Logic-Flow Analysis of Higher-Order Programs

Matt Might

http://matt.might.net/

POPL 2007

Why?

Tim Sweeney, POPL 2006

Static array-bounds checking.

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Why?

Tim Sweeney, POPL 2006

Static array-bounds checking.

Example

```
... a[i] ...
```

Will $\mathtt{0} \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Wait...

Hasn't this been done already? (Range analysis, *etc.*)

Why?

Tim Sweeney, POPL 2006

Static array-bounds checking.

Example

```
... a[i] ...
```

Will $0 \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Wait...

Hasn't this been done already? (Range analysis, *etc.*)
Yeah, but not for permutation/vertex array code.

Why?

Tim Sweeney, POPL 2006

Static array-bounds checking.

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Wait...

Hasn't this been done already? (Range analysis, *etc.*)
Yeah, but not for permutation/vertex array code.

Not the big idea

LFA is **not** about array-bounds checking.

The Idea

Theorem Proving

Flow Analysis

The Idea

Theorem Proving   Flow Analysis

The Idea

Theorem Proving

Flow Analysis

How?

Abstract interpretation

Mechanical (flow): $\widehat{\varsigma} \longrightarrow \widehat{\varsigma}' \longrightarrow \widehat{\varsigma}'' \longrightarrow \cdots$

Propositional (logic): $\Pi \dashrightarrow \Pi' \dashrightarrow \Pi'' \dashrightarrow \cdots$

How?

Abstract interpretation

Mechanical (flow):

Propositional (logic):

$$\begin{tikzcd}
\widehat{\varsigma} \arrow[r] \arrow[dr, dotted] & \widehat{\varsigma}' \arrow[r] \arrow[dr, dotted] & \widehat{\varsigma}'' \arrow[r] \arrow[dr, dotted] & \cdots \\
\Pi \arrow[r, dotted] \arrow[ur] & \Pi' \arrow[r, dotted] \arrow[ur] & \Pi'' \arrow[r, dotted] \arrow[ur] & \cdots
\end{tikzcd}$$

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Flow analysis results

- `a` is an array.

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Flow analysis results

- `a` is an array from line 10 or line 30.

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Flow analysis results

- `a` is an array from line 10 or line 30.
- `i` is an integer.

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $\texttt{0} \leq \texttt{i} < \texttt{a.length}$ always hold?

Flow analysis results

- `a` is an array from line 10 or line 30.
- `i` is a non-negative integer.

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Flow analysis results

- `a` is an array from line 10 or line 30.
- `i` is a non-negative integer.
- `a.length` is a positive integer.

Higher-order flow analysis fails

Example

```
... a[i] ...
```

Will $0 \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Flow analysis results

- `a` is an array from line 10 or line 30.
- `i` is a non-negative integer.
- `a.length` is a positive integer.

Insufficiently rich information

Frequently can't show $\mathtt{i} < \mathtt{a.length}$.

First attempt: Enrich flow values with relations

Example

```
... a[i] ...
```

Will $0 \leq \mathtt{i} < \mathtt{a.length}$ always hold?

First attempt: Enrich flow values with relations

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Hypothetical results

- `a` is an array from line 10 or line 30.
- `i` is a non-negative integer.
- `a.length` is a positive integer.

First attempt: Enrich flow values with relations

Example

```
... a[i] ...
```

Will $0 \leq \texttt{i} < \texttt{a.length}$ always hold?

Hypothetical results

- `a` is an array from line 10 or line 30.
- `i` is in $\{x : 0 \leq x < \texttt{a.length}\}$.
- `a.length` is a positive integer.

First attempt: Enrich flow values with relations

Example

```
... a[i] ...
```

Will $\mathtt{0} \leq \mathtt{i} < \mathtt{a.length}$ always hold?

Hypothetical results

- `a` is an array from line 10 or line 30.
- `i` is in $\{x : 0 \leq x < \mathtt{a.length}\}$.
- `a.length` is a positive integer.

Problems

1. What does `a.length` mean where `a` is out of scope?
2. How did this set get there in the first place?

First attempt: Enrich flow values with relations

Example

```
... a[i] ...
```

Will $\texttt{0} \le \texttt{i} < \texttt{a.length}$ always hold?

Hypothetical results

- `a` is an array from line 10 or line 30.
- `i` is in $\{x : 0 \le x < \texttt{a.length}\}$.
- `a.length` is a positive integer.

Problems

1. What does `a.length` mean where `a` is out of scope?
2. How did this set get there in the first place?

Problem 1: Meaning of variables

Ambiguity problem

- Need environment-independent **identities** for values.

Problem 1: Meaning of variables

Ambiguity problem

- Need environment-independent **identities** for values.

Solutions

- Constants. *E.g.* 5 means 5 anywhere.

Problem 1: Meaning of variables

Ambiguity problem

- Need environment-independent **identities** for values.

Solutions

- Constants. *E.g.* 5 means 5 anywhere.
- Heap locations. *E.g.* Heap location 10 offset 3.

Problem 1: Meaning of variables

Ambiguity problem

- Need environment-independent **identities** for values.

Solutions

- Constants. *E.g.* 5 means 5 anywhere.
- Heap locations. *E.g.* Heap location 10 offset 3.
- Bindings. [Shivers 1988]

Bindings

Definition

A **binding** is a variable-time pairing, *e.g.* $(\mathtt{x}, 3)$.

Bindings

Definition

A **binding** is a variable-time pairing, *e.g.* $(\mathtt{x}, 3)$.

Example (Variable *v*. binding)

- Value of variable $\mathtt{x}$ depends on environment.
- Value of $\mathtt{x}$ *bound at time 3*: Same in any environment/state.

Bindings

Definition

A **binding** is a variable-time pairing, *e.g.* $(\mathtt{x}, 3)$.

Example (Variable *v*. binding)

- Value of variable `x` depends on environment.
- Value of `x` *bound at time 3*: Same in any environment/state.

Example (Abstract bindings)

- 0CFA: All values of `x` bound at any time.
- 1CFA: All values of `x` bound at a time while calling `foo`.

Strategy

- Build binding-sensitive flow analysis.

Strategy

- Build binding-sensitive flow analysis.
- Build binding-sensitive logic.

Strategy

- Build binding-sensitive flow analysis.
- Build binding-sensitive logic.
- Build binding-sensitive propositional abstract interpretation.

Strategy

- Build binding-sensitive flow analysis.
- Build binding-sensitive logic.
- Build binding-sensitive propositional abstract interpretation.
- Weave.

Tool 1: Continuation-passing style (CPS)

Contract

- Calls never return.
- Continuations are passed to receive the result.

Example

Direct-style identity function:

```
(define (id x)
  x)
```

Example

CPS identity function:

```
(define (id x return)
 (return x))
```

CPS simplifies

In CPS,
fun call,
fun return,
conditional branch,
sequencing,
iteration,
exception throw,
coroutine switch,
continuation invocation...

CPS simplifies

In CPS,
fun call,
fun return,
conditional branch,
sequencing,
iteration,
exception throw,
coroutine switch,
continuation invocation...

...all become call to λ

CPS simplifies

In CPS,
fun call,
fun return,
conditional branch,
sequencing,
iteration,
exception throw,
coroutine switch,
continuation invocation...

...all become call to λ

Machine state without CPS

$$\varsigma \in State = CALL \times Env \times Store \times Stack \times Time$$

CPS simplifies

In CPS,
fun call,
fun return,
conditional branch,
sequencing,
iteration,
exception throw,
coroutine switch,
continuation invocation...

...all become call to λ

Machine state without CPS

$$\varsigma \in State = CALL \times Env \times Store \times Stack \times Time$$

CPS simplifies

In CPS,
fun call,
fun return,
conditional branch,
sequencing,
iteration,
exception throw,
coroutine switch,
continuation invocation...

...all become call to λ

Machine state with CPS

$$\varsigma \in State = CALL \times Env \times Store \times Time$$

Tool 2: Machine-based abstract interpretation

Idea

Abstract machine states component-wise.

Machine state

- A call site.
- An environment for variable lookup.
- A heap.
- A time.

More formally

$$\varsigma \in State = CALL \times Env \times Store \times Time$$

Tool 2: Machine-based abstract interpretation

Idea

Abstract machine states component-wise.

Abstract machine state

- An abstract call site.
- An abstract environment for variable lookup.
- An abstract heap.
- An abstract time.

More formally

$$\widehat{\varsigma} \in \widehat{State} = \widehat{CALL} \times \widehat{Env} \times \widehat{Store} \times \widehat{Time}$$

Tool 2: Machine-based abstract interpretation

Idea

Abstract machine states component-wise.

Abstract machine state

- An abstract call site.
- An abstract environment for variable lookup.
- An abstract heap.
- An abstract time.

More formally

$$\widehat{\varsigma} \in \widehat{State} = \widehat{CALL} \times \widehat{Env} \times \widehat{Store} \times \widehat{Time}$$

Binding-factored environment

Definition

An **environment** maps variables to values.

Binding-factored environment

Definition

An **environment** maps variables to values.

Definition

A **binding-factored environment** (β, ve) [Shivers 1988] maps:

- variables to their binding times. (β)
- and then, variables plus times (bindings) to values. (ve)

Tool 3: Restricted first-order logic for states

Features

- Propositions are facts about concrete machine states.
- Ground terms are identities (bindings, locations, constants).

Tool 3: Restricted first-order logic for states

Features

- Propositions are facts about concrete machine states.
- Ground terms are identities (bindings, locations, constants).

Restrictions

- No existential quantifiers.
- Only outer-level universal quantifiers.
- Quantifiers range over abstract identities.

Example: Proposition

Example

“Every value of `x` bound while calling `foo` is less than the length of every array bound to `a`.”

Example: Proposition

Example

"Every value of `x` bound while calling `foo`
is less than the length of every array bound to `a`."

In longhand:

$$\begin{array}{l}(\texttt{forall}\ x : (\texttt{x}, \widehat{t}_{\texttt{foo}})\\ \quad(\texttt{forall}\ a : (\texttt{a}, \top)\\ \qquad(\texttt{<}\ x\ (\texttt{alen}\ a))))\end{array}$$

Example: Proposition

Example

"Every value of `x` bound while calling `foo` is less than the length of every array bound to `a`."

In longhand:

$$\texttt{(forall}\ x : (\texttt{x}, \widehat{t}_{\texttt{foo}})$$
$$\texttt{(forall}\ a : (\texttt{a}, \top)$$
$$\texttt{(<}\ x\ \texttt{(alen}\ a\texttt{))))}$$

Or, in convenient (but incomplete) shorthand:

$$\texttt{(<}\ (\texttt{x}, \widehat{t}_{\texttt{foo}})\ \texttt{(alen}\ (\texttt{a}, \top)\texttt{))}$$

Logic syntax

Features

- S-Expressions.
- Just `or`, `not`.
- Relations encoded as functions.

Logic semantics

Question

How do we know when proposition ϕ is true for state ς?

Logic semantics

Question

How do we know when proposition ϕ is true for state ς?

Answer

When $\varsigma \models \phi$ holds.

Logic semantics

Question

How do we know when proposition ϕ is true for state ς?

Answer

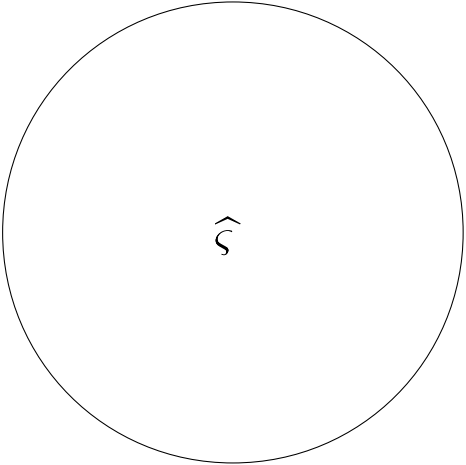
When $\varsigma \models \phi$ holds.

Means exactly what you think it means.

Filtered concretization

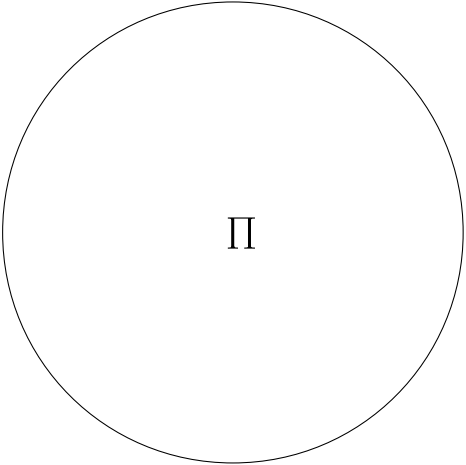
Set of conrete states ($State$)

Filtered concretization



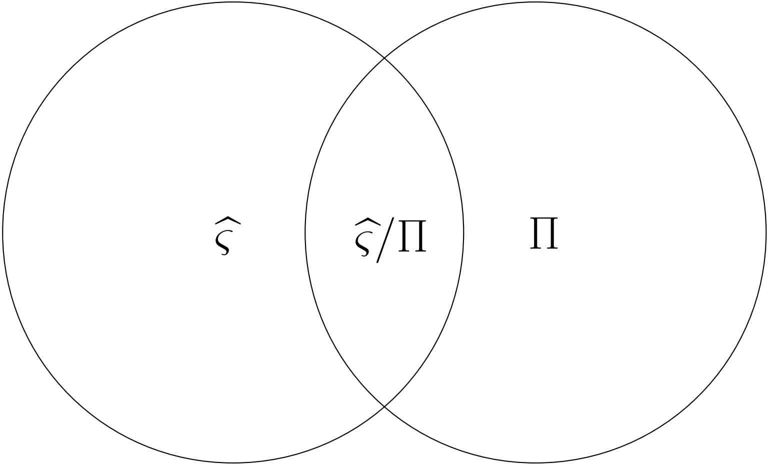
Set of conrete states ($State$) $\{\varsigma : |\varsigma| \sqsubseteq \widehat{\varsigma} \qquad \}$

Filtered concretization



Set of conrete states ($State$) $\{\varsigma : \quad \varsigma \models \Pi\}$

Filtered concretization



Set of conrete states ($State$) $\{\varsigma : |\varsigma| \sqsubseteq \widehat{\varsigma}$ and $\varsigma \models \Pi\}$

Deriving new propositions

Example

If $\varsigma \models (= x\ y)$
and $\varsigma \models (= y\ z)$,
does $\varsigma \models (= x\ z)$ hold?

Deriving new propositions

Example

If $\varsigma \models \texttt{(}= x\ y\texttt{)}$
and $\varsigma \models \texttt{(}= y\ z\texttt{)}$,
does $\varsigma \models \texttt{(}= x\ z\texttt{)}$ hold?

Answer

Yes, if $\{\texttt{(}= x\ y\texttt{)}, \texttt{(}= y\ z\texttt{)}\} \vdash \texttt{(}= x\ z\texttt{)}$ holds.

Deriving new propositions

Example

If $\varsigma \models (= x\ y)$
and $\varsigma \models (= y\ z)$,
does $\varsigma \models (= x\ z)$ hold?

Answer

Yes, if $\{(= x\ y), (= y\ z)\} \vdash (= x\ z)$ holds.

(Assm) $\dfrac{\psi \in \Pi}{\Pi \vdash \psi}$

($\vee$Ant) $\dfrac{\Pi \cup \{\phi_1\} \vdash \phi_3 \quad \Pi \cup \{\phi_2\} \vdash \phi_3}{\Pi \cup \{(\texttt{or}\ \phi_1\ \phi_2)\} \vdash \phi_3}$

(Subst) $\dfrac{\Pi \vdash (= \iota\ \iota') \quad \Pi \vdash \psi[\iota/x]}{\Pi \vdash \psi[\iota'/x]}$

(Ant) $\dfrac{\Pi \vdash \phi \quad \Pi \subseteq \Pi'}{\Pi' \vdash \phi}$

(Cases) $\dfrac{\Pi \cup \{\phi_1\} \vdash \phi_2 \quad \Pi \cup \{(\texttt{not}\ \phi_1)\} \vdash \phi_2}{\Pi \vdash \phi_2}$

(Contr) $\dfrac{\Pi \cup \{(\texttt{not}\ \phi_1)\} \vdash \phi_2 \quad \Pi \cup \{(\texttt{not}\ \phi_1)\} \vdash (\texttt{not}\ \phi_2)}{\Pi \vdash \phi_1}$

(Eq) $\dfrac{}{\Pi \vdash (= \iota\ \iota)}$

($\vee$Cons) $\dfrac{\Pi \vdash \phi_1}{\Pi \vdash (\texttt{or}\ \phi_1\ \phi_2), (\texttt{or}\ \phi_2\ \phi_1)}$

(Int) $\dfrac{\Pi \vdash (\texttt{forall}\ \boldsymbol{x} : \widehat{\boldsymbol{\iota}}\ \phi) \quad \{\phi\} \vdash \phi'}{\Pi \vdash (\texttt{forall}\ \boldsymbol{x} : \widehat{\boldsymbol{\iota}}\ (\texttt{and}\ \phi\ \phi'))}$

($\forall$Intro) $\dfrac{\Pi \vdash \psi \quad x \notin \mathit{free}(\psi)}{\Pi \vdash (\texttt{forall}\ x : \widehat{\iota}\ \psi)}$

($\forall$Swap) $\dfrac{\Pi \vdash (\texttt{forall}\ \langle x_1, x_2\rangle : \langle \widehat{\iota}_1, \widehat{\iota}_2\rangle\ \psi)}{\Pi \vdash (\texttt{forall}\ \langle x_2, x_1\rangle : \langle \widehat{\iota}_2, \widehat{\iota}_1\rangle\ \psi)}$

Trusting the theorem prover

Summary

- $\models$: What a proposition means.
- $\vdash$: What a proposition implies.

Trusting the theorem prover

Summary

- $\models$: What a proposition means.
- $\vdash$: What a proposition implies.

Question

How can we trust an external theorem prover?

Trusting the theorem prover

Summary

- $\models$: What a proposition means.
- $\vdash$: What a proposition implies.

Question

How can we trust an external theorem prover?

Theorem (Syntactic soundness)

If $\Pi \vdash \phi$ holds, then $\Pi \models \phi$ holds.

All together now

Woven state

Example (Machine, $\widehat{\varsigma}$)

call site `(f x k)`

local env

$$\begin{aligned} \mathtt{f} &\mapsto \widehat{t}_{\mathtt{foo}} \\ \mathtt{k} &\mapsto \widehat{t}_{\mathtt{foo}} \\ \mathtt{x} &\mapsto \widehat{t}_{\mathtt{foo}} \end{aligned}$$

global env

$$\begin{aligned} (\mathtt{f}, \widehat{t}_{\mathtt{foo}}) &\mapsto \cdots \\ (\mathtt{k}, \widehat{t}_{\mathtt{foo}}) &\mapsto \cdots \\ (\mathtt{x}, \widehat{t}_{\mathtt{foo}}) &\mapsto \mathit{positive} \\ (\mathtt{z}, \widehat{t}_{\mathtt{bar}}) &\mapsto \mathit{positive} \end{aligned}$$

time $\widehat{t}_{\mathtt{f}}$

Example (Assumptions, Π)

$$\begin{aligned} &(\mathtt{forall}\ x : (\mathtt{x}, \widehat{t}_{\mathtt{foo}}) \\ &\quad (\mathtt{forall}\ z : (\mathtt{z}, \widehat{t}_{\mathtt{bar}}) \\ &\qquad (\mathtt{<}\ x\ z))) \end{aligned}$$

Woven state

Example (Machine, $\widehat{\varsigma}$)

call site $(\mathtt{f\ x\ k})$

local env
$\mathtt{f} \mapsto \widehat{t}_{\mathtt{foo}}$
$\mathtt{k} \mapsto \widehat{t}_{\mathtt{foo}}$
$\mathtt{x} \mapsto \widehat{t}_{\mathtt{foo}}$

global env
$(\mathtt{f}, \widehat{t}_{\mathtt{foo}}) \mapsto \cdots$
$(\mathtt{k}, \widehat{t}_{\mathtt{foo}}) \mapsto \cdots$
$(\mathtt{x}, \widehat{t}_{\mathtt{foo}}) \mapsto \mathit{positive}$
$(\mathtt{z}, \widehat{t}_{\mathtt{bar}}) \mapsto \mathit{positive}$

time $\widehat{t}_{\mathtt{f}}$

Example (Assumptions, Π)

$(\mathtt{forall}\ x : (\mathtt{x}, \widehat{t}_{\mathtt{foo}})$
$\quad(\mathtt{forall}\ z : (\mathtt{z}, \widehat{t}_{\mathtt{bar}})$
$\quad\quad(\mathtt{<}\ x\ z)))$

Woven transition relation

Old machine state

New machine state

$$(\widehat{\varsigma}, \Pi) \mid\!\!\Rightarrow (\widehat{\varsigma}', \Pi')$$

Old assumption base

New assumption base

Example: Transition

Example

call site `(f x k)`

time $\widehat{t}_{\mathtt{f}}$

Example: Transition

Example

call site	`(f x k)`
local env	$\mathtt{f} \mapsto \widehat{t}_{\mathtt{foo}}$
time	$\widehat{t}_{\mathtt{f}}$

Example: Transition

Example

call site `(f x k)`

local env $\mathtt{f} \mapsto \widehat{t}_{\mathtt{foo}}$

global env $(\mathtt{f}, \widehat{t}_{\mathtt{foo}}) \mapsto$ a closure over `(`λ` (a q) ...)`

time $\widehat{t}_{\mathtt{f}}$

Example: Transition

Example

call site	`(f x k)`
local env	$\texttt{f} \mapsto \widehat{t}_{\texttt{foo}}$ $\texttt{x} \mapsto \widehat{t}_{\texttt{foo}}$
global env	$(\texttt{f}, \widehat{t}_{\texttt{foo}}) \mapsto$ a closure over $(\lambda$ `(a q) ...)`
time	$\widehat{t}_{\texttt{f}}$

Example: Transition

Example

call site	`(f x k)`
local env	$\texttt{f} \mapsto \widehat{t}_{\texttt{foo}}$ $\texttt{x} \mapsto \widehat{t}_{\texttt{foo}}$
global env	$(\texttt{f}, \widehat{t}_{\texttt{foo}}) \mapsto$ a closure over $\texttt{(}\lambda\ \texttt{(a q) ...)}$
time	$\widehat{t}_{\texttt{f}}$

New fact?

$$\texttt{(forall } \langle x, a \rangle \texttt{ : } \langle (\texttt{x}, \widehat{t}_{\texttt{foo}}), (\texttt{a}, \widehat{t}_{\texttt{f}}) \rangle \texttt{ (} = x\ a\texttt{))}$$

It depends.

Chaining equal values

Candidate for Π'

$$\phi = \texttt{(forall}\ \langle x, a\rangle\ \texttt{:}\ \langle(\texttt{x}, \widehat{t_{\texttt{foo}}}), (\texttt{a}, \widehat{t_{\texttt{f}}})\rangle\ \texttt{(}{=}\ x\ a\texttt{))}$$

Prerequisites

Can add it if $\Pi \vdash \phi$.

Chicken and egg

How can ϕ be in there already?

ΓCFA: Abstract counting

Idea

Keep count of concrete counterparts to abstract identities.

ΓCFA: Abstract counting

Idea

Keep count of concrete counterparts to abstract identities.

Mechanism

- Add counter to every abstract machine state.
- Counter maps each binding to times allocated.
- Stop counting after 1.

ΓCFA: Abstract counting

Idea

Keep count of concrete counterparts to abstract identities.

Mechanism

- Add counter to every abstract machine state.
- Counter maps each binding to times allocated.
- Stop counting after 1.

Theorem

If $\{\text{binding}_1\} = \{\text{binding}_2\}$,
then $\text{binding}_1 = \text{binding}_2$.

Chaining equal values

Candidate for Π'

$$\phi = \texttt{(forall}\ \langle x, a\rangle\ \texttt{:}\ \langle(\texttt{x}, \widehat{t}_{\texttt{foo}}), (\texttt{a}, \widehat{t}_{\texttt{f}})\rangle\ \texttt{(}= x\ a\texttt{))}$$

Prerequisites

- Can add it if $\Pi \vdash \phi$.
- Or, if count of $(\texttt{x}, \widehat{t}_{\texttt{foo}})$ is 1 and count of $(\texttt{a}, \widehat{t}_{\texttt{f}})$ is 0.

ΓCFA: Abstract garbage collection

Idea

Discard unreachable bindings.

ΓCFA: Abstract garbage collection

Idea

Discard unreachable bindings.

Mechanism

- Start with bindings touched by current state.
- Take transitive closure.
- Can reset unreachable bindings' counts to 0.

Chaining equal values

Candidate for Π'

$$\phi = \texttt{(forall}\ \langle x, a\rangle\ \texttt{:}\ \langle(\texttt{x}, \widehat{t}_{\texttt{foo}}), (\texttt{a}, \widehat{t}_{\texttt{f}})\rangle\ \texttt{(}= x\ a\texttt{))}$$

Prerequisites

- Can add it if $\Pi \vdash \phi$.
- Or, if count of $(\texttt{x}, \widehat{t}_{\texttt{foo}})$ is 1 and count of $(\texttt{a}, \widehat{t}_{\texttt{f}})$ is 0.
- Or, if count of $(\texttt{x}, \widehat{t}_{\texttt{foo}})$ is 1 and $(\texttt{a}, \widehat{t}_{\texttt{f}})$ is unreachable.
- (More in paper.)

Example: Invertible rebinding

Example

call site	`(f (+ x 1) k)`
local env	$\mathtt{f} \mapsto \widehat{t}_{\mathtt{foo}}$ $\mathtt{x} \mapsto \widehat{t}_{\mathtt{f}}$
global env	$(\mathtt{f}, \widehat{t}_{\mathtt{foo}}) \mapsto$ a closure over `(`λ `(x q) ...)`
time	$\widehat{t}_{\mathtt{f}}$

Updating assumption base

Can replace $(\mathtt{x}, \widehat{t}_{\mathtt{f}})$ with `(-` $(\mathtt{x}, \widehat{t}_{\mathtt{f}})$ `1)` in Π?

Example: Invertible rebinding

Example

call site	`(f (+ x 1) k)`
local env	$\mathtt{f} \mapsto \widehat{t}_{\mathtt{foo}}$ $\mathtt{x} \mapsto \widehat{t}_{\mathtt{f}}$
global env	$(\mathtt{f}, \widehat{t}_{\mathtt{foo}}) \mapsto$ a closure over $\mathtt{(}\lambda\ \mathtt{(x\ q)\ \ldots)}$
time	$\widehat{t}_{\mathtt{f}}$

Updating assumption base

Can replace $(\mathtt{x}, \widehat{t}_{\mathtt{f}})$ with $\mathtt{(-}\ (\mathtt{x}, \widehat{t}_{\mathtt{f}})\ \mathtt{1)}$ in Π?

Yes, if $(\mathtt{x}, \widehat{t}_{\mathtt{f}})$ is unreachable and its count is 1.

(*E.g.* tail recursion, `for` loops.)

(More on this in the paper.)

Example: Conditional

Example

call site `(if (< i (alen a)) ...  ...)`

Example: Conditional

Example

call site `(if (< i (alen a)) ...  ...)`

Case 1

Π can (dis)prove `(< i (alen a))`. Branch one way.

$$
\begin{tikzcd}
 & \widehat{\varsigma}_{true} \\
\widehat{\varsigma} \arrow[dr] & \\
 & \widehat{\varsigma}_{false}
\end{tikzcd}
$$

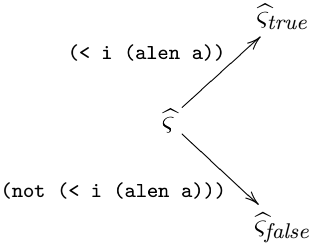
Example: Conditional

Example

call site `(if (< i (alen a)) ...  ...)`

Case 2

`(< i (alen a))` has one counterpart. Branch both ways & assert.



Example: Conditional

Example

call site `(if (< i (alen a)) ...  ...)`

Case 3

None of the above. Branch both ways. Don't touch Π'.

$$\begin{tikzcd}
 & \widehat{\varsigma}_{true} \\
\widehat{\varsigma} \arrow[ur] \arrow[dr] & \\
 & \widehat{\varsigma}_{false}
\end{tikzcd}$$

Walkthrough: Simple `for` loop

Example

```
for (i = 0; i < a.length; i++)
 print(a[i]) ;
```

Example (CPS)

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Parameters

- 0CFA contour set. (Bindings = Variables.)

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

```
(< 0 (alen a))
```

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

```
(< 0 (alen a))
```

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 i)`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 i)`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 i)`

Safe

$0 \leq$ `i` $<$ `(alen a)` holds!

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1)))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 i)`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 i)`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(= 0 (- i 1))`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 i)`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 i)`, `(< i (alen a))`

Safe

$0 \leq$ `i` $<$ `(alen a)` holds!

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1)))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 i)`, `(< i (alen a))`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 i)`, `(< i (alen a))`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 (- i 1))`, `(< (- i 1) (alen a))`

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                (if (< i (alen a))
                 (print (aget a i) (λ ()
                  (loop (+ i 1))))
                 ...))))
  (loop 0))
```

Assumption base, Π

`(< 0 (alen a))`, `(≤ 0 i)`, `(< (- i 1) (alen a))`, `(< i (alen a))`

Safe

$0 \leq$ `i` $<$ `(alen a)` holds!

Walkthrough: Simple `for` loop

```
(letrec ((loop (λ (i)
                 (if (< i (alen a))
                  (print (aget a i) (λ ()
                   (loop (+ i 1))))
                  ...))))
  (loop 0))
```

Assumption base, Π

(< 0 (alen a)), (≤ 0 i), (< i (alen a))

Finished

State already visited.

More in the paper

- Formal treatment.
- Flow analysis as oracle inference rules.
- More rules for assumption base management.
- Rules for handling arrays.
- Three-page worked example for vertex arrays.

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)
- Shivers. (Control-flow analysis w/ factored environment)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)
- Shivers. (Control-flow analysis w/ factored environment)
- Hudak. (Abstract reference counting)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)
- Shivers. (Control-flow analysis w/ factored environment)
- Hudak. (Abstract reference counting)
- Nanevski & Morrisett. (Soundness of theorem prover interaction)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)
- Shivers. (Control-flow analysis w/ factored environment)
- Hudak. (Abstract reference counting)
- Nanevski & Morrisett. (Soundness of theorem prover interaction)
- Ball. (Predicate abstraction)

Related & inspiring work

- Cousot & Cousot. (Abstract interpretation)
- Cousot & Cousot. (Invariant synthesis)
- Cousot & Cousot. (Widening & narrowing)
- Cousot & Cousot. (Relational abstract domains)
- Shivers. (Control-flow analysis w/ factored environment)
- Hudak. (Abstract reference counting)
- Nanevski & Morrisett. (Soundness of theorem prover interaction)
- Ball. (Predicate abstraction)
- ...

Related & future work

$$
\begin{tikzcd}
 & \Delta\text{CFA} \arrow[d] \\
 & \text{LFA} \\
\Gamma\text{CFA} \arrow[ur] &
\end{tikzcd}
$$

Related & future work

$$
\begin{tikzcd}
 & \Delta\text{CFA} \arrow[d] \arrow[ddl, leftrightarrow] \\
 & \text{LFA} \\
\Gamma\text{CFA} \arrow[ur] &
\end{tikzcd}
$$

Related & future work

$$\begin{tikzcd} & \Delta\text{CFA} \arrow[d] & \\ & \text{LFA} & \\ \Gamma\text{CFA} \arrow[ur] & & \Theta\text{CFA} \arrow[ul] \end{tikzcd}$$

Conclusion

HOFA + FOL = LFA

Conclusion

HOFA + FOL = LFA

Merci

Question

What are some other applications of LFA?

Answer

- Improving flow precision.
- Static checks of `assert` statements.
- Pre- and post-condition checks.

Question

Do you have an implementation?

Answer

Half of one:

- Modified ACL/2 for theorem prover.
- Modified ΓCFA.

Goal: Meet in the middle.

Question

Does a backward version exist?

Answer

- Not yet.
- Start by widening into constraint-solving form.

Question

What is the complexity of LFA?

Answer

Exponential in theory.
Usually much friendlier in pratice.
It depends on...
...the contour set.
...the degree of widening to assumption base & abstract heap.

Question

Do you support floats?

Answer

Patrick, not yet.