Scaling Abstraction Refinement via Pruning

PLDI - San Jose, CA June 8, 2011

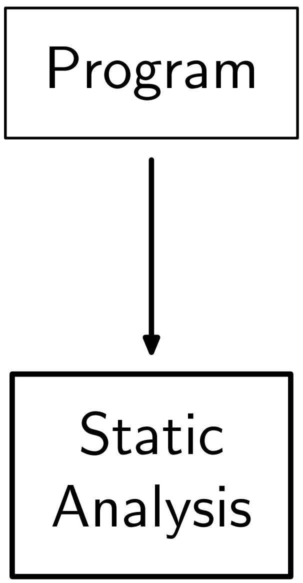
Percy Liang
UC Berkeley

Mayur Naik
Intel Labs Berkeley

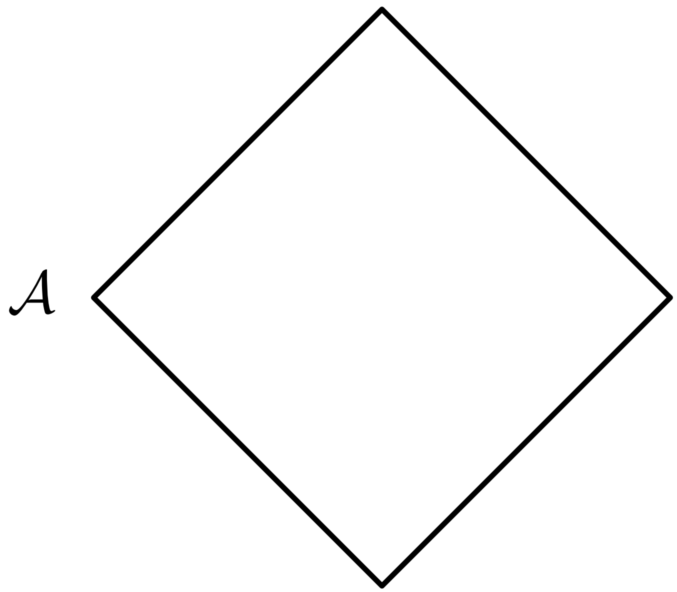
The Big Picture

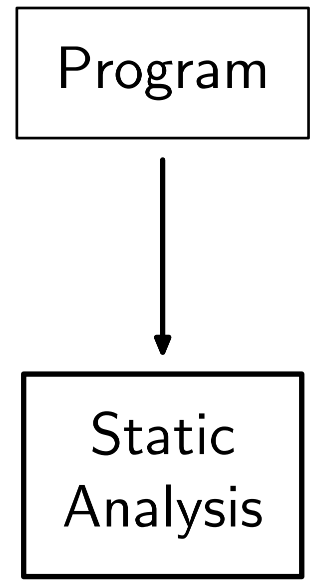
Program

The Big Picture

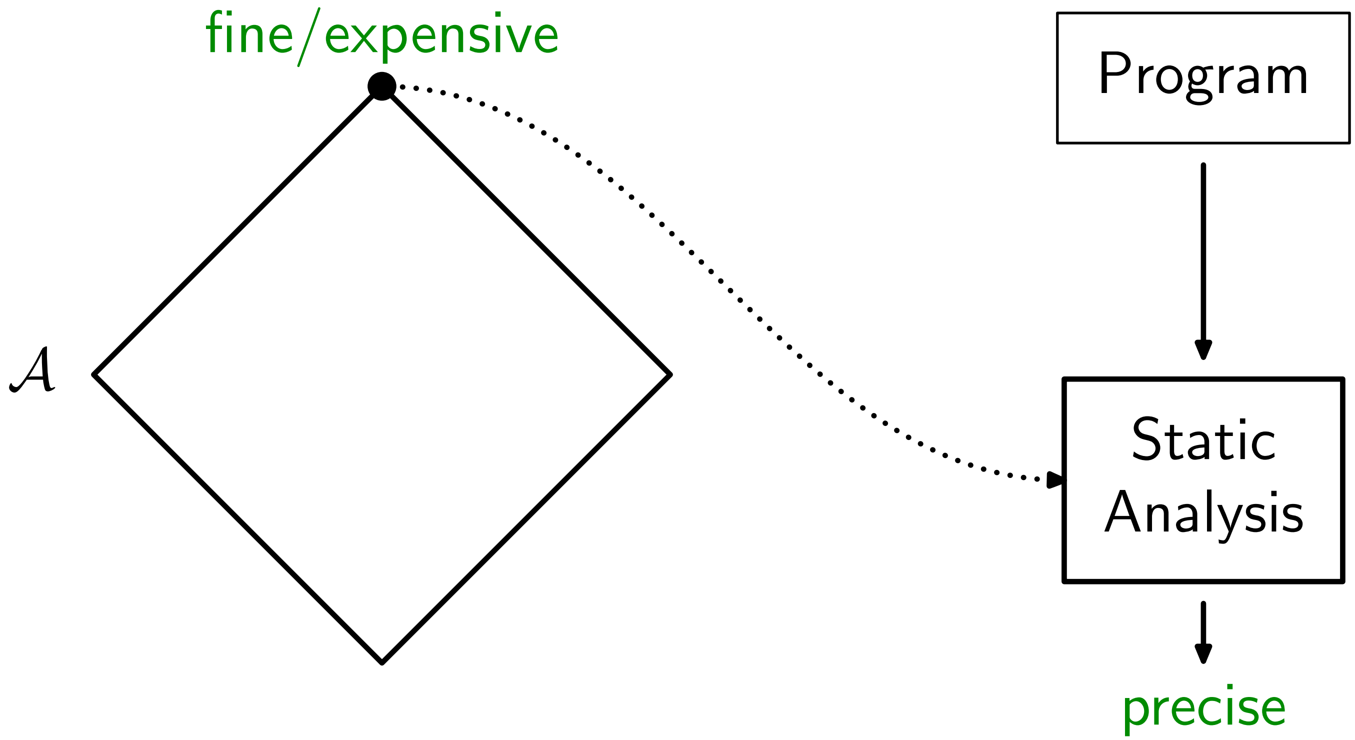


The Big Picture

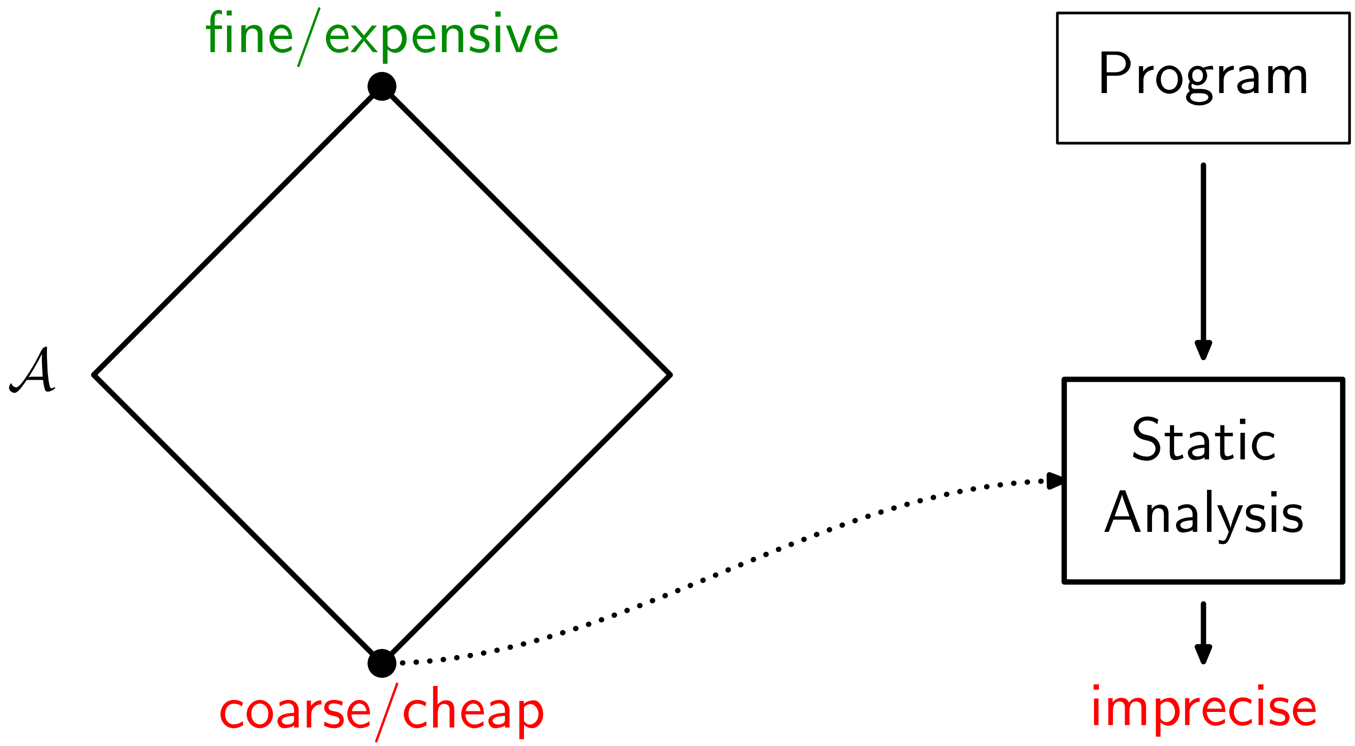




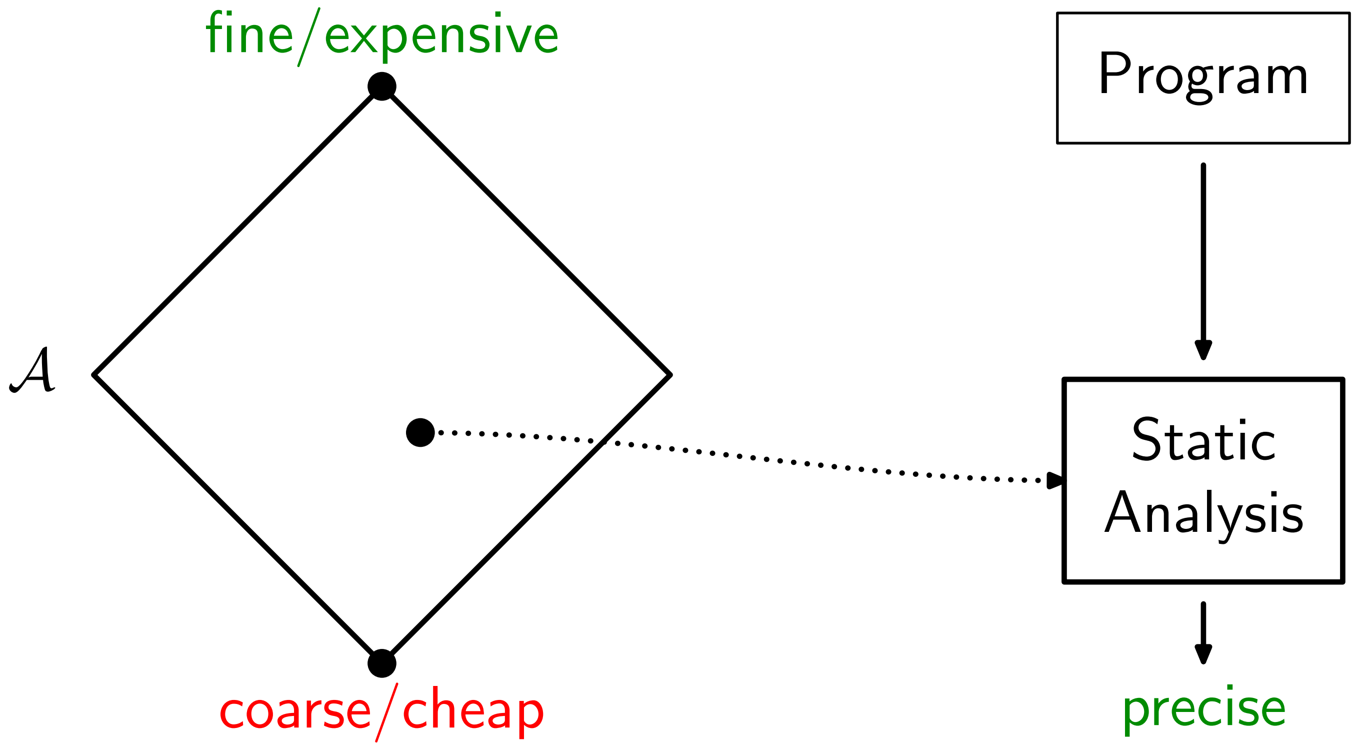
The Big Picture



The Big Picture



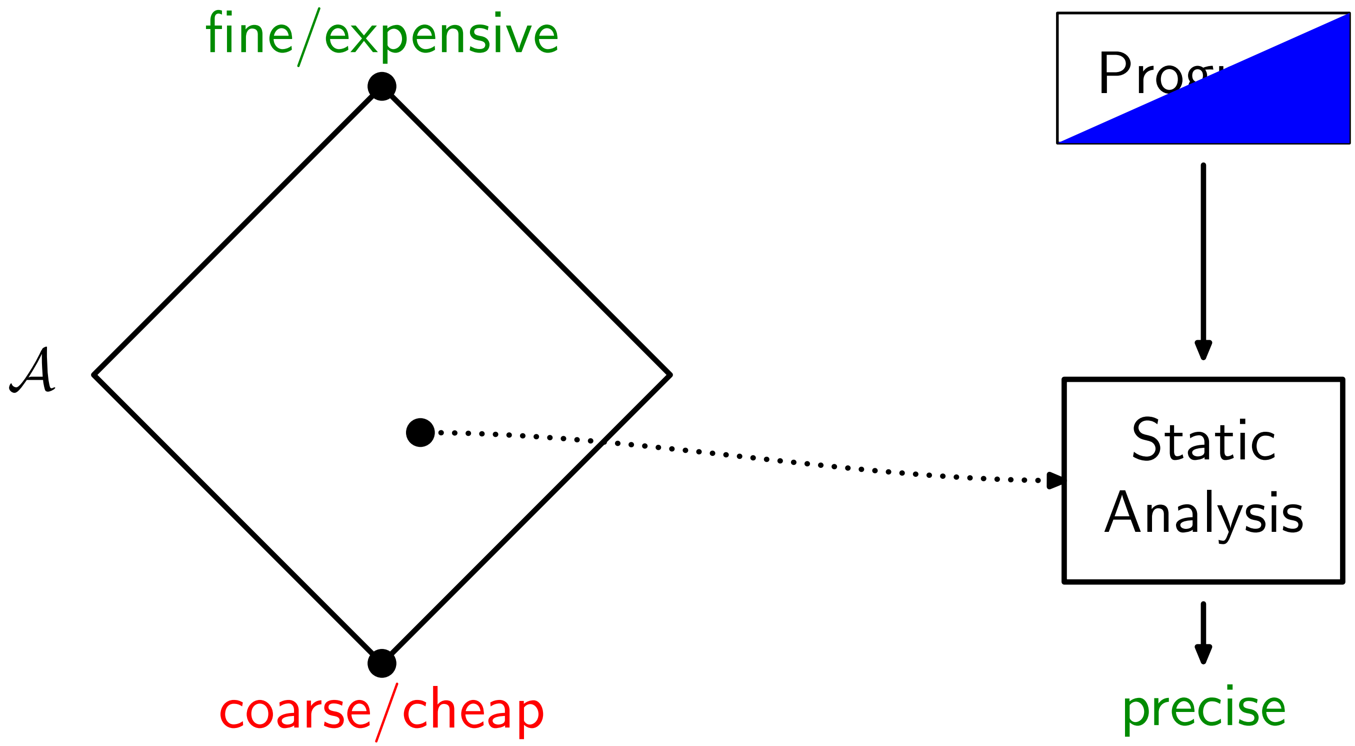
The Big Picture



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]

pruning

NEW!

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:

$\boxed{\texttt{u = new C}}$ $\boxed{\texttt{v = new C}}$

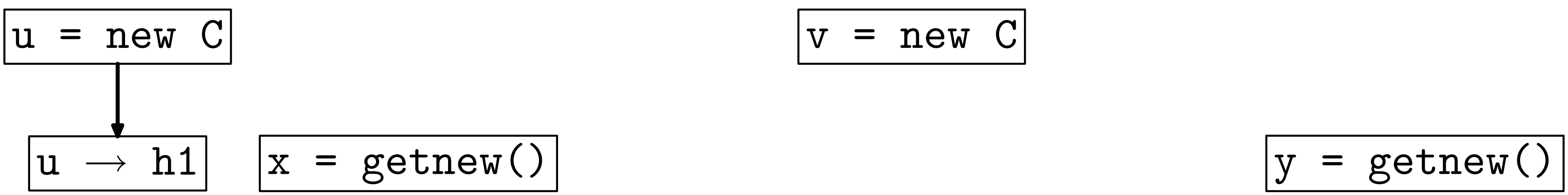
$\boxed{\texttt{x = getnew()}}$ $\boxed{\texttt{y = getnew()}}$

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:



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()

v = new C

v → h2

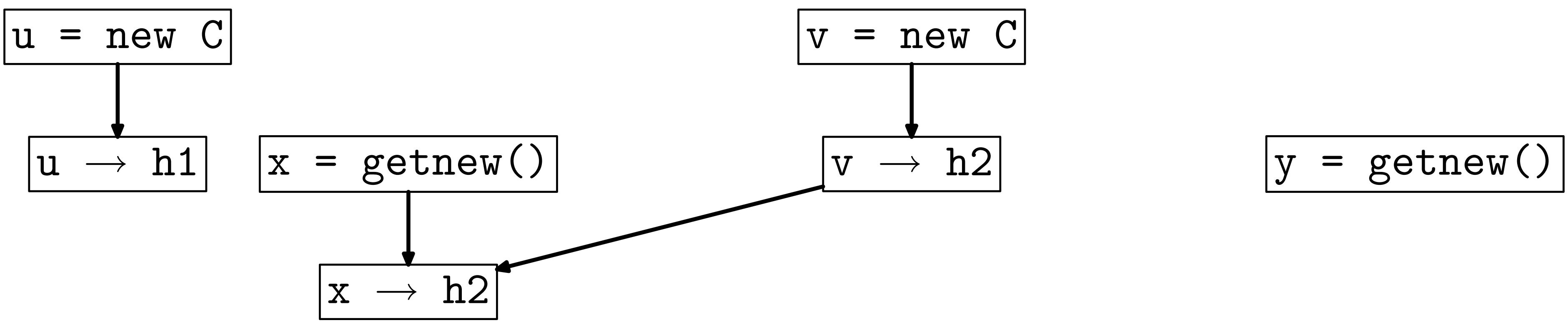
y = getnew()

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:

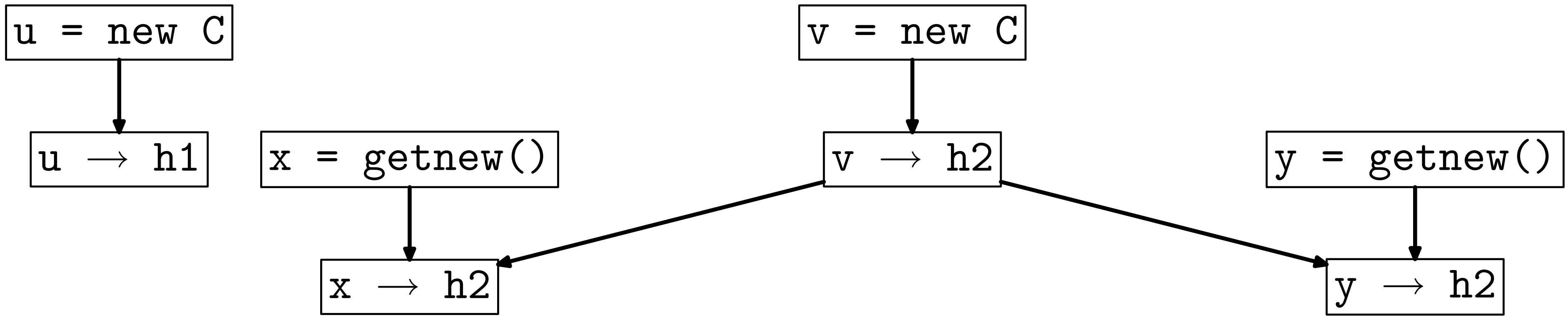


An Example of Pruning

```
    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:

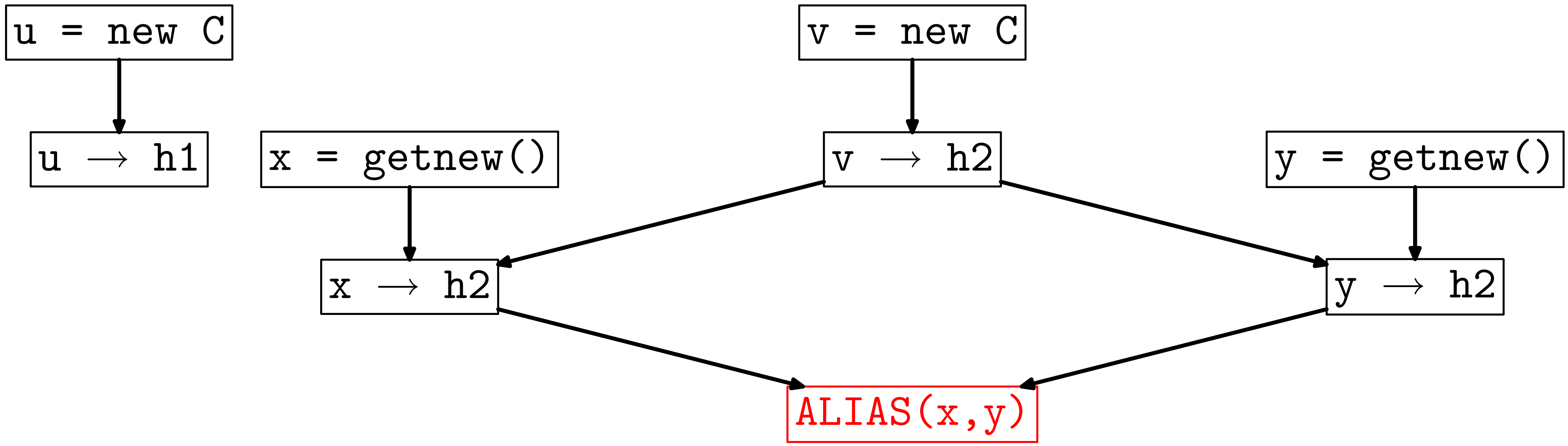


An Example of Pruning

```
    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:

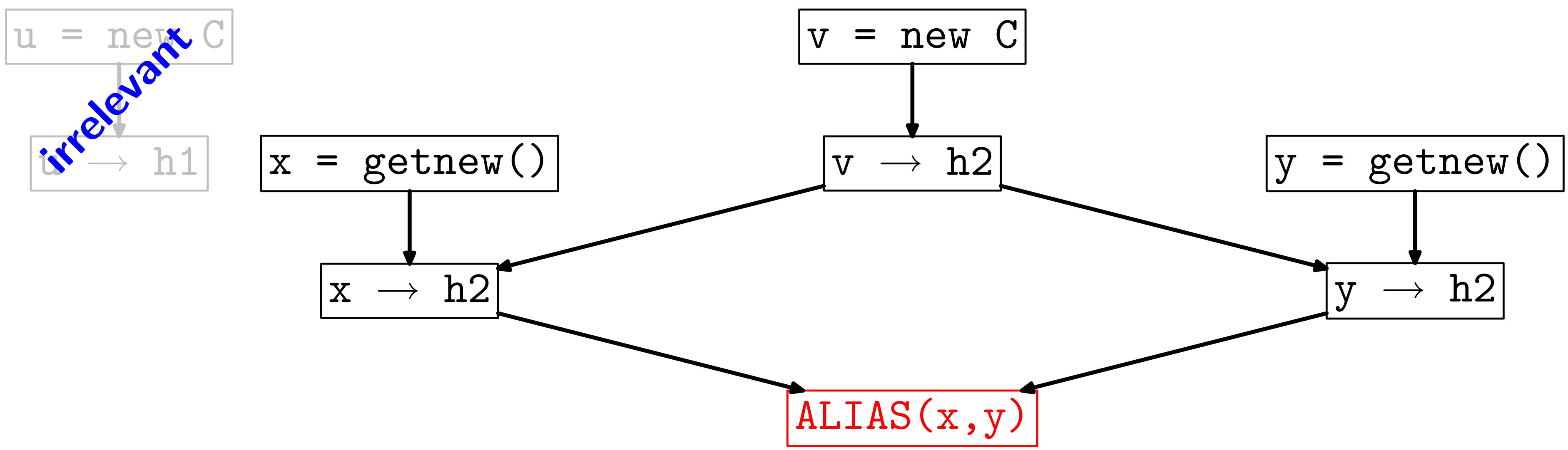


An Example of Pruning

```
    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:



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` $\longrightarrow$ `h1`

`x = getnew()`

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

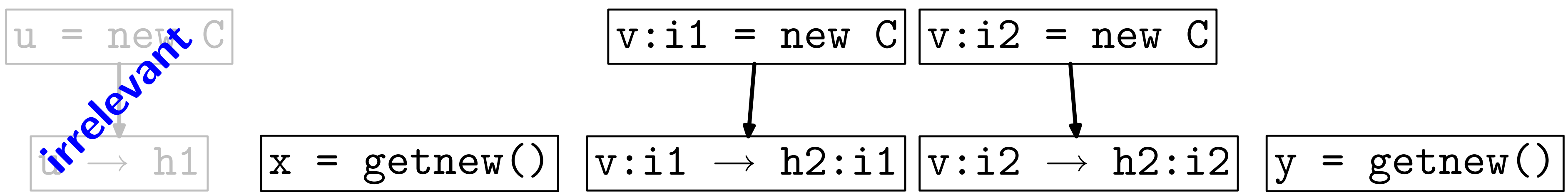
`y = getnew()`

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:

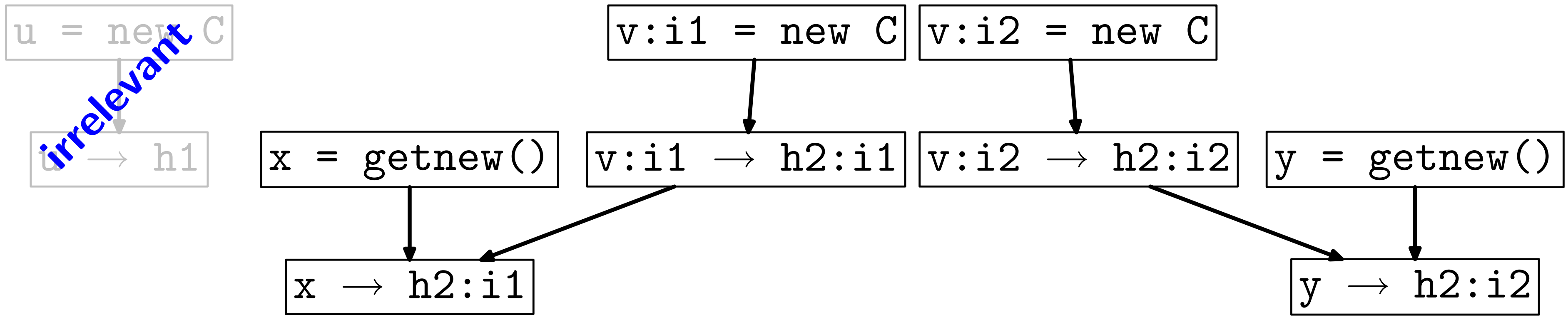


An Example of Pruning

```
    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:

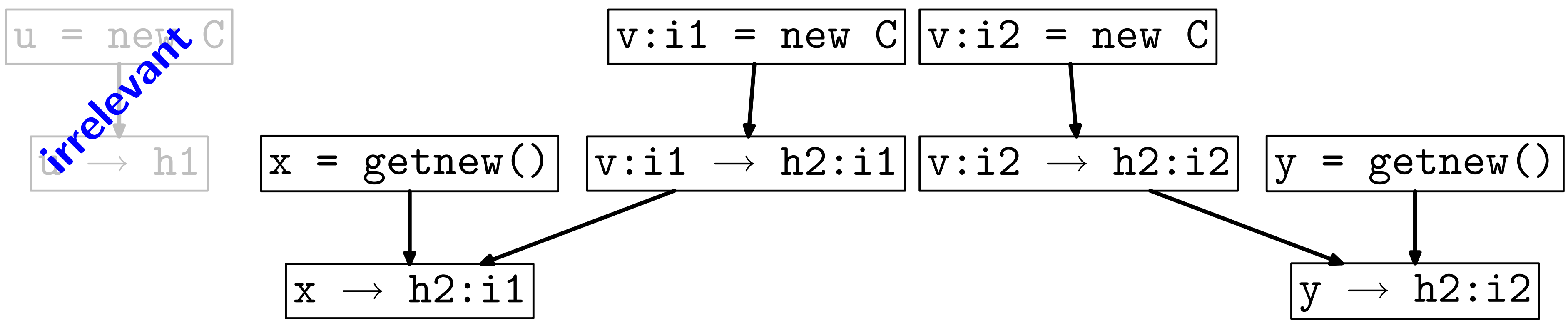


An Example of Pruning

```
    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:



(not aliasing - query proven)

General Pruning Framework

Input tuples A_0 $\boxed{\texttt{v = new C}} \cdots$

General Pruning Framework

Input tuples A_0 $\quad$ $\boxed{\texttt{v = new C}} \cdots$

Query tuple x_{Q} $\quad$ $\boxed{\texttt{ALIAS(x,y)}}$

General Pruning Framework

Input tuples A_0 $\boxed{\texttt{v = new C}} \cdots$

Query tuple x_{Q} $\boxed{\texttt{ALIAS(x,y)}}$

Datalog rules $\boxed{v_2 \ \longrightarrow \ h} \Leftarrow \boxed{v_2 \ \texttt{=} \ v_1} \texttt{,} \boxed{v_1 \ \longrightarrow \ h}$

$\cdots$

General Pruning Framework

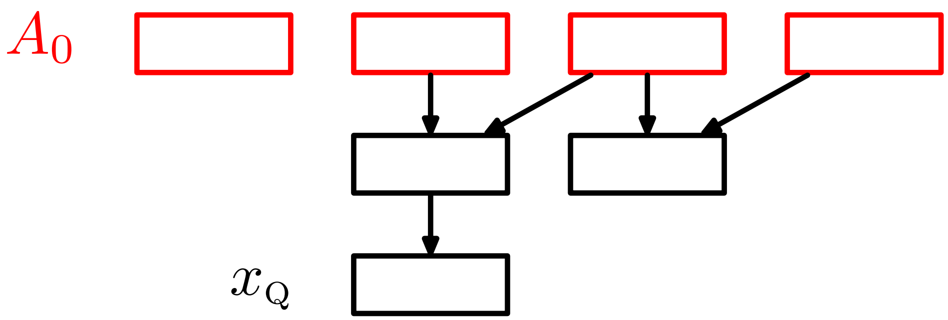
Input tuples A_0 $\boxed{\texttt{v = new C}}\cdots$

Query tuple x_Q $\boxed{\texttt{ALIAS(x,y)}}$

Datalog rules $\boxed{v_2 \;\longrightarrow\; h} \Leftarrow \boxed{v_2 \;\texttt{=}\; v_1}\,,\,\boxed{v_1 \;\longrightarrow\; h}$

$\cdots$

Prune/prove operator $\mathbf{P}$ $A_0 \xrightarrow{\mathbf{P}}$ subset of A_0 used to derive x_Q



General Pruning Framework

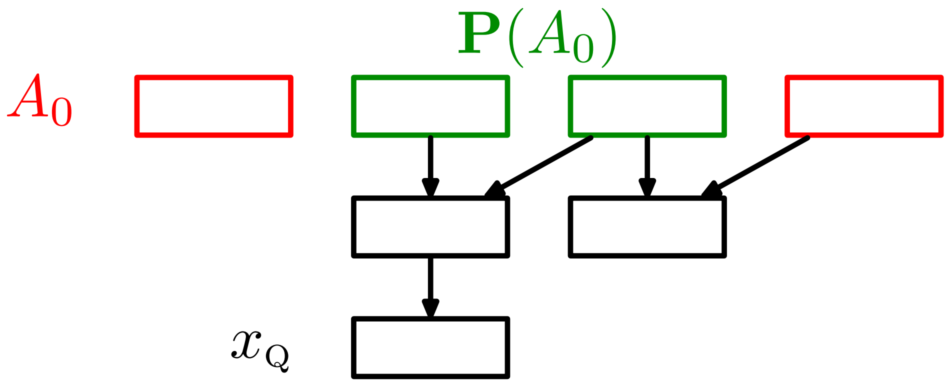
Input tuples A_0 $\boxed{\texttt{v = new C}}\ \cdots$

Query tuple x_Q $\boxed{\texttt{ALIAS(x,y)}}$

Datalog rules $\boxed{v_2 \ \rightarrow\ h} \Leftarrow \boxed{v_2 \ \texttt{=}\ v_1}\ ,\ \boxed{v_1 \ \rightarrow\ h}$

$\cdots$

Prune/prove operator $\mathbf{P}$ $A_0 \xrightarrow{\mathbf{P}}$ subset of A_0 used to derive x_Q



General Pruning Framework

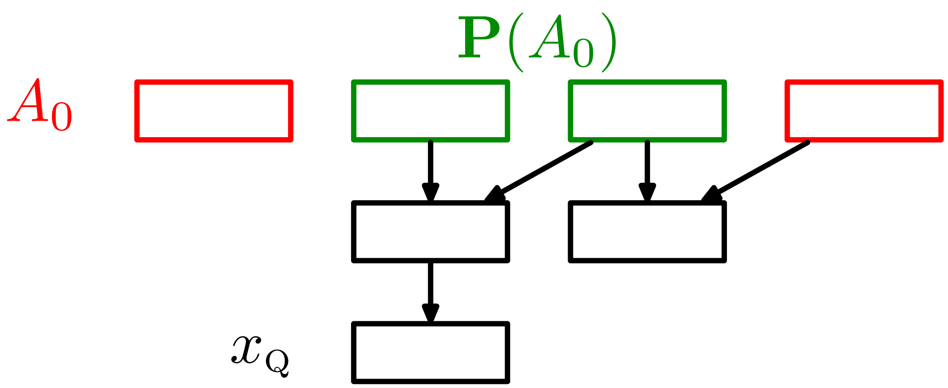
Input tuples A_0 — $\boxed{\texttt{v = new C}}\ \cdots$

Query tuple x_{Q} — $\boxed{\texttt{ALIAS(x,y)}}$

Datalog rules — $\boxed{v_2 \rightarrow h} \Leftarrow \boxed{v_2 \texttt{ = } v_1}\,,\ \boxed{v_1 \rightarrow h}$

$\cdots$

Prune/prove operator $\mathbf{P}$ — $A_0 \xrightarrow{\mathbf{P}}$ subset of A_0 used to derive x_{Q}



Query proven $\Leftrightarrow$ $\mathbf{P}(A_0) = \emptyset$

Prune and Refine

(abstract tuples) A_0 $\boxed{\texttt{u = new C}}\ \boxed{\texttt{v = new C}}\ \cdots$

Prune and Refine

(abstract tuples)	A_0	$\boxed{\texttt{u = new C}}\ \boxed{\texttt{v = new C}} \cdots$
Prune	$\downarrow \mathbf{P}$	run 0-CFA

Prune and Refine

(abstract tuples)	A_0	$\boxed{\texttt{u = new C}}$ $\boxed{\texttt{v = new C}}$ $\cdots$
Prune	$\Big\downarrow \mathbf{P}$	run 0-CFA
(relevant tuples)	$\tilde{A}_0$	$\boxed{\cancel{\texttt{u = new C}}}$ $\boxed{\texttt{v = new C}}$

Prune and Refine

(abstract tuples) A_0 `u = new C` `v = new C` $\cdots$

Prune $\big\downarrow \mathbf{P}$ run 0-CFA

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

Refine $\big\downarrow \alpha_1$

Prune and Refine

(abstract tuples)	A_0	$\boxed{\texttt{u = new C}}$ $\boxed{\texttt{v = new C}}$ $\cdots$
Prune	$\big\downarrow \mathbf{P}$	run 0-CFA
(relevant tuples)	$\tilde{A}_0$	$\boxed{\cancel{\texttt{u = new C}}}$ $\boxed{\texttt{v = new C}}$
Refine	$\big\downarrow \alpha_1$	
(refined tuples)	A_1	$\boxed{\texttt{v:i1 = new C}}$ $\boxed{\texttt{v:i2 = new C}}$

Prune and Refine

(abstract tuples) A_0 `u = new C` `v = new C` $\cdots$

Prune $\downarrow \mathbf{P}$ run 0-CFA

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

Refine $\downarrow \alpha_1$

(refined tuples) A_1 `v:i1 = new C` `v:i2 = new C`

Prune $\downarrow \mathbf{P}$ run 1-CFA

Prune and Refine

(abstract tuples) A_0 `u = new C` `v = new C` $\cdots$

Prune $\Big\downarrow \mathbf{P}$ run 0-CFA

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

Refine $\Big\downarrow \alpha_1$

(refined tuples) A_1 `v:i1 = new C` `v:i2 = new C`

Prune $\Big\downarrow \mathbf{P}$ run 1-CFA

$\emptyset$ $\Rightarrow$ 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: $\tilde{A}_t = \mathbf{P}(A_t)$. 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 $t = 0, 1, 2, \dots :$

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

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

Main Result:

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

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

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

Main Result:

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

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

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

Main Result:

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

fast slow

Rest of Talk

Pre-Pruning Extension

Experiments

Pre-Pruning

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

```
v = new C
```

Pre-Pruning

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

```
v = new C
```

$\xrightarrow{\text{Refine } \alpha_t}$

A_t

```
v:h0 = new C
v:h1 = new C
v:h2 = new C
v:h3 = new C
v:h4 = new C
v:h5 = new C
```

Pre-Pruning

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

```
v = new C
```

$\xrightarrow{\text{Refine } \alpha_t}$

A_t

```
v:h0 = new C
v:h1 = new C
v:h2 = new C
v:h3 = new C
v:h4 = new C
v:h5 = new C
```

$\xrightarrow{\text{Prune } \mathbf{P}}$

expensive!

Pre-Pruning

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

```
v = new C
```

Refine α_t $\longrightarrow$

A_t

```
v:h0 = new C
v:h1 = new C
v:h2 = new C
v:h3 = new C
v:h4 = new C
v:h5 = new C
```

Prune $\mathbf{P}$ $\longrightarrow$

$\downarrow \beta_t \preceq \alpha_t$

B_t

```
v:t0 = new C
v:t1 = new C
```

Pre-Pruning

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

```
v = new C
```

$\xrightarrow{\text{Refine } \alpha_t}$

A_t

```
v:h0 = new C
v:h1 = new C
v:h2 = new C
v:h3 = new C
v:h4 = new C
v:h5 = new C
```

$\xrightarrow{\text{Prune } \mathbf{P}}$

$\Big\downarrow \; \beta_t \preceq \alpha_t$

B_t

```
v:t0 = new C
v:t1 = new C
```

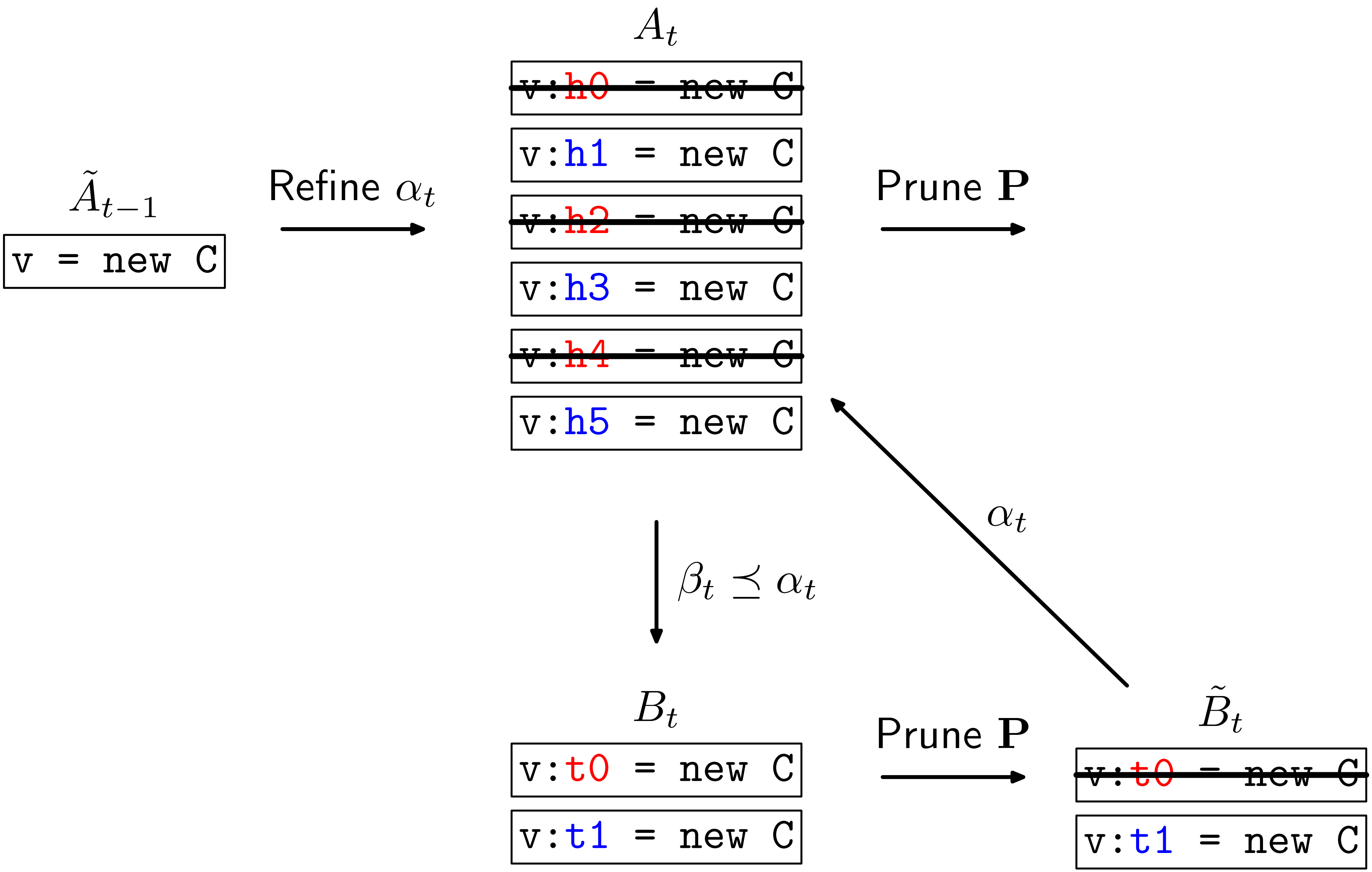
$\xrightarrow{\text{Prune } \mathbf{P}}$

$\tilde{B}_t$

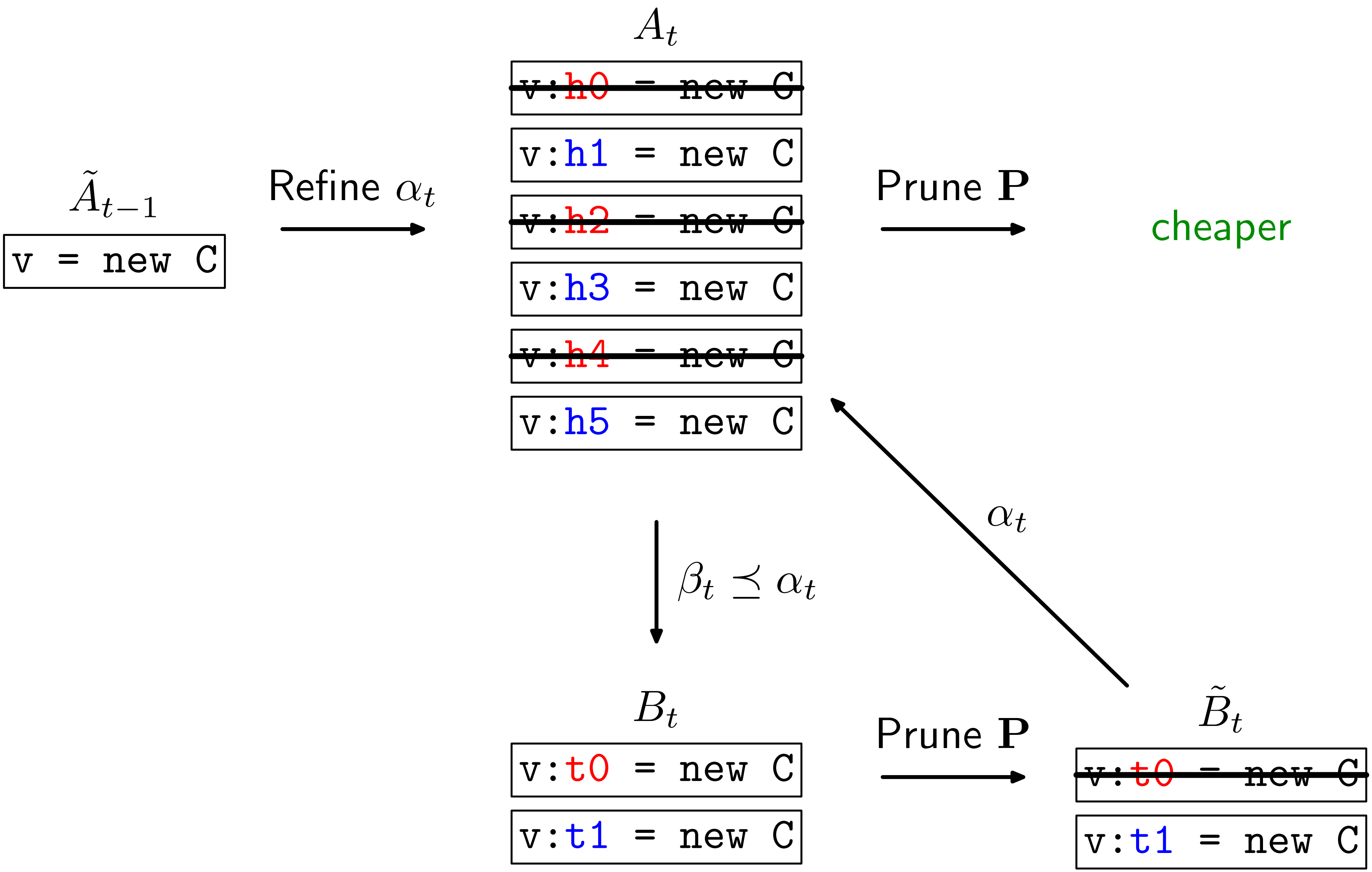
~~`v:t0 = new C`~~

```
v:t1 = new C
```

Pre-Pruning


$\tilde{A}_{t-1}$
v = new C
Refine α_t
A_t
~~v:h0 = new C~~
v:h1 = new C
~~v:h2 = new C~~
v:h3 = new C
~~v:h4 = new C~~
v:h5 = new C
Prune $\mathbf{P}$
$\beta_t \preceq \alpha_t$
α_t
B_t
v:t0 = new C
v:t1 = new C
Prune $\mathbf{P}$
$\tilde{B}_t$
~~v:t0 = new C~~
v:t1 = new C

Pre-Pruning



$\tilde{A}_{t-1}$
v = new C
Refine α_t
A_t
~~v:h0 = new C~~
v:h1 = new C
~~v:h2 = new C~~
v:h3 = new C
~~v:h4 = new C~~
v:h5 = new C
Prune $\mathbf{P}$
cheaper
α_t
$\beta_t \preceq \alpha_t$
B_t
v:t0 = new C
v:t1 = new C
Prune $\mathbf{P}$
$\tilde{B}_t$
~~v:t0 = new C~~
v:t1 = new C

Which Abstractions for Pre-Pruning?

$$
\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
 & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}
$$

Which Abstractions for Pre-Pruning?

$$\begin{array}{lccccccc} \text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\ & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\ \text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots \end{array}$$

Choose abstraction τ;

Which Abstractions for Pre-Pruning?

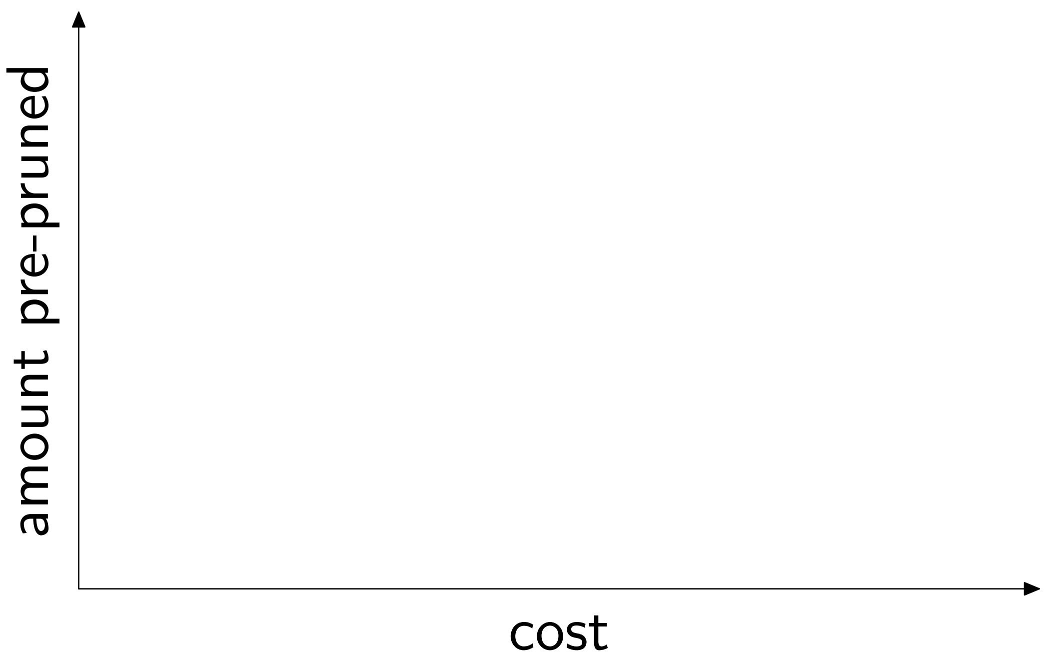
$$\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
& \curlyvee\!| & & \curlyvee\!| & & \curlyvee\!| & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}$$

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

Which Abstractions for Pre-Pruning?

$$
\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
& \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}
$$

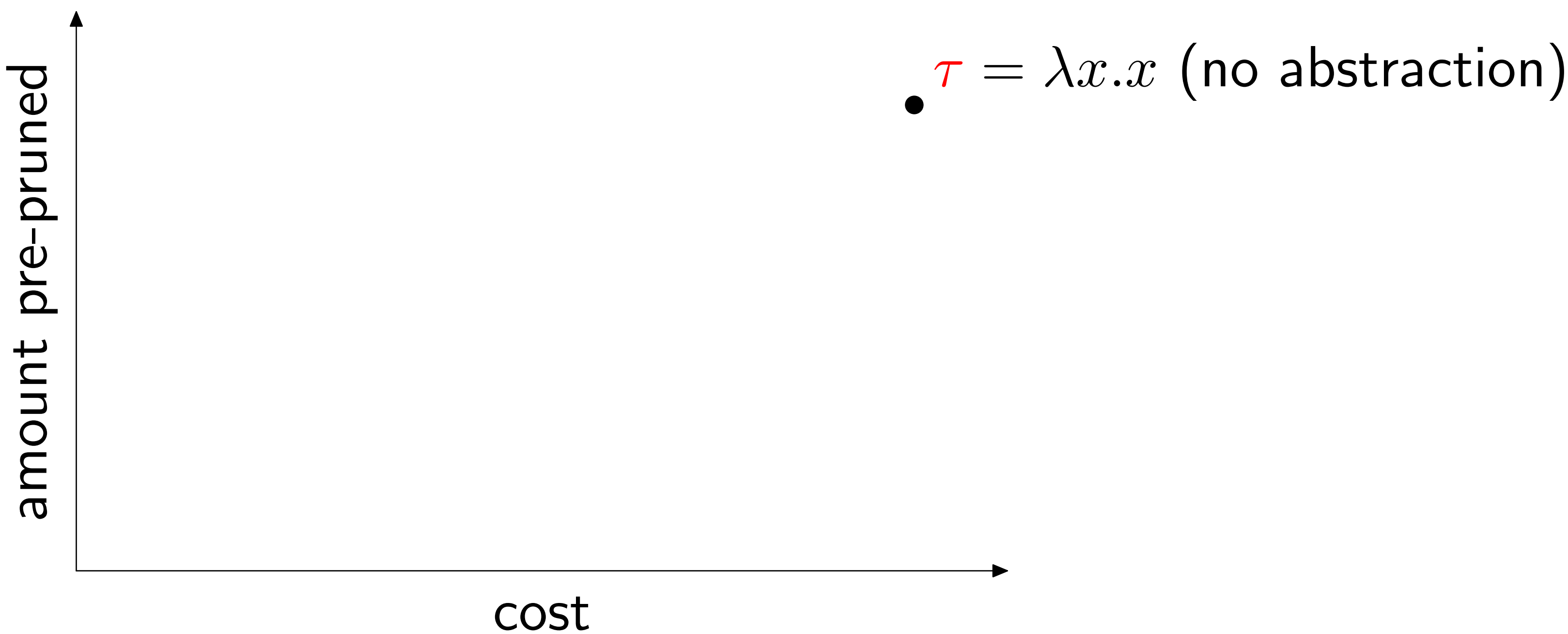
Choose abstraction τ; set $\beta_t = \alpha_t \circ \tau$



Which Abstractions for Pre-Pruning?

$$
\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
 & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}
$$

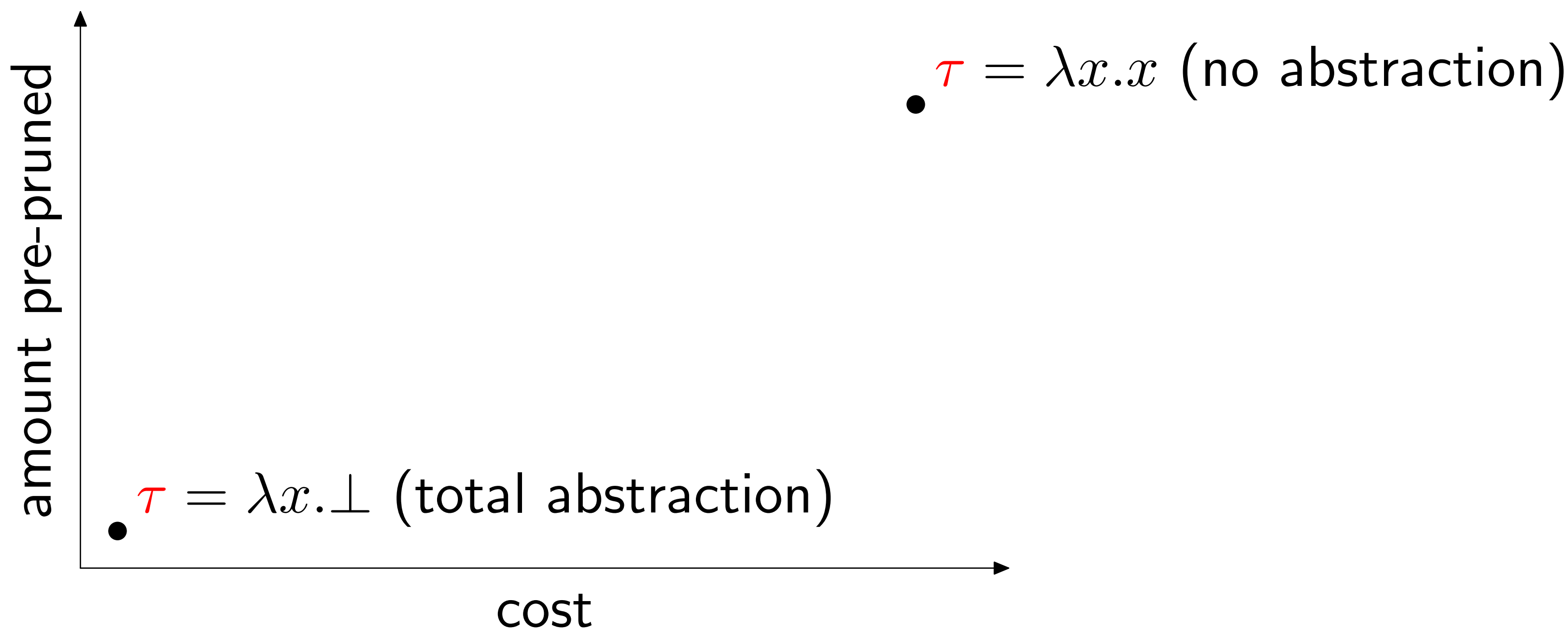
Choose abstraction τ; set $\beta_t = \alpha_t \circ \tau$



Which Abstractions for Pre-Pruning?

$$\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
& \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}$$

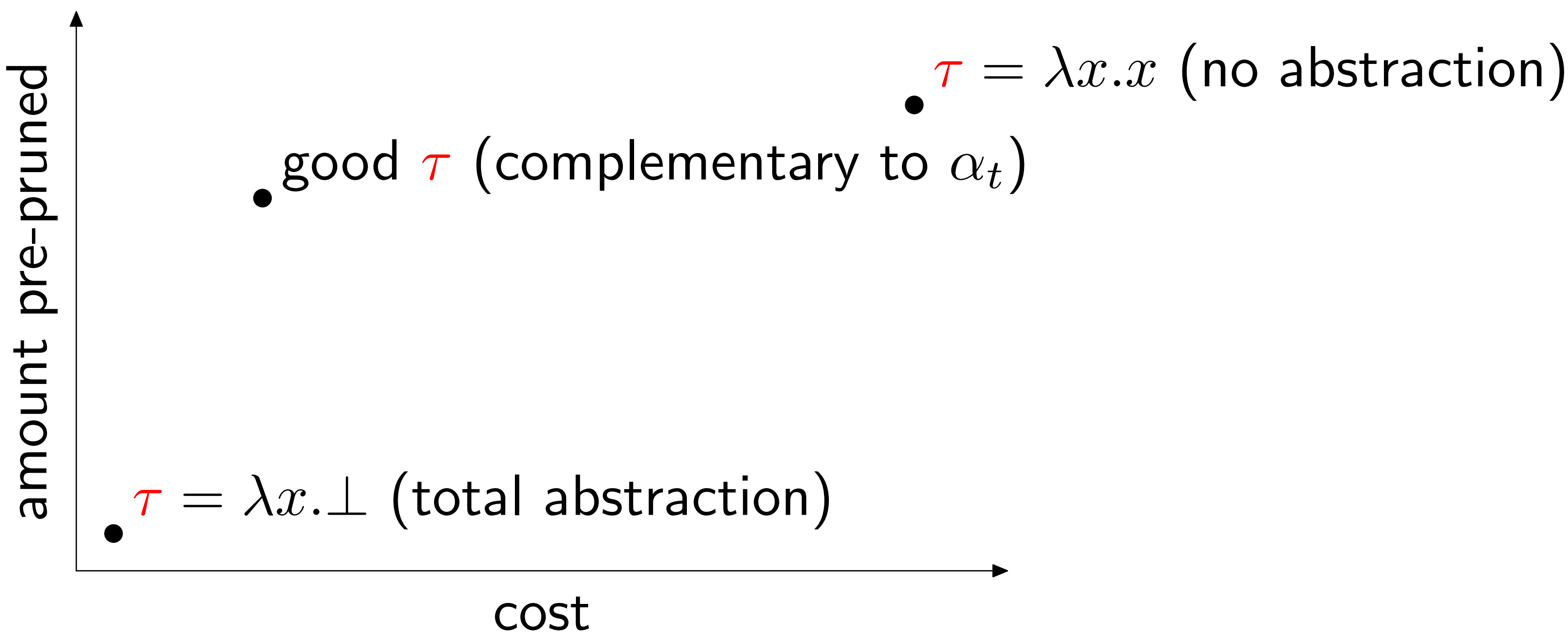
Choose abstraction τ; set $\beta_t = \alpha_t \circ \tau$



Which Abstractions for Pre-Pruning?

$$
\begin{array}{lccccccc}
\text{(main)} & \alpha_0 & \preceq & \alpha_1 & \preceq & \alpha_2 & \preceq & \cdots \\
 & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \rotatebox{90}{$\preceq$} & & \\
\text{(auxiliary)} & \beta_0 & \preceq & \beta_1 & \preceq & \beta_2 & \preceq & \cdots
\end{array}
$$

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



Type-Based Abstractions for Pre-Pruning

k-limited: $\alpha_k =$ take length k prefix

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

Type-Based Abstractions for Pre-Pruning

k-limited: α_k = take length k prefix

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

Type-based: τ = replace alloc. sites with types [Smaragdakis et al. 2011]

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

Type-Based Abstractions for Pre-Pruning

k-limited: $\alpha_k =$ take length k prefix

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

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

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

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

Type-Based Abstractions for Pre-Pruning

k-limited: $\alpha_k =$ take length k prefix

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

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

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

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

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

Type-Based Abstractions for Pre-Pruning

k-limited: $\alpha_k =$ take length k prefix

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

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

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

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

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

$$\texttt{h1} \xrightarrow{\tau} \texttt{(C1,C2)}$$

Type-Based Abstractions for Pre-Pruning

k-limited: α_k = take length k prefix

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

Type-based: τ = replace alloc. sites with types [Smaragdakis et al. 2011]

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

We use τ = type of containing class $\times$ type of allocation site

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

$$\texttt{h1} \xrightarrow{\tau} \texttt{(C1,C2)}$$

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

$$\boxed{\texttt{v:h5:h8 = new C}} \xrightarrow{\beta_1} \boxed{\texttt{v:t1 = new C}}$$

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 (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 (DOWNCAST): `x = (C)y`

Monomorphic call site detection (MONOSITE): `x.g()`

Race detection (RACE): `x.f = ...  | y.f = ...`

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 = ...  | y.f = ...`

Benchmarks:

	description	# bytecodes	# alloc. sites
`elevator`	discrete event simulation program	39K	637
`hedc`	web crawler	151K	1,494
`weblech`	website downloading and mirroring tool	230K	2,545
`lusearch`	text indexing and search tool	267K	2,822
`avrora`	AVR microcontroller simulation/analysis framework	312K	4,823

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 = ...  | y.f = ...`

Benchmarks:

	description	# bytecodes	# alloc. sites
`elevator`	discrete event simulation program	39K	637
`hedc`	web crawler	151K	1,494
`weblech`	website downloading and mirroring tool	230K	2,545
`lusearch`	text indexing and search tool	267K	2,822
`avrora`	AVR microcontroller simulation/analysis framework	312K	4,823

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
- × selected refinement [Liang et al. 2011]
- ✱ Prune-Refine without pre-pruning
- □ Prune-Refine with pre-pruning

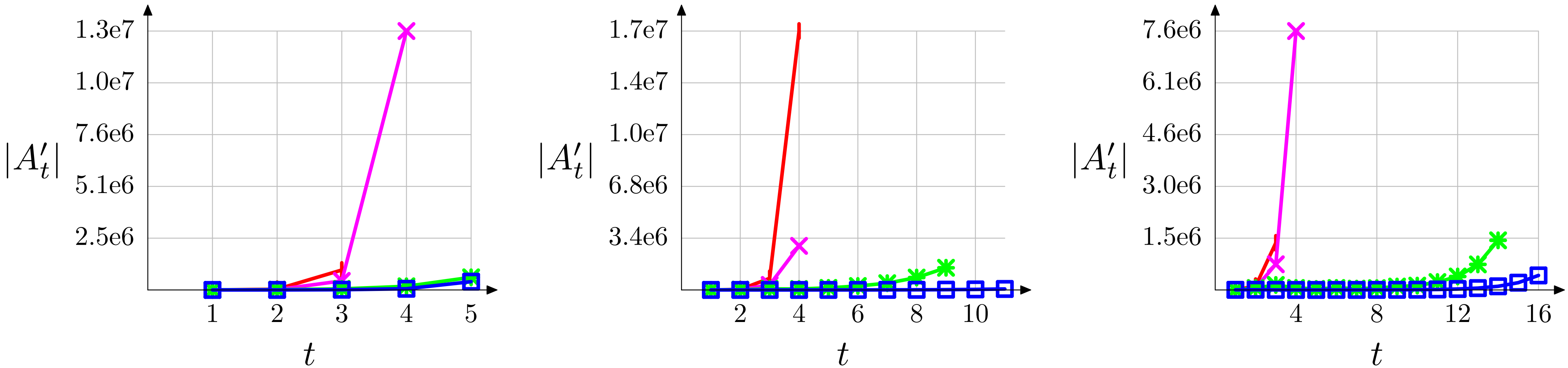
$|A'_t|$ = number of tuples passed into $\mathbf{P}$ (Datalog solver)

Curbing the Exponential Growth

Methods:

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

$|A'_t|$ = number of tuples passed into $\mathbf{P}$ (Datalog solver)



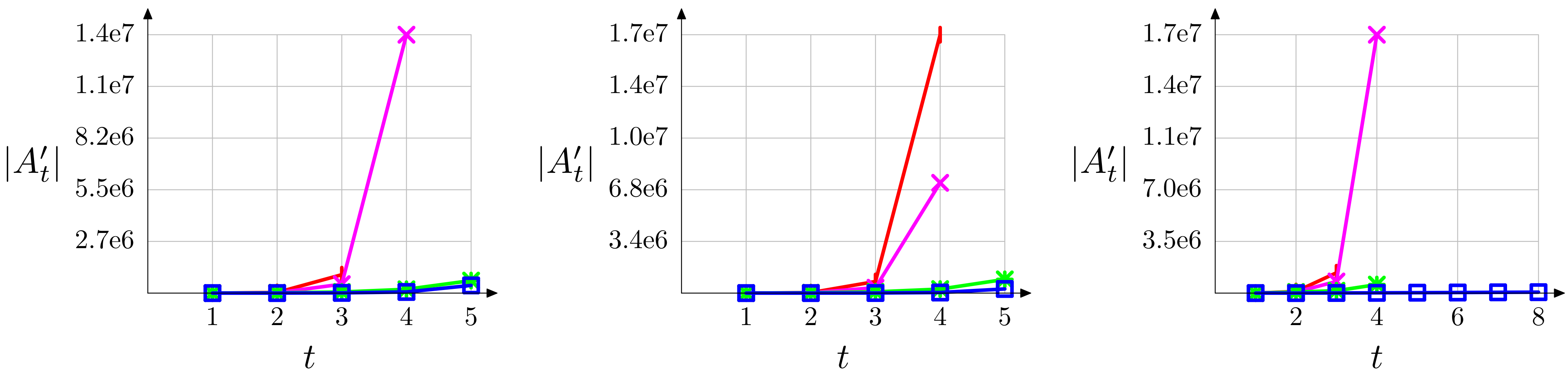
(a) DOWNCAST/`weblech` (b) DOWNCAST/`lusearch` (c) DOWNCAST/`avrora`

Curbing the Exponential Growth

Methods:

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

$|A'_t|$ = number of tuples passed into $\mathbf{P}$ (Datalog solver)



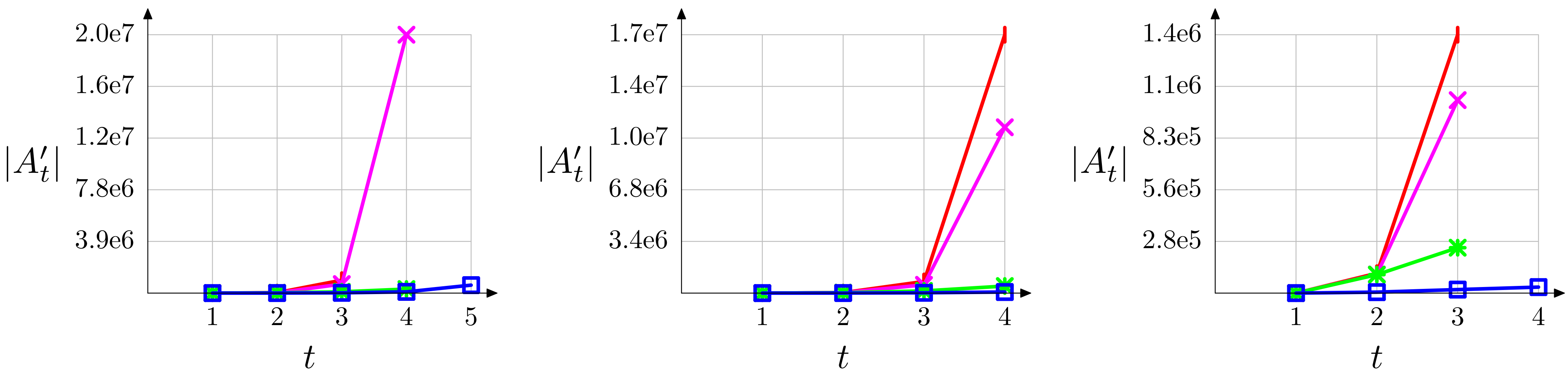
(a) MONOSITE/`weblech` (b) MONOSITE/`lusearch` (c) MONOSITE/`avrora`

Curbing the Exponential Growth

Methods:

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

$|A'_t|$ = number of tuples passed into $\mathbf{P}$ (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% 22%

How Much is Pruned?

In each iteration, what fraction of tuples are kept?

$$A_t \xrightarrow{\text{Pre-Prune}} A'_t \xrightarrow{\text{Prune } \mathbf{P}} \tilde{A}_t$$

100% 22% 18%

How Much is Pruned?

In each iteration, what fraction of tuples are kept?

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

100%	22%	18%	272%

How Much is Pruned?

In each iteration, what fraction of tuples are kept?

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

100%	22%	18%	272%

(numbers averaged across all clients/benchmarks/iterations)

How Much is Pruned?

In each iteration, what fraction of tuples are kept?

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

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?

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

Impact on Queries Proven

How many queries remain unproven?

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

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!