Learning Minimal Abstractions

POPL - Austin, TX   January 26, 2011

Percy Liang
UC Berkeley

Omer Tripp
Tel-Aviv Univ.

Mayur Naik
Intel Labs Berkeley

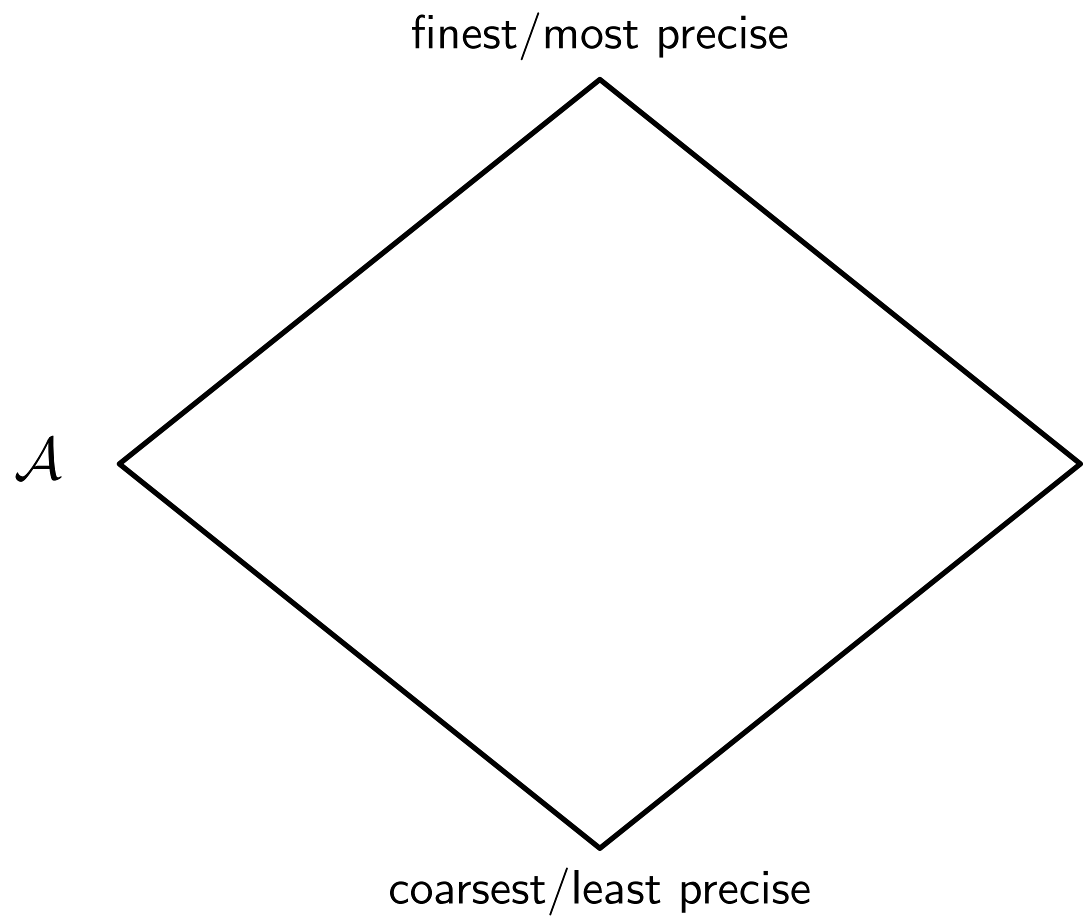
The Minimal Abstraction Problem

The Minimal Abstraction Problem

Given a family of abstractions $\mathcal{A}$

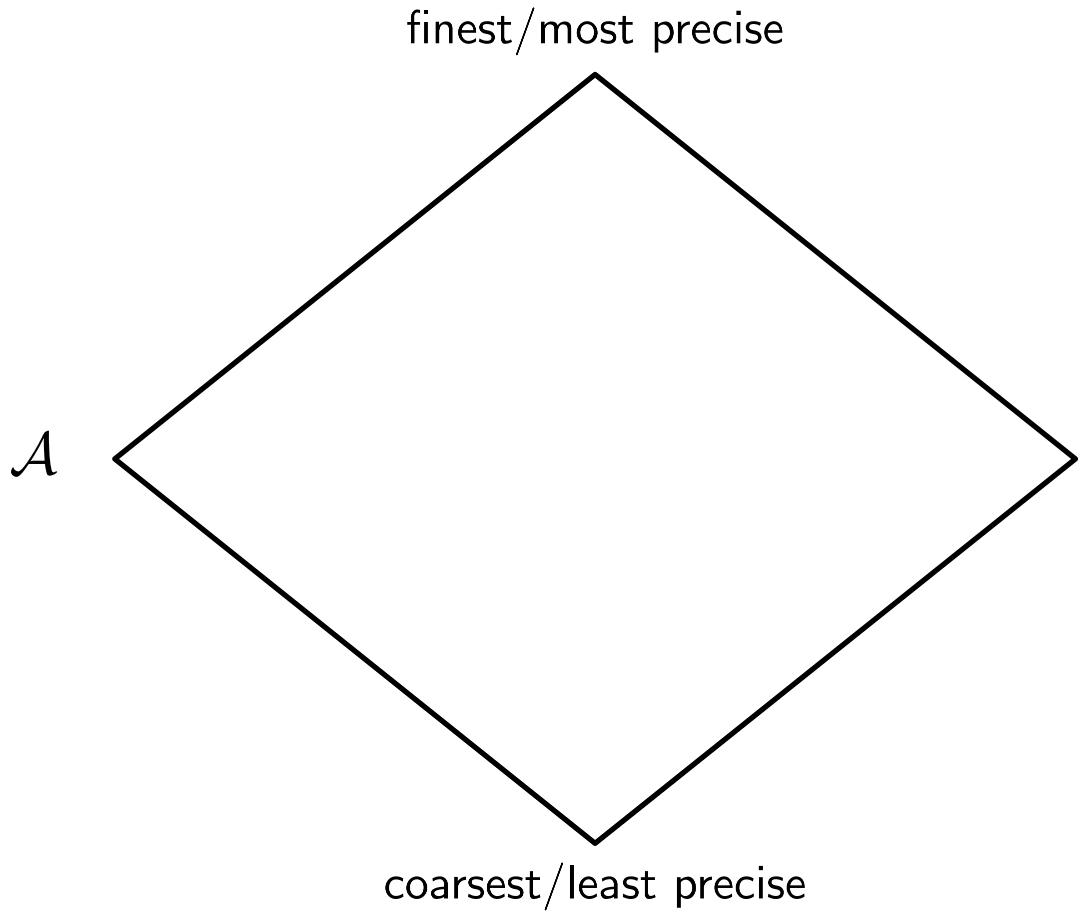
The Minimal Abstraction Problem

Given a family of abstractions $\mathcal{A}$



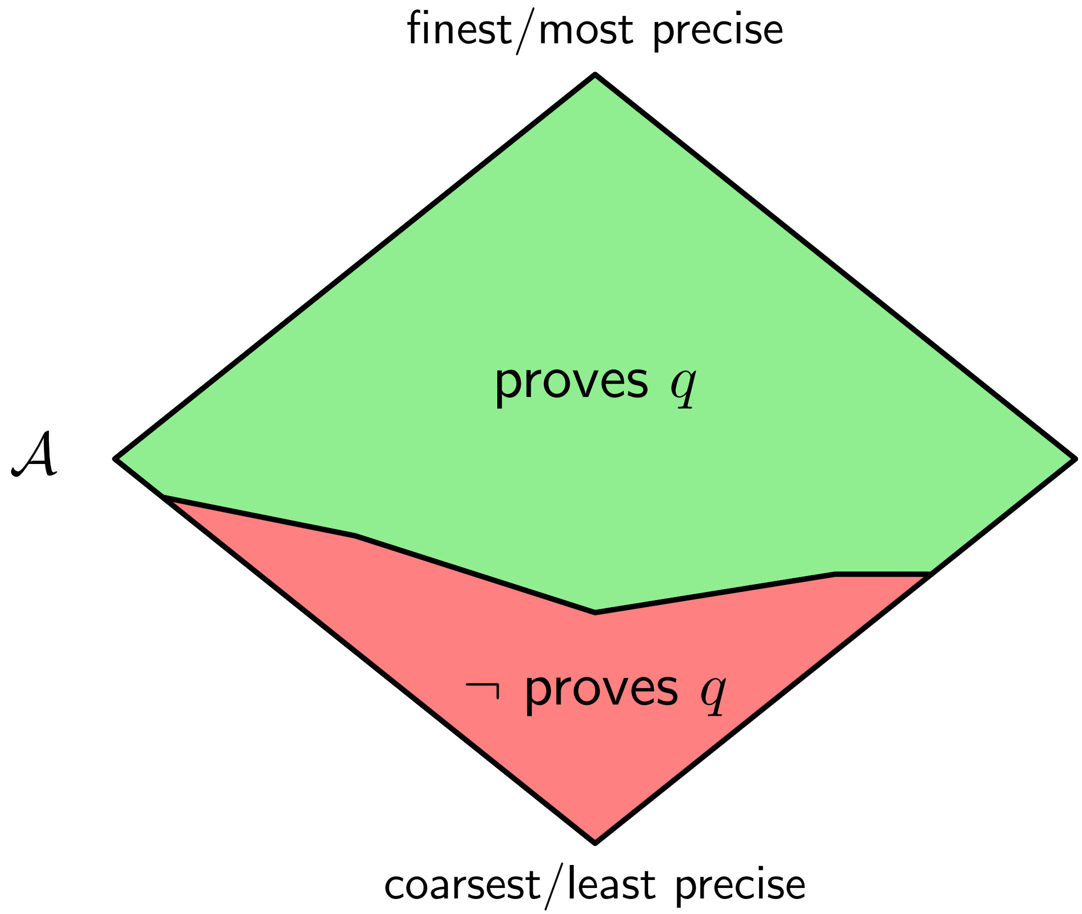
The Minimal Abstraction Problem

Given a family of abstractions $\mathcal{A}$ and a client query q...



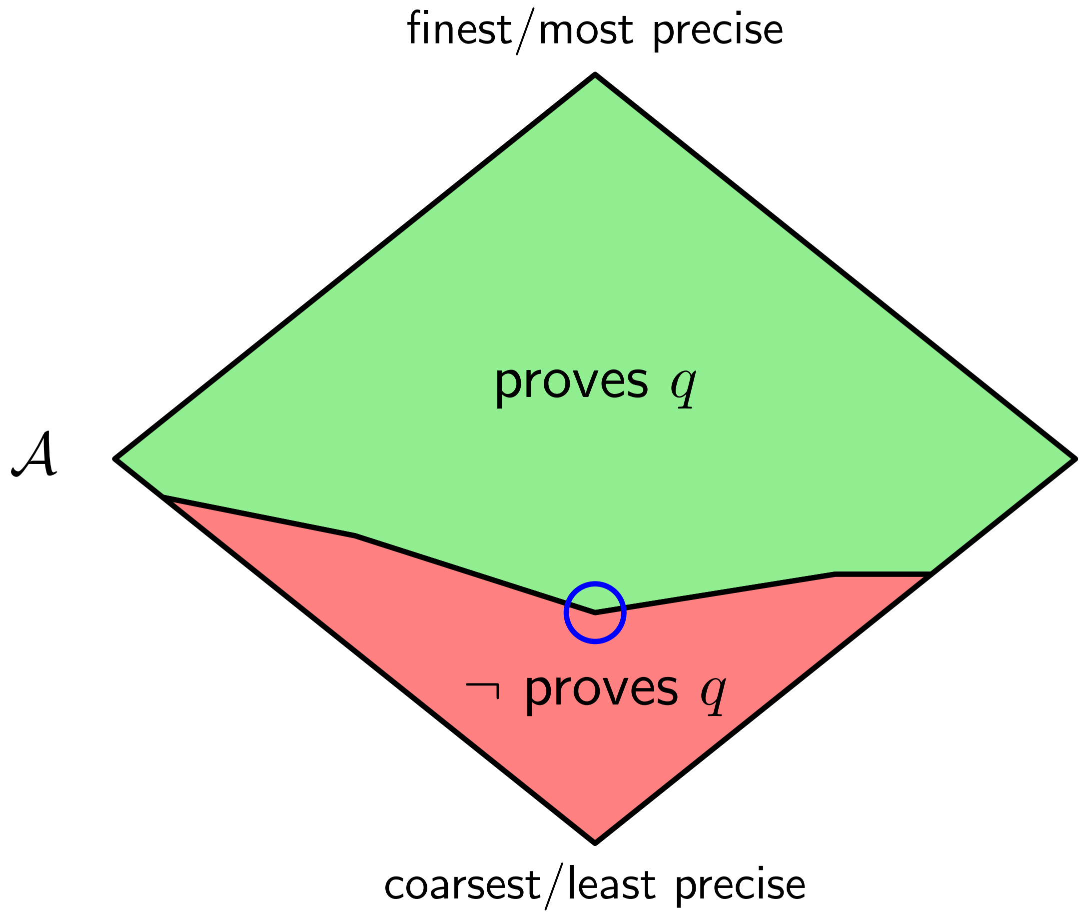
The Minimal Abstraction Problem

Given a family of abstractions $\mathcal{A}$ and a client query q...



The Minimal Abstraction Problem

Given a family of abstractions $\mathcal{A}$ and a client query q...



What is the coarsest abstraction $\mathbf{a} \in \mathcal{A}$ that proves the query q?

Motivating application: race detection

```
    getnew() {           // Thread 1       // Thread 2
h1:   z1 = new C         x = getnew()      y = getnew()
h2:   z2 = new C         x.f = ...         y.f = ...
      return z2
    }
```

Motivating application: race detection

```
    getnew() {            // Thread 1       // Thread 2
h1:   z1 = new C          x = getnew()     y = getnew()
h2:   z2 = new C          x.f = ...        y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Motivating application: race detection

```
    getnew() {              // Thread 1        // Thread 2
h1:   z1 = new C            x = getnew()       y = getnew()
h2:   z2 = new C            x.f = ...          y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Heap abstractions: for each allocation site, context sensitive (1) or not (0)

Motivating application: race detection

```
    getnew() {            // Thread 1        // Thread 2
h1:   z1 = new C          x = getnew()       y = getnew()
h2:   z2 = new C          x.f = ...          y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Heap abstractions: for each allocation site, context sensitive (1) or not (0)

h1	h2
0	0

$\neg$ proves q

Motivating application: race detection

```
    getnew() {              // Thread 1       // Thread 2
h1:   z1 = new C            x = getnew()      y = getnew()
h2:   z2 = new C            x.f = ...         y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Heap abstractions: for each allocation site, context sensitive (1) or not (0)

h1	h2
1	1

proves q

h1	h2
0	0

$\neg$ proves q

Motivating application: race detection

```
    getnew() {            // Thread 1      // Thread 2
h1:   z1 = new C          x = getnew()     y = getnew()
h2:   z2 = new C          x.f = ...        y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Heap abstractions: for each allocation site, context sensitive (1) or not (0)

h1	h2
1	1

proves q

h1	h2
1	0

$\neg$ proves q

h1	h2
0	1

proves q

h1	h2
0	0

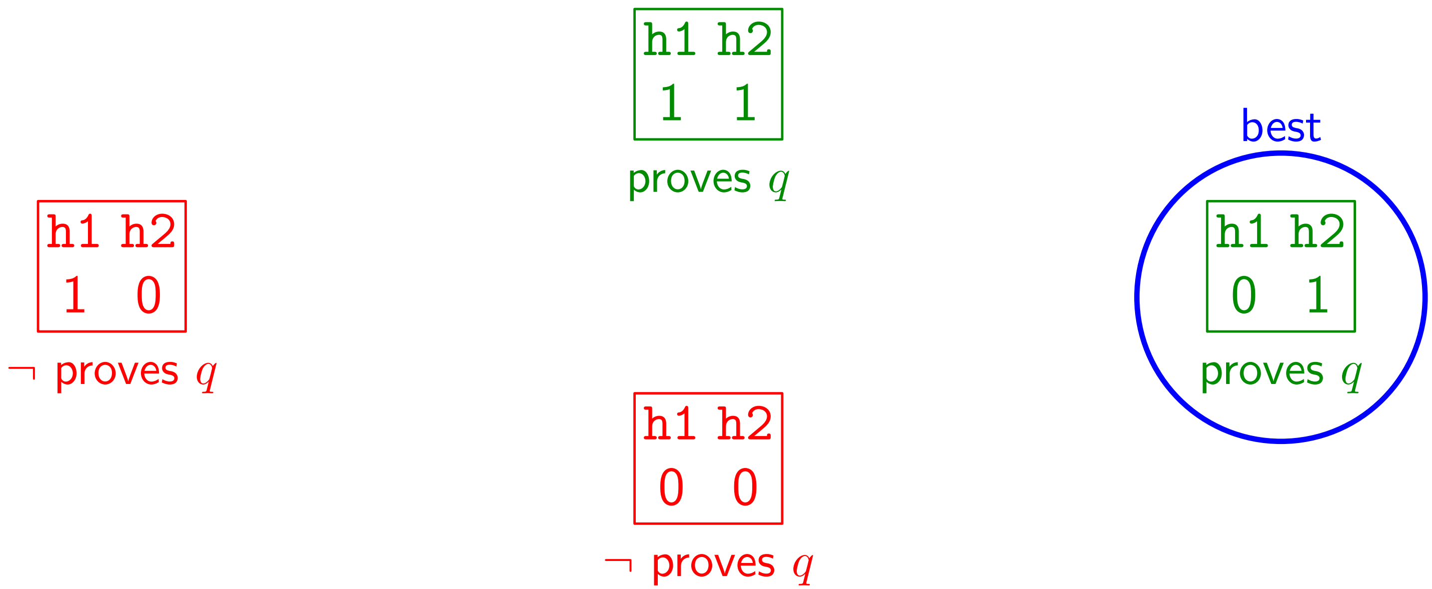
$\neg$ proves q

Motivating application: race detection

```
    getnew() {             // Thread 1       // Thread 2
h1:   z1 = new C           x = getnew()      y = getnew()
h2:   z2 = new C           x.f = ...         y.f = ...
      return z2
    }
```

Query: is there a data race between `x.f = ...` and `y.f = ...`? no

Heap abstractions: for each allocation site, context sensitive (1) or not (0)



The landscape

Motivating problem:

Given a query, try to prove it
as cheaply as possible

The landscape

Motivating problem:

Given a query, try to prove it
as cheaply as possible

Existing solutions:

Abstraction refinement [Guyer & Lin 2003]
[Heintze & Tardieu 2001]
[Sridharan et al. 2005]
[Zheng & Rugina 2008] ...

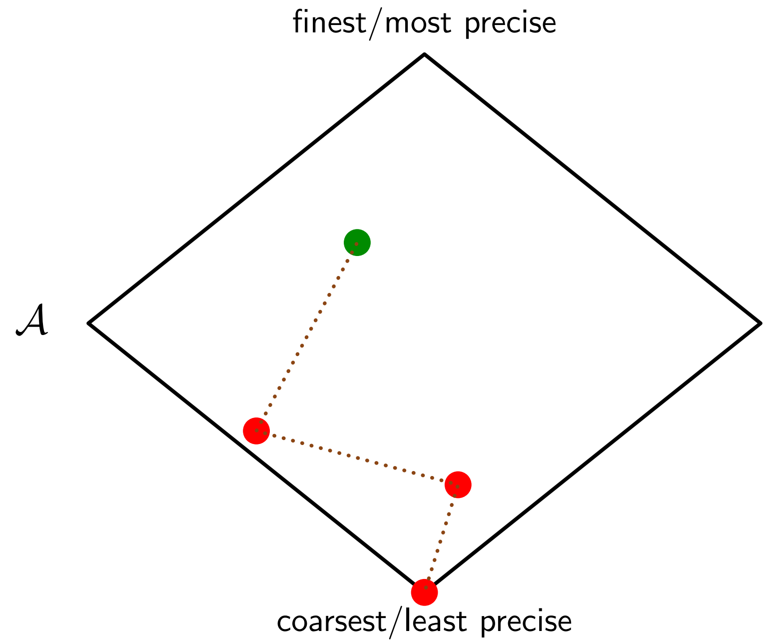
The landscape

Motivating problem:

Given a query, try to prove it
as cheaply as possible

Existing solutions:

Abstraction refinement [Guyer & Lin 2003]
[Heintze & Tardieu 2001]
[Sridharan et al. 2005]
[Zheng & Rugina 2008] ...



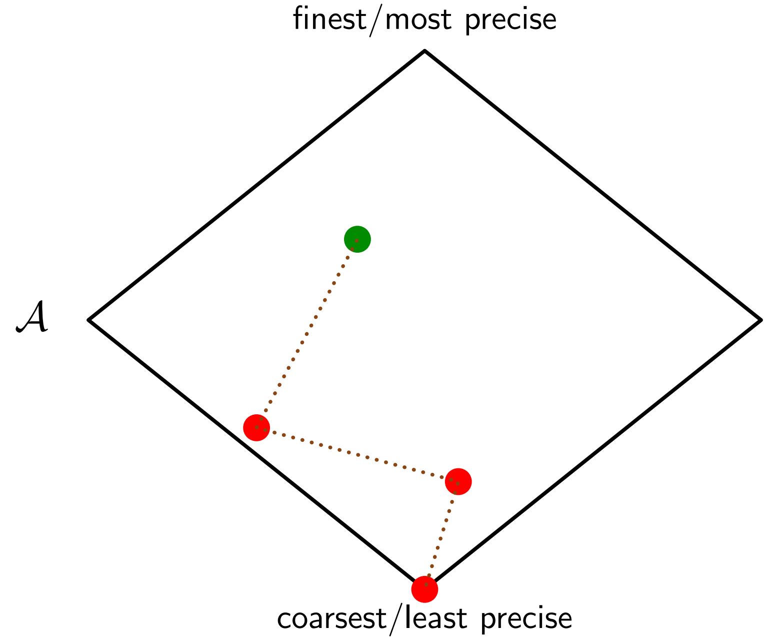
The landscape

Motivating problem:

Given a query, try to prove it
as cheaply as possible

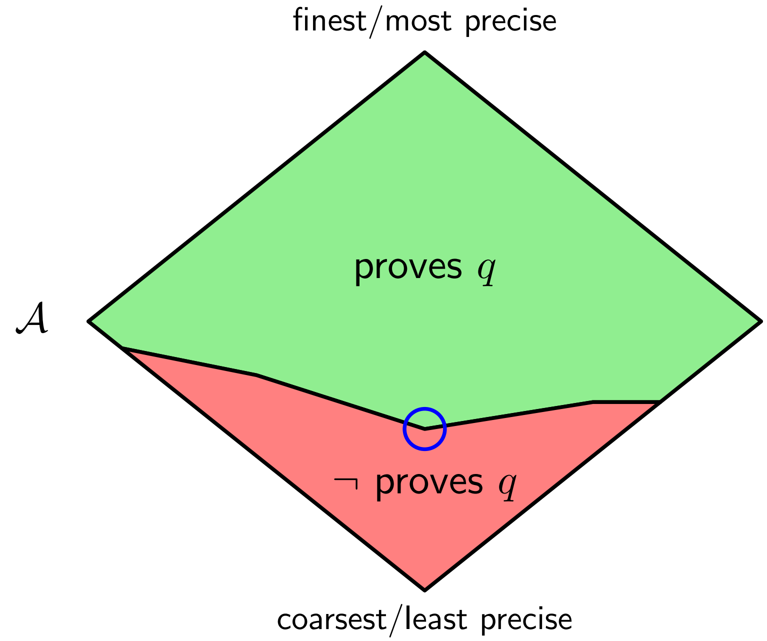
Existing solutions:

Abstraction refinement [Guyer & Lin 2003]
[Heintze & Tardieu 2001]
[Sridharan et al. 2005]
[Zheng & Rugina 2008] ...



Our problem (scientific question):

Given that we've proved a query,
cheapest abstraction in hindsight?



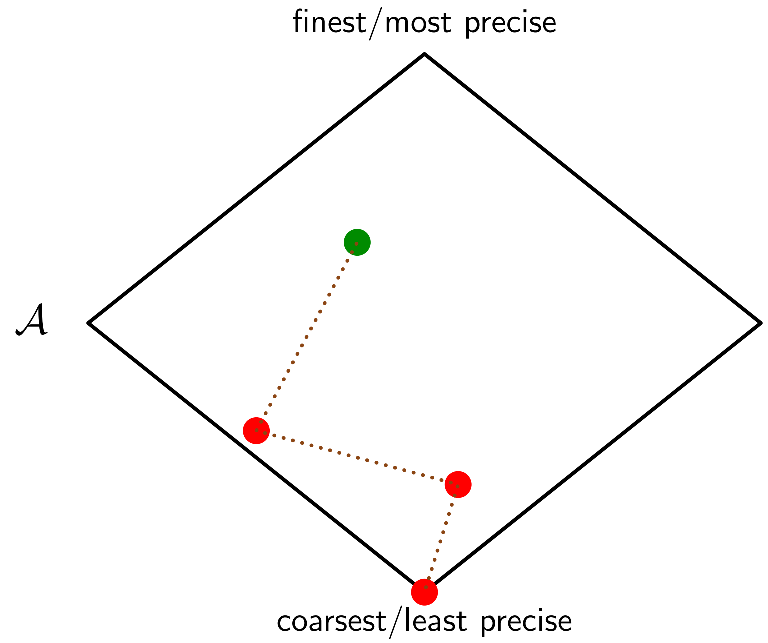
The landscape

Motivating problem:

Given a query, try to prove it as cheaply as possible

Existing solutions:

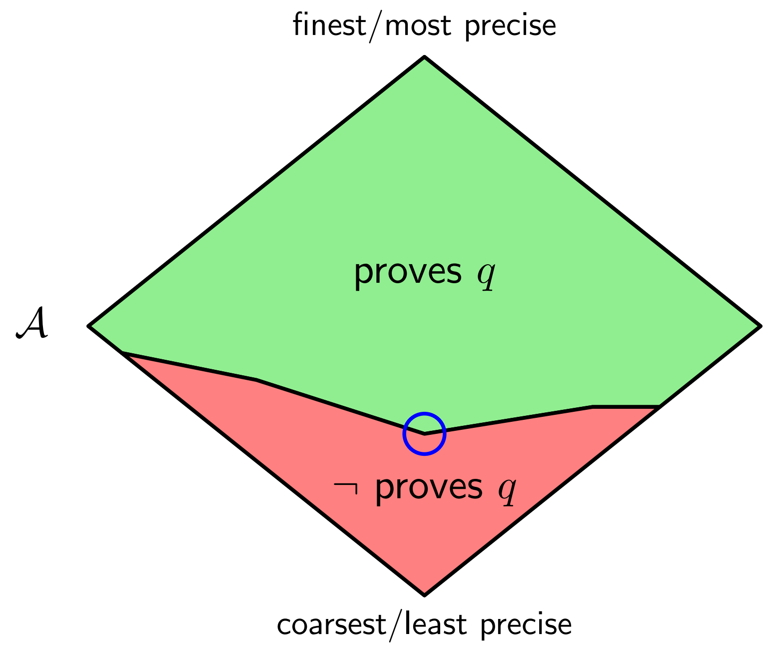
Abstraction refinement [Guyer & Lin 2003] [Heintze & Tardieu 2001] [Sridharan et al. 2005] [Zheng & Rugina 2008] ...



Our problem (scientific question):

Given that we've proved a query, cheapest abstraction in hindsight?

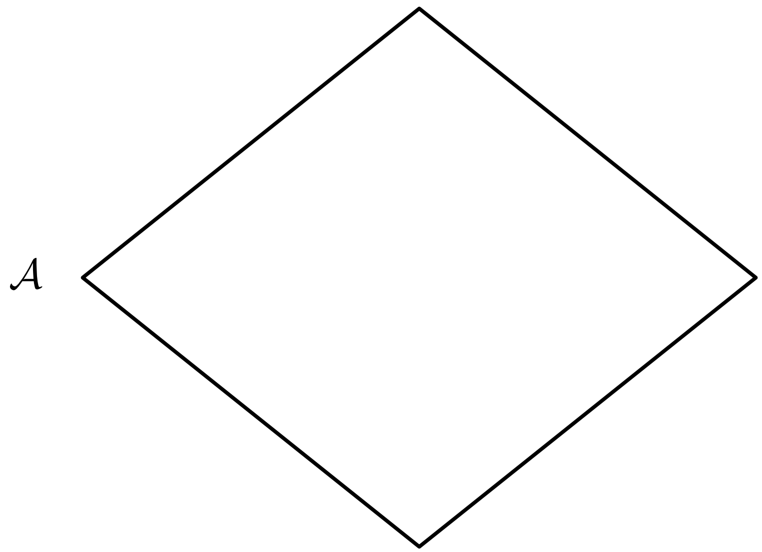
Sufficient/necessary conditions: what aspects of program to model?



Binary representation

Abstraction $\mathbf{a} \in \mathcal{A}$ is a binary vector (subset of components):

$\mathbf{a} = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$ (most precise)

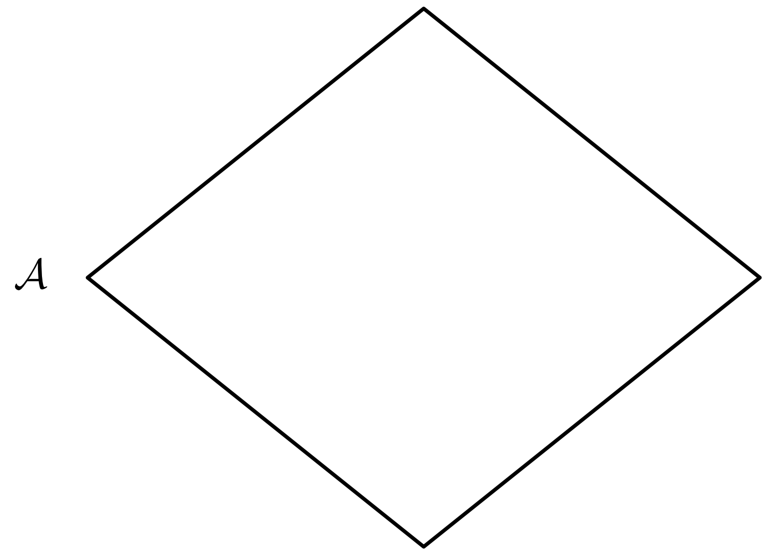


$\mathbf{a} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$ (least precise)

Binary representation

Abstraction $\mathbf{a} \in \mathcal{A}$ is a binary vector (subset of components):

$\mathbf{a} = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1$ (most precise)



$\mathbf{a} = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0$ (least precise)

Examples:

k-limited [Milanova et al. 2002]: treat site context-sensitively?

Predicate abstraction [Ball et al. 2001]: include predicate?

Shape analysis [Sagiv et al. 2002]: treat as abstraction predicate?

Finding a minimal abstraction

Given a static analysis $\mathbf{F}$:

0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0

↓

$\mathbf{F}$

↓

0 (proven) OR 1 (not proven)

Finding a minimal abstraction

Given a static analysis $\mathbf{F}$:

0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0

↓

$\mathbf{F}$

↓

0 (proven) OR 1 (not proven)

Goal: find a minimal abstraction $\mathbf{a}$ (not necessarily unique):

(i) $\mathbf{F}(\mathbf{a}) = 0$ (proves the query)

(ii) For $\mathbf{a}' \prec \mathbf{a}, \mathbf{F}(\mathbf{a}') = 1$ (can't coarsen locally)

Finding a minimal abstraction

Given a static analysis $\mathbf{F}$:

0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0

↓

$\mathbf{F}$

↓

0 (proven) OR 1 (not proven)

Goal: find a minimal abstraction $\mathbf{a}$ (not necessarily unique):

(i) $\mathbf{F}(\mathbf{a}) = 0$ (proves the query)

(ii) For $\mathbf{a}' \prec \mathbf{a}, \mathbf{F}(\mathbf{a}') = 1$ (can't coarsen locally)

Challenge: $|\mathcal{A}| = 2^{\#\text{ components}}$ abstractions to consider

Finding a minimal abstraction

Given a static analysis $\mathbf{F}$:

0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0

↓

F

↓

0 (proven) OR 1 (not proven)

Goal: find a minimal abstraction $\mathbf{a}$ (not necessarily unique):

(i) $\mathbf{F}(\mathbf{a}) = 0$ (proves the query)

(ii) For $\mathbf{a}' \prec \mathbf{a}, \mathbf{F}(\mathbf{a}') = 1$ (can't coarsen locally)

Challenge: $|\mathcal{A}| = 2^{\#\text{ components}}$ abstractions to consider

Approach: machine learning algorithms that exploit randomization

Key theme: sparsity

Sparsity hypothesis:
Only a small fraction of components of $\mathbf{a}$ need to be refined

$$\mathbf{a} = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0$$

Key theme: sparsity

Sparsity hypothesis:
Only a small fraction of components of $\mathbf{a}$ need to be refined

$$\mathbf{a} = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0$$

Main results:

Key theme: sparsity

Sparsity hypothesis:
Only a small fraction of components of $\mathbf{a}$ need to be refined

$$\mathbf{a} = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0$$

Main results:

Theoretical: machine learning algorithms are efficient under sparsity

Key theme: sparsity

> Sparsity hypothesis:
> Only a small fraction of components of $\mathbf{a}$ need to be refined

$$\mathbf{a} = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0$$

Main results:

Theoretical: machine learning algorithms are efficient under sparsity

Empirical: for k-limited race detection,
only 0.4%–2.3% components need to be 1!

Key theme: sparsity

> Sparsity hypothesis:
> Only a small fraction of components of $\mathbf{a}$ need to be refined

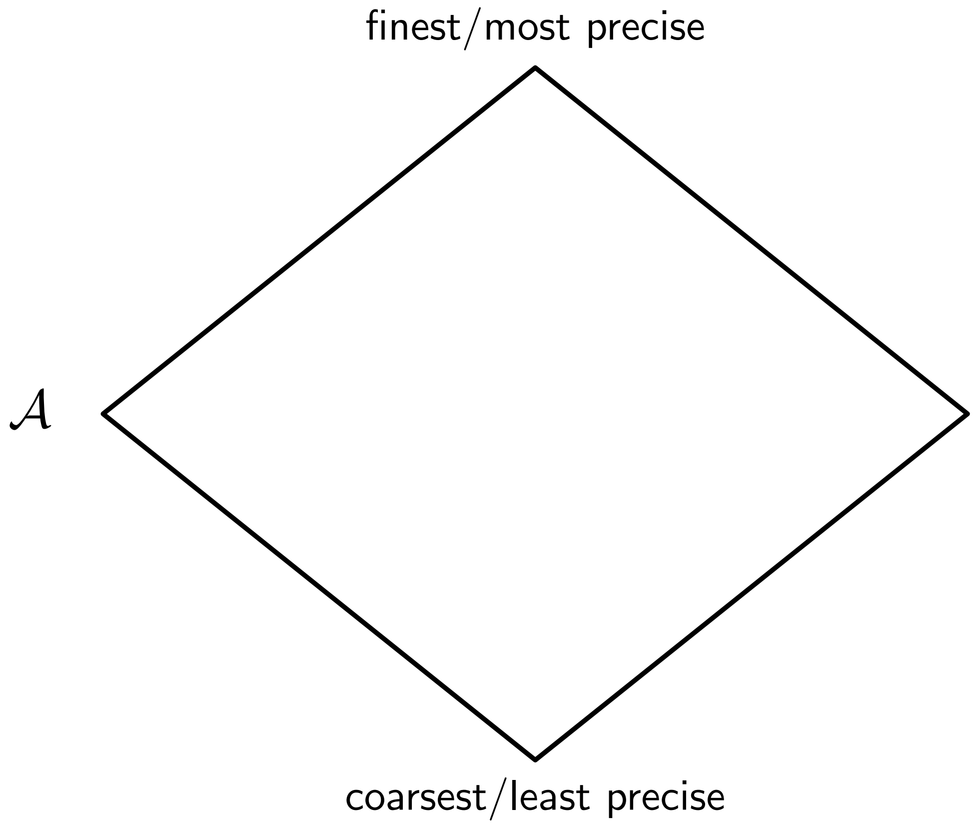
$$\mathbf{a} = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0$$

Main results:

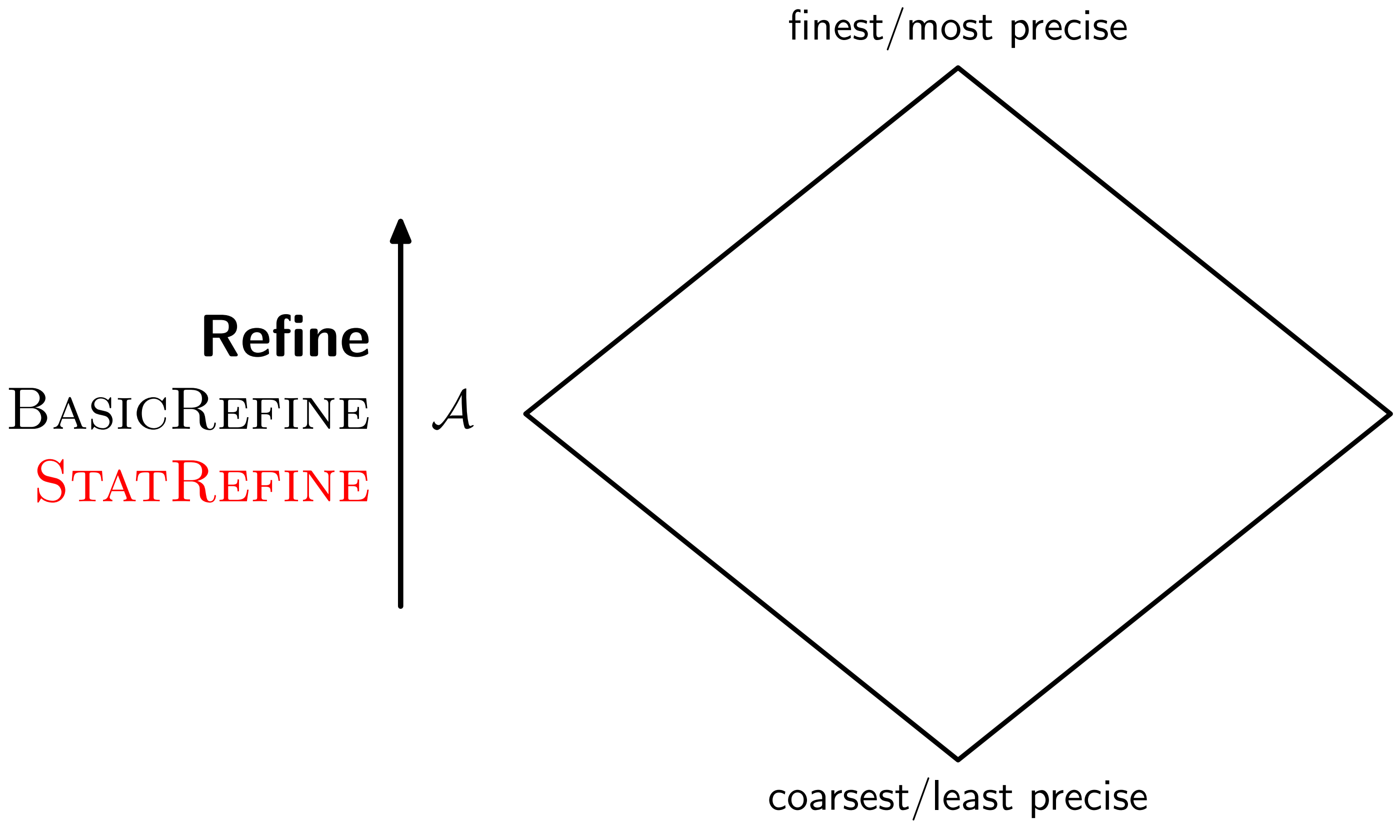
Theoretical: machine learning algorithms are efficient under sparsity

Empirical: for k-limited race detection,
only 0.4%–2.3% components need to be 1!
(effectively "0.004-CFA"–"0.023-CFA")

Algorithms

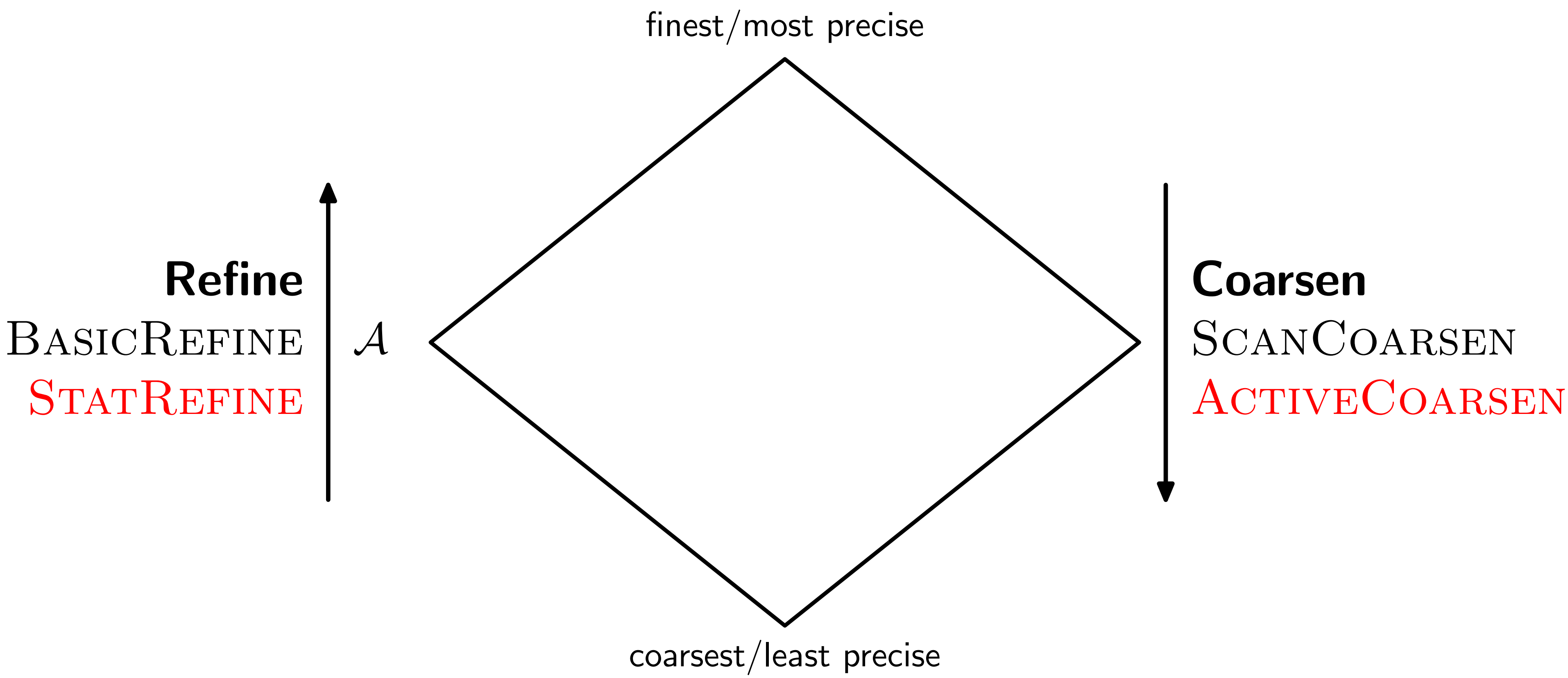


Algorithms



finest/most precise
Refine
BasicRefine
StatRefine
$\mathcal{A}$
coarsest/least precise

Algorithms


finest/most precise
Refine
BasicRefine
StatRefine
$\mathcal{A}$
Coarsen
ScanCoarsen
ActiveCoarsen
coarsest/least precise

BasicRefine

Idea: start with imprecise $\mathbf{a}$, incrementally refine "relevant" components

BasicRefine

Idea: start with imprecise $\mathbf{a}$, incrementally refine "relevant" components

$\mathbf{a} \leftarrow (0, \ldots, 0)$
Loop:
  Run analysis $\mathbf{F}(\mathbf{a})$
  Find relevant components by cause-effect analysis
  Add these components to $\mathbf{a}$

BasicRefine

Idea: start with imprecise $\mathbf{a}$, incrementally refine "relevant" components

$\mathbf{a} \leftarrow (0, \ldots, 0)$
Loop:
  Run analysis $\mathbf{F}(\mathbf{a})$
  Find relevant components by cause-effect analysis
  Add these components to $\mathbf{a}$

Reasonable iterative refinement baseline

BasicRefine

Idea: start with imprecise $\mathbf{a}$, incrementally refine "relevant" components

$\mathbf{a} \leftarrow (0, \ldots, 0)$
Loop:
  Run analysis $\mathbf{F}(\mathbf{a})$
  Find relevant components by cause-effect analysis
  Add these components to $\mathbf{a}$

Reasonable iterative refinement baseline

Solves the motivating problem of proving a new query cheaply

BasicRefine

Idea: start with imprecise $\mathbf{a}$, incrementally refine "relevant" components

$\mathbf{a} \leftarrow (0, \ldots, 0)$
Loop:
  Run analysis $\mathbf{F}(\mathbf{a})$
  Find relevant components by cause-effect analysis
  Add these components to $\mathbf{a}$

Reasonable iterative refinement baseline

Solves the motivating problem of proving a new query cheaply

Does not solve the minimal abstraction problem (it refines too much)

ScanCoarsen

Idea: start with most precise $\mathbf{a}$, incrementally discard components

ScanCoarsen

Idea: start with most precise $\mathbf{a}$, incrementally discard components

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Remove a component from $\mathbf{a}$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add component back permanently

ScanCoarsen

Idea: start with most precise $\mathbf{a}$, incrementally discard components

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Remove a component from $\mathbf{a}$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add component back permanently

Exploits monotonicity of $\mathbf{F}$:

Component whose removal causes $\mathbf{F}(\mathbf{a}) = 1$ must exist in min. abstraction

$\Rightarrow$ never visit a component more than once

ScanCoarsen

Idea: start with most precise $\mathbf{a}$, incrementally discard components

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Remove a component from $\mathbf{a}$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add component back permanently

Exploits monotonicity of $\mathbf{F}$:

Component whose removal causes $\mathbf{F}(\mathbf{a}) = 1$ must exist in min. abstraction

$\Rightarrow$ never visit a component more than once

Problem: takes $O(\#$ components) time (can be $> 10,000 \Rightarrow > 30$ days)

StatRefine

Idea: run $\mathbf{F}$ on random $\mathbf{a}$, learn correlations between components and $\mathbf{F}(\mathbf{a})$

StatRefine

Idea: run $\mathbf{F}$ on random $\mathbf{a}$, learn correlations between components and $\mathbf{F}(\mathbf{a})$

Loop:

Gather n training examples $(\mathbf{a}, \mathbf{F}(\mathbf{a}))$ where $p(\mathbf{a}_j = 1) = \alpha$

Add component j with largest # of $\mathbf{a}$ with $\mathbf{a}_j = 1$ and $\mathbf{F}(\mathbf{a}) = 0$

StatRefine

Idea: run $\mathbf{F}$ on random $\mathbf{a}$, learn correlations between components and $\mathbf{F}(\mathbf{a})$

> Loop:
>
> Gather n training examples $(\mathbf{a}, \mathbf{F}(\mathbf{a}))$ where $p(\mathbf{a}_j = 1) = \alpha$
>
> Add component j with largest # of $\mathbf{a}$ with $\mathbf{a}_j = 1$ and $\mathbf{F}(\mathbf{a}) = 0$

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
0	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	$\Rightarrow$	0
0	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	1	$\Rightarrow$	1
1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	1	0	0	1	$\Rightarrow$	0
1	1	0	0	1	1	0	0	1	1	1	1	1	0	1	0	1	0	0	1	$\Rightarrow$	1
0	1	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
0	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1	1	1	$\Rightarrow$	0
1	0	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	$\Rightarrow$	1
0	0	1	0	0	1	1	1	0	1	1	1	1	1	1	1	0	0	1	1	$\Rightarrow$	1
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	$\Rightarrow$	0
1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0	1	$\Rightarrow$	0
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	$\Rightarrow$	0

StatRefine

Idea: run $\mathbf{F}$ on random $\mathbf{a}$, learn correlations between components and $\mathbf{F}(\mathbf{a})$

Loop:

Gather n training examples $(\mathbf{a}, \mathbf{F}(\mathbf{a}))$ where $p(\mathbf{a}_j = 1) = \alpha$

Add component j with largest # of $\mathbf{a}$ with $\mathbf{a}_j = 1$ and $\mathbf{F}(\mathbf{a}) = 0$

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
0	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	$\Rightarrow$	0
0	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	1	0	0	1	$\Rightarrow$	0
0	1	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1	1	1	$\Rightarrow$	0
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	$\Rightarrow$	0
1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0	1	$\Rightarrow$	0
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	$\Rightarrow$	0

StatRefine

Idea: run $\mathbf{F}$ on random $\mathbf{a}$, learn correlations between components and $\mathbf{F}(\mathbf{a})$

Loop:
- Gather n training examples $(\mathbf{a}, \mathbf{F}(\mathbf{a}))$ where $p(\mathbf{a}_j = 1) = \alpha$
- Add component j with largest # of $\mathbf{a}$ with $\mathbf{a}_j = 1$ and $\mathbf{F}(\mathbf{a}) = 0$

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒	0
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒	0
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒	0
0	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	⇒	0
0	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	⇒	0
1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	1	0	0	1	⇒	0
0	1	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	⇒	0
1	1	1	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1	1	1	⇒	0
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	⇒	0
1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0	1	⇒	0
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	⇒	0
8	9	7	**11**	9	6	7	10	**11**	7	**11**	9	7	10	10	10	9	8	9	9		

StatRefine

Theorem:

$s = \#$ components in the largest minimal abstraction
$d = \#$ components in any minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

StatRefine

Theorem:

$s = \#$ components in the largest minimal abstraction
$d = \#$ components in any minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = \frac{s}{s+1}$

StatRefine

Theorem:

$s = \#$ components in the largest minimal abstraction
$d = \#$ components in any minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = \frac{s}{s+1}$
Obtain $n = \Omega(d^2 \log |\mathbb{J}|)$ examples

StatRefine

Theorem:

$s = \#$ components in the largest minimal abstraction
$d = \#$ components in any minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = \frac{s}{s+1}$
Obtain $n = \Omega(d^2 \log |\mathbb{J}|)$ examples

Then:

StatRefine outputs a minimal abstraction with high probability
in $O(sd^2 \log |\mathbb{J}|)$ time

Significance: s, d are small,
only logarithmic dependence on total $\#$ components

StatRefine

Theorem:

$s = \#$ components in the largest minimal abstraction
$d = \#$ components in any minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = \frac{s}{s+1}$

Obtain $n = \Omega(d^2 \log |\mathbb{J}|)$ examples

Then:

StatRefine outputs a minimal abstraction with high probability in $O(sd^2 \log |\mathbb{J}|)$ time

Significance: s, d are small,
only logarithmic dependence on total $\#$ components

Proof sketch: large deviation bounds $+$ optimization over α

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \Rightarrow 0$

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \dots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

$$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \Rightarrow 0$$
$$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \Rightarrow 1$$

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 $\Rightarrow$ 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 $\Rightarrow$ 1
1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 $\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	⇒	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			⇒	0
0	1	0	1	1	1		1	1	0		1		1	1	1	1		0		0	1	1		0	1	1	1			⇒	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			0		1	1	1	0					1	1			1	1	1			$\Rightarrow$	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \dots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$ 1

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$	1
	1		1	1	0		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \dots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$ 1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$ 1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		1	1			1	1					1	1	1						1	1			0	0	1			$\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$ 1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		1	1			1	1					1	1	1						1	1					1			$\Rightarrow$ 0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$ 1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$ 0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$ 1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$ 0
	1		1	1			1	1					1	1	1						1	1					1			$\Rightarrow$ 0
	1		1	1			0	1					0	1	0						1	1					1			$\Rightarrow$ 1

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \dots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$	1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0
	1		1	1			1	1					1	1	1						1	1					1			$\Rightarrow$	0
	1		1	1			0	1					0	1	0						1	1					1			$\Rightarrow$	1
	1		1	1			1	1					1	1	1						1	1					0			$\Rightarrow$	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

$\mathbf{a} \leftarrow (1, \ldots, 1)$
Loop:
  Try removing each component with probability $1 - \alpha$
  Run analysis $\mathbf{F}(\mathbf{a})$
  If $\mathbf{F}(\mathbf{a}) = 1$: add components back
  Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	$\Rightarrow$	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			$\Rightarrow$	0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			$\Rightarrow$	1
	1		1	1			1	1					1	1	1						1	1			1	1	1			$\Rightarrow$	0
	1		1	1			1	1					1	1	1						1	1					1			$\Rightarrow$	0
	1		1	1			0	1					0	1	0						1	1					1			$\Rightarrow$	1
	1		1	1			1	1					1	1	1						1	1								$\Rightarrow$	0

ActiveCoarsen algorithm

Idea: try to remove a constant fraction of components in each step

> $\mathbf{a} \leftarrow (1, \ldots, 1)$
> Loop:
>   Try removing each component with probability $1 - \alpha$
>   Run analysis $\mathbf{F}(\mathbf{a})$
>   If $\mathbf{F}(\mathbf{a}) = 1$: add components back
>   Else: remove components permanently

Example: $\mathbf{F}(\mathbf{a}) = \neg(\mathbf{a}_4 \wedge \mathbf{a}_9 \wedge \mathbf{a}_{11})$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	⇒	1
1	1	1	1	1	1		1	1	1		1		1	1	1	1		1		1	1	1		1	1	1	1			⇒	0
	1		1	1	1		1	1			1		1	1	1	1					1	1			1	1	1			⇒	0
	1		1	1	1		1	1					1	1	1						1	1			1	1	1			⇒	0
	1		0	0	1		0	0					0	1	1						0	1			1	1	0			⇒	1
	1		1	1			1	1					1	1	1						1	1			1	1	1			⇒	0
	1		1	1			1	1					1	1	1						1	1					1			⇒	0
	1		1	1			0	1					0	1	0						1	1					1			⇒	1
	1		1	1			1	1					1	1	1						1	1								⇒	0
	1		1	1			1	0					1	1	1						1	1								⇒	1

ACTIVECOARSEN analysis

Theorem:

$s = \#$ components in largest minimal abstraction

$|\mathbb{J}| =$ total $\#$ components

ActiveCoarsen analysis

Theorem:

$s = \#$ components in largest minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = e^{-1/s}$

ActiveCoarsen analysis

Theorem:

$s = \#$ components in largest minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = e^{-1/s}$

Then:

ActiveCoarsen outputs a minimal abstraction
in $O(s \log |\mathbb{J}|)$ expected time

Proof sketch: solve recurrence + optimization over α

ActiveCoarsen analysis

Theorem:

$s = \#$ components in largest minimal abstraction
$|\mathbb{J}| =$ total $\#$ components

If:

Set refinement probability $\alpha = e^{-1/s}$

Then:

ActiveCoarsen outputs a minimal abstraction
in $O(s \log |\mathbb{J}|)$ expected time

Proof sketch: solve recurrence $+$ optimization over α

Significance: s is small,
only logarithmic dependence on total $\#$ components

Summary of algorithms

Algorithm	Minimal	Correct	# calls to $\mathbf{F}$
BasicRefine	no	yes	$O(1)$
ScanCoarsen	yes	yes	$O(\lvert\mathbb{J}\rvert)$
StatRefine	high prob.	high prob.	$O(sd^2 \log \lvert\mathbb{J}\rvert)$
ActiveCoarsen	yes	yes	$O(s \log \lvert\mathbb{J}\rvert)$

Summary of algorithms

Algorithm	Minimal	Correct	# calls to $\mathbf{F}$
BasicRefine	no	yes	$O(1)$
ScanCoarsen	yes	yes	$O(\lvert\mathbb{J}\rvert)$
StatRefine	high prob.	high prob.	$O(sd^2 \log \lvert\mathbb{J}\rvert)$
ActiveCoarsen	yes	yes	$O(s \log \lvert\mathbb{J}\rvert)$

ActiveCoarsen: best asymptotic running time

StatRefine: parallelizes more easily, better when s, d very small

Summary of algorithms

Algorithm	Minimal	Correct	# calls to $\mathbf{F}$
BasicRefine	no	yes	$O(1)$
ScanCoarsen	yes	yes	$O(\lvert\mathbb{J}\rvert)$
StatRefine	high prob.	high prob.	$O(sd^2 \log\lvert\mathbb{J}\rvert)$
ActiveCoarsen	yes	yes	$O(s \log\lvert\mathbb{J}\rvert)$

ActiveCoarsen: best asymptotic running time

StatRefine: parallelizes more easily, better when s, d very small

Extensions:

- Adapatively refinement probability α
- Sharing computation across multiple queries

Experimental setup

Application: static race detector of [Naik et al. 2006]

Pointer analysis using k-object-sensitivity or k-CFA with heap cloning

Combination of call graph, may alias, thread escape, may happen in parallel

Experimental setup

Application: static race detector of [Naik et al. 2006]

Pointer analysis using k-object-sensitivity or k-CFA with heap cloning

Combination of call graph, may alias, thread escape, may happen in parallel

Benchmark statistics (determines # components in abstraction):

	# alloc sites	# call sites
`hedc`	1,580	7,195
`weblech`	2,584	12,405
`lusearch`	2,873	13,928

Experimental setup

Application: static race detector of [Naik et al. 2006]

Pointer analysis using k-object-sensitivity or k-CFA with heap cloning

Combination of call graph, may alias, thread escape, may happen in parallel

Benchmark statistics (determines # components in abstraction):

	# alloc sites	# call sites
`hedc`	1,580	7,195
`weblech`	2,584	12,405
`lusearch`	2,873	13,928

Number of races:

	`hedc`	`weblech`	`lusearch`
0-CFA	21,335	27,941	37,632
1-CFA	17,837	8,208	31,866
diff. (queries)	**3,498**	**19,733**	**5,766**
1-OBJ	17,137	8,063	31,428
2-OBJ	16,124	5,523	20,929
diff. (queries)	**1,013**	**2,540**	**10,499**

Experimental results (all queries)

Setting: find **one** abstraction to prove **all** queries

How large is abstraction produced by

BasicRefine (non-minimal, deterministic) and
ActiveCoarsen (minimal, randomized)?

Experimental results (all queries)

Setting: find **one** abstraction to prove **all** queries

How large is abstraction produced by

BasicRefine (non-minimal, deterministic) and

ActiveCoarsen (minimal, randomized)?

k-CFA:

	total # components	BasicRefine	ActiveCoarsen (minimal)
`hedc`	8,775	7,270 (83%)	90 (**1.0%**)
`weblech`	14,989	12,737 (85%)	157 (**1.0%**)
`lusearch`	16,801	14,864 (88%)	250 (**1.5%**)

Experimental results (all queries)

Setting: find **one** abstraction to prove **all** queries

How large is abstraction produced by

BasicRefine (non-minimal, deterministic) and

ActiveCoarsen (minimal, randomized)?

k-CFA:

	total # components	BasicRefine	ActiveCoarsen (minimal)
`hedc`	8,775	7,270 (83%)	90 (**1.0%**)
`weblech`	14,989	12,737 (85%)	157 (**1.0%**)
`lusearch`	16,801	14,864 (88%)	250 (**1.5%**)

k-object-sensitivity:

	total # components	BasicRefine	ActiveCoarsen (minimal)
`hedc`	1,580	906 (57%)	37 (**2.3%**)
`weblech`	2,584	1,768 (68%)	48 (**1.9%**)
`lusearch`	2,873	2,085 (73%)	56 (**1.9%**)

Experimental results (breakdown by query)

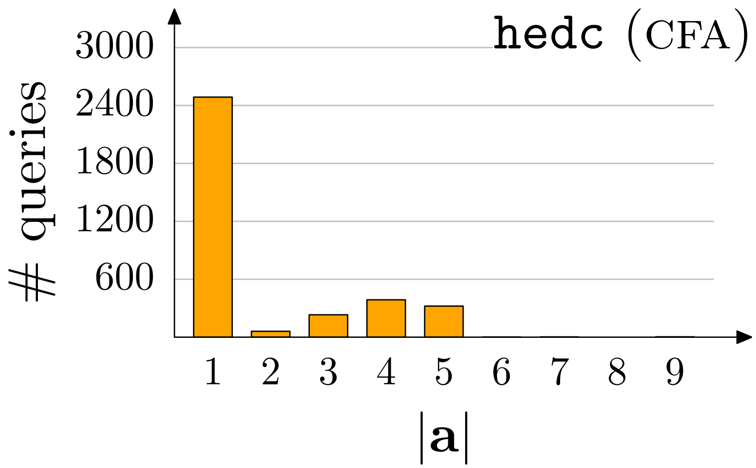
Setting: find **one** abstraction to prove **one** query

How large are the per-query minimal abstractions?

Experimental results (breakdown by query)

Setting: find **one** abstraction to prove **one** query

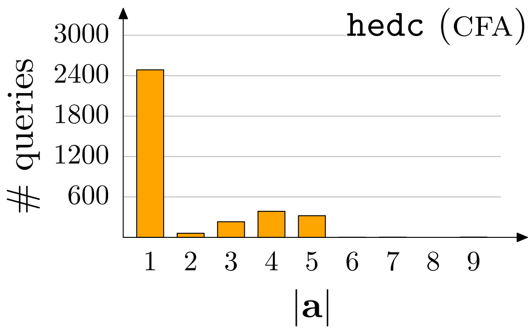
How large are the per-query minimal abstractions?

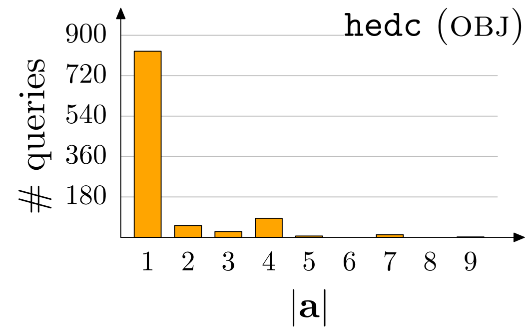


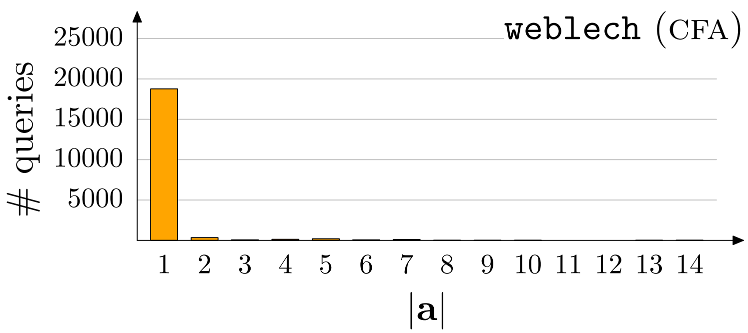
Experimental results (breakdown by query)

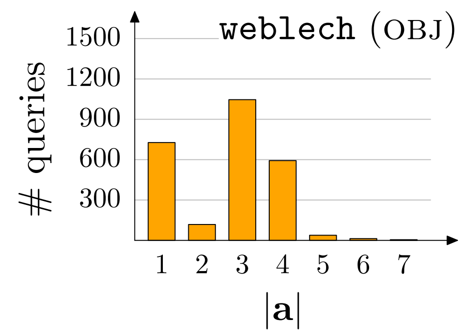
Setting: find **one** abstraction to prove **one** query

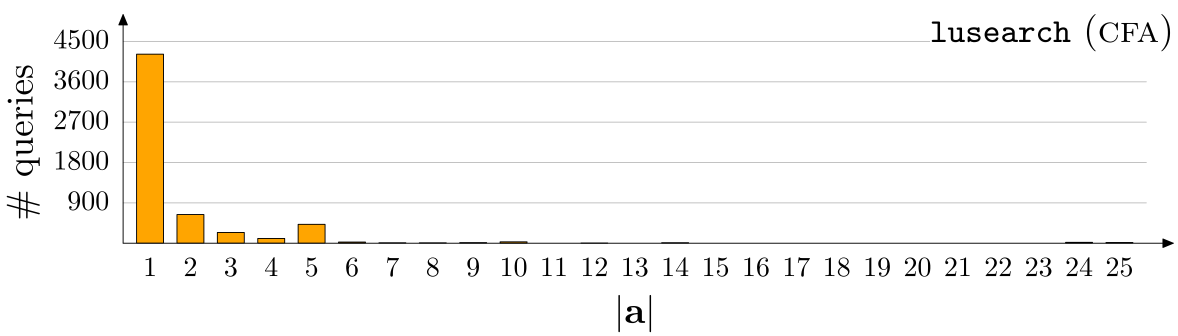
How large are the per-query minimal abstractions?

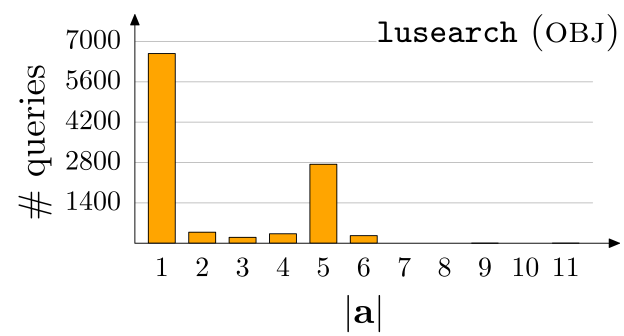












Conclusion

- Motivating problem: to scale static analyses, need cheap abstractions
- Scientific question: what's the minimal abstraction to prove a query?

Conclusion

- Motivating problem: to scale static analyses, need cheap abstractions
- Scientific question: what's the minimal abstraction to prove a query?
- **Sparsity**: very few components are needed
  - Theoretical result: leads to efficient machine learning algorithms
  - Empirical result: leads to cheap abstractions

Conclusion

- Motivating problem: to scale static analyses, need cheap abstractions
- Scientific question: what's the minimal abstraction to prove a query?
- **Sparsity**: very few components are needed
  - Theoretical result: leads to efficient machine learning algorithms
  - Empirical result: leads to cheap abstractions
- Future work: tackle motivating problem with information gathered from minimal abstractions