The Big Picture
The Big Picture

Program

Static Analysis
The Big Picture

A

Program

Static Analysis
The Big Picture

A fine/expensive Program

Static Analysis

precise
The Big Picture

A

fine/expensive

coarse/cheap

Program

Static Analysis

imprecise
The Big Picture

A fine/expensive
coarse/cheap
Program
Static
Analysis
precise
selected refinement
[Heintze & Tardieu 2001]
[Guyer & Lin 2003]
[Sridharan et al. 2005]
[Zheng & Rugina 2008]
[Liang et al. 2011]
The Big Picture

selected refinement

[Heintze & Tardieu 2001]
[Guyer & Lin 2003]
[Sridharan et al. 2005]
[Zheng & Rugina 2008]
[Liang et al. 2011]
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
        return v
}

i1: x = getnew()
i2: y = getnew()
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)
An Example of Pruning

getnew() {
    h1:  u = new C
    h2:  v = new C
        return v
}

i1:  x = getnew()
i2:  y = getnew()

Query: do x and y alias? (no)

0-CFA:

u = new C
v = new C
x = getnew()
y = getnew()
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()
Query: do x and y alias? (no)

0-CFA:
u = new C  v = new C
u → h1  x = getnew()  y = getnew()
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)

0-CFA:
u = new C v = new C
u → h1 x = getnew() v → h2 y = getnew()
An Example of Pruning

```
getnew() {
    h1: u = new C
    h2: v = new C
        return v
}
i1: x = getnew()
i2: y = getnew()
```

**Query**: do x and y alias? *(no)*

**0-CFA**:

- `u = new C`
  - `u → h1`
  - `x = getnew()`
  - `x → h2`
- `v = new C`
  - `v → h2`
- `y = getnew()`
An Example of Pruning

getnew() {
  h1: u = new C
  h2: v = new C
  return v
}
i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)

0-CFA:

u = new C
u → h1
x = getnew()
x → h2
v = new C
v → h2
y = getnew()
y → h2
An Example of Pruning

```
getnew() {
  h1: u = new C
  h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()
Query: do x and y alias? (no)
```

0-CFA:
```
u = new C v = new C
u → h1 x = getnew() v → h2 y = getnew()
x → h2 y → h2
ALIAS(x,y)
```
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()

Query: do \(x\) and \(y\) alias? (no)

0-CFA:
- \(u = \text{new } C\)
- \(v = \text{new } C\)
- \(u \rightarrow h1\)
- \(x = \text{getnew}()\)
- \(v \rightarrow h2\)
- \(y = \text{getnew}()\)
- \(x \rightarrow h2\)
- \(y \rightarrow h2\)
- \(\text{ALIAS}(x, y)\)

irrelevant
An Example of Pruning

```plaintext
getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)
```

1-CFA on pruned:

```
u = new C v:i1 = new C v:i2 = new C
u → h1 x = getnew() y = getnew() irrelevant
```

```plaintext
irrelevant → h1 x = getnew() y = getnew()
```
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}

i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)

1-CFA on pruned:

u = new C

v:i1 = new C  v:i2 = new C

x = getnew()  v:i1 → h2:i1  v:i2 → h2:i2  y = getnew()

irrelevant
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}
i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)

1-CFA on pruned:

u = new C

irrelevant

\[ u \to h1 \]

x = getnew()

\[ v:i1 = \text{new C} \quad v:i2 = \text{new C} \]

\[ v:i1 \to h2:i1 \quad v:i2 \to h2:i2 \]

y = getnew()

\[ y \to h2:i2 \]

x \to h2:i1
An Example of Pruning

getnew() {
    h1: u = new C
    h2: v = new C
    return v
}

i1: x = getnew()
i2: y = getnew()

Query: do x and y alias? (no)

1-CFA on pruned:

u = new C

x = getnew()

v:i1 = new C
v:i2 = new C

v:i1 → h2:i1
v:i2 → h2:i2

x → h2:i1
y → h2:i2

(not aliasing - query proven)
General Pruning Framework

Input tuples $A_0$

$v = \text{new } C \ldots$
General Pruning Framework

Input tuples $A_0$

Query tuple $x_Q$

$v = \text{new } C \ldots$

$\text{ALIAS}(x, y)$
General Pruning Framework

Input tuples $A_0$

Query tuple $x_Q$

Datalog rules

\[ v = \text{new } C \quad \cdots \]

\[ \text{ALIAS}(x, y) \]

\[ v_2 \rightarrow h \iff v_2 = v_1, v_1 \rightarrow h \quad \cdots \]
General Pruning Framework

Input tuples $A_0$

Query tuple $x_Q$

Datalog rules

Prune/prove operator $P$

$A_0$ \xrightarrow{P} \text{subset of } A_0 \text{ used to derive } x_Q
General Pruning Framework

Input tuples $A_0$

$\begin{array}{l}
 v = \text{new } C \\
\end{array}$

Query tuple $x_Q$

$\begin{array}{l}
 \text{ALIAS}(x,y)
\end{array}$

Datalog rules

$\begin{array}{l}
 v_2 \rightarrow h \leftarrow v_2 = v_1 , \ v_1 \rightarrow h
\end{array}$

$\cdots$

Prune/prove operator $P$

$A_0 \xrightarrow{P} \text{subset of } A_0 \text{ used to derive } x_Q$

$\begin{array}{l}
 A_0 \\
\end{array}$

$\begin{array}{l}
 P(A_0)
\end{array}$

$\begin{array}{l}
 x_Q
\end{array}$
General Pruning Framework

Input tuples $A_0$

Query tuple $x_Q$

Datalog rules

Prune/prove operator $P$

$A_0$ $\rightarrow$ subset of $A_0$ used to derive $x_Q$

Query proven $\iff P(A_0) = \emptyset$
Prune and Refine

(abstract tuples) $A_0 \quad u = \text{new } C \quad v = \text{new } C \quad \ldots$
Prune and Refine

(abstract tuples) $A_0 \quad u = \text{new } C \quad v = \text{new } C \quad \cdots$

Prune $P \quad \text{run 0-CFA}$
Prune and Refine

(abstract tuples) $A_0$ $\quad u = \text{new } C \quad v = \text{new } C \quad \cdots$

Prune $\quad P$

(relevant tuples) $\tilde{A}_0$ $\quad u = \text{new } C \quad v = \text{new } C$

run 0-CFA
Prune and Refine

(abstract tuples) \[ A_0 \quad u = \text{new } C \quad v = \text{new } C \quad \cdots \]

Prune \[ P \quad \text{run 0-CFA} \]

(relevant tuples) \[ \tilde{A}_0 \quad u = \text{new } C \quad v = \text{new } C \]

Refine \[ \alpha_1 \]
Prune and Refine

(abstract tuples) $A_0 \quad u = \text{new } C \quad v = \text{new } C \quad \cdots$

Prune $P$ run 0-CFA

(relevant tuples) $\tilde{A}_0 \quad u = \text{new } C \quad v = \text{new } C$

Refine $\alpha_1$

(refined tuples) $A_1 \quad v:i1 = \text{new } C \quad v:i2 = \text{new } C$
Prune and Refine

(abstract tuples) \( A_0 \) \( u = \text{new } C \) \( v = \text{new } C \) \( \ldots \)

Prune

\( P \) run 0-CFA

(relevant tuples) \( \tilde{A}_0 \) \( u = \text{new } C \) \( v = \text{new } C \)

Refine

\( \alpha_1 \)

(refined tuples) \( A_1 \) \( v:i1 = \text{new } C \) \( v:i2 = \text{new } C \)

Prune

\( P \) run 1-CFA
Prune and Refine

(abstract tuples) $A_0 = \{ u = \text{new } C, v = \text{new } C \cdots \}$

Prune $P \quad \text{run 0-CFA}$

(relevant tuples) $\tilde{A}_0 = \{ u = \text{new } C, v = \text{new } C \}$

Refine $\alpha_1$

(refined tuples) $A_1 = \{ v.i1 = \text{new } C, v.i2 = \text{new } C \}$

Prune $P \quad \text{run 1-CFA}$

$\emptyset \quad \Rightarrow \text{query is proven}$
Prune-Refine Algorithm

Input:

Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$
Prune-Refine Algorithm

Input:
- Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$
- $A_0$, initial set of tuples
Prune-Refine Algorithm

Input:
Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$
$A_0$, initial set of tuples

For $t = 0, 1, 2, \ldots$:

**Prune:** $\tilde{A}_t = P(A_t)$. If $\tilde{A}_t = \emptyset$: return proven.

**Refine:** $A_{t+1} = \alpha_{t+1}(\tilde{A}_t)$. 
Prune-Refine Algorithm

Input:

Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$

$A_0$, initial set of tuples

For $t = 0, 1, 2, \ldots$:

- **Prune**: $\tilde{A}_t = P(A_t)$. If $\tilde{A}_t = \emptyset$: return **proven**.
- **Refine**: $A_{t+1} = \alpha_{t+1}(\tilde{A}_t)$.

**Main Result:**

Prune-Refine after $t$ iterations
Prune-Refine Algorithm

Input:

Sequence of abstractions: \( \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \)
\( A_0 \), initial set of tuples

For \( t = 0, 1, 2, \ldots \):

- **Prune:** \( \tilde{A}_t = P(A_t) \). If \( \tilde{A}_t = \emptyset \): return proven.
- **Refine:** \( A_{t+1} = \alpha_{t+1}(\tilde{A}_t) \).

Main Result:

- Prune-Refine after \( t \) iterations
- Full Analysis on \( \alpha_t \)
Prune-Refine Algorithm

Input:

Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$

$A_0$, initial set of tuples

For $t = 0, 1, 2, \ldots$:

Prune: $\tilde{A}_t = P(A_t)$. If $\tilde{A}_t = \emptyset$: return proven.

Refine: $A_{t+1} = \alpha_{t+1}(\tilde{A}_t)$.

Main Result:

Prune-Refine after $t$ iterations $\equiv$ Full Analysis on $\alpha_t$
Prune-Refine Algorithm

Input:

Sequence of abstractions: $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$

$A_0$, initial set of tuples

For $t = 0, 1, 2, \ldots$:

Prune: $\tilde{A}_t = P(A_t)$. If $\tilde{A}_t = \emptyset$: return proven.

Refine: $A_{t+1} = \alpha_{t+1}(\tilde{A}_t)$.

Main Result:

Prune-Refine after $t$ iterations $= \text{Full Analysis on } \alpha_t$

fast $\quad = \quad$ Full Analysis on $\alpha_t$

slow
Rest of Talk

Pre-Pruning Extension

Experiments
Pre-Pruning

$\tilde{A}_{t-1}$

$v = \text{new } C$
Pre-Pruning

\[ \tilde{A}_{t-1} \xrightarrow{\text{Refine } \alpha_t} A_t \]

\[ v = \text{new C} \]

\[ \begin{align*}
A_t \\
v: h0 &= \text{new C} \\
v: h1 &= \text{new C} \\
v: h2 &= \text{new C} \\
v: h3 &= \text{new C} \\
v: h4 &= \text{new C} \\
v: h5 &= \text{new C}
\end{align*} \]
Pre-Pruning

\[ \tilde{A}_{t-1} \rightarrow \text{Refine } \alpha_t \rightarrow A_t \rightarrow \text{Prune } P \rightarrow \text{expensive!} \]

\[ v = \text{new } C \]

\[ v: h_0 = \text{new } C \]
\[ v: h_1 = \text{new } C \]
\[ v: h_2 = \text{new } C \]
\[ v: h_3 = \text{new } C \]
\[ v: h_4 = \text{new } C \]
\[ v: h_5 = \text{new } C \]
Pre-Pruning

\[ \tilde{A}_{t-1} \]

\( v = \text{new C} \)

Refine \( \alpha_t \)

\[ A_t \]

\( v: h0 = \text{new C} \)

\( v: h1 = \text{new C} \)

\( v: h2 = \text{new C} \)

\( v: h3 = \text{new C} \)

\( v: h4 = \text{new C} \)

\( v: h5 = \text{new C} \)

Prune \( P \)

\( \beta_t \preceq \alpha_t \)

\[ B_t \]

\( v: t0 = \text{new C} \)

\( v: t1 = \text{new C} \)
Pre-Pruning

$\tilde{A}_{t-1}$
$v = \text{new C}$

Refine $\alpha_t$

$A_t$
$v:h_0 = \text{new C}$
$v:h_1 = \text{new C}$
$v:h_2 = \text{new C}$
$v:h_3 = \text{new C}$
$v:h_4 = \text{new C}$
$v:h_5 = \text{new C}$

Prune $P$
$v:t_0 = \text{new C}$
$v:t_1 = \text{new C}$

$\tilde{B}_t$
$v:t_0 = \text{new C}$
$v:t_1 = \text{new C}$
Pre-Pruning

\[ \tilde{A}_{t-1} = v = \text{new C} \]

Refine \( \alpha_t \)

\[ A_t \]

\[ v: h0 = \text{new C} \]
\[ v: h1 = \text{new C} \]
\[ v: h2 = \text{new C} \]
\[ v: h3 = \text{new C} \]
\[ v: h4 = \text{new C} \]
\[ v: h5 = \text{new C} \]

Prune \( P \)

\[ \beta_t \leq \alpha_t \]

\( \alpha_t \)

\[ B_t \]

\[ v: t0 = \text{new C} \]
\[ v: t1 = \text{new C} \]

Prune \( P \)

\[ \tilde{B}_t \]

\[ v: t0 = \text{new C} \]
\[ v: t1 = \text{new C} \]
Pre-Pruning

\[ \tilde{A}_{t-1} \]

\[ v = \text{new C} \]

Refine \( \alpha_t \)

\[ \begin{align*}
A_t & \\
v: h0 & = \text{new C} \\
v: h1 & = \text{new C} \\
v: h2 & = \text{new C} \\
v: h3 & = \text{new C} \\
v: h4 & = \text{new C} \\
v: h5 & = \text{new C}
\end{align*} \]

Prune \( P \)

\[ \text{cheaper} \]

\[ \alpha_t \]

\[ \beta_t \leq \alpha_t \]

\[ B_t \]

\[ \begin{align*}
v: t0 & = \text{new C} \\
v: t1 & = \text{new C}
\end{align*} \]

Prune \( P \)

\[ \tilde{B}_t \]

\[ \begin{align*}
v: t0 & = \text{new C} \\
v: t1 & = \text{new C}
\end{align*} \]
Which Abstractions for Pre-Pruning?

(main) \( \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \)

(auxiliary) \( \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \)
Which Abstractions for Pre-Pruning?

\[(\text{main}) \quad \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \]

\[\gamma| \quad \gamma|\quad \gamma|\]

\[(\text{auxiliary}) \quad \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \]

Choose abstraction \(\tau\);
Which Abstractions for Pre-Pruning?

(main) \[ \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \]

(auxiliary) \[ \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \]

Choose abstraction \( \tau \); set \( \beta_t = \alpha_t \circ \tau \)
Which Abstractions for Pre-Pruning?

(main) $\alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots$

(auxiliary) $\beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots$

Choose abstraction $\tau$; set $\beta_t = \alpha_t \circ \tau$

amount pre-pruned

\[ \text{cost} \]
Which Abstractions for Pre-Pruning?

(main) \( \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \)

(auxiliary) \( \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \)

Choose abstraction \( \tau \); set \( \beta_t = \alpha_t \circ \tau \)

\( \tau = \lambda x. x \) (no abstraction)
Which Abstractions for Pre-Pruning?

(main) \( \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \)

(auxiliary) \( \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \)

Choose abstraction \( \tau \); set \( \beta_t = \alpha_t \circ \tau \)

\( \tau = \lambda x.x \) (no abstraction)

\( \tau = \lambda x.\bot \) (total abstraction)
Which Abstractions for Pre-Pruning?

(main) \( \alpha_0 \preceq \alpha_1 \preceq \alpha_2 \preceq \cdots \)

(auxiliary) \( \beta_0 \preceq \beta_1 \preceq \beta_2 \preceq \cdots \)

Choose abstraction \( \tau \); set \( \beta_t = \alpha_t \circ \tau \)

\( \tau = \lambda x.x \) (no abstraction)

\( \tau = \lambda x.\perp \) (total abstraction)

\( \tau \) (complementary to \( \alpha_t \))
Type-Based Abstractions for Pre-Pruning

$k$-limited: $\alpha_k = \text{take length } k \text{ prefix}$

$v:h5:h8 = \text{new } C\at \alpha_1 \rightarrow v:h5 = \text{new } C$
Type-Based Abstractions for Pre-Pruning

$k$-limited: $\alpha_k = \text{take length } k \text{ prefix}

\[
\begin{align*}
v:h5:h8 &= \text{new } C \\
\alpha_1 &\quad \Rightarrow \quad v:h5 = \text{new } C
\end{align*}
\]

Type-based: $\tau = \text{replace alloc. sites with types}$ [Smaragdakis et al. 2011]

\[
\begin{align*}
v:h5:h8 &= \text{new } C \\
\tau &\quad \Rightarrow \quad v:t1:t0 = \text{new } C
\end{align*}
\]
Type-Based Abstractions for Pre-Pruning

$k$-limited: $\alpha_k = \text{take length } k \text{ prefix}$

$\alpha_1$

$v:h5:h8 = \text{new C} \xrightarrow{\alpha_1} v:h5 = \text{new C}$

Type-based: $\tau = \text{replace alloc. sites with types}$ [Smaragdakis et al. 2011]

$v:h5:h8 = \text{new C} \xrightarrow{\tau} v:t1:t0 = \text{new C}$

We use $\tau = \text{type of containing class } \times \text{type of allocation site}$
Type-Based Abstractions for Pre-Pruning

\( k\)-limited: \( \alpha_k = \text{take length } k \text{ prefix} \)

\[
\begin{align*}
  v &: h5 : h8 &= \text{new } C \\
  \alpha_1 &
\end{align*}
\]

\[
\begin{array}{c}
  v &: h5 = \text{new } C
\end{array}
\]

Type-based: \( \tau = \text{replace alloc. sites with types} \) [Smaragdakis et al. 2011]

\[
\begin{align*}
  v &: h5 : h8 &= \text{new } C \\
  \tau &
\end{align*}
\]

\[
\begin{array}{c}
  v &: t1 : t0 = \text{new } C
\end{array}
\]

We use \( \tau = \text{type of containing class} \times \text{type of allocation site} \)

```java
class C1 {
    h1: x = new C2
}
```
Type-Based Abstractions for Pre-Pruning

$k$-limited: $\alpha_k = \text{take length } k \text{ prefix}$

$$v: h5: h8 = \text{new } C \quad \alpha_1 \quad \quad v: h5 = \text{new } C$$

Type-based: $\tau = \text{replace alloc. sites with types}$ [Smaragdakis et al. 2011]

$$v: h5: h8 = \text{new } C \quad \tau \quad \quad v: t_1: t_0 = \text{new } C$$

We use $\tau = \text{type of containing class } \times \text{type of allocation site}$

```java
class C1 {
    h1: x = new C2
}
```

$$\quad \tau \quad \quad h1 \quad \rightarrow \quad (C1, C2)$$
Type-Based Abstractions for Pre-Pruning

$k$-limited: $\alpha_k = \text{take length } k \text{ prefix}$

\[
\begin{array}{c}
v:h5:h8 = \text{new } C \\
\alpha_1 \\
v:h5 = \text{new } C
\end{array}
\]

Type-based: $\tau = \text{replace alloc. sites with types}$ [Smaragdakis et al. 2011]

\[
\begin{array}{c}
v:h5:h8 = \text{new } C \\
\tau \\
v:t1:t0 = \text{new } C
\end{array}
\]

We use $\tau = \text{type of containing class} \times \text{type of allocation site}$

\[
\begin{array}{c}
\text{class C1} \\
\{ \\
h1: x = \text{new } C2 \\
\}
\end{array}
\]

Composed: $\beta_1 = \alpha_1 \circ \tau$

\[
\begin{array}{c}
v:h5:h8 = \text{new } C \\
\beta_1 \\
v:t1 = \text{new } C
\end{array}
\]
Rest of Talk

Pre-Pruning Extension

Experiments
Experimental Setup

Clients (based on flow-insensitive $k$-object-sensitivity):
Experimental Setup

Clients (based on flow-insensitive $k$-object-sensitivity):

Downcast safety checking ($\text{DOWNCAST}$): $x = (C)y$
Experimental Setup

**Clients** (based on flow-insensitive $k$-object-sensitivity):

- Downcast safety checking (**DOWNCAST**): $x = (C)y$
- Monomorphic call site detection (**MONOSITE**): $x.g()$
Experimental Setup

Clients (based on flow-insensitive $k$-object-sensitivity):

- Downcast safety checking (\texttt{DOWNCAST}): $x = (C)y$
- Monomorphic call site detection (\texttt{MONOSITE}): $x.g()$
- Race detection (\texttt{RACE}): $x.f = \ldots \ | \ y.f = \ldots$
Experimental Setup

Clients (based on flow-insensitive $k$-object-sensitivity):

Downcast safety checking (DOWNCAST): $x = (C)y$
Monomorphic call site detection (MONOSITE): $x.g()$
Race detection (RACE): $x.f = \ldots \ | y.f = \ldots$

Benchmarks:

<table>
<thead>
<tr>
<th>description</th>
<th># bytecodes</th>
<th># alloc. sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>elevator</td>
<td>discrete event simulation program</td>
<td>39K</td>
</tr>
<tr>
<td>hedc</td>
<td>web crawler</td>
<td>151K</td>
</tr>
<tr>
<td>weblech</td>
<td>website downloading and mirroring tool</td>
<td>230K</td>
</tr>
<tr>
<td>lusearch</td>
<td>text indexing and search tool</td>
<td>267K</td>
</tr>
<tr>
<td>avrora</td>
<td>AVR microcontroller simulation/analysis framework</td>
<td>312K</td>
</tr>
</tbody>
</table>
Experimental Setup

Clients (based on flow-insensitive $k$-object-sensitivity):

Downcast safety checking (**DOWNCAST**): $x = (C)y$

Monomorphic call site detection (**MONOSITE**): $x.g()$

Race detection (**RACE**): $x.f = \ldots \mid y.f = \ldots$

Benchmarks:

<table>
<thead>
<tr>
<th>description</th>
<th># bytecodes</th>
<th># alloc. sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>elevator: discrete event simulation program</td>
<td>39K</td>
<td>637</td>
</tr>
<tr>
<td>hedc: web crawler</td>
<td>151K</td>
<td>1,494</td>
</tr>
<tr>
<td>weblech: website downloading and mirroring tool</td>
<td>230K</td>
<td>2,545</td>
</tr>
<tr>
<td>lusearch: text indexing and search tool</td>
<td>267K</td>
<td>2,822</td>
</tr>
<tr>
<td>avrora: AVR microcontroller simulation/analysis framework</td>
<td>312K</td>
<td>4,822</td>
</tr>
</tbody>
</table>

Details:

64-bit IBM J9VM 1.6, Chord with bddbddb Datalog solver

Terminate a process if it exceeds 8GB of memory
Curbing the Exponential Growth

Methods:

- no pruning
Curbing the Exponential Growth

Methods:

- no pruning
- selected refinement [Liang et al. 2011]
Curbing the Exponential Growth

Methods:

- no pruning
- selected refinement [Liang et al. 2011]
- Prune-Refine without pre-pruning
Curbing the Exponential Growth

Methods:

- no pruning
- ✓ selected refinement [Liang et al. 2011]
- ✰ Prune-Refine without pre-pruning
- □ Prune-Refine with pre-pruning
Curbing the Exponential Growth

Methods:

- no pruning
- \( \times \) selected refinement [Liang et al. 2011]
- \( \ast \) Prune-Refine without pre-pruning
- \( \Box \) Prune-Refine with pre-pruning

\[ |A'_t| = \text{number of tuples passed into } P \ (\text{Datalog solver}) \]
Curbing the Exponential Growth

Methods:
- no pruning
- \( \times \) selected refinement [Liang et al. 2011]
- \( \ast \) Prune-Refine without pre-pruning
- \( \square \) Prune-Refine with pre-pruning

\[ |A'_t| = \text{number of tuples passed into } P \text{ (Datalog solver)} \]

(a) DOWNCAST/weblech  
(b) DOWNCAST/lusearch  
(c) DOWNCAST/avrora
Curbing the Exponential Growth

Methods:

- no pruning
- selected refinement [Liang et al. 2011]
- Prune-Refine without pre-pruning
- Prune-Refine with pre-pruning

\[ |A'_t| = \text{number of tuples passed into } P \text{ (Datalog solver)} \]

(a) MONOSITE/weblech
(b) MONOSITE/lusearch
(c) MONOSITE/avrora
Curbing the Exponential Growth

Methods:

- no pruning
- selected refinement [Liang et al. 2011]
- Prune-Refine without pre-pruning
- Prune-Refine with pre-pruning

\[ |A'_t| = \text{number of tuples passed into } \mathcal{P} \text{ (Datalog solver)} \]

(a) RACE/weblech (b) RACE/lusearch (c) RACE/avrora
How Much is Pruned?

In each iteration, what fraction of tuples are kept?
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

\[ A_t \]

100%
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

\[ A_t \xrightarrow{\text{Pre-Prune}} A'_t \]

100% \quad 22%
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

\[
A_t \xrightarrow{\text{Pre-Prune}} A'_t \xrightarrow{\text{Prune } P} \tilde{A}_t
\]

<table>
<thead>
<tr>
<th>Iteration</th>
<th>100%</th>
<th>22%</th>
<th>18%</th>
</tr>
</thead>
</table>
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

\[
\begin{align*}
A_t & \xrightarrow{\text{Pre-Prune}} A'_t \xrightarrow{\text{Prune } P} \tilde{A}_t \xrightarrow{\text{Refine } \alpha_{t+1}} A_{t+1} \\
100\% & \quad 22\% \quad 18\% \quad 272\% 
\end{align*}
\]
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

$A_t$ → Pre-Prune $A_t'$ → Prune $P$ $\tilde{A}_t$ → Refine $\alpha_{t+1}$ $A_{t+1}$

100% 22% 18% 272%

(numbers averaged across all clients/benchmarks/iterations)
How Much is Pruned?

In each iteration, what fraction of tuples are kept?

\[ A_t \xrightarrow{\text{Pre-Prune}} A'_t \xrightarrow{\text{Prune } P} \tilde{A}_t \xrightarrow{\text{Refine } \alpha_{t+1}} A_{t+1} \]

100% 22% 18% 272%

(numbers averaged across all clients/benchmarks/iterations)

Take Away: Pruning (especially pre-pruning) helps a lot to maintain tractability
Impact on Queries Proven

How many queries remain unproven?
### Impact on Queries Proven

How many queries remain unproven?

<table>
<thead>
<tr>
<th>client/benchmark \ k</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNCAST/elevator</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DOWNCAST/hedc</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DOWNCAST/weblech</td>
<td>24</td>
<td>14</td>
<td>6</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>DOWNCAST/lusearch</td>
<td>36</td>
<td>14</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DOWNCAST/avrora</td>
<td>12</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>MONOSITE/elevator</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MONOSITE/hedc</td>
<td>164</td>
<td>149</td>
<td>149</td>
<td>149</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/weblech</td>
<td>273</td>
<td>258</td>
<td>252</td>
<td>252</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/lusearch</td>
<td>593</td>
<td>454</td>
<td>447</td>
<td>447</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/avrora</td>
<td>288</td>
<td>278</td>
<td>272</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/elevator</td>
<td>475</td>
<td>440</td>
<td>437</td>
<td>437</td>
<td>437</td>
</tr>
<tr>
<td>RACE/hedc</td>
<td>23,033</td>
<td>22,043</td>
<td>21,966</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/weblech</td>
<td>7,286</td>
<td>4,742</td>
<td>4,669</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/lusearch</td>
<td>33,845</td>
<td>23,509</td>
<td>16,957</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/avrora</td>
<td>62,060</td>
<td>61,807</td>
<td>61,734</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
# Impact on Queries Proven

How many queries remain unproven?

<table>
<thead>
<tr>
<th>client/benchmark \ k</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNCAST/elevator</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DOWNCAST/hedc</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DOWNCAST/weblech</td>
<td>24</td>
<td>14</td>
<td>6</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>DOWNCAST/lusearch</td>
<td>36</td>
<td>14</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DOWNCAST/avrora</td>
<td>12</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>MONOSITE/elevator</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MONOSITE/hedc</td>
<td>164</td>
<td>149</td>
<td>149</td>
<td>149</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/weblech</td>
<td>273</td>
<td>258</td>
<td>252</td>
<td>252</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/lusearch</td>
<td>593</td>
<td>454</td>
<td>447</td>
<td>447</td>
<td>-</td>
</tr>
<tr>
<td>MONOSITE/avrora</td>
<td>288</td>
<td>278</td>
<td>272</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/elevator</td>
<td>475</td>
<td>440</td>
<td>437</td>
<td>437</td>
<td>437</td>
</tr>
<tr>
<td>RACE/hedc</td>
<td>23,033</td>
<td>22,043</td>
<td>21,966</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/weblech</td>
<td>7,286</td>
<td>4,742</td>
<td>4,669</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/lusearch</td>
<td>33,845</td>
<td>23,509</td>
<td>16,957</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RACE/avrora</td>
<td>62,060</td>
<td>61,807</td>
<td>61,734</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Take Away:** By using Prune-Refine, able to prove two additional queries
Conclusion

• **Goal**: scale up static analyses
Conclusion

- **Goal**: scale up static analyses

- **Contribution**: new general pruning framework
Conclusion

- **Goal**: scale up static analyses

- **Contribution**: new general pruning framework

- **Key Idea**: use coarse abstraction to remove irrelevant tuples
Conclusion

• **Goal**: scale up static analyses

• **Contribution**: new general pruning framework

• **Key Idea**: use coarse abstraction to remove irrelevant tuples

• **Theoretical Result**: pruning is correct
Conclusion

• **Goal**: scale up static analyses

• **Contribution**: new general pruning framework

• **Key Idea**: use coarse abstraction to remove irrelevant tuples

• **Theoretical Result**: pruning is correct

• **Empirical Result**: enable much finer abstractions
Conclusion

• **Goal**: scale up static analyses

• **Contribution**: new general pruning framework

• **Key Idea**: use coarse abstraction to remove irrelevant tuples

• **Theoretical Result**: pruning is correct

• **Empirical Result**: enable much finer abstractions

http://code.google.com/p/jchord

Thank you!