e-graphs

for

next-gen PL tools

Max Willsey

University of Washington

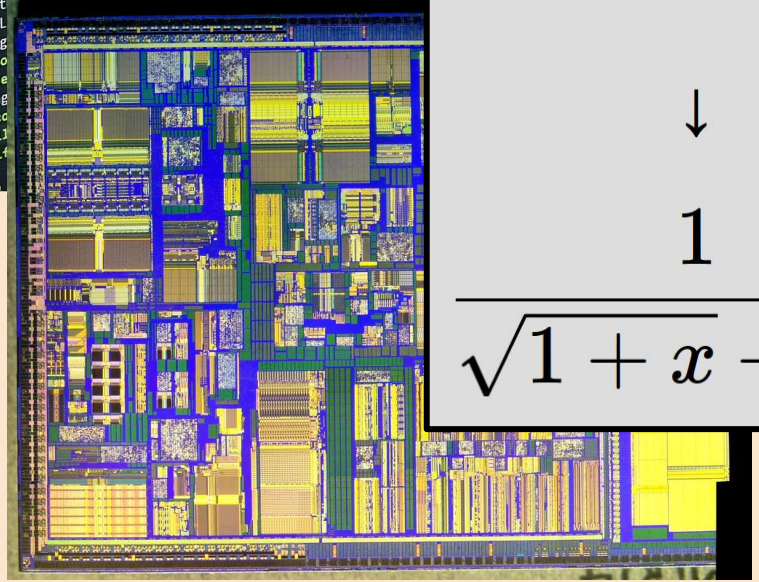
2023

PL tools


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
again = true;
continue;
} else if (sInput[iLength - 3] != '.') {
again = true;
continue;
} while (++iN < iLength) {
if (isdigit(sInput[iN])) {
continue;
} else if (iN == (iLength - 3)) {

next-gen PL tools

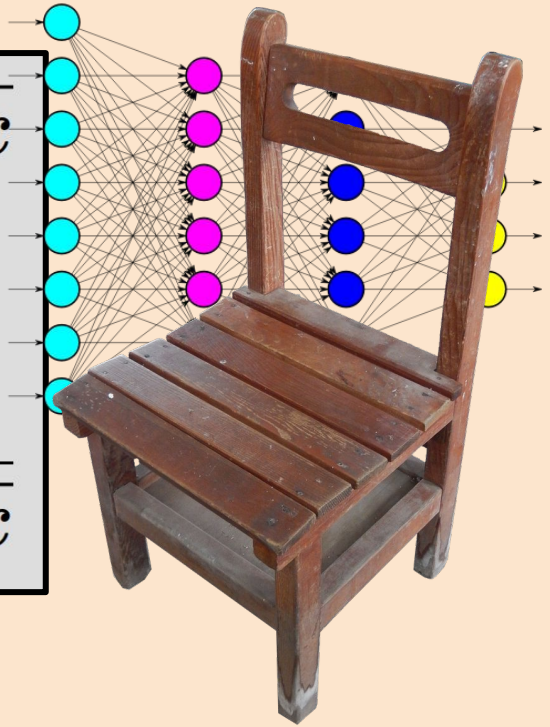
```
string sInput;
int iLength, iN;
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLength = sInput.length();
    if (iL
        ag
        co
    } else
        ag
        co
    } whil
        if
```



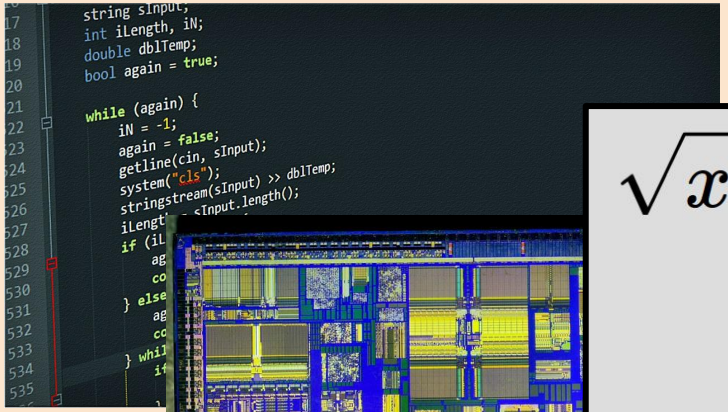
$$\sqrt{x+1} - \sqrt{x}$$

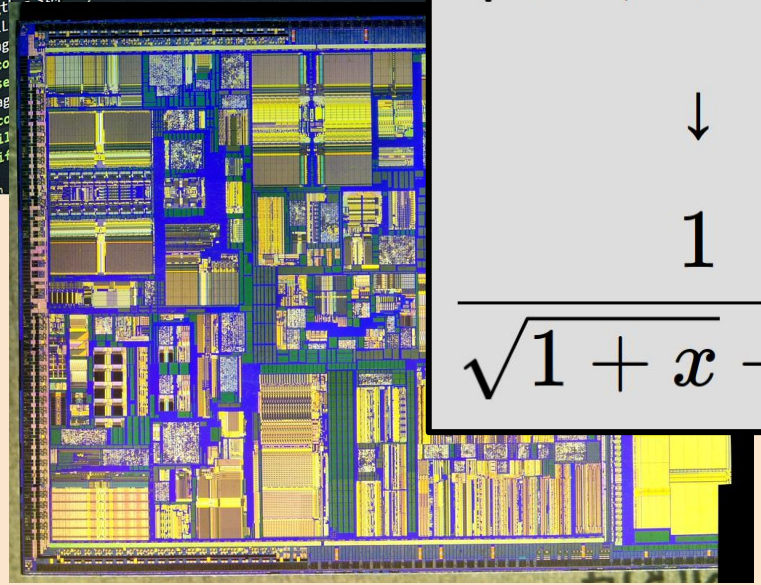
$$\downarrow$$

$$\frac{1}{\sqrt{1+x} + \sqrt{x}}$$



PL is everywhere

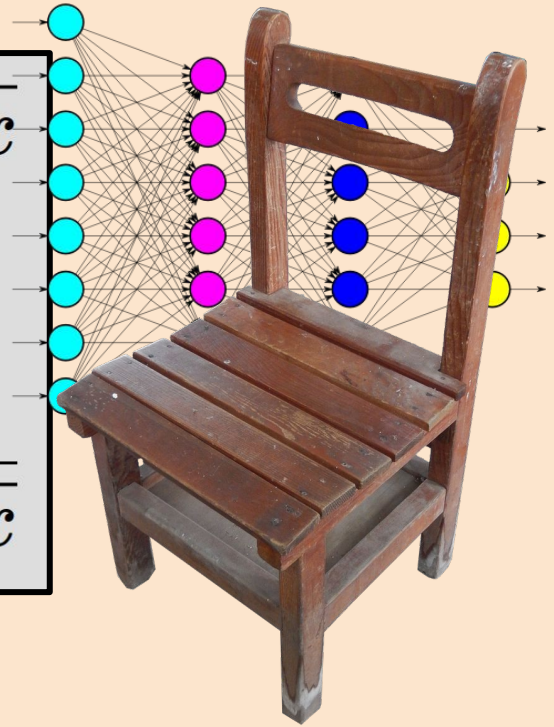




$$\sqrt{x+1}-\sqrt{x}$$

$$\downarrow$$

$$\frac{1}{\sqrt{1+x}+\sqrt{x}}$$



equational reasoning

Fastly

```
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLengt  <Input.length();
    if (iL
        ag
        co
    } else
        ag
        co
    } whil
        if
```

Intel

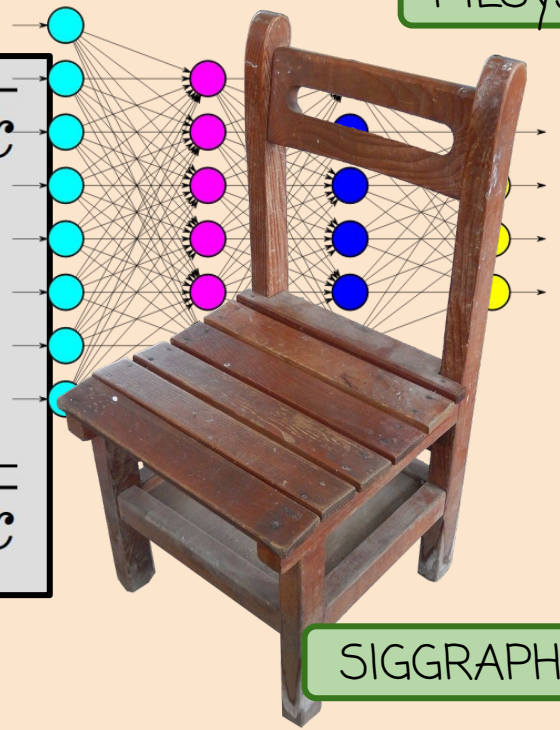
Herbie

$$\sqrt{x+1} - \sqrt{x}$$

$$\downarrow$$

$$\frac{1}{\sqrt{1+x} + \sqrt{x}}$$

MLSys '21



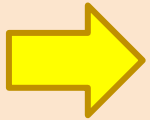
SIGGRAPH '22

$$(a * 2) / 2$$

$$a$$

$$(a * 2) / 2$$

$$a$$

(a * 2) / 2  a

rewrite it!

useful

(x * y) / z = x * (y / z)

x / x = 1

x * 1 = x

not so useful

x * 2 = x << 1

x * y = y * x

x = x * 1

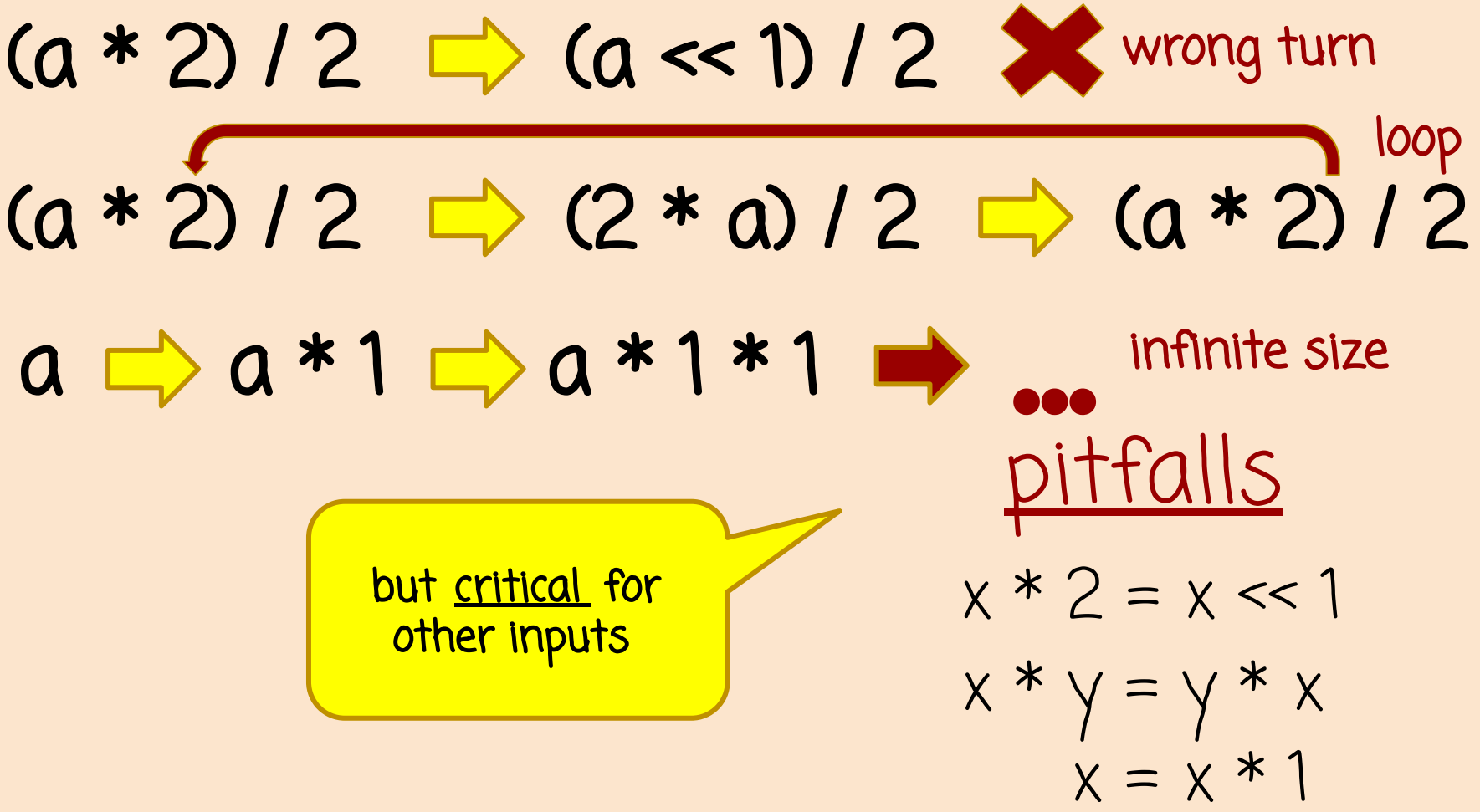
$(a * 2) / 2 \Rightarrow a * (2 / 2) \Rightarrow a * 1 \Rightarrow a$

happy path

$(x * y) / z = x * (y / z)$

$x / x = 1$

$x * 1 = x$


(a * 2) / 2
(a << 1) / 2
wrong turn
loop
(a * 2) / 2
(2 * a) / 2
(a * 2) / 2
a
a * 1
a * 1 * 1
infinite size
pitfalls
but critical for other inputs
x * 2 = x << 1
x * y = y * x
x = x * 1

(a * 2) / 2 ⇨ (a << 1) / 2 ✖ wrong turn

loop

(a * 2) / 2 ⇨ (2 * a) / 2 ⇨ (a * 2) / 2

Whitfield & Soffa 1997

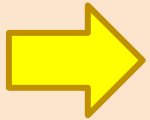
"the order of applying code transformations [...] can have an impact on the quality of code produced"

but critical for other inputs

x * 2 = x << 1

x * y = y * x

x = x * 1

(a * 2) / 2  a

which rewrite? when?

useful

(x * y) / z = x * (y / z)

x / x = 1

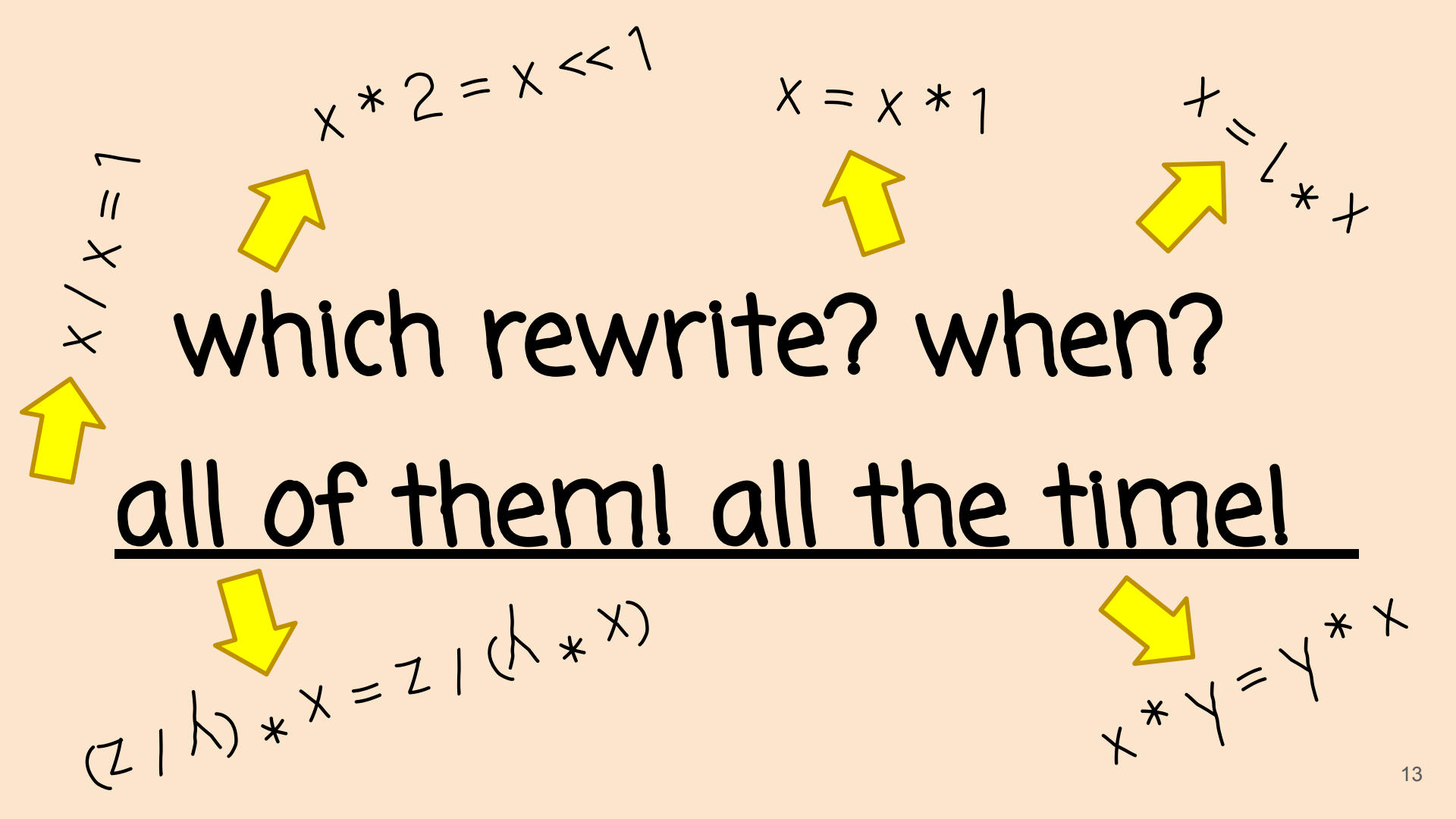
x * 1 = x

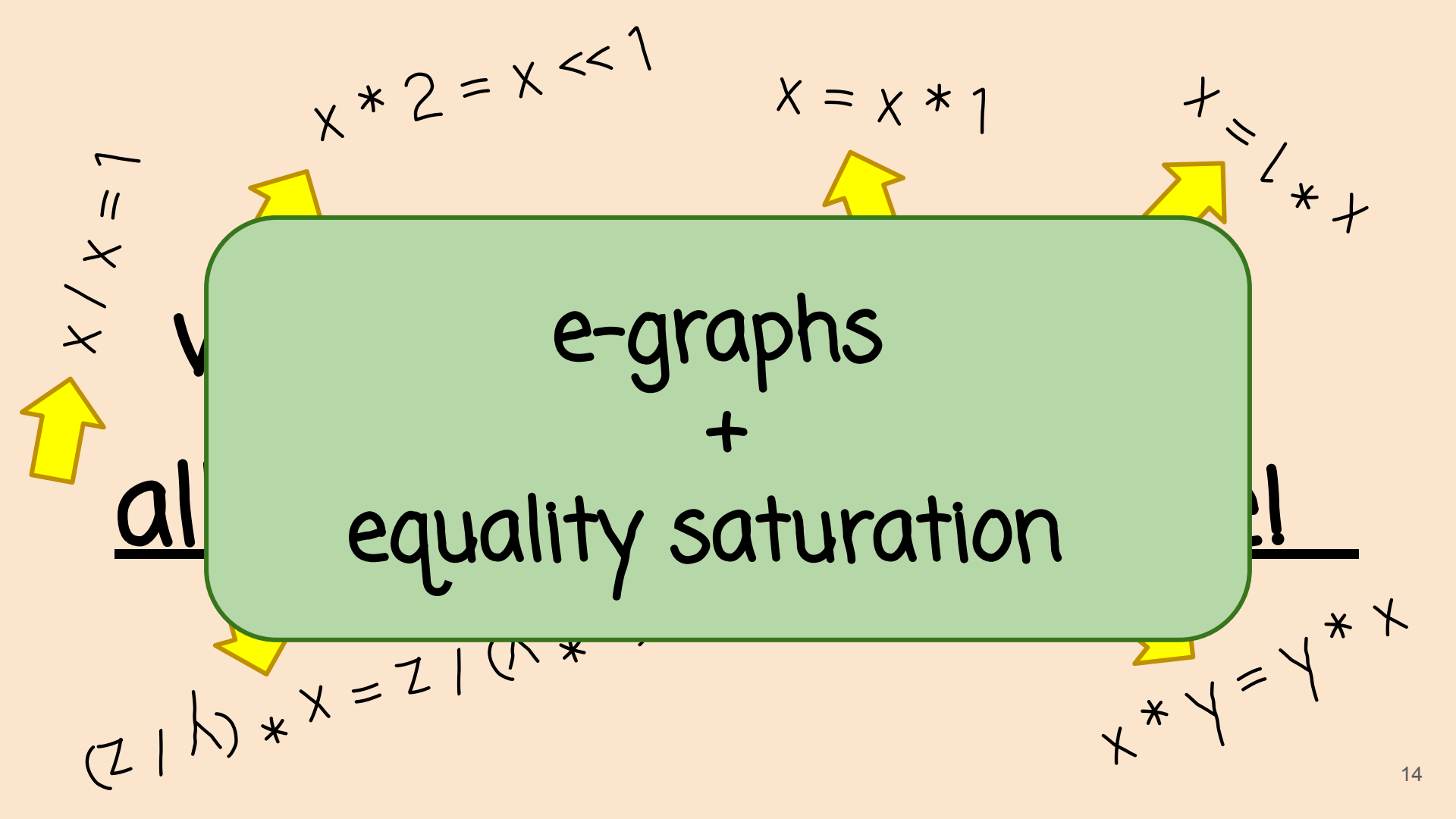
not so useful

x * 2 = x << 1

x * y = y * x

x = x * 1


x / x = 1
x * 2 = x << 1
x = x * 1
x = 1 * x
which rewrite? when?
all of them! all the time!
(x / y) * z = x / (y * z)
x * y = y * x


x * 2 = x << 1
x = x * 1
x = 1 * x
x / x = 1
e-graphs
+
equality saturation
x * y = y * x

talk in a slide

-  **background** : e-graphs and eqsat
- **egg**: fast, flexible eqsat

POPL 21★

- **intermission** : community & applications

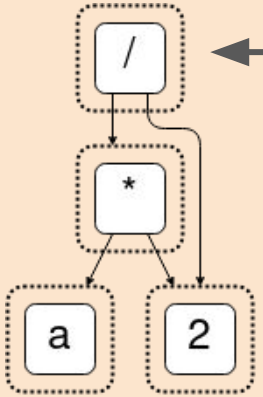
PLDI 20, MLSys 21, OOPLSA 22★, SIGGRAPH 22, FMCAD 22, POPL 23

- **current work** : connections to databa

POPL 22

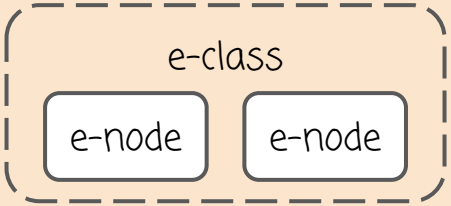
- **big picture**

e-graphs?

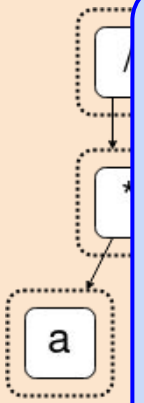


this e-class represents

(a * 2) / 2

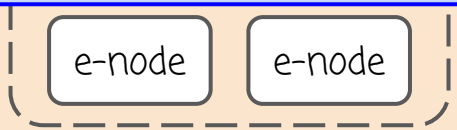


e-graphs?

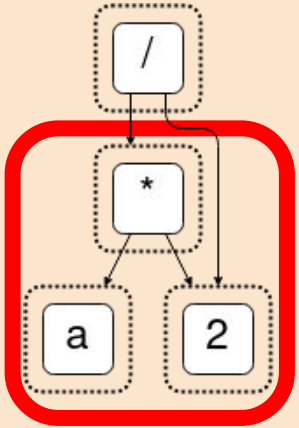


Knuth '76, Kozen '77, Downey '80, … , Nelson '80

- context-free grammars, finitely presented algebras, and/or graphs -> e-graphs
- designed for use inside theorem provers

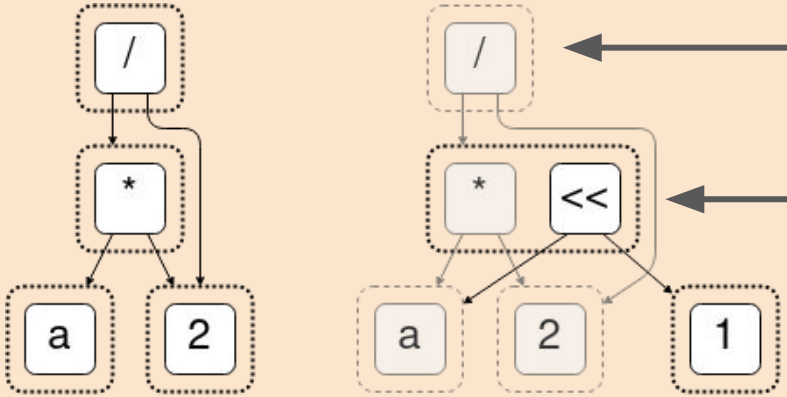


growing an e-graph



$x * 2 \rightarrow x << 1$

growing an e-graph

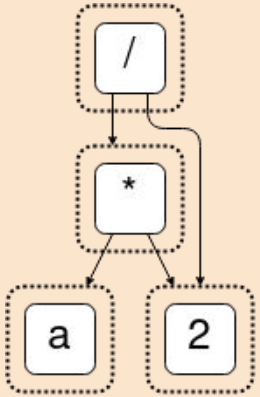


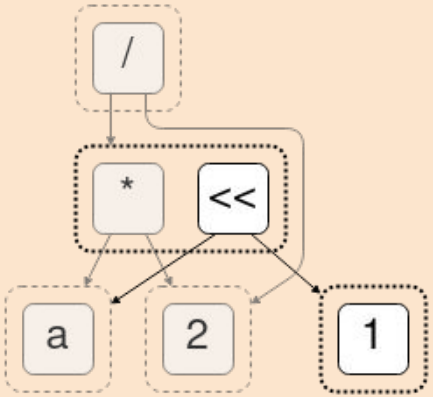
this e-class represents
(a * 2) / 2 **and** (a << 1) / 2

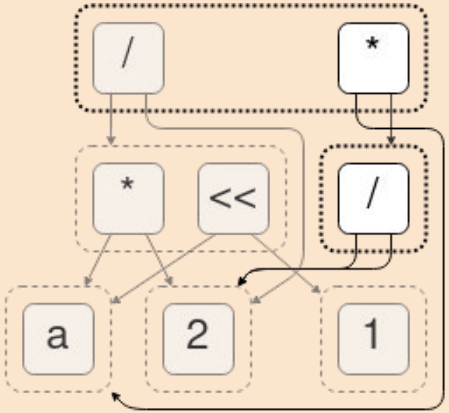
this e-class represents
(a * 2) **and** (a << 1)

x * 2 → x << 1

growing an e-graph





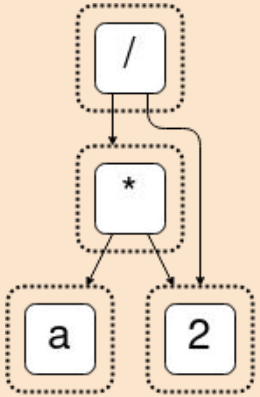


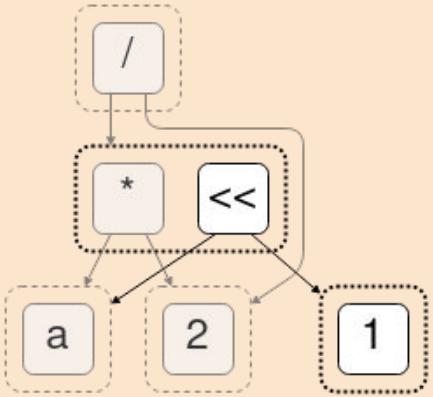
$x * 2 \rightarrow x << 1$

$(x * y) / z \rightarrow x * (y / z)$

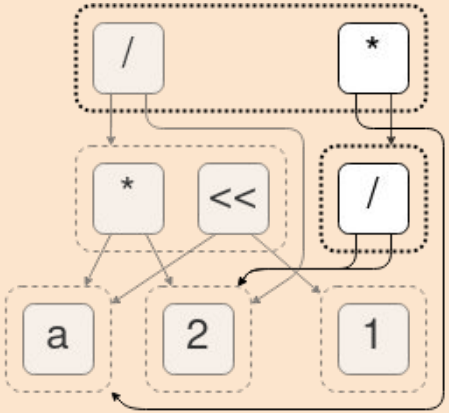
e-graphs are compact

a, a * 1,
a * 1 * 1, ...

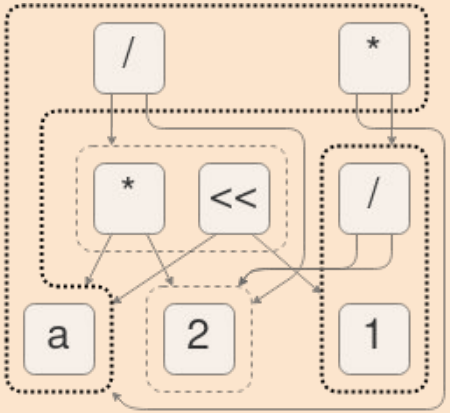




x * 2 → x << 1



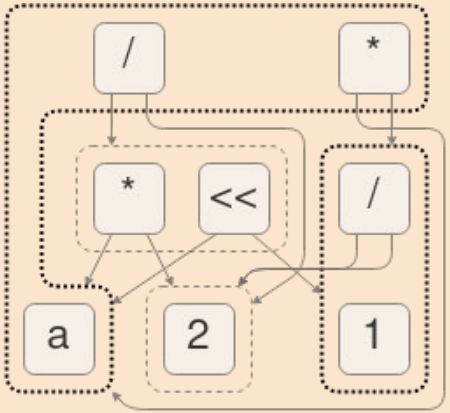
(x * y) / z → x * (y / z)



x / x → 1
x * 1 → x

saturation

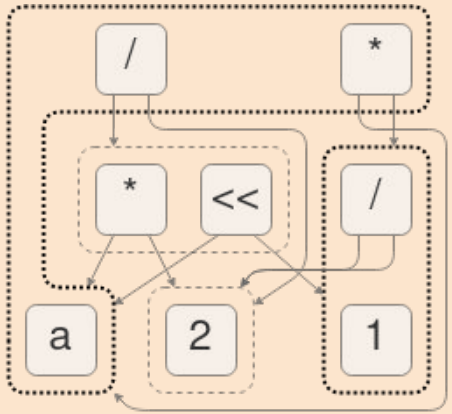
- ✓ $x * 2 \rightarrow x << 1$
- ✓ $(x * y) / z \rightarrow x * (y / z)$
- ✓ $x / x \rightarrow 1$
- ✓ $x * 1 \rightarrow x$



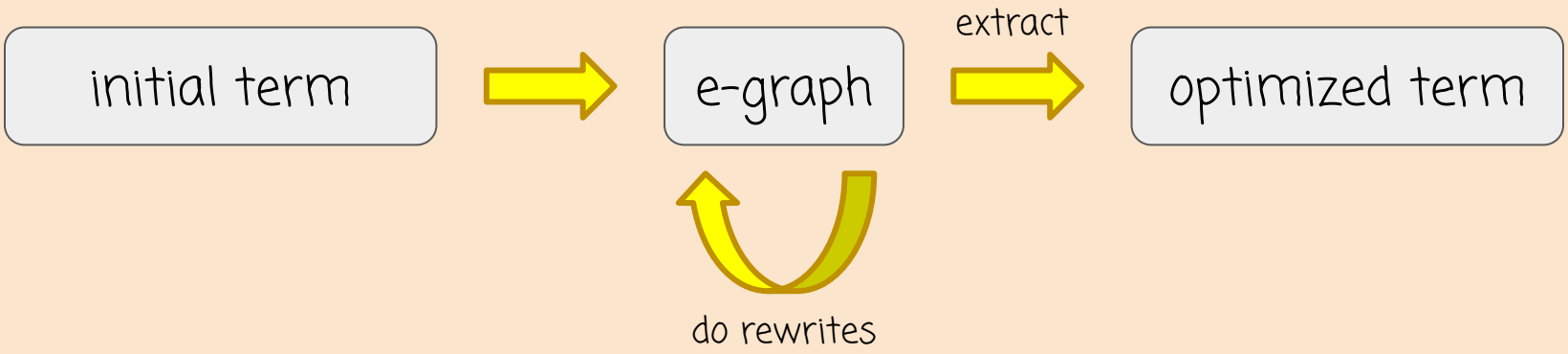
extraction

this e-class represents
(a * 2) / 2, a, a * 1, ...

pick the smallest
(cheapest) one



equality saturation



equality saturation

EqSat [Tate et al. 2009]

- PEG encoding: algebraic IR for loops
  - Introduced "theta" nodes for sequences
- specialized to Java/LLVM, ad-hoc optimizations

talk in a slide

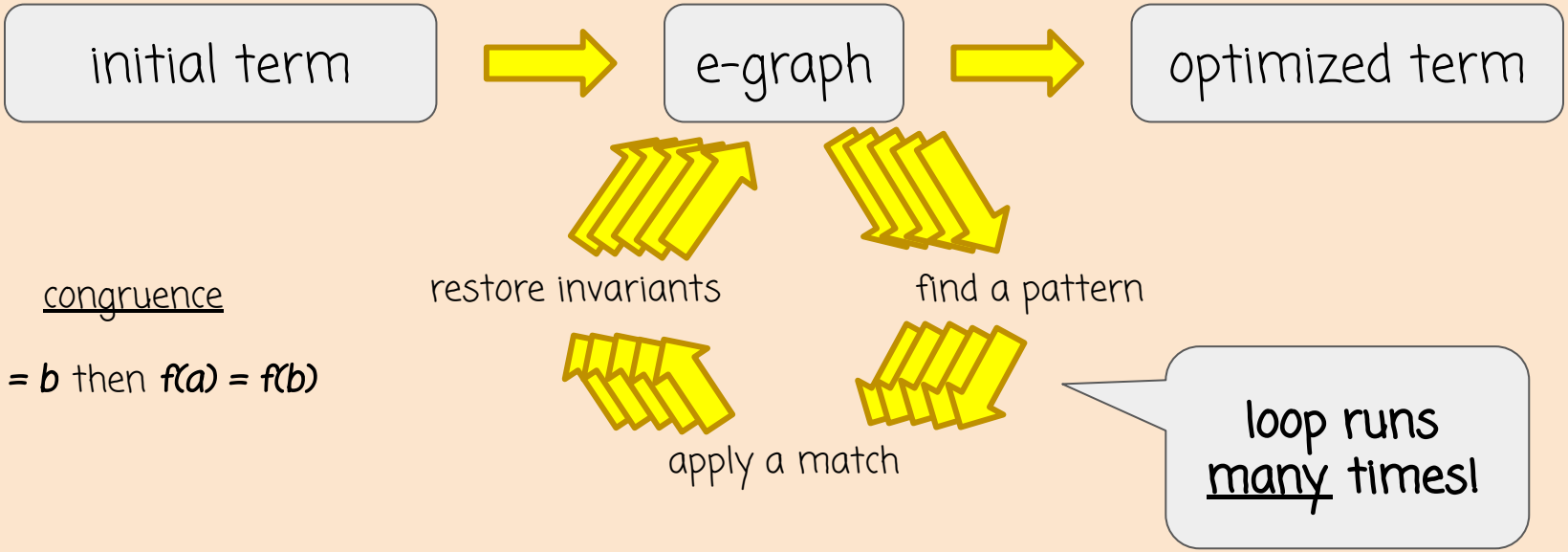
- **background** : e-graphs and eqsat
-  **egg**: fast, flexible eqsat

POPL 21★

- **intermission** : community & applications

- **current work** : connections to databases
- **going forward**

equality saturation

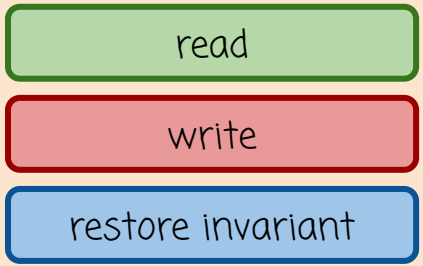


congruence

if $a = b$ then $f(a) = f(b)$

equality saturation

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

  while not egraph.is_saturated_or_timeout():
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        ec2 = egraph.add(rw.rhs.subst(subst))
        egraph.merge(ec, ec2)

  return egraph.extract_best()
```



egg's equality saturation

```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(ex
  while not egraph.is_satura
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        ec2 = egraph.add(rw.rhs.subst(subst))
        egraph.merge(ec
  return egraph.extract_best()
```

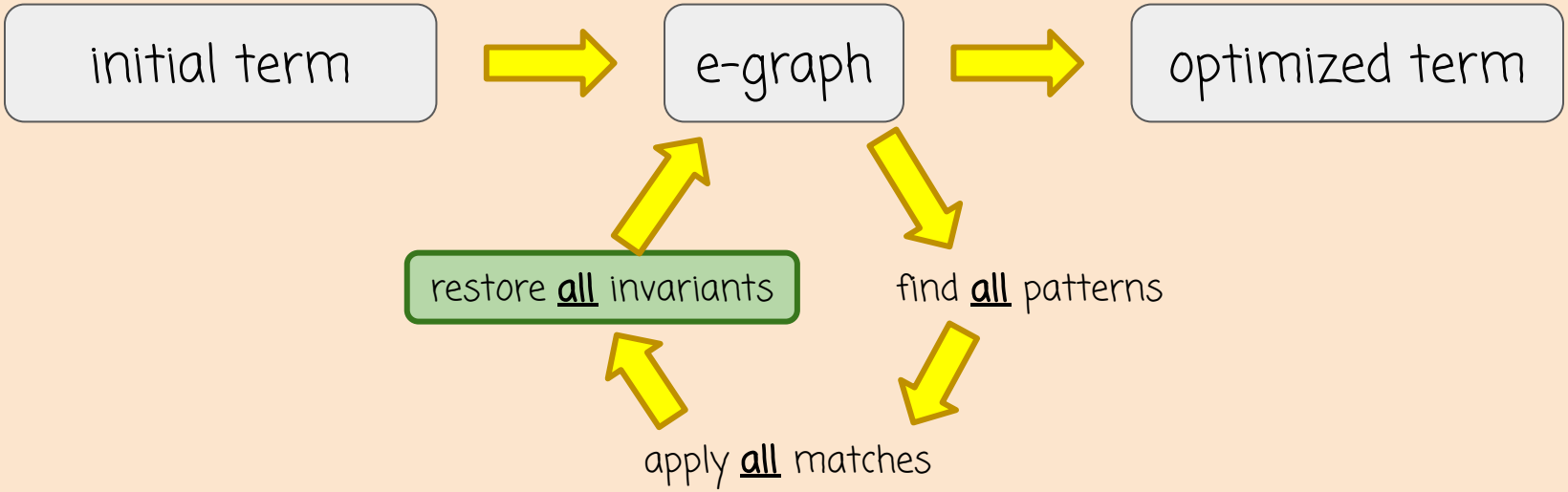
easy to parallelize

invariants broken

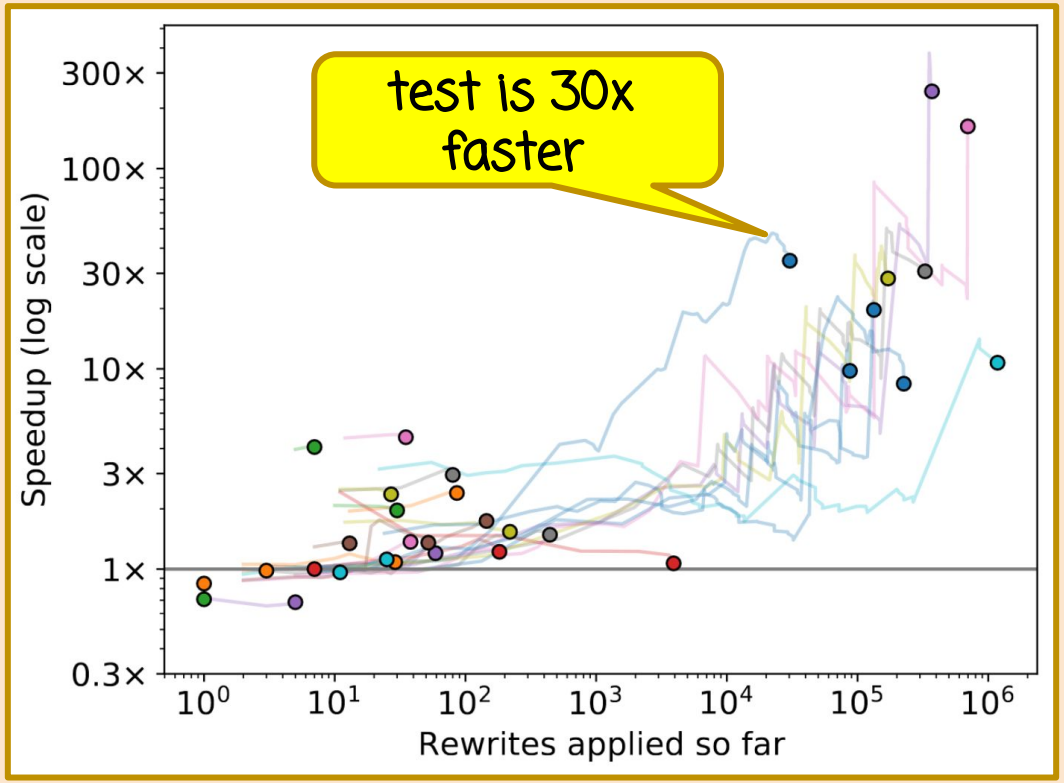
once per iteration

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

  while not egraph.is_saturated_or_timeout():
    matches = []
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        matches.append((rw, subst, ec))
    for (rw, subst, ec) in matches:
      ec2 = egraph.add(rw.rhs.subst(subst))
      egraph.merge(ec, ec2)
    egraph.rebuild()

  return egraph.extract_best()
```

deferred invariant maintenance



rebuilding is faster



fast and flexible

- there's more than *syntactic* rewriting
- sometimes, it's useful to consider *semantics*
  - 17 + 32 → 49, ...
- constant folding, nullability, tensor shape, non-zero, interval arithmetic, etc, ...

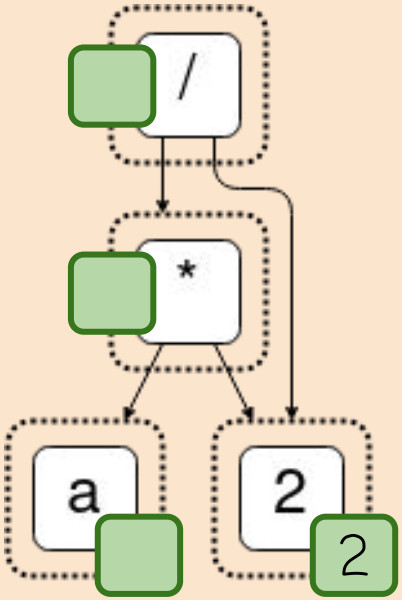
fast and flexible

analyses modulo equality

- uniform interface that works in many cases
- an understanding of analyses mean

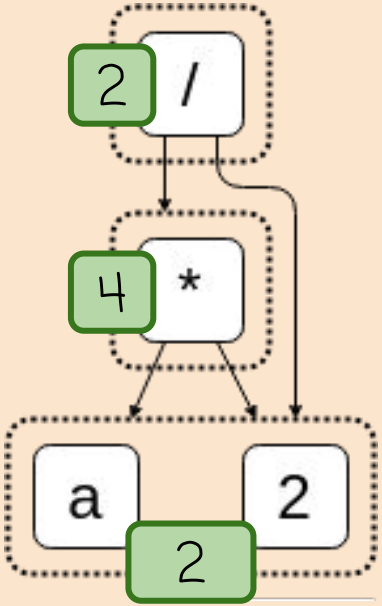
constant folding

- Option<Number> per eclass
- try to eval new e-nodes
- Option "or" on merge



constant folding

- Option<Number> per eclass
- try to eval new e-nodes
- Option "or" on merge
- it propagates up!



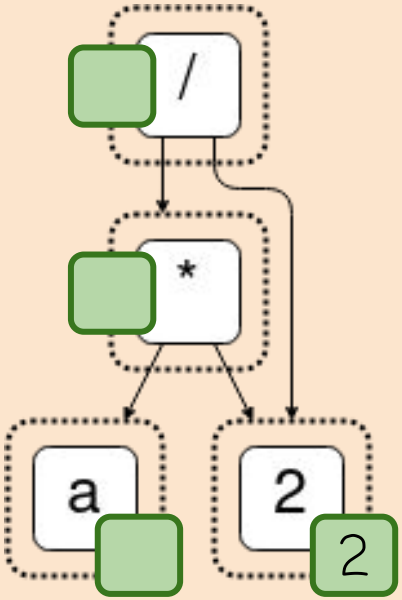
merge(a, 2)

e-class analysis

- 1 fact per e-class from a join-semilattice D
- make(n) → d_c
  - make a new analysis value for a new e-node
- join(d_{c1}, d_{c2}) → d_c
  - combine two analysis values
- modify(c) → c'
  - change the e-class (optionally)

constant folding

- D = Option<Number>
- make = eval
- join = option "or"
- modify = add the constant



e-class analysis uses

- lift program analyses to e-graphs
- conditional & dynamic rewrites
  - x / x = 1 iff x != 0
- can express other e-graph "hacks"
  - debugging, pruning, on-the-fly extraction

e-class analysis invariant

for each e-class

fixed point

$$\forall c \in G. \quad d_c = \bigvee_{n \in c} \text{make}(n) \quad \text{and} \quad \text{modify}(c) = c$$

Analysis data is
LUB
(lattice properties)

egg: fast & easy e-graphs

- Rust library for generic e-graphs and eqsat
- packaged and documented: https://docs.rs/egg
- tutorials, industrial and academic users

POPL 21★

talk in a slide

- **background** : e-graphs and eqsat
- **egg**: fast, flexible eqsat

intermission : community & applications

PLDI 20, MLSys 21, OOPLSA 22★, SIGGRAPH 22, FMCAD 22, POPL 23

- **current work** : connections to databases
- **big picture**

some egg projects

- ARITH 2021 — Herbie: floating point — 3000x faster
- VLDB 2020 — SPORES: linear algebra — 1.2-5x better
- **MLSys 2021** — Tensat: ML compute graphs — 23% better, 48x faster
- SIGMOD 2022 — Infer loop invs. for query opt — up to 10,000x faster
- **OOPSLA 2021*** — Ruler: Theory synthesis — better rulesets, 25x faster
- ARITH 2022 — Arithmetic datapath opt — up to 71% smaller
- Intel, AWS, Fastly, Certora, ...

detour: intervals

x in [0, 1]
y in [1, 2]

x + y in [1, 3]

detour: intervals

x in [0, 1]
y in [1, 2]

$1 - 2y / (x + y)$ in $[-3, 1/3]$

$= (x - y) / (x + y)$ in $[-2, 0]$

$= 2x / (x + y) - 1$ in $[-1, 1]$

intervals modulo equality

x in [0, 1]
y in [1, 2]

$$1 - 2y / (x + y) \quad \text{in} \quad [-3, 1/3]$$

$$= (x - y) / (x + y) \quad \text{in} \quad [-2, 0]$$

$$= 2x / (x + y) - 1 \quad \text{in} \quad [-1, 1]$$

$\Big\}$ **[-1, 0]**

intervals modulo equality

x in
y in

Expression	Initial	Improved	Width Change
$x^2 - 2x + 1$	[-2,3]	[0,1]	-80%
$x(2 - xy) - \frac{1}{y}$	[-5,1.5]	[-4.5, 0]	-31%
$\sqrt{x+1} - \sqrt{x}$	$[0, \sqrt{2}]$	$\left[\frac{1}{\sqrt{2}+\sqrt{3}}, \frac{1}{1+\sqrt{2}}\right]$	-93%
$\frac{x}{x+y}$	$\left[\frac{1}{4}, 1\right]$	$\left[\frac{1}{4}, \frac{3}{4}\right]$	-33%

1/3]

0]

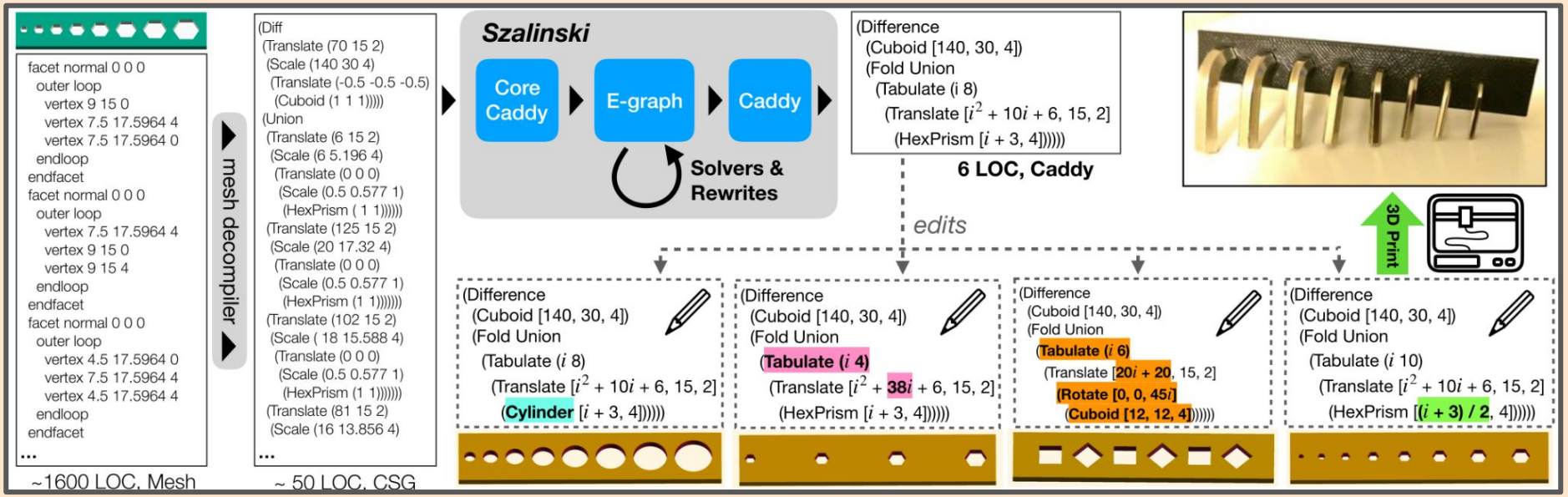
0]

intel

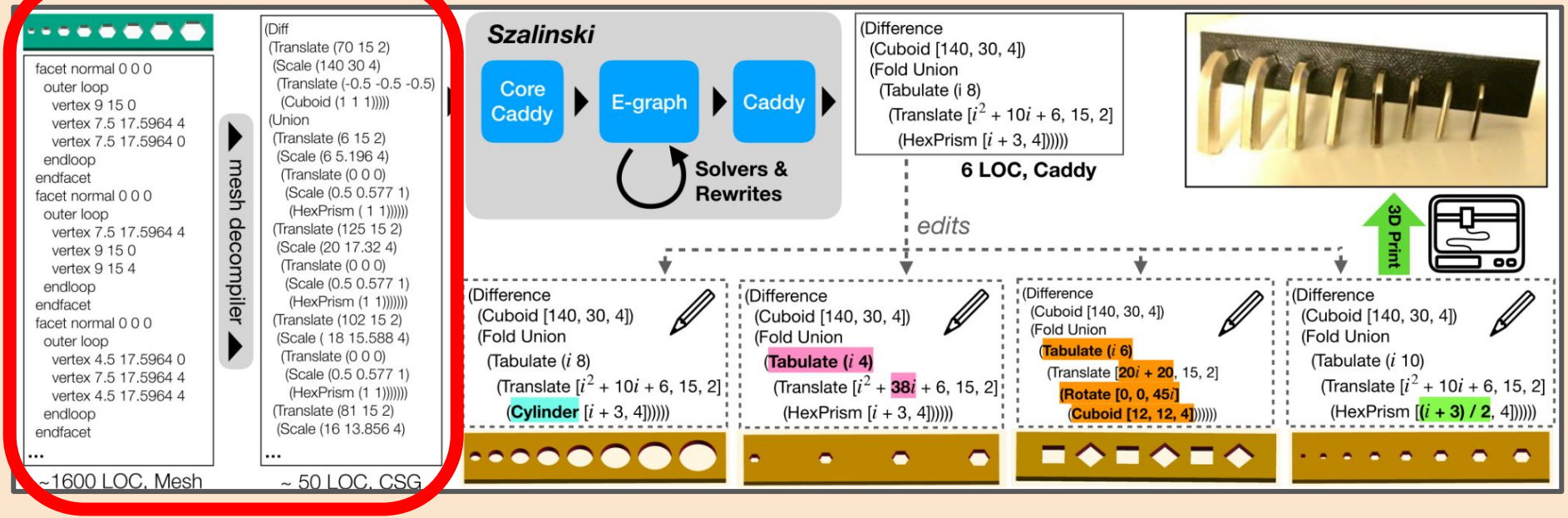
Sam Coward et. al, EGRAPHS 22


EXIT
PLDI
PLDI
adidas

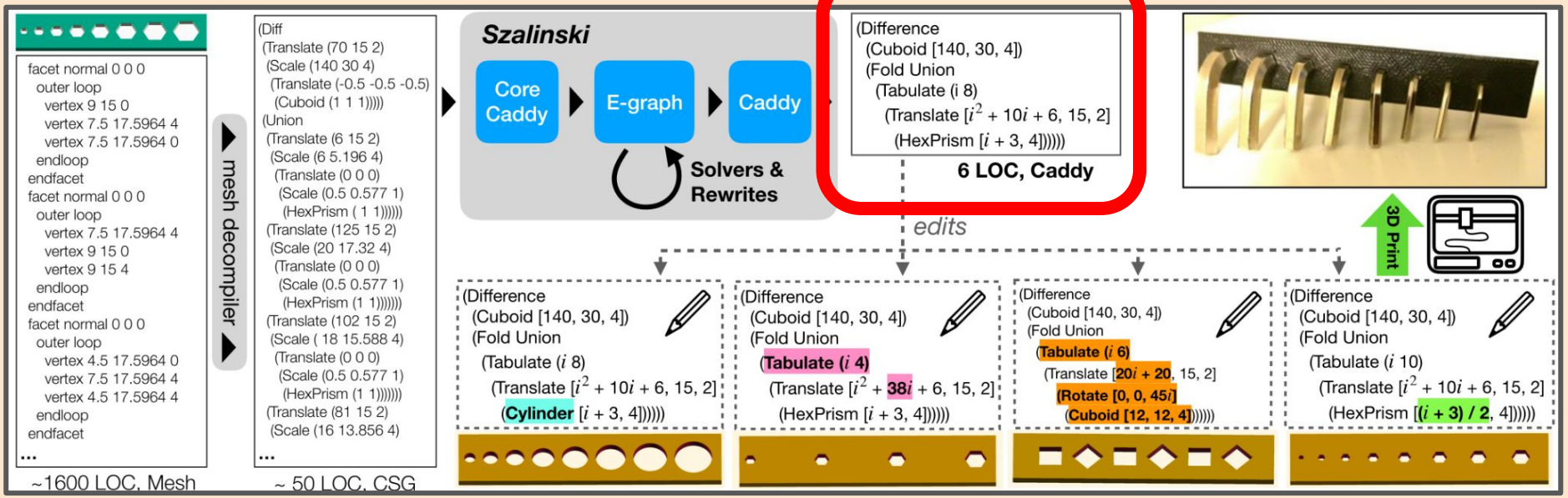
Szalinski


facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~1600 LOC, Mesh
mesh decompiler
(Diff
(Translate (70 15 2)
(Scale (140 30 4)
(Translate (-0.5 -0.5 -0.5)
(Cuboid (1 1 1)))))
(Union
(Translate (6 15 2)
(Scale (6 5.196 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1))))))
(Translate (125 15 2)
(Scale (20 17.32 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (102 15 2)
(Scale (18 15.588 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (81 15 2)
(Scale (16 13.856 4)
...
~ 50 LOC, CSG
Szalinski
Core Caddy
E-graph
Caddy
Solvers & Rewrites
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
6 LOC, Caddy
edits
3D Print
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(Cylinder [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 4)
(Translate [i^2 + 38i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 6)
(Translate [20i + 20, 15, 2]
(Rotate [0, 0, 45i]
(Cuboid [12, 12, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 10)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [(i + 3) / 2, 4])))))

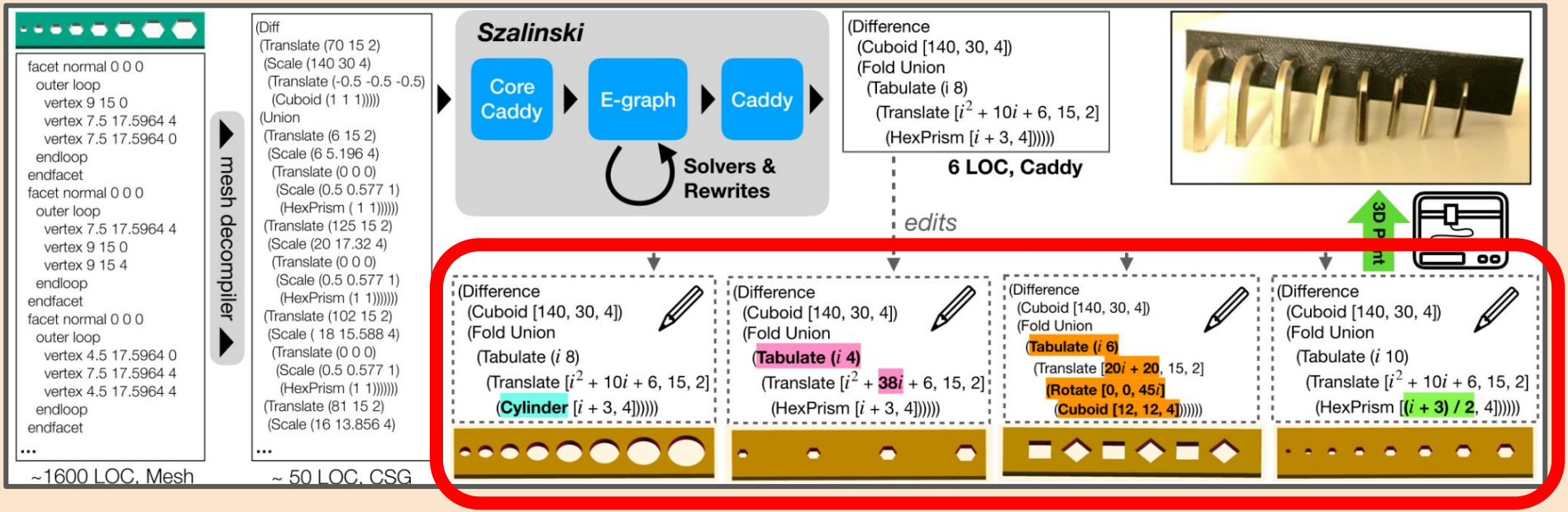
Szalinski


facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~1600 LOC, Mesh
mesh decompiler
(Diff
(Translate (70 15 2)
(Scale (140 30 4)
(Translate (-0.5 -0.5 -0.5)
(Cuboid (1 1 1)))))
(Union
(Translate (6 15 2)
(Scale (6 5.196 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1))))))
(Translate (125 15 2)
(Scale (20 17.32 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (102 15 2)
(Scale (18 15.588 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (81 15 2)
(Scale (16 13.856 4)
...
~ 50 LOC, CSG
Szalinski
Core Caddy
E-graph
Caddy
Solvers & Rewrites
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
6 LOC, Caddy
edits
3D Print
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(Cylinder [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 4)
(Translate [i^2 + 38i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 6)
(Translate [20i + 20, 15, 2]
(Rotate [0, 0, 45i]
(Cuboid [12, 12, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 10)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [(i + 3) / 2, 4])))))

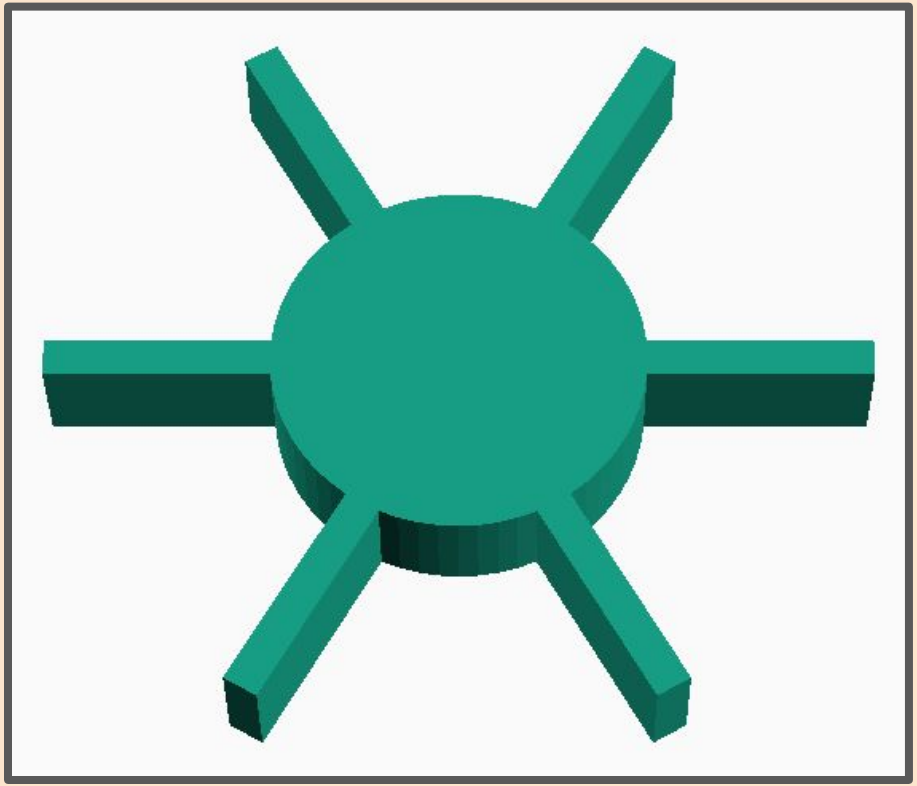
Szalinski


facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~1600 LOC, Mesh
mesh decompiler
(Diff
(Translate (70 15 2)
(Scale (140 30 4)
(Translate (-0.5 -0.5 -0.5)
(Cuboid (1 1 1)))))
(Union
(Translate (6 15 2)
(Scale (6 5.196 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1))))))
(Translate (125 15 2)
(Scale (20 17.32 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (102 15 2)
(Scale (18 15.588 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (81 15 2)
(Scale (16 13.856 4)
...
~ 50 LOC, CSG
Szalinski
Core Caddy
E-graph
Caddy
Solvers & Rewrites
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
6 LOC, Caddy
edits
3D Print
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(Cylinder [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 4)
(Translate [i^2 + 38i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 6)
(Translate [20i + 20, 15, 2]
(Rotate [0, 0, 45i]
(Cuboid [12, 12, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 10)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [(i + 3) / 2, 4])))))

Szalinski

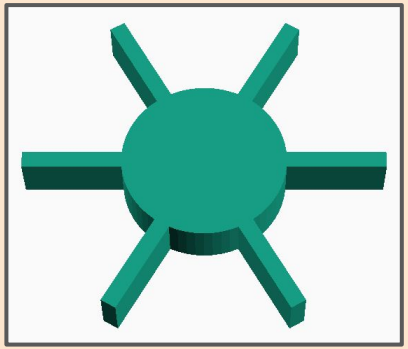

facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~1600 LOC, Mesh
mesh decompiler
(Diff
(Translate (70 15 2)
(Scale (140 30 4)
(Translate (-0.5 -0.5 -0.5)
(Cuboid (1 1 1)))))
(Union
(Translate (6 15 2)
(Scale (6 5.196 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1))))))
(Translate (125 15 2)
(Scale (20 17.32 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (102 15 2)
(Scale (18 15.588 4)
(Translate (0 0 0)
(Scale (0.5 0.577 1)
(HexPrism (1 1)))))))
(Translate (81 15 2)
(Scale (16 13.856 4)
...
~ 50 LOC, CSG
Szalinski
Core Caddy
E-graph
Caddy
Solvers & Rewrites
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
6 LOC, Caddy
edits
3D Print
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 8)
(Translate [i^2 + 10i + 6, 15, 2]
(Cylinder [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 4)
(Translate [i^2 + 38i + 6, 15, 2]
(HexPrism [i + 3, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 6)
(Translate [20i + 20, 15, 2]
(Rotate [0, 0, 45i]
(Cuboid [12, 12, 4])))))
(Difference
(Cuboid [140, 30, 4])
(Fold Union
(Tabulate (i 10)
(Translate [i^2 + 10i + 6, 15, 2]
(HexPrism [(i + 3) / 2, 4])))))

what we want



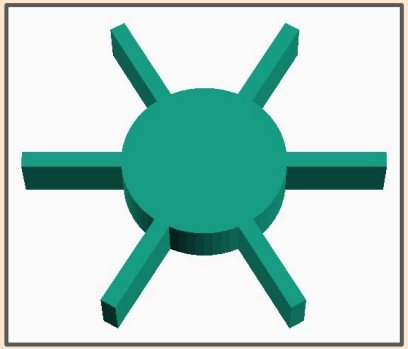
```
(Union
 (Cylinder [1, 5, 5])
 (Fold Union
  (Tabulate (i 6)
   (Rotate [0, 0, 60 * i]
    (Translate [1, -0.5, 0]
    (Cuboid [10, 1, 1]))))))
```

easy mode



```
(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])))))
```

hard mode

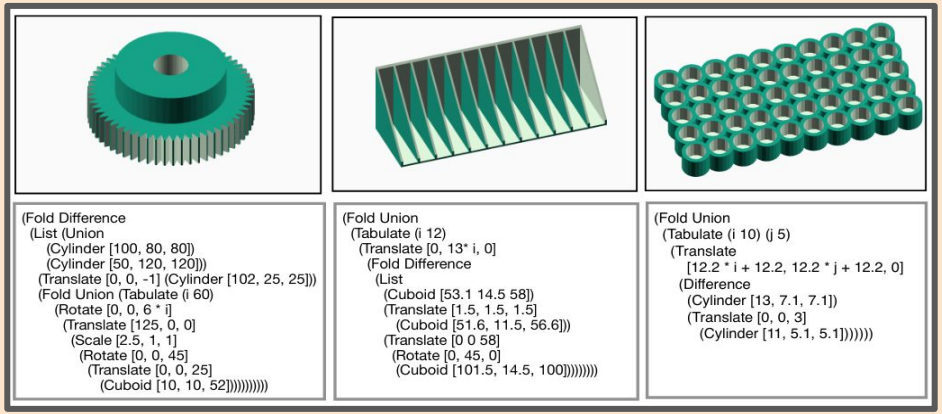


```
(Union
  (Rotate [0,0,120]  (Translate [1,-0.5,0]   (Cuboid [10,1,1])))
  (Scale  [10,1,1]   (Translate [0.1,-0.5,1] (Cuboid [1,1,1])))
  (Rotate [0,0,300]  (Translate [1,-0.5,0]   (Cuboid [10,1,1])))
  (Scale  [5,5,1]   (Cylinder  [1,1]))
  (Translate [-1,0.5,0] (Scale      [-1,-1,1] (Cuboid [10,1,1])))
  (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]))))
```

x = (Rotate [0 0 0] x)

Szalinski

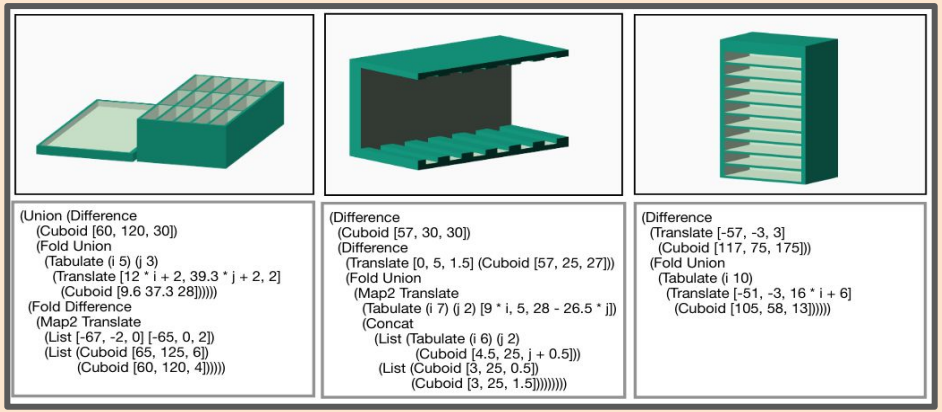
PLDI 20



```
(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 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])))))))))
```

```
(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])))))))
```



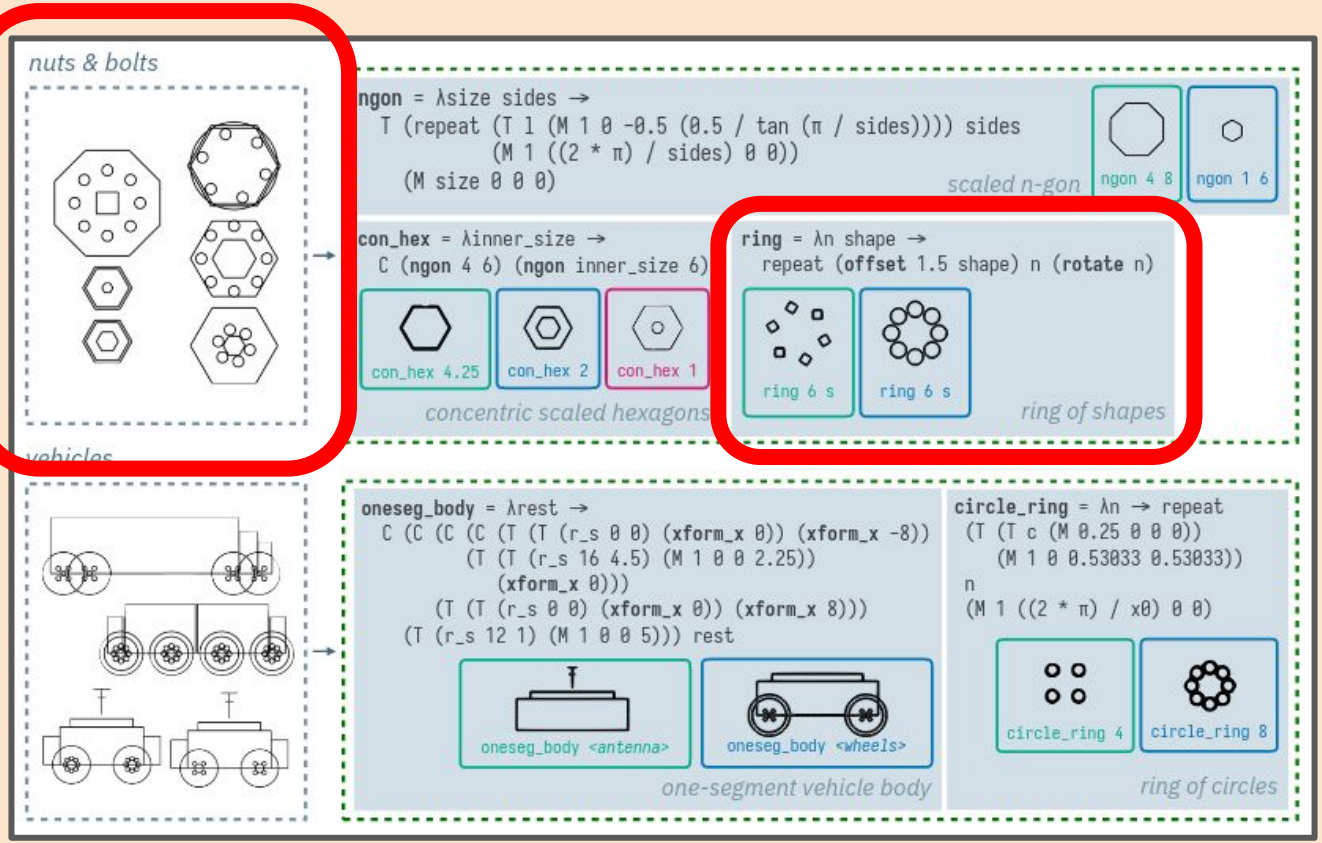
```
(Union (Difference
    (Cuboid [60, 120, 30])
    (Fold Union
      (Tabulate (i 5) (j 3)
        (Translate [12 * i + 2, 39.3 * j + 2, 2]
          (Cuboid [9.6 37.3 28])))))
  (Fold Difference
    (Map2 Translate
      (List [-67, -2, 0] [-65, 0, 2])
      (List (Cuboid [65, 125, 6])
            (Cuboid [60, 120, 4])))))
```

```
(Difference
  (Cuboid [57, 30, 30])
  (Difference
    (Translate [0, 5, 1.5] (Cuboid [57, 25, 27]))
    (Fold Union
      (Map2 Translate
        (Tabulate (i 7) (j 2) [9 * i, 5, 28 - 26.5 * j])
        (Concat
          (List (Tabulate (i 6) (j 2)
                  (Cuboid [4.5, 25, j + 0.5]))
                (List (Cuboid [3, 25, 0.5])
                      (Cuboid [3, 25, 1.5])))))))))
```

```
(Difference
  (Translate [-57, -3, 3]
    (Cuboid [117, 75, 175]))
  (Fold Union
    (Tabulate (i 10)
      (Translate [-51, -3, 16 * i + 6]
        (Cuboid [105, 58, 13])))))
```

Babble

POPL 23



talk in a slide

- **background** : e-graphs and eqsat
- **egg**: fast, flexible eqsat

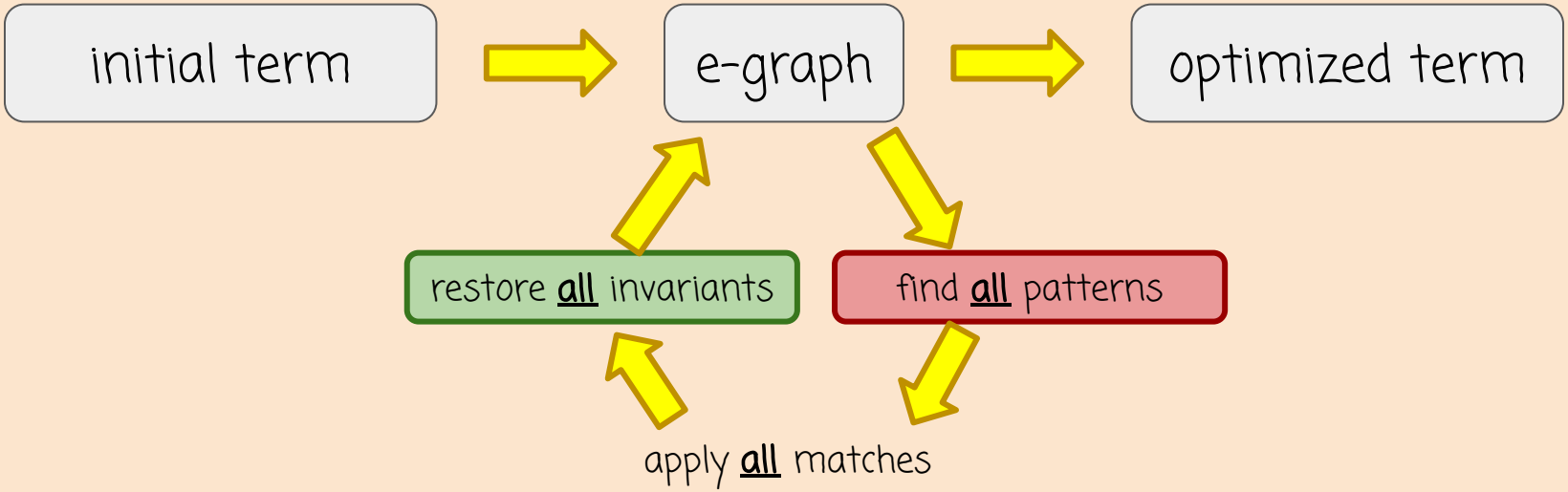
- **intermission** : community & applications

- **current work** : connections to databases

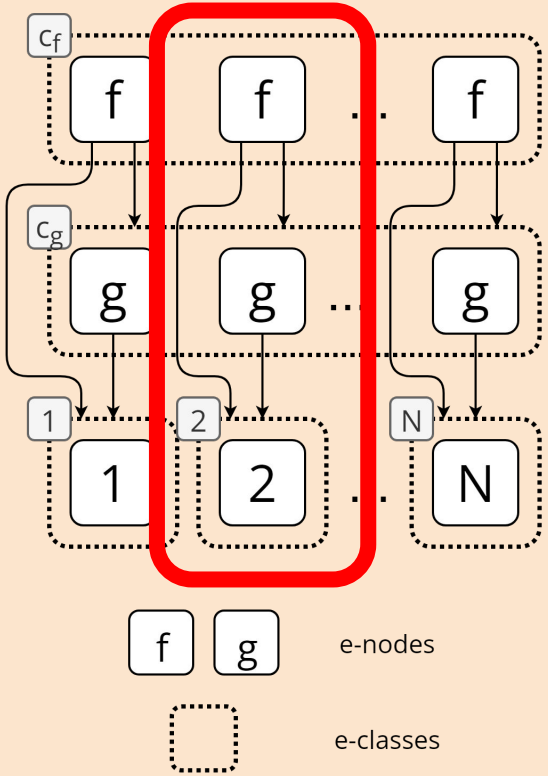
POPL 22

- big picture

equality saturation

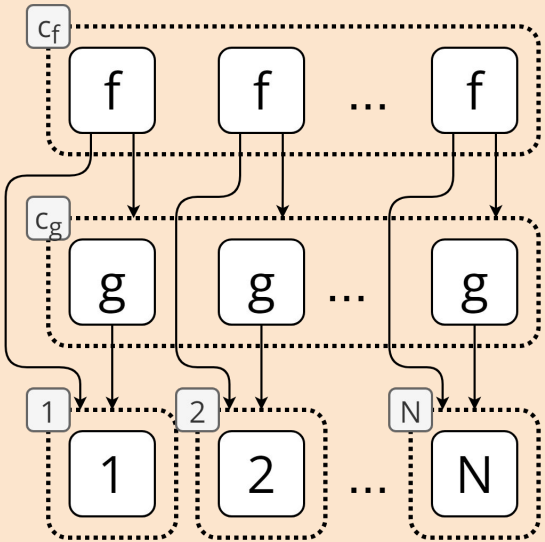


e-matching



pattern	substs	terms
f(a, g(a))	$\{a \mapsto 1\}$	*f(1, g(1))*
	$\{a \mapsto 2\}$	*f(2, g(2))*
	...	...
	$\{a \mapsto N\}$	*f(N, g(N))*

e-matching



pattern

f(a, g(a))

```
for e-class c in e-graph E:
  for f-node n1 in c:
    subst = {root ↦ c, α ↦ n1.child1}
    for g-node n2 in n1.child2:
      if subst[α] = n2.child1:
        yield subst
```

N^2 time, but only N matches

e-matching

- existing impls are **backtracking** based & **complex**
- doesn't help with **equality constraints**
- no **data complexity** results
  - NP-hard in pattern size... e-graph size??

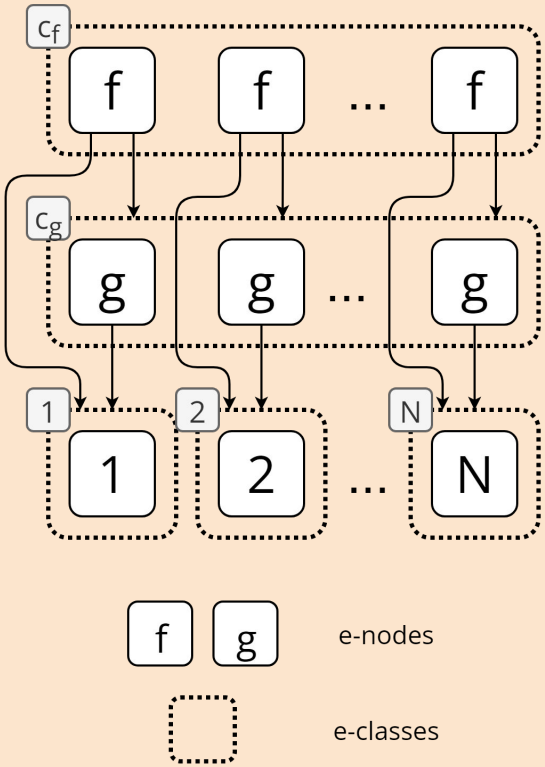
e-matching

Efficient E-matching [de Moura & Bjørner 2007]

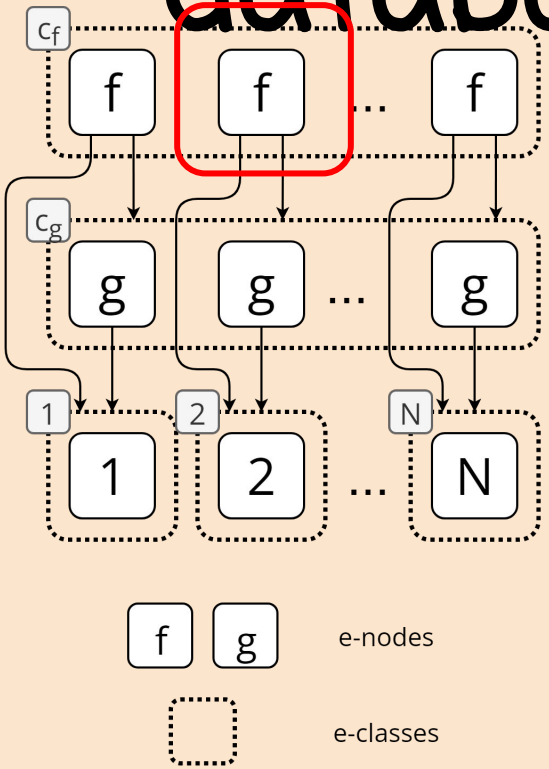
- state-of-the art in Z3
- compile to VM, many complex optimizations
- same as egg (for now...)

- NP-hard in pattern size... e-graph size??

e-graph



e-graph = database

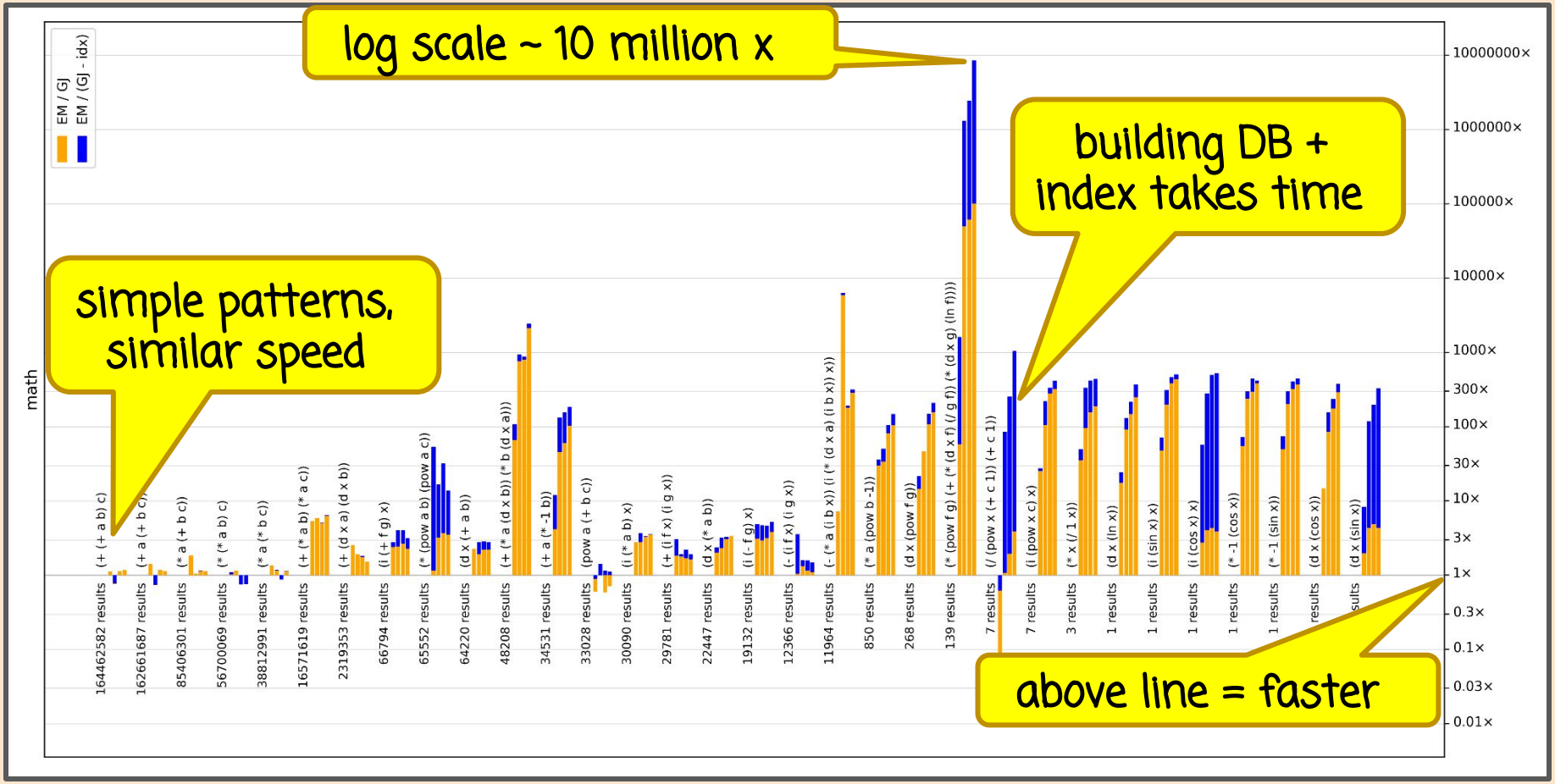


R_f

id	arg_1	arg_2
c_f	*1*	c_g
c_f	*2*	c_g
...	...	...
c_f	*N*	c_g

e-matching = db query

pattern	conjunctive query
$f(a, g(a))$	$Q(root, a) \leftarrow R_f(root, a, x), R_g(x, a)$


log scale ~ 10 million x
building DB + index takes time
simple patterns, similar speed
above line = faster
EM / GJ
EM / (GJ - idx)
math
10000000×
1000000×
100000×
10000×
1000×
300×
100×
30×
10×
3×
1×
0.3×
0.1×
0.03×
0.01×
164462582 results
162661687 results
85406301 results
56700069 results
38812991 results
16571619 results
2319353 results
66794 results
65552 results
64220 results
48208 results
34531 results
33028 results
30090 results
29781 results
22447 results
19132 results
12366 results
11964 results
850 results
268 results
139 results
7 results
7 results
3 results
1 results
1 results
1 results
1 results
1 results
results
sults

data complexity results

THEOREM 9. *Relational e-matching is worst-case optimal; that is, fix a pattern p, let $M(p, E)$ be the set of substitutions yielded by e-matching on an e-graph E with N e-nodes, relational e-matching runs in time $O(\max_E(|M(p, E)|))$.*

w.r.t "worst-case" e-graph of size N

THEOREM 10. *Fix an e-graph E with N e-nodes that compiles to a database I, and a fix pattern p that compiles to conjunctive query $Q(\overline{X}) \leftarrow R_1(\overline{X_1}), \ldots, R_m(\overline{X_m})$. Relational e-matching p on E runs in time $O\left(\sqrt{|Q(I)| \times \Pi_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$.*

w.r.t a specific e-graph

data complexity results

Worst-Case Optimal Join [Ngo, et. al, 2014 & 2018]

- beautiful, recent DB result
- efficient join algo for cyclic queries
- Generic Join (go learn it!)

w.r.t a specific e-graph

db freebies

- simple implementation
  - separate optimization pass
- multipatterns
  - a x b = split(1, (a ++ c) x b)
  - a x c = split(2, (a ++ c) x b)
- incrementality via semi-naive

talk in a slide

- **background** : e-graphs and eqsat
- **egg**: fast, flexible eqsat

- **intermission** : community & applications

- **current work** : connections to databases
-  big picture

big picture

- PL tools need **equational reasoning**
- e-graphs let you work with **multiplicity**
- egg is a sweet spot: fast, flexible, usable
- make e-graphs **the tool** for equational reasoning

what's next?

- fundamental advances
  - e-graphs + datalog, contextual equality, ...
- big applications
  - JIT, theorem proving, query optimization
- building community
  - NSF Grant, EGRAPHS 2023, courses

thank you!

mwillsey.com

egraphs-good.github.io

EGRAPHS 2023 @ PLDI
Deadline April 5