Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Carnegie Mellon University
April 23, 2014
Lab: Program analysis in the whole bug mitigation process
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis** in the **whole** bug mitigation **process**
Lab: **Program analysis in the whole bug mitigation process**

![Diagram showing the process of program analysis in the bug mitigation process.]

- **Verifier**
- **Program**
- **Thresher: Assisting Triage by Refutation Analysis**
  
  - [Blackshear+ PLDI’13, Blackshear+ SAS’11]

- **Manual Triaging**
- **Alarm Report**
  - ✔️ proof of no bug
  - ✗ Alarm
Lab: **Program analysis in the whole bug mitigation process**

![Diagram showing the process of program analysis in the bug mitigation process.](image-url)
Lab: **Program analysis in the whole bug mitigation process**

**Enforcement**
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12]

**Program-ming**

**Verifier**

**Thresher**: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11]

**Manual Triaging**

**Alarm Report**

proof of no bug
Lab: Program analysis in the whole bug mitigation process

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12]

Programming

Manual Triaging

Verifier

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11]
Lab: **Program analysis in the whole bug mitigation process**

- **Enforcement**
  - Windows: Measuring Bug Avoidance
    - [Coughlin+ ISSTA’12]

- **Programming**
  - Specification
  - Program

- **Verifier**
  - **Thresher**: Assisting Triage by Refutation Analysis
    - [Blackshear+ PLDI’13, Blackshear+ SAS’11]

- **Manual Triaging**

- **Alarm Report**
  - ✔ proof of no bug

- **Fissile Types**: Checking Almost Everywhere Invariants
  - [Coughlin+ POPL’14]
Lab: Program analysis in the whole bug mitigation process

Verifier

Proof of no bug

alarm report

Test Input

Program

Verifier

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11]

Manual Triaging

Test Output

Runner

Test

Input

Spec-
ification

Program

Alarm
Report

proof of no bug

Enforcement

Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12]

Program-
ming

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14]
Lab: **Program analysis in the whole bug mitigation process**

**Enforcement**
*Windows: Measuring Bug Avoidance* [Coughlin+ ISSTA’12]

**Programming**

**Fissile Types:**
*Checking Almost Everywhere Invariants* [Coughlin+ POPL’14]

**Manual Triaging**

**Static Incrementalization of Data Structure Checks** [under review]

**Thresher:**
*Assisting Triage by Refutation Analysis* [Blackshear+ PLDI’13, Blackshear+ SAS’11]

**Verifier**

**Runner**

**Test Input**

**Program**

**Specification**

**Test Output**

**proof of no bug**

**Alarm Report**
Lab: **Program analysis in the whole bug mitigation process**

- **Verifier**
  - Proof of no bug
  - Alarm Report
  - Thresher: Assisting Triage by Refutation Analysis [Blackshear+ PLDI’13, Blackshear+ SAS’11]
  - Fissile Types: Checking Almost Everywhere Invariants [Coughlin+ POPL’14]
  - Jsana: Abstract Domain Combinators for Dynamic Languages [Cox+ ECOOP’13]

- **Manual Triaging**

- **Runner**
  - Test Output

- **Test Input**

- **Program**

- **Specification**

- **Test Input**

- **Program**

- **Manual Triaging**

- **Enforcement Windows**: Measuring Bug Avoidance [Coughlin+ ISSTA’12]

- **Static Incrementalization of Data Structure Checks** [under review]
Lab: **Program analysis in the whole bug mitigation process**
Fissile Type Analysis:
Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Carnegie Mellon University
April 23, 2014
How to type check a program that is almost well-typed?
In this talk

Example property of interest: safety of reflective method calls

Specification system: dependent-refinement types
Reflective method call dispatches based on runtime string value

class Callback
  var sel : Str
  var obj : Obj

  def call()
    this.obj.[this.sel]()
Reflective method call dispatches based on runtime string value

class Callback
    var sel : Str
    var obj : Obj

    def call()
        this.obj.[this.sel]()
Reflective method call dispatches based on runtime string value

class Callback
  var sel : Str
  var obj : Obj

  def call()
    this.obj.[this.sel]()

Calls method with name (selector) stored in sel on object stored in obj

If sel held string “notifyDidClick” would call notifyDidClick() on obj.
Reflective method call dispatches based on runtime string value

class Callback
    var sel : Str
    var obj : Obj

def call()
    this.obj.[this.sel]()

Run time error if obj does not respond to sel — i.e., method does not exist
Method Reflection and the Great Divide
Method Reflection and the Great Divide

object.[string]()
Method Reflection and the Great Divide

reflective method call: dispatch based on run-time value (in string)

object.[string]()
Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

```object.[string]()```

“static” folks

“web 2.0” developers
Method Reflection and the Great Divide

reflective method call: dispatch based on run-time value (in string)

object.[string]()

“static” folks

“web 2.0” developers

“Static” folks, like type system designers, worry.

What gets called? What if object has no method named by string?
Method Reflection and the Great Divide

reflective method call: dispatch based on \textit{run-time value} (in string)

\texttt{object.[string]()}

“static” folks

“web 2.0” developers

“Static” folks, like type system designers, \textit{worry}.

What gets called? What if \texttt{object} has \texttt{no method} named by \texttt{string}?

“Web 2.0” developers think it’s \textit{cool}.

I can write flexible and compact code, so I will take it over \texttt{static} safety.
Method Reflection and the Great Divide

Reflective method call: dispatch based on run-time value (in string)

object. [string] ()

“static” folks

“web 2.0” developers

Bridge the divide to support both first-class reflective method call and static checking of reflection safety

“Static” folks, like type system designers, worry. What gets called? What if object has no method named by string?

“Web 2.0” developers think it’s cool. I can write flexible and compact code, so I will take it over static safety.

Bridge the divide to support both first-class reflective method call and static checking of reflection safety
class Callback
    var sel : Str
    var obj : Obj

    def call()
        this.obj.[this.sel]()
Ensure reflection safety with **dependent-refinement type** expressing required relationship

```python
class Callback
    var sel : Str
    var obj : Obj `r2 sel`

def call()
    this.obj.[this.sel]()
```

*obj must “respond to” sel*
Ensure reflection safety with dependent-refinement type expressing required relationship

class Callback
  var sel : Str
  var obj : Obj | r2 sel

  def call()
    this.obj.[this.sel]()
Ensure reflection safety with dependent-refinement type expressing required relationship

class Callback
    var sel : Str
    var obj : Obj | r2 sel

def call()
    this.obj.[this.sel]()
Similar relationship for array bounds safety

class Iterator
var idx : Int
var buf : Obj[] | indexedBy idx

def get(): Obj
  return this.buf[this.idx]
Similar relationship for array bounds safety

class Iterator
var idx : Int
var buf : Obj[] | indexedBy idx

def get(): Obj
  return this.buf[this.idx]
Similar relationship for array bounds safety

class Iterator
    var idx : Int
    var buf : Obj[]

    def get(): Obj
        return this.buf[this.idx]

idx must be a valid index into buf

Guarantees no “ArrayOutOfBounds” error
Similar relationship for array bounds safety

```java
class Iterator
    var idx : Int
    var buf : Obj[] | indexedBy idx

    def get(): Obj
        return this.buf[this.idx]
```

These kinds of relationships are important to many safety properties
Updating relationship causes type error

class Callback
  var sel : Str
  var obj : Obj | r2 sel

def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o

def call()
  this.obj.[this.sel]()
class Callback
    var sel : Str
    var obj : Obj | r2 sel

def update(s : Str, o : Obj | r2 s)
    this.sel = s
    this.obj = o

def call()
    this.obj.[this.sel]()
class Callback
var sel : Str
var obj : Obj | r2 sel

def update(s : Str, o : Obj | r2 s)
    this.sel = s
    this.obj = o

def call()
    this.obj.[this.sel]()
Upd**at**ing relationship causes type error

```python
class Callback
    var sel : Str
    var obj : Obj | r2 sel

def update(s : Str, o : Obj | r2 s)
    this.sel = s
    this.obj = o

def call()
    this.obj.[this.sel]()```

Field type says: obj must always respond to sel
guaranteed to respond to s

Type error: old obj may not respond to new sel
**Updating relationship causes type error**

```python
class Callback:
    var sel : Str
    var obj : Obj | r2 sel

def update(s : Str, o : Obj | r2 s):
    this.sel = s
    this.obj = o

def call():
    this.obj.[this.sel]()
```

- Field type says: `obj` must always respond to `sel`
- `o` guaranteed to respond to `s`
- Type error: old `obj` may not respond to new `sel`
- False alarm: no runtime error
Two styles of reasoning to determine false alarm

class Callback
    var sel : Str
    var obj : Obj | r2 sel

    def update(s : Str, o : Obj | r2 s )
       this.sel = s
       this.obj = o

    def call()
       this.obj.[this.sel]()
Two styles of reasoning to determine false alarm

class Callback
    var sel : Str
    var obj : Obj | r2 sel

    def update(s : Str, o : Obj | r2 s )
        this.sel = s
        this.obj = o

    def call()
        this.obj.[this.sel]()
Two styles of reasoning to determine false alarm

class Callback
    var sel : Str
    var obj : Obj | r2 sel

    def update(s : Str, o : Obj | r2 s)
        this.sel = s
        this.obj = o

    def call()
        this.obj.[this.sel]()

Reasoning by global invariant: call safe if relationship holds
Two styles of **reasoning** to determine false alarm

class Callback
  var sel : Str
  var obj : Obj

def update(s : Str, o : Obj)
  this.sel = s
  this.obj = o

def call()
  this.obj.[this.sel]()
Two styles of reasoning to determine false alarm

```python
class Callback:
    var sel : Str
    var obj : Obj

    def update(s : Str, o : Obj):
        this.sel = s
        this.obj = o

    def call():
        this.obj.[this.sel]()  
```

Reasoning by global invariant: call safe if relationship holds

Reasoning about effects of imperative updates

Relationship violated
Two styles of reasoning to determine false alarm

class Callback
    var sel : Str
    var obj : Obj

def update(s : Str, o : Obj):
    this.sel = s
    this.obj = o

def call():
    this.obj.[this.sel]()
Idea: Selectively alternate between reasoning styles in verification
Fissile Type Analysis combines two styles of reasoning
Fissile Type Analysis combines two styles of reasoning.

Automated reasoning about global invariants.
Fissile Type Analysis combines two styles of reasoning

Automated reasoning about global invariants

$\Gamma \vdash \cdots$

Flow-Insensitive Type Systems
Fissile Type Analysis **combines** two styles of reasoning

Automated **reasoning** about **global invariants**

Automated **reasoning** about **execution**
Fissile Type Analysis combines two styles of reasoning:

- Automated reasoning about global invariants
- Automated reasoning about execution

\[
\Gamma \vdash \cdots \\
\text{Flow-Insensitive Type Systems}
\]

\[
\gamma(\cdot) = \cdots \\
\text{Abstract Interpretation/Flow Analysis}
\]
Verification of almost-everywhere invariants with intertwined type and flow analysis
Verification of almost-everywhere invariants with intertwined type and flow analysis

Switch to symbolic analysis when global type invariant violated

type analysis

types violated
Verification of almost-everywhere invariants with intertwined type and flow analysis

- Type analysis
- Symbolic flow analysis
- Types violated
  - Switch to symbolic analysis when global type invariant violated
- Types restored
  - Back to types when invariant restored
Verification of almost–everywhere invariants with intertwined type and flow analysis

Switch to symbolic analysis when global type invariant violated

Back to types when invariant restored
Verification of almost–everywhere invariants with intertwined type and flow analysis

Switch to symbolic analysis when global type invariant violated

Back to types when invariant restored
Verification of almost--everywhere invariants with intertwined type and flow analysis

Switch to symbolic analysis when global type invariant violated

Back to types when invariant restored
Verification of *almost–everywhere invariants* with *intertwined* type and flow analysis

- Type analysis
- Symbolic flow analysis
- Type analysis
- Symbolic flow analysis
- Type analysis

**Types violated**: Switch to symbolic analysis when global type invariant violated

**Types restored**: Back to types when invariant restored

**Not changing type analysis at all**: just when applied
Verification of almost-everywhere invariants with intertwined type and flow analysis

Effective when global type invariant holds most of the time
Verification of almost–everywhere invariants with intertwined type and flow analysis

Effective when global type invariant holds most of the time

Relationship updates
Verification of almost-everywhere invariants with intertwined type and flow analysis

Effective when global type invariant holds most of the time

- Relationship updates
- Occurrence typing
Verification of almost-everywhere invariants with intertwined type and flow analysis

Type analysis

Types violated

Symbolic flow analysis

Types restored

Type analysis

Types violated

Symbolic flow analysis

Types restored

Effective when global type invariant holds most of the time

- Relationship updates
- Occurrence typing
- Tagged unions
Play to the strengths of each intertwined analysis
Play to the **strengths** of each intertwined analysis

**Flow-Insensitive Types**
- Easy to specify **global** invariants
- **Fast**
- Natural for **modular** reasoning
- **Good** error reporting
Play to the strengths of each intertwined analysis

**Flow-Insensitive Types**
- Easy to specify global invariants
- Fast
- Natural for modular reasoning
- Good error reporting

**Symbolic Flow Analysis**
- Natural for local reasoning about heap mutation
- Precise
- Can be disjunctive/path-sensitive
Play to the strengths of each intertwined analysis

**Flow-Insensitive Types**
- Easy to specify global invariants
- Fast
- Natural for modular reasoning
- Good error reporting

**Symbolic Flow Analysis**
- Natural for local reasoning about heap mutation
- Precise
- Can be disjunctive/path-sensitive

flow-sensitive typing?
ownership types?
alias types?
permissions?
effects?
Play to the strengths of each intertwined analysis

**Flow-Insensitive Types**
- Easy to specify global invariants
- Fast
- Natural for modular reasoning
- Good error reporting

**Symbolic Flow Analysis**
- Natural for local reasoning
- about heap mutation
- Precise
- Can be disjunctive/path-sensitive

Goal: keep types as simple as possible
Play to the **strengths** of each intertwined analysis

**Flow-Insensitive Types**
- Easy to **specify global** invariants
- Fast
- Natural for **modular** reasoning
- Good **error reporting**

**Symbolic Flow Analysis**
- Natural for **local** reasoning about **heap mutation**
- Precise
- Can be disjunctive/path-sensitive

---

**Goal:** keep **types as simple as possible**

**Complexity** lies in **handoff** between analyses and in **symbolic analysis**

- flow-sensitive typing?
- ownership types?
- alias types?
- permissions?
- effects?
Key Contributions

1. **Translate** type invariant into symbolic state via "symbolization" of type environment.

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization.
Key Contributions

1. Translate type invariant into symbolic state via "symbolization" of type.
   Reason precisely only when type invariant violated.

2. Leverage heap type invariant during symbolic analysis via type-consistent materialization and summarization.
Key Contributions

1. **Translate** type invariant into symbolic state via "symbolization" of type
   - Reason precisely only when type invariant violated

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization
   - Reason precisely only for locations where type invariant violated
Key Contributions

1. **Translate** type invariant into symbolic state via “symbolization” of type
   
   Reason precisely only when type invariant violated

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization
   
   Reason precisely only for locations where type invariant violated
Symbolization splits a type environment into facts about values and storage for those values
Symbolization splits a type environment into facts about values and storage for those values.

def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
Symbolization splits a type environment into facts about values and storage for those values.

```python
def update(s: Str, o: Obj | r2 s)
    this.sel = s  # X
    this.obj = o
```
Symbolization splits a type environment into facts about values and storage for those values.

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Symbolization splits a type environment into facts about values and storage for those values

Type environment

Maps local variables to dependent types

Γ

\[ s: \text{Str} \]
\[ o: \text{Obj} \mid r2 \ s \]
\[ \text{this}: \text{Callback} \]

Refinements refer to variables

```
def update(s:Str, o:Obj | r2 s):
    this.sel = s
    this.obj = o
```
Symbolization splits a type environment into facts about values and storage for those values.

**Type environment**

Maps local variables to dependent types:

\[
\Gamma \\
\begin{array}{l}
s : \text{Str} \\
o : \text{Obj} \mid r2 \ s \\
\text{this} : \text{Callback}
\end{array}
\]

**Symbolic state**

\[
\mathcal{E} \\
\begin{array}{l}
s : \to{s} \\
o : \to{o} \\
\text{this} : \to{t}
\end{array}
\]

\[
\mathcal{F} \\
\begin{array}{l}
\to{s} : \text{Str} \\
\to{o} : \text{Obj} \mid r2 \ \to{s} \\
\to{t} : \text{Callback}
\end{array}
\]

**Symbolize**
Symbolization splits a type environment into facts about values and storage for those values

Type environment
Maps local variables to dependent types
\[
\Gamma \\
\begin{align*}
s &: \text{Str} \\
o &: \text{Obj} | r2 \ s \\
\text{this} &: \text{Callback}
\end{align*}
\]

Symbolic state
Maps local variables to symbolic values
\[
\tilde{\Gamma} \\
\begin{align*}
s &: \tilde{s} \\
o &: \tilde{o} \\
\text{this} &: \tilde{t} \\
\tilde{s} &: \text{Str} \\
\tilde{o} &: \text{Obj} | r2 \ \tilde{s} \\
\tilde{t} &: \text{Callback}
\end{align*}
\]

Refinements refer to variables

def update(s:Str, o:Obj | r2 s)
    this.sel = s
    this.obj = o
Symbolization splits a type environment into facts about values and storage for those values.

Type environment

Maps local variables to dependent types

Maps symbolic values to dependent types

Maps local variables to symbolic values

Maps symbolic values to symbolic values (symbolic facts)

Refinements refer to variables

Symbolic state

symbolize

def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
Symbolization splits a type environment into facts about values and storage for those values.

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

**Type environment**
Maps local variables to dependent types

\[
\Gamma
s : Str
o : Obj | r2 s
this : Callback
\]

**Symbolic state**
Maps local variables to symbolic values

\[
\tilde{E}
\tilde{s} : \tilde{\tilde{s}}
\tilde{o} : \tilde{\tilde{o}}
this : \tilde{\tilde{t}}
\]

**Refinements**
Refer to values

Maps symbolic values to dependent types lifted to symbolic values (symbolic facts)

```
\tilde{\tilde{s}} : Str
\tilde{\tilde{o}} : Obj | r2 \tilde{\tilde{s}}
\tilde{\tilde{t}} : Callback
```
Symbolization allows local variables to hold values inconsistent with declared types

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Γ

\[
\begin{align*}
\Gamma & \quad \text{s : Str} \\
\Gamma & \quad \text{o : Obj | r2 s} \\
\Gamma & \quad \text{this : Callback}
\end{align*}
\]
Symbolization allows local variables to hold values inconsistent with declared types

A type environment constrains local variables

\[ \Gamma \\
\begin{array}{l}
s : \text{Str} \\
o : \text{Obj} \mid r2\ s \\
this : \text{Callback}
\end{array} \]

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

![Diagram showing types and variables in a program flow](image)
Symbolization allows local variables to hold values inconsistent with declared types.

A type environment constrains local variables:

\[ \Gamma \]
- \( s : \text{Str} \)
- \( o : \text{Obj} | r2 s \)
- \( \text{this} : \text{Callback} \)

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

But also constrains the reachable heap to be **type-consistent**: fields must conform to declared types.
Symbolization allows local variables to hold values inconsistent with declared types

A type environment constrains local variables

\[ \Gamma \]

\[ s : \text{Str} \]
\[ o : \text{Obj | r2 s} \]
\[ \text{this} : \text{Callback} \]

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

But also constrains the reachable heap to be type-consistent: fields must conform to declared types

This picture captures the fully type-consistent concrete state
Symbolization allows local variables to hold values inconsistent with declared types

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

\[
\Gamma \quad s : \text{Str} \\
o : \text{Obj} | r2 s \\
this : \text{Callback}
\]

\[
\text{symbolize} \quad \overset{\sim}{E} \quad \overset{\sim}{\Gamma}
\]

\[
\begin{align*}
\text{heap} & \\
this.obj & \\
this.sel & \\
s & \\
o & \\
this
\end{align*}
\]
Symbolization allows local variables to hold values inconsistent with declared types.

Symbolic environment allows, e.g., int in s

Symbolic environment

\[ \Gamma \]

\[ s : \text{Str} \]
\[ o : \text{Obj} \mid r2 \ s \]
\[ \text{this} : \text{Callback} \]

symbolize

\[ \tilde{E} \rightarrow \tilde{\Gamma} \]

heap

\[ \text{this.obj} \]
\[ \text{this.sel} \]

def update(s:Str, o:Obj | r2 s):
    this.sel = s
    this.obj = o
Symbolization allows local variables to hold values inconsistent with declared types.

Symbolic environment allows, e.g., int in s

\[ \Gamma \]

\[ s : \text{Str} \]
\[ o : \text{Obj} | \text{r2 s} \]
\[ \text{this} : \text{Callback} \]

\[ \text{def update}(s:\text{Str}, o:\text{Obj} | \text{r2 s}) \]
\[ \text{this}.\text{sel} = s \]
\[ \text{this}.\text{obj} = o \]

Immediately type-inconsistent: value stored without dereferences violates a type constraint
Symbolization allows local variables to hold values inconsistent with declared types

Symbolic environment allows, e.g., int in s

Immediately type-inconsistent: value stored without dereferences violates a type constraint

$$\Gamma$$

$$s : \text{Str}$$
$$o : \text{Obj} | \text{r2 s}$$
$$\text{this} : \text{Callback}$$

$$\text{symbolize}$$

$$\widetilde{E}$$

$$\widetilde{\Gamma}$$

def update(s:Str, o:Obj | r2 s)
this.sel = s
this.obj = o

heap

this.obj
this.sel
Symbolization allows local variables to hold values inconsistent with declared types

Symbolic environment allows, e.g., int in s

```
def update(s:Str, o:Obj | r2 s)
    this.sel = s
    this.obj = o
```

Immediately type-inconsistent: value stored without dereferences violates a type constraint
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

Type environment

Symbolic fact map

\[
\Gamma \\
\begin{array}{l}
s : \text{Str} \\
o : \text{Obj} | r2 \ s \\
\text{this} : \text{Callback}
\end{array}
\]

\[
\tilde{\Gamma} \\
\begin{array}{l}
\tilde{s} : \text{Str} \\
\tilde{o} : \text{Obj} | r2 \ \tilde{s} \\
\tilde{t} : \text{Callback}
\end{array}
\]

\[
\text{symbolize}
\]

\[
\text{heap}
\]

\[
\text{this.obj} \quad \text{this.sel}
\]

\[
\begin{array}{l}
\text{s} \\
\text{o} \\
\text{this}
\end{array}
\]

\[
\begin{array}{l}
\tilde{\text{s}} \\
\tilde{\text{o}} \\
\tilde{\text{this}}
\end{array}
\]
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

Type environment

Base types same on both sides

Symbolic fact map

$\Gamma \vdash \text{Callback} : \text{Str}$

$\Gamma \vdash \text{this} : \text{Obj}$

$\Gamma \vdash \text{Callback} : \text{Obj}$

$\tilde{\Gamma} \vdash \tilde{s} : \text{Str}$

$\tilde{\Gamma} \vdash \tilde{\text{this}} : \text{Obj}$

$\tilde{\Gamma} \vdash \tilde{\text{Callback}} : \text{Obj}$

Heap

this.obj

this.sel
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

Type environment

\[
\begin{align*}
\Gamma & \vdash \text{Callback} \\
& \triangleq \{ \text{sel} : \text{Str}, \text{obj} : \text{Obj} \mid r2 \, \text{sel} \}
\end{align*}
\]

Base types same on both sides

\[
\begin{align*}
\tilde{\Gamma} & \vdash \text{Callback} \\
& \triangleq \{ \tilde{s} : \text{Str}, \tilde{o} : \text{Obj} \mid r2 \, \tilde{s} \}
\end{align*}
\]

Symbolic fact map

Base type field refinements still refer to fields

\[
\begin{align*}
\text{this.obj} & \rightarrow \text{this.sel} \\
\text{this} & \rightarrow \text{Callback}
\end{align*}
\]
Summarize heap locations that are not immediately type-inconsistent with `okheap`

**Symbolic Heap**

\[ \sim \mathcal{H} \]

**Concrete State**

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

- `s`: Str
- `o`: Obj
- `this.sel`: selected object
- `this.obj`: object being modified
Summarize heap locations that are **not immediately type-inconsistent** with okheap.

**Symbolic Heap**

\[ \tilde{H} \]

okheap

**Concrete State**

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Formula literal: **concretization** includes every subheap that is **not immediately type inconsistent**.
Summarize heap locations that are not immediately type-inconsistent with okheap.

Symbolic Heap

\[ \tilde{H} \]

okheap

Describes storage without explicitly enumerating it.

Concrete State

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Formula literal: concretization includes every subheap that is not immediately type inconsistent.
Summarize heap locations that are **not** immediately type-inconsistent with okheap.

**Symbolic Heap**

\[ \tilde{H} \]

okheap

Describes storage without explicitly enumerating it.

**Concrete State**

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Formula literal: **concretization** includes every subheap that is **not** immediately type inconsistent.

Immediately after switch, type invariants still hold so okheap represents **entire heap**.
Key Contributions

1. Translate type invariant into symbolic state via "symbolization" of type environment.

2. Leverage heap type invariant during symbolic analysis via type-consistent materialization and summarization.
Key Contributions

1. **Translate** type invariant into symbolic state via “symbolization” of type environment.

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization.
Leverage heap type invariant via type-consistent materialization

Symbolic State

\[ \tilde{H} \]

Concrete State

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \widetilde{H} \quad \text{okheap} \ast \widetilde{\text{this}} \mapsto \{ \text{sel} \mapsto \text{sel} \ast \text{obj} \mapsto \text{obj} \} \]

def update(s:Str, o:Obj | r2 s)
    this.sel = s
    this.obj = o

Concrete State
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

$$\widetilde{H} \quad \text{okheap} \ast \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} \ast \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Must-alias and disalias guarantee requires case split on materialization

Concrete State

```python
def update(s: Str, o: Obj | r2 s)
this.sel = s
this.obj = o
```
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \widetilde{H} \quad \text{okheap} \land \widetilde{this} \mapsto \{ \text{sel} \mapsto \text{sel} \ast \text{obj} \mapsto \text{obj} \} \]

Must-alias and disalias guarantee requires case split on materialization

Concrete State

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
    if sel & obj:
        this.sel = s
        this.obj = o
```
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\( \tilde{H} \)

\( \text{okheap} \ast \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{\text{sel}} \ast \text{obj} \mapsto \tilde{\text{obj}} \} \)

Must-alias and disalias guarantee requires case split on materialization

Value stored in \( \text{obj} \) responds to value stored in \( \text{sel} \)

Concrete State

```
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
    if sel ⇤ obj
        this.obj = this.sel ⇤ obj
    return this
```

Materialized storage guaranteed to be not immediately type-inconsistent
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \tilde{H} = \text{okheap} \times \{ \text{this} \mapsto \{ \text{sel} \mapsto \text{sel}, \text{obj} \mapsto \text{obj} \} \} \]

Must-alias and disalias guarantee requires case split on materialization

Value stored in obj responds to value stored in sel

Concrete State

Represent materialized storage with

\[ \text{this.obj} \quad \text{this.sel} \]
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \tilde{H} \]

okheap * \( \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{\text{sel}} \ast \text{obj} \mapsto \tilde{\text{obj}} \} \)

Must-alias and disalias guarantee requires case split on materialization

Value stored in \( \text{obj} \) responds to value stored in \( \text{sel} \)

Concrete State

Analysis can assume that type invariant initially holds on all materialized storage
Strong updates on materialized storage to detect invariant restoration

Concrete State

Symbolic State

\[ \hat{H} \]

\[ \text{okheap} \ast \overline{\text{this}} \mapsto \{ \text{sel} \mapsto \overline{\text{sel}} \ast \text{obj} \mapsto \overline{\text{obj}} \} \]
Strong updates on materialized storage to detect invariant restoration

def update(s:Str, o:Obj | r2 s)
    this.sel = s
    this.obj = o

Concrete State

Symbolic State

\[ \tilde{H} \]

\[ \text{okheap} \ast \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{s} \ast \text{obj} \mapsto \tilde{\text{obj}} \} \]
Strong updates on materialized storage to detect invariant restoration

Symbolic State

\[ \tilde{H} \]

\[
\text{okheap } \ast \quad \tilde{\text{this}} \mapsto \{\text{sel }\mapsto \tilde{s} \ast \text{obj }\mapsto \tilde{\text{obj}}\}
\]

Concrete State

def update(s:Str, o:Obj | r2 s)
    this.sel = s
    this.obj = o

Type invariant violated
Strong updates on materialized storage to detect invariant restoration

Definition:

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Symbolic State

\[ H \]

\[ \text{okheap} \ast \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{s} \ast \text{obj} \mapsto \tilde{\text{obj}} \} \]

Surprising: can soundly permit pointers in and out of the region that is not immediately type-inconsistent

Type invariant violated
Strong updates on materialized storage to detect invariant restoration

Symbolic State

\[ \text{okheap} \ast \tilde{\text{this}} \leftrightarrow \{ \text{sel} \mapsto \tilde{s} \ast \text{obj} \mapsto \tilde{o} \} \]

Concrete State

\[
\text{def update}(s:\text{Str}, o:\text{Obj} | r2 s)
\quad \text{this}.\text{sel} = s
\quad \text{this}.\text{obj} = o
\]

\[
\tilde{\text{this}} = \{ \text{this}.\text{sel} \mapsto \tilde{s}, \text{this}.\text{obj} \mapsto \tilde{o} \}
\]
**Strong updates** on materialized storage to detect invariant restoration

Symbolic State

$\widehat{H}$

$\text{okheap} \ast \widehat{\text{this}} \mapsto \{ \text{sel} \mapsto \widehat{s} \ast \text{obj} \mapsto \widehat{o} \}$

Concrete State

No longer immediately type-inconsistent

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Safely **summarize** storage that is not immediately type inconsistent

**Symbolic State**

\[ \overset{\sim}{H} \]

\[ \text{okheap} \star \overset{\sim}{\text{this}} \mapsto \{ \text{sel} \mapsto \overset{\sim}{s} \star \text{obj} \mapsto \overset{\sim}{o} \} \]

**Concrete State**

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Safely summarize storage that is not immediately type inconsistent

Symbolic State

\[ \tilde{H} \]

okheap

Concrete State

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Safely summarize storage that is not immediately type inconsistent.

Symbolic State

\[ \tilde{H} \]

okheap

Concrete State

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Only need to reason precisely about part of heap where invariant broken, so helps manage alias explosion.
Safely **summarize** storage that is not immediately type inconsistent

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

**Symbolic State**

$\tilde{H}$ okheap

Entire heap is type consistent so safe to return to type checking

Only need to reason precisely about part of heap where invariant broken, so helps manage alias explosion
Safely summarize storage that is not immediately type inconsistent

Symbolic State

Entire heap is type consistent so safe to return to type checking

Only need to reason precisely about part of heap where invariant broken, so helps manage alias explosion

def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
Key Contributions

1. **Translate** type invariant into symbolic state via "symbolization" of type environment.

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization.
Key Contributions

1. **Translate** type invariant into symbolic state via "symbolization" of type environment

2. **Leverage** heap type invariant during symbolic analysis via type-consistent materialization and summarization
Fissile Type Analysis is sound
Fissile Type Analysis is **sound**

**Theorem** (Soundness of Handoff).

The entire state is **type-consistent** iff all locations are **not immediately type-inconsistent**.
Fissile Type Analysis is sound

**Theorem** (Soundness of Handoff).

The entire state is type-consistent iff all locations are not immediately type-inconsistent.

**Theorem** (Soundness of Materialization/Summarization).

Storage that is not immediately type-inconsistent can be safely materialized and summarized into okheap.
Evaluation

Analysis mechanics: How often is symbolic reasoning required?

Precision: What is improvement over flow-insensitive checking alone?

Cost: What is the cost of analysis in running time?
Case Study: Reflection in Objective-C

Prototype analysis implementation
  Plugin for `clang` static analyzer in C++

9 Objective-C benchmarks
  6 libraries and 3 applications
  1,000 to 176,000 lines of code

Manual type annotations
  76 r2 annotations on system libraries
  136 annotations on benchmark code
Case Study: Reflection in Objective-C

Prototype analysis implementation
Plugin for *clang* static analyzer in C++

9 Objective-C benchmarks
6 libraries and 3 applications
1,000 to 176,000 lines of code

Manual *type annotations*
76 r2 annotations on *system libraries*
136 annotations on *benchmark code*

Including *Skim*, *Adium*, and *OmniGraffle*
## Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>SCRecord</td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>125</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>
## Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OMNIFRAMEWORKS</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>125</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

Number of **successful switches to symbolic analysis and back**
## Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OMNIFRAMEWORKS</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

A **significant** number of switches: Approach successfully handles when developers **break** and **restore** global invariants.
### Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OmniFRAMESWORKS</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

**Maximum number of simultaneous materialized storage locations**

A **significant** number of **switches**: Approach successfully handles when **developers break and restore global invariants**
### Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

**A significant number of switches:**
Approach successfully handles when developers break and restore global invariants

At most 2 simultaneous materializations:
Aliasing case splits will not blow up
### Analysis mechanics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>125</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

A **significant** number of **switches**: Approach successfully handles when **developers break and restore** global invariants

At most 2 **simultaneous materializations**: Aliasing case splits will **not blow up**
## Analysis mechanics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size (loc)</th>
<th>Symbolic Sections</th>
<th>Maximum Materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>125</strong></td>
<td></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

Approaches limited to one-at-a-time materialization not sufficient

A significant number of switches:
Approach successfully handles when developers break and restore global invariants

At most 2 simultaneous materializations:
Aliasing case splits will not blow up
## Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>false alarms</th>
<th>flow-insensitive</th>
<th>almost-everywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
<td></td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
<td></td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
<td></td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
<td></td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
<td></td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
<td></td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
<td></td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
<td></td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
<td></td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- '-' indicates no data available for that column.
**Precision**

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>false alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>false alarms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>flow-insensitive</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>almost-everywhere</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRECORDER</td>
<td>2716</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 (-)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>68</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OMNIFRAMEWORKS</td>
<td>160769</td>
<td>192</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>238 (-29%)</td>
</tr>
</tbody>
</table>

**Baseline**: standard, flow-insensitive type analysis – no switching
### Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>false alarms</th>
<th>false alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OMNIFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
</tr>
</tbody>
</table>

**Baseline:** standard, flow–insensitive type analysis – no switching
## Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>false alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>flow-insensitive</td>
<td>almost-everywhere</td>
</tr>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRECODER</td>
<td>2716</td>
<td>12</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0 (-)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>68</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>238 (-29%)</strong></td>
</tr>
</tbody>
</table>

**Baseline:** standard, flow-insensitive type analysis – no switching

*Almost everywhere* techniques show 29% improvement in false alarms.
### Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>false alarms</th>
<th>almost-everywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td></td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td></td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td></td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td></td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td></td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (-29%)</td>
</tr>
</tbody>
</table>

**Baseline:** standard, flow-insensitive type analysis – no switching.

**Almost everywhere** techniques show 29% improvement in false alarms.

Also found a real reflection **bug** in Vienna, which we reported and which was **fixed**.
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24s</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10s</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>0.67s</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>2.79s</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>8.79s</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
</tr>
</tbody>
</table>
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>SCRIM</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td></td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23.0</strong></td>
</tr>
</tbody>
</table>

Includes analysis time but not parsing, base type checking.
## Cost: Analysis time

<table>
<thead>
<tr>
<th></th>
<th>size (loc)</th>
<th>analysis time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>UTH</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>SCR</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>ECORDER</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>Z</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>IP</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>K</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>IT</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>PARKLE</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td>ASIHTTPR</td>
<td>461080</td>
<td>20.09s</td>
<td>23.0</td>
</tr>
</tbody>
</table>

**Notes:**
- Includes analysis time but not parsing, base type checking.
- Does not include system headers.
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>analysis time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23.0</strong></td>
</tr>
</tbody>
</table>

**Fast:** 5 to 38 kloc/s with most time spent analyzing system headers
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time Time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCR-recorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>20.09s</td>
<td>23.0</td>
</tr>
</tbody>
</table>

**Fast:** 5 to 38 kloc/s with most time spent analyzing system headers

**Interactive speeds**
Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (Time)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCReCORDER</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>AIDIUM</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23.0</strong></td>
</tr>
</tbody>
</table>

Fast: 5 to 38 kloc/s with most time spent analyzing system headers

Higher rate for projects with larger translation units
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (Time)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33.0</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>20.09s</td>
<td>23.0</td>
</tr>
</tbody>
</table>

Fast: 5 to 38 kloc/s with most time spent analyzing system headers.

Maintains key benefit of flow-insensitive analyses: speed.
Summary

- Check **almost everywhere** heap invariants with **intertwined type** and **symbolic flow analysis**
- **Translate** type environment into symbolic state with **symbolization**
- **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**
- Approach is **very fast** and scales to large programs
Fissile Type Analysis yields significant precision improvement at little cost in performance
Fissile Type Analysis yields significant precision improvement at little cost in performance

Why?
Fissile Type Analysis yields significant precision improvement at little cost in performance.

Why?

Because almost–everywhere invariants hold almost everywhere.
## Manual annotation burden

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>reflective call sites</th>
<th>annotation count</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>5</td>
<td>2 (-71%)</td>
<td>0.30s</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>0 (-100%)</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
<td>0.10s</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0</td>
<td>4 (1 (-75%))</td>
<td>0.78s</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
<td>50 (10 (-80%))</td>
<td>0.15s</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>49</td>
<td>82 (74 (-10%))</td>
<td>4.61s</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
<td>59 (38 (-36%))</td>
<td>2.57s</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
<td>43 (43 (-0%))</td>
<td>2.55s</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87 (70 (-20%))</td>
<td>7.50s</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334 (238 (-29%))</td>
<td>18.83</td>
</tr>
</tbody>
</table>
## Manual annotation burden

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>reflective call sites</th>
<th>annotation count</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>0.30s</td>
</tr>
<tr>
<td>Recorder</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>2 (10%)</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
<td>0.10s</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0</td>
<td>4 (75%)</td>
<td>0.78s</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
<td>50 (80%)</td>
<td>0.15s</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (10%)</td>
<td>4.61s</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
<td>59 (36%)</td>
<td>2.57s</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
<td>43 (0%)</td>
<td>2.55s</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87 (20%)</td>
<td>7.50s</td>
</tr>
<tr>
<td>Combined</td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334 (29%)</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

essentially zero for clients of reflection
<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>annotation count</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>5</td>
<td>7 (−71%)</td>
<td>0.30s</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>2 (−100%)</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (&lt;−)</td>
<td>0.10s</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0</td>
<td>4 (−75%)</td>
<td>0.78s</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
<td>50 (−80%)</td>
<td>0.15s</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>49</td>
<td>82 (−10%)</td>
<td>4.61s</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
<td>59 (−36%)</td>
<td>2.57s</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
<td>43 (−0%)</td>
<td>2.55s</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87 (−20%)</td>
<td>7.50s</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334 (−29%)</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

**essentially zero for clients of reflection**

**higher for frameworks exporting reflective interfaces**
## Manual annotation burden

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>annotation count</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>0.30s</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>2</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0.10s</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0</td>
<td>4</td>
<td>0.78s</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
<td>50</td>
<td>0.15s</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>49</td>
<td>82</td>
<td>4.61s</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
<td>59</td>
<td>2.57s</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
<td>43</td>
<td>2.55s</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87</td>
<td>7.50s</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
<td>133</td>
<td>334</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow-insens.</td>
<td>Time</td>
</tr>
<tr>
<td>almost-everywhere</td>
<td>Rate (kloc/s)</td>
</tr>
<tr>
<td>symbolic sections</td>
<td></td>
</tr>
<tr>
<td>max. materializations</td>
<td></td>
</tr>
</tbody>
</table>

- **Time (s)**: 0.30, 0.28, 0.10, 0.78, 0.15, 4.61, 2.57, 2.55, 7.50, 18.83
- **Rate (kloc/s)**: 4.1, 9.8, 33.8, 6.8, 90.2, 34.9, 14.5, 23.6, 23.6, 24.5

Essentially zero for clients of reflection, higher for frameworks exporting reflective interfaces, in-between for applications and large frameworks (which do both)
application code

```java
class MyButton {
    var cb : Callback = ...

    def setState(s : Str)
        var m = "draw" + s
        cb.update(self, m)
    end

    def draw()
        cb.call()
    end

    def drawUp() ... end
    def drawDown() ... end
}
```

library code

```java
class Callback {
    var sel : Str = ...
    var obj : Obj = ...

    def update(s : Str,
               o : Obj)
        this.sel = s
        this.obj = o
    end

    def call()
        this.obj.[this.sel]()
    end
}
```

*Idiomatic reflection decouples callbacks and avoids boilerplate*
Idiomatic reflection decouples callbacks and avoids boilerplate
Idiomatic reflection decouples callbacks and avoids boilerplate
class MyButton {
    var cb : Callback = ...

    def setState(s : Str)
        var m = “draw” + s
        cb.update(self, m)
    end

    def draw()
        cb.call()
    end

    def drawUp() ... end
    def drawDown() ... end
}

class Callback {
    var sel : Str = ...
    var obj : Obj = ...

    def update(s : Str,
        o : Obj)
        this.sel = s
        this.obj = o
    end

    def call()
        this.obj.[this.sel]()
    end
}

**Idiomatic reflection decouples callbacks and avoids boilerplate**
class MyButton {
    var cb : Callback = ...

    def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
end

def draw()
    cb.call()
end

def drawUp() ... end

def drawDown() ... end
}

class Callback {
    var sel : Str = ...
    var obj : Obj = ...

    def update(s : Str, o : Obj)
        this.sel = s
        this.obj = o
    end

    def call()
        this.obj.[this.sel]().
    end
}

Idiomatic reflection decouples callbacks and avoids boilerplate