Synthesizing Backward through the Geometry Pipeline

Zachary Tatlock
Allen School of Computer Science and Engineering
University of Washington
tatlock.net
Amazing Collaborators

Chandakana Nandi
Max Willsey
Adam Anderson
Remy Wang
Oliver Flatt

Eva Darulova
James Wilcox
Pavel Panchekha
Taylor Blau
Dan Grossman
Today’s Menu
Today’s Menu

1. Everything is a Program (Output)

   ![Diagram](image)

   - CAD
   - CSG
   - STL
   - G-code

   [ICFP 2018]
   [PLDI 2020]
Today’s Menu

1. Everything is a Program (Output)

   - CAD → CSG → STL → G-code → 3D Printer

2. Decompiling Surfaces to Expressions

   - CSG ← STL

[ICFP 2018]
Today’s Menu

1. Everything is a Program (Output)
   - CAD → CSG → STL → G-code → [3D Printer]
2. Decompiling Surfaces to Expressions
   - CSG ← STL [ICFP 2018]
3. Rerolling Loops for Editability
   - CAD ← CSG [PLDI 2020]
Today’s Menu

1. Everything is a Program (Output)
   - CAD ➔ CSG ➔ STL ➔ G-code ➔ [3D Printer]

2. Decompiling Surfaces to Expressions
   - CSG ➔ STL [ICFP 2018]

3. Rerolling Loops for Editability
   - CAD ➔ CSG [PLDI 2020]

4. egg : E-Graphs Good

\[\text{Input Term} \rightarrow \text{Initialize} \rightarrow \text{E-graph} \rightarrow \text{Extract} \rightarrow \text{Optimized Term} \rightarrow \text{Apply Rewrites}\]
Everything is a Program (Output)

CAD → CSG → STL → G-code → 3D Printer
Everything is a Program (Output)

CAD ➔ CSG ➔ STL ➔ G-code ➔ Output

Much of our environment and many of our belongings began life as code.
Much of our environment and many of our belongings began life as code.

The following is a cartoon version of that process through the lens of “desktop manufacturing”.
Everything is a Program (Output)

CAD → CSG → STL → G-code → [output image]
Everything is a Program (Output)
Everything is a Program (Output)

CAD → CSG → STL → G-code → 

[Image of hexagonal objects]

[Image of a person]

[Image of Allen keys]

[Image of the IKEA logo]
Everything is a Program (Output)

CAD → CSG → STL → G-code →  

(Difference
  (Cuboid [60, 20, 4])
  (Fold Union
    (Tabulate (i 4)
      (Translate \[i^2 + 10i + 8, 10, 2\]
        (HexPrism \[i + 3, 4\]))))))

(IKEA tools image)
Everything is a Program (Output)

Geometric Primitives
- Cuboid, HexPrism, ...

Geometric Operators
- Difference, Union, Translate, ...

Loops, Control Flow, Combinators
- Fold, Tabulate, Lambdas, ...

(Difference
  (Cuboid [60, 20, 4])
  (Fold Union
    (Tabulate (i 4)
      (Translate [\(i^2 + 10i + 8\), 10, 2]
        (HexPrism [i + 3, 4]))))))
Everything is a Program (Output)

- CAD
- CSG
- STL
- G-code

(Difference
  (Cuboid [60, 20, 4])
  (Fold Union
    (Tabulate (i 4)
      (Translate \([i^2 + 10i + 8, 10, 2] \)
        (HexPrism \([i + 3, 4]\))))))

Geometric Primitives
- Cuboid, HexPrism, ...

Geometric Operators
- Difference, Union, Translate, ...

Loops, Control Flow, Combinators
- Fold, Tabulate, Lambdas, ...

In the real world, this would be built interactively in a GUI.
Everything is a Program (Output)

(CAD) → (CSG) → (STL) → (G-code) → (Output)

(Difference
    (Cuboid [60, 20, 4])
    (Fold Union
        (Tabulate (i 4)
            (Translate [(i^2 + 10i + 8, 10, 2)]
                (HexPrism [(i + 3, 4)]))))
)
Everything is a Program (Output)

(Difference
  (Cuboid [60, 20, 4])
  (Fold Union
    (Tabulate (i 4)
      (Translate [i^2 + 10i + 8, 10, 2]
        (HexPrism [i + 3, 4])))
  ))

(CAD)

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4]))
  ))

(CSG)
Everything is a Program (Output)

<table>
<thead>
<tr>
<th>CAD</th>
<th>CSG</th>
<th>STL</th>
<th>G-code</th>
<th>3D Printer</th>
</tr>
</thead>
</table>

(Difference
  (Cuboid [60, 20, 4])
  (Fold Union
    (Tabulate (i 4)
      (Translate \[ i^2 + 10i + 8, 10, 2 \]
        (HexPrism [i + 3, 4]))))))

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4])))))

Geometric Primitives
- Cuboid, HexPrism, ...

Geometric Operators
- Difference, Union, Translate, ...

Loops, Control Flow, Combinators
- Fold, Tabulate, Lambdas, ...
Primitives are still parameterized, but we’ve lost some *editability*!

What if we want to add more holes?
What if we want to change the spacing?
What if we want to change hex to circle?

Some tweaks *easier* though, e.g., rotate just one hole.

---

**Geometric Primitives**
- Cuboid, HexPrism, ...

**Geometric Operators**
- Difference, Union, Translate, ...

**Loops, Control Flow, Combinators**
- Fold, Tabulate, Lambdas, ...
Everything is a Program (Output)

(CAD) → (CSG) → (STL) → (G-code) → (3D Printer)

(CSG)

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
     (HexPrism [3, 4]))
    (Translate [18, 10, 2]
     (HexPrism [4, 4]))
    (Translate [32, 10, 2]
     (HexPrism [5, 4]))
    (Translate [64, 10, 2]
     (HexPrism [6, 4])))
)

CSG

Output

Everything is a Program (Output)
(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
     (HexPrism [3, 4]))
    (Translate [18, 10, 2]
     (HexPrism [4, 4]))
    (Translate [32, 10, 2]
     (HexPrism [5, 4]))
    (Translate [64, 10, 2]
     (HexPrism [6, 4]))))

solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
  endfacet
  facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 0
  endloop
  endfacet
  facet normal 0 -1 0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
  endfacet
  facet normal 0 -1 -0
  outer loop
    ... (870 LOC)
Everything is a Program (Output)

Geometric Primitives
- Cuboid, HexPrism, ...

Geometric Operators
- Difference, Union, Translate, ...

Oriented Triangles
- Absolute position for all surfaces

(CAD) → (CSG) → (STL) → (G-code) → (3D Printer)

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4]))))

solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
  endfacet
  facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 0
  endloop
  endfacet
  facet normal 0 -1 0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
  endfacet
  facet normal 0 -1 -0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
  endfacet
... (870 LOC)
Everything is a Program (Output)

CAD ➔ CSG ➔ STL ➔ G-code ➔ Print

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4])))

solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
  endfacet
  facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
    vertex 6.5 7.40192 0
  endloop
  endfacet
  facet normal 0 -1 0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
  endfacet
  facet normal 0 -1 -0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
  endfacet
  ... (870 LOC)

Oriented Triangles
  - Absolute position for all surfaces

No parameterization, even less editability!

Essentially not human-readable.

Some new tweaks possible now though, e.g., freeform, organic distortions.
(Difference
(Cuboid [60, 20, 4])
(Union
(Translate [8, 10, 2]
(HexPrism [3, 4])
(Translate [18, 10, 2]
(HexPrism [4, 4])
(Translate [32, 10, 2]
(HexPrism [5, 4])
(Translate [64, 10, 2]
(HexPrism [6, 4])))

CSG solid facet normal 0 1 -0
outer loop vertex 9.5 7.40192 0
vertex 6.5 7.40192 4
vertex 9.5 7.40192 4
endloop
endfacet
facet normal 0 1 0
outer loop vertex 6.5 7.40192 4
vertex 9.5 7.40192 0
vertex 6.5 7.40192 0
endloop
endfacet
facet normal 0 -1 0
outer loop vertex 6.5 12.5981 0
vertex 9.5 12.5981 4
vertex 6.5 12.5981 4
endloop
endfacet
... (870 LOC)

Oriented Triangles
- Absolute position for all surfaces

Some new tweaks possible now though, e.g., freeform, organic distortions.

Topography Optimization
toms3d.org
Everything is a Program (Output)

CAD → CSG → STL → G-code → Output

(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4]))
  ))

solid
facet normal 0 1 -0
outer loop
  vertex 9.5 7.40192 0
  vertex 6.5 7.40192 4
  vertex 9.5 7.40192 4
endloop
endfacet
facet normal 0 1 0
outer loop
  vertex 6.5 7.40192 4
  vertex 9.5 7.40192 0
  vertex 6.5 7.40192 0
endloop
endfacet
facet normal 0 -1 0
outer loop
  vertex 6.5 7.40192 4
  vertex 9.5 7.40192 0
  vertex 6.5 7.40192 0
endloop
endfacet
facet normal 0 -1 -0
outer loop
  vertex 6.5 12.5981 0
  vertex 9.5 12.5981 4
  vertex 6.5 12.5981 4
endloop
endfacet
facet normal 0 0 -1
outer loop
  vertex 6.5 7.40192 0
  vertex 9.5 7.40192 4
  vertex 6.5 7.40192 4
endloop
endfacet
... (870 LOC)

Oriented Triangles
- Absolute position for all surfaces

No parameterization, even less *editability*!
Essentially not human-readable.
Some new tweaks possible now though, e.g., freeform, organic distortions.
Everything is a Program (Output)

solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
endfacet

facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 0
  endloop
endfacet

facet normal 0 -1 0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
endfacet

facet normal 0 -1 -0
  outer loop
  ... (870 LOC)
Everything is a Program (Output)

solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
endfacet
facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 0
  endloop
endfacet
facet normal 0 -1 0
  outer loop
    vertex 6.5 12.5981 0
    vertex 9.5 12.5981 4
    vertex 6.5 12.5981 4
  endloop
endfacet
facet normal 0 -1 -0
  outer loop
    vertex 6.5 12.5981 4
    vertex 9.5 12.5981 0
    vertex 6.5 12.5981 4
  endloop
endfacet

G-code

; tool H0.234 W0.400
G1 Z0.234 F1000
G1 X60.465 Y101.772 F6000
G1 E0.0000 F2400
G92 E0.0000
G1 X60.465 Y77.961 E0.8803 F750
G1 X63.160 Y75.267 E1.0211
G1 X126.971 Y75.267 E3.3801
G1 X129.665 Y77.961 E3.5210
G1 X129.665 Y101.772 E4.4013
G1 X126.971 Y104.467 E6.9011
G1 X60.465 Y101.772 E7.0420
G92 E0.0000
G1 E-1.5000 F2400
G1 X129.265 Y101.607 F6000
G1 E0.0000 F2400
G92 E0.0000
G1 X126.805 Y104.067 E0.1286 F750
G1 X63.326 Y104.067 E2.4754
G1 X60.865 Y101.607 E2.6040
G1 X60.865 Y78.127 E3.4720
G1 X63.326 Y75.667 E3.6006
... (10,000 LOC)
Process-specific Machine Code
- Flat, imperative commands
- Move, enable, disable tools
- Format semi-standard
- Semantics machine-dependent
- Interpreted by embedded sys
- Basically “Turtle Graphics”
Everything is a Program (Output)

- CAD 
- CSG 
- STL 
- G-code

Forget editability. (except maybe professional machinists)

No control structure, error handling, etc.

No portability.

Process-specific Machine Code
- Flat, imperative commands
- Move, enable, disable tools
- Format semi-standard
- Semantics machine-dependent
- Interpreted by embedded sys
- Basically “Turtle Graphics”

G-code

... (870 LOC)

... (10,000 LOC)
Everything is a Program (Output)

- CAD → CSG → STL → G-code → 3D Printing

Rest of pipeline also very interesting!
- Interpret G-code into motor commands
- Fast control on wimpy embedded CPUs
- Often open loop, so no feedback
- Many potential optimizations
Everything is a Program (Output)

Rest of pipeline also very interesting!

- Interpret G-code into motor commands
- Fast control on wimpy embedded CPUs
- Often *open loop*, so no feedback
- Many potential optimizations
Everything is a Program (Output)

Like all programming, fabrication is an iterative process.
Everything is a Program (Output)

- CAD → CSG → STL → G-code → Fabrication

So, why should PL researchers care?
- Fabrication is critical infrastructure.
- Relevant languages both easy and hard:
  - Easy: short, bounded, “mathy”
  - Hard: geometry, approximation, ill-posed
- Enable people to program their environment!
Today's Menu

1. Everything is a Program (Output)

2. Decompiling Surfaces to Expressions

3. Rerolling Loops for Editability

4. egg : E-Graphs Good
Why Decompile Surfaces to Expressions?
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.
Why Decompile Surfaces to Expressions?

Design is difficult.

Folks just want to download and print stuff.

- Easy
- Incomplete
- Poor Editability
Decompiling to Tweak Wrench Holder
Decompiling to Tweak Wrench Holder
Decompiling to Tweak Wrench Holder
Decompiling to Tweak Wrench Holder

Bent wrench not seating square :(
Decompiling to Tweak Wrench Holder

Want to rotate 5th hole.
Pretty difficult on just the triangles...

Bent wrench not seating square :(
Decompiling to Tweak Wrench Holder

solid
  facet normal 0 1 -0
  outer loop
   vertex 9.5 7.40192 0
   vertex 6.5 7.40192 4
   vertex 9.5 7.40192 4
  endloop
endfacet
facet normal 0 1 0
  outer loop
   vertex 6.5 7.40192 4
   vertex 9.5 7.40192 0
   vertex 6.5 7.40192 0
  endloop
endfacet
...
Decompiling to Tweak Wrench Holder

```
solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
endfacet
facet normal 0 1 0
outer loop
  vertex 6.5 7.40192 4
  vertex 9.5 7.40192 0
  vertex 6.5 7.40192 0
endloop
endfacet
...
```

```
(Difference
  (Cube 60 20 4)
  (Union
    (Translate 8 10 2
      (HexPrism 3 4))
    (Translate 18 10 2
      (HexPrism 4 4))
    (Translate 32 10 2
      (HexPrism 5 4))
    (Translate 64 10 2
      (HexPrism 6 4))
    (Translate 88 10 2
      (HexPrism 6 4))
    (Translate 110 10 2
      (HexPrism 6 4))
    ...
  )
)
```

Manual Reverse Engineering

**STL**

**CSG**
Decompiling to Tweak Wrench Holder

Manual Reverse Engineering

**STL**

```
solid
  facet normal 0 1 -0
  outer loop
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 4
  endloop
endfacet
facet normal 0 1 0
  outer loop
    vertex 6.5 7.40192 4
    vertex 9.5 7.40192 0
    vertex 6.5 7.40192 0
  endloop
endfacet
...```

**CSG**

```
(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4])))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4]))
    (RotateZ [35]
      (Translate [88, 10, 2]
        (HexPrism [6, 4])))
    (Translate [110, 10, 2]
      (HexPrism [6, 4]))
  ))```

**CAD**
Decompiling to Tweak Wrench Holder

Manual Reverse Engineering

 STL

(CSG
(Difference
  (Cuboid [60, 20, 4])
  (Union
    (Translate [8, 10, 2]
      (HexPrism [3, 4]))
    (Translate [18, 10, 2]
      (HexPrism [4, 4]))
    (Translate [32, 10, 2]
      (HexPrism [5, 4]))
    (Translate [64, 10, 2]
      (HexPrism [6, 4]))
    (RotateZ [35]
      (Translate [88, 10, 2]
        (HexPrism [6, 4]))
      (Translate [110, 10, 2]
        (HexPrism [6, 4]))
      ... ))
)

Easy edit fixed the issue!
Decompiling to Tweak Wrench Holder

Goal: Decompile STL to more-editable CSG

_manual reverse engineering_

**Cuboid [60, 20, 4]**

**Union**

**HexPrism [3, 4]**

**Translate [8, 10, 2]**

**HexPrism [4, 4]**

**Translate [18, 10, 2]**

**HexPrism [5, 4]**

**Translate [32, 10, 2]**

**HexPrism [6, 4]**

**RotateZ [35]**

**Translate [88, 10, 2]**

**HexPrism [6, 4]**

**Translate [110, 10, 2]**

**HexPrism [6, 4]**

...
To *Decompile*, First Understand *Compile*
To *Decompile*, First Understand *Compile*

\[
C ::= \text{Mesh } M \\
| \text{Empty} \\
| \text{Cube} \\
| \text{Cylinder } \mathbb{N} \\
| \ldots \\
| \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 C \\
| \text{Binop } \text{op } C C
\]

\[
M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^* \\
\text{op} ::= \text{Union} \\
| \text{Inter} \\
| \text{Diff}
\]
To *Decompile*, First Understand *Compile*

From CSG perspective, triangle meshes (~ STLs) are values.

Compilation should reduce an expression to a Mesh.

\[
C ::= \text{Mesh } M \\
| \quad \text{Empty} \\
| \quad \text{Cube} \\
| \quad \text{Cylinder } N \\
| \quad \ldots \\
| \quad \text{Affine } R^{3x3} P \\
| \quad \text{Binop op } C
\]

\[
M ::= (R^3, R^3, R^3)^* \\
\]

\[
op ::= \text{Union} \\
| \quad \text{Inter} \\
| \quad \text{Diff}
\]
To Decompile, First Understand Compile

\[
C ::= \text{Mesh } M \\
| \text{Empty} \\
| \text{Cube} \\
| \text{Cylinder } \mathbb{N} \\
| \ldots \\
| \text{Affine } \mathbb{R}^{3x3} \mathbb{R}^3 C \\
| \text{Binop op } C C
\]

\[
M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^* \\
op ::= \text{Union} \\
| \text{Inter} \\
| \text{Diff}
\]
To *Decompile*, First Understand *Compile*

\[
C ::= \text{Mesh } M \\
\quad \text{Empty} \\
\quad \text{Cube} \\
\quad \text{Cylinder } \mathbb{N} \\
\quad \cdots \\
\quad \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 C \\
\quad \text{Binop } \text{op } C C
\]

\[
M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^*
\]

\[
op ::= \text{Union} \\
\quad \text{Inter} \\
\quad \text{Diff}
\]
To **Decompile**, First Understand **Compile**

\[
C ::= \text{Mesh } M \\
| \text{Empty} \\
| \text{Cube} \\
| \text{Cylinder } \mathbb{N} \\
| \ldots \\
| \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 C \\
| \text{Binop op } C C
\]

- Various geometric primitives
- Rotation, translation, scale, ...
- Set theoretic operators

\[
M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^* \\
op ::= \text{Union} \\
| \text{Inter} \\
| \text{Diff}
\]
To **Decompile**, First Understand **Compile**

\[ C ::= \text{Mesh } M \]
- Empty
- Cube
- Cylinder \( \mathbb{N} \)
- \ldots
- Affine \( \mathbb{R}^{3 \times 3} \mathbb{R}^3 C \)
- Binop \( \text{op } C C \)

\[ M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^* \]

\[ \text{op} ::= \text{Union} \]
- Inter
- Diff
To Decompile, First Understand Compile

\[
C ::= \text{Mesh } M
\]

\[
\begin{align*}
\text{Empty} & \quad \boxed{\text{Empty}} = \{\} \\
\text{Cube} & \quad \boxed{\text{Cube}} = (0, 1)^3 \\
\text{Cylinder } \mathbb{N} & \quad \boxed{\text{Affine } p \; q \; c} = \{pv + q \mid v \in \boxed{c}\} \\
\ldots & \\
\text{Affine } \mathbb{R}^{3 \times 3} \; \mathbb{R}^3 \; C & \quad \boxed{\text{Binop } o \; c_1 \; c_2} = \boxed{c_1} \boxed{o} \boxed{c_2} \\
\text{Binop } \text{op } C \; C & \quad \boxed{\text{Union}} = \bigcup \boxed{\text{Inter}} = \bigcap \boxed{\text{Diff}} = \setminus
\end{align*}
\]

\[
M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)^* \\
op ::= \text{Union} \\
\text{Inter} \\
\text{Diff}
\]

\[
\boxed{\text{Mesh } m} = \boxed{m}
\]
To **Decompile**, First Understand **Compile**

Can also denote a CSG program directly to a *regular open* set of points.

Used in compiler correctness proof. (See ICFP 2018 paper for more!)

\[
\begin{align*}
\llbracket \text{Empty} \rrbracket &= \{\} \\
\llbracket \text{Cube} \rrbracket &= (0, 1)^3 \\
\llbracket \text{Affine } p \ q \ c \rrbracket &= \{pv + q \mid v \in \llbracket c \rrbracket\} \\
\llbracket \text{Binop } o \ c_1 \ c_2 \rrbracket &= \llbracket c_1 \rrbracket \llbracket o \rrbracket \llbracket c_2 \rrbracket \\
\llbracket \text{Union} \rrbracket &= \cup \\
\llbracket \text{Inter} \rrbracket &= \cap \\
\llbracket \text{Diff} \rrbracket &= \setminus \\
\llbracket \text{Mesh } m \rrbracket &= \llbracket m \rrbracket
\end{align*}
\]

\[M ::= (\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)\]

\[op ::= \text{Union} \mid \text{Inter} \mid \text{Diff}\]
To Decompile, First Understand Compile

Can also denote a CSG program directly to a regular open set of points.

Used in compiler correctness proof. (See ICFP 2018 paper for more!)

Regular opens critical for getting set theoretic operators right.

Careful with Union, avoid “holes”.

\[
\begin{align*}
[\text{Empty}] & = \{\} \\
[\text{Cube}] & = (0, 1)^3 \\
[\text{Affine} \ p \ q \ c] & = \{p\nu + q \mid \nu \in [c]\} \\
[\text{Binop} \ o \ c_1 \ c_2] & = [c_1][o][c_2] \\
[\text{Union}] & = \cup \\
[\text{Inter}] & = \cap \\
[\text{Diff}] & = \setminus \\
[\text{Mesh} \ m] & = [m]
\end{align*}
\]
To Decompile, First Understand Compile

Can also denote a CSG program directly to a regular open set of points.

Used in compiler correctness proof. (See ICFP 2018 paper for more!)

Regular opens critical for getting set theoretic operators right.

Careful with Union, avoid “holes”.

Mesh denotation given by ray casting.

InsideVia(m, pt, d) : Mesh × point × Direction → bool
InsideVia(m, pt, d) = let h = (pt, d) in
{ f | f ∈ m ∧ intersect(f, h) = Other } = ∅ ∧ |
{ f | f ∈ m ∧ intersect(f, h) = InteriorPt }| mod 2 = 1

\[ \begin{align*}
\text{[Empty]} & = \{\} \\
\text{[Cube]} & = \,(0,1)^3 \\
\text{[Affine } p\, q\, c\text{]} & = \{p\nu + q | \nu \in [c]\} \\
\text{[Binop } o\, c_1\, c_2\text{]} & = \,[c_1]\,[o]\,[c_2]\ \\
\text{[Union]} & = \bigcup \quad \text{[Inter]} = \bigcap \quad \text{[Diff]} = \setminus \\
\text{[Mesh } m\text{]} & = \,[m]\ \\
\end{align*} \]
To **Decompile**, First Understand **Compile**

Can also denote a CSG program directly to a *regular open* set of points.

Used in compiler correctness proof. (See ICFP 2018 paper for more!)

Denotational semantics too “big step-ish” to suggest effective synthesis procedure.

So, let’s build an equivalent **small step operational semantics**.

\[
\begin{align*}
\llbracket \text{Empty} \rrbracket &= \emptyset \\
\llbracket \text{Cube} \rrbracket &= (0,1)^3
\end{align*}
\]

\[
\begin{align*}
\text{InsideVia}(m, pt, d) &:\quad \text{Mesh} \times \text{point} \times \text{Direction} \to \text{bool} \\
\text{InsideVia}(m, pt, d) &= \text{let } h = (pt, d) \text{ in} \\
&\quad \{ f \mid f \in m \land \text{intersect}(f, h) = \text{Other} \} = \emptyset \\
&\quad \land |\{ f \mid f \in m \land \text{intersect}(f, h) = \text{InteriorPt} \}| \mod 2 = 1
\end{align*}
\]

\[
\begin{align*}
\llbracket \cdot \rrbracket &: \quad \text{Mesh} \to \mathcal{P}(\text{point}) \\
\llbracket m \rrbracket &= \{ pt \mid \exists d. \text{InsideVia}(m, pt, d) \}
\end{align*}
\]
To *Decompile*, First Understand *Compile*
To *Decompile*, First Understand *Compile*

\[
E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop op } E \, C \mid \text{Binop op } (\text{Mesh } m) \, E
\]
To **Decompile**, First Understand **Compile**

\[ E ::= [\cdot] \mid Affine \mathbb{R}^{3\times3} \mathbb{R}^3 E \mid Binop \ op \ E \ C \mid Binop \ op \ (Mesh \ m) \ E \]

Evaluation contexts focus semantics on “where the action is” (redexes).
To Decompile, First Understand Compile

\[
E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop op } E \ C \mid \text{Binop op (Mesh m) E}
\]

\[
\frac{c \to_p c'}{E[c] \to_c E[c']}
\]

\[
\text{Cube} \to_p \text{Mesh m}_{\text{cube}}
\]

\[
\text{map}_{\text{vertex}} \ (\lambda v. pv + q) \ m = m'
\]

\[
\text{Affine } p \ q \ (\text{Mesh } m) \to_p \text{Mesh } m'
\]

\[
m\text{Bop}(o)(m_1, m_2) = m'
\]

\[
\text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \to_p \text{Mesh } m'
\]

Evaluation contexts focus semantics on “where the action is” (redexes).
To *Decompile*, First Understand *Compile*

\[ E ::= [\cdot] \mid Affine \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid Binop \text{ op } E \text{ C} \mid Binop \text{ op } (\text{Mesh } m) E \]

\[ c \rightarrow_p c' \]
\[ E[c] \rightarrow_c E[c'] \]

\[ Cube \rightarrow_p \text{Mesh } m_{cube} \]

Evaluation contexts focus semantics on “where the action is” (redexes).

A compilation step is just a primitive step under an evaluation context.

\[ \text{map}_{\text{vertex}} (\lambda u. pu + q) m = m' \]

\[ \text{Affine } p \text{ q } (\text{Mesh } m) \rightarrow_p \text{Mesh } m' \]

\[ mBop(o)(m_1, m_2) = m' \]

\[ Binop \text{ o } (\text{Mesh } m_1) (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m' \]
To *Decompile*, First Understand *Compile*

\[
E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop } \text{op } E \text{ C} \mid \text{Binop } \text{op } (\text{Mesh } m) E
\]

\[
c \rightarrow_p c' \quad \frac{E[c] \rightarrow_c E[c']}{\text{Cube} \rightarrow_p \text{Mesh } m_{\text{cube}}}
\]

Evaluation contexts focus semantics on “where the action is” (redexes).

A compilation step is just a primitive step under an evaluation context.

Can directly give triangle mesh of each primitive.

\[
\text{map}_{\text{vertex}}(\lambda v. pv + q) m = m' \quad \frac{\text{Affine } p q (\text{Mesh } m) \rightarrow_p \text{Mesh } m'}{m\text{Bop}(o)(m_1, m_2) = m'}
\]

\[
\text{Binop } o (\text{Mesh } m_1) (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'
\]
To *Decompile*, First Understand *Compile*

\[
E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^{3} E \mid \text{Binop } \text{op } E \ C \mid \text{Binop } \text{op } (\text{Mesh } m) \ E
\]

\[
c \rightarrow_{p} c' \\
\overline{E[c] \rightarrow_{c} E[c']}
\]

\[
\text{Cube} \rightarrow_{p} \text{Mesh } m_{\text{cube}}
\]

\[
\text{map}_{\text{vertex}} (\lambda v. \ p v + q) \ m = m' \\
\overline{\text{Affine } p \ q \ (\text{Mesh } m) \rightarrow_{p} \text{Mesh } m'}
\]

\[
mBop(o)(m_1, m_2) = m' \\
\overline{\text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \rightarrow_{p} \text{Mesh } m'}
\]

Evaluation contexts focus semantics on “where the action is” (redexes).

A compilation step is just a primitive step under an evaluation context.

Can directly give triangle mesh of each primitive.

Just call into mesh library for affine and set ops.
To *Decompile*, First Understand *Compile*

$$E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop op } E \ C \mid \text{Binop op } (\text{Mesh } m) \ E$$

$$\begin{align*}
    c \to_p c' & \quad \Rightarrow \quad E[c] \to_c E[c'] \\
    \text{Cube} \to_p \text{Mesh } m_{\text{cube}}
\end{align*}$$

Evaluation contexts focus semantics on "where the action is" (redexes).

A compilation step is just a primitive step under an evaluation context.

Can directly give triangle mesh of each primitive.

Just call into mesh library for affine and set ops.

Simple. Deterministic.
Compile Example

union
cube
trans (2, 0, 0) cube
Compile Example

union
cube
trans (2, 0, 0) cube → $C$

union
[
trans (2, 0, 0) cube
cube
]
Compile Example

union
cube
trans (2, 0, 0) cube

→

union
[
trans (2, 0, 0) cube
]
c

→

union
[
trans (2, 0, 0) cube
]
c

→

union
[
Mesh [m_{cube}]
]
Compile Example

union cube
trans (2, 0, 0) cube → C [union []
trans (2, 0, 0) cube]
c [union []
cube] → C [union []
trans (2, 0, 0) cube]
Mesh [m_{cube}] → C
* 
union [Mesh [m_{cube}] []
trans (2, 0, 0) cube] → C [union []
cube] → C [union []
trans (2, 0, 0) cube] → C [union []
cube] → C [union []
trans (2, 0, 0) cube]
Compile Example

union
cube
trans (2, 0, 0) cube

union
cube
trans (2, 0, 0) cube

union
Mesh [m\(c_{\text{cube}}\)]

union
Mesh [m\(c_{\text{trans (2, 0, 0) cube}}\)]

union
trans (2, 0, 0) cube

Mesh [m\(c_{\text{cube}}\)]
Compile Example

union cube
trans (2, 0, 0) cube

union [.
trans (2, 0, 0) cube
cube]

union [.
trans (2, 0, 0) cube
Mesh [m\textsubscript{cube}]]

union [.
Mesh [m\textsubscript{cube}]
Mesh [m\textsubscript{trans (2, 0, 0) cube}]]

union [.
Mesh [m\textsubscript{trans (2, 0, 0) cube}]
Mesh [m\textsubscript{trans (2, 0, 0) cube}]
trans (2, 0, 0) cube]
Compile Example

union cube
trans (2, 0, 0) cube → $C$ [union [.]
trans (2, 0, 0) cube $C$ [Mesh $[m_{cube}]$]

union [.]
Mesh $[m_{cube}]$ $C$ union [.]
Mesh $[m_{trans (2, 0, 0) cube}]$

union [.]
Mesh $[m_{trans (2, 0, 0) cube}]$ $C$ union [.]
Mesh $[m_{trans (2, 0, 0) cube}]$ $C$ union [.]
trans (2, 0, 0) cube
Compile Example

Locate next redex, always unique.

Compile to mesh.

Bubble meshes “up the AST”.
OK, but what about *Decomposition*?

\[ E ::= [·] \mid \textit{Affine} \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \textit{Binop} \text{ op } E \text{ C} \mid \textit{Binop} \text{ op } (\textit{Mesh} \text{ m}) E \]

\[
\frac{c \rightarrow_p c'}{E[c] \rightarrow_c E[c']} \\
\frac{\text{Cube} \rightarrow_p \text{Mesh } m_{\text{cube}}}{\text{map}_{\text{vertex}}(\lambda \nu. \ p \nu + q) \ m = m'}
\]

\[
\frac{\text{Affine } p \ q \ (\text{Mesh } m) \rightarrow_p \text{Mesh } m'}{\text{mBop}(o)(m_1, \ m_2) = m'}
\]

\[
\frac{\text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'}{} 
\]
OK, but what about Decomposition?

\[ E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop op } E \ C \mid \text{Binop op } (\text{Mesh } m) \ E \]

\[
\begin{align*}
    c \rightarrow_p c' \\
    \frac{E[c] \rightarrow_c E[c']}{\text{Cube } \rightarrow_p \text{Mesh } m_{\text{cube}}} \\
    \text{map}_{\text{vertex}} (\lambda v. pv + q) \ m = m' \\
    \frac{\text{Affine } p \ q \ (\text{Mesh } m) \rightarrow_p \text{Mesh } m'}{m_{\text{Bop}}(o)(m_1, m_2) = m'} \\
    \frac{\text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'}
\end{align*}
\]

Just “flip the arrows”!
Compile Example → Synthesis Strategy

union cube
trans (2, 0, 0) cube
→ C
union [ . ]
trans (2, 0, 0) cube
→ C
union [ . ]
trans (2, 0, 0) cube
→ C
union [ . ]
trans (2, 0, 0) cube
→ C
Mesh [ m_{cube} ]

[ . ] union
Mesh [ m_{cube} ]
Mesh [ m_{trans (2, 0, 0) cube} ]
→ C
union Mesh [ m_{cube} ]
[ . ]
→ C
union Mesh [ m_{cube} ]
[ . ]
→ C
union Mesh [ m_{cube} ]
[ . ]
→ C
Mesh [ m_{trans (2, 0, 0) cube} ]
Mesh [ m_{trans (2, 0, 0) cube} ]
Mesh [ m_{trans (2, 0, 0) cube} ]
Mesh [ m_{trans (2, 0, 0) cube} ]
trans (2, 0, 0) cube
Mesh [...]

CAD ← CSG ← STL
Compile Example → Synthesis Strategy
Guess some way to decompose mesh.
Decompile to some CSG operator.
Push meshes “down the AST”.

Mesh [...]
Decompile: Just Flip the Arrows!

\[ E ::= [\cdot] | \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E | \text{Binop op } E \ C | \text{Binop op } (\text{Mesh m}) \ E \]

\[
\begin{align*}
& c \rightarrow_p c' \\
\hline
& E[c] \rightarrow_c E[c']
\end{align*}
\]

\[
\begin{align*}
& \text{Cube} \rightarrow_p \text{Mesh } m_{\text{cube}}
\end{align*}
\]

\[
\begin{align*}
& \text{map}_{\text{vertex}} (\lambda v. pv + q) \ m = m' \\
& \text{Affine } p \ q \ (\text{Mesh } m) \rightarrow_p \text{Mesh } m'
\end{align*}
\]

\[
\begin{align*}
& m_{\text{Bop}(o)}(m_1, m_2) = m' \\
& \text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'
\end{align*}
\]
Decompile: Just Flip the Arrows!

\[ E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3\times3} \mathbb{R}^3 E \mid \text{Binop } \text{op } E \text{ C } \mid \text{Binop } \text{op } (\text{Mesh } m) E \]

\[
\frac{c \rightarrow p \ c'}{E[c] \rightarrow_c E[c']}
\]

\[
\frac{m \rightarrow_\Omega c}{E[\text{Mesh } m] \rightarrow_s E[c]}
\]

\[
\frac{p \in \Omega_{\text{prim}}(m)}{\text{Mesh } m \rightarrow_\Omega p}
\]

\[
\frac{\text{map}_{\text{vertex}} (\lambda v. \ p v + q) \ m = m'}{\text{Affine } p \ q \ (\text{Mesh } m) \rightarrow_p \text{Mesh } m'}
\]

\[
\frac{\text{mbop}(o)(m_1, m_2) = m'}{\text{Binop } o \ (\text{Mesh } m_1) \ (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'}
\]

\[
\frac{(m_1, m_2) \in \Omega_{\text{add}}(m)}{\text{Mesh } m \rightarrow_\Omega \text{Binop Union } (\text{Mesh } m_1) \ (\text{Mesh } m_2)}
\]

\[
\frac{(m_1, m_2) \in \Omega_{\text{sub}}(m)}{\text{Mesh } m \rightarrow_\Omega \text{Binop Diff } (\text{Mesh } m_1) \ (\text{Mesh } m_2)}
\]
Decompile: Just Flip the Arrows!

\[ E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3\times3} \mathbb{R}^3 E \mid \text{Binop op } E C \mid \text{Binop op } (\text{Mesh } m) E \]

Focus on some promising part of the program not yet decompiled.

\[ E[c] \quad \frac{m \rightarrow_{\Omega} c}{E[\text{Mesh } m] \rightarrow_{s} E[c]} \]

\[ p \in \Omega_{\text{prim}}(m) \quad \frac{\text{Mesh } m \rightarrow_{\Omega} p}{\text{Mesh } m \rightarrow_{\Omega} p} \]

\[ \text{map}_{\text{vertex}} (\lambda v. pv + q) m = m' \quad \frac{\text{Affine } p q (\text{Mesh } m) \rightarrow_{\rho} \text{Mesh } m'}{\text{Mesh } m \rightarrow_{\Omega} \text{Binop Union } (\text{Mesh } m_1) (\text{Mesh } m_2)} \]

\[ m_{\text{Bop}}(o)(m_1, m_2) = m' \quad \frac{\text{Binop } o (\text{Mesh } m_1) (\text{Mesh } m_2) \rightarrow_{\rho} \text{Mesh } m'}{\text{Mesh } m \rightarrow_{\Omega} \text{Binop Diff } (\text{Mesh } m_1) (\text{Mesh } m_2)} \]
Decompile: Just Flip the Arrows!

\[ E ::= [\cdot] \mid Affine \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid Binop op E C \mid Binop op (Mesh m) E \]

Focus on some promising part of the program not yet decompiled.

\[ E[c], E[Mesh m] \rightarrow_s E[c] \]

Use computational geometry to recognize primitives, e.g., RANSAC.

\[ p \in \Omega_{prim}(m) \quad Mesh m \rightarrow_\Omega p \]

mapvertex (\(\lambda v. pv + q\)) \(m = m'\)

\[ Affine p q (Mesh m) \rightarrow_p Mesh m' \]

mBop(o)(m₁, m₂) = m'

\[ Binop o (Mesh m₁) (Mesh m₂) \rightarrow_p Mesh m' \]

(m₁, m₂) \(\in\) \(\Omega_{add}(m)\)

\[ Mesh m \rightarrow_\Omega Binop Union (Mesh m₁) (Mesh m₂) \]

(m₁, m₂) \(\in\) \(\Omega_{sub}(m)\)

\[ Mesh m \rightarrow_\Omega Binop Diff (Mesh m₁) (Mesh m₂) \]
Decompile: Just Flip the Arrows!

$E ::= [\cdot] \mid \text{Affine } \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid \text{Binop op } E C \mid \text{Binop op } (\text{Mesh } m) E$

Focus on some promising part of the program not yet decompiled.

$E[c] \rightarrow_{\omega} c$

$m \rightarrow_{\omega} c$

$E[\text{Mesh } m] \rightarrow_s E[c]$

Use computational geometry to recognize primitives, e.g., RANSAC.

$\text{Cube } \rightarrow_p p$

$p \in \Omega_{\text{prim}}(m)$

$\text{Mesh } m \rightarrow_{\omega} p$

Use heuristics to split and bound in additive and subtractive strategies.

$m \text{Bop}(o)(m_1, m_2) = m'$

$\text{Binop o } (\text{Mesh } m_1) (\text{Mesh } m_2) \rightarrow_p \text{Mesh } m'$

$(m_1, m_2) \in \Omega_{\text{add}}(m)$

$\text{Mesh } m \rightarrow_{\omega} \text{Binop Union } (\text{Mesh } m_1) (\text{Mesh } m_2)$

$(m_1, m_2) \in \Omega_{\text{sub}}(m)$

$\text{Mesh } m \rightarrow_{\omega} \text{Binop Diff } (\text{Mesh } m_1) (\text{Mesh } m_2)$
Decompile: Just Flip the Arrows!

\[ E ::= \{\} \mid Affine \mathbb{R}^{3 \times 3} \mathbb{R}^3 E \mid Binop \ op \ E \ C \mid Binop \ op \ (Mesh \ m) \ E \]

Focus on some promising part of the program not yet decompiled.

Use computational geometry to recognize primitives, e.g., RANSAC.

Use heuristics to split and bound in additive and subtractive strategies.

Parameterized by geometry oracles.

\[ p \in \Omega_{\text{prim}}(m) \]
\[ Mesh \ m \rightarrow \Omega \ p \]

\[ (m_1, m_2) \in \Omega_{\text{add}}(m) \]
\[ Mesh \ m \rightarrow \Omega \ \text{Binop Union} \ (Mesh \ m_1) \ (Mesh \ m_2) \]

\[ (m_1, m_2) \in \Omega_{\text{sub}}(m) \]
\[ Mesh \ m \rightarrow \Omega \ \text{Binop Diff} \ (Mesh \ m_1) \ (Mesh \ m_2) \]
Focus on some promising part of the program not yet decompiled.

Use computational geometry to recognize primitives, e.g., RANSAC.

Use heuristics to split and bound in additive and subtractive strategies.

Challenges:
- Loss of Information
- Geometric Heuristics
- Non-determinism

Parameterized by geometry oracles.
Decomposition Challenges

- **Loss of Information**
  - + evaluation context IS geometric context!

- **Geometric Heuristics**
  - + mesh suggests profitable decompositions

- **Non-determinism**
  - + priority queue beam search, embarrassingly parallel
Decompilation Challenges

Loss of Information
+ evaluation context IS geometric context!

Geometric Heuristics
+ mesh suggests profitable decompositions

Non-determinism
+ priority queue beam search, embarrassingly parallel

See ICFP 2018 for more.
Decompilation Results from Reincarnate

Hexagonal Candle Holder
Ultimate 22 Hex-Wrench Holder
40mm Cube Test Object
25mm Calibration with Empty Top
Measuring Cylinder
Basic Box with Lid
Modular Memory Holder (USB)
Circle Cell Block Generator
Jewelry Box with Inlay
SD Card Rack
Gordian knot 3D Puzzle
Today’s Menu

1. Everything is a Program (Output)

   ![Diagram](image1)

2. Decompiling Surfaces to Expressions

   ![Diagram](image2)

3. Rerolling Loops for Editability

   ![Diagram](image3)

4. egg : E-Graphs Good

   ![Diagram](image4)
Some Editability Still Lacking
Some Editability Still Lacking

We’ve recovered an expression from a flat set of triangles!

Primitives now parameterized, but we’ve lost some editability!

What if we want to add more holes?
What if we want to change the spacing?
What if we want to change hex to circle?
Some Editability Still Lacking

We’ve recovered an expression from a flat set of triangles!

Primitives now parameterized, but we’ve lost some editability!

We’re gonna need a wider aspect ratio for what comes next …

(CSG)

(Translate [18, 10, 2]
 (HexPrism [4, 4]))
(Translate [32, 10, 2]
 (HexPrism [5, 4]))
(Translate [64, 10, 2]
 (HexPrism [6, 4])))
n = 6;
cylinder(h= 2, r=5, $fn=50$);

for (i = [0:n-1]) {
  rotate([0, 0, i * 360 / n])
  translate([1, -0.5, 0])
  cube([10, 1, 2]);
}
n = 6;
cylinder(h= 2, r=5, $fn=50);

for (i = [0:n-1]) {
    rotate([0, 0, i * 360 / n])
    translate([1, -0.5, 0])
    cube([10, 1, 2]);
}
Mesh Decompilers Recover Flat Programs

* Reincarnate [ICFP 2018], InverseCSG [SIGGRAPH Asia 2018], Shape2Prog [ICLR 2019], CSGNet [CVPR 2018], …
Mesh Decompilers Recover Flat Programs

Repetition of spokes is not captured by flat program

*Reincarnate [ICFP 2018],
InverseCSG [SIGGRAPH Asia 2018],
Shape2Prog [ICLR 2019], CSGNet [CVPR 2018], …

> 1500 LOC

Mesh Decompilers *

(Union
(Scale [5,5,1] (Cylinder [1,1]))
(Union
(Rotate [0,0,120]
(Translate [1,−0.5,0] (Cuboid [10,1,1])))
(Scale [10,1,1]
(Translate [0.1,−0.5,1] (Cuboid [1,1,1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Translate [−1,0.5,0]
(Scale [−1,−1,1] Cuboid [10,1,1]))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
Szalinski: CSG → Parametrized CAD

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Union
    (Rotate [0,0,120]
      (Translate [1,-0.5,0] (Cuboid [10,1,1])))
    (Scale [10,1,1]
      (Translate [0.1,-0.5,1] (Cuboid [1,1,1])))
    (Rotate [0, 0, 300]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
    )
    (Translate [-1,0,5,0]
      (Scale [-1,-1,1] Cuboid [10,1,1]))
    )
  )
  )
)
Szalinski: CSG → Parametrized CAD

\[
\text{Union} \left( \text{Scale} \left[ \frac{5}{2}, 1, 1 \right] \left( \text{Cylinder} \left[ 1, 1 \right] \right) \right) \\
\text{Union} \\
\left( \text{Rotate} \left[ 0, 0, 120 \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \\
\left( \text{Scale} \left[ 10, 1, 1 \right] \\
\left( \text{Translate} \left[ 0.1, -0.5, 1 \right] \left( \text{Cuboid} \left[ 1, 1, 1 \right] \right) \right) \right) \\
\left( \text{Rotate} \left[ 0, 0, 300 \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \\
\left( \text{Translate} \left[ -1, 0.5, 0 \right] \left( \text{Scale} \left[ -1, -1, 1 \right] \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \\
\left( \text{Rotate} \left[ 0, 0, 240 \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \\
\left( \text{Rotate} \left[ 0, 0, 60 \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \\
\left( \text{Rotate} \left[ 0, 0, 60 \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \\
\left( \text{Fold Union} \\
\left( \text{Tabulate} \left( i \ 6 \right) \\
\left( \text{Rotate} \left[ 0, 0, 60i \right] \
\left( \text{Translate} \left[ 1, -0.5, 0 \right] \left( \text{Cuboid} \left[ 10, 1, 1 \right] \right) \right) \right) \right) \right)
\]
Szalinski: CSG → Parametrized CAD

A language, called Caddy that supports CAD features & functional programming features like Fold, Tabulate, Map

Fold and Tabulate represent loops
Szalinski: CSG → Parametrized CAD

Fold and Tabulate represent loops

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Union
    (Rotate [0,0,120]
      (Translate [1,−0.5,0] (Cuboid [10,1,1])))
    (Scale [10,1,1]
      (Translate [0,−0.5,1] (Cuboid [1,1]))
      (Rotate [0,0,300]
        (Translate [1,−0.5,0] (Cuboid [10,1,1]))
        (Translate [−1,0,5,0]
          (Scale [−1,−1,1] Cuboid [10,1,1]))
        (Rotate [0,0,240]
          (Translate [1,−0.5,0] (Cuboid [10,1,1]))
          (Rotate [0,0,60]
            (Translate [1,−0.5,0] (Cuboid [10,1,1])))))
  )
)
Szalinski: CSG $\rightarrow$ Parametrized CAD

Automatically infer loops from straight line programs into **Folds, Maps, and Tabulates**

Hypothesis: Parametrized CAD programs are easier to read/customize than flat CSG programs.
Ideal Input to Szalinski

(Union
 (Cylinder [1, 5])
(Union
 (Rotate [0, 0, 0]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 180]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))
Ideal Input to Szalinski

binary nested operators like Union are presented as left-associative over multiple arguments
Term Rewriting

(Union (Cylinder [1, 5])
 (Union
   (Rotate [0, 0, 0]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 60]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 120]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 180]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 240]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 300]
     (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))
Term Rewriting

Fold Union Rewrite

(Union (Cylinder [1, 5])
 (Union
  (Rotate [0, 0, 0]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 60]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 120]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 180]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 240]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 300]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

(Union (Cylinder [1, 5, 5])
 (Fold Union (List
  (Rotate [0, 0, 0]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 60]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 120]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 180]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 240]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 300]
   (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

...
(Union (Cylinder [1, 5])
(Union
(Rotate [0, 0, 0]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 60]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
  (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
))

Fold Union Rewrite

(Union (Cylinder [1, 5, 5])
  (Fold Union (List
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

Structure Finder

(Union (Cylinder [1, 5, 5])
  (Fold Union
    (Map2 Rotate
      (List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
      (List
        (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
        (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
        ...)))
Term Rewriting

(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
)

(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
)

(Custom Solver
(Union (Cylinder [1, 5, 5])
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
)

(Structure Finder
(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])) ... )))
)

(Fold Union Rewrite
(Union (Cylinder [1, 5])
(Union
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

(Structure Finder
(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])) ... ))))

Term Rewriting

Fold Union Rewrite

Custom Solver

Structure Finder
Term Rewriting

Fold Union Rewrite

(Union (Cylinder [1, 5])
(Union
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
))

Structure Finder

(Union (Cylinder [1, 5])
(Fold Union
(List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
)

Lift Tabulate Rewrite

(Union (Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
))

Custom Solver

(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
...
))

Structure Finder

(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
...
))

Term Rewriting

Fold Union Rewrite

(Union (Cylinder [1, 5])
(Union
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
))

Structure Finder

(Union (Cylinder [1, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
...
))

Lift Tabulate Rewrite

(Union (Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
))

Custom Solver

(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
...
))

Structure Finder

(Union (Cylinder [1, 5, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
...
))
Structure Finder

List
(Op [param 1] (arg 1))
(Op [param 2] (arg 2))
(Op [param 3] (arg 3)) …

→

Map2 Op
(List [param 1] [param 2] [param 3])
(List (arg 1) (arg 2) (arg 3))
Structure Finder

Fold Union Rewrite

(Union (Cylinder [1, 5])
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

(Union (Cylinder [1, 5])
(Fold Union
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] [0, 0, 120] [0, 0, 180] [0, 0, 240] [0, 0, 300])
(List
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])) ...

List 1
List 2
Map2 applies the operator to the $i^{th}$ element of the first list and $i^{th}$ element of the second list.
Custom Solvers

The concrete list of vectors is passed to a custom solver that finds a closed form arithmetic expression.

Structure Finder

(Union (Cylinder [1, 5, 5])
 (Fold Union
  (Map2 Rotate
   (List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
   (Map2 Translate
    (Repeat 6 [1, −0.5, 0]
    (Repeat 6 (Cuboid [10, 1, 1]))))))

Custom solver

(Union (Cylinder [1, 5, 5])
 (Fold Union
  (Map2 Rotate
   (Tabulate (i 6) (0, 0, 60i))
   (Map2 Translate
    (Repeat 6 [1, −0.5, 0]
    (Repeat 6 (Cuboid [10, 1, 1]))))))
(Union (Cylinder [1, 5])
(Union
  (Rotate [0, 0, 0]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 60]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 120]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 180]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 240]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 300]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))

(Union (Cylinder [1, 5, 5])
(Fold Union (List (Rotate [0, 0, 0]
  (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
 (Rotate [0, 0, 60]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 120]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 180]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 240]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
  (Rotate [0, 0, 300]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))
Inputs to Szalinski are rarely ideal!

Outputs of mesh decompilers are correct but obscure overall structure
Ideal Input vs Actual Input

Ideal Input:
(Union
  (Cylinder [1, 5])
  (Union
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    )
  )
  (Rotate [0, 0, 120]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  )
  (Rotate [0, 0, 180]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  )
  (Rotate [0, 0, 240]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  )
  (Rotate [0, 0, 300]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
  ))

Actual Input:
(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Union
    (Rotate [0,0,120]
      (Translate [1,−0.5,0] (Cuboid [10,1,1]))
    )
    (Scale [10,1,1]
      (Translate [0.1,−0.5,1] (Cuboid [1,1,1]))
    )
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    )
    (Translate [−1,0.5,0]
      (Scale [−1,−1,1] Cuboid [10,1,1])
    )
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    )
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    ))
)
Ideal Input vs Actual Input

(Union
  (Cylinder [1, 5])
  (Union
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  ))

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Union
    (Rotate [0,0,120]
      (Translate [1,−0.5,0] (Cuboid [10,1,1])))
    (Scale [10,1,1]
      (Translate [0.1,−0.5,1] (Cuboid [1,1,1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Translate [−1,0.5,0]
      (Scale [−1,−1,1] Cuboid [10,1,1]))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  ))
Ideal Input vs Actual Input

Previous rewriting strategy no longer works!
Ideal Input vs Actual Input

Must **interleave** rewriting strategy with CAD identities to line up subexpressions
Ideal Input vs Actual Input

Must **interleave** rewriting strategy with CAD identities to line up subexpressions.

Phase ordering problem: order of rewriting matters!
Ideal Input vs Actual Input

Must **interleave** rewriting strategy with CAD identities to line up subexpressions.

Phase ordering problem: order of rewriting matters!

E-graphs* can mitigate phase ordering.

* Equality Saturation: A New Approach to Optimization. Tate, Stepp, Tatlock, Lerner. POPL’09
Semantically Equivalent, Syntactically Different

\[
\text{Rotate } [0, 0, 180] \\
\text{(Translate } [1, -0.5, 0] \text{ (Cube}[10,1,1])] \quad \equiv \quad \text{Scale } [-1, -1, 1] \\
\text{(Translate } [1, -0.5, 0] \text{ (Cube}[10,1,1])]\\n\]

Syntactic rewrite

\[
\text{Rotate } (0, 0, 180, c) \iff \text{Scale } (-1, -1, 1, c))
\]
Store Expressions in an E-graph

\[
\text{Rotate } [0, 0, 180] \\
(\text{Translate } [1, -0.5, 0] \ (\text{Cube}[10,1,1]))
\]

\[
\overset{\text{Syntactic rewrite}}{=} \\
\text{Scale } [-1, -1, 1] \\
(\text{Translate } [1, -0.5, 0] \ (\text{Cube}[10,1,1]))
\]
Store Expressions in an E-graph

\[
\text{Rotate } [0, 0, 180] \\
(\text{Translate } [1, -0.5, 0] \ (\text{Cube}[10, 1, 1])) = \text{Scale } [-1, -1, 1] \\
(\text{Translate } [1, -0.5, 0] \ (\text{Cube}[10, 1, 1]))
\]

Syntactic rewrite

\[
\text{Rotate } (0, 0, 180, c) \leftrightarrow \text{Scale } (\ -1, -1, 1, c)
\]
Store Expressions in an E-graph

Rotate $[0, 0, 180]$ (Translate $[1, -0.5, 0]$ (Cube$[10, 1, 1]$))

= Scale $[-1, -1, 1]$ (Translate $[1, -0.5, 0]$ (Cube$[10, 1, 1]$))

Syntactic rewrite

$Rotate (0, 0, 180, c)) \leftrightarrow Scale (-1, -1, 1, c)$
Custom Solvers in E-graph

Union

Fold

Outer List
Custom Solvers in E-graph

(Union (Cylinder [1, 5]))
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))...))
Custom Solvers in E-graph

Structure Finder

(Union (Cylinder [1, 5])
(Fold Union (List
(Rotate [0, 0, 0]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 60]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 120]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 180]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 240]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 300]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))
(Fold Union (List
(Rotate [0, 0, 0]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 60]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 120]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 180]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 240]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 300]
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))
(Map2 Rotate
(List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
(List
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
   (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])) ... )
Custom Solvers in E-graph

Structure Finder

(Union (Cylinder [1, 5])
 (Fold Union (List
   (Rotate [0, 0, 0]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 60]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 120]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 180]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 240]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Rotate [0, 0, 300]
     (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
   (Map2 Rotate
     (List [0, 0, 60] ... [0, 0, 300])
     (List
       (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
       (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]) ...)
   )))
))
Custom Solvers in E-graph

Structure Finder

(Union (Cylinder [1, 5, 5])
  (Fold Union (List
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))
  (Map2 Rotate
    (List [0, 0, 0] [0, 0, 60] ... [0, 0, 300])
  (List
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]) ...))
)
Custom Solvers in E-graph

Structure Finder

Union
(Cylinder [1, 5])
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 60]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 120]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 180]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 240]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 300]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))

Fold

Map2

Rotate

Outer List

List

[0, 0, 0]
[0, 0, 60]
[0, 0, 120]
[0, 0, 180]
[0, 0, 240]
[0, 0, 300]

[0, 0, 60 * i]
(Union (Cylinder [1, 5])
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))

[0, 0, 0] [0, 0, 60] [0, 0, 120] [0, 0, 180] [0, 0, 240] [0, 0, 300]

[0, 0, 60 * i]
Custom Solvers in E-graph

Union
(Fold Union (List (Rotate [0, 0, 0] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))) (Rotate [0, 0, 120] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))) (Rotate [0, 0, 180] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))) (Rotate [0, 0, 240] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))) (Rotate [0, 0, 300] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))) (List (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))

Structure Finder

Tabulate
Binds i to 6

Fold
Map2
Outer List

Outer list

[0, 0, 0] [0, 0, 60] [0, 0, 120] [0, 0, 180] [0, 0, 240] [0, 0, 300]

[0, 0, 60 * i]
(Union (Cylinder [1, 5]))
(Fold Union (List
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
...))

Structure Finder

Custom Solvers in E-graph

Fold

Union

Map2

Outer List

Rotate

List

Tabulate

Outer list

[0, 0, 0] [0, 0, 60] [0, 0, 120] [0, 0, 180] [0, 0, 240] [0, 0, 300]

[0, 0, 60 * i]
(Union (Cylinder [1, 5])
  (Fold Union (List
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  )
)

(Union (Cylinder [1, 5, 5])
  (Fold Union (List
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  )
)

Subexpressions may be in arbitrary order!
Custom Solvers for Non-Ideal Inputs

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))))
Custom Solvers for Non-Ideal Inputs

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))
  (Rotate [0, 0, 300]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  (Rotate [0, 0, 180]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  (Rotate [0, 0, 240]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
  (Rotate [0, 0, 60]
    (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))))

Expressions are arbitrarily ordered
Parameters of Rotate are not sorted
(Union
  (Scale [5, 5, 1] (Cylinder [1, 1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))
  )
)
Handling Arbitrarily Ordered Subexpressions

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))

Parameters of Rotate are NOT sorted
Handling Arbitrarily Ordered Subexpressions

The solver will need to sort the input list of parameters to find a minimal closed form solution.

Parameters of \textbf{Rotate} are NOT sorted

\[
\text{(Union} \\
\hspace{1cm} \text{(Scale [5,5,1] (Cylinder [1,1]))} \\
\hspace{1cm} \text{(Fold Union (List} \\
\hspace{2cm} \text{(Rotate [0, 0, 120]}} \\
\hspace{3cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1]))})) \\
\hspace{1cm} \text{(Rotate [0, 0, 0]}} \\
\hspace{2cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1]))})) \\
\hspace{1cm} \text{(Rotate [0, 0, 300]}} \\
\hspace{2cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1]))})) \\
\hspace{1cm} \text{(Rotate [0, 0, 180]}} \\
\hspace{2cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1]))})) \\
\hspace{1cm} \text{(Rotate [0, 0, 240]}} \\
\hspace{2cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1]))})) \\
\hspace{1cm} \text{(Rotate [0, 0, 60]}} \\
\hspace{2cm} \text{(Translate [1, \ -0.5, \ 0] (Cuboid [10, 1, 1])))))))
\]
Handling Arbitrarily Ordered Subexpressions

This unification is NOT sound!

Parameters of \textbf{Rotate} are NOT sorted

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 0]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))
Naive Solution for Finding Closed Form

(Union
  (Scale [5,5,1] (Cylinder [1,1]))
  (Fold Union (List
    (Rotate [0, 0, 120]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 0]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))

Add all permutations of the list elements in the E-graph
Naive Solution Causes the AC-Matching Problem

Add all permutations of the list elements in the E-graph

Exponentially many choices in an E-graph due to associative-commutative operations like permuting lists, called AC-matching in the SMT community
Inverse Transformations

Key insight: allows solvers to speculatively transform their inputs to enable more profitable rewriting
Inverse Transformations

Key insight: allows solvers to speculatively transform their inputs to enable more profitable rewriting

If a solver cannot simplify $A$, but it can simplify $f(A)$ to $B$, then $f^{-1}(B)$ can be unified with $A$
Inverse Transformations

(Union
 (Scale [5,5,1] (Cylinder [1,1])))
(Fold Union (List
 (Rotate [0, 0, 120]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
 (Rotate [0, 0, 0]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
 (Rotate [0, 0, 300]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
 (Rotate [0, 0, 180]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
 (Rotate [0, 0, 240]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
 (Rotate [0, 0, 60]
 (Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))

Goal
(Union
 (Cylinder [1, 5, 5])
 (Fold Union
 (Tabulate (i 6)
 (Rotate [0, 0, 60i]
 (Translate [1, −0.5, 0]
 (Cuboid [10, 1, 1]))))))
Inverse Transformations

Goal

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60]
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Structure Finder

(Union
(Scale [5,5,1] (Cylinder [1,1]))
(Fold Union (List
(Rotate [0, 0, 120]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 0]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 300]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 240]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))
(Rotate [0, 0, 60]
(Translate [1, −0.5, 0] (Cuboid [10, 1, 1]))))))
Inverse Transformations

Structure Finder

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))))

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 120]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))
}

Union
(Scale [5, 5, 1] (Cylinder [1, 1]))
(Fold Union (List
(Rotate [0, 0, 120]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 0]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 300]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 180]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 240]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))
(Rotate [0, 0, 60]
(Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))
Inverse Transformations

Structure Finder

Goal

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))
)
)...
Inverse Transformations

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))))

Parameters of Rotate are NOT sorted

Solver must sort the list to find closed form!

This unification is NOT sound!

Goal
(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

Solvers allowed to transform their input however they want!

BUT they must 'undo' the transformation to restore equivalence.

Goal

(Union
 (Cylinder [1, 5, 5])
 (Fold Union
  (Tabulate (i 6)
   (Rotate [0, 0, 60i]
    (Translate [1, -0.5, 0]
     (Cuboid [10, 1, 1]))))))
Inverse Transformations

(Solver sorts the list to find closed form!)

Goal

Solvers allowed to transform their input however they want!

BUT they must 'undo' the transformation to restore equivalence
Inverse Transformations

Solvers allowed to transform their input however they want!

BUT they must 'undo' the transformation to restore equivalence
Inverse Transformations

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))

(Solver sorts the list to find closed form!
(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Solvers allowed to transform their input however they want!

BUT they must 'undo' the transformation to restore equivalence
Inverse Transformations

Solver sorts the list to find closed form!

Solver annotates the expression with the profitable permutation

Solvers allowed to transform their input however they want!

BUT they must 'undo' the transformation to restore equivalence
Inverse Transformations

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))

Solver sorts the list to find closed form!

(Solver annotates the expression with the profitable permutation)

If a solver cannot simplify A, but it can simplify \( f(A) \) to B, then \( f^{-1}(B) \) can be unified with A
Inverse Transformations

(Fold Union
(Map2 Rotate
(List [0, 0, 120] [0, 0, 0] [0, 0, 300] [0, 0, 180] [0, 0, 240] [0, 0, 60])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))

Solver sorts the list to find closed form!
(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i])
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))

Solver annotates the expression with the profitable permutation

This unification is sound!

Union
(Fold Union
(Tabulate (i 6) [0, 0, 60 * i])
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6) (Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Goal
This unification is sound!

If a solver cannot simplify A, but it can simplify f(A) to B, then f⁻¹(B) can be unified with A
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))))

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))
)

Propagate and Eliminate
Syntactic
rewrites

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))
)

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))
)
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Propagate and Eliminate

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Syntactic rewrites

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
  (Map2 Rotate
    (Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
  (Repeat 6
    (Translate [1, −0.5, 0]
      (Cuboid [10, 1, 1]))))))

Propagate and Eliminate

(Syntactic rewrites
  Effectively a no-op, but allows sorting the concrete list equivalent to Map2)

(Fold Union
  (Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
  (Map2 Rotate
    (Unsort <1 5 0 3 4 2>
      (Tabulate (i 6) [0, 0, 60 * i]))
  (Repeat 6
    (Translate [1, −0.5, 0]
      (Cuboid [10, 1, 1]))))))

Goal

(Union
  (Cylinder [1, 5, 5])
  (Fold Union
    (Tabulate (i 6)
      (Rotate [0, 0, 60i]
        (Translate [1, −0.5, 0]
          (Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))})

Syntactic rewrites
Propagate and Eliminate
Effectively a no-op, but allows sorting the concrete list equivalent to Map2

(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>
(Map2 Rotate
(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]) ))))

Unsort added to the e-class of Map2 and the Outer List

Goal
(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60 * i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))) )))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))))

Propagate and Eliminate
Syntactic rewrites

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1]))))))

Goal
(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1])))))))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Syntactic rewrites

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 180]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))
Invert Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1])))))

Propagate and Eliminate

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>
(Map2 Rotate
(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Syntactic rewrites

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1,−0.5,0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i])))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Propagate and Eliminate
Syntactic rewrites

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

Sorts Outer List which is equivalent to Map2

Goal
(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Propagate and Eliminate
Syntactic rewrites

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i]))
(Repeat 6
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Effectively a no-op, but allows sorting the concrete list equivalent to Map2

Structure Finder and Custom Solvers apply on this sorted list

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))
Inverse Transformations

(Fold Union
(Unsort <1 5 0 3 4 2> (Sort <1 5 0 3 4 2>)
(Map2 Rotate
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) [0, 0, 60 * i])
(Repeat 6
(Translate [1, -0.5, 0]
(Cuboid [10, 1, 1]))))))

Custom solvers on the sorted outer list

(Fold Union
(Unsort <1 5 0 3 4 2> (Tabulate (i 6) (Rotate [0, 0, 60 * i]) (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))

Structure Finder and Custom Solvers apply on this sorted list

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6) (Rotate [0, 0, 60i] (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))
Inverse Transformations

Fold Union is invariant to list order

Syntactic rewrite to eliminate Unsort

Custom solvers on the sorted outer list
Inverse Transformations

Syntactic rewrite to eliminate

Unsort

(Fold Union (Cylinder [1, 5, 5])

(Fold Union (Tabulate (i 6)

(Rotate [0, 0, 60i]

(Translate [1,

−

0.5, 0]

(Cuboid [10, 1, 1])))))))

Goal

(Fold Union (Unsort <1 5 0 3 4 2>

(Tabulate (i 6)

(Rotate [0, 0, 60 * i]

(Translate [1,

−

0.5, 0]

(Cuboid [10, 1, 1]))))))

Fold Union is invariant to list order

(Fold Union (Unsort <1 5 0 3 4 2>

(Sort <1 5 0 3 4 2>

(Map2 Rotate

(Unsort <1 5 0 3 4 2>

(Tabulate (i 6) [0, 0, 60 * i]))

(Repeat 6

(Translate [1,

−

0.5, 0]

(Cuboid [10, 1, 1]))))))

Custom solvers on the sorted outer list

Example transformations: sorting, partitioning, cartesian-to-spherical

Rewrites applied until saturation (or timeout) and a cost function (AST size) used to extract best program

Flexibly combines solvers with an Egraph-driven rewrite system
End-to-End Evaluation

Results of running Szalinski on outputs of Reincarnate*

* [ICFP 2018]
2127 programs from Thingiverse:

- Tiny, size < 30: 769 programs
- Small, size < 100: 786 programs
- Medium, size < 300: 374 programs
- Large, size > 300: 198 programs

Larger programs shrink more.

All optimizations take < 1 second.

Scalability

- % shrunk
- Tiny
- Small
- Medium
- Large

- 0
- 20
- 40
- 60
- 80
- 100

All optimizations take < 1 second.
Examples

(Fold Difference
  (List (Union
    (Cylinder [100, 80, 80])
    (Cylinder [50, 120, 120]))
  (Translate [0, 0, -1] (Cylinder [102, 25, 25]))
  (Fold Union (Tabulate (i 60)
    (Rotate [0, 0, 6 * i]
    (Translate [125, 0, 0]
    (Scale [2.5, 1, 1]
    (Rotate [0, 0, 45]
    (Translate [0, 0, 25]
    (Cuboid [10, 10, 52])))))))))

350 LOC

250 LOC

(Fold Union
  (Tabulate (i 10) (j 5)
    (Translate
      [12.2 * i + 12.2, 12.2 * j + 12.2, 0]
    (Difference
      (Cylinder [13, 7.1, 7.1])
    (Translate [0, 0, 3]
      (Cylinder [11, 5.1, 5.1]))))))

250 LOC

100 LOC

(Fold Union
  (Tabulate (i 12)
    (Translate [0, 13* i, 0]
      (Fold Difference
        (List
          (Cuboid [53.1 14.5 58])
        (Translate [1.5, 1.5, 1.5]
          (Cuboid [51.6, 11.5, 56.6]))
        (Translate [0 0 58]
          (Rotate [0, 45, 0]
            (Cuboid [101.5, 14.5, 100]))))))
Implementation

~ 2000 LOC in Rust, ~ 65 rewrites
https://github.com/uwplse/szalinski

Built on Max Willsey’s egg E-graph library
https://github.com/mwillsey/egg
Some Editability Still Lacking

We’ve recovered an expression from a flat set of triangles!

Primitives now parameterized, but we’ve lost some *editability*!

We’re gonna need a wider aspect ratio for what comes next …

See PLDI 2020 for more.
Today’s Menu

1. Everything is a Program (Output)
   - CAD ➔ CSG ➔ STL ➔ G-code ➔ [Printed Object]

2. Decompiling Surfaces to Expressions
   - CSG ➔ STL
   - [ICFP 2018]

3. Rerolling Loops for Editability
   - CAD ➔ CSG
   - [PLDI 2020]

4. egg : E-Graphs Good 🍳
   - Input Term ➔ Initialize ➔ E-graph ➔ Extract ➔ Optimized Term ➔ Apply Rewrites
Equality Saturation in egg

Input Term → Initialize → E-graph → Extract → Optimized Term

Apply Rewrites
Equality Saturation in egg

E-graphs originally developed within the automated theorem proving community.

Used to in congruence procedures to efficiently represent many equivalent terms.
Equality Saturation in egg

E-graphs originally developed within the automated theorem proving community.

Used to in congruence procedures to efficiently represent many equivalent terms.

\[ a \equiv b \text{ implies } f(a) \equiv f(b) \]
Equality Saturation in egg

E-graphs originally developed within the automated theorem proving community.

Equality saturation repurposes e-graphs to program optimization.
Equality Saturation in egg

```python
1  def equality_saturation(expr, rewrites):
2      egraph = initial_egraph(expr)
3
4      while not egraph.is_saturated_or_timeout():
5
6          for rw in rewrites:
7              for (subst, eclass) in egraph.ematch(rw.lhs):
8                  eclass2 = egraph.add(rw.rhs.subst(subst))
9                  egraph.merge(eclass, eclass2)
10
11      return egraph.extract_best()
```
**Equality Saturation in egg**

Simple: ~ 10 line algorithm!

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)

    return egraph.extract_best()
```
Equality Saturation in egg

Simple: ~ 10 line algorithm!

Extensible: add domain-specific rewrites

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)

    return egraph.extract_best()
```
Equality Saturation in egg

```python
1  def equality_saturation(expr, rewrites):
2      egraph = initial_egraph(expr)
3
4      while not egraph.is_saturated_or_timeout():
5
6          for rw in rewrites:
7              for (subst, eclass) in egraph.ematch(rw.lhs):
8                  eclass2 = egraph.add(rw.rhs, subst)
9                  egraph.merge(eclass, eclass2)
10    return egraph.extract_best()
```
Equality Saturation in egg

Simple: ~ 10 line algorithm!

Extensible: add domain-specific rewrites

(potentially) Complete: saturation implies you’ve found all equivalent programs.

Extensible: use domain-specific cost functions

```
def equality_saturation(expr):
    egraph = initial_egraph(expr)
    while not egraph.is_saturated_or_timeout():
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):
                eclass2 = egraph.add(rw.rhs(rw.lhs, subst), eclass)
                egraph.merge(eclass, eclass2)
    return egraph.extract_best()
```
Needs of Equality Saturation

Need to scale: e-graphs never forget.

Unlike SMT, equality saturation never makes a mistake.
What’s the bottleneck? Congruence!
What’s the bottleneck? Congruence!
What’s the bottleneck? Congruence!

Violates key congruence invariant!
What’s the bottleneck? Congruence!

Violates key congruence invariant!
What’s the bottleneck? Congruence!

Violates key congruence invariant!

Congruence restored.
Restoring deduce invariant also easy now!
Step 1: Phased EQSAT

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():

        # reading and writing is mixed
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):

                # in traditional equality saturation,
                # matches can be applied right away
                # because invariants are always maintained
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)

                # restore the invariants after each merge
                egraph.rebuild()

        return egraph.extract_best()
```

Traditional:
Always maintain invariants
Step 1: Phased EQSAT

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():

        # reading and writing is mixed
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):

                # in traditional equality saturation,
                # matches can be applied right away
                # because invariants are always maintained
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)

                # restore the invariants after each merge
                egraph.rebuild()

        return egraph.extract_best()
```

Traditional:
Always maintain invariants

Restore congruence, dedup, etc. after every rewrite :\
Step 1: Phased EQSAT

```python
1  def equality_saturation(expr, rewrites):
2      egraph = initial_egraph(expr)
3
4      while not egraph.is_saturated_or_timeout():
5
6          # reading and writing is mixed
7          for rw in rewrites:
8              for (subst, eclass) in egraph.ematch(rw.lhs):
9                  # in traditional equality saturation,
10                 # matches can be applied right away
11                 # because invariants are always maintained
12                 eclass2 = egraph.add(rw.rhs.subst(subst))
13                 egraph.merge(eclass, eclass2)
14
15          # restore the invariants after each merge
16          egraph.rebuild()
17
18      return egraph.extract_best()
```

### Traditional:
Always maintain invariants

### Phased:
Separate reads, writes.
Restore inv each iter.
Step 1: Phased EIQSAT

**Traditional:**
Always maintain invariants

**Phased:**
Separate reads, writes. Restore inv each iter.

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        # reading and writing is mixed
        for rw in rewrites:
            for (s, ec) in egraph.ematch(rw.lhs):
                # invariants are always maintained
                eclass = egraph.add(rw.rhs.subst(s, ec))
                egraph.merge(eclass, eclass)
                egraph.rebuild()

    return egraph.extract_best()
```

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        matches = []

        # read-only phase, invariants are preserved
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):
                matches.append((rw, subst, eclass))

        # write-only phase, temporarily break invariants
        for (rw, subst, eclass) in matches:
            eclass = egraph.add(rw.rhs.subst(subst))
            eclass = egraph.merge(eclass, eclass)
            egraph.rebuild()

    return egraph.extract_best()
```
Step 1: Phased EQSAT

**Traditional:**
Always maintain invariants

**Phased:**
Separate reads, writes. Restore inv each iter.

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        # reading and writing is mixed
        for rw in rewrites:
            for (s, subst, eclass) in rw:
                # invariants are always maintained
                # matches can be applied right away
                # because invariants are always maintained
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)

        # restore the invariants once per iteration
        egraph.rebuild()

    return egraph.extract_best()
```

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        matches = []

        # read-only phase, invariants are preserved
        for rw in rewrites:
            for (subst, eclass) in egraph.ematch(rw.lhs):
                matches.append((rw, subst, eclass))

        # write-only phase, temporarily break invariants
        for (rw, subst, eclass) in matches:
            eclass2 = egraph.add(rw.rhs.subst(subst))
            egraph.merge(eclass, eclass2)

        # restore the invariants once per iteration
        egraph.rebuild()

    return egraph.extract_best()
```
Step 1: Phased Eqsat

Traditional:
- Always maintain invariants

Phased:
- Separate reads, writes.
- Restore invariant each iteration

---

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        # reading and writing is mixed
        for rw in rewrites:
            for (sub, eclass) in egraph.ematch(rw.lhs):
                eclass2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(eclass, eclass2)
        egraph.result()

    return egraph
```

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        matches = []

        # read-only phase, invariants are preserved
        for rw in rewrites:
            for (sub, eclass) in egraph.ematch(rw.lhs):
                matches.append((rw, sub, eclass))

        # write-only phase, temporarily break invariants
        for (rw, subst, eclass) in matches:
            eclass2 = egraph.add(rw.rhs.subst(subst))
            egraph.merge(eclass, eclass2)

        egraph.rebuild()

    return egraph.extract_best()
```
Step 2: Rebuilding

```python
def rebuild():
    while self.worklist.len() > 0:
        # empty the worklist into a local variable
        todo = take(self.worklist)
        # canonicalize and deduplicate the eclass refs
        # to save calls to repair
        todo = { self.find(eclass) for eclass in todo }
        for eclass in todo:
            self.repair(eclass)

    # update the hashcons so it always points
    # canonical enodes to canonical eclasses
    for (p_node, p_eclazz) in eclass.parents:
        self.hashcons.remove(p_node)
        p_node = self.canonicalize(p_node)
        self.hashcons[p_node] = self.find(p_eclazz)

    # deduplicate the parents, noting that equal
    # parents get merged and put on the worklist
    new_parents = {}
    for (p_node, p_eclazz) in eclass.parents:
        p_node = self.canonicalize(p_node)
        if p_node in new_parents:
            self.merge(p_eclazz, new_parents[p_node])
        new_parents[p_node] = self.find(p_eclazz)
    eclass.parents = new_parents
```

Simple worklist algorithm iteratively restores invariants.

Key: cache-friendly datastructures!

Possible b/c no need for rollback.
Does it work?
Does it work?
Does it work?
Does it work?

Yes.
Does it work?

Yes.
Case Study: Herbie

Find and fix floating-point problems.

Try  •  Install  •  Learn

\[ \sqrt{x+1} - \sqrt{x} \rightarrow \frac{1}{\sqrt{x+1} + \sqrt{x}} \]

Herbie detects inaccurate expressions and finds more accurate replacements. The red expression is inaccurate when \( x > 1 \); Herbie's replacement, in blue, is accurate for all \( x \).

https://herbie.uwplse.org
Case Study: Herbie

![Bar chart showing minutes spent in simplification for different stages of the Racket implementation.](chart.png)
Case Study: Herbie

Faster and better.
More egg Users

\[
A \oplus (B \oplus C) = \oplus(A, B, C) \quad (\oplus \text{ is assoc. \& comm.})
\]

\[
A \otimes (B \otimes C) = \otimes(A, B, C) \quad (\otimes \text{ is assoc. \& comm.})
\]

\[
A \otimes (B \oplus C) = A \otimes B \oplus A \otimes C \quad (\otimes \text{ distributes over } \oplus)
\]

\[
\sum_i (A \oplus B) = \sum_i A \oplus \sum_i B
\]

\[
\sum_i \sum_j A = \sum_{i,j} A \quad \text{(requires } i \notin A) \]

\[
A \otimes \sum_i B = \sum_i (A \otimes B) \quad \text{(requires } i \notin A) \]

\[
\sum_i A = A \otimes \text{dimension}(i)
\]

Optimize linear algebra via relational algebra.
[VLDB 20]

Remy Wang et al.
Today's Menu

1. Everything is a Program (Output)
   - CAD ➔ CSG ➔ STL ➔ G-code ➔ [ICFP 2018]

2. Decompiling Surfaces to Expressions
   - CSG ➔ STL ➔ [ICFP 2018]

3. Rerolling Loops for Editability
   - CAD ➔ CSG ➔ [PLDI 2020]

4. egg : E-Graphs Good
   - Input Term ➔ Initialize E-graph ➔ Extract Optimized Term ➔ Apply Rewrites
Amazing Collaborators

Chandakana Nandi
Max Willsey
Adam Anderson
Remy Wang
Oliver Flatt

Eva Darulova
James Wilcox
Pavel Panchekha
Taylor Blau
Dan Grossman
For more, check out:

Reincarnate and Szalinski

http://incarnate.uwplse.org

Equality Saturation with egg

https://github.com/mwillsey/egg
Thank You!

1. Everything is a Program (Output)
   - CAD ➔ CSG ➔ STL ➔ G-code ➔ [ICFP 2018]

2. Decompiling Surfaces to Expressions
   - CSG ➔ STL ➔ [ICFP 2018]

3. Rerolling Loops for Editability
   - CAD ➔ CSG ➔ [PLDI 2020]

4. egg : E-Graphs Good