**Szalinski**: A Tool for Synthesizing Structured CAD Models with Equality Saturation and Inverse Transformations

PLDI 2020

Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, Zachary Tatlock
Designing Physical Objects is Programming!
CAD and 3D Printing everywhere!

Make your own models

Share with others
CAD and 3D Printing everywhere!

Make your own models

Editability is key!

Share with others
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], …

(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])))
  ))
Mesh Decompilers Recover Flat Programs

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

> 1500 LOC

(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])))))
Mesh Decompilers Recover Flat Programs

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

> 1500 LOC

(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]))))))

Affine operators

Primitives

Mesh Decompilers

Scale [5,5,1] (Cylinder [1,1])
Mesh Decompilers Recover Flat Programs

> 1500 LOC

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

**Primitives**

- Union
  - (Scale [5, 5, 1] (Cylinder [1, 1]))

**Binary operators**

- 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])))

**Affine operators**
Mesh Decompilers Recover Flat Programs

Repetition of spokes is not captured by flat program

> 1500 LOC

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

(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: flat CAD $\rightarrow$ 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]))
    (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: flat CAD → 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,-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])}}}
  {Rotate [0,0,60]
    {Translate [1,-0.5,0] (Cuboid [10,1,1])}}}
```

Szalinski
This talk
Szalinski: flat CAD  ➔ parametrized CAD

Fold and Tabulate represent loops

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

\[
\text{(Union}
\text{)}
\text{(Scale [5,5,1] (Cylinder [1,1])))
\text{(Union}
\text{)}
\text{(Rotate [0,0,120]}
\text{(Translate [1,−0.5,0] (Cuboid [10,1,1])))}
\text{(Scale [10,1,1])}
\text{(Translate [0.1,−0.5,1] (Cuboid [1,1,1])))}
\text{(Rotate [0, 0, 300]}
\text{(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))}
\text{(Translate [−1,0,5,0]}
\text{(Scale [−1,−1,1] Cuboid [10,1,1]))}
\text{(Rotate [0, 0, 240]}
\text{(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))}
\text{(Rotate [0, 0, 60]}
\text{(Translate [1, −0.5, 0] (Cuboid [10, 1, 1])))}
\text{)}
\text{(Cylinder [1, 5, 5])}
\text{(Fold Union}
\text{(Tabulate (i 6))}
\text{(Rotate [0, 0, 60i]}
\text{(Translate [1,−0.5,0]}
\text{(Cuboid [10, 1, 1]))))}
Szalinski: flat CAD $\rightarrow$ parametrized CAD

**This talk**

- (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 [10,1,1])))
    - (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])))
)

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

- (Union
  - (Cylinder [1,5,5])
  - (Fold Union
    - (Tabulate (i 3)
      - (Rotate [0,0,120i]
        - (Translate [1,−0.5,0] (Cuboid [10,1,1]))))))

- (Union
  - (Cylinder [1,5,5])
  - (Fold Union
    - (Tabulate (i 8)
      - (Rotate [0,0,45i]
        - (Translate [1,−0.5,0] (Cuboid [15,1,1]))))))

- (Union
  - (Cylinder [1,5,5])
  - (Fold Union
    - (Tabulate (i 6)
      - (Rotate [0,0,60i]
        - (Translate [1,−0.5,0] (Cuboid [15,1,1]))))))

- (Union
  - (Cylinder [1,5,5])
  - (Fold Union
    - (Tabulate (i 6)
      - (Rotate [0,0,45i]
        - (Translate [1,−0.5,0] (Cuboid [15,1,1]))))))

**Fold and Tabulate represent loops**
Szalinski: flat CAD  ➔ parametrized CAD

Automatically infer loops from straight line programs in the form of **Folds, Maps, and Tabulates**

Hypothesis: Parametrized programs are easier to read/customize than flat programs
(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
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])))
)))
(Union (Cylinder [1, 5])
  (Union
    (Rotate [0, 0, 0]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 60]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 120]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 180]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 240]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
    (Rotate [0, 0, 300]
      (Translate [1, 0, 0] (Cuboid [10, 1, 1])))
  )))

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

(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]))))

(Fold Union Rewrite
(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

(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 (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])))
  …

(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])))
    …

(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]))
  …

(*
(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]))
    …

(Map2 Translate
 (Repeat 6 [1, -0.5, 0]
  (Repeat 6 (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])))))
  (Fold Union (List
    (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]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))
(Union (Cylinder [1, 5])
  (Fold Union (List
    (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]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))

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

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])))))
  (Fold Union (List
    (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]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1])))))
(Union (Cylinder [1, 5])
  (Fold Union (List
    (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]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))))))

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

List 2
(Union (Cylinder [1, 5])
  (Fold Union (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)))
Map2 applies the operator to the $i^{th}$ element of the first list and $i^{th}$ element of the second list.
Map2 applies the operator to the $i^{th}$ element of the first list and $i^{th}$ element of the second list. Also applies to this list.
The concrete list of vectors is passed to a custom solver that finds a closed form arithmetic expression.
Term Rewriting

Fold Union Rewrite

Structure Finder

Custom Solver

Structure Finder

Lift Tabulate Rewrite

(Fold Union (List (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]))) ...)

(Union (Cylinder [1, 5, 5]) (Fold Union (Tabulate (i 6) (0, 0, 60i)) (Map2 Translate (Repeat 6 [1, −0.5, 0] (Repeat 6 (Cuboid [10, 1, 1])))))))

(Union (Cylinder [1, 5, 5]) (Fold Union (Tabulate (i 6) (0, 0, 60i)) (Map2 Translate (Repeat 6 [1, −0.5, 0] (Repeat 6 (Cuboid [10, 1, 1]))))))
Term Rewriting

Inputs to Szalinski are rarely ideal!
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])))))

(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])))
(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])))))

(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])))
(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

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

Must **interleave** rewriting strategy with CAD identities to line up subexpressions
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 solve phase ordering

---

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

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

Rotate [0, 0, 180] is replaced by equivalent Scale [-1, -1, 1]
Semantically Equivalent, Syntactically Different

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

Syntactic rewrite

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

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

Syntactic rewrite

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

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

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

\(\text{Rotate} \ (0, 0, 180, c) \iff \text{Scale} \ (-1, -1, 1, c)\)
(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

Outer List
Custom Solvers in E-graph

Structure Finder

Union

Fold

Map2

Outer List

(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, 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
Fold
Map2
Outer List
Rotate
List

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

(Outer List
(Fold Union (List
(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])) ...)))

CAD arguments
Custom Solvers in E-graph

Structure Finder

Union
(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

Rotate

List

Map2

Outer List

Fold

[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

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]))
                (Rotate [0, 0, 60 * i] (Cuboid [10, 1, 1]))
              ))
            ))
          ))
    (List
      (Translate [1, -0.5, 0] (Cuboid [10, 1, 1]))
      ...)}})
Custom Solvers in E-graph

Structure Finder

Union

Fold

Map2

Outer List

Fold

List

Tabulate

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

[0, 0, 60 * i]
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
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]))))))

[0, 0, 60 * i]
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]))))))
Parameters of **Rotate** are NOT sorted
Parameters of \text{Rotate} are NOT sorted.

List of vectors must be sorted for the solver to be able to find the closed form and unify the Tabulate with the concrete list.

Custom Solvers for Non-Ideal Inputs
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])))
)))

Parameters of Rotate are NOT sorted

[0, 0, 60 * i]
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

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

Goal
Inverse Transformations

Structure Finder

Goal
Inverse Transformations

Structure Finder

Goal

Union

(Fold Union

(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])))))

(Fold Union

(Cylinder [1, 5, 5])

(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]))))))

(Union

(Cylinder [1, 5, 5])

(Fold Union

(Tabulate (i 6)

(Rotate [0, 0, 60i]

(Repeat 6

(Translate [1, -0.5, 0]

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

(Union

(Rotate [0, 0, 60]

(Translate [1, -0.5, 0]

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

(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

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])))))

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

Goal

(Union
(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60i]
(Rotate [0, 0, 60i]
(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])))))

Solver permutes the list to find closed form!

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

Goal
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]))))))

(Scene 2)
(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]))))))

Solver permutes the list to find closed form!

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

Goal
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 permutes the list to find closed form!)

(Fold Union
(Map2 Rotate
(Union <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)

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
(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 permutes 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]))))))

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 permutes 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])))))

Solver annotates the expression with the profitable permutation

Goal

Flexibly combines solvers with an egraph-driven rewrite system

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 permutes 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]))))

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 permutes
the list to find closed form!

Solver annotates
the expression with the profitable permutation

(Unsort <1 5 0 3 4 2>
(Tabulate (i 6) [0, 0, 60 * i]))

-Unsort
Tabulate

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

(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
Propagate and Eliminate

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

Goal

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

Syntactic rewrites
Propagate and Eliminate
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
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

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

(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

(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

Goal

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

<1 5 0 3 4 2>
Tabulate

(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
List
Sort
Fold
Union
Map2
Outer List
Unsort

<1 5 0 3 4 2>
Tabulate

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

Unsort
Fold
Union
Map2
Outer List
Tabulate
List
Unsort
Sort
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

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

(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]))))))
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

(Map2 Rotate
(Tabulate (i 6) [0, 0, 60 * i]))
(Translate [1, −0.5, 0]
(Cuboid [10, 1, 1]))))))

Syntactic
rewrites

Propagate and
Eliminate
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

Structure Finder and Custom Solvers apply on this sorted list

Goal

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

Custom solvers on the sorted outer list

Structure Finder and Custom Solvers apply on this sorted list

Goal
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])))))))

Fold Union is invariant to list order

Syntactic rewrite to eliminate Unsort

(goal)

(Cylinder [1, 5, 5])
(Fold Union
(Tabulate (i 6)
(Rotate [0, 0, 60 * i])
(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
Implementation

~ 2000 LOC in Rust

65 rewrites

https://github.com/uwplse/szalinski

Uses the Egg E-graph library: https://github.com/mwillsey/egg

Talk to Max about Egg!
End-to-End Evaluation

Results of running Szalinski on outputs of Reincarnate*

* [ICFP 2018]
2127 programs from Thingiverse

Tiny: AST size < 30
Small: 30 < AST size < 100
Medium: 100 < AST size < 300
Large: AST size > 300

Larger programs shrink more
< 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]))))))))

(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]))))))

(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]))))))))
Szalinski: A Tool for Synthesizing Structured CAD Models with Equality Saturation and Inverse Transformations

https://github.com/uwplse/szalinski

Inverse Transformations with E-graphs to find concise, structured programs in < 1 second

Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, Zachary Tatlock