Improving Flow Analyses via ΓCFA

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

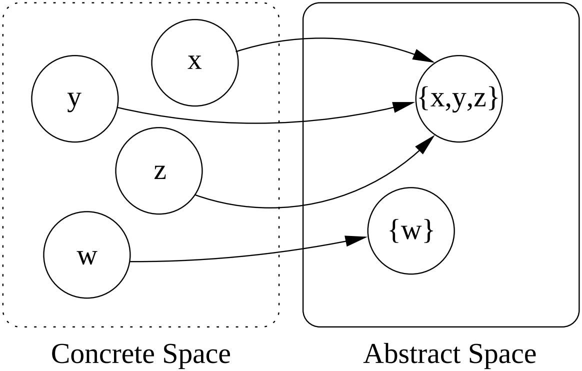
The Process

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



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

- $\{positive\} \oplus \{positive\} = \{positive\}$
- $\{positive, negative\} \oplus \{zero\} = \{positive, negative\}$
- $\{positive\} \oplus \{negative\} = \{negative, zero, positive\}$

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

Example

Analyze: $-4 + 4$

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

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

An Example: Analyzing Integer Arithmetic

Example

Analyze: $-4 + 4$

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

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

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

Imprecision!

$\{zero\}$ is the tightest, safest answer.

0CFA & Precision

Flow analysis question: *What "values" could flow to each expression?*

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

0CFA thinks...

$$\begin{aligned}
\texttt{id} &\mapsto \\
\texttt{x} &\mapsto \\
\texttt{(id 3)} &\mapsto \\
\texttt{y} &\mapsto \\
\texttt{(id 4)} &\mapsto
\end{aligned}$$

0CFA & Precision

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`.

$$
\begin{aligned}
\texttt{id} &\mapsto \texttt{(}\lambda\texttt{ (x) x)}\\
\texttt{x} &\mapsto\\
\texttt{(id 3)} &\mapsto\\
\texttt{y} &\mapsto\\
\texttt{(id 4)} &\mapsto
\end{aligned}
$$

0CFA & Precision

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`.

`id`	$\mapsto$	`(λ (x) x)`
`x`	$\mapsto$	`3`
`(id 3)`	$\mapsto$	
`y`	$\mapsto$	
`(id 4)`	$\mapsto$	

0CFA & Precision

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)`.

$$
\begin{aligned}
\texttt{id} &\mapsto \texttt{(}\lambda\ \texttt{(x) x)} \\
\texttt{x} &\mapsto 3 \\
\texttt{(id 3)} &\mapsto 3 \\
\texttt{y} &\mapsto 3 \\
\texttt{(id 4)} &\mapsto
\end{aligned}
$$

0CFA & Precision

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`.

`id` $\mapsto$	`(λ (x) x)`
`x` $\mapsto$	3, 4
`(id 3)` $\mapsto$	3
`y` $\mapsto$	3
`(id 4)` $\mapsto$	

0CFA & Precision

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)`!?

$\texttt{id} \mapsto \texttt{(}\lambda\ \texttt{(x) x)}$

$\texttt{x} \mapsto 3, 4$

$\texttt{(id 3)} \mapsto 3$

$\texttt{y} \mapsto 3$

$\texttt{(id 4)} \mapsto 3, 4$

Problem

Flow analyses overlap different bindings to the same variable.

0CFA & Precision

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)`!?

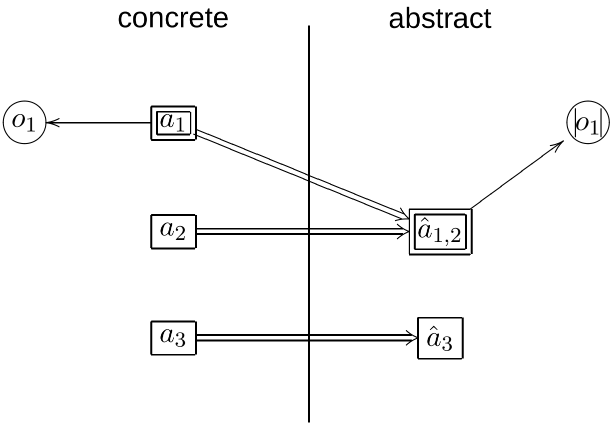
`id` $\mapsto$ `(λ (x) x)`
`x` $\mapsto$ 3, 4
`(id 3)` $\mapsto$ 3
`y` $\mapsto$ 3
`(id 4)` $\mapsto$ 3, 4

Solution

Garbage collect dead bindings mid-analysis.

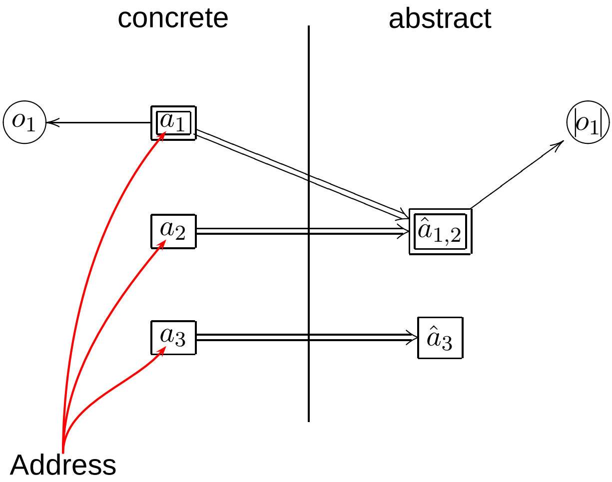
Example: Abstract Garbage Collection

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



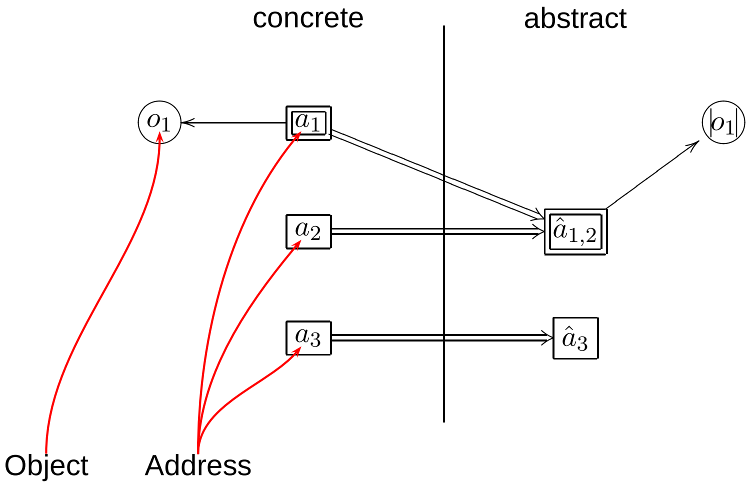
Example: Abstract Garbage Collection

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



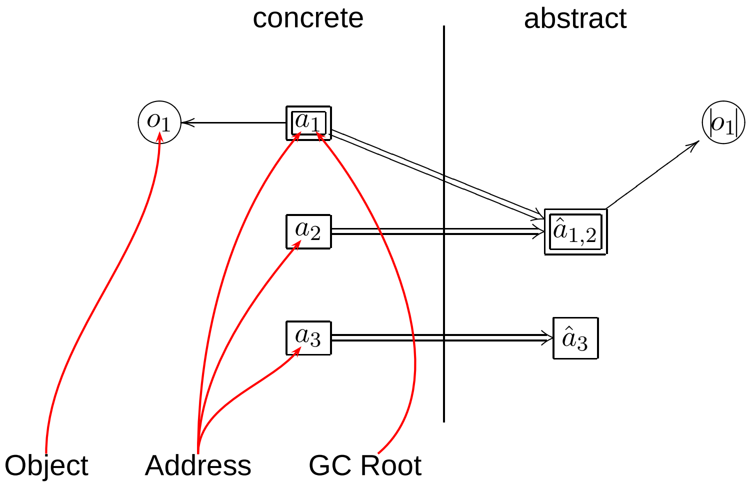
Example: Abstract Garbage Collection

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



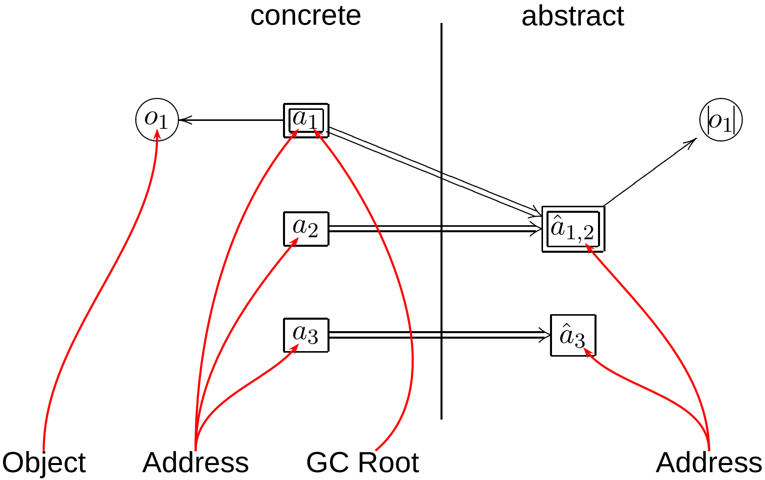
Example: Abstract Garbage Collection

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



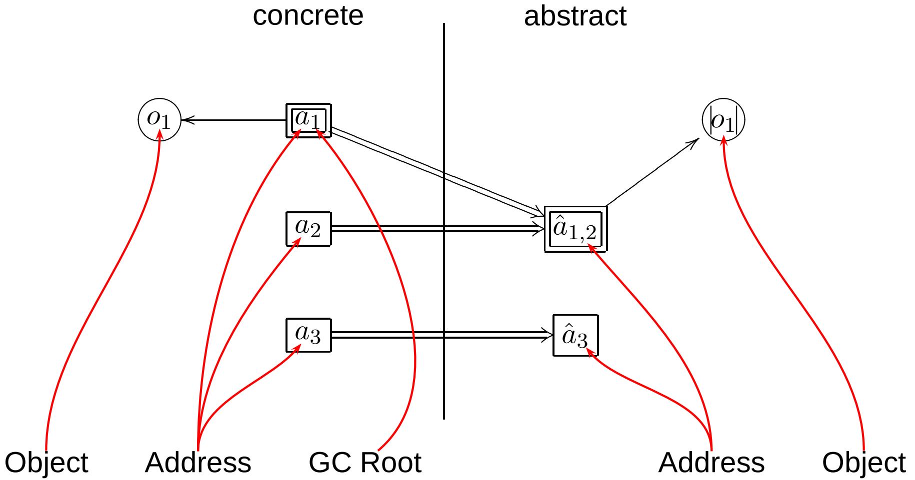
Example: Abstract Garbage Collection

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



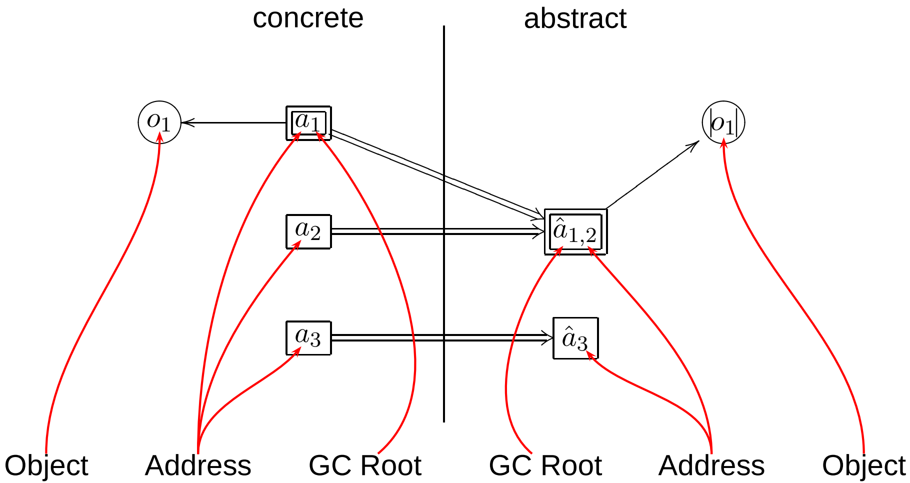
Example: Abstract Garbage Collection

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



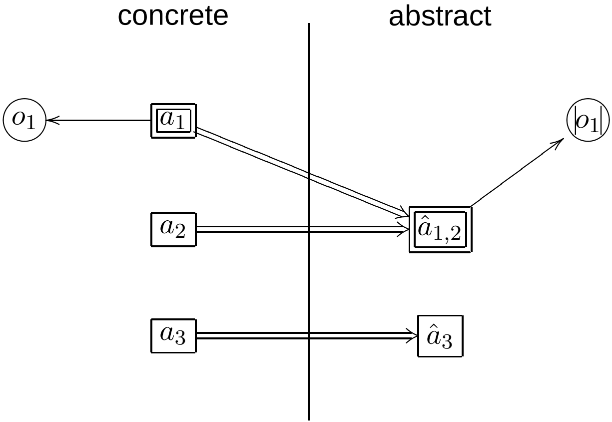
Example: Abstract Garbage Collection

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



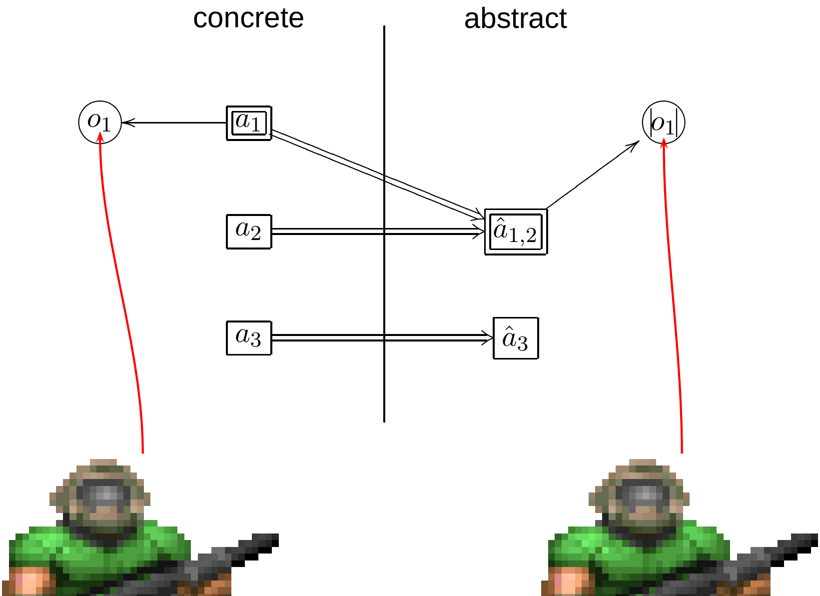
Example: Abstract Garbage Collection

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



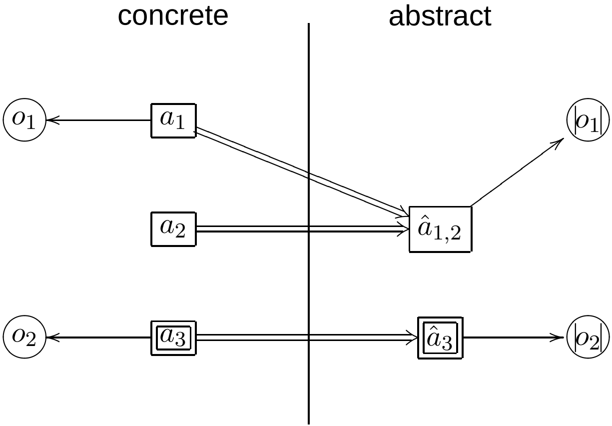
Example: Abstract Garbage Collection

Next: Allocate object o_2 to address a_3. Shift root to a_3.



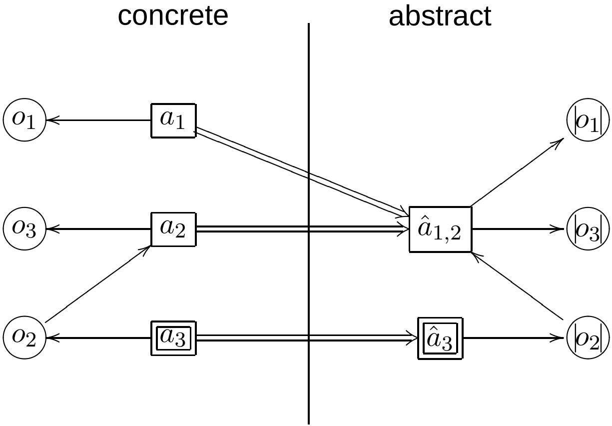
Example: Abstract Garbage Collection

Next: Allocate object o_3 to address a_2. Point o_2 to a_2.



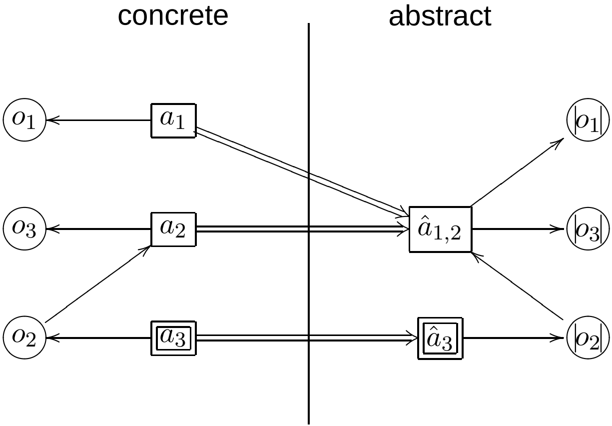
Example: Abstract Garbage Collection

Uh-oh! Zombie born. Concrete-abstract symmetry broken.



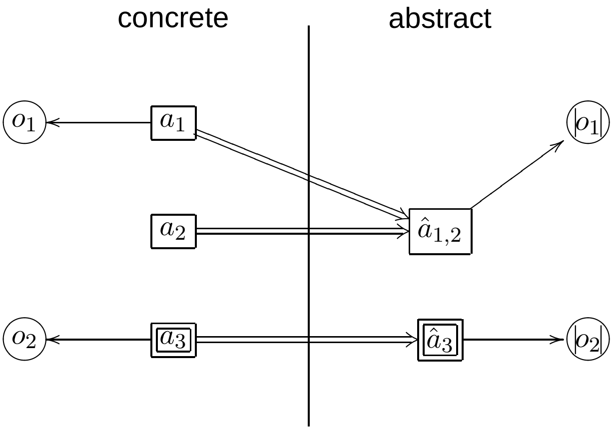
Example: Abstract Garbage Collection

Solution: Rewind and garbage collect first.



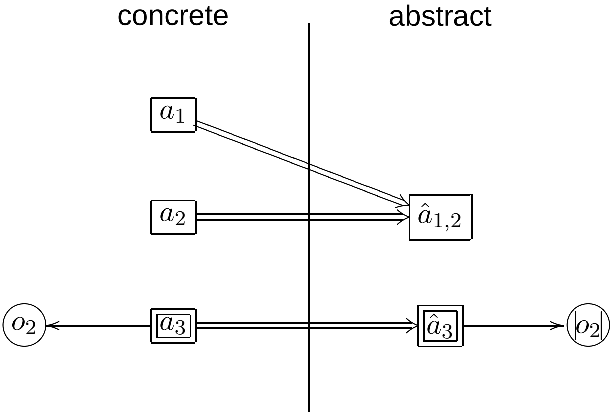
Example: Abstract Garbage Collection

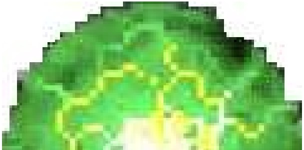
As it was:

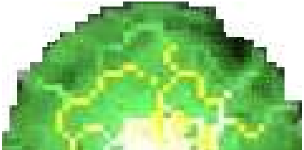


Example: Abstract Garbage Collection

After garbage collection:

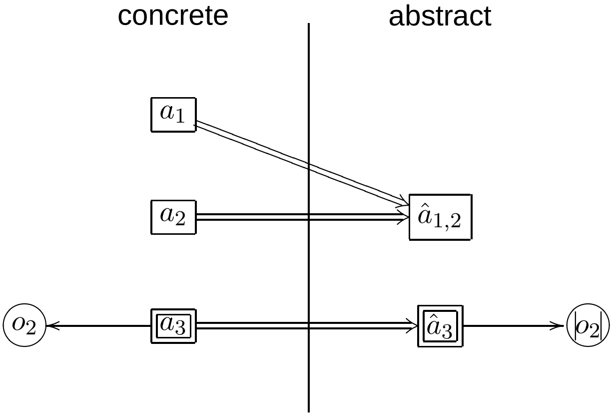






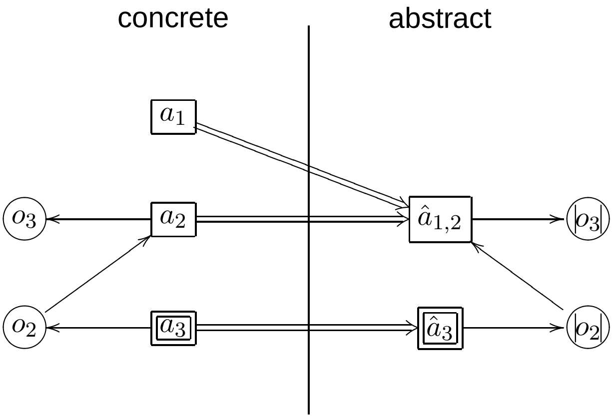
Example: Abstract Garbage Collection

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.

$$\overbrace{\begin{aligned} e, f \in EXP &::= v \\ &\mid\ (\lambda\ (v_1 \cdots v_n)\ call) \\ call \in CALL &::= (f\ e_1 \cdots e_n) \end{aligned}}^{\text{CPS } \lambda\text{-calculus}}$$

CPS Narrows Concern

λ is universal representation of control & env.

Construct	encoding
fun call	call to λ
fun return	call to λ
iteration	call to λ
sequencing	call to λ
conditional	call to λ
exception	call to λ
coroutine	call to λ
$\vdots$	$\vdots$

Advantage

Now λ is fine-grained construct.

Eval-to-Apply Transition

$$\frac{proc = \mathcal{A}(f, \beta, ve) \qquad d_i = \mathcal{A}(e_i, \beta, ve)}{([\![(f\ e_1 \cdots e_n)]\!], \beta, ve, t) \Rightarrow (proc, \mathbf{d}, ve, t+1)}$$

Apply-to-Eval Transition

$$\frac{proc = ([\![(\lambda\ (v_1 \cdots v_n)\ \ call)]\!], \beta')}{(proc, \mathbf{d}, ve, t) \Rightarrow (call, \beta'[v_i \mapsto t], ve[(v_i, t) \mapsto d_i], t)}$$

Domains

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

Lookup function

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

Eval-to-Apply Transition

$$\frac{\widehat{proc} \in \widehat{\mathcal{A}}(f, \widehat{\beta}, \widehat{ve}) \qquad \widehat{d}_i = \widehat{\mathcal{A}}(e_i, \widehat{\beta}, \widehat{ve})}{(\llbracket (f\ e_1 \cdots e_n) \rrbracket, \widehat{\beta}, \widehat{ve}, \widehat{t}) \approx> (\widehat{proc}, \widehat{\mathbf{d}}, \widehat{ve}, \widehat{succ}(\widehat{t}))}$$

Apply-to-Eval Transition

$$\frac{\widehat{proc} = (\llbracket (\lambda\ (v_1 \cdots v_n)\ call) \rrbracket, \widehat{\beta}')}{(\widehat{proc}, \widehat{\mathbf{d}}, \widehat{ve}, \widehat{t}) \approx> (call, \widehat{\beta}'[v_i \mapsto \widehat{t}], \widehat{ve} \sqcup [(v_i, \widehat{t}) \mapsto \widehat{d}_i], \widehat{t})}$$

Domains

$$\begin{array}{rcrcl}
\widehat{\varsigma} & \in & \widehat{Eval} & = & CALL \times \widehat{BEnv} \times \widehat{VEnv} \times \widehat{Time} \\
 & + & \widehat{Apply} & = & \widehat{Proc} \times \widehat{D}^* \times \widehat{VEnv} \times \widehat{Time} \\
\widehat{\beta} & \in & \widehat{BEnv} & = & VAR \to \widehat{Time} \\
\widehat{ve} & \in & \widehat{VEnv} & = & VAR \times \widehat{Time} \to \widehat{D} \\
\widehat{proc} & \in & \widehat{Proc} & = & \widehat{Clo} + \{halt\} \\
\widehat{clo} & \in & \widehat{Clo} & = & LAM \times \widehat{BEnv} \\
\widehat{d} & \in & \widehat{D} & = & \mathcal{P}(\widehat{Proc}) \\
\widehat{t} & \in & \widehat{Time} & = & \text{finite set of times (contours)}
\end{array}$$

Lookup function

$$\widehat{\mathcal{A}}(lam, \widehat{\beta}, \widehat{ve})$$
$$= \{(lam, \widehat{\beta})\}$$
$$\widehat{\mathcal{A}}(v, \widehat{\beta}, \widehat{ve})$$
$$= \widehat{ve}(v, \widehat{\beta}(v))$$

Eval-to-Apply Transition

$$\frac{\widehat{proc} \in \widehat{\mathcal{A}}(f, \hat{\beta}, \widehat{ve}) \qquad \hat{d}_i = \widehat{\mathcal{A}}(e_i, \hat{\beta}, \widehat{ve})}{([\![(f\ e_1 \cdots e_n)]\!], \hat{\beta}, \widehat{ve}, \hat{t}) \approx\!> (\widehat{proc}, \hat{\mathbf{d}}, \widehat{ve}, \widehat{succ}(\hat{t}))}$$

Apply-to-Eval Transition

$$\frac{\widehat{proc} = ([\![(\lambda\ (v_1 \cdots v_n)\ \ call)]\!], \hat{\beta}')}{(\widehat{proc}, \hat{\mathbf{d}}, \widehat{ve}, \hat{t}) \approx\!> (call, \hat{\beta}'[v_i \mapsto \hat{t}], \widehat{ve} \sqcup [(v_i, \hat{t}) \mapsto \hat{d}_i], \hat{t})}$$

Domains

$$\begin{array}{rcccl}
\hat{\varsigma} & \in & \widehat{Eval} & = & CALL \times \widehat{BEnv} \times \widehat{VEnv} \times \widehat{Time} \\
 & + & \widehat{Apply} & = & \widehat{Proc} \times \hat{D}^* \times \widehat{VEnv} \times \widehat{Time} \\
\hat{\beta} & \in & \widehat{BEnv} & = & VAR \rightarrow \widehat{Time} \\
\widehat{ve} & \in & \widehat{VEnv} & = & VAR \times \widehat{Time} \rightarrow \hat{D} \\
\widehat{proc} & \in & \widehat{Proc} & = & \widehat{Clo} + \{halt\} \\
\widehat{clo} & \in & \widehat{Clo} & = & LAM \times \widehat{BEnv} \\
\hat{d} & \in & \hat{D} & = & \mathcal{P}(\widehat{Proc}) \\
\hat{t} & \in & \widehat{Time} & = & \text{finite set of times (contours)}
\end{array}$$

Lookup function

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

Concrete *v.* Abstract Interpretations

Interpretation

Concrete: ς_1

Abstract: $\widehat{\varsigma_1}$

Concrete *v.* Abstract Interpretations

Interpretation

Concrete: $\varsigma_1 \longrightarrow \varsigma_2$

Abstract: $\widehat{\varsigma}_1 \longrightarrow \widehat{\varsigma}_2$

Concrete *v.* Abstract Interpretations

Interpretation

Concrete: $\varsigma_1 \longrightarrow \varsigma_2 \longrightarrow \varsigma_3$

Abstract: $\widehat{\varsigma}_1 \longrightarrow \widehat{\varsigma}_2 \longrightarrow \widehat{\varsigma}_3$

Concrete *v.* Abstract Interpretations

Interpretation

Concrete: $\varsigma_1 \longrightarrow \varsigma_2 \longrightarrow \varsigma_3 \longrightarrow \varsigma_4$

Abstract:

$$\begin{tikzcd}
 & & & \widehat{\varsigma}_{3.1,1} \\
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[r] \arrow[ur] \arrow[dr] & \widehat{\varsigma}_4 \\
 & & & \widehat{\varsigma}_{3.2,1}
\end{tikzcd}$$

Concrete *v.* Abstract Interpretations

Interpretation

Concrete:

$$\begin{tikzcd} \varsigma_1 \arrow[r] & \varsigma_2 \arrow[r] & \varsigma_3 \arrow[r] & \varsigma_4 \arrow[r] & \varsigma_5 \end{tikzcd}$$

Abstract:

$$\begin{tikzcd}
 & & & \widehat{\varsigma}_{3.1,1} \arrow[r] & \widehat{\varsigma}_{3.1,2} \\
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[r] \arrow[ur] \arrow[dr] & \widehat{\varsigma}_4 \arrow[ur] & \\
 & & & \widehat{\varsigma}_{3.2,1} \arrow[ull, bend left] &
\end{tikzcd}$$

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 $\widehat{\beta}_1(v) = \widehat{\beta}_2(v)$*,*
and $\widehat{\mu}(v, \widehat{\beta}_1(v)) = \widehat{\mu}(v, \widehat{\beta}_2(v)) = 1$*,*
then $\beta_1(v) = \beta_2(v)$*.*

ΓCFA Environment Condition

Basic Principle

If $\{x\} = \{y\}$, then $x = y$.

Theorem (Environment condition)

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

ΓCFA Environment Condition

Basic Principle

If $\{x\} = \{y\}$, then $x = y$.

Theorem (Environment condition)

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

ΓCFA Environment Condition

Basic Principle

If $\{x\} = \{y\}$, then $x = y$.

Theorem (Environment condition)

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

ΓCFA Environment Condition

Basic Principle

If $\{x\} = \{y\}$, then $x = y$.

Theorem (Environment condition)

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

Increase in Power

Enables the Super-β class of optimizations.

ΓCFA Counting Machinery

Abstract binding counter, $\widehat{\mu}$: "Bindings" $\to \{0, 1, \infty\}$.

Eval

$$([\![(f\ e_1 \cdots e_n)]\!], \widehat{\beta}, \widehat{ve}, \quad \widehat{t}) \approx> (\widehat{proc}, \widehat{\mathbf{d}}, \widehat{ve}, \quad \widehat{succ}(\widehat{t}))$$

where $\begin{cases} \widehat{proc} \in \widehat{\mathcal{A}}(f, \widehat{\beta}, \widehat{ve}) \\ \widehat{d}_i = \widehat{\mathcal{A}}(e_i, \widehat{\beta}, \widehat{ve}) \end{cases}$

Apply

$$(([\![(\lambda\ (v_1 \cdots v_n)\ call)]\!], \widehat{\beta}_b), \widehat{\mathbf{d}}, \widehat{ve}, \quad \widehat{t}) \approx> (call, \widehat{\beta}', \widehat{ve}', \quad \widehat{t})$$

where $\begin{cases} \widehat{\beta}' = \widehat{\beta}_b[v_i \mapsto \widehat{t}] \\ \widehat{ve}' = \widehat{ve} \sqcup [(v_i, \widehat{t}) \mapsto \widehat{d}_i] \end{cases}$

ΓCFA Counting Machinery

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

Eval

$$([\![(f\ e_1 \cdots e_n)]\!], \widehat{\beta}, \widehat{ve}, \widehat{\mu}, \widehat{t}) \approx> (\widehat{proc}, \widehat{\mathbf{d}}, \widehat{ve}, \widehat{\mu}, \widehat{succ}(\widehat{t}))$$

$$\text{where} \begin{cases} \widehat{proc} \in \widehat{\mathcal{A}}(f, \widehat{\beta}, \widehat{ve}) \\ \widehat{d}_i = \widehat{\mathcal{A}}(e_i, \widehat{\beta}, \widehat{ve}) \end{cases}$$

Apply

$$(([\![(\lambda\ (v_1 \cdots v_n)\ call)]\!], \widehat{\beta}_b), \widehat{\mathbf{d}}, \widehat{ve}, \widehat{\mu}, \widehat{t}) \approx> (call, \widehat{\beta}', \widehat{ve}', \widehat{\mu}', \widehat{t})$$

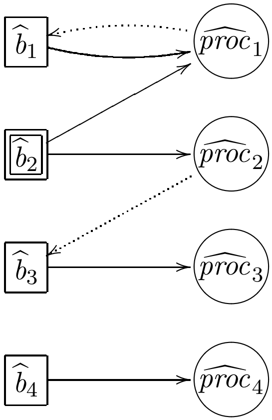
$$\text{where} \begin{cases} \widehat{\beta}' = \widehat{\beta}_b[v_i \mapsto \widehat{t}] \\ \widehat{ve}' = \widehat{ve} \sqcup [(v_i, \widehat{t}) \mapsto \widehat{d}_i] \\ \widehat{\mu}' = \widehat{\mu} \oplus [(v_i, \widehat{t}) \mapsto 1] \end{cases}$$

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

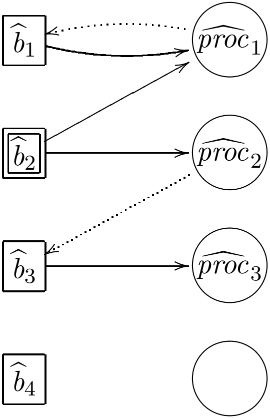


Edge Types

$\widehat{b} \rightarrow \widehat{proc}$ iff $\widehat{proc} \in \widehat{ve}(\widehat{b})$

$\widehat{proc} \dashrightarrow \widehat{b}$ iff $\widehat{b} \in \widehat{\mathcal{T}}(\widehat{proc})$

Looking at the Variable Environment



Edge Types

$\widehat{b} \rightarrow \widehat{proc}$ iff $\widehat{proc} \in \widehat{ve}(\widehat{b})$

$\widehat{proc} \dashrightarrow \widehat{b}$ iff $\widehat{b} \in \widehat{\mathcal{T}}(\widehat{proc})$

ΓCFA GC Machinery

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

$$\widehat{\mathcal{T}}(lam, \widehat{\beta}) = \left\{(v, \widehat{\beta}(v)) : v \in \mathit{free}(lam)\right\}$$

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

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

or, by a state:

$$\widehat{\mathcal{T}}(call, \widehat{\beta}, \widehat{ve}, \widehat{\mu}, \widehat{t}) = \left\{(v, \widehat{\beta}(v)) : v \in \mathit{free}(call)\right\}$$

$$\widehat{\mathcal{T}}(\widehat{proc}, \widehat{\mathbf{d}}, \widehat{ve}, \widehat{\mu}, \widehat{t}) = \widehat{\mathcal{T}}(\widehat{proc}) \cup \widehat{\mathcal{T}}(\widehat{d}_1) \cup \cdots \cup \widehat{\mathcal{T}}(\widehat{d}_n)$$

ΓCFA GC Machinery

Relation $\hat{\rightsquigarrow}_{\widehat{ve}}$ is set of “touching” edges between abstract bindings:

$$\widehat{b} \;\hat{\rightsquigarrow}_{\widehat{ve}}\; \widehat{b}' \iff \widehat{b}' \in \widehat{\mathcal{T}}(\widehat{ve}(\widehat{b}))$$

ΓCFA GC Machinery

Bindings reachable by state, $\widehat{\mathcal{R}} : \widehat{State} \to \mathcal{P}(\widehat{Bind})$:

$$\widehat{\mathcal{R}}(\widehat{\varsigma}) = \left\{ \widehat{b}' : \widehat{b} \in \widehat{\mathcal{T}}(\widehat{\varsigma}) \text{ and } \widehat{b} \mathrel{\hat{\rightsquigarrow}}^{*}_{\widehat{ve}_{\widehat{\varsigma}}} \widehat{b}' \right\}$$

ΓCFA GC Machinery

GC routine, $\widehat{\Gamma} : \widehat{State} \to \widehat{State}$:

$$\widehat{\Gamma}(\widehat{\varsigma}) = \begin{cases} (\widehat{proc}, \widehat{\boldsymbol{d}}, \widehat{ve}|\widehat{\mathcal{R}}(\widehat{\varsigma}), \widehat{\mu}|\widehat{\mathcal{R}}(\widehat{\varsigma}), \widehat{t}) & \widehat{\varsigma} = (\widehat{proc}, \widehat{\boldsymbol{d}}, \widehat{ve}, \widehat{\mu}, \widehat{t}) \\ (call, \widehat{\beta}, \widehat{ve}|\widehat{\mathcal{R}}(\widehat{\varsigma}), \widehat{\mu}|\widehat{\mathcal{R}}(\widehat{\varsigma}), \widehat{t}) & \widehat{\varsigma} = (call, \widehat{\beta}, \widehat{ve}, \widehat{\mu}, \widehat{t}). \end{cases}$$

Improving Speed *and* Precision

CFA: $\widehat{\varsigma}_1$

ΓCFA: $\widehat{\varsigma}_1$

Improving Speed *and* Precision

CFA: $\widehat{\varsigma_1} \longrightarrow \widehat{\varsigma_2}$

ΓCFA: $\widehat{\varsigma_1} \longrightarrow \widehat{\varsigma_2}$

Improving Speed *and* Precision

CFA: $\widehat{\varsigma}_1 \longrightarrow \widehat{\varsigma}_2 \longrightarrow \widehat{\varsigma}_3$

ΓCFA: $\widehat{\varsigma}_1 \longrightarrow \widehat{\varsigma}_2 \longrightarrow \widehat{\varsigma}_3$

$(f\ e_1 \cdots e_n)$

In CFA: $f \mapsto clo_1,\ clo_2,\ clo_3$

In ΓCFA: $f \mapsto clo_1$

Improving Speed *and* Precision

CFA:

$$
\begin{tikzcd}
 & & & \widehat{\varsigma}_{3.1,1} \\
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[ur] \arrow[r] \arrow[dr] & \widehat{\varsigma}_4 \\
 & & & \widehat{\varsigma}_{3.2,1}
\end{tikzcd}
$$

ΓCFA:

$$
\begin{tikzcd}
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[r] & \widehat{\varsigma}_4
\end{tikzcd}
$$

$(f\ e_1 \cdots e_n)$

In CFA: $f \mapsto clo_1,\ clo_2,\ clo_3$

In ΓCFA: $f \mapsto clo_1$

Improving Speed *and* Precision

CFA:

$$\begin{tikzcd}
 & & & \widehat{\varsigma}_{3.1,1} \arrow[r] & \widehat{\varsigma}_{3.1,2} \\
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[r] \arrow[ur] \arrow[dr] & \widehat{\varsigma}_4 \arrow[ur] & \\
 & & & \widehat{\varsigma}_{3.2,1} \arrow[ull, bend left] &
\end{tikzcd}$$

ΓCFA:

$$\begin{tikzcd}
 & & & & \widehat{\varsigma}_{3.1,2} \\
\widehat{\varsigma}_1 \arrow[r] & \widehat{\varsigma}_2 \arrow[r] & \widehat{\varsigma}_3 \arrow[r] & \widehat{\varsigma}_4 \arrow[ur] &
\end{tikzcd}$$

ΓCFA & Precision

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

ΓCFA thinks...

$$\begin{aligned} \texttt{id} &\mapsto \\ \texttt{x} &\mapsto \\ \texttt{(id 3)} &\mapsto \\ \texttt{y} &\mapsto \\ \texttt{(id 4)} &\mapsto \end{aligned}$$

ΓCFA & Precision

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

ΓCFA thinks...

1. `(λ (x) x)` flows to `id`.

$$
\begin{aligned}
\texttt{id} &\mapsto \texttt{(}\lambda\texttt{ (x) x)} \\
\texttt{x} &\mapsto \\
\texttt{(id 3)} &\mapsto \\
\texttt{y} &\mapsto \\
\texttt{(id 4)} &\mapsto
\end{aligned}
$$

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`.

`id`	$\mapsto$	`(λ (x) x)`
`x`	$\mapsto$	3
`(id 3)`	$\mapsto$	
`y`	$\mapsto$	
`(id 4)`	$\mapsto$	

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` $\mapsto$ `3` now dead.

`id`	$\mapsto$	`(λ (x) x)`
`x`	$\mapsto$	~~`3`~~
`(id 3)`	$\mapsto$	`3`
`y`	$\mapsto$	`3`
`(id 4)`	$\mapsto$	

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` $\mapsto$ `3` now dead.
4. Then, `4` flows to `x`.

`id` $\mapsto$	`(λ (x) x)`
`x` $\mapsto$	~~`3`~~ `4`
`(id 3)` $\mapsto$	`3`
`y` $\mapsto$	`3`
`(id 4)` $\mapsto$	

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` $\mapsto$ 3 now dead.
4. Then, `4` flows to `x`.
5. Then, `4` flows to `(id 4)`.

`id`	$\mapsto$ `(λ (x) x)`
`x`	$\mapsto$ ~~3~~ 4
`(id 3)`	$\mapsto$ 3
`y`	$\mapsto$ 3
`(id 4)`	$\mapsto$ 4

Important Details: Concrete Correctness

Theorem (Correctness of GC semantics)

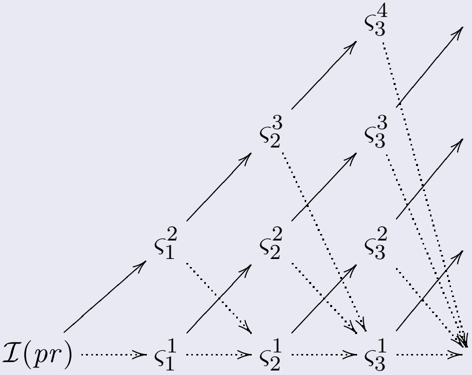
All states at any depth in execution tree are equivalent modulo GC.

$$\begin{tikzcd}
& & \mathcal{I}(pr) \arrow[dl, dotted] \arrow[dr] & & \\
& \varsigma_1^1 \arrow[dl, dotted] \arrow[d] & & \varsigma_1^2 \arrow[d, dotted] \arrow[dr] & \\
\varsigma_2^1 & \varsigma_2^2 & & \varsigma_2^3 & \varsigma_2^4
\end{tikzcd}$$

Important Details: Concrete Correctness

Theorem (Correctness of GC semantics)

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



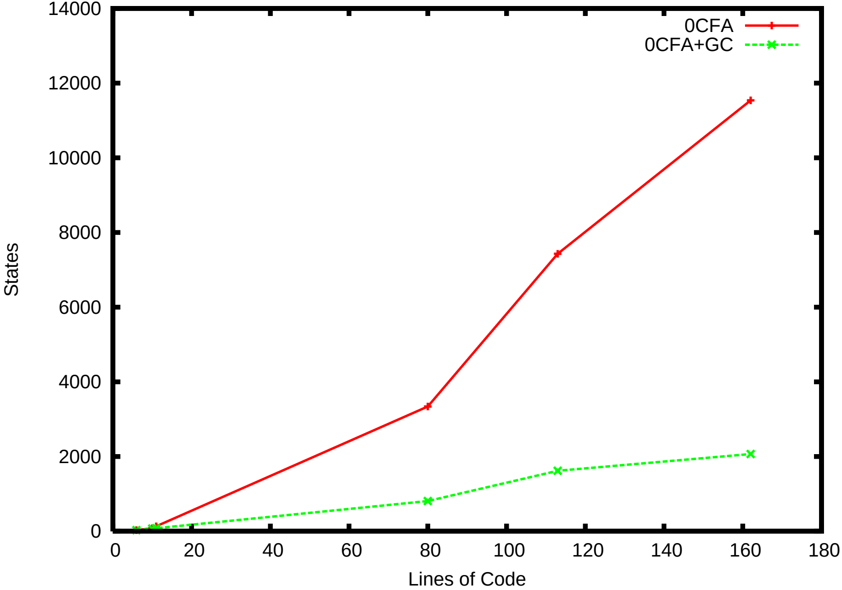
Important Details: Abstract Correctness

Theorem (Correctness of ΓCFA)

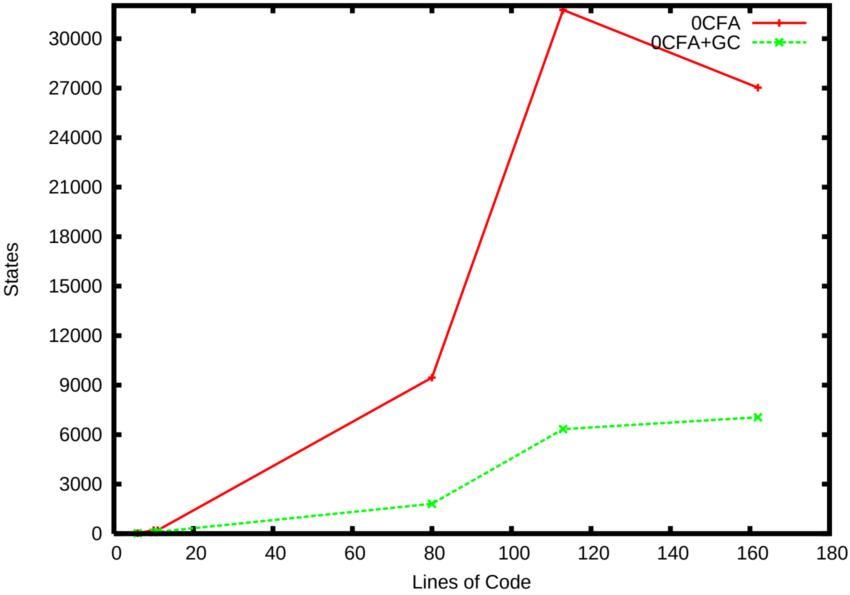
Γ*CFA simulates the concrete semantics.*

$$
\begin{tikzcd}
\varsigma \arrow[r, "|\cdot|"] \arrow[d, "\Rightarrow"'] & |\varsigma| \arrow[r, "\sqsubseteq"] & \widehat{\varsigma} \arrow[d, dashed, "\approx>"] \\
\varsigma' \arrow[r, "|\cdot|"] & |\varsigma'| \arrow[r, "\sqsubseteq"] & \widehat{\varsigma}'
\end{tikzcd}
$$

Early Results: 0CFA



Early Results: 1CFA



Related Work

Family

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

Friends

- Jagannathan, *et al.*: “Singleness” analysis.
- Wand & Steckler: Invariance sets.

Ongoing and Future Work

- Implementation in MLton.
  - CPS phase added. (Ben Chambers & Daniel Harvey)
  - k-CFA effort underway. (Ben Chambers)
  - ΓCFA 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.