A Type System for Format Strings

Konstantin Weitz	weitzkon@uw.edu
Gene Kim	genelkim@uw.edu
Siwakorn Srisakaokul	ping128@uw.edu
Michael D. Ernst	mernst@uw.edu

Format String APIs

```
printf(“name: %s age: %d”,
       “Konstantin”, 25);
```

“name: Konstantin age: 25”

Format String APIs

```
printf("name: %s age: %d",
       "Konstantin", 25);
```

"name: Konstantin age: 25"

Problem: easy to misuse

Implications of Misuse

- Unintelligible Output

```
printf(“cannot open %s”);

> cannot open ▯oN▯
```

Implications of Misuse

- Unintelligible Output
- Program Crash

```
printf(“%d”, “str”);
```

Implications of Misuse

- Unintelligible Output
- Program Crash
- Security Vulnerability

```
printf("%.*d%n",
   attack_code, 0,
   return_addr);
```

Root Causes of Misuse

- Invalid Format String Syntax

```
printf(“%y”);
```

Root Causes of Misuse

- Invalid Format String Syntax
- Wrong Number of Arguments

```
printf(“%d %s”, 42);
```

Root Causes of Misuse

- Invalid Format String Syntax
- Wrong Number of Arguments
- Wrong Type of Arguments

```
printf(“%d”, 7.0);
```

Goal

Statically guarantee that
format methods are not misused

Goal

Statically guarantee that
format methods are not misused

- Verify Format String Syntax

Goal

Statically guarantee that
format methods are not misused

- Verify Format String Syntax
- Verify Number of Arguments

Goal

Statically guarantee that
format methods are not misused

- Verify Format String Syntax
- Verify Number of Arguments
- Verify Type of Arguments

Goal

Statically guarantee that
format methods are not misused

- Verify Format String Syntax
- Verify Number of Arguments
- Verify Type of Arguments
- Ease of Use

Types Prevent Errors

```
var fs;




printf(fs, 5);
```

Types Prevent Errors

```
var fs;
fs = 42;
fs = “%y”;
fs = “%d %c”;
fs = “%f”;
fs = “%d”;
printf(fs, 5);
```

Types Prevent Errors

```
var fs;
fs = 42;
fs = “%y”;      // invalid syntax
fs = “%d %c”; // invalid number of args
fs = “%f”;      // invalid type of args
fs = “%d”;
printf(fs, 5);
```

Types Prevent Errors

```
String fs;
fs = 42;
fs = "%y";      // invalid syntax
fs = "%d %c"; // invalid number of args
fs = "%f";      // invalid type of args
fs = "%d";
printf(fs, 5);
```

Types Prevent Errors

```
@Format String fs;
fs = 42;
fs = “%y”;      // invalid syntax
fs = “%d %c”; // invalid number of args
fs = “%f”;      // invalid type of args
fs = “%d”;
printf(fs, 5);
```

Types Prevent Errors

```
@Format(INT) String fs;
fs = 42;
fs = "%y";        // invalid syntax
fs = "%d %c";     // invalid number of args
fs = "%f";        // invalid type of args
fs = "%d";
printf(fs, 5);
```

Types Prevent Errors

Conversion Category

```
@Format(INT) String fs;
fs = 42;
fs = "%y";      // invalid syntax
fs = "%d %c";   // invalid number of args
fs = "%f";      // invalid type of args
fs = "%d";
printf(fs, 5);
```

Java Conversion Categories

```
printf("%d", (T)v );
```

$T \in \boxed{\text{INT}}$

=

{Byte, Short, Integer, Long}

Java Conversion Categories

```
printf("%f", (T)v );
```

$\boxed{\text{INT}} = \{\texttt{Byte, Short, Integer, Long}\}$

$T \in \boxed{\text{FLOAT}} = \{\texttt{Float, Double}\}$

Java Conversion Categories

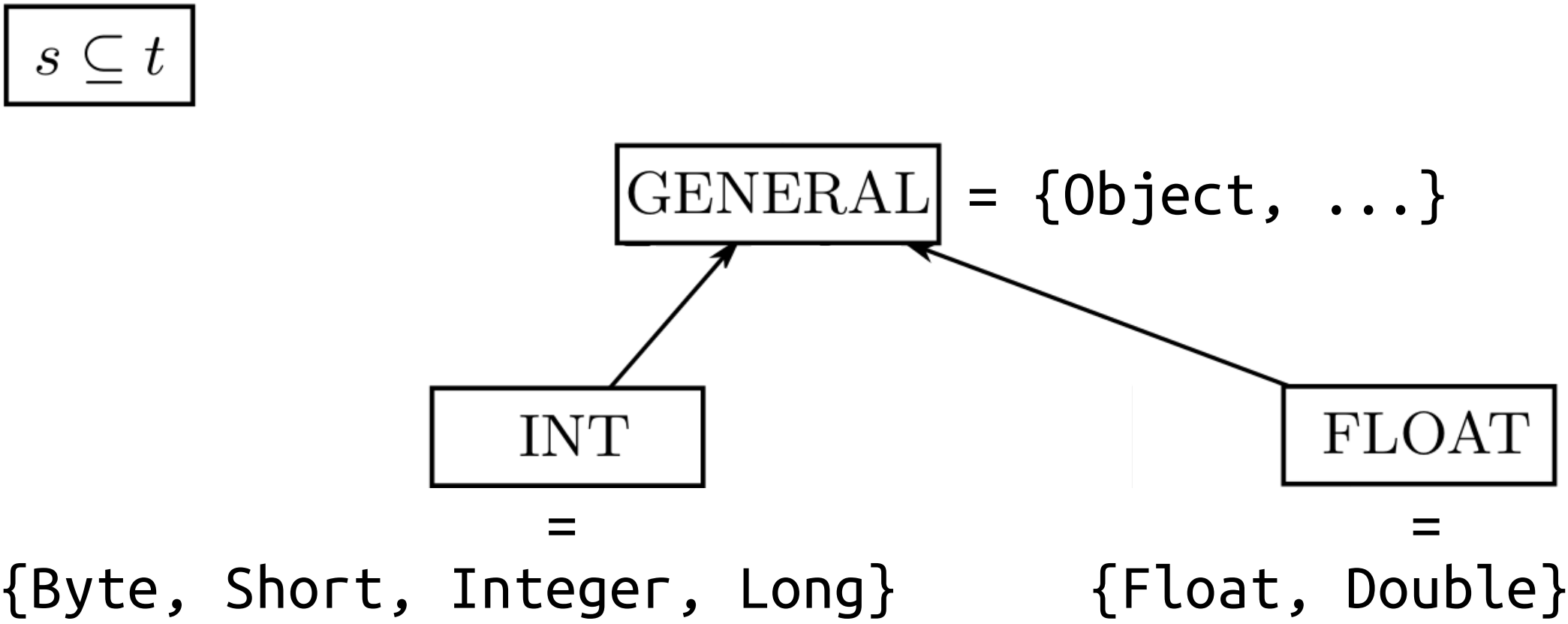
```
printf("%s", (T)v );
```

$$T \in \boxed{\text{GENERAL}} = \{\texttt{Object}, \ldots\}$$

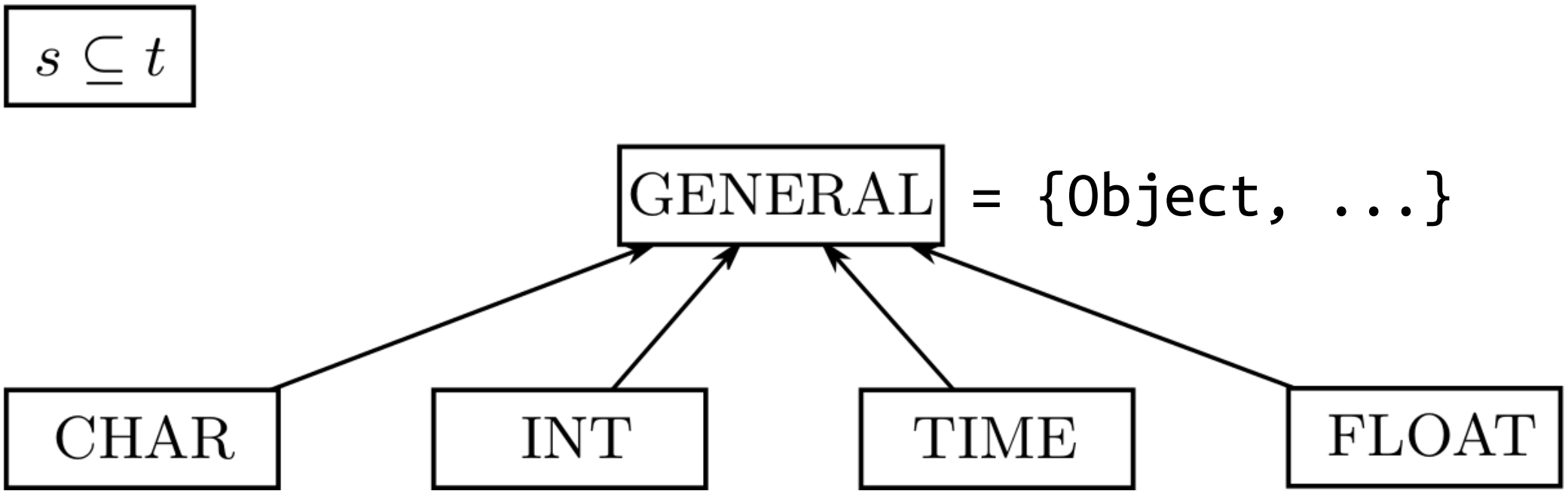
$$\boxed{\text{INT}} = \{\texttt{Byte}, \texttt{Short}, \texttt{Integer}, \texttt{Long}\}$$

$$\boxed{\text{FLOAT}} = \{\texttt{Float}, \texttt{Double}\}$$

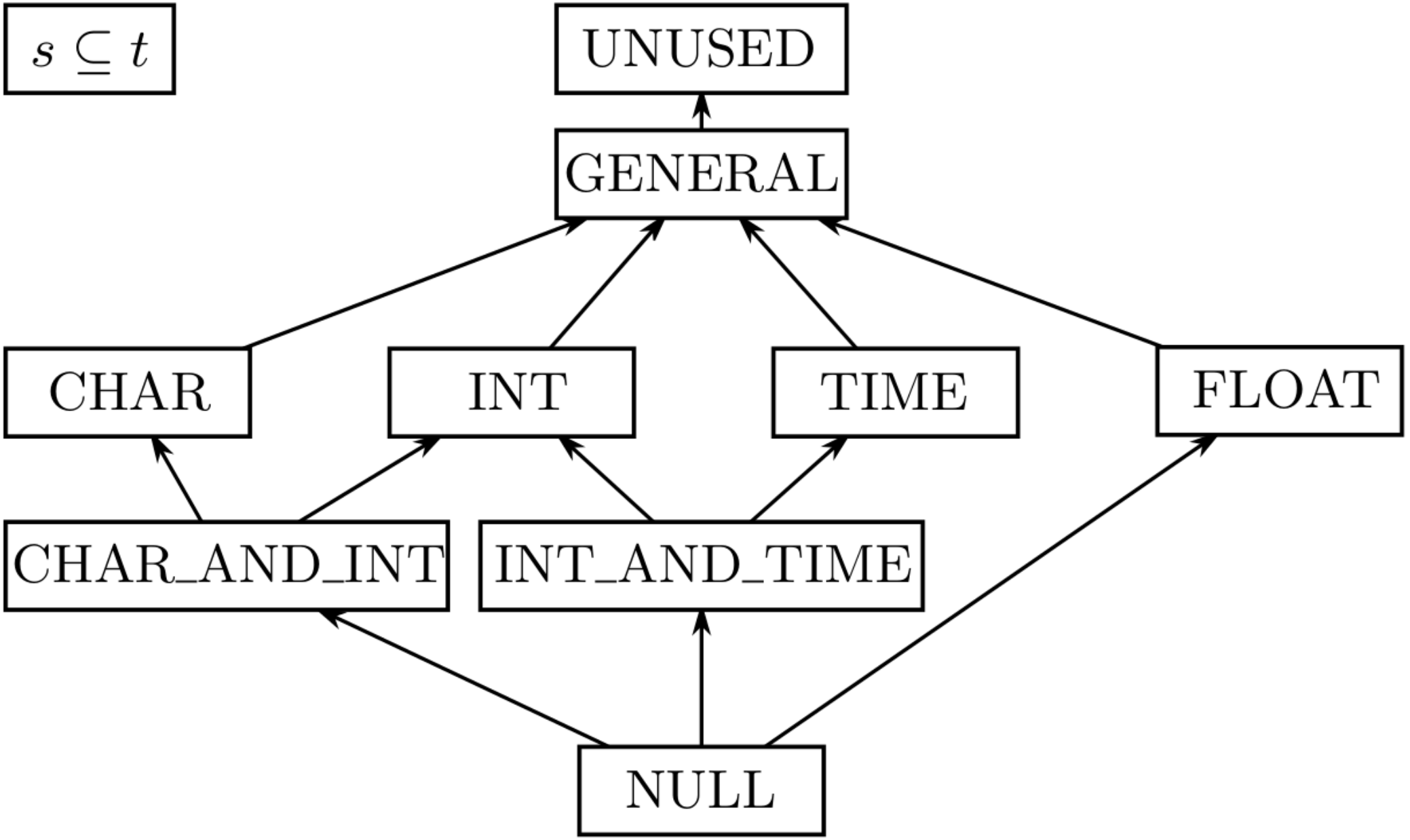
Java Conversion Categories



Java Conversion Categories



Java Conversion Categories



Subtyping

```
@Format(FLOAT) String fs;

printf(fs, 3.14);
```

Subtyping

```
@Format(FLOAT) String fs;

fs = “%f”   // ok
fs = “%s”   // ok: %s weaker than %f
fs = “  ”   // ok: argument ignored

printf(fs, 3.14);
```

Subtyping

$$\boxed{S \leq T}$$

$$\frac{\forall i \in 0..n\ t_i \subseteq s_i}{@\mathrm{Format}(s_0, \ldots, s_n) \leq @\mathrm{Format}(t_0, \ldots, t_n, \ldots, t_{n+k})}$$

```
@Format(FLOAT) String fs;

fs = “%f”   // ok
fs = “%s”   // ok: %s weaker than %f
fs = “  ”   // ok: argument ignored

printf(fs, 3.14);
```

Subtyping

$$\boxed{S \leq T}$$

$$\frac{\forall i \in 0..n \; t_i \subseteq s_i}{@\mathrm{Format}(s_0, \ldots, s_n) \leq @\mathrm{Format}(t_0, \ldots, t_n, \ldots, t_{n+k})}$$

```
@Format(FLOAT) String fs;

fs = “%f”   // ok
fs = “%s”   // ok: %s weaker than %f
fs = “  ”   // ok: argument ignored

printf(fs, 3.14);
```

Polymorphism

```
void log(String fs,
         Object... args)
{
  printf(fs, args);
}

log(“%f”, 3.14);
log(“%d”, 1337);
```

Polymorphism

```
void log(@FormatFor(“args”) String fs,
         Object... args)
{
  printf(fs, args);
}

log(“%f”, 3.14);
log(“%d”, 1337);
```

Complex Format Strings

```
@Format(FLOAT,GENERAL) String
fs = “%2$s = %1$+10.4f”;

printf(fs, 3.14, “pi”);
```

Type System Instantiation

- C's printf API

  `“%s”`

- Go's fmt module

  `“%[1]s”`

- Java's i18n API

  `“{0}”`

- Java's Formatter API

  `“%1$s”`

Goal

Statically guarantee that
format methods are not misused

- Verify Format String Syntax
- Verify Number of Arguments
- Verify Type of Arguments
- Ease of Use

Goal

Statically guarantee that
format methods are not misused

- ✔ Verify Format String Syntax
- Verify Number of Arguments
- Verify Type of Arguments
- Ease of Use

Goal

Statically guarantee that
format methods are not misused

- ✔ Verify Format String Syntax
- ✔ Verify Number of Arguments
- ✔ Verify Type of Arguments
- Ease of Use

Goal

Statically guarantee that
format methods are not misused

- ✔ Verify Format String Syntax
- ✔ Verify Number of Arguments
- ✔ Verify Type of Arguments
- Ease of Use ?

Evaluation

Project	LoC	Bugs	
		Submit	Fixed
Hadoop	678k	3	2
Hive	538k	1	0
Lucene	664k	0	0
HBase	569k	2	2
Daikon	205k	95	95
FindBugs	122k	3	3
Total	**2777k**	**104**	**102**

Evaluation - Usage Effort

Project	Format Calls	Type Annotations		False Positives	Bugs
		@Format	@FormatFor	@Suppress Warnings	
Hadoop	332	20	6	22	3
Hive	213	0	1	7	1
Lucene	148	2	0	0	0
HBase	96	0	0	1	2
Daikon	1583	0	30	7	95
FindBugs	133	7	1	3	3
Total	**2505**	**29**	**38**	**40**	**104**

Evaluation - Usage Effort

Project	Format Calls	Type Annotations		False Positives	Bugs
		@Format	@FormatFor	@Suppress Warnings	
Hadoop	332	20	6	22	3
Hive	213	0	1	7	1
Lucene	148	2	0	0	0
HBase	96	0	0	1	2
Daikon	1583	0	30	7	95
FindBugs	133	7	1	3	3
Total	**2505**	**29**	**38**	**40**	**104**

Annotation Burden 107

Evaluation - Usage Effort

Project	Format Calls	Type Annotations		False Positives	Bugs
		@Format	@FormatFor	@Suppress Warnings	
Hadoop	332	20	6	22	3
Hive	213	0	1	7	1
Lucene	148	2	0	0	0
HBase	96	0	0	1	2
Daikon	1583	0	30	7	95
FindBugs	133	7	1	3	3
Total	**2505**	**29**	**38**	**40**	**104**

Annotation Burden 107

Bugs Revealed 104

Evaluation - Usage Effort

Project	Format Calls	Type Annotations		False Positives	Bugs
		@Format	@FormatFor	@Suppress Warnings	
Hadoop	332	20	6	22	3
Hive	213	0	1	7	1
Lucene	148	2	0	0	0
HBase	96	0	0	1	2
Daikon	1583	0	30	7	95
FindBugs	133	7	1	3	3
Total	**2505**	**29**	**38**	**40**	**104**

$$\frac{\text{Annotation Burden}}{\text{Bugs Revealed}} = \frac{107}{104} = 1.0$$

Evaluation - Usage Effort

Project	Format Calls	Type Annotations		False Positives	Bugs
		@Format	@FormatFor	@Suppress Warnings	
Hadoop	332	20	6	22	3
Hive	213	0	1	7	1
Lucene	148	2	0	0	0
HBase	96	0	0	1	2
Daikon	1583	0	30	7	95
FindBugs	133	7	1	3	3
Total	**2505**	**29**	**38**	**40**	**104**

Evaluation – False Positives

Project	Constant Propagation	Dynamic Width	Exception Handled	Misc
Hadoop	10	6	0	6
Hive	3	2	1	1
Lucene	2	0	0	0
HBase	0	0	0	1
Daikon	0	6	0	1
FindBugs	0	0	3	0
Total	**13**	**14**	**4**	**9**

Evaluation – False Positives

Project	Constant Propagation	Dynamic Width	Exception Handled	Misc
Hadoop	10	6	0	6
Hive	3	2	1	1
Lucene	2	0	0	0
HBase	0	0	0	1
Daikon	0	6	0	1
FindBugs	0	0	3	0
Total	**13**	**14**	**4**	**9**

```
printf("%"+"d", 42);
```

Evaluation – False Positives

Project	Constant Propagation	Dynamic Width	Exception Handled	Misc
Hadoop	10	6	0	6
Hive	3	2	1	1
Lucene	2	0	0	0
HBase	0	0	0	1
Daikon	0	6	0	1
FindBugs	0	0	3	0
Total	**13**	**14**	**4**	**9**

```
String fs = “%” + width + “d”;
printf(fs, 42);
```

Evaluation – False Positives

Project	Constant Propagation	Dynamic Width	Exception Handled	Misc
Hadoop	10	6	0	6
Hive	3	2	1	1
Lucene	2	0	0	0
HBase	0	0	0	1
Daikon	0	6	0	1
FindBugs	0	0	3	0
Total	**13**	**14**	**4**	**9**

```
try {
  printf(userInput, 4.12);
} catch (FormatExp e) {/*error handling*/}
```

Evaluation – False Positives

Project	Constant Propagation	Dynamic Width	Exception Handled	Misc
Hadoop	10	6	0	6
Hive	3	2	1	1
Lucene	2	0	0	0
HBase	0	0	0	1
Daikon	0	6	0	1
FindBugs	0	0	3	0
Total	**13**	**14**	**4**	**9**

```
<T> void f(String fs, Iterator<T> iter) {
  System.out.format(fs, iter.next());
}
```

Goal

Statically guarantee that
format methods are not misused

- ✔ Verify Format String Syntax
- ✔ Verify Number of Arguments
- ✔ Verify Type of Arguments
- Ease of Use

Goal

Statically guarantee that
format methods are not misused

- ✔ Verify Format String Syntax
- ✔ Verify Number of Arguments
- ✔ Verify Type of Arguments
- ✔ Ease of Use

Related Work

- Dynamic Checking[0][1][2]

☺ Easy to use

☹ No compile time guarantee

[0] C. Cowan, et al. USENIX Security Symposium. 2001.
[1] T. Tsai, et al. Avaya Labs. 2001.
[2] M. F. Ringenburg and D. Grossman. CCS 2005

Related Work

- Dynamic Checking[0][1][2]
- Alternative APIs[3][4]

```
cout << “We detected ”
     << setw(10) << n << “bugs”;
```

☺ Guarantees no misuse

☹
- No i18n
- Less readable

[3] Danvy. Journal of FP. 1998.
[4] ISO/IEC 14882:2011. C++, 2011.

Related Work

- Dynamic Checking[0][1][2]
- Alternative APIs[3][4]
- Dependent Type Systems[5]

☺
- Expressive
- Guarantees no misuse

☹
- No mainstream language support
- Hard to use

[5] J. Gronski, et.al. SFP Workshop, 2006.

Related Work

- Dynamic Checking[0][1][2]
- Alternative APIs[3][4]
- Dependent Type Systems[5]
- Lightweight Analysis[6][7][8]

☺ Guarantees no misuse ...

☹ … for constant format strings only

[6] Leroy, et al. The OCaml system release 4.01.
[7] GCC -Wformat. gcc.gnu.org/onlinedocs/gcc/Warning-Options.html
[8] Edward Aftandilian, et al. SCAM 2012.

Related Work

- Dynamic Checking[0][1][2]
- Alternative APIs[3][4]
- Dependent Type Systems[5]
- Lightweight Analysis[6][7][8]
- Static Taint Analysis[9]

☺ Guards against format strings from input

☹ Type/number of arguments and syntax not verified

[9] U. Shankar, et al. USENIX Security Symposium. 2001.

Contributions

- Type system with guarantee that format methods are not misused
- Instantiation for Java
- Evaluation shows type system:
  - Finds bugs (104 bugs, 102 fixed)
  - Easy to use (1.0 annotations / bug)

http://checkerframework.org