babble

learning better abstractions with e-graphs and anti-unification

david cao†, rose kunkel†, chandrakana nandi, max willsey, zachary tatlock, nadia polikarpova

uc san diego university of washington certora, inc.
† equal contribution
input 1
[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
move -2 -1.5 (rotate 90 line),
move 2 -1.5 (rotate 90 line),
move 0 2 (x-scale 3 line)]
<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
</tr>
</thead>
</table>
| `[scale 5 circle,  
move -2 -1.5 circle,  
move 2 -1.5 circle,  
move 0 2 (x-scale 3 circle)]` | `[scale 5 circle,  
move -2 -1.5 (rotate 90 line),  
move 2 -1.5 (rotate 90 line),  
move 0 2 (x-scale 3 line)]` |
input 1

[ scale 5 circle,
mov -2 \ -1.5 \ circle,
mov 2 \ -1.5 \ circle,
mov 0 \ 2 \ (x-scale \ 3 \ circle) ]

input 2

[ scale 5 circle,
mov -2 \ -1.5 \ (rotate \ 90 \ line),
mov 2 \ -1.5 \ (rotate \ 90 \ line),
mov 0 \ 2 \ (x-scale \ 3 \ line) ]
input 1
[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
moves 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
move -2 -1.5 (rotate 90 line),
move 2 -1.5 (rotate 90 line),
moves 0 2 (x-scale 3 line)]
common pattern: face, where mouth is stretched out eyes

input 1
[scale 5 circle,
mové -2 -1.5 circle,
mové 2 -1.5 circle,
mové 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
mové -2 -1.5 (rotate 90 line),
mové 2 -1.5 (rotate 90 line),
mové 0 2 (x-scale 3 line)]
let face = $\lambda$ \textit{shape} →  
[ scale 5 circle,  
move -2 -1.5 (rotate 90 \textit{shape}),  
move 2 -1.5 (rotate 90 \textit{shape}),  
move 0 2 (x-scale 3 \textit{shape}) ]

face circle

face line

let square = ...  
in face square

etc.
input 1
[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
move -2 -1.5 (rotate 90 line),
move 2 -1.5 (rotate 90 line),
move 0 2 (x-scale 3 line)]

let face = λshape →
[scale 5 circle,
move -2 -1.5 (rotate 90 shape),
move 2 -1.5 (rotate 90 shape),
move 0 2 (x-scale 3 shape)]

face circle

face line

let square = ...
in face square

etc.
let face = \( \lambda \text{shape} \rightarrow \) 
[ 
  [scale 5 circle, 
  move -2 -1.5 (rotate 90 \( \text{shape} \)),
  move 2 -1.5 (rotate 90 \( \text{shape} \)),
  move 0 2 (x-scale 3 \( \text{shape} \))]
]

face circle

face line

let square = ... in face square

e etc.

humans are really good at this!
input 1

```
[scale 5 circle, 
move -2 -1.5 circle, 
move 2 -1.5 circle, 
move 0 2 (x-scale 3 circle)]
```

input 2

```
[scale 5 circle, 
move -2 -1.5 (rotate 90 line), 
move 2 -1.5 (rotate 90 line), 
move 0 2 (x-scale 3 line)]
```

```
let face = \(shape \rightarrow \)
[scale 5 circle, 
move -2 -1.5 (rotate 90 shape), 
move 2 -1.5 (rotate 90 shape), 
move 0 2 (x-scale 3 shape)]
```

```
face circle
```

```
face line
```

```
let square = ... in face square
```

humans are really good at this!

how can we get computers to do this?

etc.
library learning

let face = λshape → [scale 5 circle,  
  move -2 -1.5 (rotate 90 shape),  
  move 2 -1.5 (rotate 90 shape),  
  move 0 2 (x-scale 3 shape)]

face circle

face line

algorithm to learn "best" abstractions
Algorithm to learn "best" abstractions

i.e. best compression: minimize abstraction + program size

library learning
algorithm to learn "best" abstractions
i.e. best compression: minimize abstraction + program size

```plaintext
let face = \( \lambda \text{shape} \rightarrow \)
[scale 5 circle,
move -2 -1.5 (rotate 90 \( \text{shape} \)),
move 2 -1.5 (rotate 90 \( \text{shape} \)),
move 0 2 (x-scale 3 \( \text{shape} \))]

face circle

face line
```
who cares about library learning?

fpga design
learn best library of operations to optimize hardware for
pick 2:
map  filter  foldl

improved program synthesis
use past synthesis solutions to learn improved DSLs

modeling human perception
make algorithm to examine how humans recognize visual structure

Let face = λshape →
[scale 5 circle, move -2 -1.5 circle, move 2 -1.5 circle, move 0 2 (x-scale 3 circle)]

[Ellis et al. 2021]

14
what's the challenge?
how does babble work?
how well does it work?

algorithm to learn "best" abstractions
i.e. best compression: minimize abstraction +

let face = λshape →
[scale 5 circle,
move -2 -1.5 (rotate 90 shape),
move 2 -1.5 (rotate 90 shape),
move 0 2 (x-scale 3 shape)]
what's the challenge?

how does babble work?

how well does it work?

let face = \( \lambda \text{shape} \rightarrow \) 
[scale 5 circle, 
move -2 -1.5 \text{circle}, 
moved 2 -1.5 \text{circle},
moved 0 2 (x-scale 3 \text{circle})]

algorithm to learn "best" abstractions
i.e. best compression: minimize abstraction + program size

face circle
what's the challenge?

```plaintext
[scale 5 circle,
 move -2 -1.5 circle,
 move 2 -1.5 circle,
 move 0 2 (x-scale 3 circle)]
```

a non-exhaustive list of requirements:

```plaintext
[scale 5 circle,
 move -2 -1.5 (rotate 90 line),
 move 2 -1.5 (rotate 90 line),
 move 0 2 (x-scale 3 line)]
```
what's the challenge?

A non-exhaustive list of requirements:

- **be scalable**
  run reasonably quickly on large corpuses with complex programs
what's the challenge?

a non-exhaustive list of requirements:

be scalable
run reasonably quickly on large corpuses with complex programs

learn abstractions in subterms
not just abstractions which can be applied at the top level
what's the challenge?

A non-exhaustive list of requirements:

**be scalable**
run reasonably quickly on large corpuses with complex programs

**learn abstractions in subterms**
not just abstractions which can be applied at the top level

**handles nested abstractions**
allowing for common structure across abstractions themselves to be shared
what's the challenge?

what babble tackles:

incorporating semantic equivalence
what's the challenge?

```
[scale 5 circle,
  move -2 -1.5 circle,
  move 2 -1.5 circle,
  move 0 2 (x-scale 3 circle)]
```

```
[scale 5 circle,
  move -2 -1.5 (rotate 90 line),
  move 2 -1.5 (rotate 90 line),
  move 0 2 (x-scale 3 line)]
```
what's the challenge?
syntactic alignment

input 1
[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
move -2 -1.5 (rotate 90 line),
move 2 -1.5 (rotate 90 line),
move 0 2 (x-scale 3 line)]
what's the challenge?
syntactic alignment

input 1
[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

input 2
[scale 5 circle,
move -2 -1.5 (rotate 90 line),
move 0 2 (rotate 90 line),
move 0 2 (x-scale 3 line)]

arg 1: circle (rotate 90 line)
what's the challenge?
syntactic alignment

Input 1:

```
[scale 5 circle,
  move -2 -1.5 circle,
  move 2 -1.5 circle,
  move 0 2 (x-scale 3 circle)]
```

Input 2:

```
[scale 5 circle,
  move -2 -1.5 (rotate 90 line),
  move 2 -1.5 (rotate 90 line),
  move 0 2 (x-scale 3 line)]
```

Arg 1:

```
circle (rotate 90 line)
```

Arg 2:

```
circle line
```
what's the challenge?
syntactic alignment

Let face-ish = \( \lambda \text{eye} \ \text{mouth} \rightarrow \)

\[
\begin{align*}
\text{[scale 5 circle, } & \text{move } -2 \ -1.5 \ \text{circle}, \\
& \text{move } 2 \ -1.5 \ \text{circle}, \\
& \text{move } 0 \ 2 \ (x\text{-scale } 3 \ \text{circle})]
\end{align*}
\]

\[
\begin{align*}
\text{face-ish } & \text{circle circle}
\end{align*}
\]

\[
\begin{align*}
\text{[scale 5 circle, } & \text{move } -2 \ -1.5 \ \text{eye}, \\
& \text{move } 2 \ -1.5 \ \text{eye}, \\
& \text{move } 0 \ 2 \ (x\text{-scale } 3 \ \text{mouth})]
\end{align*}
\]

\[
\begin{align*}
\text{face-ish } & \ (\text{rotate } 90 \ \text{line}) \ \text{line}
\end{align*}
\]
what's the challenge?
syntactic alignment

purely syntactic

let face-ish = \( \text{eye} \ \text{mouth} \rightarrow \)
[ scale 5 circle, move -2 -1.5 \( \text{eye} \),
move 2 -1.5 \( \text{eye} \),
move 0 2 (x-scale 3 \( \text{mouth} \)) ]

face-ish circle circle

ideal

let face = \( \text{shape} \rightarrow \)
[ scale 5 circle,
move -2 -1.5 (rotate 90 \( \text{shape} \)),
move 2 -1.5 (rotate 90 \( \text{shape} \)),
move 0 2 (x-scale 3 \( \text{shape} \)) ]

face circle

face-ish (rotate 90 line) line

face line
what's the challenge?
syntactic alignment

purely syntactic

let face-ish = \( \text{let face-ish = } \lambda \text{ eye mouth } \rightarrow \)

[scale 5 circle,
move -2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

let face-ish = \( \text{let face-ish = } \lambda \text{ eye mouth } \rightarrow \)

[scale 5 circle,
move -2 -1.5 \text{ eye},
move 2 -1.5 \text{ eye},
move 0 2 (x-scale 3 \text{ mouth})]

face-ish circle circle

[rotate 90 line)

face-ish (rotate 90 line) line

face-ish circle circle

face-ish (rotate 90 line) line

face-ish circle circle

ideal

let face = \( \text{let face = } \lambda \text{ shape } \rightarrow \)

[scale 5 circle,
move -2 -1.5 (rotate 90 shape),
move 2 -1.5 (rotate 90 shape),
move 0 2 (x-scale 3 shape)]

face circle

face line

face circle

face line

face circle
what's the challenge? syntactic alignment

[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

ideal

let face = \( \lambda \text{shape} \rightarrow \)
[scale 5 circle,
move -2 -1.5 (rotate 90 shape),
move 2 -1.5 (rotate 90 shape),
move 0 2 (x-scale 3 shape)]

face circle

face line
what's the challenge?
syntactic alignment

Let face = λshape →
[scale 5 circle,
move -2 -1.5 (rotate 90 shape),
move 2 -1.5 (rotate 90 shape),
move 0 2 (x-scale 3 shape)]

ideal

face circle

face line
what's the challenge?
syntactic alignment

let face = \( \lambda \text{shape} \rightarrow \)
[ scale 5 circle,
move -2 -1.5 (rotate 90 \text{shape}),
move 2 -1.5 (rotate 90 \text{shape}),
move 0 2 (x-scale 3 \text{shape}) ]

https://www.example.com

semantically equivalent version of input shares common structure!
what's the challenge?

A library learning approach which discovers **more precise structure** by considering **semantically equivalent versions** of our programs.

\[
\text{ideal face} = \lambda \text{shape} \rightarrow \\
\left[ \text{scale 5 circle, move } -2 -1.5 \text{ (rotate 90 circle), move } 2 -1.5 \text{ (rotate 90 circle), move } 0 2 \text{ (x-scale 3 circle)} \right] \\
\]

\[
\text{face circle} = \\
\left[ \text{scale 5 circle, move } -2 -1.5 \text{ (rotate 90 circle), move } 2 -1.5 \text{ (rotate 90 circle), move } 0 2 \text{ (x-scale 3 circle)} \right] \\
\]

\[
\text{face line} = \\
\left[ \text{scale 5 circle, move } -2 -1.5 \text{ (rotate 90 line), move } 2 -1.5 \text{ (rotate 90 line), move } 0 2 \text{ (x-scale 3 line)} \right] \\
\]
what's the challenge?

how does babble work?

how well does it work?
how does babble work?

intuition

\[\text{input 1, version 2} \equiv \text{input 1, version 3}\]

\[\text{idea: consider every equivalent version of our inputs}\]
how does babble work?
changing the problem

let face = λshape →
[scale 5 circle,
  move -2 -1.5 (rotate 90 shape),
  move 2 -1.5 (rotate 90 shape),
  move 0 2 (x-scale 3 shape)]

face circle

face line
how does babble work?
changing the problem

user inputs

program corpus

list of equivalences

circle = rotate 90 circle
?x = scale 1 ?x
...

babble

babble outputs

learned library

let face = λshape → face circle face line

rewritten inputs
how does babble work?

contribution 1: library learning modulo theory (LLMT)
how does babble work?
library learning modulo theory (LLMT)

user inputs

- program corpus
- list of equivalences
  - circle ≡ rotate 90 circle
  - ?x = scale 1 ?x
  - ...

babble

learned library

let face = λshape → face circle

rewritten inputs

face circle  face line
How does babble work?

Library learning modulo theory (LLMT)

User inputs:
- Program corpus
- List of equivalences
  - circle = rotate 90 circle
  - ?x = scale 1 ?x
  - ...

Insight:
- Use e-graphs
  - To create and store infinite equivalent input variants

Babble:

Rewritten inputs:
- Let face = λ shape →
- 
- Face circle
- Face line

Learned library:
how does babble work?
library learning modulo theory (LLMT)

**insight**

**use e-graphs**

to create and store infinite equivalent input variants

circle \equiv \text{rotate 90 circle}

?x \equiv \text{scale 1 ?x}

"#

**contribution 2**

**e-graph anti-unification**

to propose candidate abstractions in the presence of input variants

\[
\text{let face} = \lambda \text{shape} \rightarrow \text{rotate 90 ?x}
\]

rewritten inputs

face circle  
face line

learned library

...
how does babble work?
library learning modulo theory (LLMT)

insight
**use e-graphs**
to create and store infinite equivalent input variants

contribution 2
**e-graph anti-unification**
to propose candidate abstractions in the presence of input variants

contribution 3
**targeted CSE**
to pick the best programs and abstractions

user inputs
- program corpus
- list of equivalences
  - circle = rotate 90 circle
  - ?x = scale 1 ?x
  - ...

babble outputs
- learned library
- rewritten inputs
  - face circle
  - face line

let face = \(\lambda\)shape → ...
how does babble work?
library learning modulo theory (LLMT)

user inputs

program corpus

list of equivalences

"circle ≡ rotate 90 circle
?x = scale 1 ?x"
...

insight

**use e-graphs**

to create and store infinite equivalent input variants

contribution 2

**e-graph anti-unification**

to propose candidate abstractions in the presence of input variants

contribution 3

**targeted CSE**

to pick the best programs and abstractions

babble outputs

learned library

rewritten inputs

let face = λshape ➞

face circle

face line

let f = λx -> ... in
let f = λx -> ... in
let f = λx -> ... in
f 2
how does babble work?
library learning modulo theory (LLMT)

**user inputs**

- program corpus
- list of equivalences

```
circle = rotate 90 circle
?x = scale 1 ?x
...
```

**insight**

**use e-graphs**

to create and store infinite equivalent input variants

- circle
- scale
- rotate 90

**contribution 2**

**e-graph anti-unification**

to propose candidate abstractions in the presence of input variants

- rotate 90 ?x

**contribution 3**

**targeted CSE**

to pick the best programs and abstractions

```
let f = \x \rightarrow \_ in
let f = \_ \rightarrow \_ in
if \ f 2
```

**babble outputs**

- learned library
- rewritten inputs

```
let face = \shape \rightarrow

let face = \shape \rightarrow

face circle
face line
```

see the paper for more on this!
how does babble work?
library learning modulo theory (LLMT)

insight
use e-graphs
- to create and store infinite equivalent input variants

contribution 2
e-graph anti-unification
- to propose candidate abstractions in the presence of input variants

contribution 3
targeted CSE
- to pick the best programs and abstractions

see the paper for more on this!
how do e-graphs work?
how do **e-graphs** work?

why **e-graphs**?

```
[scale 5 circle,
 move -2 -1.5 circle,
 move 2 -1.5 circle,
 move 0 2 (x-scale 3 circle)]
```

```
[scale 5 circle,
 move -2 -1.5 (rotate 90 line),
 move 2 -1.5 (rotate 90 line),
 move 0 2 (x-scale 3 line)]
```
how do **e-graphs** work?

why **e-graphs**?

**user inputs**

- [scale 5 circle,
  move -2 -1.5 circle,
  move 2 -1.5 circle,
  move 0 2 (x-scale 3 circle)]

- [scale 5 circle,
  move -2 -1.5 (rotate 90 circle),
  move 2 -1.5 (rotate 90 circle),
  move 0 2 (x-scale 3 circle)]

- [scale 5 circle,
  move -2 -1.5 (rotate 90 line),
  move 2 -1.5 (rotate 90 line),
  move 0 2 (x-scale 3 line)]

**list of equivalences**

circle = rotate 90 circle

?x = scale 1 ?x

...
how do e-graphs work?

why e-graphs?

user inputs

```
[scale 5 circle, 
move -2 -1.5 circle, 
move 2 -1.5 circle, 
move 0 2 (x-scale 3 circle)]
```

```
[scale 5 circle, 
move -2 -1.5 (rotate 90 circle), 
move 2 -1.5 (rotate 90 circle), 
move 0 2 (x-scale 3 circle)]
```

```
[scale 5 circle, 
move -2 -1.5 (rotate 90 circle), 
move 2 -1.5 (rotate 90 circle), 
move 0 2 (x-scale 3 line)]
```

list of equivalences

circle = rotate 90 circle

?x = scale 1 ?x

...
how do e-graphs work?

why e-graphs?

user inputs

[rotate 90 circle, move -2 -1.5 (rotate 90 circle),
move 2 -1.5 (rotate 90 circle),
move 0 2 (x-scale 3 line)]

[rotate 90 line, move -2 -1.5 (rotate 90 line),
move 2 -1.5 (rotate 90 line),
move 0 2 (x-scale 3 line)]

input 1, version 2

input 1, version 3

challenge: when do we stop rewriting?
how do e-graphs work?

why e-graphs?

user inputs

input 1, version 2

[scale 5 circle,
move -2 -1.5 circle,
move 2 -1.5 circle,
move 0 2 (x-scale 3 circle)]

input 1, version 3

[scale 5 circle,
move -2 -1.5 (rotate 90 circle),
move 2 -1.5 (rotate 90 circle),
move 0 2 (x-scale 3 circle)]

list of equivalences

circle = rotate 90 circle

?x = scale 1 ?x

...
how do e-graphs work?

why e-graphs?

User inputs:

- Input 1, version 2:
  - [scale 5 circle, move -2 -1.5 circle, move 2 -1.5 circle, move 0 2 (x-scale 3 circle)]

- Input 1, version 3:
  - [scale 5 circle, move -2 -1.5 (rotate 90 circle), move 2 -1.5 (rotate 90 circle), move 0 2 (x-scale 3 circle)]

List of equivalences:

- circle = rotate 90 circle
- ?x = scale 1 ?x
  ...

Challenge: when do we stop rewriting?
**how do e-graphs work?**

**why e-graphs?**

**User Inputs**

- [scale 5 circle, move -2 -1.5 circle, move 2 -1.5 circle, move 0 2 (x-scale 3 circle)]
- [scale 5 circle, move -2 -1.5 (rotate 90 circle), move 2 -1.5 (rotate 90 circle), move 0 2 (x-scale 3 circle)]

**List of equivalences**

- circle = rotate 90 circle
- ?x = scale 1 ?x
- ...

**Challenge:**

When do we stop rewriting?
how do e-graphs work?

scale 1 circle

e-graphs compactly represent sets of equivalent terms!

[Tate et al. 2009]
[Willsey et al. 2021]
how do e-graphs work?

e-graphs compactly represent sets of equivalent terms!

circle, scale 1 circle, scale 1 (scale 1 circle), ...

[Tate et al. 2009]
[Willsey et al. 2021]
how do **e-graphs** work?

circle = rotate 90 circle
?x = scale 1 ?x
...

circle, scale 1 circle, scale 1 (scale 1 circle), ...
how do e-graphs work?

circle $\equiv$ rotate 90 circle
?x $\equiv$ scale 1 ?x
...

circle, scale 1 circle, rotate 90 circle,
rotate 90 (scale 1 circle), scale 1 (rotate 90 circle), ...
**how do e-graphs work?**

**equivalences**

\[
\text{circle} = \text{rotate } 90 \text{ circle} \\
?x = \text{scale } 1 \ ?x \\
\ldots
\]

**stop conditions:**
- fixpoint (in some cases)
- timeout, e-graph size (in practice)

```
circle, scale 1 circle, rotate 90 circle, 
rotate 90 (scale 1 circle), scale 1 (rotate 90 circle), ...
```
how do **e-graphs** work?
how does babble work?

**user inputs**
- program corpus
- list of equivalences

**insight**
**use e-graphs**
- to create and store infinite equivalent input variants

**contribution 2**
**e-graph anti-unification**
- to propose candidate abstractions in the presence of input variants

**contribution 3**
**targeted CSE**
- to pick the best programs and abstractions

**babble outputs**
- learned library
- rewritten inputs

**let face = \( \lambda \text{shape} \to \ldots \) in
let f = \( \lambda \text{x} \to \ldots \) in
let f = \( \lambda \text{x} \to \ldots \) in
f 2

let face circle
let face line
how does babble work?

**user inputs**

- program corpus
- list of equivalences

- circle = rotate 90 circle
- ?x = scale 1 ?x
- ...

**use e-graphs**

to create and store infinite equivalent input variants

**insight**

**e-graph anti-unification**

to propose candidate abstractions in the presence of input variants

**contribution 2**

**targeted CSE**

to pick the best programs and abstractions

**contribution 3**

**babble outputs**

- learned library
- rewritten inputs
- face circle
- face line

- let face = \( \lambda \text{shape} \rightarrow \) ...
- let f = \( \lambda x \rightarrow \) ...
- let f = \( \lambda x \rightarrow \) ...
- f 2
how does e-graph anti-unification work?
how does e-graph anti-unification work?

finding common structure

circle $\equiv$ rotate 90 circle

$?x \equiv$ scale 1 $?x$

$\ldots$

input 1
rotate 90 (scale 4 line)

input 2
circle
i.e. scale 1 circle
rotate 90 circle
scale 1 (rotate 90 circle)
rotate 90 (scale 1 circle)
etc.
how does **e-graph anti-unification** work?

**Finding common structure**

- **circle** = rotate 90 circle
  - ?x = scale 1 ?x
  - ...

**Input 1**
- rotate 90 (scale 4 line)

**Input 2**
- circle
  - i.e. scale 1 circle
    - rotate 90 circle
    - scale 1 (rotate 90 circle)
    - rotate 90 (scale 1 circle)
  - etc.

**Criterion 1.** **occurs multiple times**
- scale 4 ?x
- won't work
how does **e-graph anti-unification** work?

**finding common structure**

circle = rotate 90 circle

?x = scale 1 ?x

... 

criterion 1. **occurs multiple times**

scale 4 ?x won't work

---

input 1
rotate 90 (scale 4 line)

input 2
circle

i.e. scale 1 circle
rotate 90 circle
scale 1 (rotate 90 circle)
rotate 90 (scale 1 circle)

etc.
how does **e-graph anti-unification** work?

finding common structure

circle $\equiv$ rotate 90 circle

$?x \equiv$ scale 1 ?x

... equivelances

input 1
rotate 90 (scale 4 line)

input 2
circle

i.e. scale 1 circle
  rotate 90 circle
  scale 1 (rotate 90 circle)
  rotate 90 (scale 1 circle)
  etc.

rotate

scale

90

1

line

4

criterion 1. **occurs multiple times**

scale 4 ?x won't work

criterion 2. **prefer specific abstractions**

rotate 90 ?x works, but we can do better
how does e-graph anti-unification work?

term anti-unification

A top-down approach to finding common structure. (prior work!)
how does e-graph anti-unification work?

**term anti-unification**

**input 1**
rotate 90 (scale 4 line)

**input 2**
rotate 90 (scale 1 circle)

(Starting from the root of both terms)

**option 1** if nodes are same, add to pattern & recurse

**current pattern**

rotate
**how does e-graph anti-unification work?**

**term anti-unification**

(starting from the root of both terms)

**option 1**  
if nodes are same, add to pattern & recurse

---

**input 1**  
rotate 90 (scale 4 line)

- rotate
- 90
- scale
- 4
- line

**input 2**  
rotate 90 (scale 1 circle)

- rotate
- 90
- scale
- 1
- circle

---

**current pattern**  
rotate 90
how does e-graph anti-unification work?

term anti-unification

(input 1) rotate 90 (scale 4 line)

(rotate) 90

(scale) 4 line

( starting from the root of both terms) option 1 if nodes are same, add to pattern & recurse

(input 2) rotate 90 (scale 1 circle)

(rotate) 90

(scale) 1 circle

current pattern

rotate 90 (scale )
how does e-graph anti-unification work?

**term anti-unification**

(Starting from the root of both terms)

**Option 1**  
if nodes are same, add to pattern & recurse

**Option 2**  
if nodes differ, insert hole

Current pattern

rotate 90 (scale ?x )
how does **e-graph anti-unification** work?

**term anti-unification**

(starting from the root of both terms)

**option 1**  if nodes are same, add to pattern & recurse

**option 2**  if nodes differ, insert hole

**current pattern**

rotate 90 (scale ?x ?y)
how does e-graph anti-unification work?

term anti-unification

(Starting from the root of both terms)

**Option 1**: if nodes are same, add to pattern & recurse

**Option 2**: if nodes differ, insert hole

Challenge: how to apply this to e-graphs?
how does **e-graph anti-unification** work?

**Input programs (in e-graph)**
- rotate
- scale
- 4
- line
- rotate
- circle
- scale
- 90
- 1

**(the intuition behind)**
A top-down approach to finding common structure in the presence of e-graphs.
how does **e-graph anti-unification** work?

**step 1** pick two e-classes

![Diagram of e-graph anti-unification process]

- input programs (in e-graph)
- output pattern(s)

- rotate
- scale
- 4
- line
- 90
- rotate
- circle
- scale
- 1
- rotate
- circle
- scale
- 1
- rotate
- circle
- scale
- 1
how does e-graph anti-unification work?

step 1  pick two e-classes

step 2a  if e-classes contain matching e-nodes, for each pair of matching e-nodes,
add to pattern & run step 2 with matching e-nodes' children
how does **e-graph anti-unification** work?

step 1  pick two e-classes

step 2a  if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children
how does **e-graph anti-unification** work?

**step 1**  
pick two e-classes

**step 2a**  
if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children

input programs (in e-graph)

output pattern(s)

rotate 90 (scale 1 )
how does e-graph anti-unification work?

step 1  pick two e-classes
step 2a if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children
step 2b otherwise, insert hole in pattern
how does e-graph anti-unification work?

step 1  pick two e-classes
step 2a  if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children
step 2b  otherwise, insert hole in pattern

rotate 90 (scale ?x ?y)

input programs (in e-graph)
how does e-graph anti-unification work?

step 1  pick two e-classes
step 2a if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children
step 2b otherwise, insert hole in pattern
step 3  do this for all pairs of e-classes in the e-graph
how does e-graph anti-unification work?

step 1  pick two e-classes
step 2a if e-classes contain matching e-nodes, for each pair of matching e-nodes, add to pattern & run step 2 with matching e-nodes' children
step 2b otherwise, insert hole in pattern
step 3  do this for all pairs of e-classes in the e-graph
how does e-graph anti-unification work?
how does babble work?

**user inputs**
- program corpus
  - user inputs
- list of equivalences
  - circle = rotate 90 circle
  - ?x = scale 1 ?x
  - ...

**babble**

**insight**
- use e-graphs
  - to create and store infinite equivalent input variants

**contribution 2**
- e-graph anti-unification
  - to propose candidate abstractions in the presence of input variants

**contribution 3**
- targeted CSE
  - to pick the best programs and abstractions

**babble outputs**
- learned library
- rewritten inputs
  - face circle
  - face line

let face = \( \lambda \text{shape} \rightarrow \)
what's the challenge?

**how does babble work?**

**how well does it work?**

---

**user inputs**

- **program corpus**
- **list of equivalences**
  - `circle = rotate 90 circle`

---

**babble**

**insight**

**use e-graphs**

to create and store infinite equivalent input variants

**contribution 2**

**e-graph anti-unification**

to propose candidate abstractions in the presence of input variants

---

**babble outputs**

**learned library**

- `let face = λshape →` (image of a face)

**rewritten inputs**

- (image of rewritten inputs)
what's the challenge?
how does babble work?
how well does it work?
how well does it work?
how well does it work?
qualitative eval

\[
\text{ngon} = \lambda \text{size sides} \rightarrow \text{T (repeat (T l (M 1 0 -0.5 (0.5 / \tan (\pi / \text{sides})))) \text{sides}} \\
\text{\text{(M 1 ((2 * \pi) / \text{sides}) 0 0))}} \\
\text{(M size 0 0 0)}
\]

\[
\text{con_hex} = \lambda \text{inner_size} \rightarrow C (\text{ngon 4 6}) (\text{ngon inner_size 6})
\]

\[
\text{ring} = \lambda \text{shape} \rightarrow \text{repeat (offset 1.5 shape) n (rotate n)}
\]

nuts & bolts
how well does it work?

qualitative eval
how well does it work?
qualitative eval

check the paper for more examples!
how well does it work?
quantitative eval

does it scale?
does it compress?
how well does it work?
quantitative eval
how well does it work?
quantitative eval

![Graph showing compression ratio vs. time (seconds)]

we want our results to go in this direction!
how well does it work?
quantitative eval

physics domain results

does it scale?
does it compress?
how well does it work?  
quantitative eval

physics domain results
how well does it work?
quantitative eval

physics domain results

does it scale?
does it compress?
how well does it work?
quantitative eval

does it scale?
list domain results

does it compress?
how well does it work?
quantitative eval

list domain results

does it scale?

does it compress?
how well does it work?
quantitative eval

does it scale?
does it compress?
list domain results
the functions make sense
we compress and scale well

what's the challenge?
how does babble work?
how well does it work?
Babble: learning better abstractions with e-graphs and anti-unification

User inputs:
- Program corpus
- List of equivalences

Babble outputs:
- Learned library
- Rewritten inputs

Insight:
- Use e-graphs to create and store infinite equivalent input variants

Contribution 2:
- E-graph anti-unification to propose candidate abstractions in the presence of input variants

Contribution 3:
- Targeted CSE to pick the best programs and abstractions

Example:
- Let face = λshape →...