Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin    Bor-Yuh Evan Chang

University of Colorado Boulder
How to **type check** a program that is **almost** well-typed?
In this talk

Example property of interest:

\texttt{safety of reflective method calls}

Type system:

\texttt{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]()

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

Run time error if obj does not respond to sel — i.e., method does not exist
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**

```java
class Callback {
  var sel : Str
  var obj : Obj | r2 sel

  def call() {
    this.obj.[this.sel]()  // obj must “respond to” sel
  }
}
```

**Shorthand** for `obj :: {ν : Obj | ν r2 sel}`

---

5
Ensure reflection safety with **dependent refinement type** expressing required **relationship**

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

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

**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[] | indexedBy idx

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

*idx* must be a valid index into *buf*.
Similar relationship for **array bounds** safety

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

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

**idx** must be a valid index into **buf**

 Guarantees **no** “ArrayOutofBounds” **error**
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

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

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

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

def call()
  this.obj.[this.sel]()
Updating relationship causes type error

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

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

def call()
    this.obj.[this.sel]()
**Updating relationship causes type error**

```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]()"
```
**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

○ guaranteed 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

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 \texttt{reasoning} to determine \texttt{false alarm}

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

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

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

Reasoning by global \texttt{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]()

Reasoning by global invariant: call safe if relationship holds
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

class Callback
    var sel : Str
    var obj : Obj

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

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

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

Automated reasoning about global invariants

\[ \Gamma \vdash \cdots \]

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

Automated reasoning about global invariants

Automated reasoning about execution

\[ \Gamma \vdash \cdots \]

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

Automated reasoning about global invariants

\[ \Gamma \vdash \cdots \]

Flow-Insensitive Type Systems

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

Abstract Interpretation/Flow Analysis

Automated reasoning about execution
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 **violated**

- **type analysis**
  - types **restored**

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

- **type analysis**
  - types **violated**

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

- **type analysis**
  - types **violated**
  - **Switch** to symbolic analysis when global type **invariant violated**
  - types **restored**
  - **Back** to types when invariant **restored**

- **symbolic flow analysis**
  - types **violated**
  - types **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

**Type analysis**

- Types **violated**
- Types **restored**

**Symbolic flow analysis**

- Type analysis
  - Types **violated**
  - Types **restored**

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

- **Type analysis**
  - Types **violated**
  - Types **restored**

- **Symbolic flow analysis**
  - Types **violated**
  - Types **restored**

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

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

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

- **Type analysis**
  - Types violated
  - Effective when **global** type invariant holds most of the time
  
  - **Symbolic flow analysis**
  - Types restored
  - Relationship updates
  - Occurrence typing
  - Tagged unions
Play to the **strengths** of each intertwined **analysis**
Play to the **strengths** of each intertwined analysis

**Flow-Insensitive Types**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1. **Translate** type invariant into symbolic state via "symbolization" of type environment.
   - 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 summarization. Reason precisely only for locations where type invariant violated.
Key Contributions

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

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

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

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

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

```
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**

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

**Symbolic state**

\[
\text{def update}(s: \tilde{s}, o: \tilde{o}) \\
\text{this}.sel = \tilde{s} \\
\text{this}.obj = \tilde{o}
\]

**Symbolize**

**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**
- Maps local variables to symbolic values

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

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*

\[ \Gamma \]

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

**Symbolic state**

Maps local variables to *symbolic values*

\[ \bar{E} \]

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

**symbolize**

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

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

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

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

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

**Symbolization**
- Lifted to symbolic values (symbolic facts)

- Maps symbolic values to dependent types
- Refinements refer to variables
Symbolization allows local variables to hold values inconsistent with declared types

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

Γ

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

A type environment constrains local variables

$\Gamma$

$s : \text{Str}$

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

this : Callback

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

A type environment constrains local variables

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

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

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

A type environment constrains local variables

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

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

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

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

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

\[ \Gamma \xrightarrow{s : \text{Str}} \xrightarrow{o : \text{Obj | r2 s}} \text{symbolize} \xrightarrow{\tilde{E}} \tilde{\Gamma} \]

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

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

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

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

Symbolic environment allows, e.g., int in

\[ \Gamma \]

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

symbolize

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

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

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

Symbolic environment allows, e.g., int in

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

Symbolic environment

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

```
\[ \widetilde{E} \quad \widetilde{\Gamma} \]
```

Heap

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

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

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

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

Grey indicates storage that is not immediately type-inconsistent.
Symbolization **unpacks local cells**, but **symbolic facts** about **values** still **constrain the heap**

**Type environment**

\[ \Gamma \]

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

**Symbolic fact map**

\[ \tilde{\Gamma} \]

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

**Diagram**

- **S**
- **O**
- **this**

**Heap**

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

Type environment

\[ \Gamma \]

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

**Base types same on both sides**

Symbolic fact map

\[ \tilde{\Gamma} \]

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

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**

\[
\Gamma \quad \text{Callback} \quad \text{symbolize} \quad \hat{\Gamma}
\]

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

\[
\hat{\Gamma} \quad \hat{s} : \text{Str} \\
\quad \hat{o} : \text{Obj} \mid r^2 \hat{s} \\
\quad \hat{t} \quad \text{Callback}
\]

**Callback** \( \triangleq \{ \text{sel} : \text{Str}, \text{obj} : \text{Obj} \mid r^2 \text{sel} \} \)

**Base type field refinements still refer to fields**

**Base types same on both sides**

**Symbolic fact map**

heap

this.obj

this.sel

S

O

this
Summarize heap locations that are \textit{not} immediately type-inconsistent with \textit{okheap}

Symbolic Heap

\[ \tilde{H} \]

\[ \text{okheap} \]

Concrete State

\[
\text{def update}(s:\text{Str}, o:\text{Obj} \mid r2\ s) \\
\quad \text{this.sel} = s \\
\quad \text{this.obj} = o
\]
Summarize heap locations that are not immediately type-inconsistent with okheap

Symbolic Heap

$$\tilde{H}$$

okheap

Concrete State

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

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

Symbolic Heap

\[ \tilde{H} \]

`okheap`

Describes storage **without** explicitly enumerating it

Concrete State

```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 \textbf{not} immediately type-inconsistent with okheap

Symbolic Heap

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

\( \overset{\sim}{H} \) okheap

Describes storage \textbf{without} explicitly enumerating it

Concrete State

\begin{verbatim}
def update(s:Str, o:Obj | r2 s):
    this.sel = s
    this.obj = o
\end{verbatim}

Formula literal: \textit{concretization} includes every subheap that is \textbf{not} immediately type inconsistent

Immediately after switch, type invariants still hold so \textit{okheap} represents \textbf{entire heap}
Key Contributions

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

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

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

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

**Symbolic State**

\[ \tilde{H} \] okheap

**Concrete State**

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

.heap

\( \text{this.obj} \quad \text{this.sel} \)
Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard **separation-logic** explicit heap

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

```
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**

$$\sim H$$

$$\text{def update}(s:\text{Str}, o:\text{Obj} | r2 s)$$

$$\text{this}.\text{sel} = s$$

$$\text{this}.\text{obj} = o$$

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

**Concrete State**

- $s$
- $o$
- `this`
- Heap with `this.sel`, `this.obj`
Leverage **heap type invariant** via **type-consistent materialization**

**Materialize onto standard separation-logic explicit heap**

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

**Materialized storage guaranteed to be not immediately type-inconsistent**

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

**Concrete State**
Leverage **heap type invariant** via **type-consistent materialization**

**Concrete State**

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

Materialize onto standard separation-logic explicit heap

$\widetilde{H}$

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

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

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

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

**Concrete State**

Materialize onto standard **separation-logic** explicit heap

\[ \tilde{H} \]

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

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

Materialized storage guaranteed to be **not immediately type-inconsistent**

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

Represent **materialized** storage with **cutout**

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

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

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

\[ \text{if} \]

\[ \text{sel} \]

\[ \text{f} \]

\[ \text{obj} \]

\[ \text{\(\tilde{7}\!)\} \}

\[ \text{sel} \]

\[ \text{\(\tilde{7}\!) \mapsto \}

\[ \text{obj} \]

\[ \text{\(\tilde{7}\!}\}

\[ \text{Value stored in } \text{obj} \text{ responds to value stored in } \text{sel} \]
Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard separation-logic explicit heap

\[ \widetilde{H} \]

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

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

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

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

Symbolic State

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

Concrete State

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

![Concrete State Diagram]
Strong updates on materialized storage to detect invariant restoration

Symbolic State

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

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

Concrete State

Symbolic State

\[ \hat{H} \]

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

Type invariant violated

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

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

**Symbolic State**

\[ \widetilde{H} \]

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

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

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

Symbolic State

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

Concrete State

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

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

![Symbolic State Diagram](image)

\[ \tilde{H} \]

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

![Concrete State Diagram](image)

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

Symbolic State

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

Concrete State

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

S

O

this

heap

this.obj

this.sel
Safely **summarize** storage that is not immediately type inconsistent

**Symbolic State**

\[ \tilde{H} \]

\[ \text{okheap} \]

**Concrete State**

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

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

**Symbolic State**

\[ \tilde{H} \]

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

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

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

**Concrete State**

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

**Symbolic State**

\[ \hat{H} \]

\( \text{okheap} \)

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

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

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

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

Number of **successful** switches to **symbolic analysis and back**

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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size (loc)</th>
<th>Symbolic sections</th>
<th>Maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>

**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</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><strong>combined</strong></td>
<td>461080</td>
<td>125</td>
<td>2</td>
</tr>
<tr>
<td><strong>OmniFrameWorks</strong></td>
<td>160769</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td><strong>Vienna</strong></td>
<td>37327</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td><strong>Skim</strong></td>
<td>60211</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Adium</strong></td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum number of **simultaneous** materialized storage locations

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

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>symbolic sections</th>
<th>maximum materializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>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><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>APU</td>
<td>176629</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td><strong>Combined</strong></td>
<td><strong>125</strong></td>
<td></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

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

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>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>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (-29%)</td>
</tr>
</tbody>
</table>

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

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

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

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

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

Almost everywhere techniques show 29% improvement in false alarms
## 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>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>Scrapy</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>SFlm</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>FFile</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Skim</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>ADIUM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>20.09s</td>
<td>23.0</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time</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**.
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