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

National Taiwan University 國立臺灣大學
August 1, 2014
Lab: Program analysis in the whole bug mitigation process
Lab: **Program analysis in the whole bug mitigation process**

![Diagram showing program analysis process]

- **Program**
- **Verifier**
- **Alarm Report**
  - Green checkmark: *proof of no bug*
  - Red X: *Alarm Report*

The diagram illustrates the process of program analysis within the bug mitigation process, highlighting the role of the verifier in confirming the absence of bugs.
Lab: **Program analysis** in the whole bug mitigation **process**

- **Verifier**
  - Program
  - Manual Triaging
  - Alarm Report
    - proof of no bug
    - alarm
      - Report
  - ✔ proof of no bug

Diagram:
- Program
- Verifier
- Manual Triaging
- Alarm Report
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis in the whole bug mitigation process**

Verifier

Program

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**

- **Verifier**
  - **Thresher**: Assisting Triage by Refutation Analysis
    - [Blackshears PLDI’13, Blackshears SAS’11, under review]
  - **proof of no bug**
  - **Alarm Report**

- **Manual Triaging**

- **Program**

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

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

- **Program-**
  - Programming

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

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

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

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

- **Verifier**

- **Manual Triaging**

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

**Enforcement**

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

**Programming**

**Speciﬁcation**

**Test Input**

**Program**

**Verifier**

**Runner**

**Test Output**

**Manual Triaging**

**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**
  - **Test Input**
  - **Specification**
    - **Program**
  - **Runner**
    - **Verifier**
      - **Spec**
      - **ification**
        - **Program**
      - **Thresher**: Assisting Triage by Refutation Analysis
        - [Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]
      - **Alarm Report**
        - proof of no bug
      - [Static Incrementalization of Data Structure Checks
        [under review]]

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

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

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

**Programming**

**Test Input**

**Speciﬁcation**

**Program**

**Verifier**

**Runnner**

Static Incrementalization of Data Structure Checks
[under review]

**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**

**Test Output**

**Alarm Report**

✔ proof of no bug

✗ Alarm Report

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

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

**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]

Static Incrementalization of Data Structure Checks
[under review]
Lab: **Program analysis in the whole bug mitigation process**

**This Talk**

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

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

- **Programming**

- **Verification**
  - **Verifier**
    - **Runners**
    - **Test Input**
    - **Test Output**

- **Manual Triaging**

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

- **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]
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

National Taiwan University 國立臺灣大學
August 1, 2014
Fissile Type Analysis:

Types and Separation Logic, Better Together

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

National Taiwan University 國立臺灣大學
August 1, 2014
How to **type check** a program that is **almost** well-typed?
In this talk

Example property of interest: safety of reflective method calls

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

class Callback
  var sel : Str
  var obj : Obj

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

class Callback
    var sel : Str
    var obj : Obj

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

class Callback
  var sel : Str
  var obj : Obj

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

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

    def call():
        this.obj.[this.sel]()   # obj must “respond to” sel
```

Ensure reflection safety with dependent-refinement type expressing required relationship

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

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

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

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

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**

```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*
Similar relationship for array bounds safety

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

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

idx must be a valid index into buf

Guarantees no "ArrayOutOfBounds" error
Similar relationship for array bounds safety

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
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]()
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]()

Field type says: obj must always respond to sel

 guarantted to respond to s

Type error: old obj may not respond to new 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

```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]()
```

<table>
<thead>
<tr>
<th>Two styles of reasoning to determine false alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Callback</td>
</tr>
<tr>
<td>var sel : Str</td>
</tr>
<tr>
<td>var obj : Obj</td>
</tr>
<tr>
<td>def update(s : Str, o : Obj</td>
</tr>
<tr>
<td>this.sel = s</td>
</tr>
<tr>
<td>this.obj = o</td>
</tr>
<tr>
<td>def call()</td>
</tr>
<tr>
<td>this.obj.<a href="">this.sel</a></td>
</tr>
</tbody>
</table>
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

```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

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

class Callback
    var sel : Str
    var obj : Obj

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

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

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

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

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

Reasoning about effects of imperative updates

Relationship violated

Relationship restored

Reasoning by global invariant: call safe if relationship holds
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.

\[ \Gamma \vdash \cdots \]

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

type analysis

types 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

- **type analysis**
  - **types violated**
  - **symbolic flow analysis**
  - **types restored**
  - **types violated**

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

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

Switch to symbolic analysis when global type invariant violated

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

Switch to symbolic analysis when global type invariant violated

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

Switch to symbolic analysis when global type invariant violated

Back to types when invariant restored

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

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

Effective when global type invariant holds most of the time

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

Effective when global type invariant holds most of the time

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

- 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
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

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

Goal: keep **types** as simple as possible
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 environment.
   - Reason precisely only when type invariant violated.

2. Leverage heap type invariant during symbolic analysis via type-consistent materialization and summary.
   - 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 annotation
   - 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.

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

**Type environment**

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

\[ \Gamma \]

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

**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

\[ \Gamma \]

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

Symbolic state

\[ \tilde{\Gamma} \]

\[ \tilde{E} \]

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

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

Type environment

Maps local variables to dependent types

\[ \Gamma \]

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

Symbolic state

\[ \tilde{\Gamma} \]

\[ \tilde{E} \]

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

Refinements refer to variables
**Type environment**

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

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

**Symbolic state**

Maps local **variables** to **symbolic values**

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

**Symbolization**

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

**Example Code**

```java
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

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

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

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

**Refinements** refer to variables

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

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

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

\[ \Gamma \]

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

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

\[ \tilde{\Gamma} \]

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

**Symbolize**

**Refinements refer to variables**

**Maps symbolic values to dependent types 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
```

Γ

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

A type environment constrains local variables

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

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

![Diagram showing the relationship between local variables and heap elements](image)
Symbolization allows local variables to hold values inconsistent with declared types

A type environment constrains local variables

\[ \Gamma \]

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

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

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

A type environment constrains local variables:

\[ \Gamma \]

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

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

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

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

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

Symbolize:

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

declare variables


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

symbolize

heap

this.obj

this.sel
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

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

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

\[ \Gamma \]

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

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

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

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

Heap

\[ \text{this.obj} \]
\[ \text{this.sel} \]
Symbolization allows local variables to hold values inconsistent with declared types.

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

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

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

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.

\[ \Gamma, \mathit{this} : \text{Callback} \]

\[ \Gamma, o : \text{Obj} | r2 \sim s \]

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

\[ \tilde{\text{Callback}} \]
Symbolization unpacks local cells, but symbolic facts about values still constrain the heap.

\[ \Gamma \vdash \text{Callback} \supseteq \{ \text{sel : Str, obj : Obj | r2 sel} \} \]

Base types same on both sides

Symbolic fact map

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

Base type field refinements still refer to fields

Type environment

\[ \Gamma \]

\begin{align*}
\text{s : Str} \\
\text{o : Obj} \\
\text{this : Callback}
\end{align*}

\[ \tilde{\Gamma} \]

\begin{align*}
\tilde{\text{s : Str}} \\
\tilde{\text{o : Obj}} \\
\tilde{\text{t : Callback}}
\end{align*}
Summarize heap locations that are **not** immediately type-inconsistent with \textit{okheap}

### Symbolic Heap

\[ \tilde{H} \]

\textit{okheap}

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

### Concrete State

\[ s \]  \[ o \]  \[ \text{this} \]
Summarize heap locations that are **not** immediately type-inconsistent with okheap

**Symbolic Heap**

\[ \tilde{H} \]

**Concrete State**

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

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

Symbolic Heap

\[ \tilde{H} \]

okheap

Describes storage without explicitly enumerating it

Concrete State

```
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

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

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

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

\[
\begin{align*}
\text{this.obj} &
\end{align*}
\]

\[
\begin{align*}
\text{this.sel} &
\end{align*}
\]
Key Contributions

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

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

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

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

Symbolic State

\[ \tilde{H} \]

Concrete State

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

Materialize onto standard separation-logic explicit heap

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

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

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

**Concrete State**

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

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

Materialize onto standard separation-logic explicit heap

\[ \text{okheap } \star \text{this} \mapsto \{ \text{sel }\mapsto \text{sel} \star \text{obj }\mapsto \text{obj} \} \]

Must-alias and disalias guarantee requires case split on materialization

Concrete State

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

Materialize onto standard separation-logic explicit heap

\[ \hat{H} \]
\[ \text{okheap} \times \hat{\text{this}} \mapsto \{ \text{sel} \mapsto \hat{\text{sel}} \ast \text{obj} \mapsto \hat{\text{obj}} \} \]

Must-alias and disalias guarantee requires case split on materialization

Value stored in obj responds to value stored in sel

Concrete State

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

\[ \text{s} \quad \text{o} \quad \text{this} \]
Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard separation-logic explicit heap

\[ \widetilde{H} \]

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

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

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

Concrete State

Represent materialized storage with

\[ \begin{align*}
    \text{s} & \\
    \text{o} & \\
    \text{this} & \\
\end{align*} \]
Leverage heap type invariant via type-consistent materialization

Materialize onto standard separation-logic explicit heap

Materialized storage guaranteed to be not immediately type-inconsistent

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

Must-alias and disalias guarantee requires case split on materialization

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

Concrete State

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

Symbolic State

\[ \tilde{H} \]

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

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

\[ \tilde{H} \]

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

Concrete State

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

\[
\text{this.sel}
\]

\[
\text{this.obj}
\]
Strong updates on materialized storage to detect invariant restoration

Symbolic State

$$\tilde{H}$$

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

Concrete State

```
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

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

Symbolic State

\[ \widetilde{H} \]

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

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

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

Symbolic State

\[ \tilde{H} \]

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

Concrete State

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

\( \text{Heap} \)

this.obj \quad this.sel
Strong updates on materialized storage to detect invariant restoration

Symbolic State

$$\tilde{H}$$

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

Concrete State

No longer immediately type-inconsistent

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

\[ \tilde{H} \]

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

Symbolic State

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

Concrete State

Symbolic State

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

\[
\tilde{H} \ 	ext{okheap}
\]

\[
\text{def update}(s: \text{Str}, o: \text{Obj} | r2 \ s)
\]
\[
\text{this.sel} = s
\]
\[
\text{this.obj} = o
\]
Safely summarize storage that is not immediately type inconsistent

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

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
```

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

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

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

2. **Leverages** 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>OmniFramework</td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td>461080</td>
<td><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>Vienne</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</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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>benchmark</td>
<td>size</td>
<td>symbolic sections</td>
<td>maximum materializations</td>
</tr>
<tr>
<td></td>
<td>(loc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Maximum number of simultaneous materialized storage locations**

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

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

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

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

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

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

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

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

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

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</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>2743</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>CoreDNS</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>Skirm</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23.0</strong></td>
</tr>
</tbody>
</table>

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

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

*Does not include system headers*

*Includes analysis time but not parsing, base type checking*
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>SRecord 3 on E</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>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>OMNI FRAMEWORKS</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23.0</strong></td>
</tr>
</tbody>
</table>

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

**Interactive speeds**
## Cost: Analysis time

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

![Diagram showing type-consistent state transition]
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.”

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

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

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

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.

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

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

```
s  o
this
```

```
heap
this.obj
this.sel
```

```
priv
```

```
s  o
this
```

```
heap
this.obj
this.sel
```

```
priv
```

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

"foregate"*

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**.

\[
\sigma \models_{V} M_{1} \triangleleft M_{2} \quad \text{iff} \quad \sigma = \sigma_{1} \cup \sigma_{2} \text{ for some } \sigma_{1}, \sigma_{2} \\
\text{where } \sigma_{1} \models_{V} M_{1} \text{ and } \sigma_{2} \models_{V} M_{2} \\
\text{and } \text{dom}(\sigma_{1}) \cap \text{dom}(\sigma_{2}) = \emptyset \\
\text{and } \text{rng}(\sigma_{1}) \cap \text{dom}(\sigma_{2}) = \emptyset
\]

Slightly stronger than \( \star \): No **direct** pointers from "foregate" to "aftgate"
Next Steps: Type-Intertwined Framing

“foregate”

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

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

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

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

symbolic flow analysis

type analysis
Next Steps: Type–Intertwined Framing

“foregate”

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

```
\[ \Gamma \]

\[ s \quad o \quad this \]

This obj  This sel

Heap

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

"foregate"

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.

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size</th>
<th>false alarms</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(loc)</td>
<td>reflective call sites</td>
<td>annotation count</td>
</tr>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>9</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>0</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>49</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>24</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>7</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>40</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>12</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
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
Idiomatic reflection decouples callbacks and avoids boilerplate
class MyButton {
    var cb : Callback = ...
    
    def setState(s : Str)
        var m = "draw" + s
        cb.update(self, m)
    end

    def draw()
        cb.call()
    end

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

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

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

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

Idiomatic reflection decouples callbacks and avoids boilerplate