** Computes predominators for each node in the graph. */
<T> Map<T, List<T>>
dominators(Map<T, List<@KeyFor("#1") T>> predecessors) {
  ...
  for (T node : predecessors.keySet()) {
    for (T pred : predecessors.get(node)) {   // no NPE
      ... predecessors.get(pred) ...          // no NPE
```


Suppressing warnings

Because of Checker Framework false positives

```
@SuppressWarnings(“nullness”)
```

- Use smallest possible scope (e.g., local var)
- Write the rationale as a comment

```
assert x != null : “@AssumeAssertion(nullness)”;
```

More: https://checkerframework.org/manual/#suppressing-warnings

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

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. Flow-refinement
   - better types than the programmer wrote

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!

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!

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)

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

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

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

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

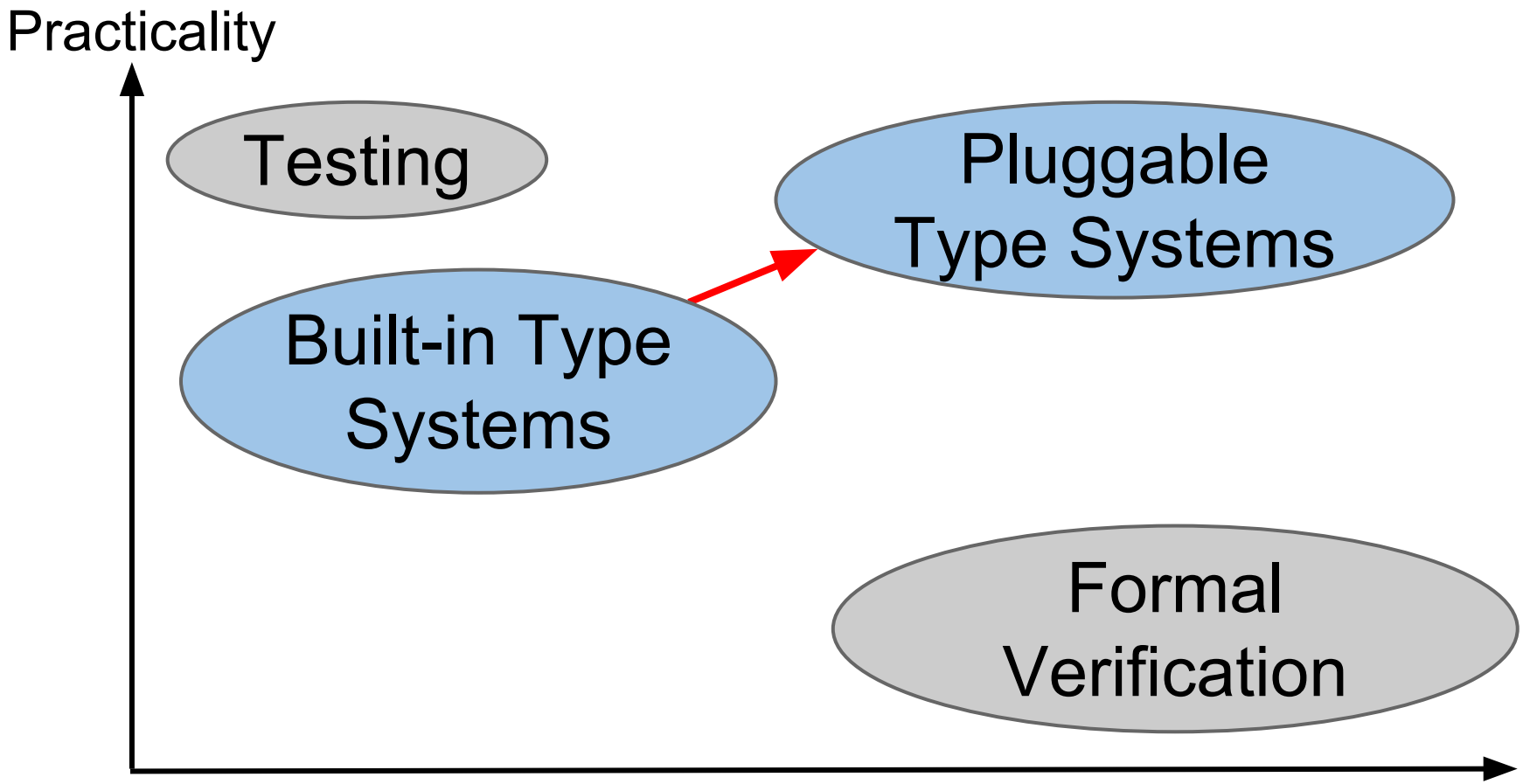
$$
\text{OS-Read}\ \frac{\begin{array}{c}\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})\end{array}}{\mathtt{h}, {}^{\mathtt{r}}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow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

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

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

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

$$
\left.\begin{array}{l}{}^{\mathtt{r}}\mathtt{T} = \iota'\ \_\langle\_\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}\langle: \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}}\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}\langle: \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}_a}\rangle \Rightarrow \iota \vdash \overline{{}^{\mathtt{r}}\mathtt{T}}\ {}^{\mathtt{r}}\langle: \overline{{}^{\mathtt{r}}\mathtt{T}_a} \\ \mathtt{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathtt{free}({}^{\mathtt{s}}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}\right\} \implies \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{N} \rhd {}^{\mathtt{s}}\mathtt{T}, \mathtt{h}, {}^{\mathtt{r}}\Gamma)
$$


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

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.

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
- >13,000 commits, 75 authors
- Welcoming & responsive community
- Google Summer-of-Code participant

Checker Framework plans

More type systems:

- Immutability
- Index-out-of-bounds
- `Optional<T>` type
- Determinism
- Signed vs. unsigned numbers

Type inference

Combined static & dynamic enforcement

More at JAX 2018

Preventing Errors before they happen – Using the Checker Framework

Today, 15:30-16:30

Zagreb B

Pluggable type-checking improves code

Checker Framework for optional type checkers

- Featureful, effective, easy to use, scalable

Prevent null pointers at compile time

Improve your code!

`http://CheckerFramework.org/`