

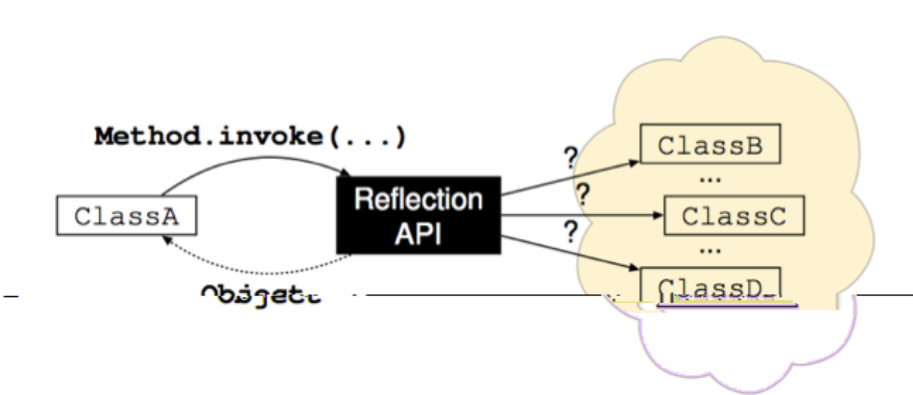
Static Resolution of Implicit Control Flow for Reflection and Message-Passing

Paulo Barros, René Just, Suzanne Millstein,
Paul Vines, Werner Dietl, Marcelo d'Amorim and
Michael D. Ernst

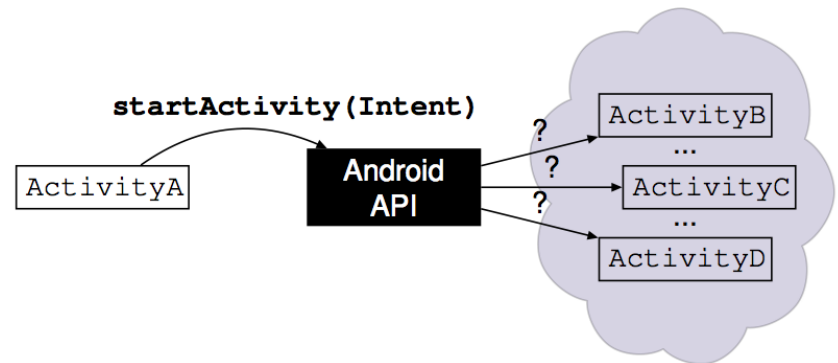


Implicit control flow

- Indirect method call
- Design pattern that allows coding flexibility



Reflection



Message-Passing
(Android Intents)

Problem: imprecise summaries for static analyses

```
...a.foo(b, c);...
```

Problem: imprecise summaries for static analyses

...a.foo(b,c);...



What does foo do?

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`



Use method summary.



What does foo do?

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`



Use method summary.



What does foo do?

`...myMethod.invoke(a,b,c);...`

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`

Use method summary.

What does foo do?

`...myMethod.invoke(a,b,c);...`

What does invoke do?

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`

Use method summary.

What does foo do?

`...myMethod.invoke(a,b,c);...`

Anything!

What does invoke do?

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`

Use method summary.

What does foo do?

`...myMethod.invoke(a,b,c);...`

Anything!

What does invoke do?

- Sound analysis → Imprecise

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`

Use method summary.

What does foo do?

`...myMethod.invoke(a,b,c);...`

Anything!

What does invoke do?

- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe

Problem: imprecise summaries for static analyses

`...a.foo(b,c);...`

Use method summary.

What does foo do?

`...myMethod.invoke(a,b,c);...`

Anything!

What does invoke do?

- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe
- Goal → Soundness and high precision

Android

- Over 1 billion active users
- Over 1.6 million apps
- Analyzing apps is important
- Example: Malware detection
 - Soundness is crucial



Implicit control flow is pervasive in Android



- F-Droid is a repository of Android apps
- F-Droid apps
 - 39% use reflection
 - 69% share data through intents
- **Conclusion** → Static analysis on Android apps must handle implicit control flow

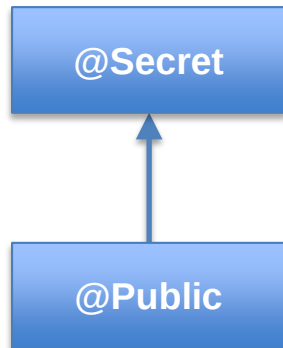
Resolving implicit control flow

- **Goal** → Soundly resolve implicit control flows
- **Observation** → Statically resolvable in F-Droid
 - 93% of reflective calls
 - 88% of sent intents
- **Solution** → We developed type systems that model implicit control flows
- **Results**
 - Improves the precision by 400x
 - Soundness is maintained
 - Low developer effort

Reflection and intents in real apps

Non-interference type system

- Guarantees that the program does not leak sensitive data
- **Privacy-types:**
 - **@Secret:** Sensitive-data values
 - **@Public:** Non-sensitive-data values

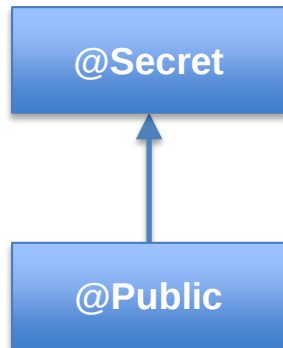


```
@Public String var;  
@Secret String password = getPassword();  
var = password; ❌
```

@Public $\leftarrow$ ❌ @Secret

Non-interference type system

- Guarantees that the program does not leak sensitive data
- **Privacy-types:**
 - **@Secret:** Sensitive-data values
 - **@Public:** Non-sensitive-data values



```
@Public String var;  
@Secret String password = getPassword();  
var = password; ❌
```

@Public $\leftarrow$ ❌ @Secret

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) { ←
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}...
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class; ←
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
}
```

Use of reflection – Aarddict

```
// Library Annotations:  
class Activity {  
    // In Android SDK ≥ 11.  
    @Public ActionBar getActionBar() {...}  
}  
class Method {  
    @Secret Object invoke(Object obj, Object... args) {...}  
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this); ←  
    ...  
}...
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
    // In Android SDK ≥ 11.
    @Public ActionBar getActionBar() {...}
}
class Method {
    @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
    Class<?> clazz = Activity.class;
    Method mtd = clazz.getMethod("getActionBar");
    @Public Object actionBar = mtd.invoke(this);
    ...
    @Public ← @Secret
}...
```

Use of reflection – Aarddict

```
// Library Annotations:  
class Activity {  
    // In Android SDK ≥ 11.  
    @Public ActionBar getActionBar() {...}  
}  
class Method {  
    @Secret Object invoke(Object obj, Object... args) {...}  
}
```

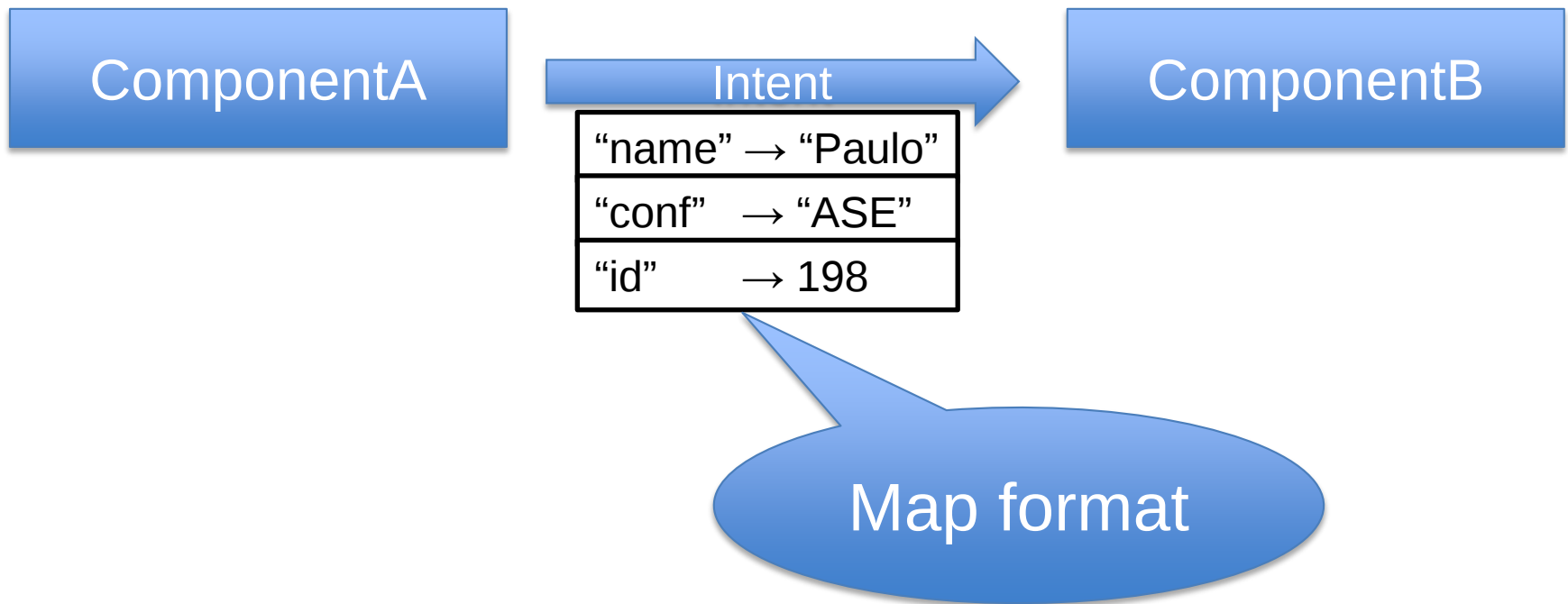
Conservative
annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this); ✓  
    ...  
    @Public ← @Public  
}...
```

Intent payloads



Intent payloads



Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
    {  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
    }  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }...}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent(); ←  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
    }...}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
    {  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
    }  
}
```

~~@Public~~ ← ~~@Secret~~

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    }  
}
```

Conservative
annotation

```
// Library Annotations  
class Intent {  
    @Secret String  
    getStringExtra(String key) {...}  
}
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
    ...  
}
```

@Public ← **@Public**

Reflection Analysis

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this);  
    ...  
}...
```

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this);  
    ...  
}...
```

The type of `clazz` is inferred to represent `Activity`

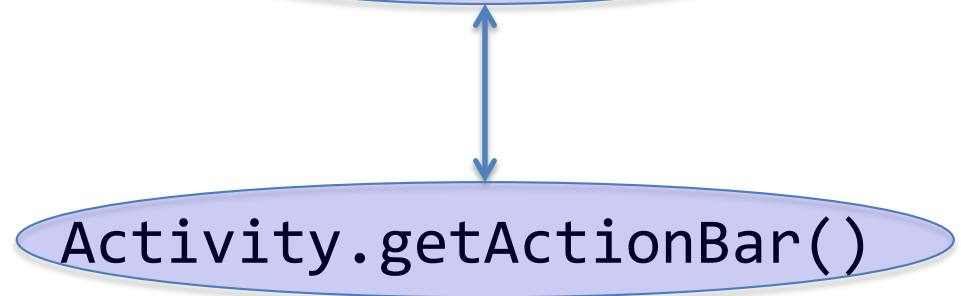
Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this);  
    ...  
}
```

The type of `mtd` is inferred to represent `Activity.getActionBar()`

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {  
    Class<?> clazz = Activity.class;  
    Method mtd = clazz.getMethod("getActionBar");  
    @Public Object actionBar = mtd.invoke(this);  
    ...  
}...
```



*Conceptual replacement

Reflection type system

Refines the Java type system

- Indicates an exact class
 - Example
 - **@ClassVal**("java.util.HashMap")
- Indicates an upper bound of a class
 - Example
 - **@ClassBound**("java.util.HashMap")

Reflection type system

Refines the Java type system

- Indicates an exact class
 - Example
 - **@ClassVal**("java.util.HashMap")
- Indicates an upper bound of a class
 - Example
 - **@ClassBound**("java.util.HashMap")

Reflection type system

Refines the Java type system

- Indicates a method
 - Example
 - **@MethodVal**("java.util.HashMap.containsKey(Object)")

Constant value analysis

- Constant folding
- Constant propagation
- Multiple values, not just one
- Evaluate side-effect-free methods
- Infer and track length of arrays
- Implemented as a type system and dataflow analysis

Constant value inference

```
void restrictFileAccess(String path) {  
    String fileUtilsClassName = "android.os.FileUtils";  
    Class<?> clazz = Class.forName(fileUtilsClassName);  
    Method mtd = clazz.getMethod("setPermissions",  
                                   String.class, int.class);  
    mtd.invoke(null, path, 0700);  
}
```

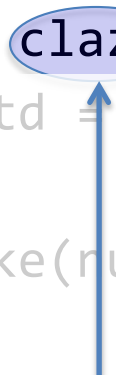
Constant value inference

```
void restrictFileAccess(String path) {  
    String fileUtilsClassName = "android.os.FileUtils";  
    Class<?> clazz = Class.forName(fileUtilsClassName);  
    Method mtd = clazz.getMethod("setPermissions",  
                                  String.class, int.class);  
    mtd.invoke(null, path, 0700);  
}
```

@StringVal("android.os.FileUtils")

Constant value inference

```
void restrictFileAccess(String path) {  
    String fileUtilsClassName = "android.os.FileUtils";  
    Class<?> clazz = Class.forName(fileUtilsClassName);  
    Method mtd = clazz.getMethod("setPermissions",  
                                String.class, int.class);  
    mtd.invoke(null, path, 0700);  
}
```

A blue oval highlights the variable 'clazz' in the code. A blue arrow points from this oval down to the '@ClassVal' annotation below the code block.

@ClassVal("android.os.FileUtils")

- Inference of **@ClassVal**
 - C.class
 - Class.forName(arg)
 - ClassLoader.loadClass(arg)

Constant value inference

```
void restrictFileAccess(String path) {  
    String fileUtilsClassName = "android.os.FileUtils";  
    Class<?> clazz = Class.forName(fileUtilsClassName);  
    Method mtd = clazz.getMethod("setPermissions",  
                                   String.class, int.class);  
    mtd.invoke(null, path, 0700);  
}
```



@MethodVal("android.os.FileUtils.setPermissions(String,int)")

- Inference of **@MethodVal**
 - `Class.getMethod(String n, Class<?> pT)`
 - `Class.getConstructor(String n, Class<?> pT)`

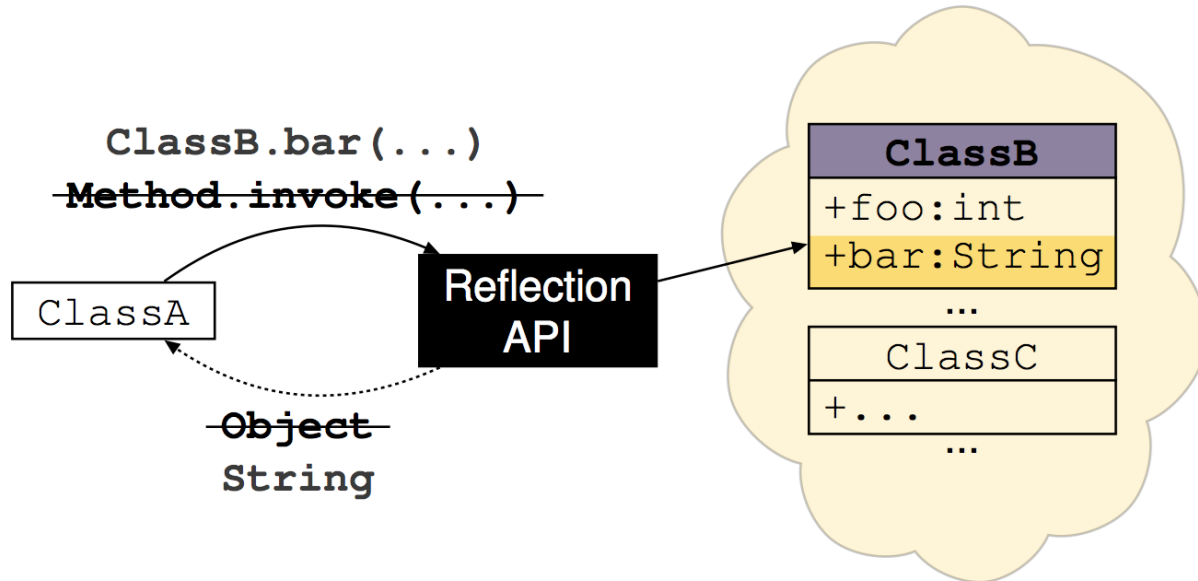
Constant value inference

```
void restrictFileAccess(String path) {  
    String fileUtilsClassName = "android.os.FileUtils";  
    Class<?> clazz = Class.forName(fileUtilsClassName);  
    Method mtd = clazz.getMethod("setPermissions",  
                                String.class, int.class);  
    mtd.invoke(null, path, 0700);  
}
```

@MethodVal("android.os.FileUtils.setPermissions(String,int)")

*Conceptual replacement

Reflection resolver



- Procedure summary is narrowed based on the Reflection type system
- Program remains unchanged
- Downstream analysis remains unchanged

Message-passing analysis (Android Intents)

Intent analysis

ComponentA

```
Intent i = buildIntent();  
i.putExtra("key",getPass());  
startActivity(i);
```

ComponentB

```
Intent i = getIntent();  
int val = i.getIntExtra("key");  
sendToEverybody(val);
```

- Intents present two challenges to static analyses:
 - Control flow
 - Component Communication Pattern (CCP)
[D. Ocateau *et al.* *USENIX '13*]
 - Data flow analysis
 - Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();  
i.putExtra("key",getPass());  
startActivity(i);
```

ComponentB

```
Intent i = getIntent();  
int val = i.getIntExtra("key");  
sendToEverybody(val)
```

- Intents present two challenges to static analyses:

Who receives
this message?

Who sent this
message?

- Control flow ←

- Component Communication Pattern (CCP)

[D. Ocateau *et al.* *USENIX* '13]

- Data flow analysis

- Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();  
i.putExtra("key",getPass());  
startActivity(i);
```

ComponentB

```
Intent i = getIntent();  
int val = i.getIntExtra("key");  
sendToEverybody(val)
```

- Intents present two challenges to static analyses:

Who receives
this message?

Who sent this
message?

- Control flow

- Component Communication Pattern (CCP)

[D. Ocateau *et al.* *USENIX '13*]

- Data flow analysis

- Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();  
i.putExtra("key",getPass());  
startActivity(i);
```

ComponentB

```
Intent i = getIntent();  
int val = i.getIntExtra("key");  
sendToEverybody(val);
```

- Intents present two challenges to static analyses:

- Control flow

- Component Communication Pattern (CCP)

[D. Ocateau *et al.* *USENIX '13*]

- Data flow analysis ←

- Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();  
i.putExtra("key", getPass());  
startActivity(i);
```

ComponentB

```
Intent i = getIntent();  
int val = i.getIntExtra("key");  
sendToEverybody(val);
```

- Intents present two challenges to static analyses:

- Control flow

- Component Communication Pattern (CCP)

[D. Ocateau *et al.* *USENIX '13*]

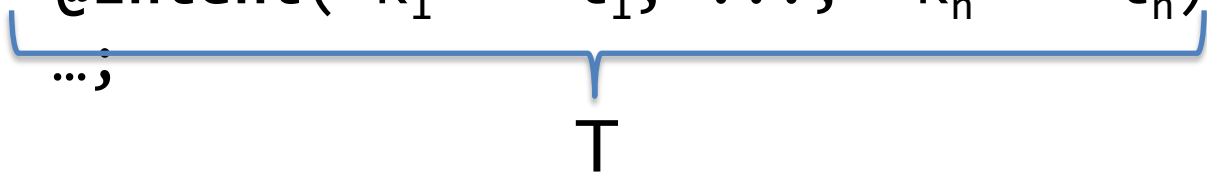
- Data flow analysis

- Intent type system

Our contribution

Intent type system

- Syntax

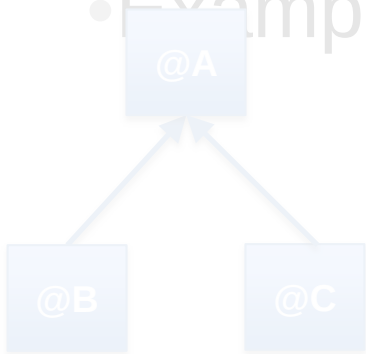
$\text{--@Intent}("K_1" \rightarrow t_1, \dots, "K_n" \rightarrow t_n) \text{ Intent } i =$


- Semantics

–(C1): Keys accessed in i must be a subset of T 's keys

–(C2): $\forall k \in \text{domain}(T) . i.\text{get}^*\text{Extra}(k) : t[k]$

- Example



$\text{--@Intent}("k" \rightarrow \text{@C}) \text{ Intent } i = \dots$

$\text{@A int } e1 = i.\text{getIntExtra}("k");$ ✓ // Legal

$i.\text{getIntExtra}("otherKey");$ ✗ // Violates (C1)

$\text{@B int } e3 = i.\text{getIntExtra}("k");$ ✗ // Violates (C2)

Intent type system

- Syntax

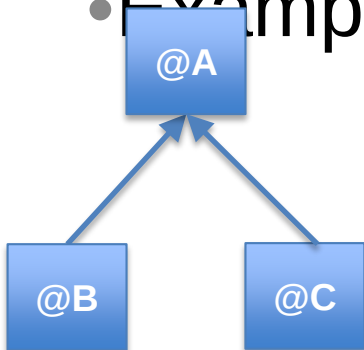
$\underbrace{\text{@Intent}("K_1" \rightarrow t_1, \dots, "K_n" \rightarrow t_n)}_{T} \text{ Intent } i = \dots;$

- Semantics

- (C1): Keys accessed in i must be a subset of T 's keys

- (C2): $\forall k \in \text{domain}(T) . i.\text{get}^*\text{Extra}(k) : t[k]$

- Example



$\text{@Intent}("k" \rightarrow \text{@C}) \text{ Intent } i = \dots$

$\text{@A int } e1 = i.\text{getIntExtra}("k");$ ✓ // Legal

$i.\text{getIntExtra}("otherKey");$ ✗ // Violates (C1)

$\text{@B int } e3 = i.\text{getIntExtra}("k");$ ✗ // Violates (C2)

Intent type system inference

```
Intent i = new Intent();  
@Secret int secret = ...;  
i.putExtra("akey", secret);  
// i now has type @Intent("akey" → @Secret)
```

i:T

- Calls *i.putExtra(key, value)* always refine the type of *i*, **except when**:
 - *i* has aliases
 - or
 - The declared type of *i* has *key* in its domain and *T[key]* is a subtype of the refined type

Revisiting example

Aarddict

```
class LookupWord extends Activity {  
    void translateWord(@Public String sentence) {  
        @Intent("sentence" → @Public)  
        Intent i = new Intent(this, WordTranslator.class);  
        i.putExtra("sentence", sentence);  
        startActivity(i);  
    } ... }
```

```
class WordTranslator extends Activity {  
    void onCreate(Bundle savedInstanceState)  
        @Intent("sentence" → @Public)  
        Intent i = getIntent();  
        @Public String sentence = i.getStringExtra("sentence");  
        ...  
    } ... }
```



Evaluation

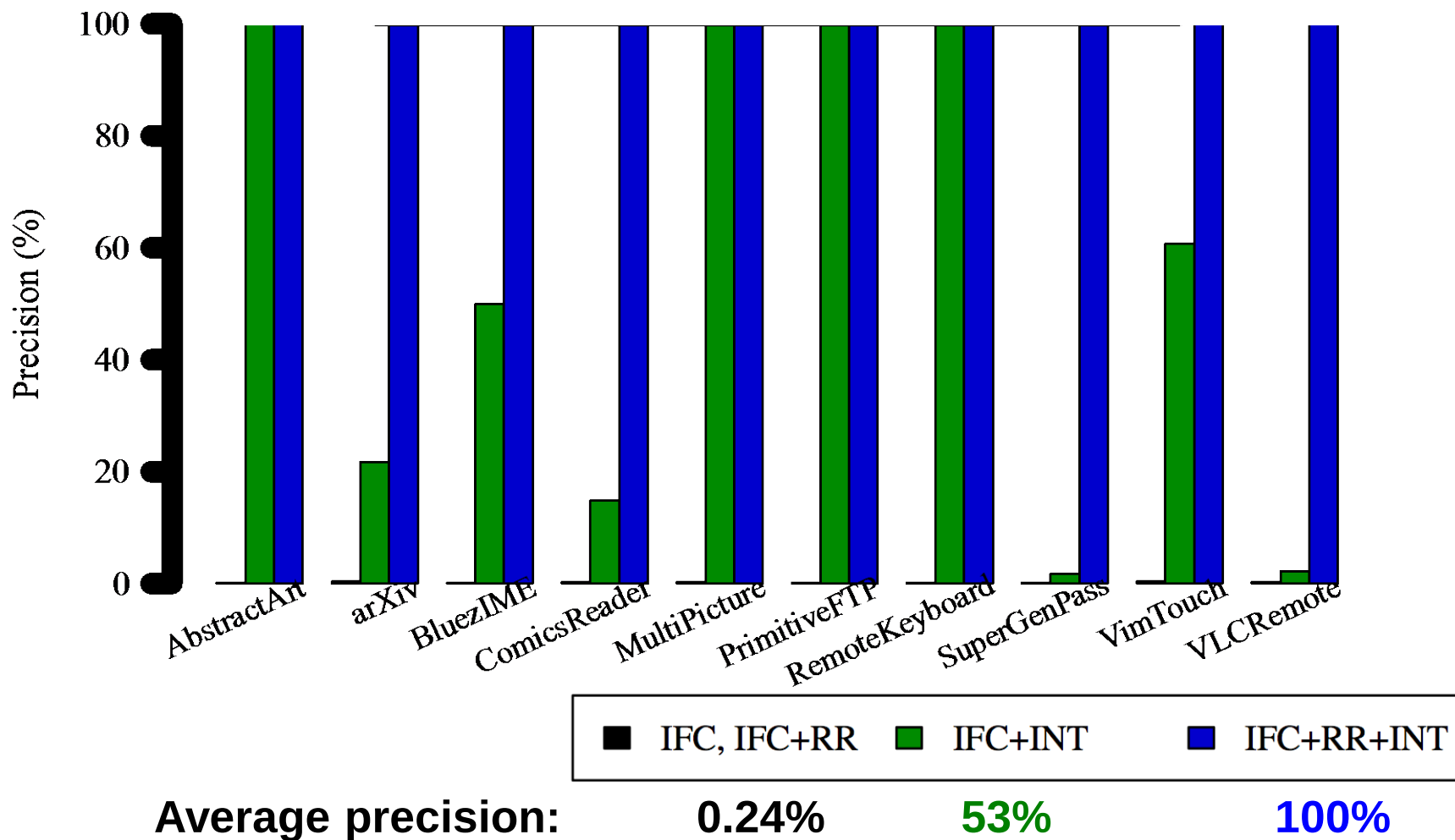
Research questions

1. How much do our reflection and intent analyses improve the precision of a downstream analysis?
2. What is the annotation overhead for programmers?

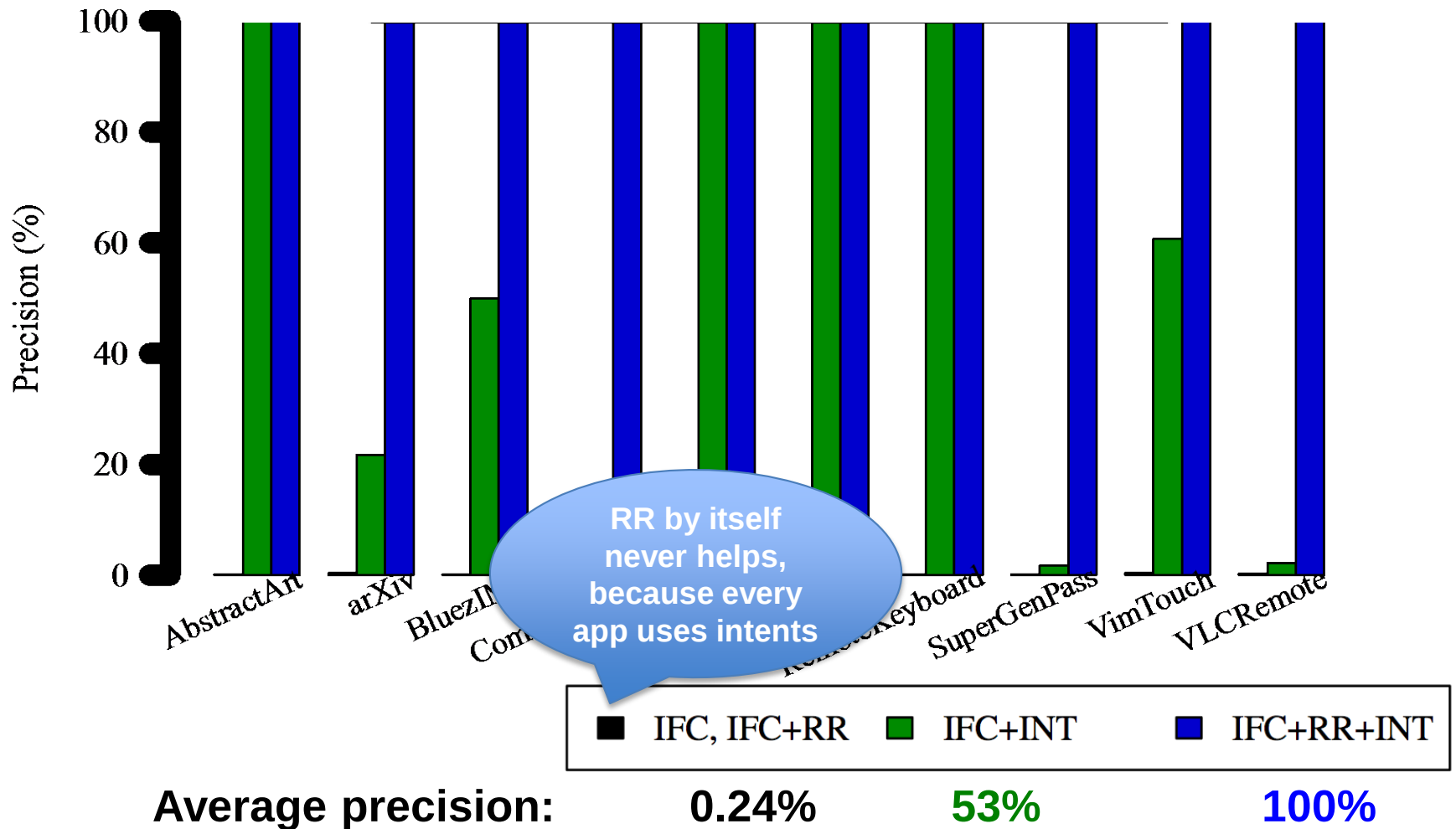
Experimental setup

- 10 F-Droid apps
 - Each contain uses of reflection and intents
 - Average complexity → 5.3K LOC
- Downstream analysis
 - Information Flow Checker (IFC)
<https://github.com/typetools/sparta>
- Metrics
 - Recall → 100%
 - Precision
 - $\# \text{ Real Flows} / \# \text{ Flows Reported}$
 - Programmer overhead → Number of annotations

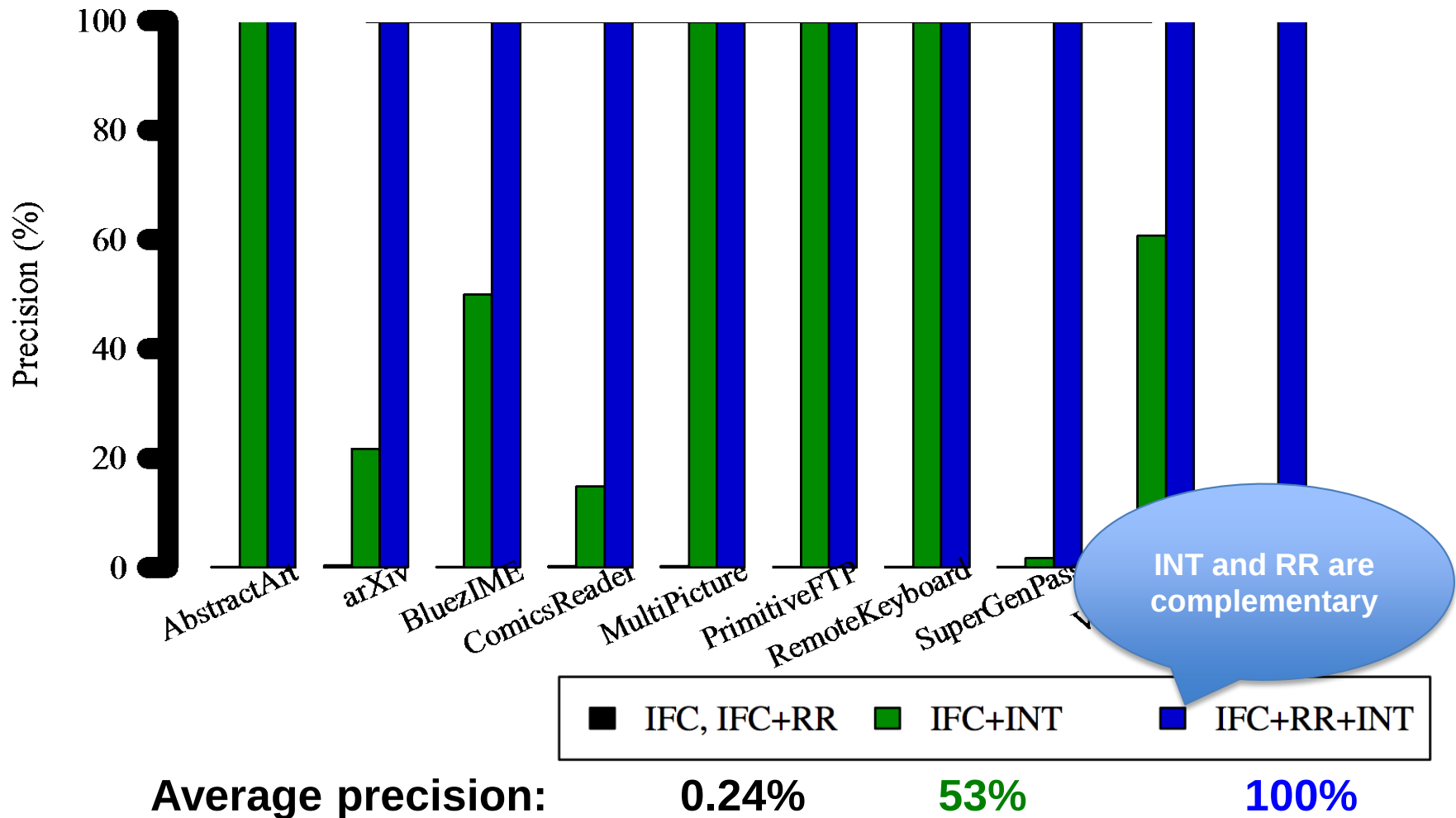
Precision



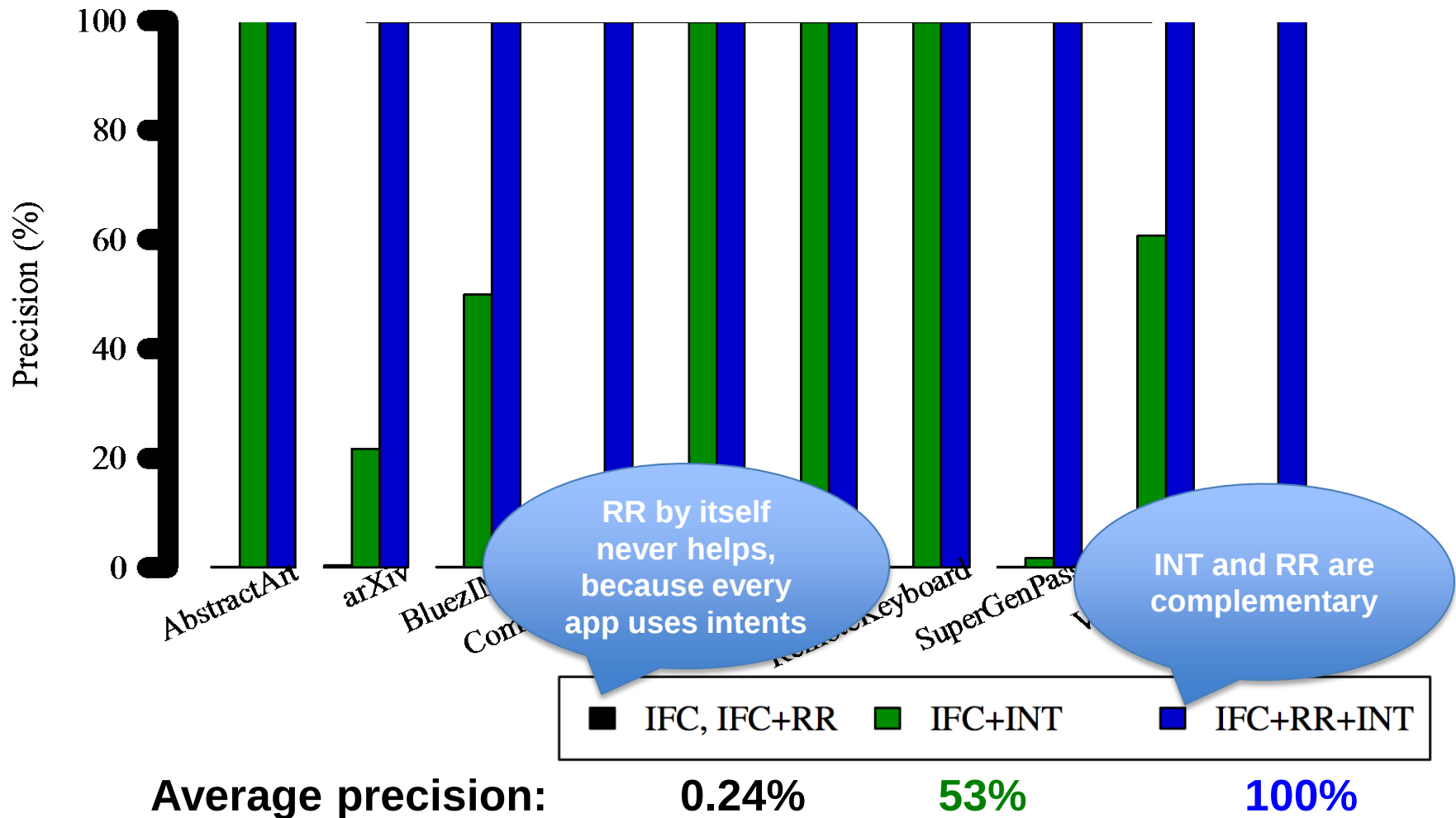
Precision



Precision



Precision



Annotation overhead

10 Android Apps	LOC	Reflection and Intent uses	# of annotations	
			IFC	REF+INT
Total	52,614	405	5,583	98

2% extra annotations

- For REF+INT → One annotation every ~2K LOC

Related work

- Reflection – sound, but limited:
 - M. S. Tschantz and M. D. Ernst, *OOPSLA'15*
 - Livshits *et al.*, *APLAS '05*
 - M. Tatsubori *et al.*, *PPL'04*
- Reflection – unsound:
 - Y. Li *et al.*, *ECOOP'14*
 - Bodden *et al.*, *ICSE'11*
- Intents – unsound:
 - L. Li *et al.*, *ICSE'15*

Conclusion

- Two sound analyses for implicit control flows
 - Reflection
 - Message-Passing
- High precision for Android apps
 - Resolved 93% of reflective calls
 - Resolved 88% of sent intents
- Can be integrated with any downstream analysis
 - Improved precision by 400x
- Implementations are available



CHECKER
framework

<http://CheckerFramework.org/>

Problem: imprecise summaries for static analyses

•...a.foo(b,c);...

Use method summary.

What does foo do?

•...myMethod.invoke(a,b,c);...

Anything!

What does invoke do?

- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe
- Goal → Soundness and high precision

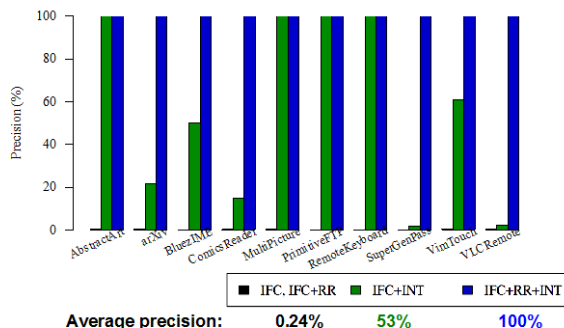
Resolving implicit control flow

- **Goal** → Soundly resolve implicit control flows
- **Observation** → Statically resolvable in F-Droid
 - 93% of reflective calls
 - 88% of sent intents
- **Solution** → We developed type systems that model implicit control flows
- **Results**
 - Improves the precision by up to 400x
 - Soundness is maintained
 - Low developer effort

Research questions

1. How much do our reflection and intent analyses improve the precision of a downstream analysis?
2. What is the annotation overhead for programmers?

Precision



Annotation overhead

10 Android Apps	LOC	Reflection and Intent uses	# of annotations	
			IFC	REF+INT
Total	52,614	405	5,583	98

2% extra annotations

- For REF+INT → One annotation every ~2K LOC

Conclusion

- Two sound analyses for implicit control flows
 - Reflection
 - Message-Passing
- High precision for Android apps
 - Resolved 93% of reflective calls
 - Resolved 88% of sent intents
- Can be integrated with any downstream analysis
 - Improved precision by 400x
- Implementations are available



CHECKER framework <http://CheckerFramework.org/>



Paulo Barros - pbsf@cin.ufpe.br

Uses of reflection in Android apps

- 35 F-droid apps were evaluated (10+25)
 - Total of 142 reflective invocations
 - 81% to provide backward compatibility
 - 6% to access non-public/hidden methods
 - 13% are for other cases (duck-typing)

Reflection type inference rules

$e : \text{String}$ val is the statically computable value of e

$e : @StringVal(val)$

$e : \text{int}$ val is the statically computable value of e

$e : @IntVal(val)$

$e : @IntVal(\pi)$

$\text{new } C[e] : @ArrayLen(\pi)$

$\text{new } C[]\{e_1, \dots, e_n\} : @ArrayLen(n)$

fqn is the fully-qualified class name of c

$C.\text{class} : @ClassVal(fqn)$

$s : @StringVal(V)$

$\text{Class.forName}(s) : @ClassVal(V)$

fqn is the fully-qualified class name of the static type of e

$e.\text{getClass}() : @ClassBound(fqn)$

$(e : @ClassBound(V) \vee e : @ClassVal(V))$

$s : @StringVal(\mu) \quad p : @ArrayLen(\pi)$

$e.\text{getMethod}(s, p) : @MethodVal(cn=V, mn=\mu, np=\pi)$

$e : @ClassVal(V) \quad p : @ArrayLen(\pi)$

$e.\text{getConstructor}(p) : @MethodVal(cn=V, mn="<init>", np=\pi)$

Intent type system rules

Subtyping

$$(ST) \quad \frac{\forall k \in \text{keys}(\tau_2). \ k \in \text{keys}(\tau_1) \wedge \tau_1[k] = \tau_2[k]}{\tau_1 <: \tau_2}$$

$$(CP) \quad \frac{\forall k \in \text{keys}(\tau_2). \ k \in \text{keys}(\tau_1) \wedge \tau_1[k] <: \tau_2[k]}{\tau_1 <_{\text{copyable}} \tau_2}$$

Well-formedness

$$(OR) \quad \frac{}{\text{void onReceive}(\tau \ i)} \quad \text{No precondition}$$

Typing judgments

$$(SI) \quad \frac{\forall \text{onReceive}(b, j). \ \langle \text{sendIntent}(a, i), \text{onReceive}(b, j) \rangle \in CCP \quad i : \tau_i \quad j : \tau_j \quad \tau_i <_{\text{copyable}} \tau_j}{\text{sendIntent}(a, i) : \text{int}}$$

$$(PE1) \quad \frac{e : \tau \quad v : \tau[k] \quad k \in \text{keys}(\tau) \quad s : @StringVal(k)}{e.\text{putExtra}(s, v) : \tau}$$

$$(PE2) \quad \frac{e : \tau \quad k \notin \text{keys}(\tau) \quad e \text{ is unaliased} \quad s : @StringVal(k)}{e.\text{putExtra}(s, v) : \tau}$$

$$(GE) \quad \frac{e : \tau \quad k \in \text{keys}(\tau) \quad s : @StringVal(k)}{e.\text{getExtra}(s) : \tau[k]}$$

Type inference rules

$$\frac{e.\text{putExtra}(s, v) \quad e : \tau \quad v : \sigma \quad k \notin \text{keys}(\tau) \quad e \text{ is unaliased} \quad s : @StringVal(k)}{e : \tau \cup \{k \rightarrow \sigma\}}$$

$$\frac{e.\text{putExtra}(s, v) \quad e : \tau \cup \{k \rightarrow _ \} \quad v : \sigma \quad e \text{ is unaliased} \quad s : @StringVal(k)}{e : \tau \cup \{k \rightarrow \sigma\}}$$

Type inference evaluation

- Inference used on reflective calls
 - 52% required intra-procedural inference
 - 41% required inter-procedural inference
 - 7% cannot be resolved by any static analysis
- Inference used on send intent calls
 - 67% required intra-procedural inference
 - 21% required inter-procedural inference
 - 12% required a better aliasing analysis

Annotation burden

- Annotations are required for two reasons
 - The downstream analysis is a modular analysis
 - Express facts that no static analysis can infer
- The average time to add an annotation was one minute