Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

University of Maryland, College Park
February 26, 2014
Lab: Program analysis in the whole bug mitigation process
Lab: **Program analysis in the whole bug mitigation process**

![Diagram showing program analysis process]

- **Program**
- **Verifier**
  - Green check: proof of no bug
  - Red cross: alarm report
Lab: Program analysis in the whole bug mitigation process

Verifier

Program

Manual Triaging

Alarm Report

proof of no bug

✔ proof of no bug

✘
Lab: **Program analysis in the whole bug mitigation process**
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]
Lab: Program analysis in the whole bug mitigation process

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

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**

- **Speciﬁcation**

- **Program**

- **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]

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

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

- **Verifier**
  - Spec-ification
  - Program

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

- **Verifier**
  - **Proof of no bug**
  - Alarm Report

- **Test Input**
- **Test Output**

- **Runner**
- **Speciﬁcation**
- **Program**

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

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

- **Programming**

- **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**

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

- **Static Incrementalization of Data Structure Checks**
  - [in preparation]

- **Test**
  - Input
  - Output

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

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

- **Programming**
  - Test Input
  - Specification
  - Program

- **Verifier**
  - Test Output
  - Alarm Report
  - Static Incrementalization of Data Structure Checks
    - [in preparation]
  - Jsana: Abstract Domain Combinators for Dynamic Languages
    - [Cox+ ECOOP’13]
  - Thresher: Assisting Triage by Refutation Analysis
    - [Blackshear+ PLDI’13, Blackshear+ SAS’11]

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

- **Runner**
Lab: Program analysis in the whole bug mitigation process

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

Programming

Test Input

Verifier

Static Incrementalization of Data Structure Checks
[in preparation]

Alarm
Report

Test Output

proof of no bug

Runner

Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP’13]

Verifier

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

Manual Triaging

This Talk

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14]
Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

University of Maryland, College Park
February 26, 2014
How to type check a program that is almost well-typed?
In this talk

Example property of interest: safety of reflective method calls

Type 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

```python
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

```python
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
```

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 run-time value (in string)

object. [string]()

“static” folks

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

What gets called? What if object has no method named by string?

“web 2.0” developers

“web 2.0” developers think it’s cool.

I can flexible and compact code, so I will take it over 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

“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 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.
Ensure reflection safety with **dependent refinement type** expressing required relationship

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

    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]()
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

```java
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”
Similar relationship for array bounds safety

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

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

idx must be a valid index into buf

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]()`
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]()
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]()
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]()
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 | 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

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
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

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.

Γ ⊢ · · ·

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

Automated reasoning about global invariants

Automated reasoning about execution

\[ \Gamma \vdash \ldots \]

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

Automated reasoning about **global** invariants

Automated reasoning about **execution**

\[ \Gamma \vdash \cdots \]

**Flow-Insensitive Type Systems**

\[ \gamma(\cdot) = \cdots \]

**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
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.

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

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
- Symbolic flow analysis
- Types restored
- Types violated
- Relationship updates
- Occurrence typing
- Tagged unions

Effective when **global type invariant holds most of the time**
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
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?*

**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
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.

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

**Type environment**

Maps local **variables** to dependent **types**

\[
\Gamma \\
\begin{align*}
    s &: Str \\
    o &: Obj | r2 s \\
    this &: Callback
\end{align*}
\]

**Refinements refer to variables**
Symbolization splits a type environment into facts about values and storage for those values.

Type environment

Maps local variables to dependent types

\[ \Gamma \]

- \( s : \text{Str} \)
- \( o : \text{Obj} \mid r2 \ s \)
- \( \text{this} : \text{Callback} \)

Symbolic state

\[ \tilde{E} \]

- \( s : \tilde{s} \)
- \( o : \tilde{o} \)
- \( \text{this} : \tilde{t} \)

Def: Update function

\[ \text{def update}(s:\text{Str}, o:\text{Obj} \mid r2 \ s) \]

- \( \text{this}.sel = s \)
- \( \text{this}.obj = o \)

Refinements refer to variables

symbolize
Symbolization splits a type environment into facts about values and storage for those values.

Type environment
Maps local variables to dependent types

$\Gamma$

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

Symbolic state
Maps local variables to symbolic values

$\tilde{\Gamma}$

$\tilde{s} : \tilde{s}$
$\tilde{o} : \tilde{o}$
this : $\tilde{t}$

$\tilde{E}$

$s : \tilde{s}$
$o : \tilde{o}$
this : $\tilde{t}$

$\tilde{\Gamma}$

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

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

\[ \Gamma \]

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

Symbolic state

Maps local variables to symbolic values

\[ \tilde{\Gamma} \]

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

symbolize

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

Refinements refer to variables

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

Type environment
- Maps local variables to dependent types

Symbolic state
- Maps local variables to symbolic values
- Refinements refer to values

\[ \Gamma \]
\[
\begin{align*}
  s &: Str \\
  o &: Obj | r2 s \\
  this &: Callback
\end{align*}
\]

\[ \tilde{E} \]
\[
\begin{align*}
  \tilde{s} &: \tilde{\tilde{S}} \\
  \tilde{o} &: \tilde{\tilde{O}} \\
  this &: \tilde{\tilde{t}} \\
  \tilde{s} &: Str \\
  \tilde{o} &: Obj | r2 \tilde{s} \\
  \tilde{t} &: Callback
\end{align*}
\]

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

Maps local variables to dependent types

Maps local variables to symbolic values

Maps symbolic values to dependent types

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

Reefinements refer to variables
Symbolization allows local variables to hold values inconsistent with declared types

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

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

A type environment constrains local variables

Γ
s : Str
o : Obj | r2 s
this : Callback

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

A type environment constrains local variables

\[
\Gamma = \\
\begin{align*}
&\text{s : Str} \\
&o : \text{Obj | r2 s} \\
&\text{this : Callback}
\end{align*}
\]

```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

Γ

s : Str
o : Obj | r2 s
this : Callback

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
```

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

\[\text{symbolize} \quad \tilde{E} \quad \tilde{\Gamma}\]  

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

\[
\begin{array}{c}
\text{this.obj} \\
\text{this.sel}
\end{array}
\]  

heap
Symbolization allows local variables to hold values inconsistent with declared types.

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

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

symbolize \( \tilde{E} \leftarrow \tilde{\Gamma} \)

\[ \begin{align*}
\text{def update}(s &: \text{Str}, o &: \text{Obj} \mid r2 \ s) \\
\text{this}.\text{sel} &= s \\
\text{this}.\text{obj} &= o
\end{align*} \]
Symbolization allows local variables to hold values inconsistent with declared types.

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

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

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

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 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

\[
\begin{align*}
\Gamma & \quad s : \text{Str} \\
& \quad o : \text{Obj} \mid r2\ s \\
& \quad \text{this} : \text{Callback}
\end{align*}
\]

\[
\text{def update}(s : \text{Str}, o : \text{Obj} \mid r2\ s)
\]
\[
\text{this}.\text{sel} = s \\
\text{this}.\text{obj} = o
\]
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.
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

Type environment

Base types same on both sides

Symbolic fact map

\[ \Gamma \vdash s : \text{Str} \quad o : \text{Obj} \mid r^2 s \quad \text{this} \] symbolize \[ \tilde{\Gamma} \quad \tilde{s} : \text{Str} \quad \tilde{o} : \text{Obj} \mid r^2 \tilde{s} \quad \text{this.Callback} \]

Heap

\[ \text{this.obj} \quad \text{this.sel} \]
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

Type environment

\[ \Gamma \vdash \text{Callback} \]

Base types same on both sides

\[ \tilde{\Gamma} \vdash \tilde{\text{Callback}} \]

Symbolic fact map

Callback $\triangleq \{ \text{sel} : \text{Str}, \text{obj} : \text{Obj} \mid r2 \text{ sel} \}$

Base type field refinements still refer to fields

- $s : \text{Str}$
- $o : \text{Obj} \mid r2 s$
- $\text{this} : \text{Callback}$
- $\tilde{s} : \text{Str}$
- $\tilde{o} : \text{Obj} \mid r2 \tilde{s}$
- $\tilde{\text{this}} : \text{Callback}$
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
```
Summarize heap locations that are **not** immediately type-inconsistent with okheap.

**Symbolic Heap**

$$\tilde{H}$$

**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**

\[ \tilde{H} \]

\[ H \]

okheap

**Concrete State**

Describes storage without explicitly enumerating it

```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

\( \widetilde{H} \)

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

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

\[ H \]

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

\[ \tilde{H} \]

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

Concrete State

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

\[ \text{this.obj} \]

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

Materialize onto standard separation-logic explicit heap

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

**Concrete State**

Concrete State Diagram:

- `s`
- `o`
- `this`
- `this.obj`
- `this.sel`

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

**Must-alias and disalias guarantee requires case split on materialization**
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \widetilde{H} \]

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

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
    if sel ⇤
guard this
    obj ⇤
7!
{ sel ⇤ sel \ast obj ⇤ obj }
```

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

Materialize onto standard separation-logic explicit heap

\[ \tilde{H} \]

Materialized storage guaranteed to be not immediately type-inconsistent

Must-alias and disalias guarantee requires case split on materialization

Concrete State

Value stored in obj responds to value stored in 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 obj responds to value stored in sel

Concrete State

Represent materialized storage with
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 \texttt{obj} responds to value stored in \texttt{sel}

Concrete State

Analysis can **assume that type invariant initially holds on all materialized storage**

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

Materialized storage guaranteed to be not immediately type-inconsistent
Strong updates on materialized storage to detect invariant restoration

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

Symbolic State

$\tilde{H}$

$\text{okheap} \ast \tilde{\text{this}} \mapsto \{\text{sel} \mapsto \tilde{\text{sel}} \ast \text{obj} \mapsto \tilde{\text{obj}}\}$

Concrete State
Strong updates on materialized storage to detect invariant restoration

Concrete State

Symbolic State

\[
\tilde{H} = \text{okheap} \star \begin{array}{c}
\text{this} \mapsto \{ \text{sel} \mapsto \tilde{s} \star \text{obj} \mapsto \tilde{\text{obj}} \}
\end{array}
\]

```
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
```
Strong updates on materialized storage to detect invariant restoration

Symbolic State

\[ \tilde{H} \]

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

Concrete State

Concrete State

```python
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

Symbolic State

\[ \overset{\sim}{H} \]
\[ \text{okheap} \ast \overset{\sim}{\text{this}} \mapsto \{ \text{sel} \mapsto \overset{\sim}{s} \ast \text{obj} \mapsto \overset{\sim}{\text{obj}} \} \]

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

Type invariant violated

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Strong updates on materialized storage to detect invariant restoration

Symbolic State

\[ \tilde{H} \]
okheap \* \[ \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{s}, \text{obj} \mapsto \tilde{o} \} \]

Concrete State

```
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Strong updates on materialized storage to detect invariant restoration

Concrete State

Symbolic State

\[ \tilde{H} \]
\[
\text{okheap} \times \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{s} \times \text{obj} \mapsto \tilde{o} \}
\]

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

No longer immediately type-inconsistent
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**

\[
\hat{H}
\]

\[
\text{okheap} \times \hat{\text{this}} \mapsto \{ \text{sel} \mapsto \hat{s} \times \text{obj} \mapsto \hat{o} \}
\]

**Concrete State**

```
S
```

```
O
```

```
this
```
Safely **summarize** storage that is not immediately type inconsistent

**Symbolic State**

\[ \tilde{H} \]

\[ \text{okheap} \]

**Concrete State**

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

Diagram showing the symbolic and concrete states with variables `s`, `o`, and `this`.
Safely **summarize** storage that is not immediately type inconsistent

**Symbolic State**

\[ \tilde{H} \]

okheap

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

**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.

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

**Concrete State**

**Symbolic State**

\[ H \]

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

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

\[ \tilde{H} \]

**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
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>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>
## 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>
### 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>OmniFramework</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>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>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.
## 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><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><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>
</tbody>
</table>

Combining: 125 2

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

*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>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>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>
## Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective calls</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</td>
<td>2 (−71%)</td>
</tr>
<tr>
<td>SCRECORER</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>0 (−100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0 (−)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1</td>
<td>1 (−75%)</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10</td>
<td>10 (−80%)</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74</td>
<td>74 (−10%)</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38</td>
<td>38 (−36%)</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43</td>
<td>43 (−0%)</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70</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</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>
<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>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>OmniFramework</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
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>

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

Does **not** include system headers

Includes analysis time but **not** parsing, base type checking

Does **not** include system headers

Cost: Analysis **time**

Includes analysis time but **not** parsing, base type checking
## 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><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</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><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

**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>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

**Higher rate** for projects with larger translation units
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>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.

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>(loc)</th>
<th>reflective call sites</th>
<th>annotation count</th>
<th>false alarms</th>
<th>almost-everwhere</th>
<th>symbolic sections</th>
<th>max. materializations</th>
<th>Time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>2 (-71%)</td>
<td>7</td>
<td>1</td>
<td>0.30s</td>
<td>4.1</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>2</td>
<td>0 (-100%)</td>
<td>2</td>
<td>2</td>
<td>0.28s</td>
<td>9.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0 (-)</td>
<td>0</td>
<td>0</td>
<td>0.10s</td>
<td>33.8</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0</td>
<td>4</td>
<td>1 (-75%)</td>
<td>3</td>
<td>1</td>
<td>0.78s</td>
<td>6.8</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
<td>50</td>
<td>10 (-80%)</td>
<td>59</td>
<td>2</td>
<td>0.15s</td>
<td>90.2</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>49</td>
<td>82</td>
<td>74 (-10%)</td>
<td>7</td>
<td>1</td>
<td>4.61s</td>
<td>34.9</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
<td>59</td>
<td>38 (-36%)</td>
<td>28</td>
<td>2</td>
<td>2.57s</td>
<td>14.5</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
<td>43</td>
<td>43 (-0%)</td>
<td>0</td>
<td>0</td>
<td>2.55s</td>
<td>23.6</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87</td>
<td>70 (-20%)</td>
<td>16</td>
<td>1</td>
<td>7.50s</td>
<td>23.6</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334</td>
<td>238 (-29%)</td>
<td>125</td>
<td>2</td>
<td>18.83</td>
<td>24.5</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>symbolic sections</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248 (loc)</td>
<td>7</td>
<td>5</td>
<td>7 (2 (-71%))</td>
<td>7</td>
<td>0.30s</td>
</tr>
<tr>
<td>SCRECORER</td>
<td>2716</td>
<td>12</td>
<td>9</td>
<td>2 (0 (~100%))</td>
<td>2</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (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>3</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>59</td>
<td>0.15s</td>
</tr>
<tr>
<td>OmniFRAMEWORKS</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (~10%)</td>
<td>7</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>28</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>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>16</td>
<td>7.50s</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334 (238 (~29%))</td>
<td>125</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

essentially zero for clients of reflection
## Manual annotation burden

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td>reflective call sites</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>annotation count</td>
<td>Rate (kloc/s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>flow-insens.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>almost-everwhere</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>symbolic sections</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>max. materializations</td>
<td></td>
</tr>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>0.30s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>4.1</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>0.28s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>9.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0.10s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>33.8</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>0.78s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>6.8</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>0.15s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>90.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>4.61s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>49</td>
<td>34.9</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>2.57s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td>14.5</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>2.55s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>23.6</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>7.50s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>23.6</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>18.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>136</td>
<td>24.5</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></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>2 (0 (-100%))</td>
<td>0.28s</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (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>Skium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
<td>87 (70 (-20%))</td>
<td>7.50s</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>134</td>
<td>334 (238 (-29%))</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

essentially zero for clients of reflection
higher for frameworks exporting reflective interfaces
in-between for applications and large frameworks (which do both)
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() ...
    def drawDown() ...
}

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
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
Idiomatic reflection decouples callbacks and avoids boilerplate
Idiomatic reflection decouples callbacks and avoids boilerplate