Improving Flow Analyses via $\Gamma CFA$

Abstract Garbage Collection and Counting

Matthew Might*    Olin Shivers†

*Georgia Institute of Technology
†Northeastern University

ICFP 2006
The Big Idea

The Process

1. Add garbage collection to a concrete semantics.
The Big Idea

The Process

1. Add garbage collection to a concrete semantics.
2. Create an abstract interpretation of these semantics.
The Big Idea

The Process
1. Add garbage collection to a concrete semantics.
2. Create an abstract interpretation of these semantics.

The Payoff
The abstract GC improves both speed and precision.
The Problem: Imprecision in Abstract Interpretation

Concrete Space

Concrete Space

Abstract Space

Abstract Space

Abstract Interpretation

Larger space mapped to smaller space: **Overlap leads to imprecision.**
An Example: Analyzing Integer Arithmetic

Goal

Given an arithmetic expression, safely approximate its sign.
An Example: Analyzing Integer Arithmetic

Goal

Given an arithmetic expression, safely approximate its sign.

Example

- 4 + 3 could be positive.
- 4 - 10 could be negative.
- 4 + (3 - 10) could be positive or negative. (Imprecision allowed.)
An Example: Analyzing Integer Arithmetic

Abstracting the Integers

Integers abstract to a singleton set of their sign.

Example

- $|4| = \{positive\}$
- $|0| = \{zero\}$
- $|-3| = \{negative\}$
An Example: Analyzing Integer Arithmetic

Abstracting Addition

Addition abstracts “naturally” to sets of signs.

Example

- \{ \text{positive} \} \oplus \{ \text{positive} \} = \{ \text{positive} \}
- \{ \text{positive}, \text{negative} \} \oplus \{ \text{zero} \} = \{ \text{positive}, \text{negative} \}
- \{ \text{positive} \} \oplus \{ \text{negative} \} = \{ \text{negative}, \text{zero}, \text{positive} \}
An Example: Analyzing Integer Arithmetic

Example

Analyze: $-4 + 4$
An Example: Analyzing Integer Arithmetic

Example

Analyze: $-4 + 4$

$\Rightarrow$ $|{-4}| \oplus |4|$
An Example: Analyzing Integer Arithmetic

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| Analyze: $-4 + 4$  
| $\Rightarrow$ $|\neg 4| \oplus |4|$  
| $\Rightarrow$ $\{negative\} \oplus \{positive\}$ |
An Example: Analyzing Integer Arithmetic

Example

Analyze: $-4 + 4$

$\Rightarrow \quad |-4| \oplus |4|$

$\Rightarrow \quad \{negative\} \oplus \{positive\}$

$\Rightarrow \quad \{negative, zero, positive\}$

Imprecision!

$\{zero\}$ is the tightest, safest answer.
Flow analysis question: *What “values” could flow to each expression?*

(let* ((id (λ (x) x))
       (y (id 3)))
  (id 4))

0CFA thinks...

\[\text{id} \mapsto
\]
\[\text{x} \mapsto
\]
\[(\text{id} \; 3) \mapsto
\]
\[\text{y} \mapsto
\]
\[(\text{id} \; 4) \mapsto\]
0CFA & Precision

Flow analysis question: What “values” could flow to each expression?

(let* ((id (\(x\) x))
     (y (id 3)))
(id 4))

0CFA thinks...

1. \((\lambda (x) x)\) flows to id.

\[
\begin{align*}
\text{id} & \mapsto (\lambda (x) x) \\
\text{x} & \mapsto \\
\text{y} & \mapsto \\
\text{id 4} & \mapsto \\
\end{align*}
\]
Flow analysis question: What “values” could flow to each expression?

(let* ((id (\(x\) x))
        (y (id 3)))
    (id 4))

0CFA thinks...

1. \((\lambda (x) x)\) flows to id.

2. Then, 3 flows to x.

\[ \begin{align*}
\text{id} & \mapsto (\lambda (x) x) \\
\text{x} & \mapsto 3 \\
\text{id 3} & \mapsto \\
\text{y} & \mapsto \\
\text{id 4} & \mapsto 
\end{align*} \]
Flow analysis question: *What “values” could flow to each expression?*

(\(\text{let*} ((\text{id} (\lambda (x) x))\) \\
  (\(y \ (\text{id} \ 3))\))

(\(\text{id} \ 4))

\(\text{0CFA thinks...}\)

1. \((\lambda (x) x)\) flows to \(\text{id}\).

   \(\text{id} \mapsto (\lambda (x) x)\)

2. Then, 3 flows to \(x\).

   \(x \mapsto 3\)

3. Then, 3 flows to \(y, (\text{id} \ 3)\).

   \((\text{id} \ 3) \mapsto 3\)

   \(y \mapsto 3\)

   \((\text{id} \ 4) \mapsto\)
Flow analysis question: *What “values” could flow to each expression?*

(let* ((id (λ (x) x))
       (y (id 3)))
  (id 4))

0CFA thinks...

1. $(\lambda (x) x)$ flows to $id$.  
   $id \mapsto (\lambda (x) x)$
2. Then, 3 flows to $x$.  
   $x \mapsto 3, 4$
3. Then, 3 flows to $y$, (id 3).  
   $(id 3) \mapsto 3$
4. Then, 4 flows to $x$.  
   $y \mapsto 3$
   $(id 4) \mapsto 4$
Flow analysis question: *What “values” could flow to each expression?*

(let* ((id (λ (x) x))
       (y (id 3)))
     (id 4))

0CFA thinks...

1. (λ (x) x) flows to id.

2. Then, 3 flows to x.

3. Then, 3 flows to y, (id 3).

4. Then, 4 flows to x.

5. Then, 3 or 4 could flow to (id 4)!?

---

**Problem**

Flow analyses overlap different bindings to the same variable.
Flow analysis question: *What “values” could flow to each expression?*

\[(\text{let* }((\text{id } (\lambda (x) x))
  (y (\text{id } 3)))
  (\text{id } 4))\]

0CFA thinks...

1. \((\lambda (x) x)\) flows to \(\text{id}\).
2. Then, \(3\) flows to \(x\).
3. Then, \(3\) flows to \(y, (\text{id } 3)\).
4. Then, \(4\) flows to \(x\).
5. Then, \(3\ or\ 4\) could flow to \((\text{id } 4)\)?

**Solution**

Garbage collect dead bindings mid-analysis.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.

```
<table>
<thead>
<tr>
<th>concrete</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>o_1</td>
<td>o_1</td>
</tr>
<tr>
<td>a_1</td>
<td>a_1,2</td>
</tr>
<tr>
<td>a_2</td>
<td></td>
</tr>
<tr>
<td>a_3</td>
<td>a_3</td>
</tr>
</tbody>
</table>
```

Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.
Example: Abstract Garbage Collection

3-address concrete heap. 2-address abstract counterpart.

```
concrete

abstract

o1 ← a1

a2 → ˆa1,2

a3 → ˆa3

| o1 |

o1
```


Example: Abstract Garbage Collection

Next: Allocate object $o_2$ to address $a_3$. Shift root to $a_3$. 
Example: Abstract Garbage Collection

Next: Allocate object $o_3$ to address $a_2$. Point $o_2$ to $a_2$. 
Example: Abstract Garbage Collection


concrete

abstract

\[ o_1 \quad \rightarrow \quad a_1 \quad \rightarrow \quad \hat{a}_{1,2} \quad \rightarrow \quad \hat{a}_3 \quad \rightarrow \quad o_2 \]

\[ o_3 \quad \rightarrow \quad a_2 \quad \rightarrow \quad \hat{a}_{1,2} \quad \rightarrow \quad \hat{a}_3 \quad \rightarrow \quad o_3 \]
Example: Abstract Garbage Collection

Solution: Rewind and garbage collect first.

concrete

abstract

\[ o_1 \rightarrow a_1 \rightarrow o_1 \]

\[ o_3 \rightarrow a_2 \rightarrow o_3 \]

\[ o_2 \rightarrow a_3 \rightarrow o_2 \]

\[ \hat{a}_{1,2} \rightarrow \hat{a}_3 \rightarrow \hat{a}_{1,2} \]
Example: Abstract Garbage Collection

As it was:

```
Example: Abstract Garbage Collection
As it was:
```

```
concrete

abstract

\( o_1 \)

\( a_1 \)

\( o_1 \)

\( a_2 \)

\( a_{1,2} \)

\( a_3 \)

\( o_2 \)

\( a_{3} \)

\( o_2 \)

```
Example: Abstract Garbage Collection

After garbage collection:

concrete

abstract

\[ a_1 \]

\[ a_2 \]

\[ \hat{a}_{1,2} \]

\[ o_2 \]

\[ a_3 \]

\[ \hat{a}_3 \]

\[ |o_2| \]
Example: Abstract Garbage Collection

Try again: Allocate object \( o_3 \) to address \( a_2 \). Point \( o_2 \) to \( a_2 \).
Example: Abstract Garbage Collection

No overapproximation!
Implementation: ΓCFA
Tool: Continuation-Passing Style (CPS)

Contract

- Calls don’t return.
- Continuations are passed—to receive return values.

CPS $\lambda$-calculus

\[
\begin{align*}
e, f \in EXP & ::= v \\
& \quad \mid (\lambda (v_1 \cdots v_n) \text{ call}) \\
call \in CALL & ::= (f \ e_1 \cdots e_n)
\end{align*}
\]
CPS Narrows Concern

\( \lambda \) is universal representation of control & env.

<table>
<thead>
<tr>
<th>Construct</th>
<th>encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun call</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>fun return</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>iteration</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>sequencing</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>conditional</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>exception</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>coroutine</td>
<td>call to ( \lambda )</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Advantage

Now \( \lambda \) is fine-grained construct.
### Eval-to-Apply Transition

\[
\text{proc} = A(f, \beta, ve) \quad d_i = A(e_i, \beta, ve) \\
\left(\left\llbracket (f \ e_1 \cdots e_n) \right\rrbracket, \beta, ve, t \right) \Rightarrow (\text{proc}, d, ve, t + 1)
\]

### Apply-to-Eval Transition

\[
\text{proc} = \left(\left\llbracket (\lambda (v_1 \cdots v_n) \text{ call}) \right\rrbracket, \beta' \right) \\
(\text{proc}, d, ve, t) \Rightarrow (\text{call}, \beta'[v_i \mapsto t], ve[(v_i, t) \mapsto d_i], t)
\]

### Domains

\[
\begin{align*}
\varsigma & \in \text{Eval} = \text{CALL} \times \text{BEnv} \times \text{VEnv} \times \text{Time} \\
+ \text{Apply} & = \text{Proc} \times \text{D}^* \times \text{VEnv} \times \text{Time} \\
\beta & \in \text{BEnv} = \text{VAR} \rightarrow \text{Time} \\
ve & \in \text{VEnv} = \text{VAR} \times \text{Time} \rightarrow \text{D} \\
\text{proc} & \in \text{Proc} = \text{Clo} + \{\text{halt}\} \\
\text{clo} & \in \text{Clo} = \text{LAM} \times \text{BEnv} \\
d & \in \text{D} = \text{Proc} \\
t & \in \text{Time} = \text{infinite set of times (contours)}
\end{align*}
\]

### Lookup function

\[
\begin{align*}
A(lam, \beta, ve) & = (lam, \beta) \\
A(v, \beta, ve) & = ve(v, \beta(v))
\end{align*}
\]
Eval-to-Apply Transition

\[ \hat{\text{proc}} \in \hat{\mathcal{A}}(f, \hat{\beta}, \hat{\nu}) \Rightarrow \hat{d} = \hat{\mathcal{A}}(e_i, \hat{\beta}, \hat{\nu}) \]

Apply-to-Eval Transition

\[ \hat{\text{proc}} = (\llbracket (\lambda (v_1 \cdots v_n) \text{ call}) \rrbracket, \hat{\beta'}) \Rightarrow (\hat{\text{proc}}, \hat{d}, \hat{\nu}, \text{ succ}(\hat{t})) \]

Domains

\[ \hat{\varsigma} \in \hat{\mathcal{E}val} = \text{CALL} \times \hat{\mathcal{B}Env} \times \hat{\mathcal{V}Env} \times \hat{\mathcal{Time}} \]
\[ + \hat{\text{Apply}} = \hat{\mathcal{Proc}} \times \hat{\mathcal{D}^*} \times \hat{\mathcal{V}Env} \times \hat{\mathcal{Time}} \]
\[ \hat{\beta} \in \hat{\mathcal{B}Env} = \text{VAR} \rightarrow \hat{\mathcal{Time}} \]
\[ \hat{\nu} \in \hat{\mathcal{V}Env} = \text{VAR} \times \hat{\mathcal{Time}} \rightarrow \hat{\mathcal{D}} \]
\[ \hat{\text{proc}} \in \hat{\mathcal{Proc}} = \hat{\text{Clo}} + \{\text{halt}\} \]
\[ \hat{\text{clo}} \in \hat{\text{Clo}} = \text{LAM} \times \hat{\mathcal{B}Env} \]
\[ \hat{d} \in \hat{\mathcal{D}} = \mathcal{P}(\hat{\mathcal{Proc}}) \]
\[ \hat{t} \in \hat{\mathcal{Time}} = \text{finite set of times (contours)} \]

Lookup function

\[ \hat{\mathcal{A}}(\hat{\text{lam}}, \hat{\beta}, \hat{\nu}) = \{(\hat{\text{lam}}, \hat{\beta})\} \]
\[ \hat{\mathcal{A}}(\hat{v}, \hat{\beta}, \hat{\nu}) = \hat{\nu}(\hat{v}, \hat{\beta}(\hat{v})) \]
Eval-to-Apply Transition

$$\widehat{\text{proc}} \in \widehat{\mathcal{A}}(f, \hat{\beta}, \hat{ve}) \quad \widehat{d}_i = \widehat{\mathcal{A}}(e_i, \hat{\beta}, \hat{ve})$$

$$((f \ e_1 \cdots e_n],[\hat{\beta}, \hat{ve}, \hat{t}] \Rightarrow (\widehat{\text{proc}}, \widehat{d}, \hat{ve}, \text{succ}(\hat{t})))$$

Apply-to-Eval Transition

$$\widehat{\text{proc}} = ([(\lambda (v_1 \cdots v_n) \ \text{call})], \hat{\beta}')$$

$$(\widehat{\text{proc}}, \widehat{d}, \hat{ve}, \hat{t}) \Rightarrow (\text{call}, \hat{\beta}'[v_i \mapsto \hat{t}], \hat{ve} \sqcup [(v_i, \hat{t}) \mapsto \hat{d}_i], \hat{t})$$

Domains

$$\hat{\varsigma} \in \hat{\text{Eval}} = \text{CALL} \times \hat{\text{BEnv}} \times \hat{\text{VEnv}} \times \hat{\text{Time}}$$
$$+ \hat{\text{Apply}} = \hat{\text{Proc}} \times \hat{\mathcal{D}}^* \times \hat{\text{VEnv}} \times \hat{\text{Time}}$$

$$\hat{\beta} \in \hat{\text{BEnv}} = \text{VAR} \rightarrow \hat{\text{Time}}$$
$$\hat{ve} \in \hat{\text{VEnv}} = \text{VAR} \times \hat{\text{Time}} \rightarrow \hat{\mathcal{D}}$$
$$\hat{\text{proc}} \in \hat{\text{Proc}} = \hat{\text{Clo}} + \{\text{halt}\}$$
$$\hat{\text{clo}} \in \hat{\text{Clo}} = \hat{\text{LAM}} \times \hat{\text{BEnv}}$$
$$\hat{d} \in \hat{\mathcal{D}} = \mathcal{P}(\hat{\text{Proc}})$$
$$\hat{t} \in \hat{\text{Time}} = \text{finite set of times (contours)}$$

Lookup function

$$\hat{\mathcal{A}}(\text{lam}, \hat{\beta}, \hat{ve})$$
$$= \{(\text{lam}, \hat{\beta})\}$$

$$\hat{\mathcal{A}}(v, \hat{\beta}, \hat{ve})$$
$$= \hat{ve}(v, \hat{\beta}(v))$$
Concrete v. Abstract Interpretations

<table>
<thead>
<tr>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concrete: $s_1$</td>
</tr>
<tr>
<td>Abstract: $\hat{s}_1$</td>
</tr>
</tbody>
</table>
Concrete v. Abstract Interpretations

**Interpretation**

Concrete: \( s_1 \rightarrow s_2 \)

Abstract: \( \hat{s}_1 \rightarrow \hat{s}_2 \)
Concrete v. Abstract Interpretations

Concrete: \( s_1 \rightarrow s_2 \rightarrow s_3 \)

Abstract: \( \widehat{s}_1 \rightarrow \widehat{s}_2 \rightarrow \widehat{s}_3 \)
Concrete v. Abstract Interpretations

**Concrete:** $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$

**Abstract:** $\hat{s}_1 \rightarrow \hat{s}_2 \rightarrow \hat{s}_3 \rightarrow \hat{s}_4$

- $\hat{s}_{3.1,1}$
- $\hat{s}_{3.2,1}$
Concrete v. Abstract Interpretations

### Interpretation

Concrete:

\[ s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \]

Abstract:

\[ \hat{s}_1 \rightarrow \hat{s}_2 \rightarrow \hat{s}_3 \rightarrow \hat{s}_4 \]

\[ \hat{s}_{3,1,1} \rightarrow \hat{s}_{3,1,2} \]

\[ \hat{s}_{3,2,1} \]
Technique: Abstract Counting

The Idea

1. Count times an abstract resource has been allocated.
2. Count of one means only one concrete counterpart.
ΓCFA Environment Condition

Basic Principle

If \( \{ x \} = \{ y \} \), then \( x = y \).

Theorem (Environment condition)

If \( \hat{\beta}_1(v) = \hat{\beta}_2(v) \),
and \( \hat{\mu}(v, \hat{\beta}_1(v)) = \hat{\mu}(v, \hat{\beta}_2(v)) = 1 \),
then \( \beta_1(v) = \beta_2(v) \).
Basic Principle

If \( \{ x \} = \{ y \} \), then \( x = y \).

Theorem (Environment condition)

If \( \hat{\beta}_1(v) = \hat{\beta}_2(v) \),
and \( \hat{\mu}(v, \hat{\beta}_1(v)) = \hat{\mu}(v, \hat{\beta}_2(v)) = 1 \),
then \( \beta_1(v) = \beta_2(v) \).
Basic Principle

If \( \{x\} = \{y\} \), then \( x = y \).

Theorem (Environment condition)

If \( \hat{\beta}_1(v) = \hat{\beta}_2(v) \),
and \( \hat{\mu}(v, \hat{\beta}_1(v)) = \hat{\mu}(v, \hat{\beta}_2(v)) = 1 \),
then \( \beta_1(v) = \beta_2(v) \).
ΓCFA Environment Condition

**Basic Principle**

If \( \{x\} = \{y\} \), then \( x = y \).

**Theorem (Environment condition)**

If \( \hat{\beta}_1(v) = \hat{\beta}_2(v) \),
and \( \hat{\mu}(v, \hat{\beta}_1(v)) = \hat{\mu}(v, \hat{\beta}_2(v)) = 1 \),
then \( \beta_1(v) = \beta_2(v) \).
## Basic Principle

If \( \{x\} = \{y\} \), then \( x = y \).

## Theorem (Environment condition)

If \( \hat{\beta}_1(v) = \hat{\beta}_2(v) \),  
and \( \hat{\mu}(v, \hat{\beta}_1(v)) = \hat{\mu}(v, \hat{\beta}_2(v)) = 1 \),  
then \( \beta_1(v) = \beta_2(v) \).

## Increase in Power

Enables the Super-\( \beta \) class of optimizations.
CFA Counting Machinery

Abstract binding counter, $\hat{\mu}$ : “Bindings” $\rightarrow \{0, 1, \infty\}$.

**Eval**

$$
([([f \ e_1 \cdots e_n]), \hat{\beta}, \hat{ve}, \hat{t}) \Rightarrow (\hat{proc}, \hat{d}, \hat{ve}, \hat{\text{succ}(t)}))
$$

where

$$
\begin{align*}
\hat{proc} & \in \hat{A}(f, \hat{\beta}, \hat{ve}) \\
\hat{d}_i & = \hat{A}(e_i, \hat{\beta}, \hat{ve})
\end{align*}
$$

**Apply**

$$
(([[(\lambda \ (v_1 \cdots v_n) \ call))], \hat{\beta}_b), \hat{d}, \hat{ve}, \hat{t}) \Rightarrow (\text{call}, \hat{\beta}', \hat{ve}', \hat{t})
$$

where

$$
\begin{align*}
\hat{\beta}' & = \hat{\beta}_b[v_i \mapsto \hat{t}] \\
\hat{ve}' & = \hat{ve} \sqcup [(v_i, \hat{t}) \mapsto \hat{d}_i]
\end{align*}
$$
CFA Counting Machinery

Abstract binding counter, $\hat{\mu} : \text{"Bindings"} \rightarrow \{0, 1, \infty\}$.

**Eval**

$\left( \llbracket (f \ e_1 \cdots e_n) \rrbracket, \hat{\beta}, \hat{\nu}, \hat{\mu}, t \right) \Rightarrow (\hat{\text{proc}}, \hat{d}, \hat{\nu}, \hat{\mu}, \text{succ}(t))$

where

\[
\hat{\text{proc}} \in \hat{\mathcal{A}}(f, \hat{\beta}, \hat{\nu}) \\
\hat{d}_i = \hat{\mathcal{A}}(e_i, \hat{\beta}, \hat{\nu})
\]

**Apply**

$\left( \llbracket (\lambda (v_1 \cdots v_n) \ \text{call}) \rrbracket, \hat{\beta}_b, \hat{d}, \hat{\nu}, \hat{\mu}, t \right) \Rightarrow (\text{call}, \hat{\beta}', \hat{\nu}', \hat{\mu}', t)$

where

\[
\begin{align*}
\hat{\beta}' &= \hat{\beta}_b[v_i \mapsto \hat{t}] \\
\hat{\nu}' &= \hat{\nu} \sqcup [(v_i, \hat{t}) \mapsto \hat{d}_i] \\
\hat{\mu}' &= \hat{\mu} \oplus [(v_i, \hat{t}) \mapsto 1]
\end{align*}
\]
Technique: Abstract Garbage Collection

The Idea

1. Trace out bindings reachable from current state.
2. Restrict domain of environment to these bindings.
Looking at the Variable Environment

\[ \hat{b}_1 \rightarrow \hat{proc}_1 \]
\[ \hat{b}_2 \rightarrow \hat{proc}_2 \]
\[ \hat{b}_3 \rightarrow \hat{proc}_3 \]
\[ \hat{b}_4 \rightarrow \hat{proc}_4 \]

**Edge Types**

\[ \hat{b} \rightarrow \hat{proc} \text{ iff } \hat{proc} \in \hat{ve}(\hat{b}) \]
\[ \hat{proc} \rightarrow \hat{b} \text{ iff } \hat{b} \in \hat{T}(\hat{proc}) \]
Looking at the Variable Environment

\[ \hat{b}_1 \rightarrow \hat{proc}_1 \]
\[ \hat{b}_2 \rightarrow \hat{proc}_2 \]
\[ \hat{b}_3 \rightarrow \hat{proc}_3 \]
\[ \hat{b}_4 \rightarrow \hat{proc}_4 \]

Edge Types

\[ \hat{b} \rightarrow \hat{proc} \iff \hat{proc} \in \hat{ve}(\hat{b}) \]
\[ \hat{proc} \rightarrow \hat{b} \iff \hat{b} \in \hat{T}(\hat{proc}) \]
CFA GC Machinery

Bindings touched by an object, $\widehat{T}$:

$$
\widehat{T}(\text{lam}, \widehat{\beta}) = \{(v, \widehat{\beta}(v)) : v \in \text{free(\text{lam})}\}
$$

$$
\widehat{T}(\text{halt}) = \{
\}
$$

$$
\hat{T}\{\widehat{\text{proc}}_1, \ldots, \widehat{\text{proc}}_n\} = \hat{T}(\widehat{\text{proc}}_1) \cup \cdots \cup \hat{T}(\widehat{\text{proc}}_n)
$$

or, by a state:

$$
\hat{T}(\text{call}, \widehat{\beta}, \widehat{ve}, \widehat{\mu}, t) = \{(v, \widehat{\beta}(v)) : v \in \text{free(\text{call})}\}
$$

$$
\hat{T}(\widehat{\text{proc}}, \widehat{d}, \widehat{ve}, \widehat{\mu}, t) = \hat{T}(\widehat{\text{proc}}) \cup \hat{T}(\widehat{d}_1) \cup \cdots \cup \hat{T}(\widehat{d}_n)
$$
Relation $\sim_{ve}$ is set of “touching” edges between abstract bindings:

$$
\hat{b} \sim_{ve} \hat{b}' \iff \hat{b}' \in \hat{T}(\hat{ve}(\hat{b}))
$$
Bindings reachable by state, $\hat{\mathcal{R}} : \hat{\text{State}} \rightarrow \mathcal{P}(\hat{\text{Bind}})$:

$$\hat{\mathcal{R}}(\hat{\varsigma}) = \left\{ \hat{b}' : \hat{b} \in \hat{\mathcal{T}}(\hat{\varsigma}) \text{ and } \hat{b} \overset{*}{\leadsto}_{\hat{\nu} e \hat{\varsigma}} \hat{b}' \right\}$$
GC routine, $\hat{\Gamma} : \widehat{\text{State}} \rightarrow \widehat{\text{State}}$:

$\hat{\Gamma}(\hat{\varsigma}) = \begin{cases} 
(\widehat{\text{proc}}, \hat{d}, \hat{ve}|\hat{\mathcal{R}}(\hat{\varsigma}), \hat{\mu}|\hat{\mathcal{R}}(\hat{\varsigma}), \hat{t}) & \hat{\varsigma} = (\widehat{\text{proc}}, \hat{d}, \hat{ve}, \hat{\mu}, \hat{t}) \\
(\text{call}, \hat{\beta}, \hat{ve}|\hat{\mathcal{R}}(\hat{\varsigma}), \hat{\mu}|\hat{\mathcal{R}}(\hat{\varsigma}), \hat{t}) & \hat{\varsigma} = (\text{call}, \hat{\beta}, \hat{ve}, \hat{\mu}, \hat{t}).
\end{cases}$
Improving Speed *and* Precision

CFA: \( \hat{\varsigma}_1 \)

Gamma CFA: \( \hat{\varsigma}_1 \)
Improving Speed and Precision

\[ \hat{\varsigma}_1 \rightarrow \hat{\varsigma}_2 \]

\[ \Gamma \text{CFA: } \hat{\varsigma}_1 \rightarrow \hat{\varsigma}_2 \]
Improving Speed and Precision

In CFA:
\[ f \mapsto \text{clo}_1, \text{clo}_2, \text{clo}_3 \]

In ΓCFA:
\[ f \mapsto \text{clo}_1 \]
Improving Speed and Precision

In CFA: \( f \mapsto \text{clo}_1, \text{clo}_2, \text{clo}_3 \)

In \( \Gamma \text{CFA} \): \( f \mapsto \text{clo}_1 \)
Improving Speed and Precision

CFA: \[ \hat{s}_1 \rightarrow \hat{s}_2 \rightarrow \hat{s}_3 \rightarrow \hat{s}_4 \]

\[ \hat{s}_{3.1,1} \rightarrow \hat{s}_{3.1,2} \]

\[ \hat{s}_{3.2,1} \]

\[ \hat{s}_{3.1,2} \]

ΓCFA: \[ \hat{s}_1 \rightarrow \hat{s}_2 \rightarrow \hat{s}_3 \rightarrow \hat{s}_4 \]
(let* ((id (λ (x) x))
      (y (id 3)))
  (id 4))

ΓCFA thinks...

id ----> x
  (id 3) ----> y
  (id 4) ---->
(let* ((id (λ (x) x))
       (y (id 3)))
  (id 4))

ΓCFA thinks...

1. (λ (x) x) flows to id.

\[ \begin{align*}
  \text{id} & \mapsto (\lambda \ (x) \ x) \\
  x & \mapsto \\
  \text{id} \ 3 & \mapsto \\
  y & \mapsto \\
  \text{id} \ 4 & \mapsto
\end{align*} \]
ΓCFA & Precision

(let* ((id (λ (x) x))
       (y (id 3)))
  (id 4))

ΓCFA thinks...

1. \(\lambda (x) x\) flows to id.

2. Then, 3 flows to x.

\[
\begin{align*}
\text{id} & \mapsto (\lambda (x) x) \\
\text{x} & \mapsto 3 \\
\text{id 3} & \mapsto \\
y & \mapsto \\
\text{id 4} & \mapsto
\end{align*}
\]
ΓCFA & Precision

(let* ((id (λ (x) x))
        (y (id 3)))
(id 4))

ΓCFA thinks...

1. (λ (x) x) flows to id.

2. Then, 3 flows to x.

3. Then, 3 flows to y, (id 3); x ↦→ 3 now dead.

id ↦→ (λ (x) x)
x ↦→ 3-

(id 3) ↦→ 3
y ↦→ 3
(id 4) ↦→
(let* ((id (λ (x) x))
       (y (id 3)))
  (id 4))

ΓCFA thinks...

1. $(\lambda (x) x)$ flows to id.

2. Then, 3 flows to x.

3. Then, 3 flows to y, (id 3); x $\mapsto$ 3 now dead.

4. Then, 4 flows to x.
(let* ((id (λ (x) x))
    (y (id 3)))
  (id 4))

ΓCFA thinks...

1. $(\lambda (x) x)$ flows to $id$.

2. Then, 3 flows to $x$.  

3. Then, 3 flows to $y$, $(id$ 3); $x \mapsto 3$ now dead.

4. Then, 4 flows to $x$.

5. Then, 4 flows to $(id$ 4).
Theorem (Correctness of GC semantics)

All states at any depth in execution tree are equivalent modulo GC.

\[ I(pr) \]

\[ s_1 \]

\[ s_2 \]

\[ s_1 \]

\[ s_2 \]

\[ s_3 \]

\[ s_4 \]
Theorem (Correctness of GC semantics)

All states at any depth in execution tree are equivalent modulo GC.
Theorem (Correctness of $\Gamma_{\text{CFA}}$)

$\Gamma_{\text{CFA}}$ simulates the concrete semantics.
Early Results: 0CFA
Early Results: 1CFA

The diagram plots the relationship between the number of states and the lines of code for two different systems: 1CFA and 1CFA+GC. The x-axis represents the lines of code, while the y-axis represents the number of states. The graph shows that as the lines of code increase, the number of states also increases, with the 1CFA+GC system generally having fewer states than the 1CFA system.
Related Work

Family
- Cousot & Cousot: Abstract interpretation.
- Steele: CPS as intermediate representation.
- Hudak: Abstract reference counting.

Friends
- Wand & Steckler: Invariance sets.
Ongoing and Future Work

- Implementation in MLton.
  - CPS phase added. (Ben Chambers & Daniel Harvey)
  - $k$-CFA effort underway. (Ben Chambers)
  - $\Gamma$CFA to follow.
  - 100,000+ line benchmarks.
- Fully exploit counting: weave in theorem proving, LFA/ΠCFA.
- Adaptations for direct-style, ANF, SSA.
  - Possible, but so far, uglier.
- Investigate relationship with constraint-based flow analyses.
Thank you.
Question
How often do you garbage collect?

Answer
Whenever precision loss is imminent.

In practice, roughly one in four transitions.
Question
What is the worst-case complexity?

Answer
In theory, the same as the underlying abstract interpretation.

Polyvariance Conjecture
Polyvariance for...
- Non-recursive functions.
- Non-escaping variables.
- Tail-recursive functions.
- Continuation variables.