Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang 張博聿
University of Colorado Boulder

Academia Sinica 中央研究院
July 30, 2014
Lab: **Program analysis** in the **whole** bug mitigation **process**
Lab: **Program analysis in the whole bug mitigation process**

- Program
- Verifier
- Alarm Report

- ✔ proof of no bug
- ✘ Alarm Report
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

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

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

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

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

**Program:**

- Specification
- 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

- **Programming**
  - Enforcement Windows: Measuring Bug Avoidance
    - [Coughlin+ ISSTA'12]
  - Fissile Types: Checking Almost Everywhere Invariants
    - [Coughlin+ POPL'14, in prep]

- **Manual Triaging**

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

- **Manual Triaging**
  - Program
    - Specification

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

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

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

- **Manual Triaging**

- **Verifier**
  - Thresher: Assisting Triage by Refutation Analysis
    - [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**

**Test Input**  
**Verifier**  
**Program**  
**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]

**Manual Triaging**

**Runner**

**Test Output**

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

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

- **Program-**

- **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]

- **Manual Triaging**

  - Static Incrementalization of Data Structure Checks  
    [under review]

- **Runner**

  - Test Input

  - Specification

  - Program

- **Test Output**

  - proof of no bug

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

Verifier: Static Incrementalization of Data Structure Checks
[under review]

Runner

Manual Triaging

Test
Input

Test
Output

Alarm
Report

✔ proof of no bug

alarm
Report

reason

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

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

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

Programming

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

Manual Triaging
Lab: Program analysis in the whole bug mitigation process

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

- **Program**
  - Programming
  - Specification
  - Program

- **Verifier**
  - Test Input
  - Spec-ification
  - Program
  - Test Output

- **Runner**
  - Alarm Report

- **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

- **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 張博聿
University of Colorado Boulder

Academia Sinica 中央研究院
July 30, 2014
Fissile Type Analysis:

Types and Separation Logic, Better Together

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang 張博聿
University of Colorado Boulder

Academia Sinica 中央研究院
July 30, 2014
How to type check a program that is almost well-typed?
In this talk

Example property of interest: safety of reflective method calls

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

class Callback
    var sel : Str
    var obj : Obj

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

class Callback
var sel : Str
var obj : Obj

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

```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" it would call `notifyDidClick()` on `obj`. 
Reflective method call dispatches based on runtime string value

class Callback
  var sel : Str
  var obj : Obj

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

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

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

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

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

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

object. [string]()

“static” folks

“web 2.0” developers
Method Reflection and the Great Divide

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

object.[string]()
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

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

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

    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]()
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]()

Guarantees no MethodNotFound error in call()
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]
Similar relationship for array bounds safety

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

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

Guarantees no "ArrayOutOfBounds" error

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]

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

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

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

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

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

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

```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]().
```

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]()
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]()`
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

Relationship violated

Relationship restored
Idea: Selectively alternate between reasoning styles in verification
Fissile Type Analysis combines two styles of reasoning
Fissile Type Analysis combines two styles of reasoning

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

Automated reasoning about global invariants

\[ \Gamma \vdash \cdots \]

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

Automated reasoning about global invariants

Automated reasoning about execution

\[ \Gamma \vdash \ldots \]

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/
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

- Type analysis
- Symbolic flow analysis

Types violated

Switch to symbolic analysis when global type invariant violated

Types restored

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

Switch to symbolic analysis when global type invariant violated

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

- Type analysis
  - Symbolic flow analysis
  - Type analysis
  - Symbolic 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
symbolic flow analysis
type analysis
symbolic flow analysis
type analysis

Effective when global type invariant holds most of the time

• Relationship updates

types violated

types restored

types violated

types restored

types restored
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

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

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.

```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
```

✘
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
```

**Type environment**

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

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

Refinements refer to variables
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

\[ \Gamma \]

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

Symbolic state

\[ \sim \]

\[ \tilde{\Gamma} \]

- \( \tilde{s} : \tilde{\text{Str}} \)
- \( \tilde{o} : \tilde{\text{Obj}} | r2 \tilde{s} \)
- \( \tilde{\text{this}} : \tilde{\text{t}} \)

Refinements refer to variables
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

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

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

- $\tilde{\Gamma}: \tilde{s} : \tilde{\text{Str}}$
- $\tilde{o} : \tilde{\text{Obj}}$
- $\text{this} : \tilde{t}$
- $\tilde{\text{this}} : \tilde{\text{Callback}}$

**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

\[ \Gamma \]

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

Symbolic state

Maps local variables to symbolic values

\[ E \]

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

Refinements refer to variables

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

\[ \Gamma \]

\[ \sim \]

\[ \sim \]
Symbolization splits a type environment into facts about values and storage for those values.

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

Maps local variables to dependent types

Def. update(s:Str, o:Obj | r2 s)

\[
\text{this}.\text{sel} = s \\
\text{this}.\text{obj} = o
\]

Maps local variables to symbolic values

Maps symbolic values to dependent types

Symbolic state

\[
\begin{array}{c}
  \sim \\
  \tilde{s} : \text{Str} \\
  \tilde{o} : \text{Obj} | \text{r2} \tilde{s} \\
  \text{this} : \tilde{t}
\end{array}
\]

Refinements refer to values

Refinements refer to variables

Symbolize

Maps symbolic values lifted to symbolic values (symbolic facts)
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
```

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

A type environment constrains local variables.

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

```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

A type environment constrains local variables

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

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

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

A type environment constrains local variables

\( \Gamma \)

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

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
```

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

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

Symbolic environment

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

Symbolize

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

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

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

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

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

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
```

Symbolic environment

\[
\Gamma
\]

\[
\Gamma
\]

\[
\Gamma
\]

\[
\Gamma
\]

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

\[
\Gamma
\]

\[
\Gamma
\]

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

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

Grey indicates storage that is not immediately type-inconsistent
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

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

Base types same on both sides

Symbolic fact map

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

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

\[ \tilde{\Gamma} \]

Symbolic fact map

\[ \tilde{s} : \text{Str} \]

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

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 okheap

Symbolic Heap

\[ \tilde{H} \]

okheap

Concrete State

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

\[ \tilde{H} \]

okheap

Concrete State

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

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

**Symbolic Heap**

\[ \tilde{H} \]

\[ \text{okheap} \]

Describes storage without explicitly enumerating it.

**Concrete State**

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

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

**Symbolic Heap**

\[ \text{okheap} \]

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

Concrete State

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

Materialize onto standard separation-logic explicit heap

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

Concrete State

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

This diagram illustrates the update operation and its effect on the heap and state variables.`
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
```

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

**Must-alias and disalias guarantee** requires case split on materialization

Concrete State

- `s`
- `o`
- `this`

heap

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

Materialize onto standard separation-logic explicit heap

\[ \tilde{H} \]

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

Must-alias and disalias guarantee requires case split on materialization

Concrete State

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

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

\[ \text{heap} \]
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

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

Must-alias and disalias guarantee requires case split on materialization

Concrete State

Value stored in obj responds to value stored in sel

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

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

Materialize onto standard separation-logic explicit heap

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

Must-alias and disalias guarantee requires case split on materialization

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

Concrete State

Represent materialized storage with

\[ \begin{array}{c}
\text{S} \\
\text{O} \\
\text{this} \\
\end{array} \]

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

\[ \text{heap} \]

def update(s: Str, o: Obj | r2 s)
this.sel = s
this.obj = o
f
sel ⇤
g
this
7!
{ sel ⇤ obj ⇤ }
7!

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

Materialize onto standard separation-logic explicit heap

\[ \text{Concrete State} \]

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

Materialized storage guaranteed to be not immediately type-inconsistent

\[ \tilde{H} \]

Must-alias and disalias guarantee requires case split on materialization

Value stored in \( \text{obj} \) responds to value stored in \( \text{sel} \)
Strong updates on materialized storage to detect invariant restoration

Symbolic State

\[ \tilde{H} \]

\[ \tilde{H} \] is defined as:

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

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

\[ \widehat{H} \]

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

Concrete State

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

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

\[ \text{heap} \]
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

\[ \tilde{H} \]

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

Concrete State

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

Symbolic State

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

\[ \tilde{H} \]
okheap * \( \tilde{\text{this}} \) \( \rightarrow \) \{sel \( \rightarrow \) \( \tilde{s} \) * obj \( \rightarrow \) \( \tilde{obj} \)\}

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

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

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

Symbolic State

\[ \hat{H} \]
okheap * \( \hat{\text{this}} \) \( \mapsto \) \{ sel \mapsto \hat{s} \) * \( \text{obj} \mapsto \hat{o} \} \)

Concrete State
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

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

Concrete State

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

Symbolic State

\[ \tilde{H} \]

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

Concrete State

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

This diagram illustrates the state transitions and interactions between the objects and the heap in the context of summarization.
Safely **summarize** storage that is not immediately type inconsistent

Symbolic State

\[ \tilde{H} \]

okheap

Concrete State

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

heap

this.obj

this.sel
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

Symbolic State

\[ \tilde{H} \]

okheap

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

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

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

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

Symbolic State

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>SCRrecorder</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>Aidiom</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**

---

**Note:** The table above lists the size in LOC (lines of code) for each benchmark, the number of symbolic sections, and the maximum number of materializations.
## 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>

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>OMNI_FRAMEWORKS</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

**Maximum number of simultaneous materialized storage locations**

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIHTTPR</td>
<td>14620</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>EKORDER</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>IP</td>
<td>3301</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KIT</td>
<td>5289</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>OMNIFRAMEWORKS</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>SCR</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td></td>
<td></td>
</tr>
<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>combined</td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>SCRRecorder</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>Adiум</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>125</strong></td>
<td><strong>2</strong></td>
<td></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>(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>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (−29%)</td>
</tr>
</tbody>
</table>
## 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>

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>false alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td>reflective call sites</td>
</tr>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td>1327</td>
</tr>
</tbody>
</table>

**Baseline:** standard, **flow-insensitive** type analysis – no switching
<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>SCRECORER</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>

**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>(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>2 (-71%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2716</td>
<td>12</td>
<td>0 (-100%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3301</td>
<td>28</td>
<td>0 (-)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5289</td>
<td>40</td>
<td>1 (-75%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14620</td>
<td>68</td>
<td>10 (-80%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>160769</td>
<td>192</td>
<td>74 (-10%)</td>
<td></td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>38 (-36%)</td>
<td></td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43 (-0%)</td>
<td></td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>70 (-20%)</td>
<td></td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>238 (-29%)</td>
<td></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>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>
## Cost: Analysis time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (s)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28</td>
<td>10.8</td>
</tr>
<tr>
<td>ZIPKit</td>
<td>3301</td>
<td>0.10</td>
<td>33.0</td>
</tr>
<tr>
<td>Server</td>
<td>5289</td>
<td>0.67</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>14620</td>
<td>0.50</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>160769</td>
<td>4.25</td>
<td>37.8</td>
</tr>
<tr>
<td></td>
<td>37327</td>
<td>2.79</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09</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>(loc)</th>
<th>Time</th>
<th>Rate</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>IP</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>KIT</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>S PARKLE</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>ASI HTTP</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>ERQUEST</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>OMNI</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td>ADIUM</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</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
## 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>OmniFramework</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**
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.

**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.

**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 if 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

"foregate"

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.

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

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.

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

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 $$\sigma = \sigma_1 \cup \sigma_2$$ 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**.

\[ \sigma \models_{V} M_1 \triangleleft M_2 \iff \sigma = \sigma_1 \cup \sigma_2 \text{ for some } \sigma_1, \sigma_2 \]
\[ \text{where } \sigma_1 \models_{V} M_1 \text{ and } \sigma_2 \models_{V} M_2 \]
\[ \text{and } \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \]
\[ \text{and } \text{rng}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset \]
Next Steps: Gated Separation

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

$$\sigma \models_V M_1 \triangleleft M_2$$ iff $$\sigma = \sigma_1 \cup \sigma_2$$ 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.

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

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

Slightly stronger than \( \ast \) :

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

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

"foregate"

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

"foregate"

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

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

```
<table>
<thead>
<tr>
<th>s</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>this.obj</td>
</tr>
</tbody>
</table>
```

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

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

Gamma symbol:

- **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.

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

<table>
<thead>
<tr>
<th>Time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.30s</td>
<td>4.1</td>
</tr>
<tr>
<td>0.28s</td>
<td>9.8</td>
</tr>
<tr>
<td>0.10s</td>
<td>33.8</td>
</tr>
<tr>
<td>0.78s</td>
<td>6.8</td>
</tr>
<tr>
<td>0.15s</td>
<td>90.2</td>
</tr>
<tr>
<td>4.61s</td>
<td>34.9</td>
</tr>
<tr>
<td>2.57s</td>
<td>14.5</td>
</tr>
<tr>
<td>2.55s</td>
<td>23.6</td>
</tr>
<tr>
<td>7.50s</td>
<td>23.6</td>
</tr>
<tr>
<td>18.83</td>
<td>24.5</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() ... 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
Idiomatic reflection decouples callbacks and avoids boilerplate
application code

class MyButton {
  var cb : Callback = ...

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

  def draw()
    cb.call()
  end

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

library code

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