Fissile Type Analysis:
Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Japan Advanced Institute of Science and Technology
August 6, 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

Verifier

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

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

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

- **Verifier**
  - proof of no bug

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

- **Manual Triaging**
  - [Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]
Lab: **Program analysis** in the whole bug mitigation process

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

- **Programming**
  - Spec-ification
  - Program

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

- **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]

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

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

**Manual Triaging**

**Runner**

**Verifier**

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

**Test Input**

**Program**

**Specification**

**Test Output**

**Alarm Report**

proof of no bug

Enforcement Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12]
Lab: **Program analysis in the whole bug mitigation process**

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

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

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

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

- **Runner**
  - Test Input
  - Specification
  - Program

- **Alarm Report**
  - ✔ proof of no bug
  - ✘
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, in prep]

**Verifier**

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

**Jsana**: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP’13, Cox+ SAS’14, under review]

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

**Manual Triaging**

Test Input
Specification
Program

Test 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**

**Proof of no bug**

**Alarm Report**

**Runner**

**Test Output**

**Jsana: Abstract Domain Combinators for Dynamic Languages**
[Cox+ ECOOP’13, Cox+ SAS’14, under review]

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

**Manual Triaging**

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

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

**This Talk**
Fissile Type Analysis:
Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Japan Advanced Institute of Science and Technology
August 6, 2014
Fissile Type Analysis:

Types and Separation Logic, Better Together

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Japan Advanced Institute of Science and Technology
August 6, 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

```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
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]()
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

“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 write 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 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

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[]

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

idx must be a valid index into buf
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*

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

```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`
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]()
Two styles of reasoning to determine false alarm

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

    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]()}
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 about effects of imperative updates:

Relationship violated:

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]()
Idea: Selectively alternate between reasoning styles in verification
Fissile Type Analysis \textbf{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

\( \Gamma \vdash \ldots \)

Flow-Insensitive Type Systems
Automated reasoning 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

\[ \Gamma \vdash \cdots \]

Flow-Insensitive Type Systems

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

Abstract Interpretation/Flow Analysis/Model Checking
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

- Type analysis
  - Types violated
  - Symbolic flow analysis
  - Types restored
  - Type analysis
  - Types restored
  - Symbolic flow analysis
  - Types violated
  - Type 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 violated
      - Types restored
        - Types violated
          - 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**
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 environment.
   - 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

```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.

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

Type environment

Maps local variables to dependent types

\[ \Gamma \]

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

Refinements refer to variables

```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

Symbolic state

Refinements refer to variables

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

Symbolize

\[ \tilde{\Gamma} \]
\[ s : \tilde{s} \]
\[ o : \tilde{o} \]
\[ \text{this} : \tilde{\text{t}} \]

\[ \hat{\Gamma} \]
\[ \tilde{s} : \text{Str} \]
\[ \tilde{o} : \text{Obj} \mid r2 \ \tilde{s} \]
\[ \tilde{\text{t}} : \text{Callback} \]

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

\[ \Gamma \]

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

Symbolic state

Maps local variables to symbolic values

\[ \bar{\bar{\Xi}} \]

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

Refinements refer to variables

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

\[ \begin{array}{c}
\Gamma \\
\hline
s : Str \\
on : Obj | r2 s \\
this : Callback
\end{array} \]

**Symbolic state**

Maps local variables to symbolic values

\[ \begin{array}{c}
\tilde{\Gamma} \\
\hline
\tilde{s} : \tilde{\tilde{s}} \\
\tilde{o} : \tilde{o} \\
\tilde{this} : \tilde{\tilde{t}} \\
\tilde{s} : Str \\
\tilde{o} : Obj | r2 \tilde{s} \\
\tilde{t} : Callback
\end{array} \]

**symbolize**

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

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

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

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

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

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

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

**Refinements refer to values**

**Refinements refer to variables**

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

```python
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

```
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

\[ \Gamma \]

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

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

\[
\text{heap}
\]

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

A type environment constrains local variables:

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

```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

\[ \Gamma \]

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

```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.

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

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

symbolize $\sim E \sim \Gamma$

$\sim E$

heap

this.obj

this.sel

S

O

this
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
```
Symbolization allows local variables to hold values inconsistent with declared types

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

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

\[ \text{Immediately type-inconsistent: value stored without dereferences violates a type constraint} \]
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
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
```

Grey indicates storage that is not immediately type-inconsistent.

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

\[ \Gamma \]

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

Symbolic fact map

\[ \tilde{\Gamma} \]

\[
\begin{align*}
\tilde{\Gamma} & \quad \tilde{s} : \text{Str} \\
& \quad \tilde{o} : \text{Obj} | r2 \ \tilde{s} \\
& \quad t : \text{Callback}
\end{align*}
\]
Symbolization un packs local cells, but symbolic facts about values still constrain the heap.

Type environment: $\Gamma$

$s : \text{Str}$

$o : \text{Obj} \mid r2 \ s$

this \ Callback

Base types same on both sides

Symbolic fact map: $\tilde{\Gamma}$

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

$\tilde{o} : \text{Obj} \mid r2 \ \tilde{s}$

this.obj \ Callback

this.sel

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

Type environment

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

\[ s : \text{Str}, \;
\text{this} : \text{Callback} \]

\[ \text{Base types same on both sides} \]

Symbolic fact map

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

\[ \tilde{s} : \text{Str}, \;
\tilde{o} : \text{Obj} | r2 \tilde{s} \]

Callback \( \triangleq \{ \text{sel} : \text{Str}, \text{obj} : \text{Obj} | r2 \text{sel} \} \)

Base type field refinements still refer to fields.
Summarize heap locations that are not immediately type-inconsistent with \textit{okheap}.

\begin{itemize}
  \item [\textbf{Symbolic Heap}]
    \[ \tilde{H} \]
    \text{okheap}
  
  \item [\textbf{Concrete State}]
    \texttt{def update(s:Str, o:Obj \mid r2 s)}
    \begin{align*}
      & \text{this.sel} = s \\
      & \text{this.obj} = o
    \end{align*}
\end{itemize}
Summarize heap locations that are **not** immediately type-inconsistent with okheap

Concrete State

Symbolic Heap

\[ \tilde{H} \]

okheap

```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

\[ \tilde{H} \]

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 \( \text{okheap} \)

**Symbolic Heap**

\[ \tilde{H} \]

**Concrete State**

Describes storage without explicitly enumerating it

\[ H_{\text{okheap}} \]

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

Immediately after switch, type invariants still hold so \( \text{okheap} \) represents entire heap

```python
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.
Leverage heap type invariant via type-consistent materialization

Symbolic State

$\tilde{H} \quad \text{okheap}$

Concrete State

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

heap

this.obj

this.sel
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

$$\tilde{H}$$

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

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

Concrete State

This diagram represents the concrete state of an object with attributes `s`, `o`, and `this`. The diagram shows the relationship between these attributes and the heap, illustrating how materialization is applied to maintain the heap type invariant.
Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard *separation-logic* explicit heap

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

![Concrete State Diagram]

**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

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

Materialized storage guaranteed to be not immediately type-inconsistent

\[ \sim H \quad \text{okheap} * \sim \text{this} \mapsto \{ \text{sel} \mapsto \sim \text{sel} * \text{obj} \mapsto \sim \text{obj} \} \]

Must-alias and disalias guarantee requires case split on materialization

Concrete State
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

\[ \tilde{H} \]

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

Must-alias and disalias guarantee requires case split on materialization

Value stored in obj responds to value stored in sel

Concrete State

\[ \text{s} \]
\[ \text{o} \]
\[ \text{this} \]

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

heap
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

Materialized storage guaranteed to be not immediately type-inconsistent

\[ \tilde{H} \]

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

Must-alias and disalias guarantee requires case split on materialization

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

Concrete State

Represent materialized storage with

\[ \begin{align*}
S & \\
O & \\
\text{this} & \\
\end{align*} \]
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

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

Symbolic State

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

Concrete State

```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

\[ \overline{\mathcal{H}} \]
\[ \text{okheap} \land \overline{\text{this}} \mapsto \{ \text{sel} \mapsto \overline{s} \land \text{obj} \mapsto \overline{\text{obj}} \} \]

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

Symbolic State

\[ \tilde{H} \]

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

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

Symbolic State

\[ \tilde{H} \quad \text{okheap} \land \exists \tilde{\text{this}} \mapsto \{ \text{sel} \mapsto \tilde{s} \land \text{obj} \mapsto \tilde{o} \} \]

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

Type invariant violated

\[ \text{def update(s:Str, o:Obj | r2 s)} \]
\[ \quad \text{this.sel} = s \]
\[ \quad \text{this.obj} = o \]
Strong updates on materialized storage to detect invariant restoration

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

Symbolic State

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

Concrete State
Strong updates on materialized storage to detect invariant restoration

Symbolic State

Concrete State

No longer immediately type-inconsistent
Safely **summarize** storage that is not immediately type inconsistent

**Concrete State**

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

**Symbolic State**

\[
\hat{H} \quad \text{okheap} \ast \\{ \text{this} \mapsto \{ \text{sel} \mapsto \tilde{s} \ast \text{obj} \mapsto \tilde{o} \} \}
\]
Safely **summarize** storage that is not immediately type inconsistent

**Symbolic State**

\[ \tilde{H} \]

**Concrete State**

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

\[ H \]
Safely **summarize** storage that is not immediately type inconsistent.

**Concrete State**

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

**Symbolic State**

- `okheap
  \[ \tilde{H} \]

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

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

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

**Symbolic State**

\[ \widetilde{H} \]

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

**Concrete State**

\[ \text{okheap} \]

**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>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><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>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**

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>SCR</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>IP</td>
<td>2716</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>KT</td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ECORDER</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>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>

Maximum number of simultaneous materialized storage locations

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>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><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>Total</strong></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>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><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 call sites</th>
<th>false alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</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>4</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>50</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>82</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>87</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
<td>334</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></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>false alarms (flow-insensitive)</th>
<th>false alarms (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>ASIHTTRequest</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>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (-29%)</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>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

Also found a real reflection bug in Vienna, which we reported and which was fixed.

Almost everywhere techniques show 29% improvement in false alarms.
## 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>SCRECORER</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>
## 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>CPrepper</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>KIT</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>S</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>

- 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 (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>SCRecoerDR</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>combined</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
### 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>SCRecord</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

**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**.
**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.
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**.
Reiteration: 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.
Reiteration: Fissile Type Analysis is sound

**Theorem (Soundness of Handoff).**

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

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

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

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

---

**No locality or framing**
Next Steps: Gated Separation

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Gated Separation

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Gated Separation

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

no pointer from top to bottom
Next Steps: Gated Separation

The entire state is **type-consistent** iff all locations are **not** immediately type-inconsistent.
Next Steps: Gated Separation

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Gated Separation

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Gated Separation

The entire state is **type-consistent** iff all locations are **not** immediately type-inconsistent.
Next Steps: Gated Separation

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Gated Separation

“The entire state is type-consistent iff all locations are not immediately type-inconsistent.”
Next Steps: Gated Separation

“foregate”

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

\[ \sigma \models_V M_1 \trianglelefteq M_2 \quad \text{iff} \quad \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2 \]

where \( \sigma_1 \models_V M_1 \) and \( \sigma_2 \models_V M_2 \)

and \( \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)

and \( \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
Next Steps: Gated Separation

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

\[ \sigma \models_V M_1 \triangleleft M_2 \text{ iff } \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2 \]

where \( \sigma_1 \models_V M_1 \) and \( \sigma_2 \models_V M_2 \)
and \( \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
and \( \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
Next Steps: Gated Separation

“foregate”

The entire state is \textit{type-consistent} iff all locations are \textbf{not} immediately \textit{type-inconsistent}.

\[
\sigma \models_V M_1 \triangleleft M_2 \iff \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2
\]

where \( \sigma_1 \models_V M_1 \) and \( \sigma_2 \models_V M_2 \)

and \( \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)

and \( \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
Next Steps: Gated Separation

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

\[ \sigma \models_V M_1 \triangleleft M_2 \quad \text{iff} \quad \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2 \]

where \( \sigma_1 \models_V M_1 \) and \( \sigma_2 \models_V M_2 \) and \( \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \) and \( \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
Next Steps: Gated Separation

"foregate"

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

\[\sigma \models_{V} M_1 \triangleleft M_2 \iff \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2 \]

where \(\sigma_1 \models_{V} M_1\) and \(\sigma_2 \models_{V} M_2\) and \(\text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset\) and \(\text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset\)
Next Steps: Gated Separation

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

\[
\sigma \models_V M_1 \triangleleft M_2 \iff \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2
\]
where \( \sigma_1 \models_V M_1 \) and \( \sigma_2 \models_V M_2 \)
and \( \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)
and \( \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \)

Slightly stronger than \( \star \): No direct pointers from “foregate” to “aftgate”
Next Steps: Type–Intertwined Framing

The entire state is type-consistent iff all locations are not immediately type-inconsistent.
Next Steps: Type–Intertwined Framing

"foregate"

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

symbolic flow analysis
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis
type analysis

"foregate"
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis
type analysis
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

The entire state is \textit{type-consistent} iff all locations are \textbf{not immediately type-inconsistent}.

\[ \Gamma \]

"foregate"

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

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

symbolic flow analysis

**foregate**

type analysis
Next Steps: Type–Intertwined Framing

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

Type–intertwined framing is sound because “aftgate” is not reachable.
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.
Fissile Type Analysis yields significant precision improvement at little cost in performance

Why?

Because almost-everywhere invariants hold almost everywhere

www.cs.colorado.edu/~bec
pl.cs.colorado.edu
Fissile Type Analysis yields significant precision improvement at little cost in performance.

Why?

Because almost-everywhere invariants hold almost everywhere.

www.cs.colorado.edu/~bec
pl.cs.colorado.edu
## 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>symbolically insensitive</th>
<th>almost everywhere</th>
<th>symbolic sections</th>
<th>max. materializations</th>
<th>Time (s)</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></td>
<td>0.30</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></td>
<td>0.28</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></td>
<td>0.10</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></td>
<td>0.78</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></td>
<td>0.15</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></td>
<td>4.61</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></td>
<td>2.57</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></td>
<td>2.55</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></td>
<td>7.50</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></td>
<td>18.83</td>
<td>24.5</td>
</tr>
</tbody>
</table>
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>Flow-insens.</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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (−10%)</td>
<td>7</td>
<td>1</td>
<td>4.61s</td>
<td>34.9</td>
<td></td>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>Combined</td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334</td>
<td>238 (−29%)</td>
<td>125</td>
<td>2</td>
<td>18.83s</td>
<td>24.5</td>
<td></td>
</tr>
</tbody>
</table>

essentially zero for clients of reflection
## 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>combined</td>
<td>461080</td>
<td>1327</td>
<td>136</td>
<td>334</td>
<td>18.83s</td>
</tr>
</tbody>
</table>

- **Time:**
- **Rate (kloc/s):**

- **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>flow-insens.</th>
<th>almost-everwhere</th>
<th>symbolic sections</th>
<th>max. materializations</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>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>OmniFrameworks</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><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>12</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
<td><strong>125</strong></td>
<td><strong>2</strong></td>
<td><strong>18.83</strong></td>
<td><strong>24.5</strong></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)
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
}

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() ...
    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**
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