Equality Saturation & egg

Zachary Tatlock, University of Washington

2022-10-08 @ ETH Workshop on Dependable and Secure Software Systems


Max Willsey


Chandra Nandi


Oliver Flatt


Remy Wang


Yihong Zhang


Brett Saiki


Sam Coward


Amy Zhu


Adam Anderson


Anjali Pal


Philip Zucker


Pavel Panchekha


Adriana Schulz


Dan Grossman


Zachary Tatlock

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

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

REWRITE!

```
(a * 2) / 2 ⇒ a
```

REWRITE!

Useful

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

      x / x = 1

      x * 1 = x
```

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

REWRITE!

Useful

```
(x * y) / z = x * (y / z)
x / x = 1
x * 1 = x
```

Less Useful

```
x * 2 = x << 1
x * y = y * x
x = x * 1
```

```
(a * 2) / 2
```

"happy path"

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

x / x = 1

x * 1 = x
```

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

"happy path"

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

$$x / x = 1$$

$$x * 1 = x$$

```
(a * 2) / 2 ⇒ a * (2 / 2) ⇒ a * 1
```

"happy path"

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

x / x = 1

x * 1 = x
```

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

“happy path”

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

$$x / x = 1$$

$$x * 1 = x$$

```
(a * 2) / 2 ⇒ a * (2 / 2) ⇒ a * 1 ⇒ a
```

“happy path”

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

       x / x = 1

       x * 1 = x
```

✅

```
(a * 2) / 2
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

`(a * 2) / 2 ⇒ (a << 1) / 2` ❌ order

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```

❌ order

```
(a * 2) / 2
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

(a * 2) / 2 ⇒ (a << 1) / 2 ❌ order

(a * 2) / 2 ⇒ (2 * a) / 2

Pitfalls

x * 2 = x << 1

x * y = y * x

x = x * 1

```
(a * 2) / 2 ⇒ (a << 1) / 2
```
❌ order

```
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```
❌ order

```
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2
```
❌

diverge

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```
❌ order

```
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2
```
❌

diverge

```
a
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```
❌ order

```
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2
```
❌

diverge

```
a ⇒ a * 1
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

```
(a * 2) / 2 ⇒ (a << 1) / 2
```
❌ order

```
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2
```
❌

diverge

```
a ⇒ a * 1 ⇒ a * 1 * 1
```

Pitfalls

```
x * 2 = x << 1
x * y = y * x
    x = x * 1
```

(a * 2) / 2 ⇒ (a << 1) / 2 ❌ order

(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2 ❌

diverge

a ⇒ a * 1 ⇒ a * 1 * 1 ⇒ ... ❌ infinite size

Pitfalls

x * 2 = x << 1

x * y = y * x

x = x * 1

`(a * 2) / 2 ⇒ (a << 1) / 2` ❌ order

`(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2` ❌

diverge

`a ⇒ a * 1 ⇒ a * 1 * 1 ⇒ ...` ❌ infinite size

Critical for other inputs!

Pitfalls

`x * 2 = x << 1`

`x * y = y * x`

`x = x * 1`

`(a * 2) / 2 ⇒ a`

Which rewrite? When?

Useful	Less Useful
`(x * y) / z = x * (y / z)`	`x * 2 = x << 1`
`x / x = 1`	`x * y = y * x`
`x * 1 = x`	`x = x * 1`

`(a * 2) / 2 ⇒ a`

Which rewrite? When?

Equality Saturation

Try applying all the rules in every order!

`(a * 2) / 2` ⇒ `a`

Which rewrite? When?

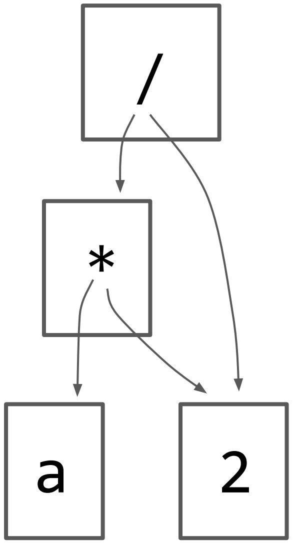
Equality Saturation

Try applying **all** the rules in **every** order?!

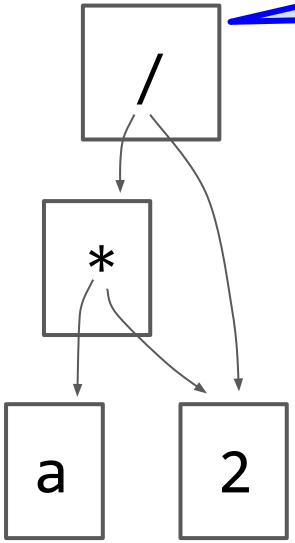
E-graphs

- Data structure from Greg Nelson's PhD thesis (1980)
- Used for congruence closure (Downey, Sethi, Tarjan 1980)
  - Intuition: union-find (Tarjan 1975) but function-aware
- Key for equality and uninterpreted funcs (EUF) theory in SMT
  - Intuition: the "glue" that connects other theories to SAT
- Historically: "baked in" to SMT solvers, no general libraries 😐

E-graphs

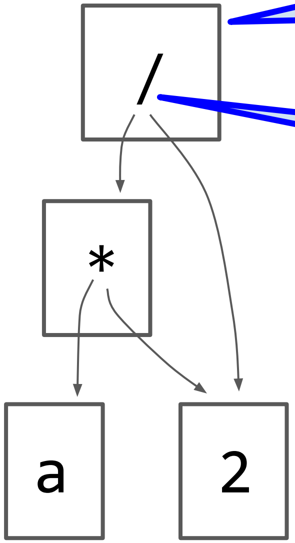


E-graphs



e-classes contain e-nodes (ops)

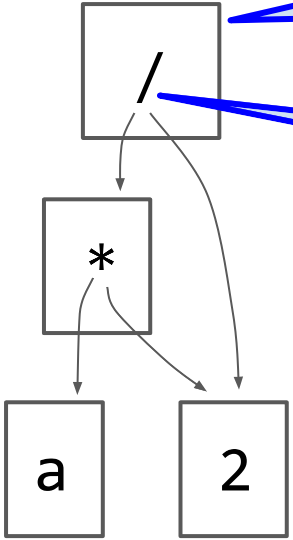
E-graphs



e-classes contain e-nodes (ops)

e-nodes' arguments are e-classes!

E-graphs

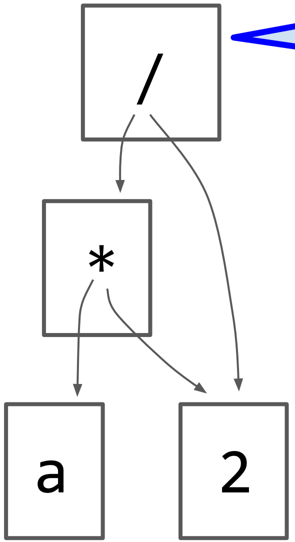


e-classes contain e-nodes (ops)

e-nodes' arguments are e-classes!

e-graphs maximize sharing
(no copies of same e-node)

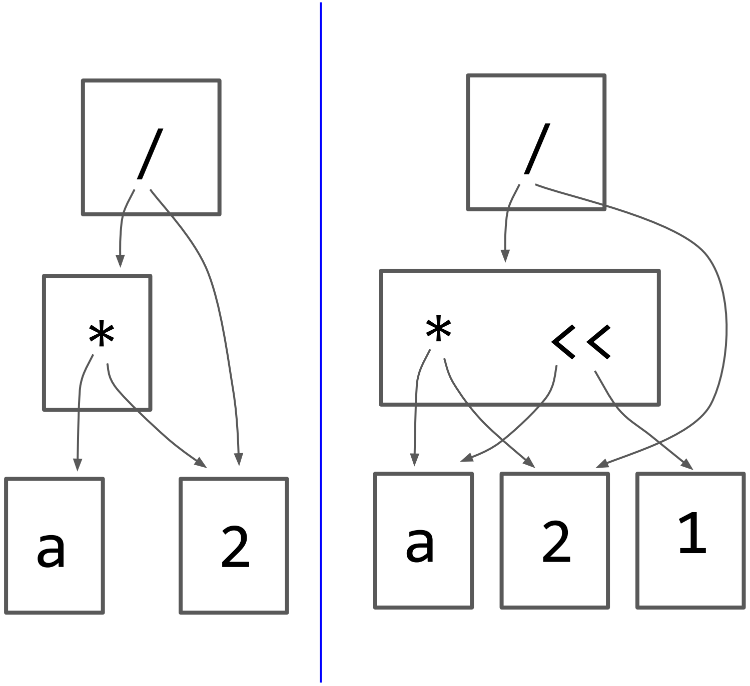
E-graphs



This e-classes *represents*

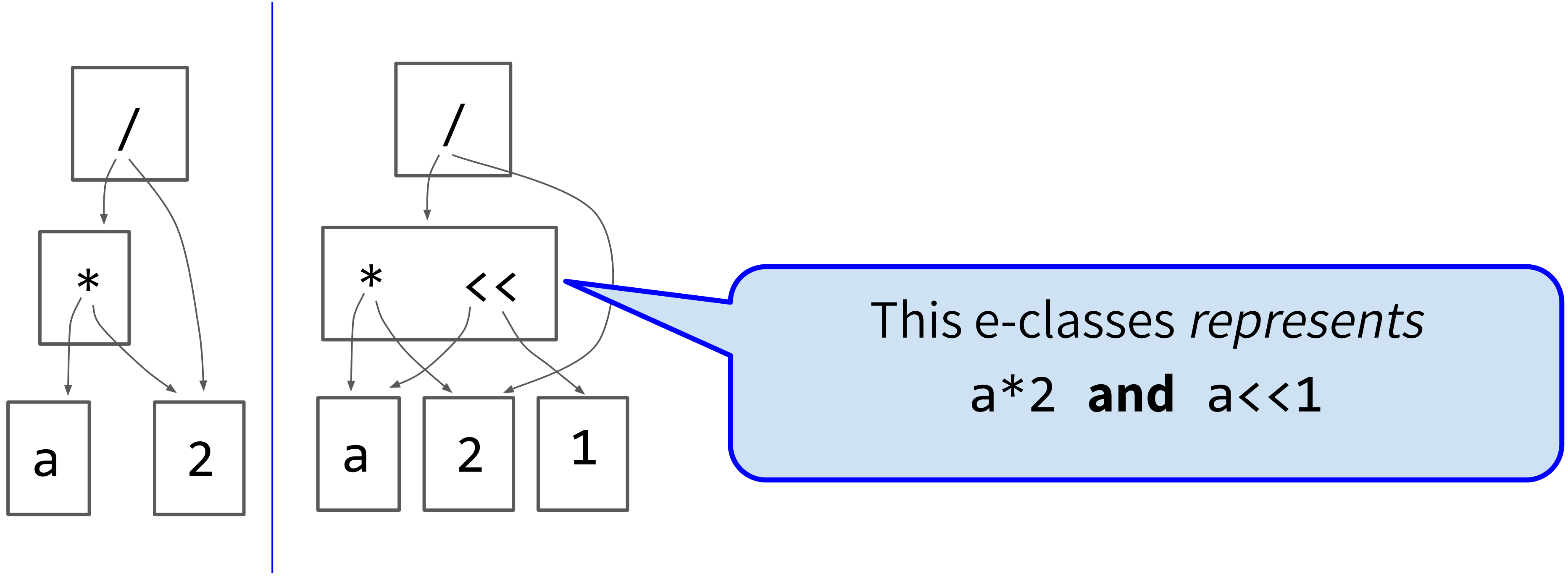
`(a * 2) / 2`

E-graphs: applying rewrite rules



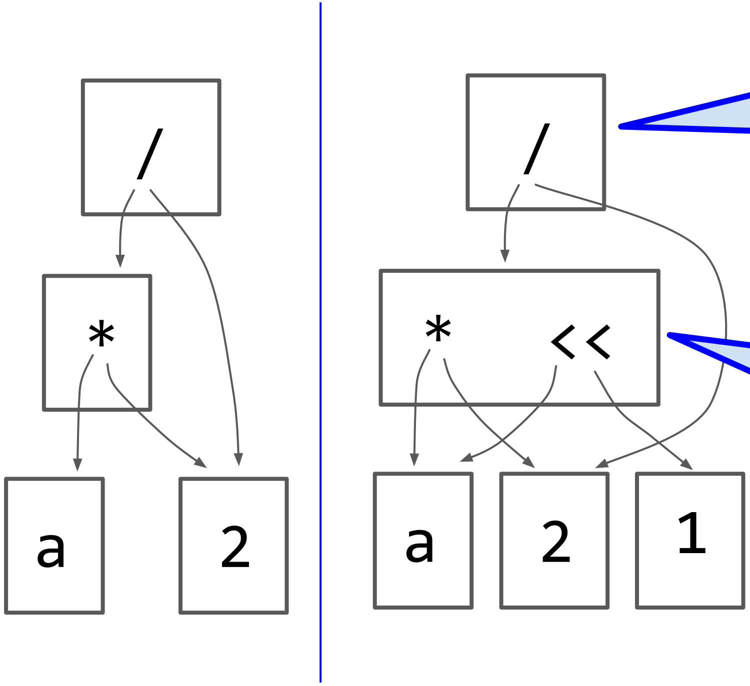
`x * 2 → x << 1`

E-graphs: applying rewrite rules



x * 2 → x << 1

E-graphs: applying rewrite rules

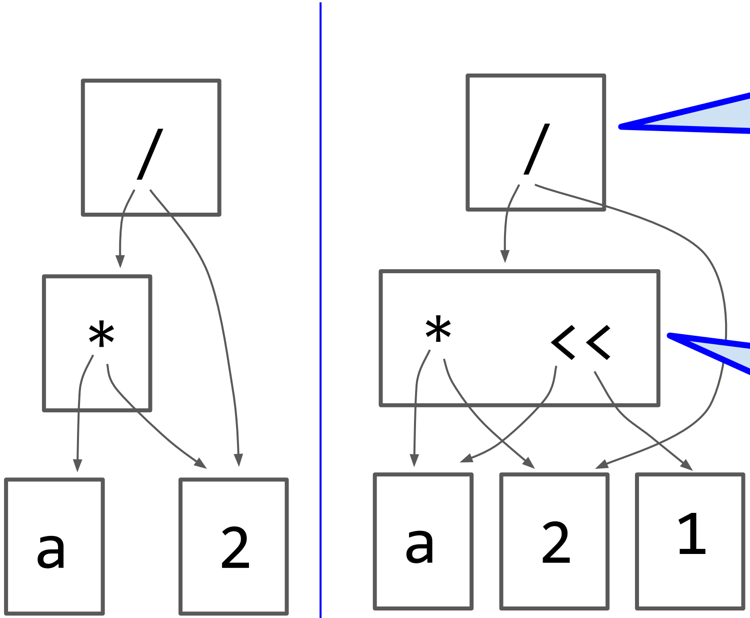


This e-classes *represents*
`(a*2)/2` **and** `(a<<1)/2`

This e-classes *represents*
`a*2` **and** `a<<1`

`x * 2 → x << 1`

E-graphs: applying rewrite rules

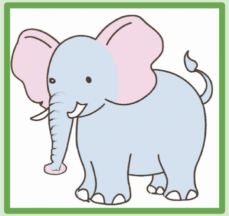


x * 2 → x << 1

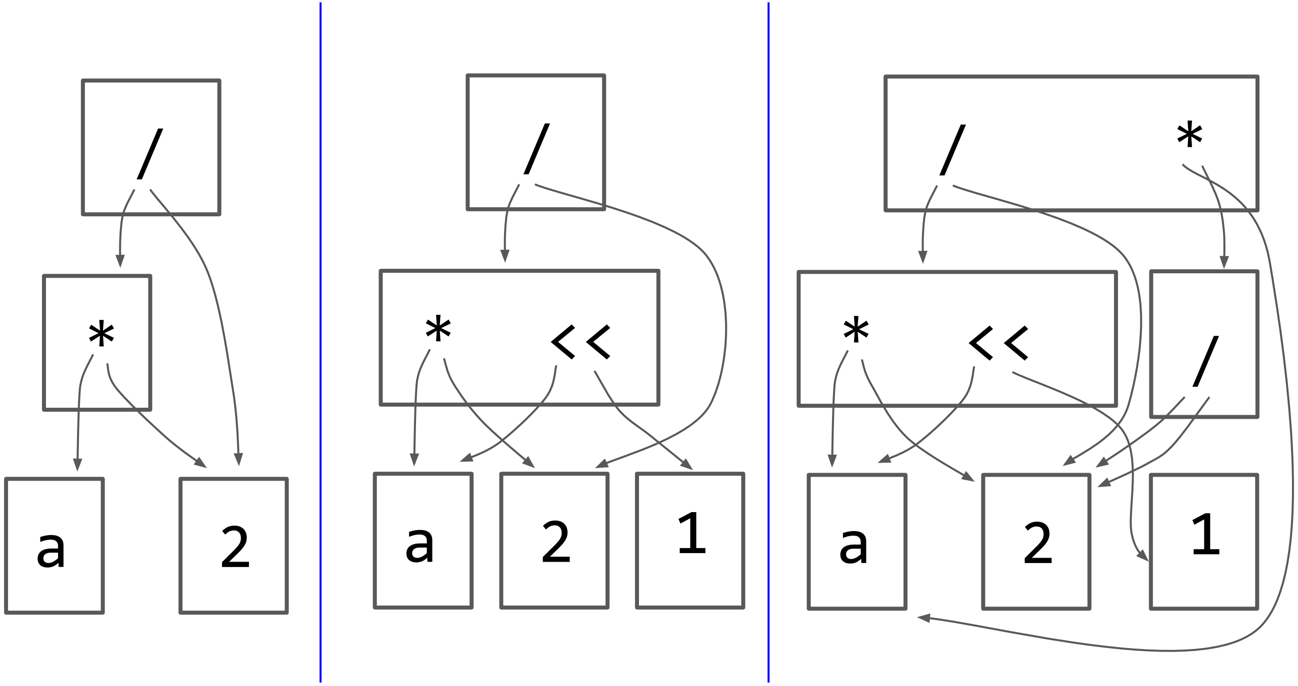
This e-classes *represents*
`(a*2)/2` **and** `(a<<1)/2`

This e-classes *represents*
`a*2` **and** `a<<1`

E-graphs never forget.
Rewrites don't lose info!

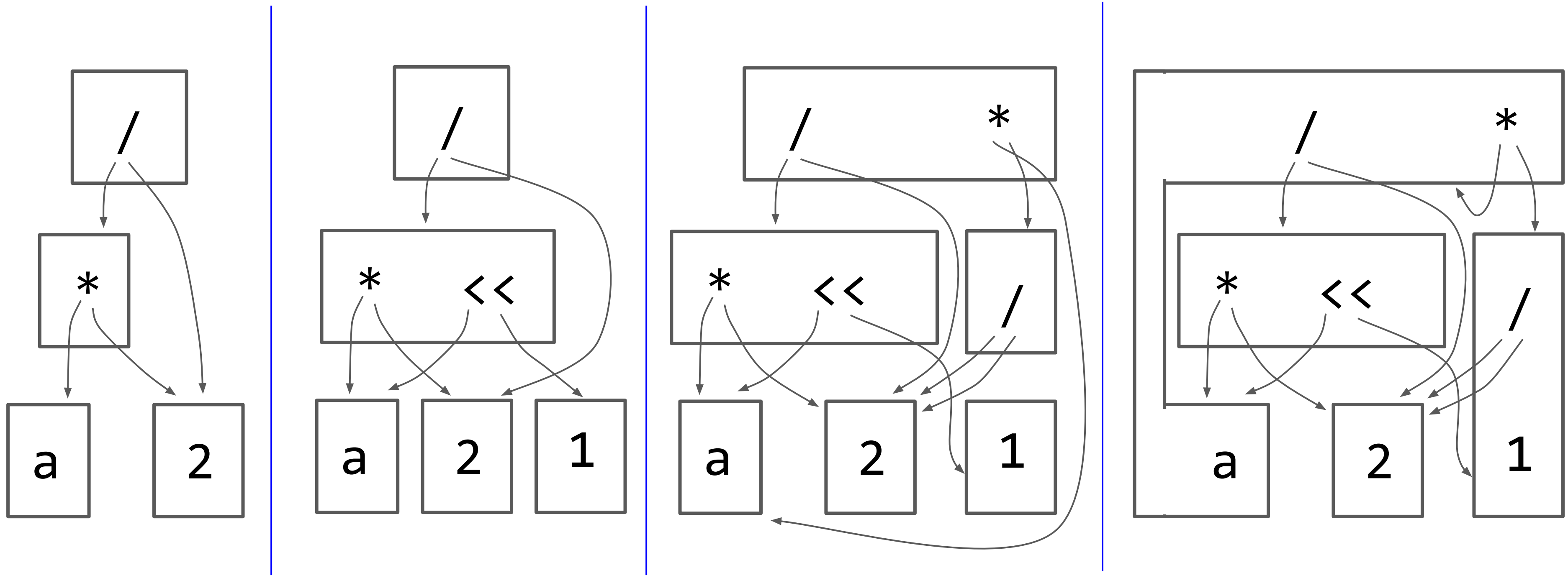


E-graphs: applying rewrite rules



x * 2 → x << 1   (x*y)/z → x*(y/z)

E-graphs: applying rewrite rules



x * 2 → x << 1

(x*y)/z → x*(y/z)

x / x → 1

x * 1 → x

E-graphs: compact representation

Rewrites can **shrink** e-graphs!

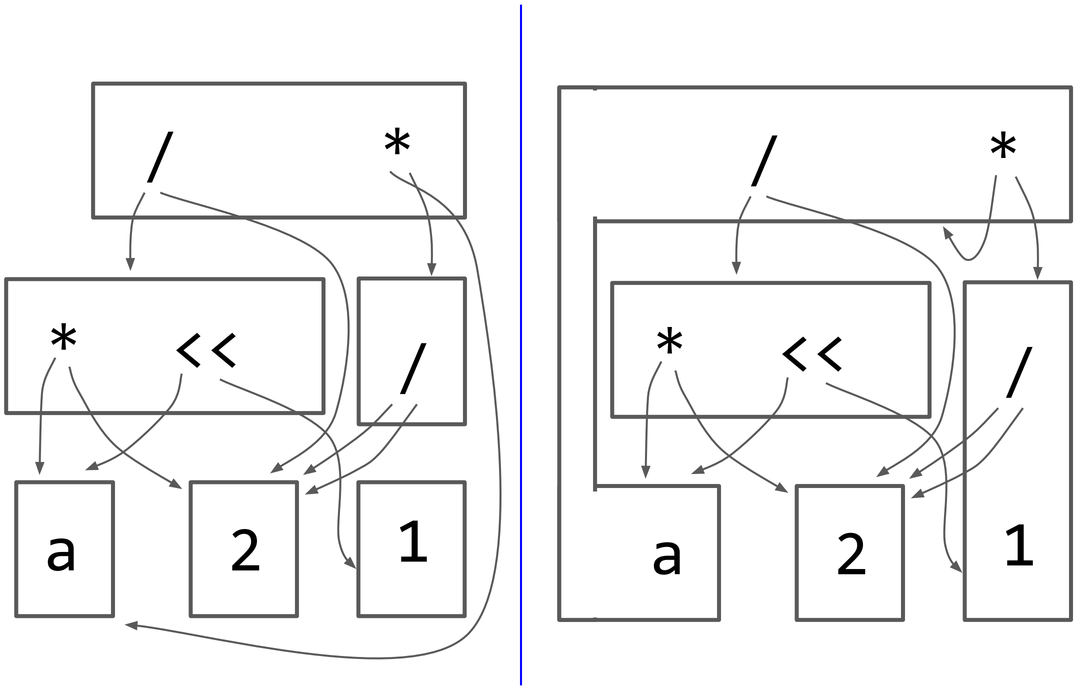
- 6 → 5 eclasses

E-graphs can represent ∞ terms

- `a, a * 1, a * 1 * 1, …`

E-graphs can "*saturate*"

- learn all derivable eqs ✅



`x / x → 1`

`x * 1 → x`

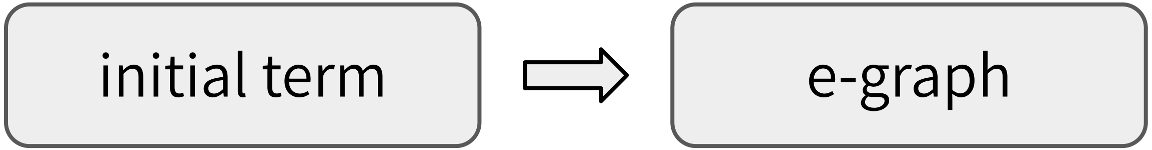
Equality Saturation

- Technique first used in Denali (Joshi, Nelson, Randall 2002)
  - Optimizing straight-line assembly kernels for Alpha
- Extended to loops in Peggy [POPL 2009]
  - Coined term “Equality Saturation”
  - Coinductive stream operators for algebraic loop rewrites
  - Used Rete algo from expert sys for incremental e-matching

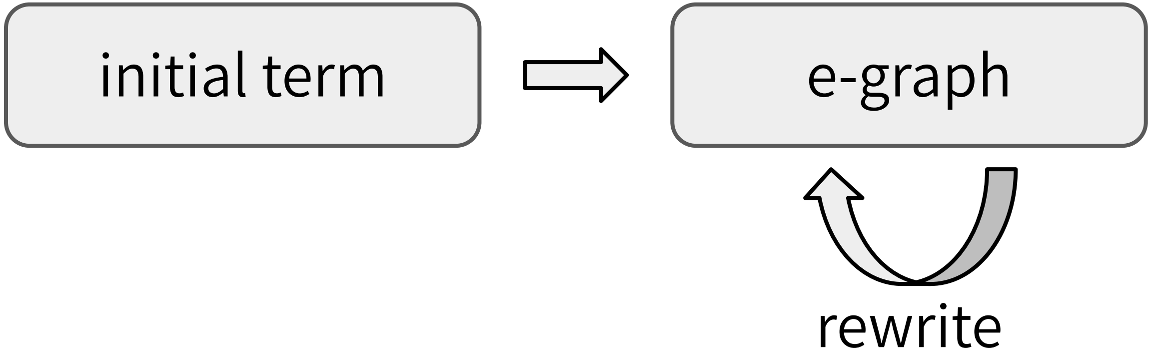
Equality Saturation

initial term

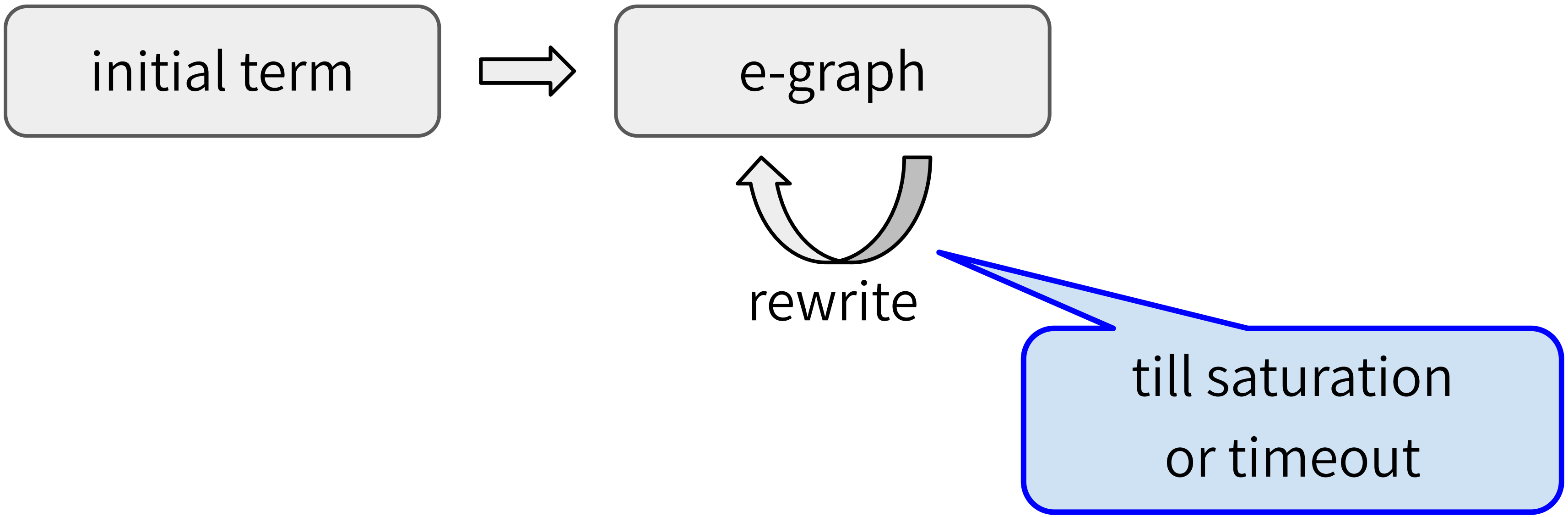
Equality Saturation



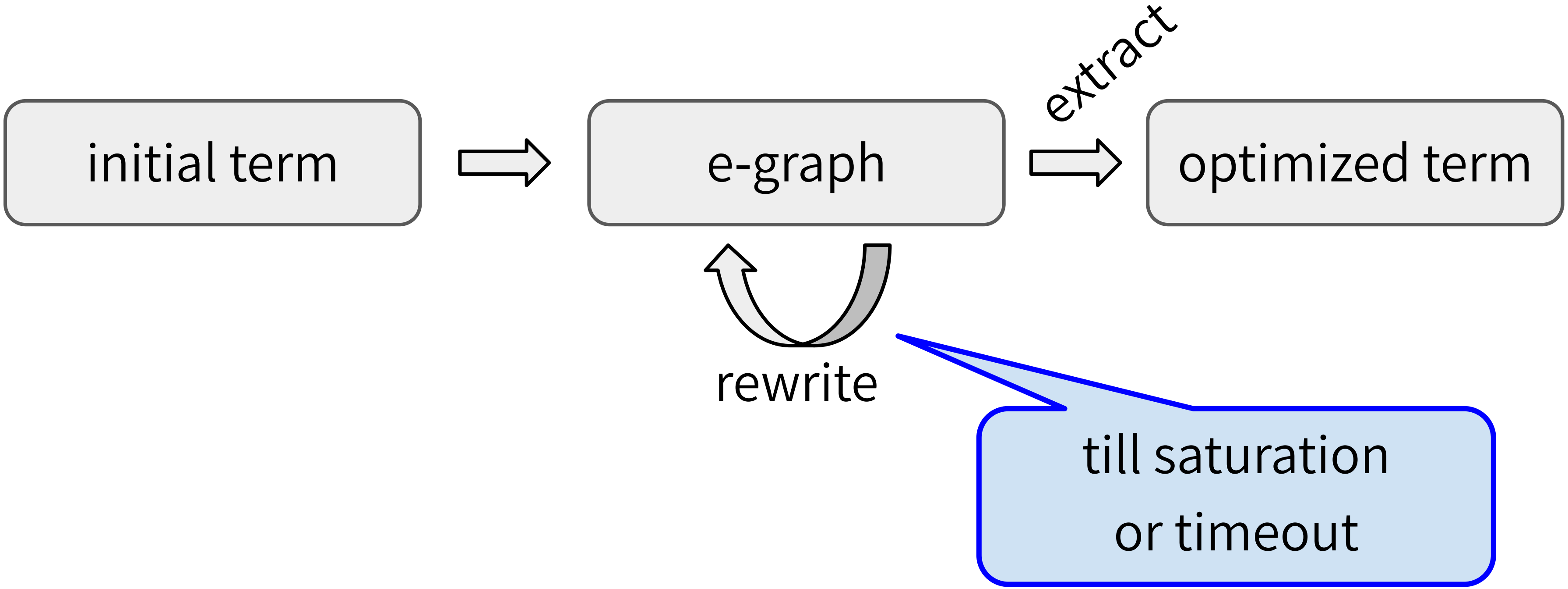
Equality Saturation



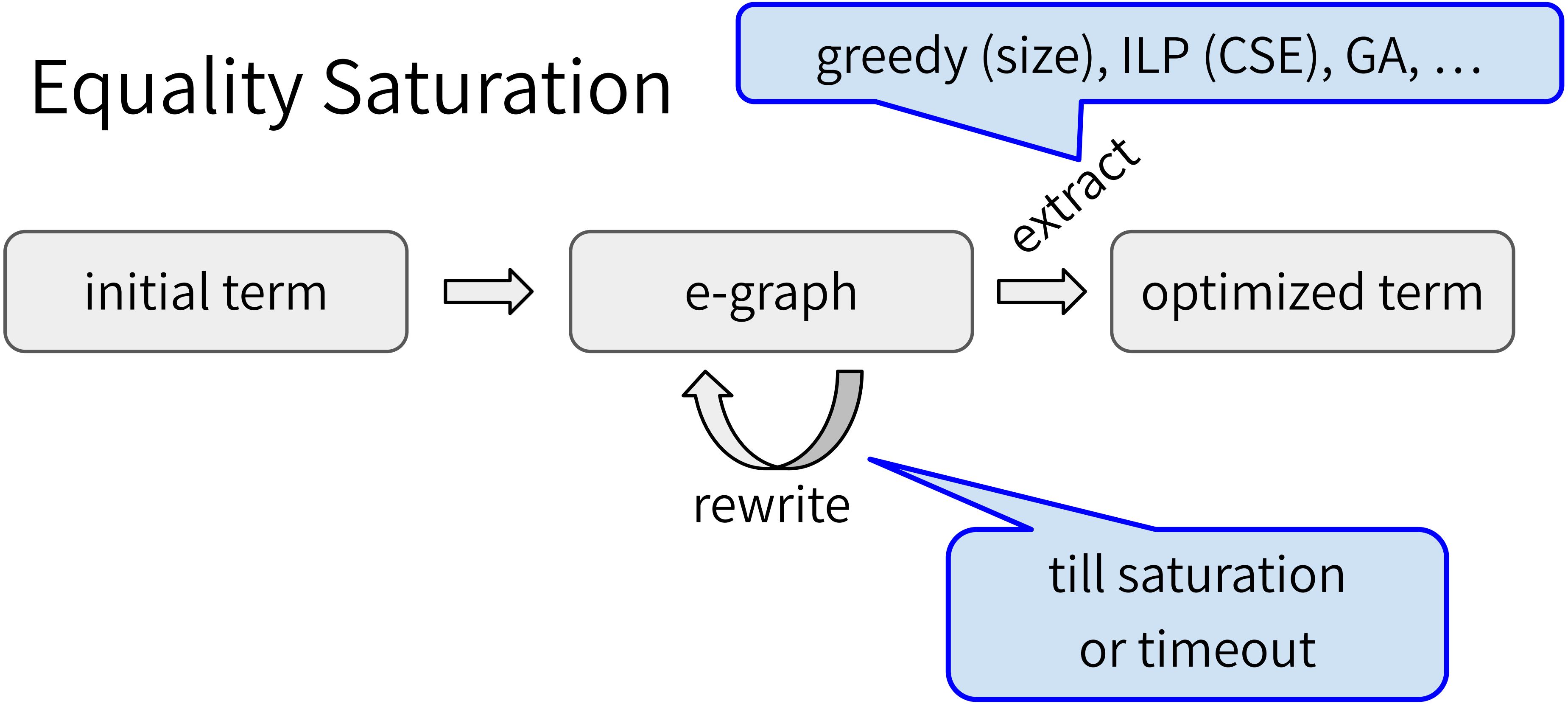
Equality Saturation



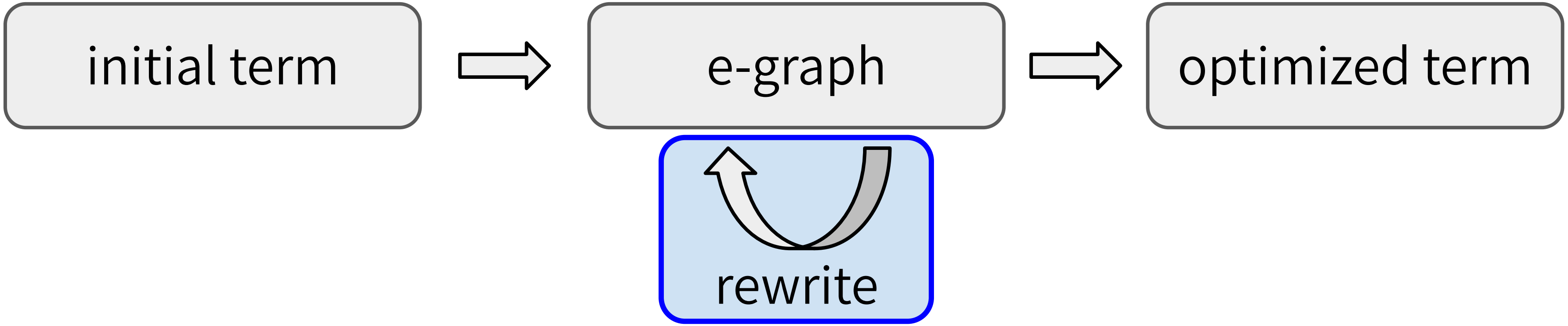
Equality Saturation



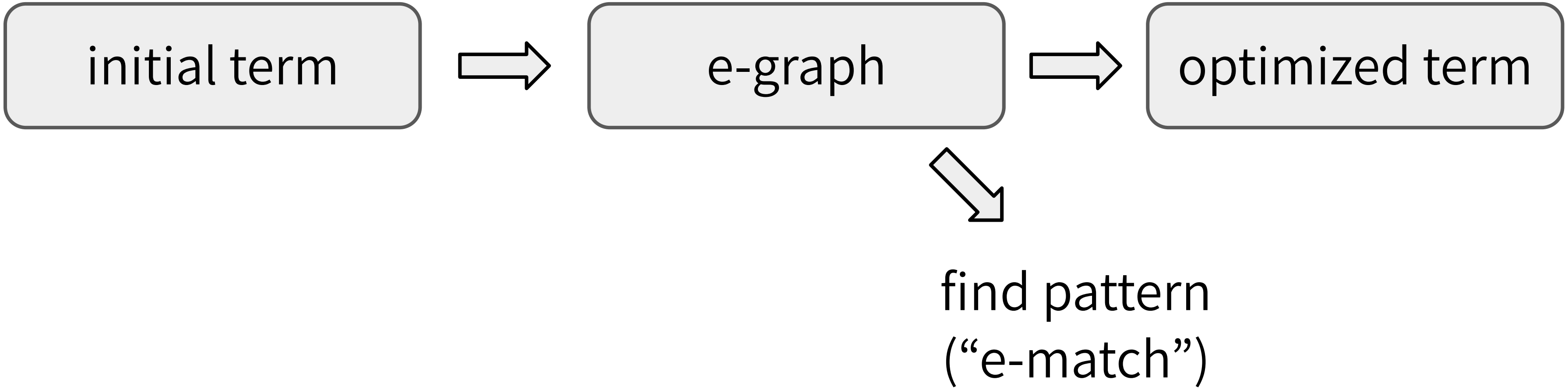
Equality Saturation



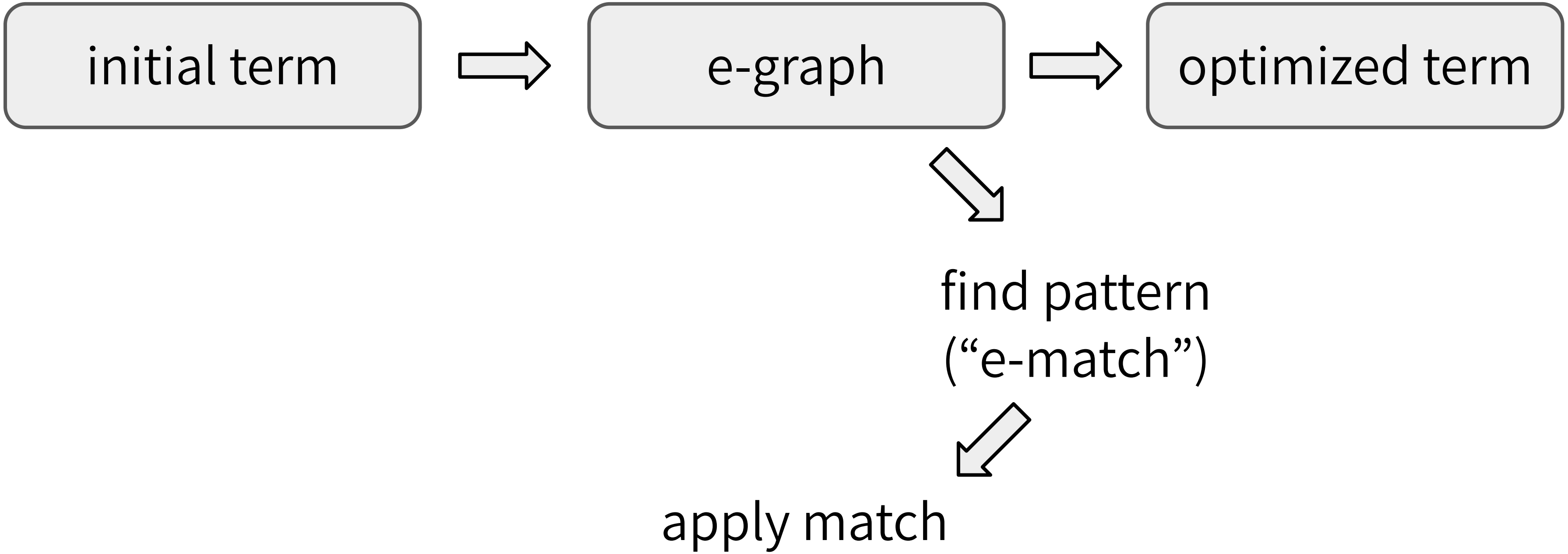
Equality Saturation



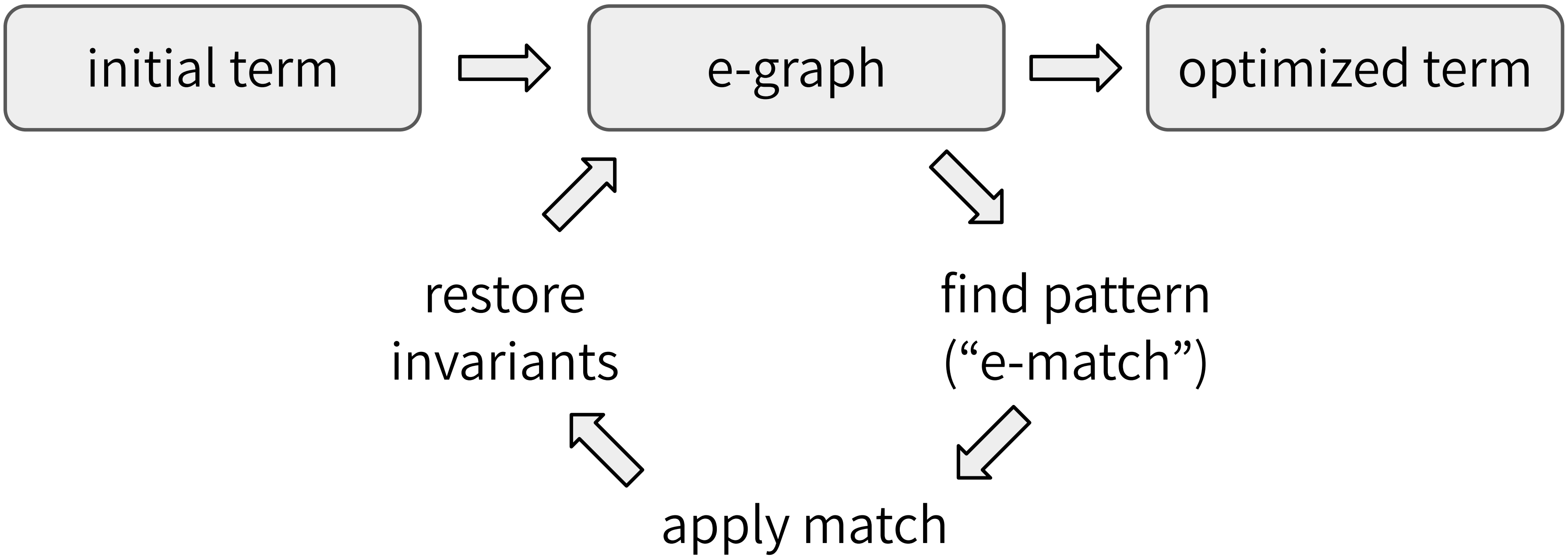
Equality Saturation



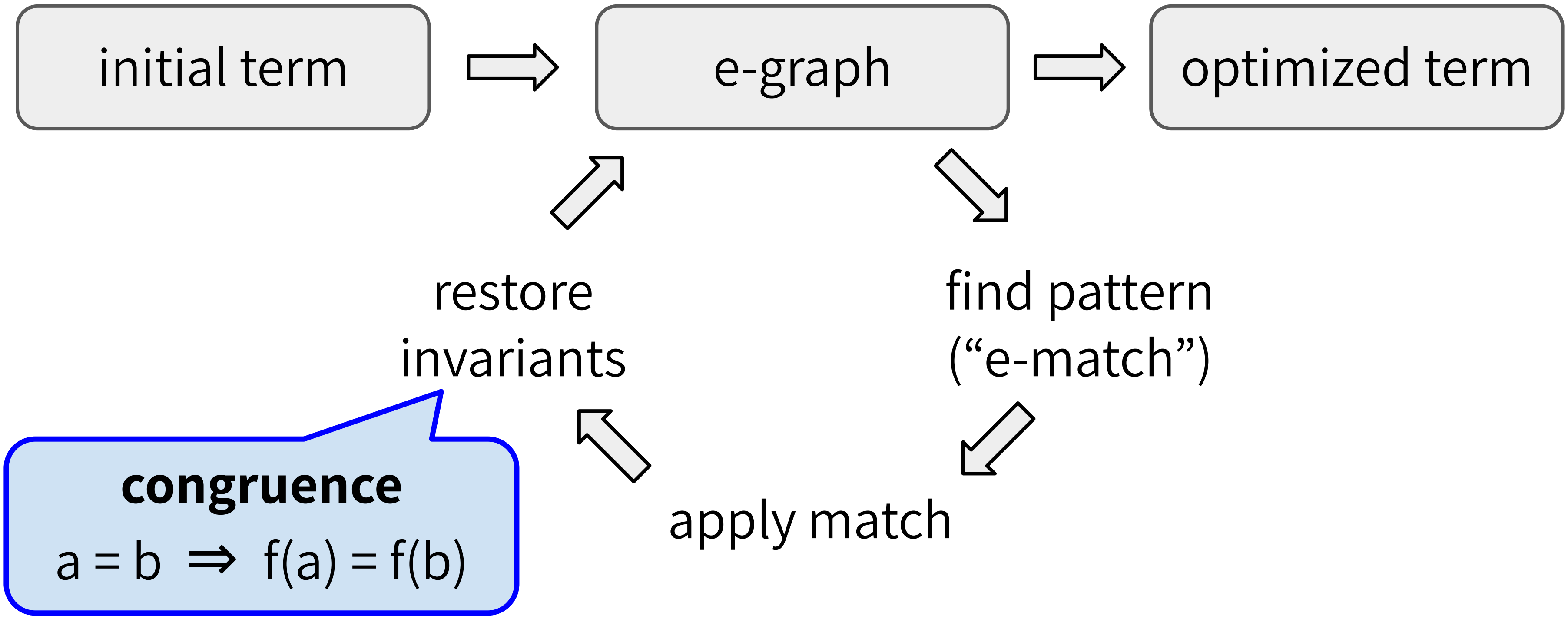
Equality Saturation



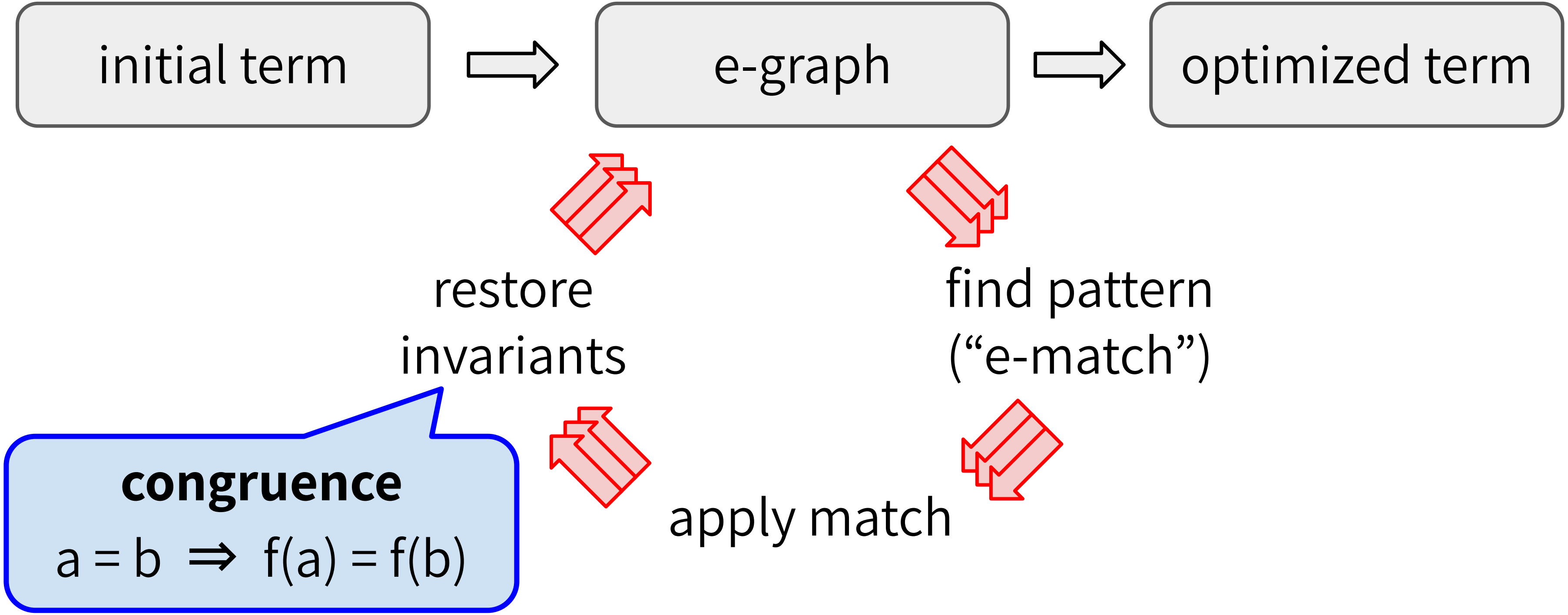
Equality Saturation



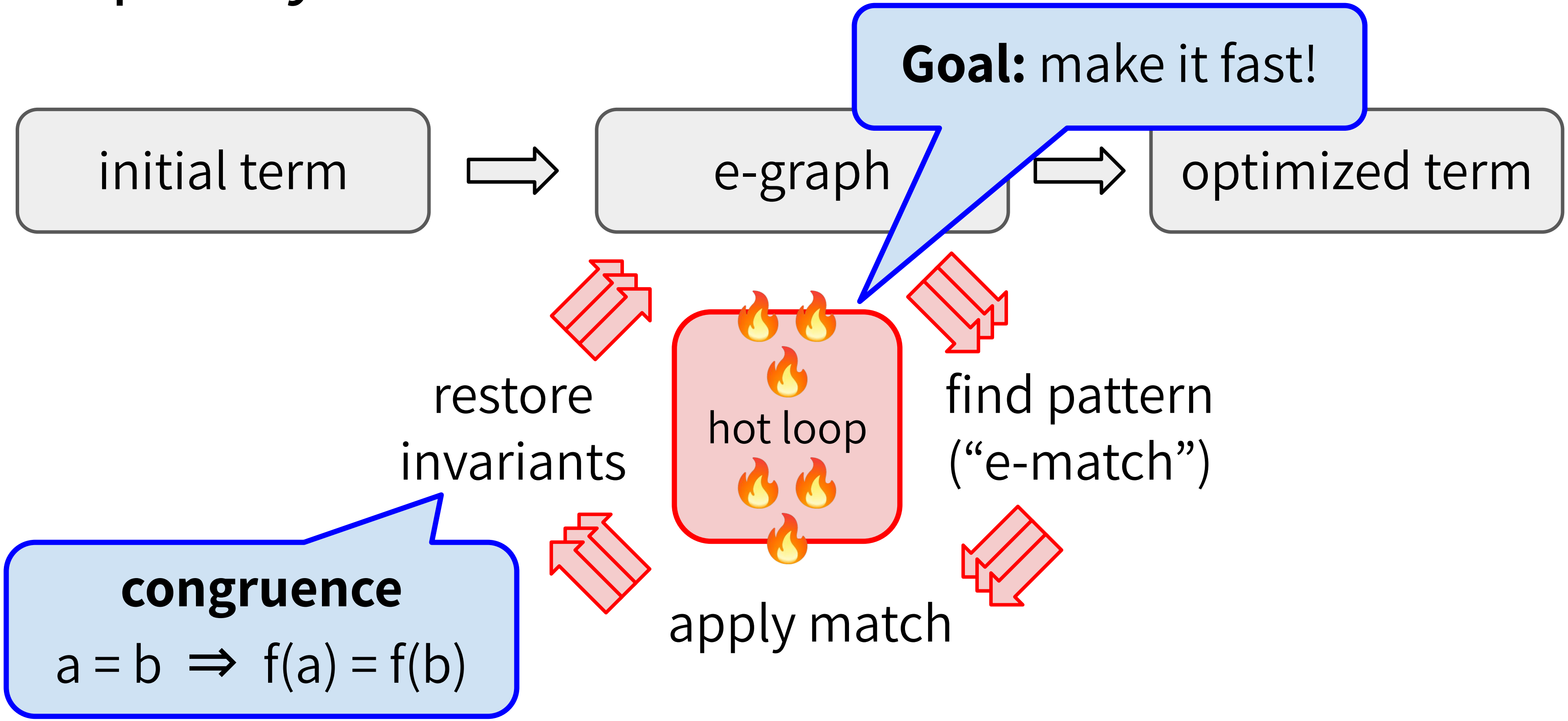
Equality Saturation



Equality Saturation



Equality Saturation



egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- ❑ Deferred invariant maintenance & batching
- ❑ Relational e-matching [POPL 2022]
- ❑ E-class analyses
- ❑ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ❑ Applications
  - ❑ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ❑ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …

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()
```

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()
```

read

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()
```

read

write

Equality Saturation

```python
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()
```

read

write

restore invariant

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()
```

- rewrites are ordered
- read/write interleaved
  - more invariant maint
- invariants baked-in

Deferred Invariant Maintenance in egg

```
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()
```

```
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 in egg

```
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()
```

```
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 in egg

```
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()
```

```
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 in egg

```
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()
```

```
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 in egg

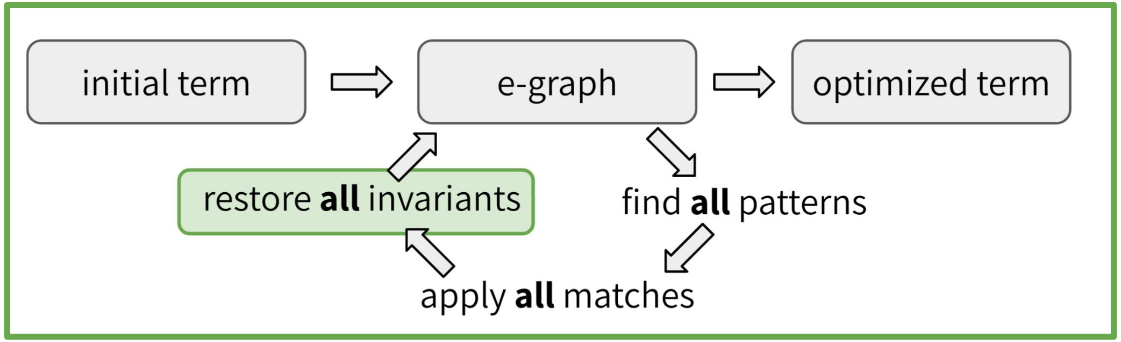
```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  while not egraph.is_sat
    for rw in rewrites:
      for (subst, ec) i
        ec2 = egraph.a
        egraph.merge(e

  return egraph.extr
```

batch reads

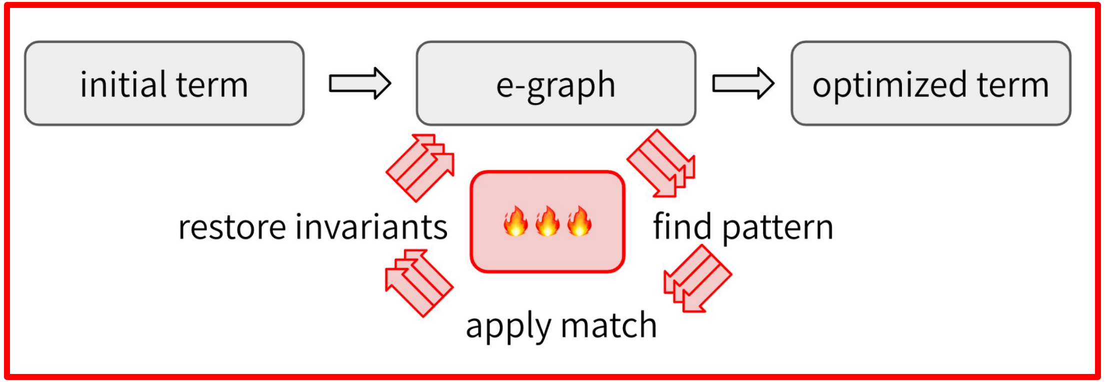
batch writes
(invariants broken)

invariants restored
once per iteration

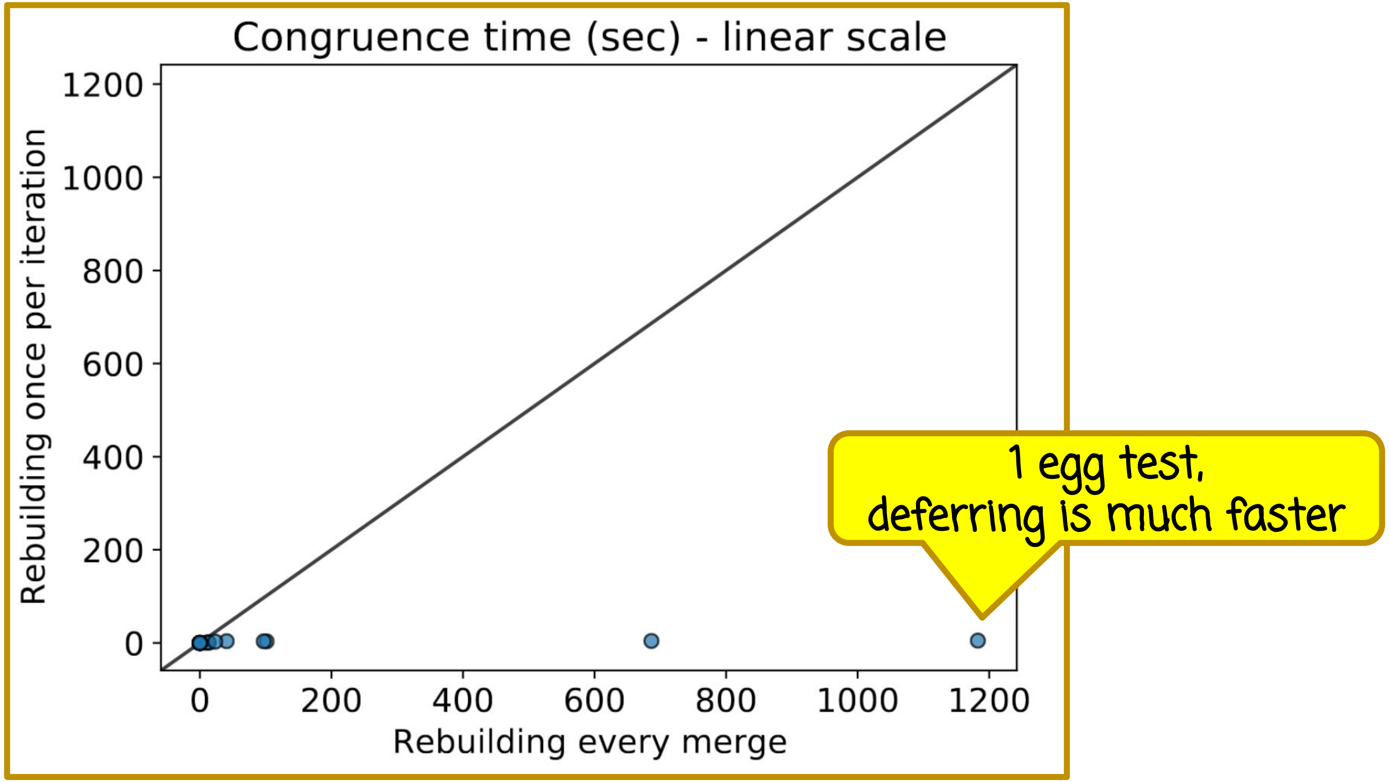
```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  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()
```


initial term
e-graph
optimized term
restore all invariants
find all patterns
apply all matches

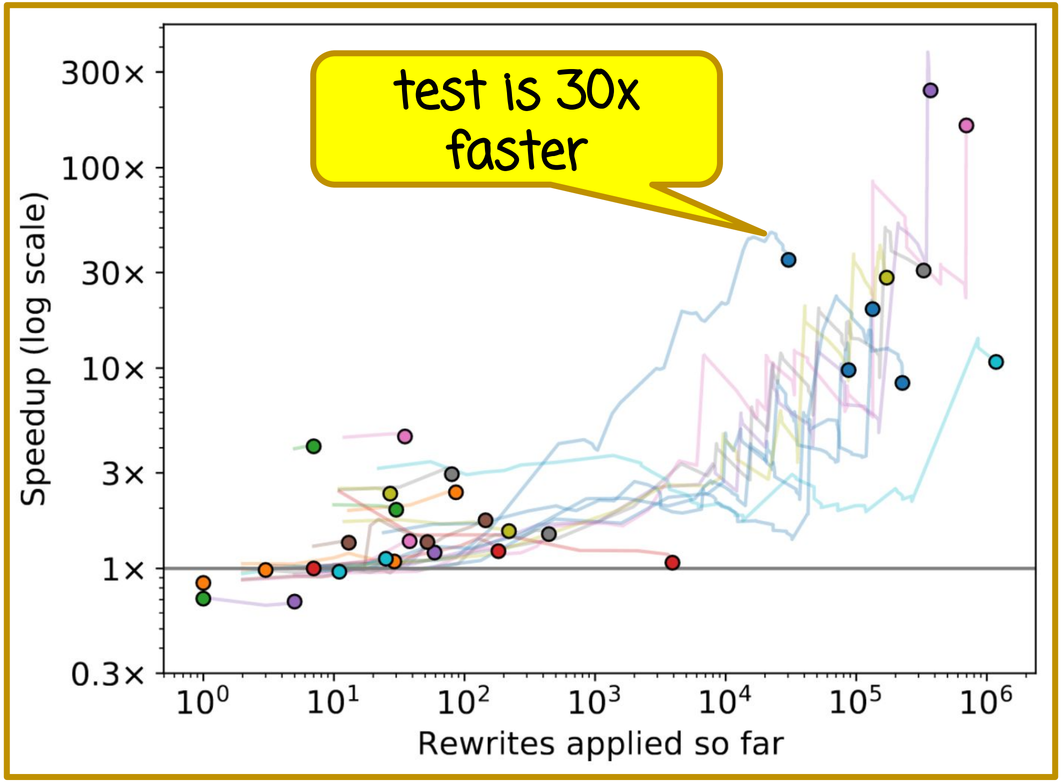
versus


initial term
e-graph
optimized term
restore invariants
find pattern
apply match

Rebuilding is faster



Rebuilding is faster

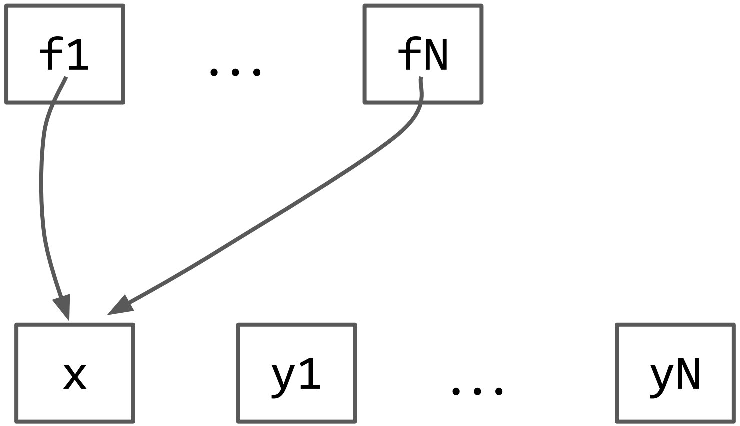


Why is rebuilding is faster?

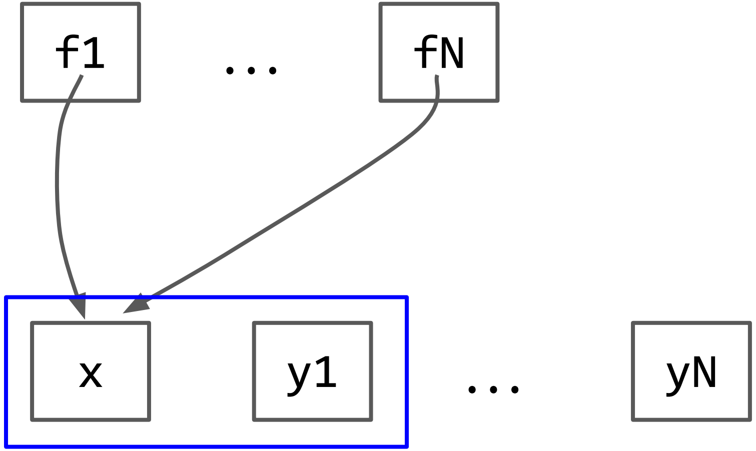
- Consider $f_1(x)$ … $f_n(x)$ and y_1 … y_n
- Workload: merge(x, y_1) … merge(x, y_n)
- Traditional: $O(n^2)$ hashcons updates
- Deferred only does $O(n)$ updates

Downey, Sethi, Tarjan 1980

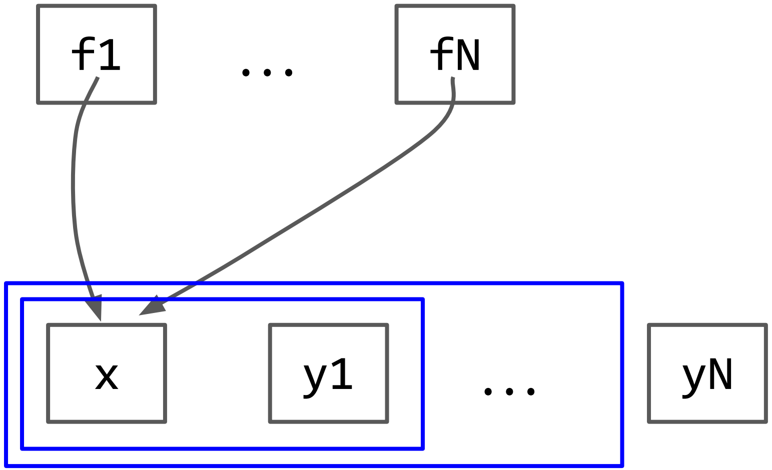
Why is rebuilding is faster?



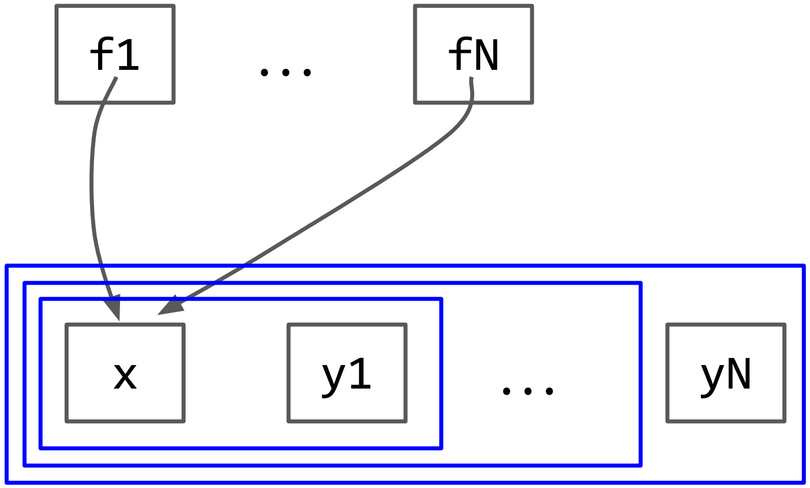
Why is rebuilding is faster?



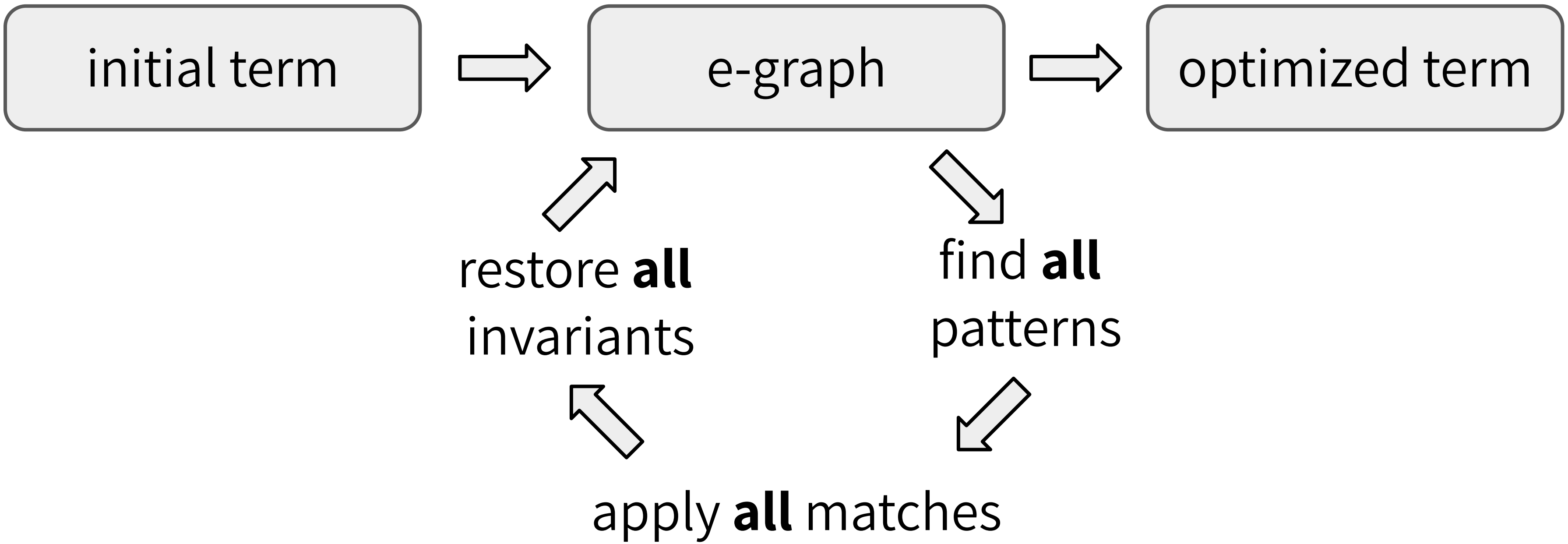
Why is rebuilding is faster?



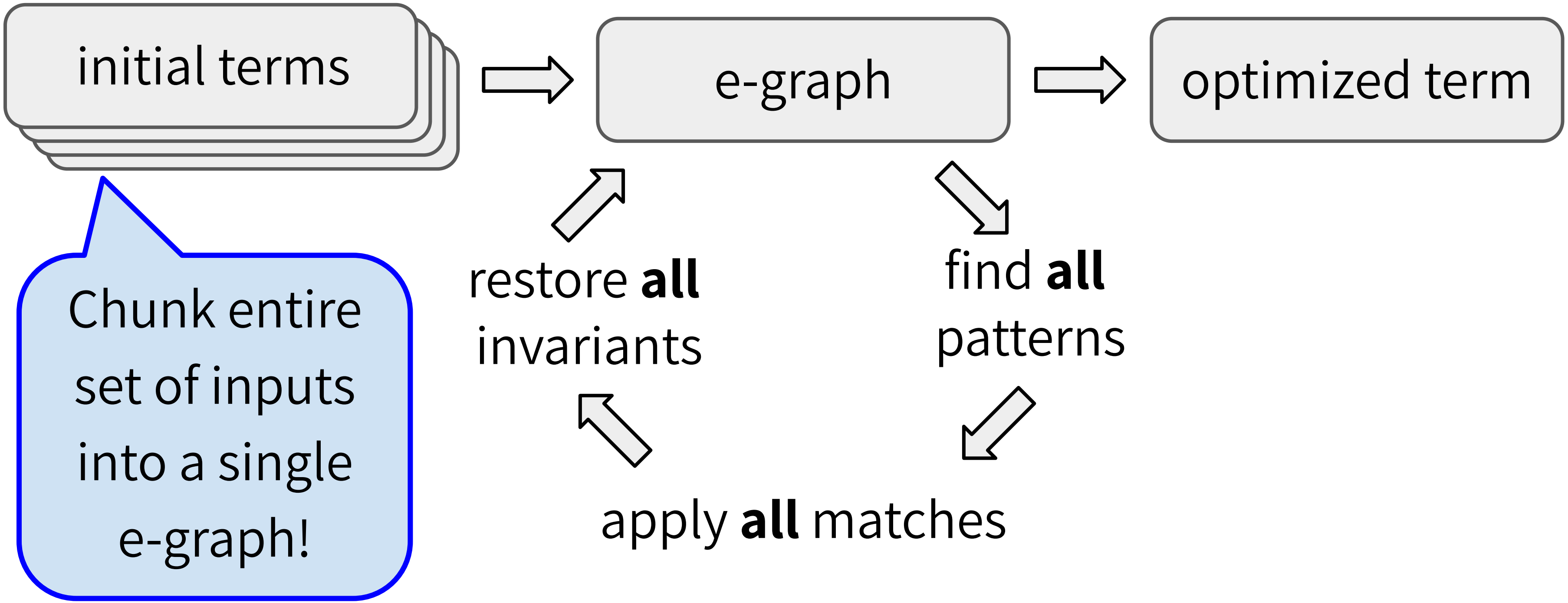
Why is rebuilding is faster?



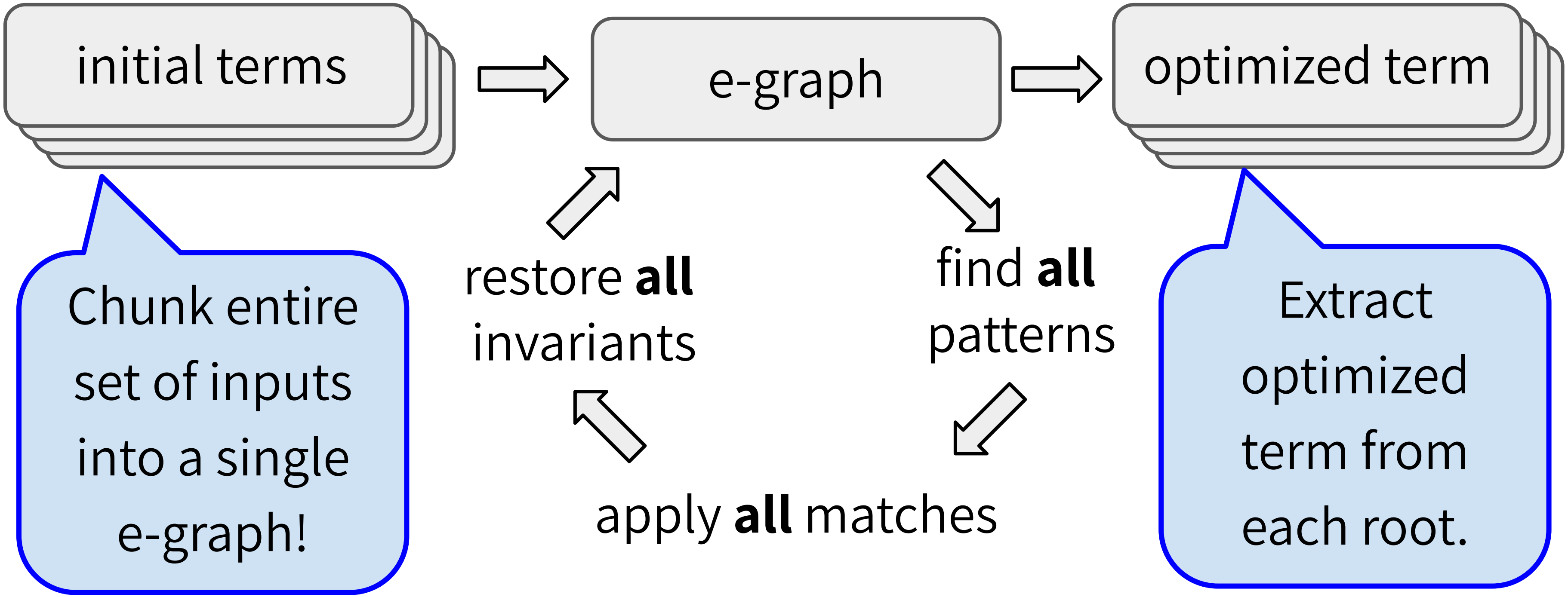
More amortization via *batching* in egg



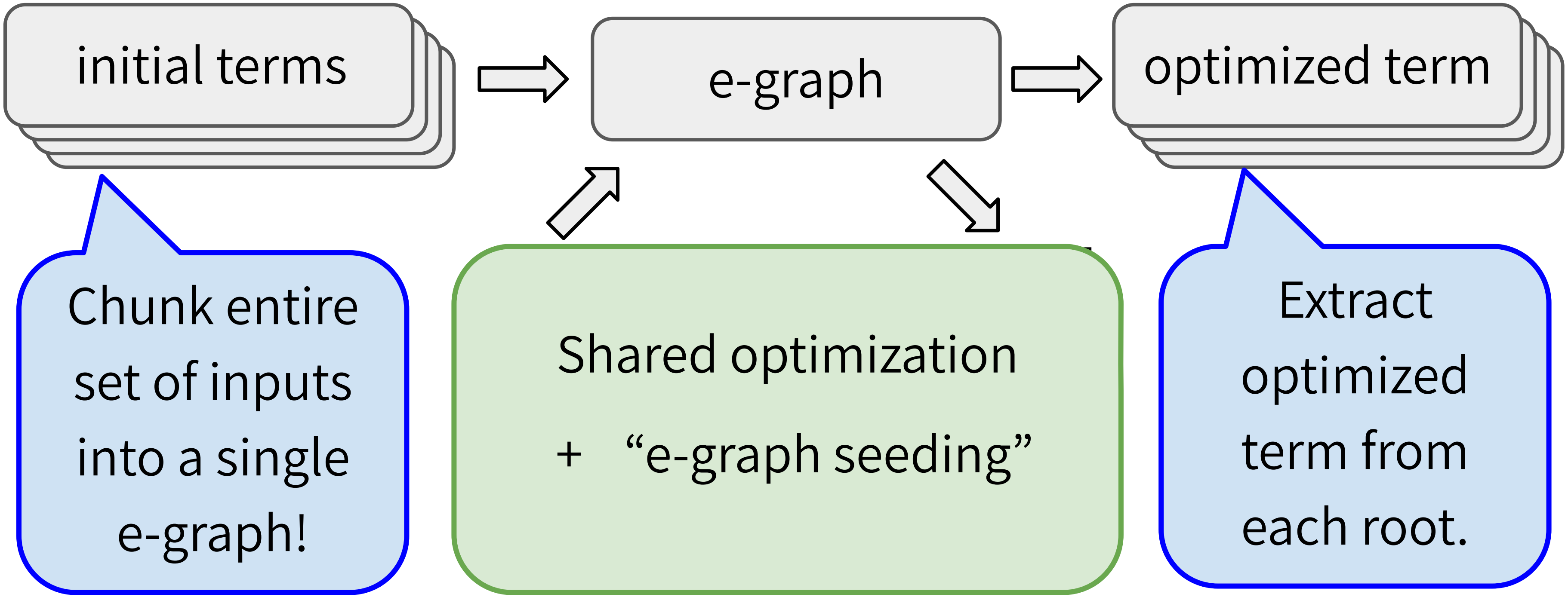
More amortization via *batching* in egg



More amortization via *batching* in egg



More amortization via *batching* in egg

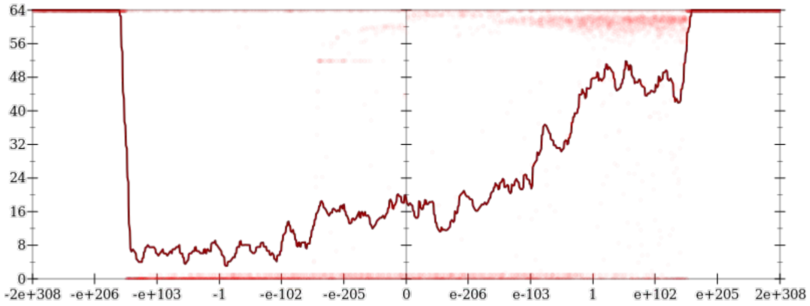


E-graphs in Herbie

$$\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}$$

E-graphs in Herbie

$$\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}$$



E-graphs in Herbie

$$\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}$$

$$\downarrow$$

if $b \leq -2.1714197031320663 \cdot 10^{+114}$:

$$-\frac{b}{a}$$

elif $b \leq 2.9809086538561536 \cdot 10^{-153}$:

$$\frac{\sqrt{\mathsf{fma}(b, b, c \cdot (a \cdot -4))} - b}{a \cdot 2}$$

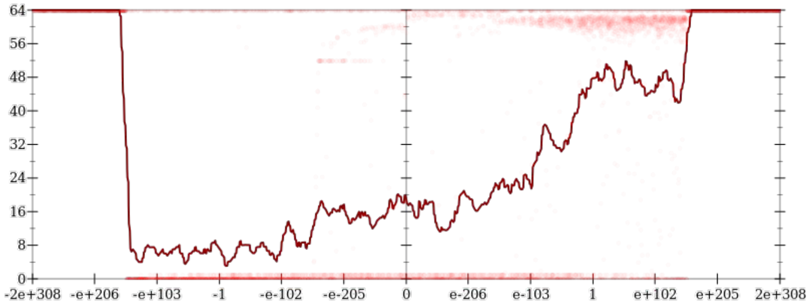
elif $b \leq 3.095118518558678 \cdot 10^{+20}$:

$$t_0 := 4 \cdot (a \cdot c)$$

$$\frac{t_0 \cdot \frac{0.5}{a}}{(-b) - \sqrt{b \cdot b - t_0}}$$

else :

$$-\frac{c}{b}$$



E-graphs in Herbie

$$\frac{(-b)+\sqrt{b\cdot b-4\cdot(a\cdot c)}}{2\cdot a}$$

$$\downarrow$$

if $b \le -2.1714197031320663\cdot 10^{+114}$:

$$-\frac{b}{a}$$

elif $b \le 2.9809086538561536\cdot 10^{-153}$:

$$\frac{\sqrt{\mathsf{fma}(b,b,c\cdot(a\cdot -4))}-b}{a\cdot 2}$$

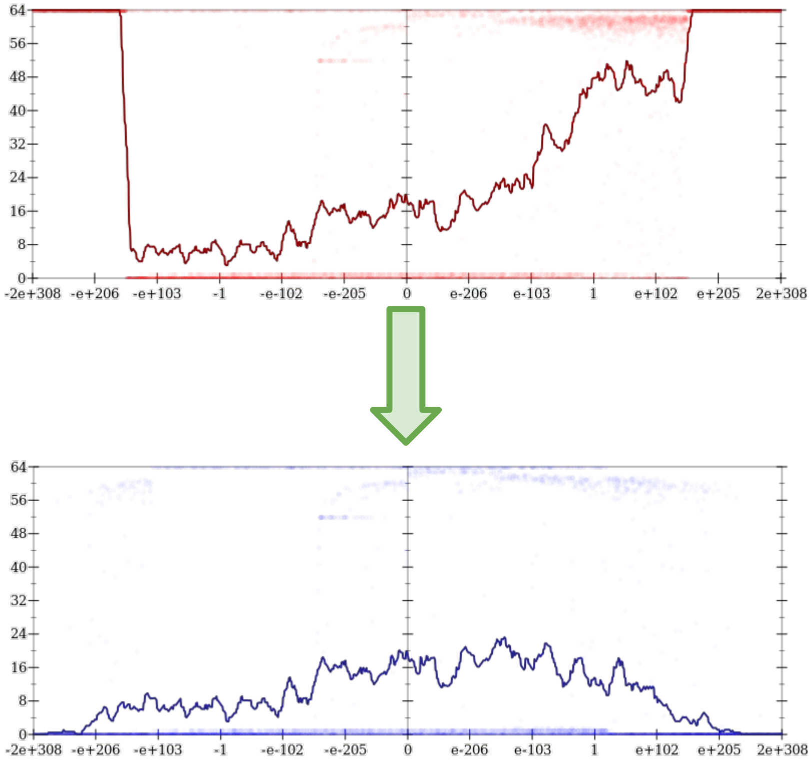
elif $b \le 3.095118518558678\cdot 10^{+20}$:

$$t_0 := 4\cdot(a\cdot c)$$

$$\frac{t_0\cdot\frac{0.5}{a}}{(-b)-\sqrt{b\cdot b-t_0}}$$

else :

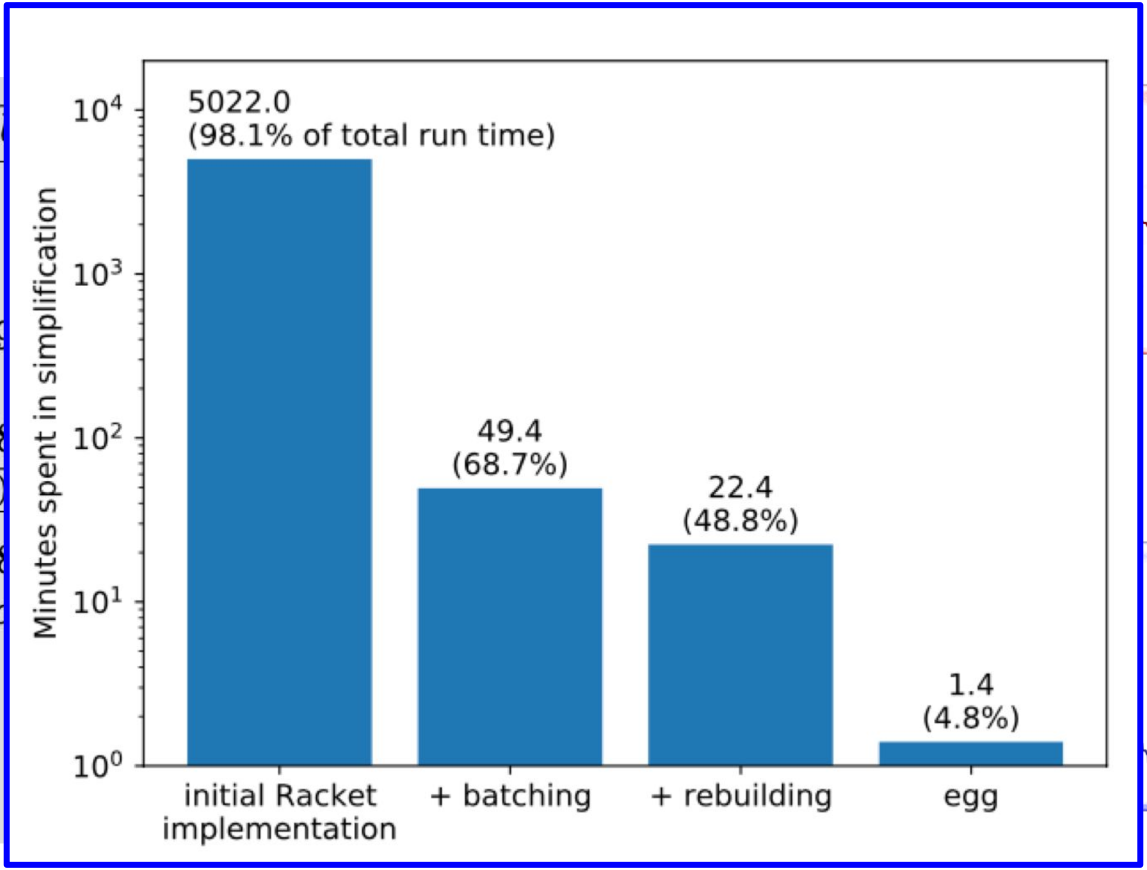
$$-\frac{c}{b}$$

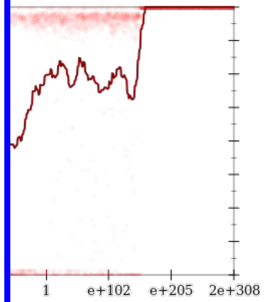


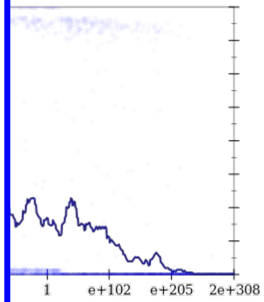
egg Herbie

$$\frac{(-b) + \sqrt{\cdots}}{\cdots}$$

if $b \leq -2.171419\ldots$
$-\frac{b}{a}$
elif $b \leq 2.980908\ldots$
$\frac{\sqrt{\mathsf{fma}(b, b, c \cdot (a \cdot -4))}\ldots}{a \cdot 2}$
elif $b \leq 3.095118\ldots$
$t_0 := 4 \cdot (a \cdot c\ldots$
$\frac{t_0 \cdot \frac{0.5}{a}}{(-b) - \sqrt{b \cdot b - t_0}}$
else :
$-\frac{c}{b}$



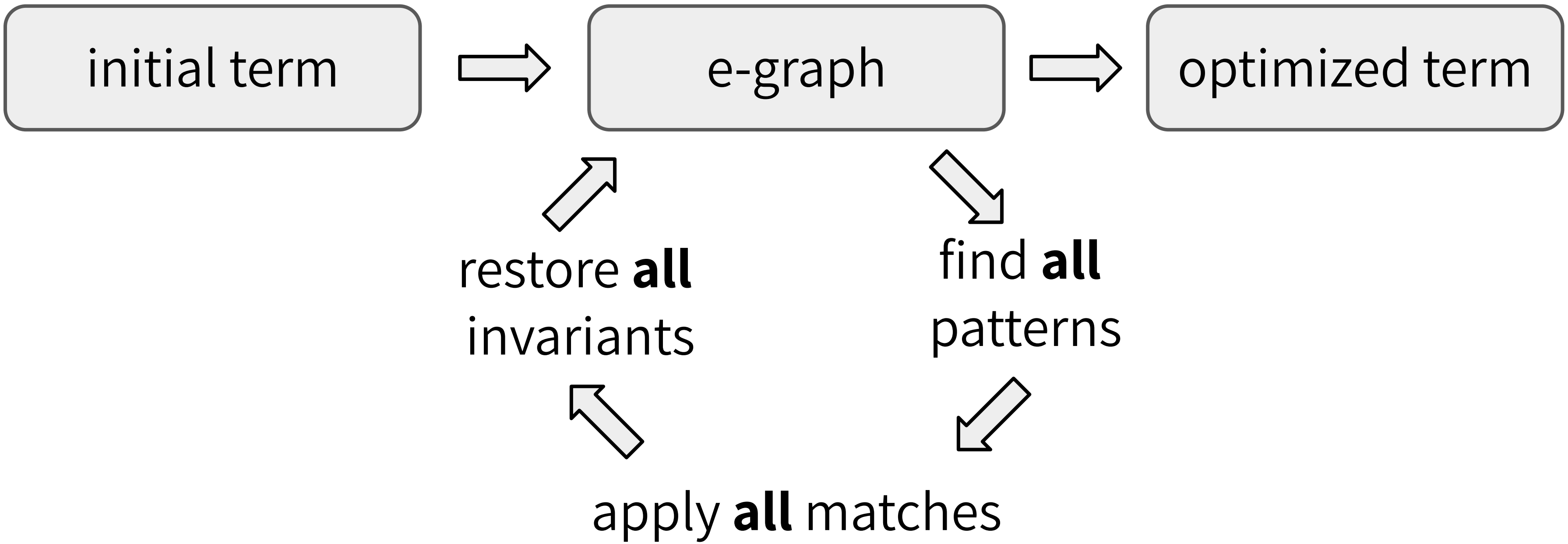




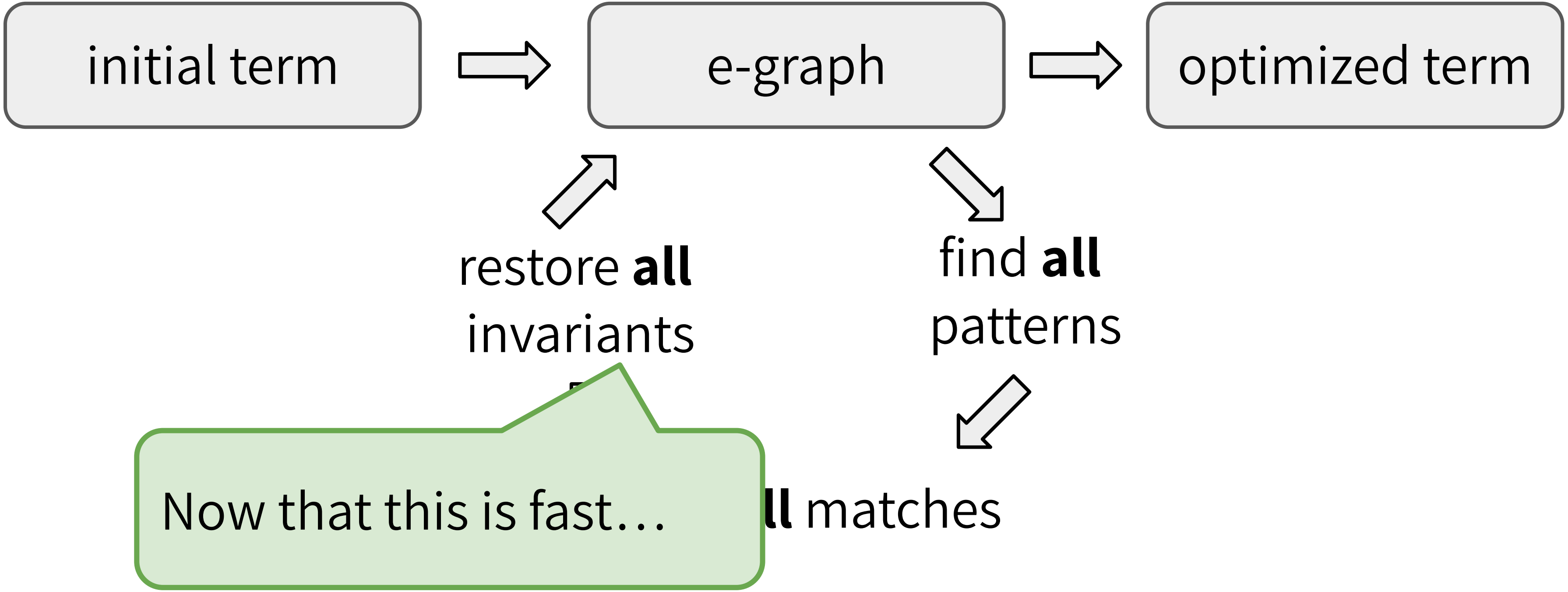
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- ✓ Deferred invariant maintenance & batching
- ❏ Relational e-matching [POPL 2022]
- ❏ E-class analyses
- ❏ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ❏ Applications
  - ❏ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ❏ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …

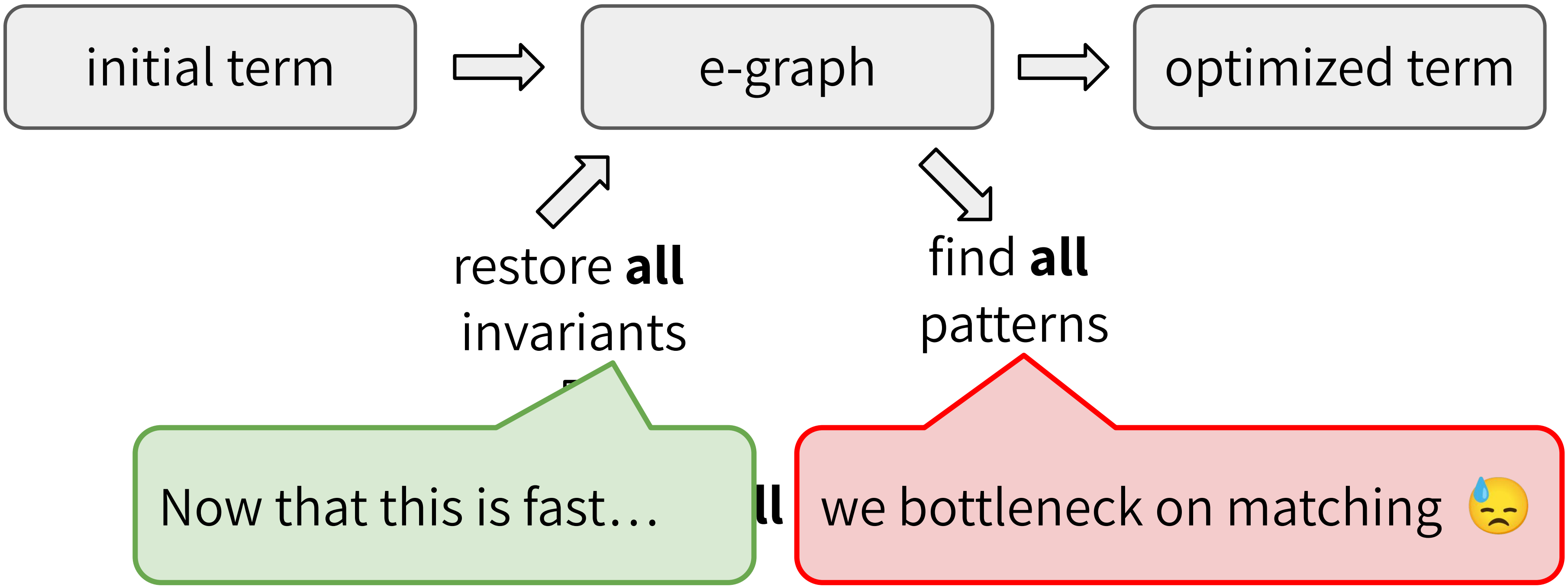
egg's Equality Saturation



egg's Equality Saturation



egg's Equality Saturation

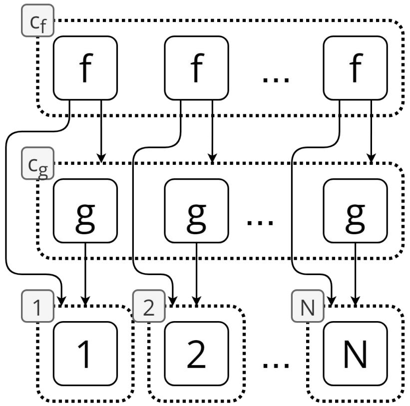


E-matching: pattern matching over e-graphs

- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph

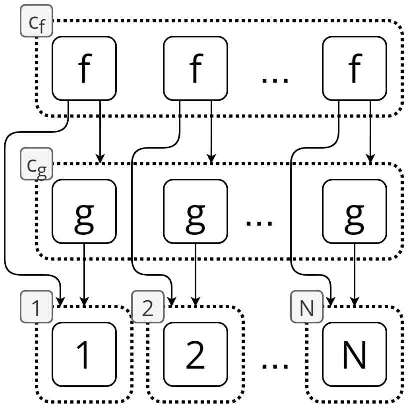
E-matching: pattern matching over e-graphs

- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph



E-matching: pattern matching over e-graphs

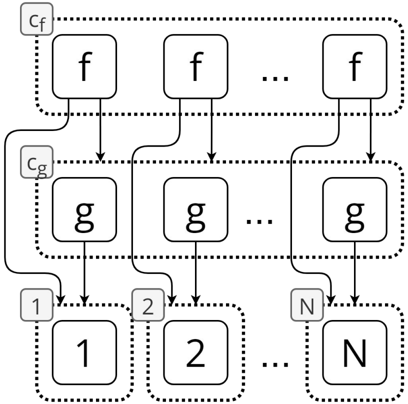
- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph



$$f(\alpha, g(\alpha)) \text{ will match } \begin{matrix} f(1, g(1)) \\ f(2, g(2)) \\ \ldots \\ f(N, g(N)) \end{matrix}, \text{ witnessed by } \begin{matrix} \{\alpha \mapsto 1\} \\ \{\alpha \mapsto 2\} \\ \ldots \\ \{\alpha \mapsto N\} \end{matrix}.$$

E-matching: pattern matching over e-graphs

- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph



f(α, g(α)) will match

f(1, g(1))
f(2, g(2))
…
f(N, g(N))

, witnessed by

{*α* ↦ *1*}
{*α* ↦ *2*}
…
{*α* ↦ *N*}

.

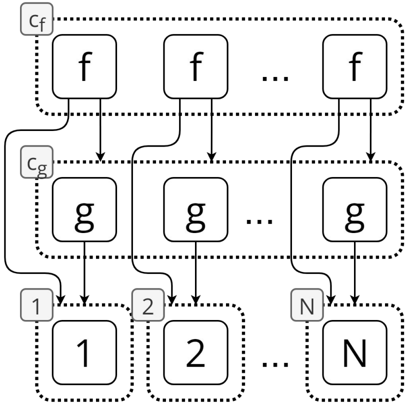
f(1, α) will match

f(1, g(1))
f(1, g(2))
…
f(1, g(N))

, witnessed by {*α* ↦ c_g}.

E-matching: pattern matching over e-graphs

- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977) 😱

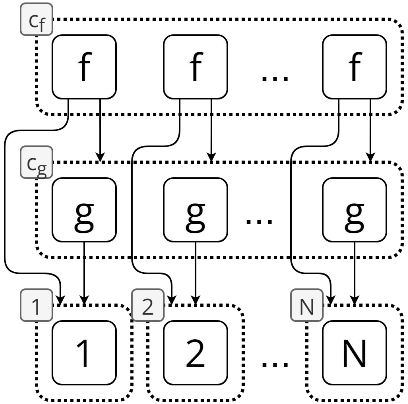


f(a, g(a)) will match *f(1, g(1))*, *f(2, g(2))*, ..., *f(N, g(N))*, witnessed by *{a ↦ 1}*, *{a ↦ 2}*, ..., *{a ↦ N}*.

f(1, a) will match *f(1, g(1))*, *f(1, g(2))*, ..., *f(1, g(N))*, witnessed by *{a ↦ c_g}*.

E-matching: pattern matching over e-graphs

- *E-matching*: find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977) 😱

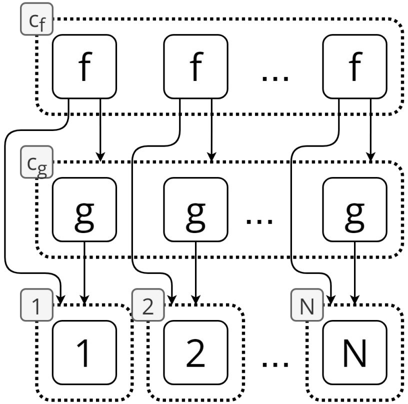


But patterns are often small…

$f(1, \alpha)$ will match $f(1, g(1))$, $f(1, g(2))$, … $f(1, g(N))$, witnessed by $\{\alpha \mapsto c_g\}$.

E-matching: pattern matching over e-graphs

- *E-matching* : find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977) 😱



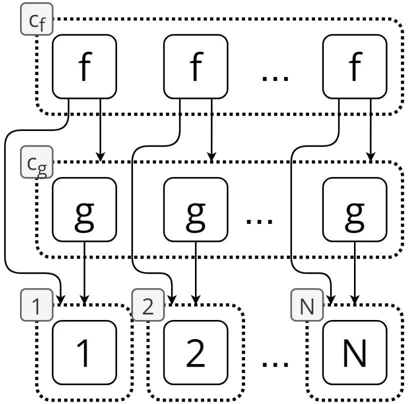
But patterns are often small…

…

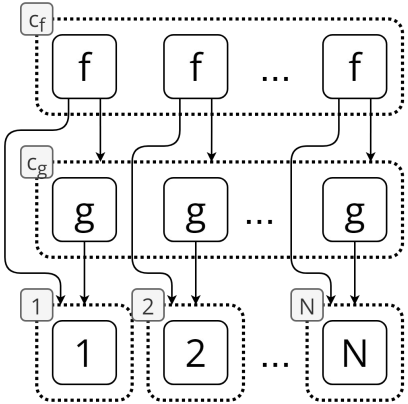
f(1, g(N))

💡 # of matches is much better metric!

Traditional e-matching via backtracking

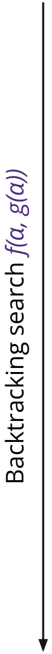


Traditional e-matching via backtracking

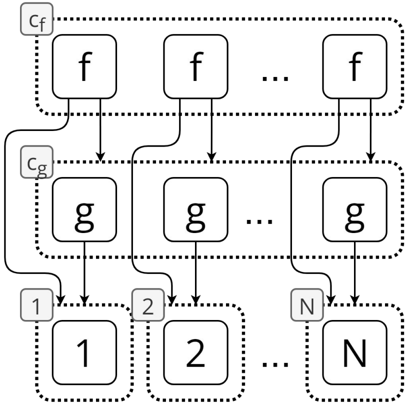


f(α, g(α))

```
for e-class c in e-graph E:
```

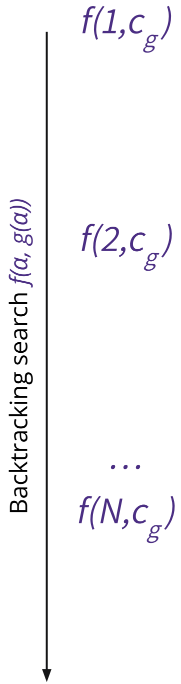


Traditional e-matching via backtracking

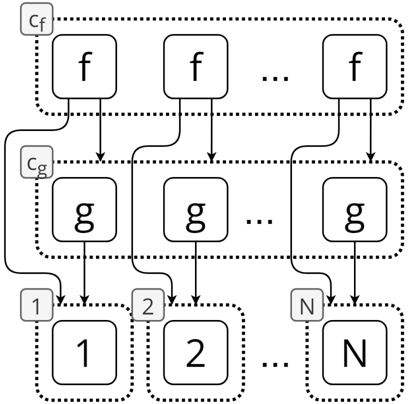


$$f(\alpha, g(\alpha))$$

```
for e-class c in e-graph E:
  for f-node n_1 in c:
```

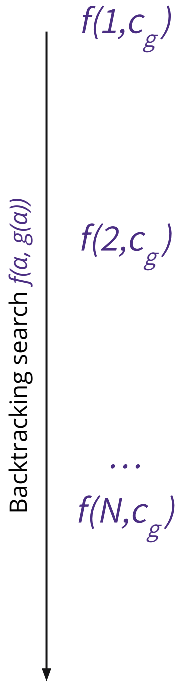


Traditional e-matching via backtracking

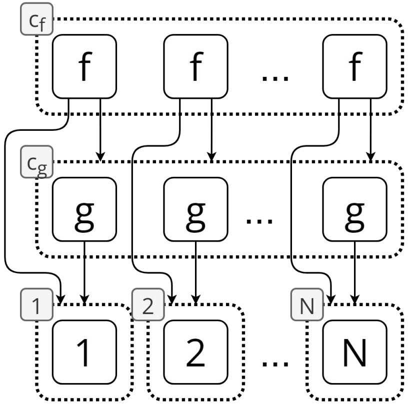


$$f(\alpha, g(\alpha))$$

```
for e-class c in e-graph E:
  for f-node n₁ in c:
    subst = {root ↦ c, α ↦ n₁.child₁}
```



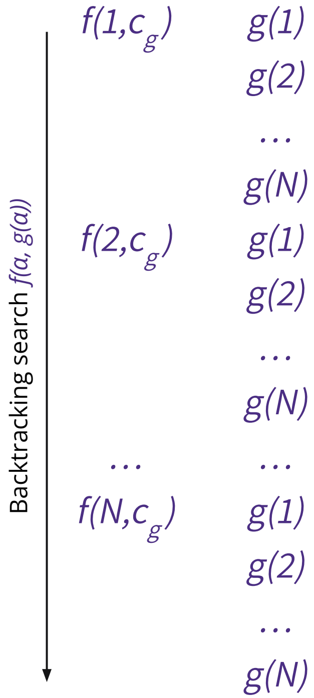
Traditional e-matching via backtracking



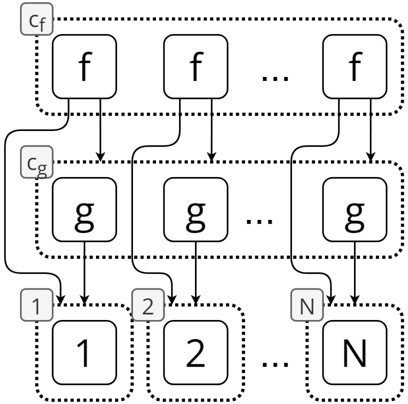
f(α, g(α))

⇩

```
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:
```

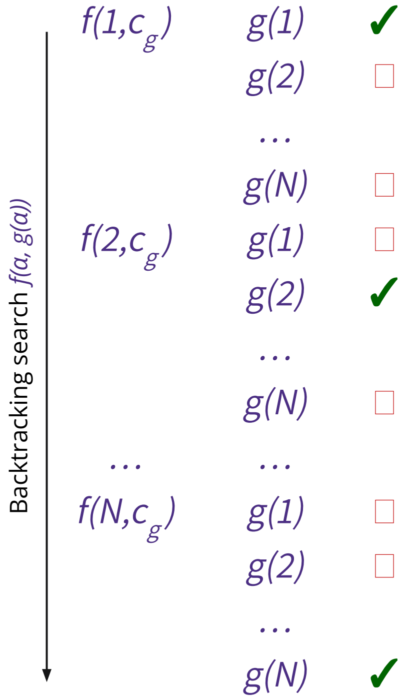


Traditional e-matching via backtracking

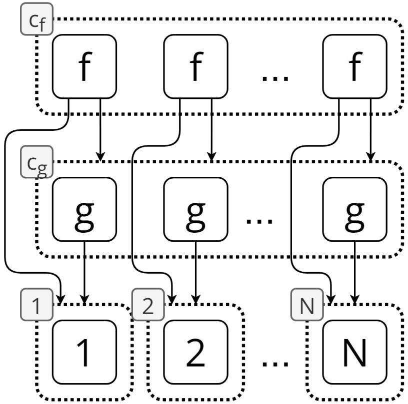


$$f(\alpha, g(\alpha))$$

```
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:
```



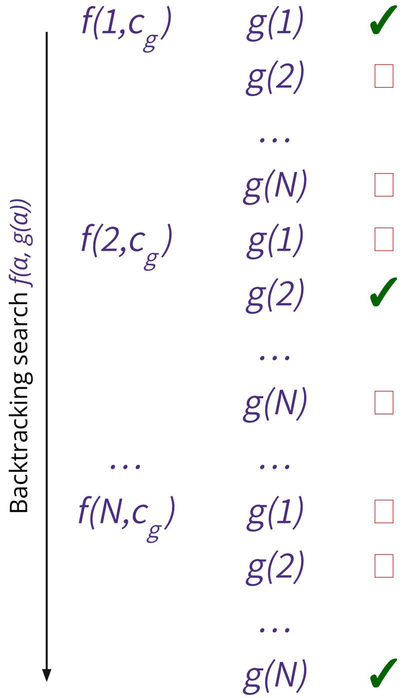
Traditional e-matching via backtracking



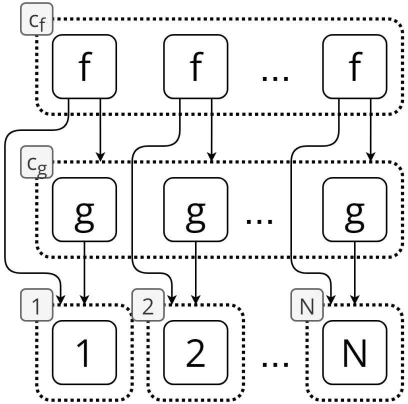
f(α, g(α))

⇩

```
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
```



Traditional e-matching via backtracking

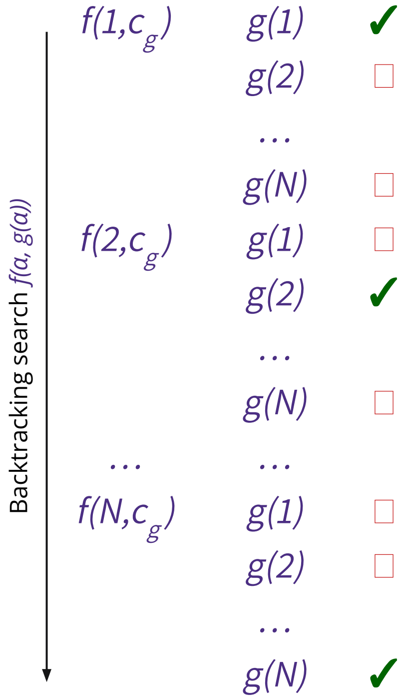


f(α, g(α))

⇓

```
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:
```

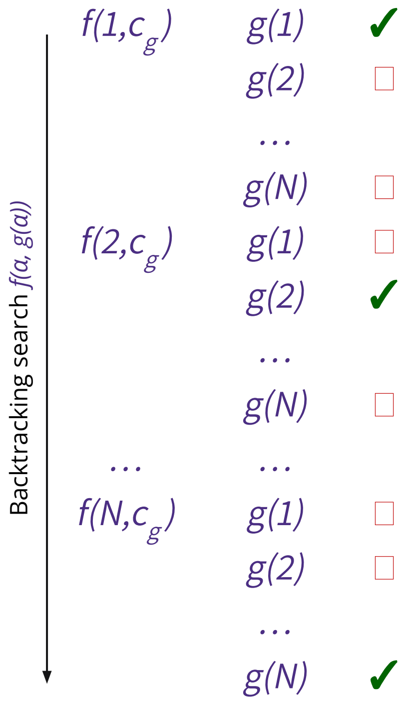
O(N^2), yet at most O(N) matches 😫



Traditional e-matching via backtracking

- Many optimizations in literature
  - custom VMs for “CSE”
  - specific patterns
  - mod-time analysis

- No data complexity bounds!



💡 Key insight: e-matching is a DB problem!

E-matching in e-graphs

Finding substitutions such that substituted terms are represented in an e-graph.

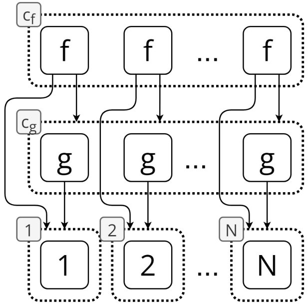
Conjunctive queries in DBs

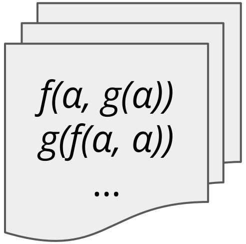
Finding substitutions such that substituted atoms are present in a relational DB.

egg’s relational e-matching

egg’s relational e-matching

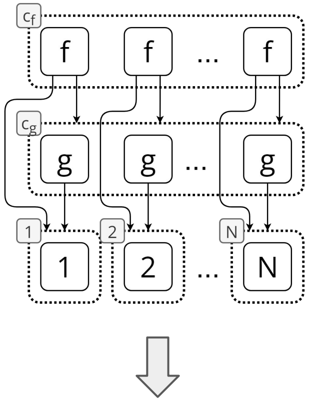
- Given e-graph + patterns





egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables

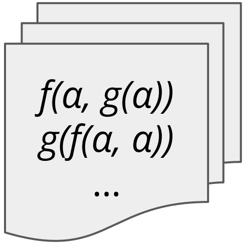


R_f

id	arg_1	arg_2
c_f	1	c_g
c_f	2	c_g
…	…	…
c_f	N	c_g

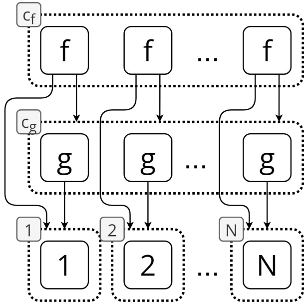
R_g

id	arg_1
c_g	1
c_g	2
…	…
c_g	N



egg's relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries

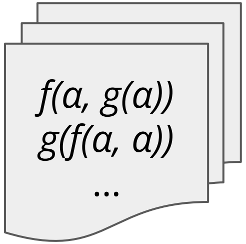


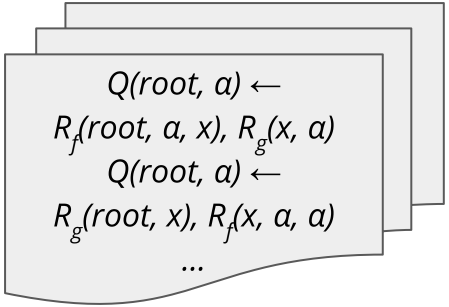
R_f

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

R_g

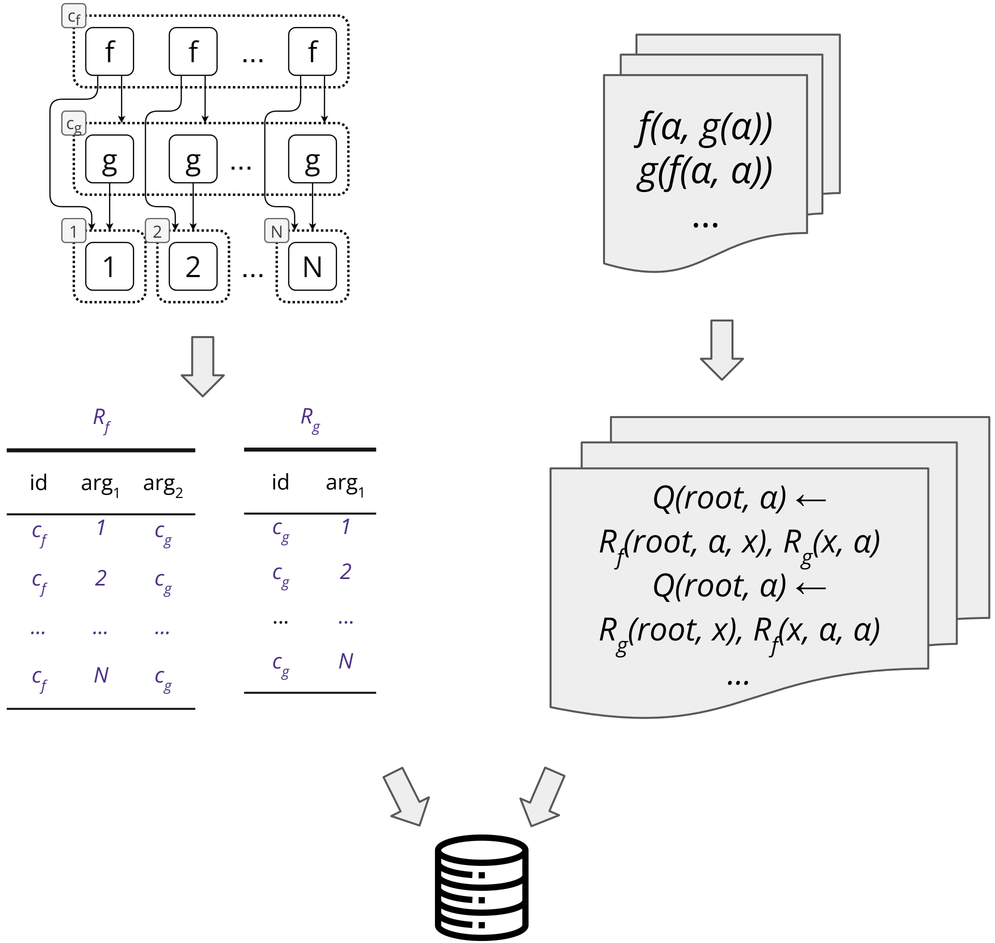
id	arg_1
c_g	1
c_g	2
...	...
c_g	N





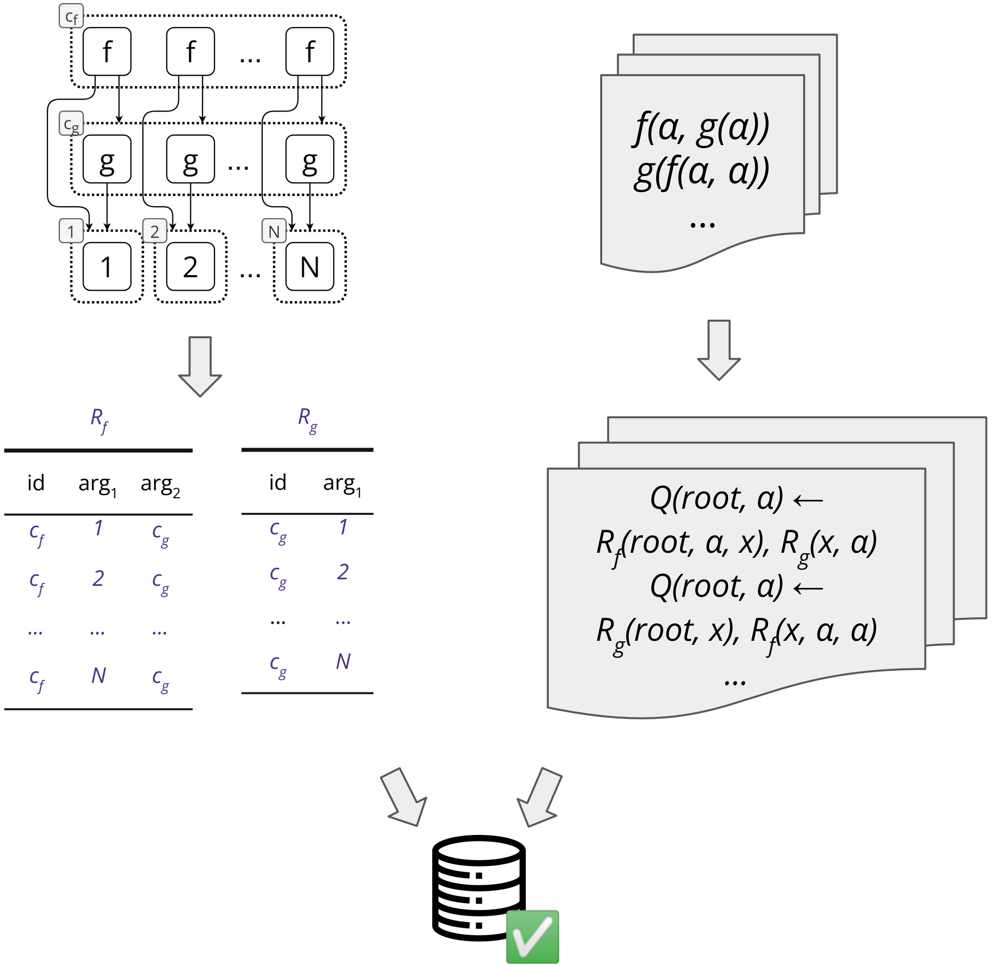
egg's relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries
- Use DB query engine to e-match!

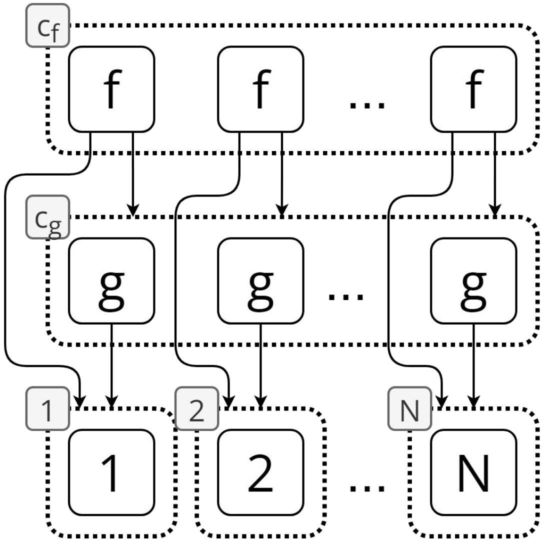


egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries
- Use DB query engine to e-match!
- Derive bounds from DB theory!



E-graphs as tables (relational DBs)



R_f

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

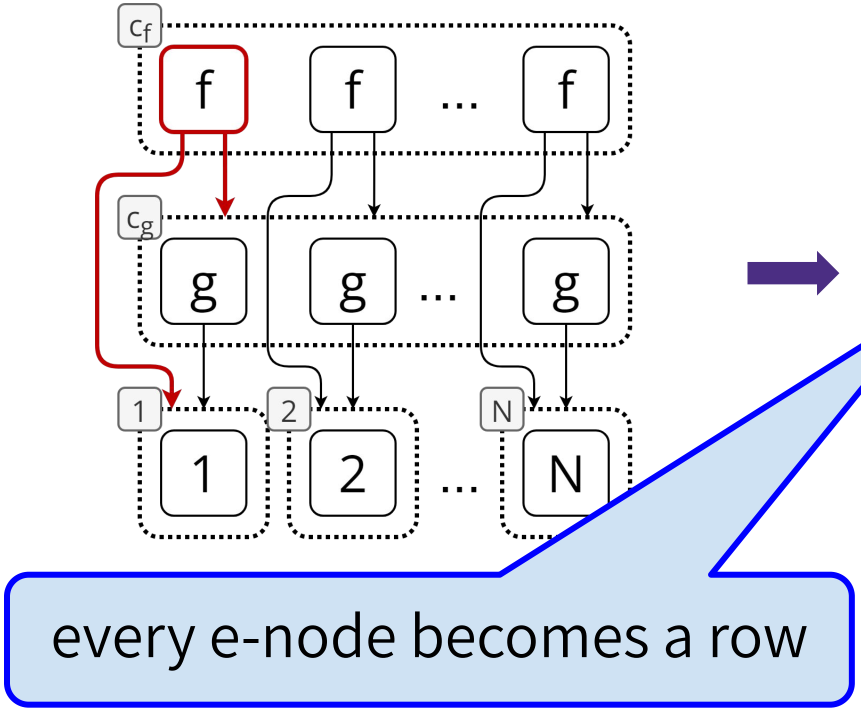
R_g

id	arg_1
c_g	*1*
c_g	*2*
…	…
c_g	*N*

$R_{i=1...N}$

id
i

E-graphs as tables (relational DBs)



R_f

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

R_g

id	arg_1
c_g	1
c_g	2
...	...
c_g	N

$R_{i=1...N}$

id
i

E-match patterns as conjunctive queries

$f(\alpha, g(\alpha))$

E-match patterns as conjunctive queries

$f(\alpha, g(\alpha))$

$$Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$$

E-match patterns as conjunctive queries

$f(\alpha, g(\alpha))$

$$Q(root, \alpha) \leftarrow$$
$$R_f(root, \alpha, x), R_g(x, \alpha)$$

```
ind = {}
for (x, α) in R_g: # build index
  ind.insert((x, α))
```

build hash

$R_g(c_g, 1)$
$R_g(c_g, 2)$
...
$R_g(c_g, N)$

E-match patterns as conjunctive queries

$f(\alpha, g(\alpha))$

$$Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$$

```
ind = {}
for (x, α) in R_g: # build index
  ind.insert((x, α))
for (root, α, x) in R_f: # probe
  if (α, x) in ind:
    yield {root ↦ root, α ↦ α}
```

build hash

$R_g(c_g, 1)$
$R_g(c_g, 2)$
…
$R_g(c_g, N)$

probe

$R_f(c_f, 1, c_f)$ ✓
$R_f(c_f, 2, c_f)$ ✓
…
$R_f(c_f, N, c_f)$ ✓

Why is relational e-matching faster?

$f(\alpha, g(\alpha))$

Enum all terms of shape $f(\alpha, g(\beta))$

Check if $\alpha = \beta$ only before yielding

$Q(root, \alpha) \leftarrow$
$R_f(root, \alpha, x), R_g(x, \alpha)$

Build indices on both α and x.

Only enum terms where constraints on both x and α are satisfied.

structural constraints

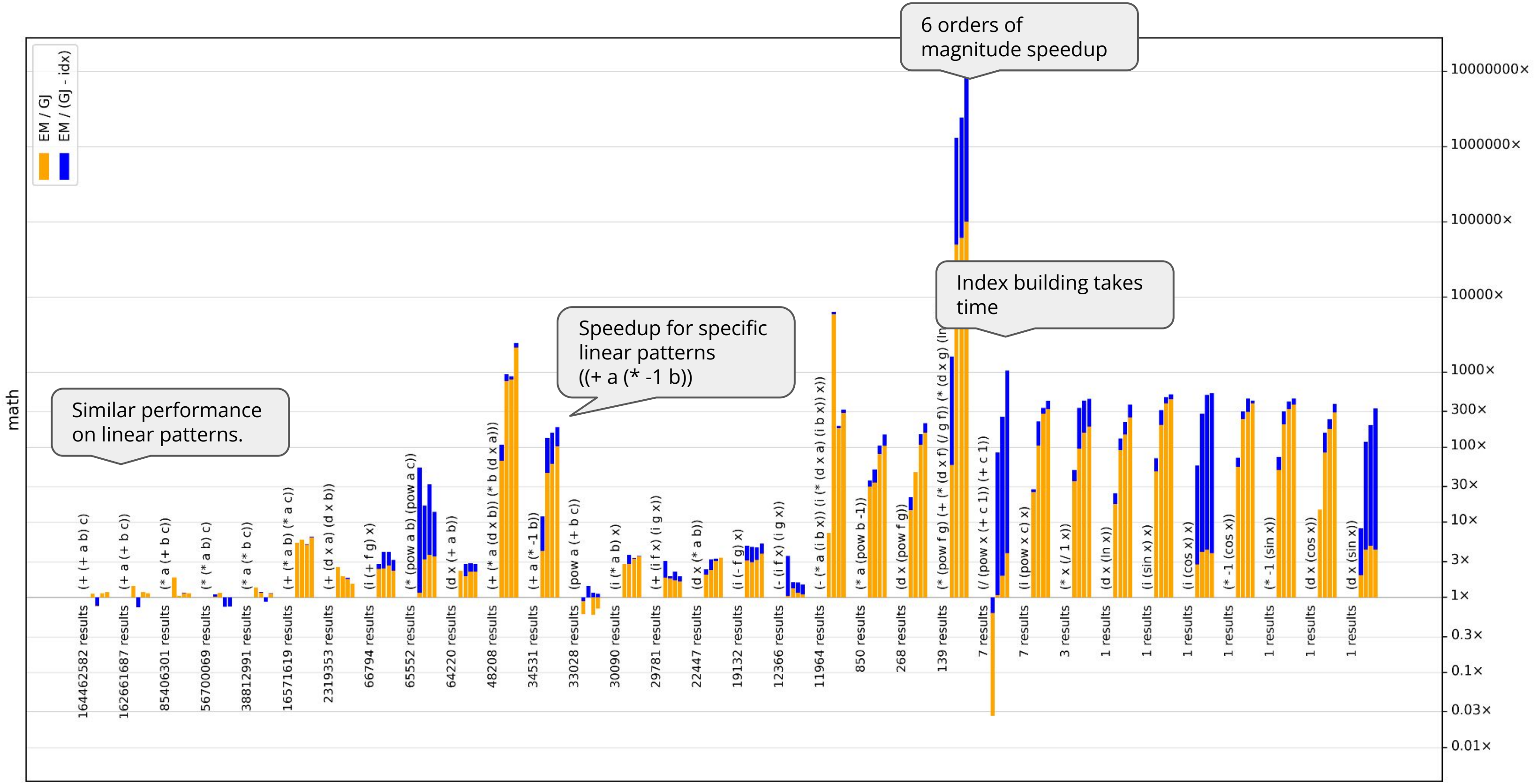
equality constraints

Data complexity results (see paper)

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 $O(\max_E(|M(p, E)|))$.*

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 $O\left(\sqrt{|Q(I)| \times \Pi_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$.*

Relational e-matching : asymptotic speedup



New Capabilities: *Multi-patterns*

```
x = matmul(a, b),
y = matmul(a, c)
```

➡

```
x = split1(matmul(a,concat(b, c))),
y = split2(matmul(a,concat(b, c)))
```

New Capabilities: *Multi-patterns*

```
x = matmul(a, b),
y = matmul(a, c)
```

search for two patterns *anywhere* in the e-graph

➡

```
x = split1(matmul(a,concat(b, c))),
y = split2(matmul(a,concat(b, c)))
```

New Capabilities: *Multi-patterns*

```
x = matmul(a, b),
y = matmul(a, c)
```

search for two patterns *anywhere* in the e-graph

➡

```
x = split1(matmul(a,concat(b, c))),
y = split2(matmul(a,concat(b, c)))
```

perform two merges, each on a separate e-class!

egg EqSat Toolkit [POPL 2021, Distinguished Paper]

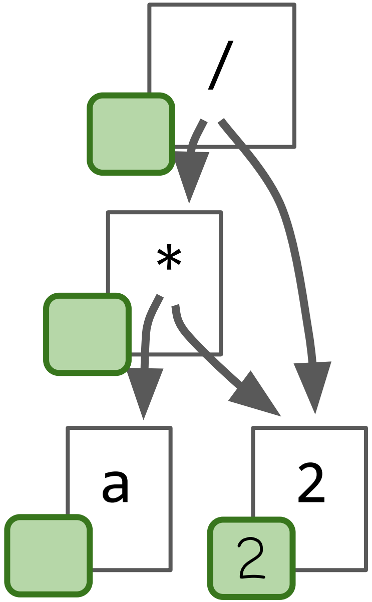
- ✓ Deferred invariant maintenance & batching
- ✓ Relational e-matching [POPL 2022]
- ❏ E-class analyses
- ❏ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ❏ Applications
  - ❏ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ❏ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …

Syntactic rewriting is not enough…

- How many rules do we need for constant folding?
  - `2 + 2 → 4,  3 + 4 → 6,  4 + 6 → 10, …` *a lot!*
- What about satisfying guards for conditional rules?
  - `x / x → 1` only ok if `x <> 0`
- In general, many optimizations depend on analyses!
  - nullability, tensor shape, intervals, free variables, …

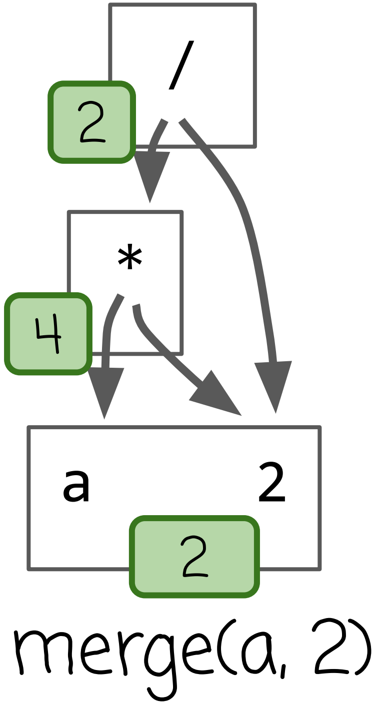
Constant folding

- 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!



E-class analyses

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

E-class analysis invariant

for each e-class

fixed point

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

Analysis data is LUB
(lattice properties)

Program analysis modulo equivalence

- Tightest summary over all equivalent represented terms!

To demonstrate an advantage of this approach, consider the following example, for $x \in [0, 1], y \in [1, 2]$, where the following concrete-equivalences are discovered via rewriting:

$$1 - \frac{2y}{x+y} \qquad \in \left[-3, \frac{1}{3}\right] \qquad (5)$$

$$\cong \frac{x-y}{x+y} \qquad \in [-2, 0] \qquad (6)$$

$$\cong \frac{2x}{x+y} - 1 \qquad \in [-1, 1]. \qquad (7)$$

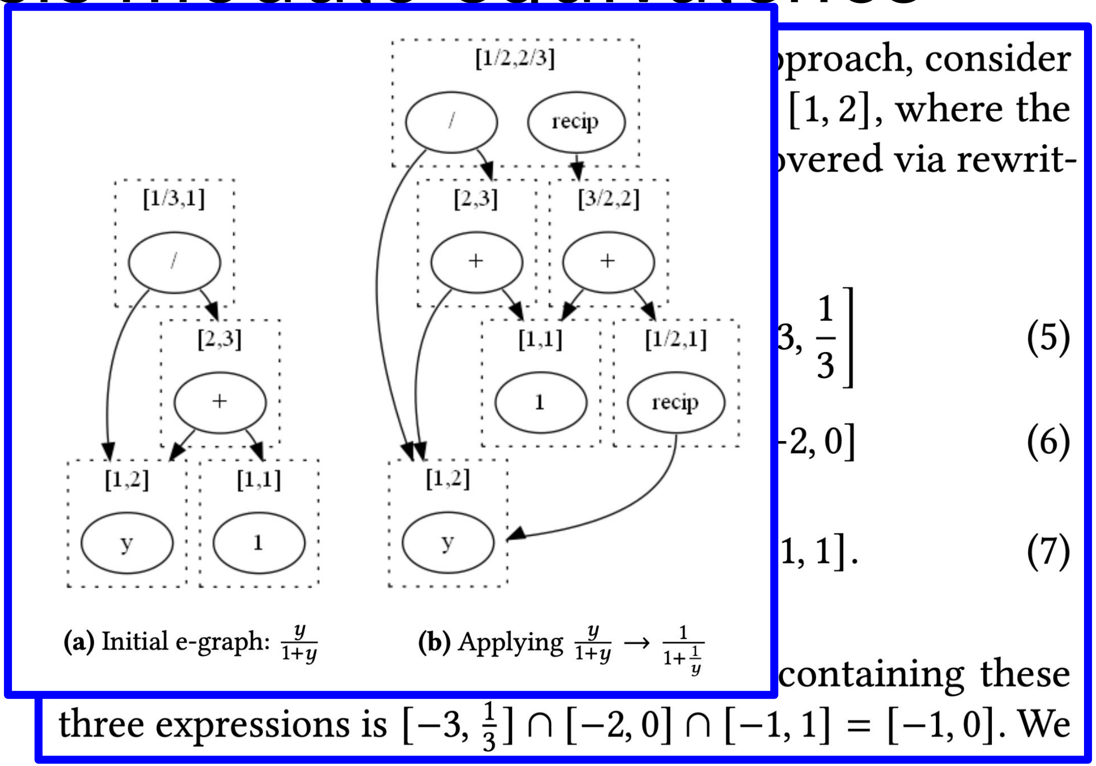
The interval associated with the e-class containing these three expressions is $[-3, \frac{1}{3}] \cap [-2, 0] \cap [-1, 1] = [-1, 0]$. We

Sam Coward et al. (2022)

Program analysis modulo equivalence

- Tightest summary over all equivalent represented terms!

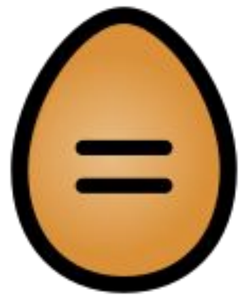
- *Virtuous cycle:*
  facts enable rewrites,
  rewrites improve facts!



Sam Coward et al. (2022)

egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- ✓ Deferred invariant maintenance & batching
- ✓ Relational e-matching [POPL 2022]
- ✓ E-class analyses
- ❑ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ❑ Applications
  - ❑ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ❑ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …

🐓 & 

🐓 & 🥚

- EqSat and egg can only be as good as user's rules…

🐓 & 🥚

- EqSat and egg can only be as good as user's rules…

Where do rules come from?

- Typically hand written by experts
- Time consuming, often takes years
- Too few / too many / unsound rules

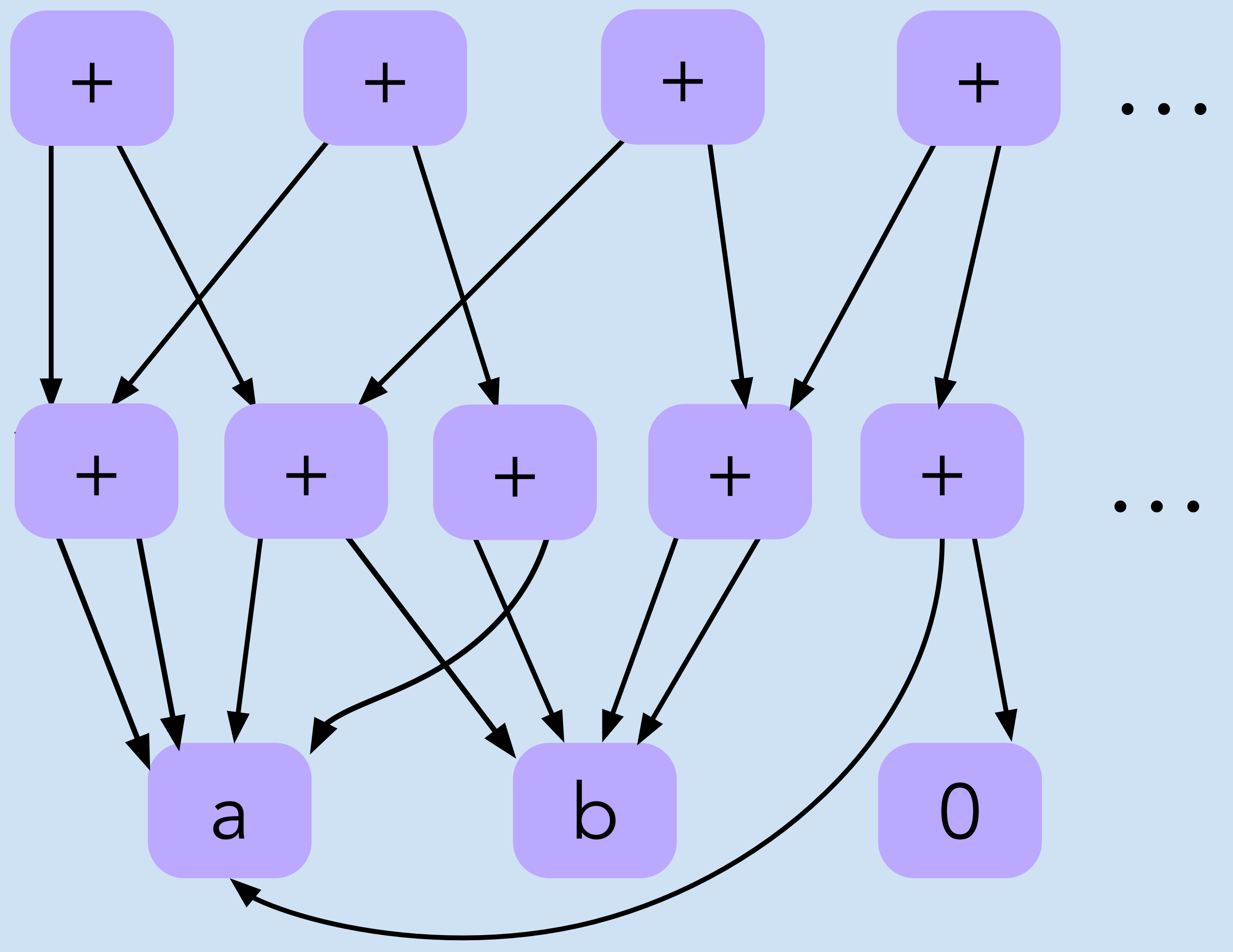
A 3-step approach for inferring rewrite rules

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

a, b, 0, +, …

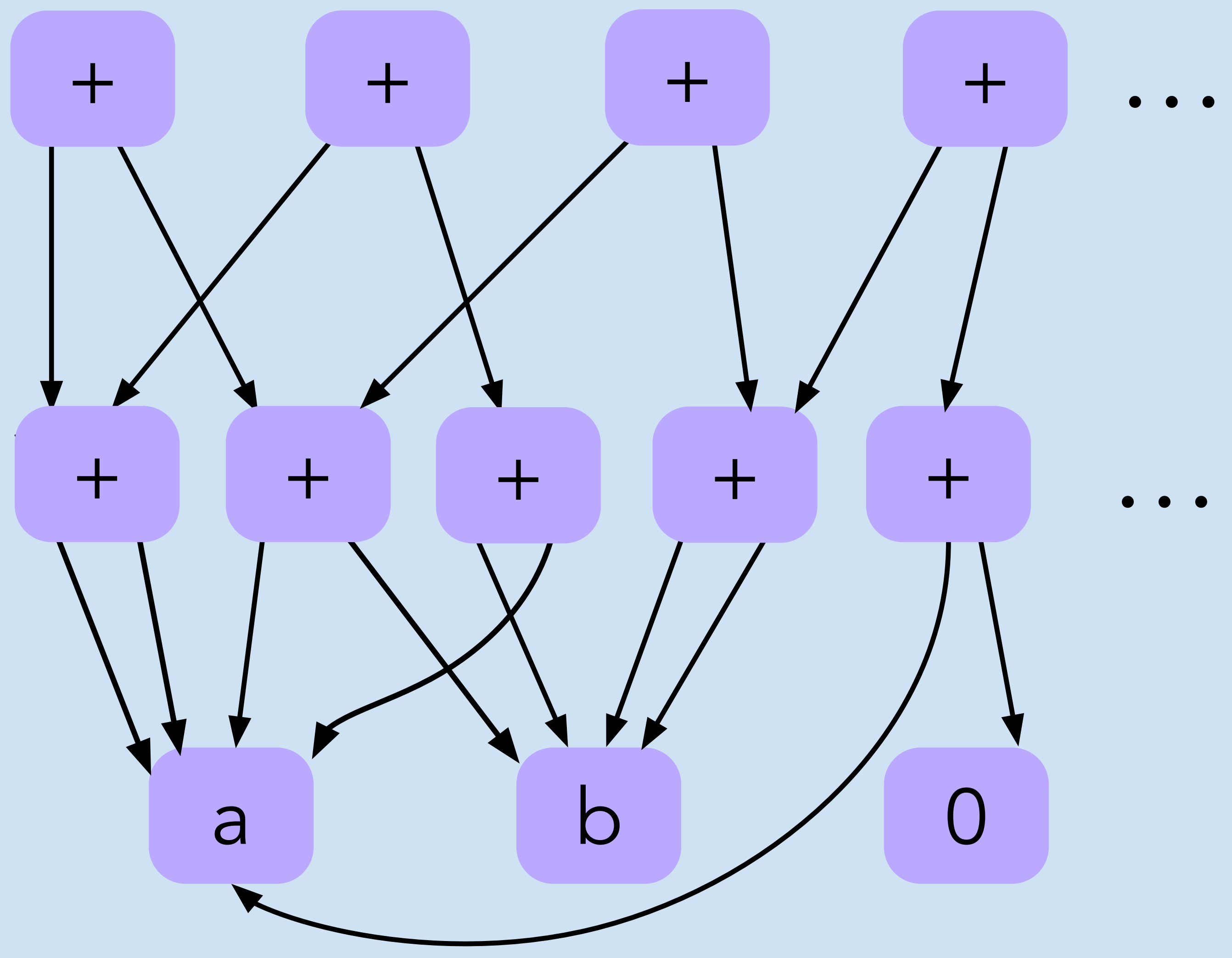


Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

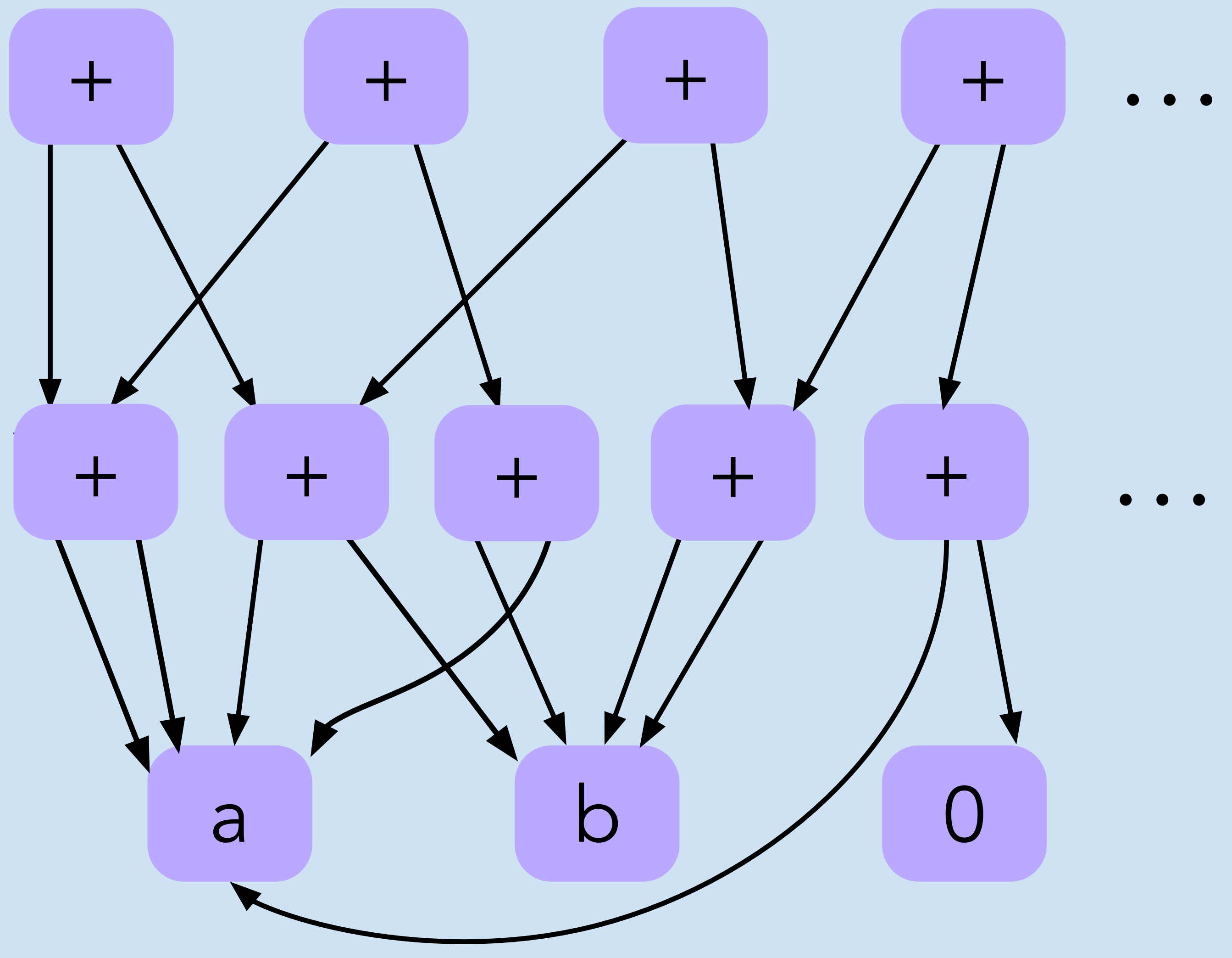
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”

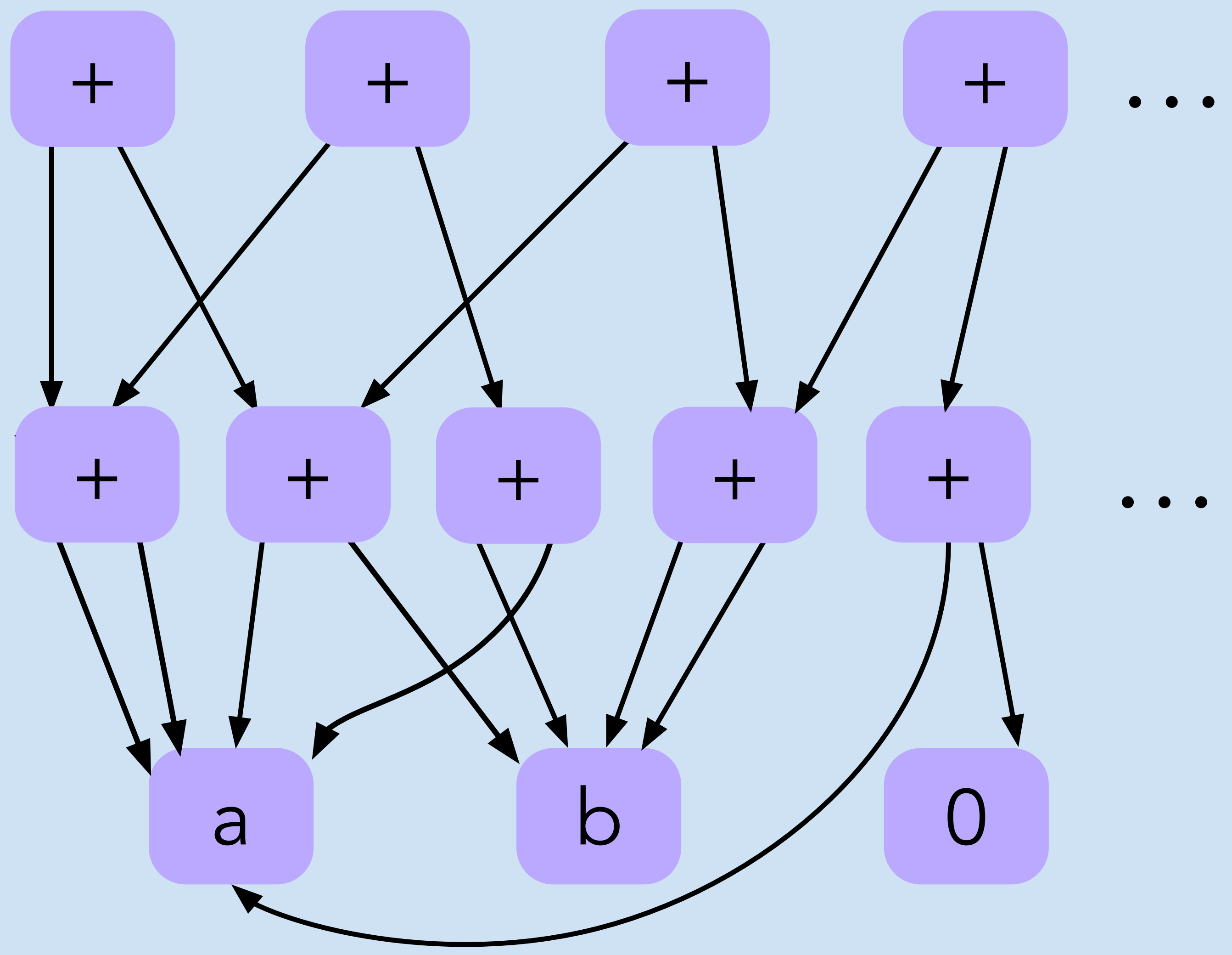


Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

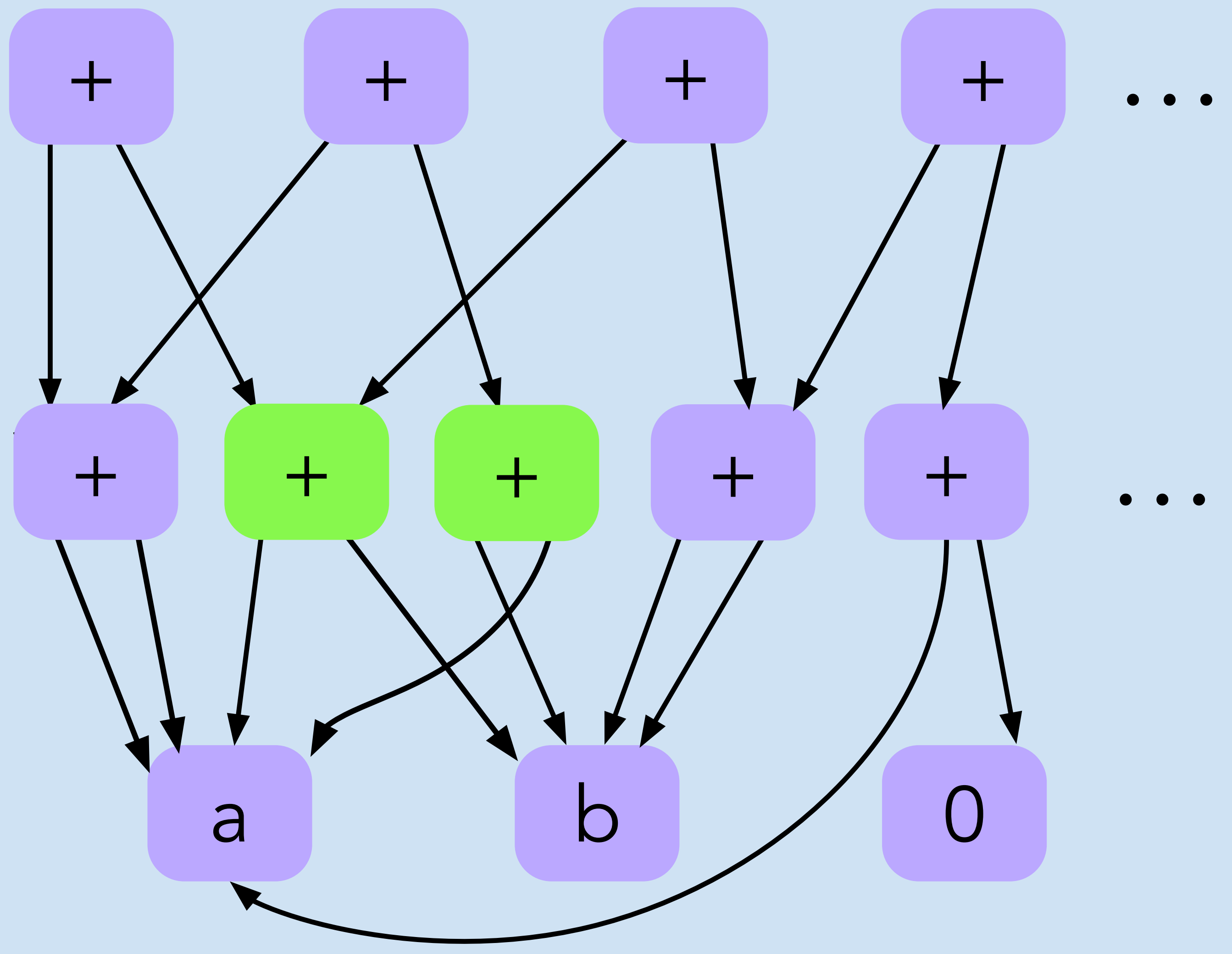
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”



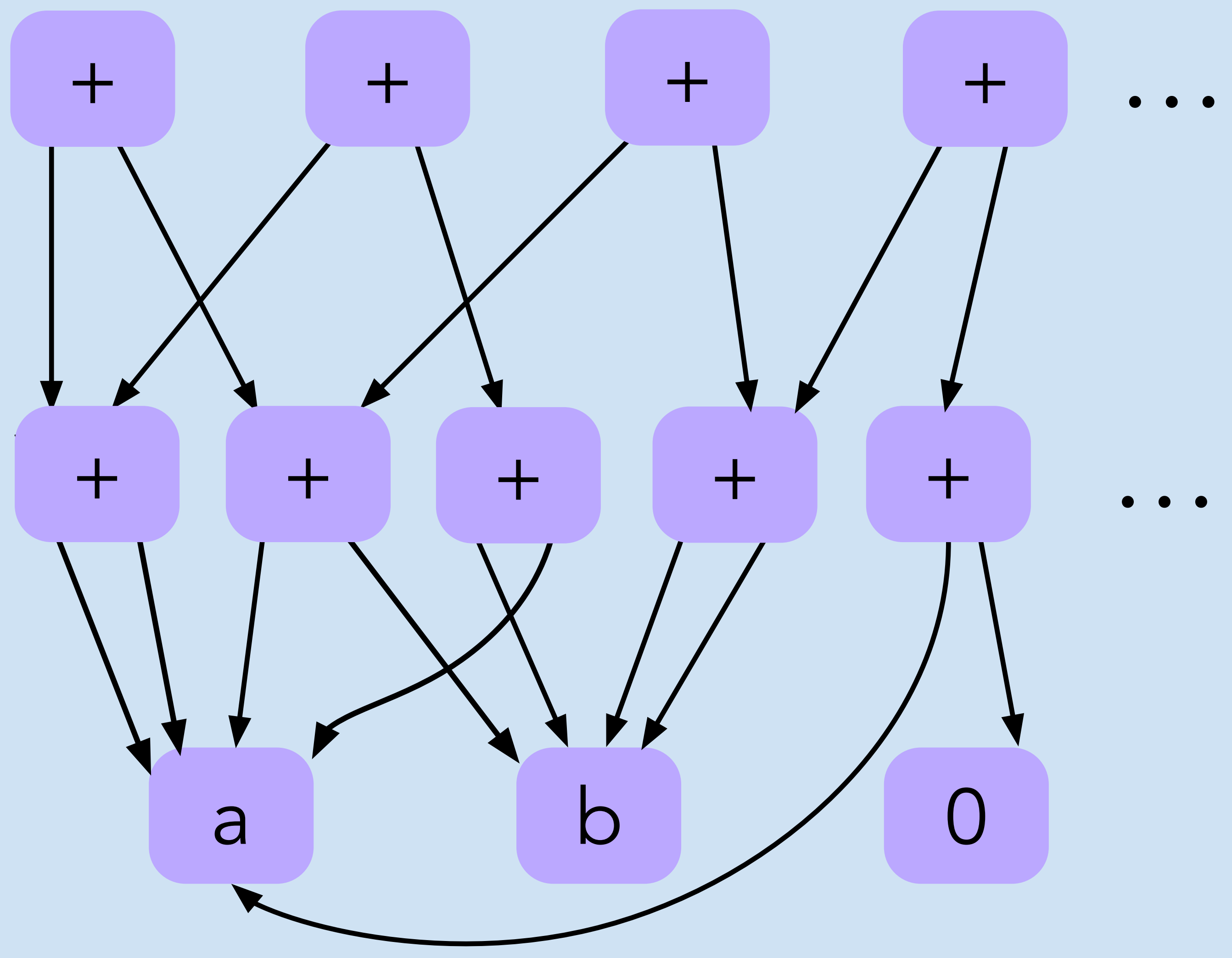
(x + y) ↔ (y + x)

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

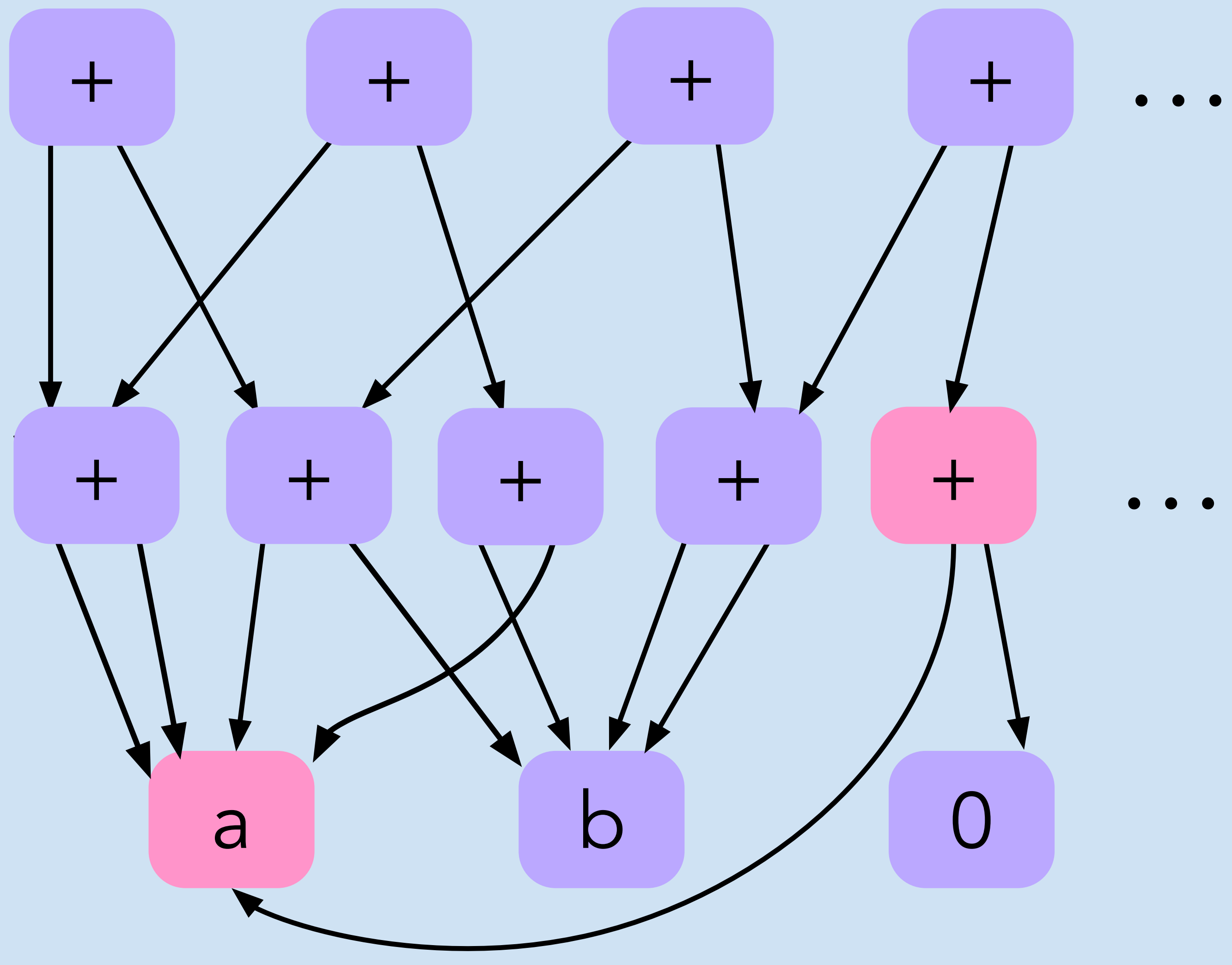
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

"Fingerprints"



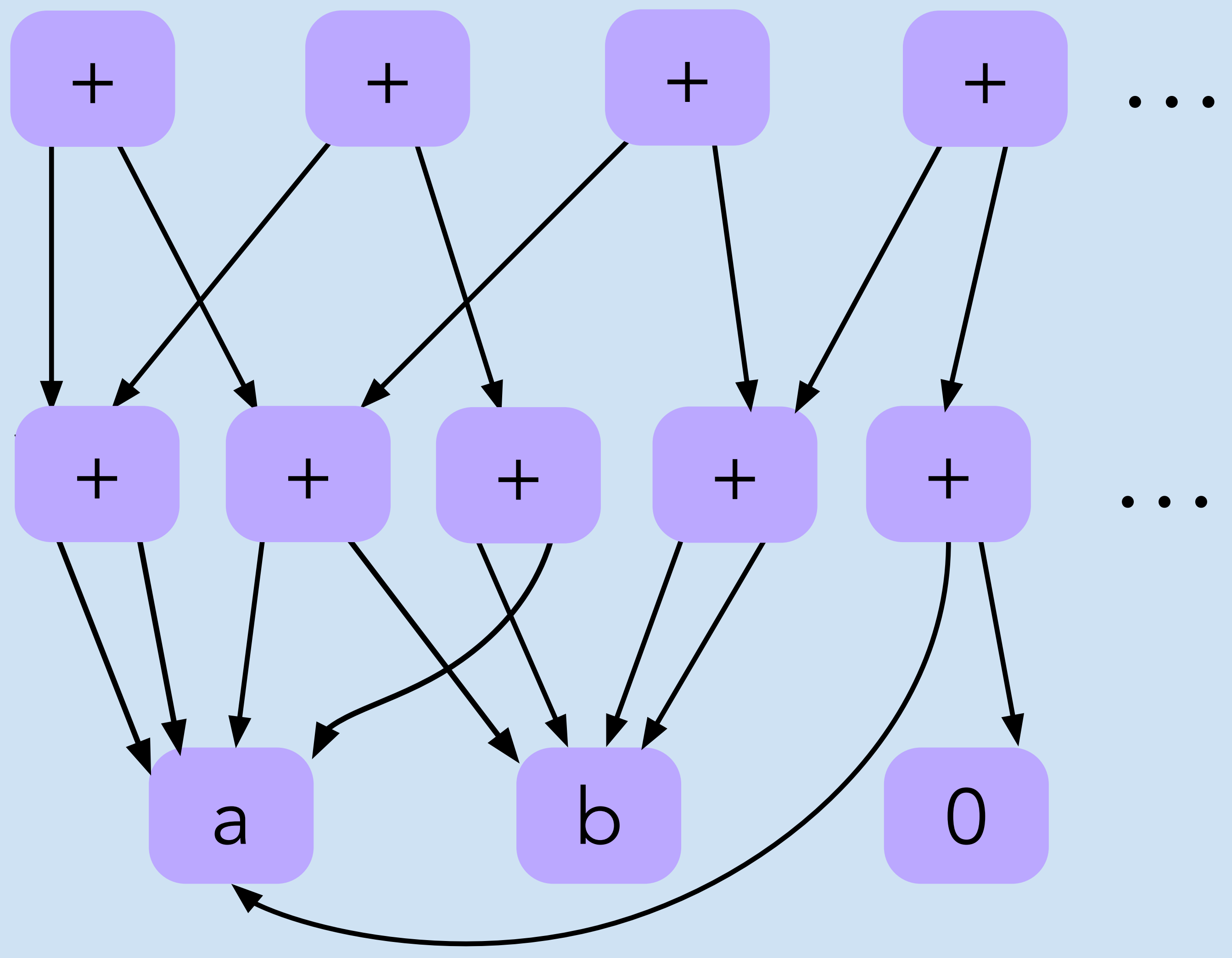
(x + 0) ⟷ x

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

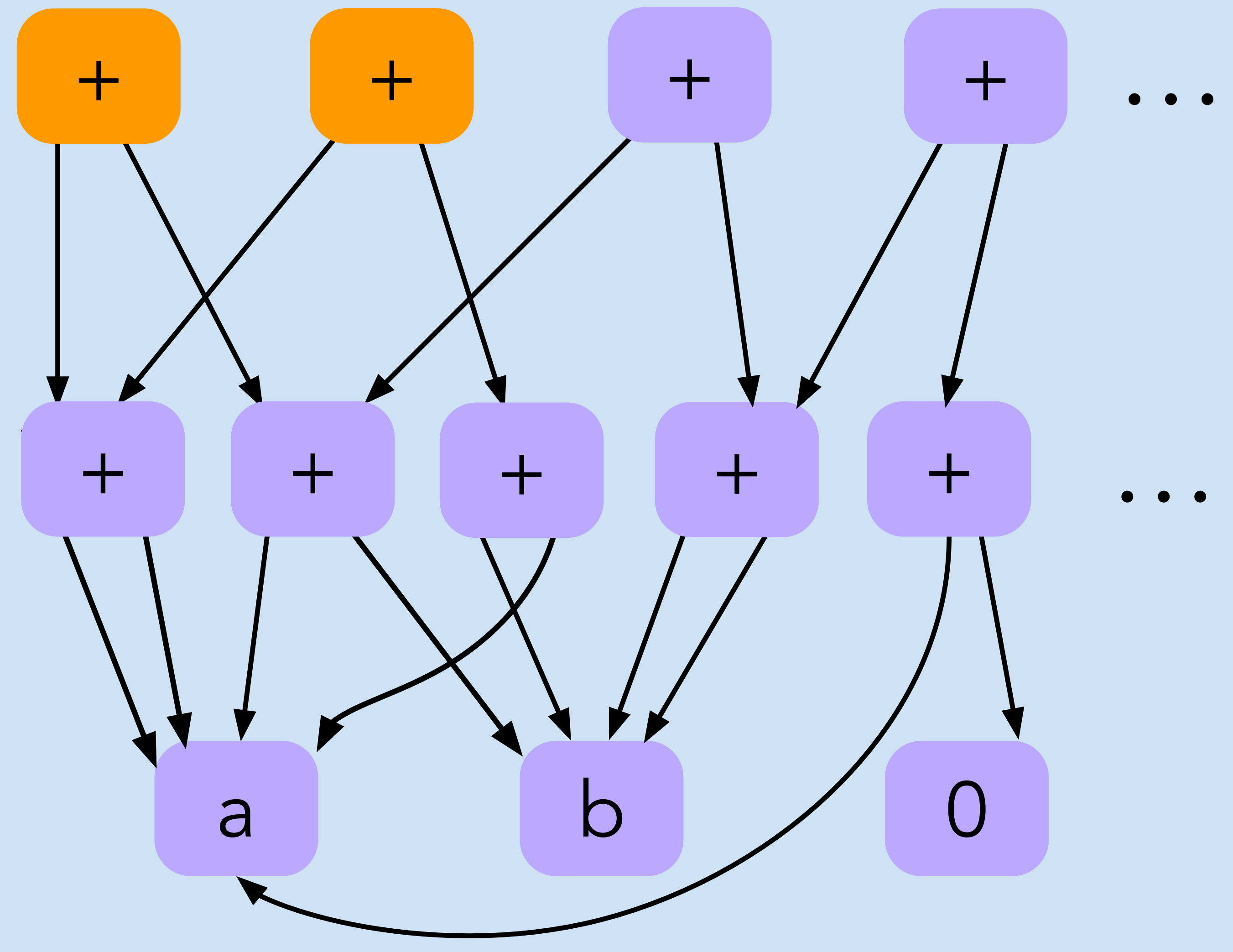
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”



(x + x) + (x + y)

↕

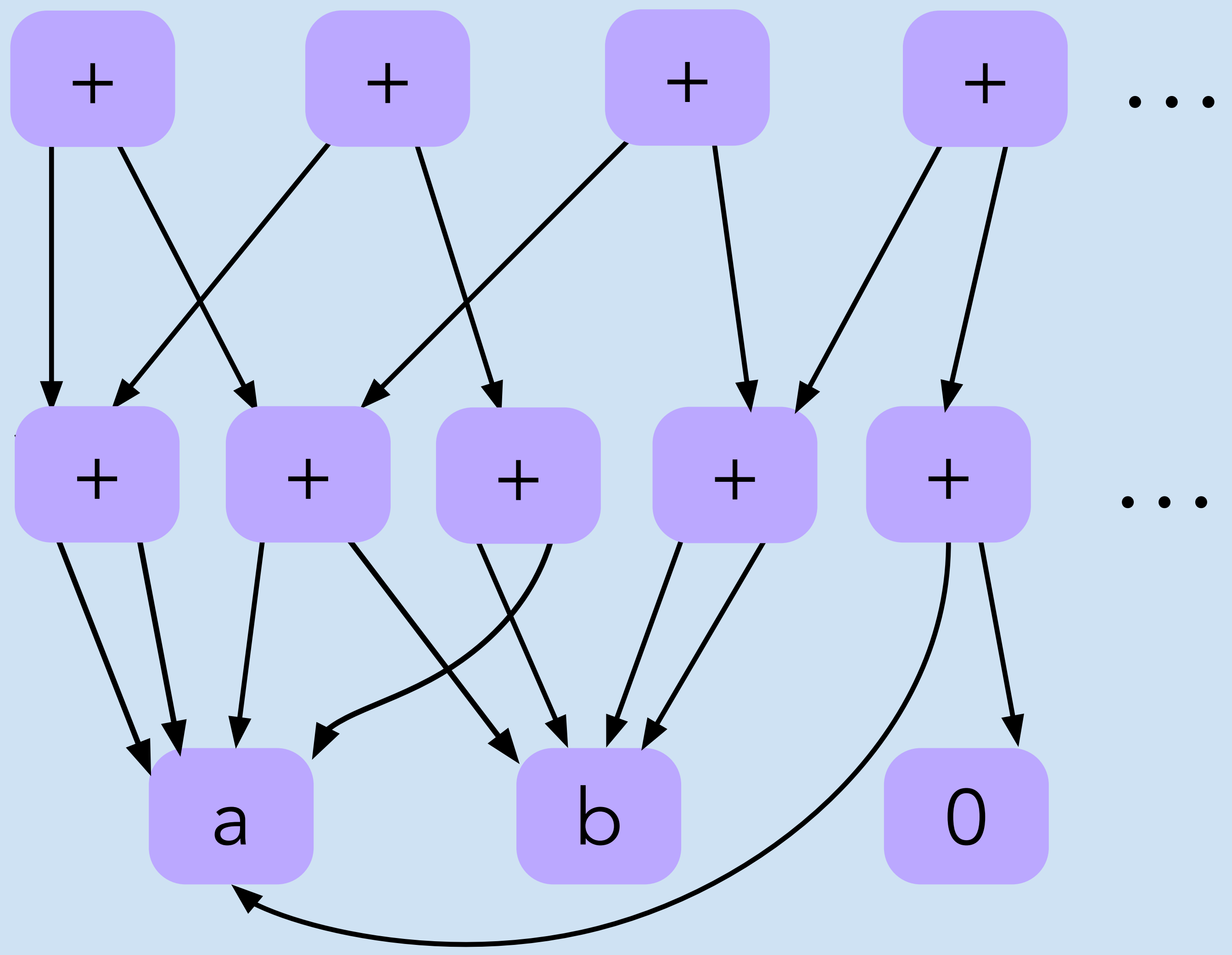
(x + x) + (y + x)

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

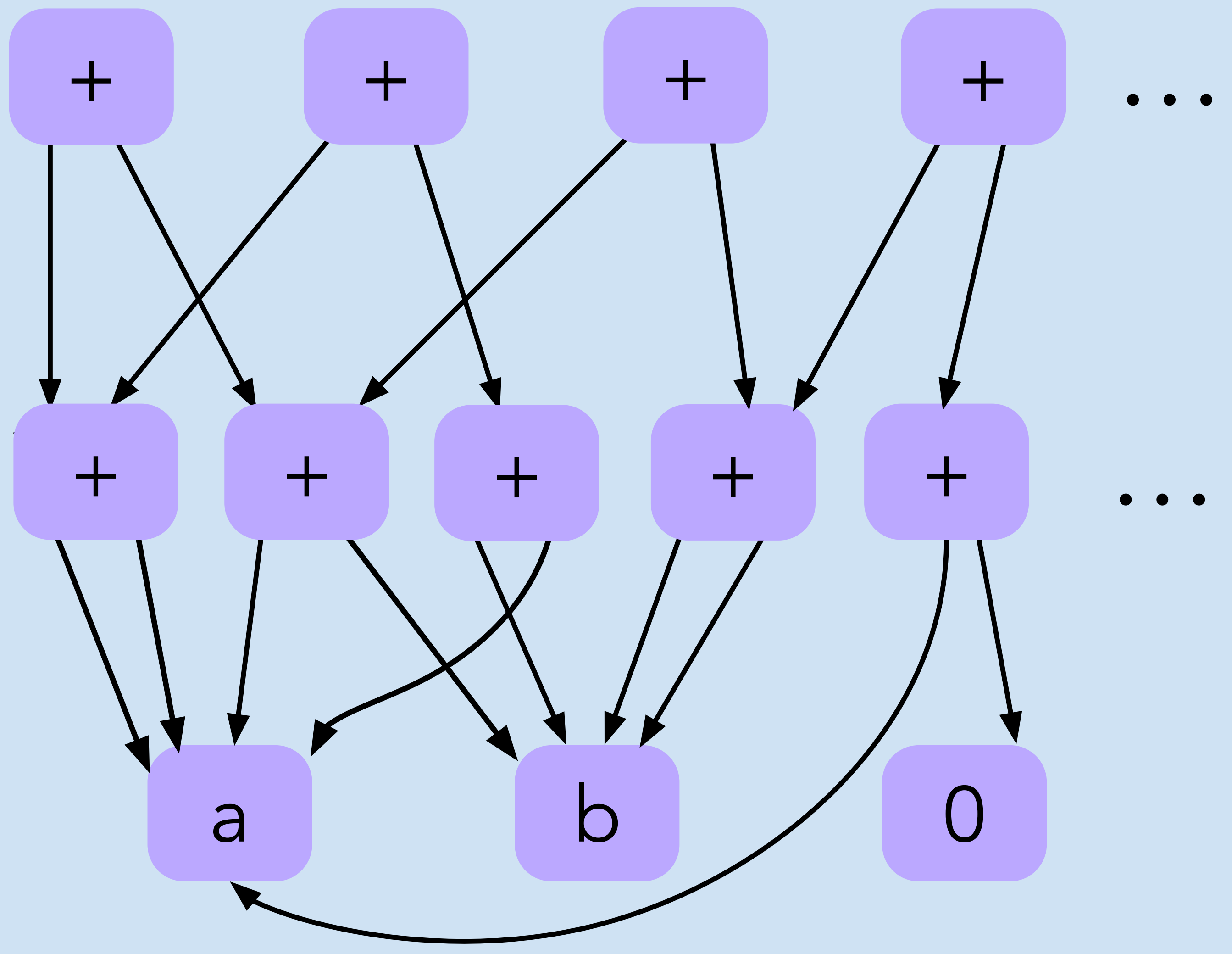
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

 "Fingerprints"



Filter candidates to get final ruleset

Remove redundant rules

x + 0 ⟷ 0 + x

y + 0 ⟷ 0 + y

x + y ⟷ y + x

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

Exponentially many terms!

Find candidates: interpret over concrete inputs

Too many candidates, some potentially unsound!

Filter candidates to get final ruleset

Hard to find a small, useful ruleset

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

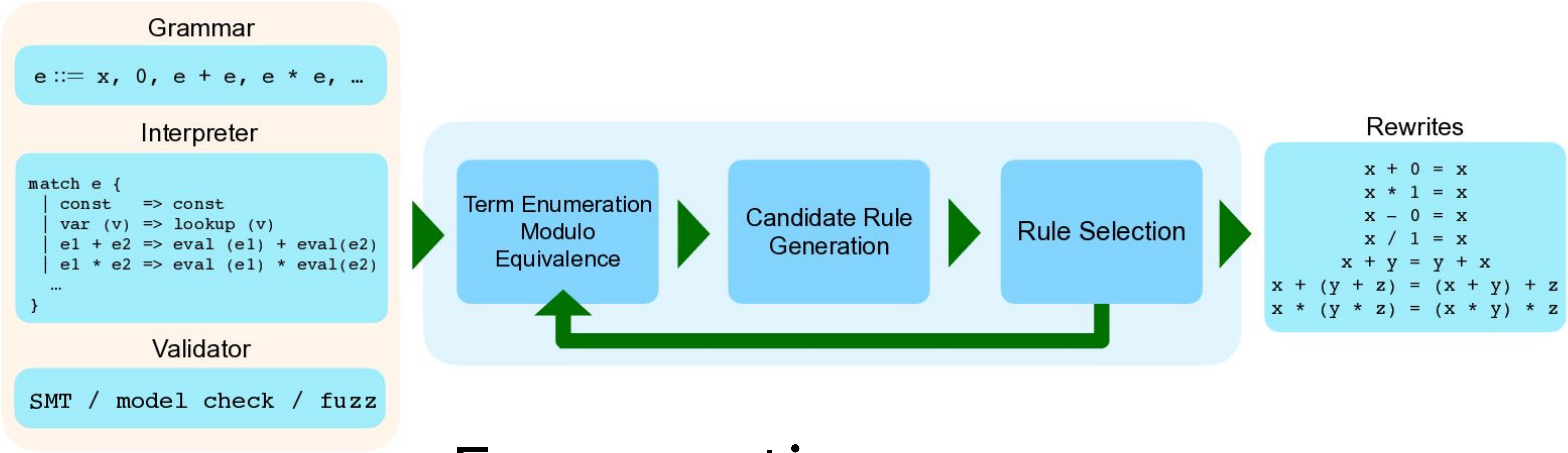
A 3-step approach for inferring rewrite rules

Inferring ***Small, Useful*** Rulesets ***Faster*** using Equality Saturation!

Equality Saturation for not just applying rewrites, but also ***inferring*** them!

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

Ruler

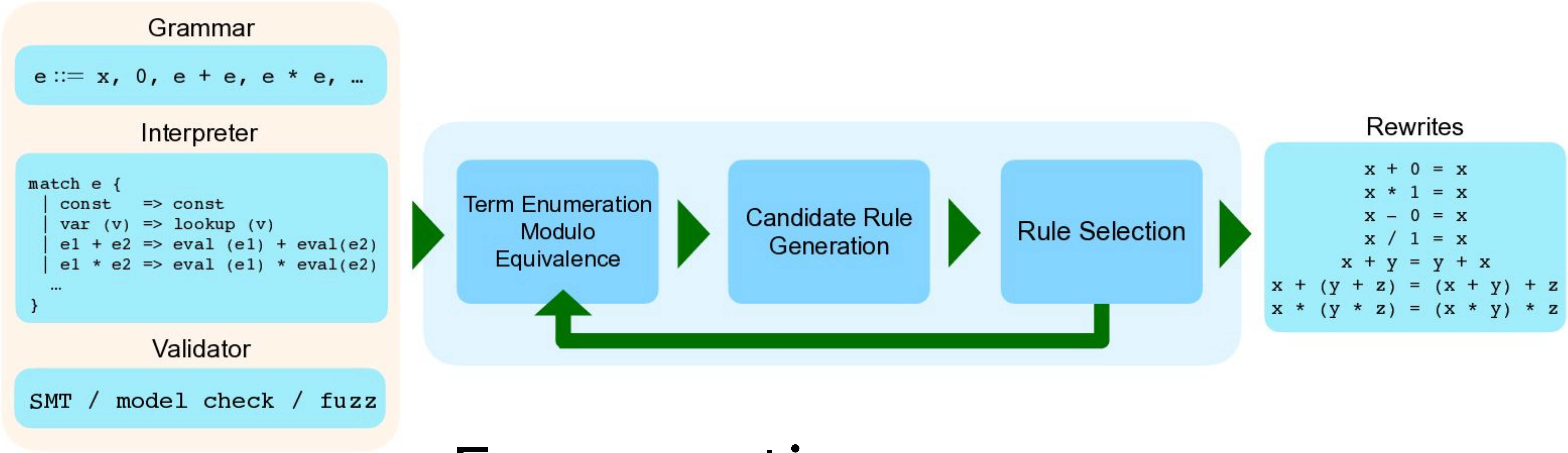


Enumeration

Candidate Generation

Rule Selection

Ruler



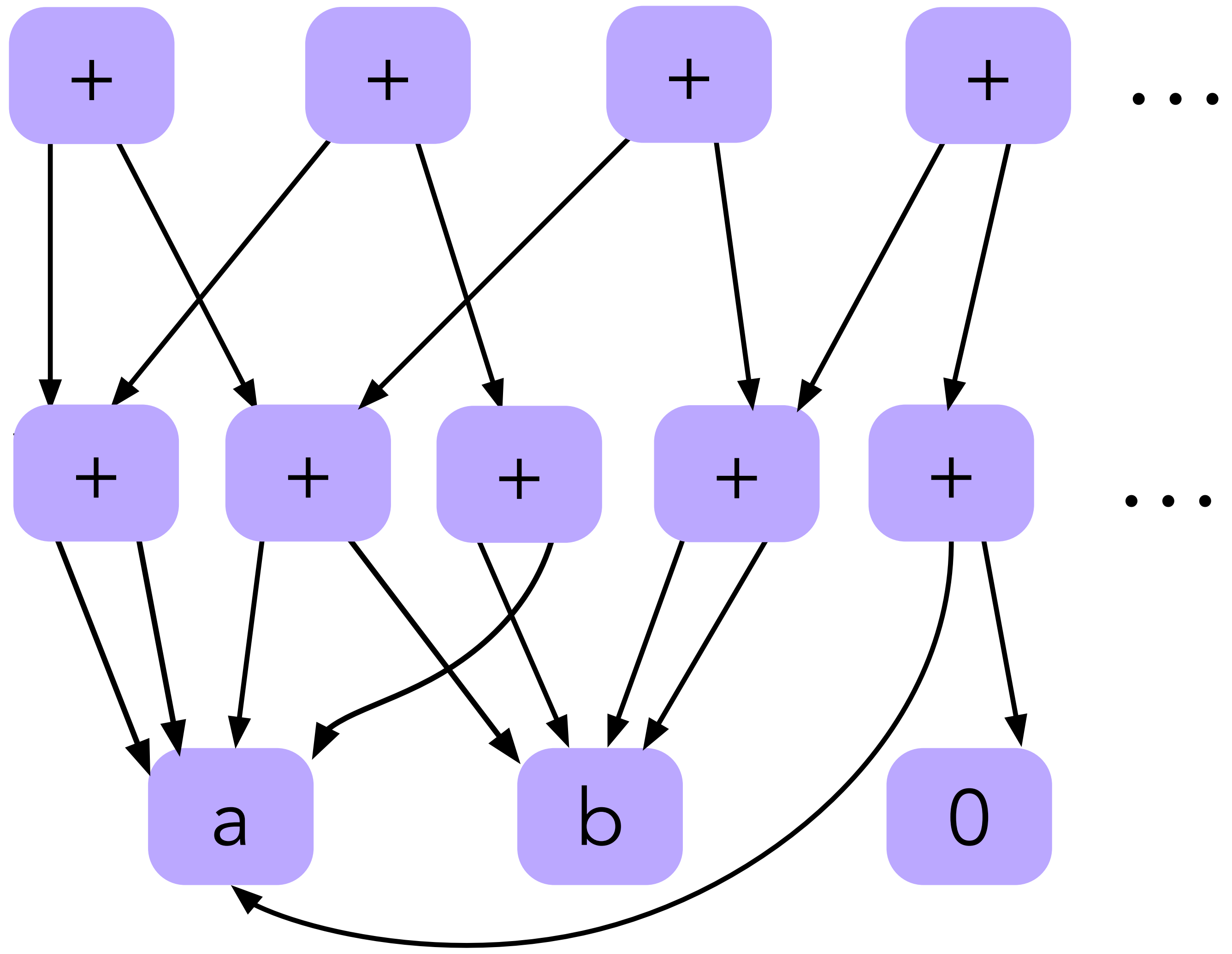
Enumeration

Candidate Generation

Rule Selection

Enumeration modulo equality saturation

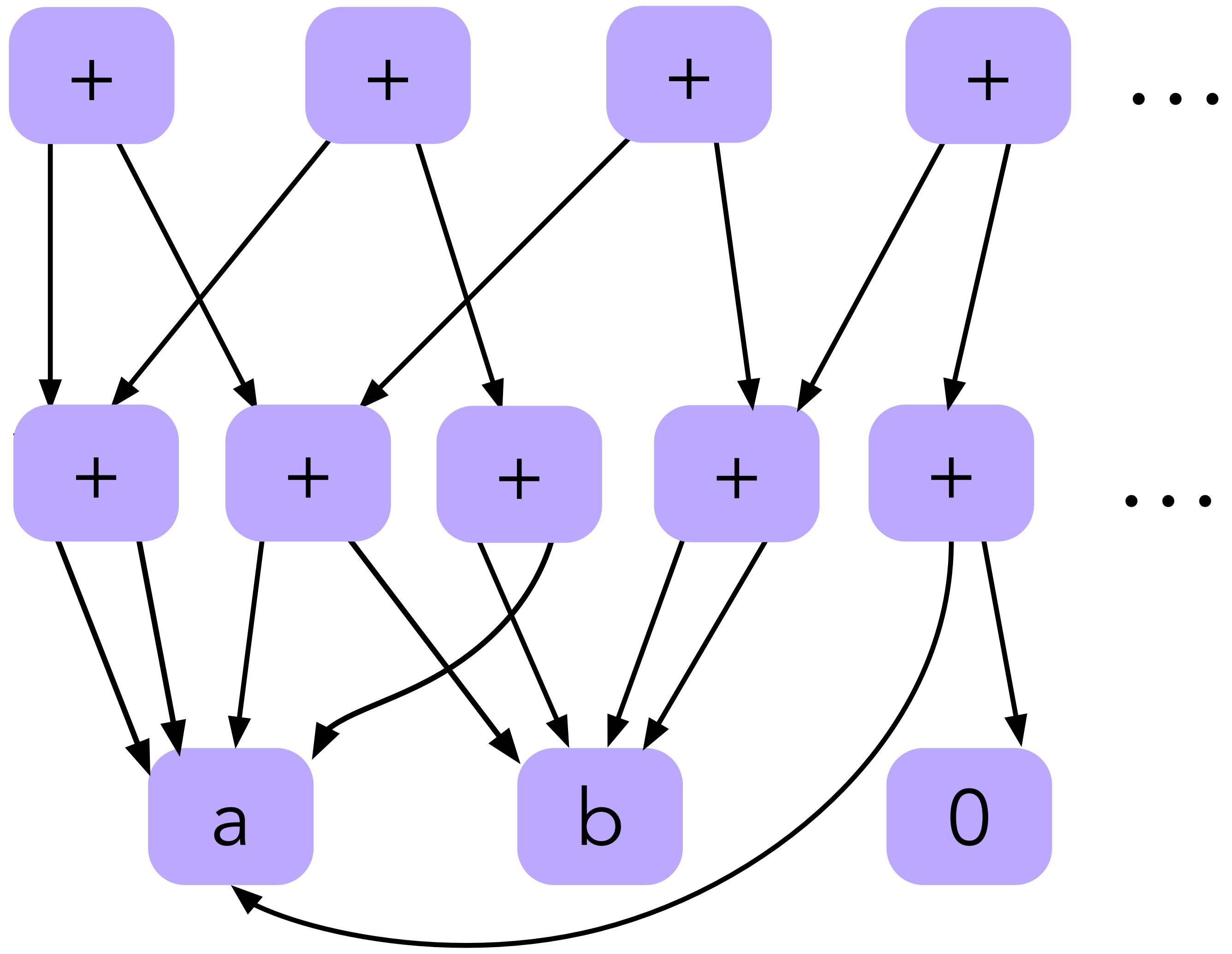
a, b, 0, +, …



Exponentially many terms!

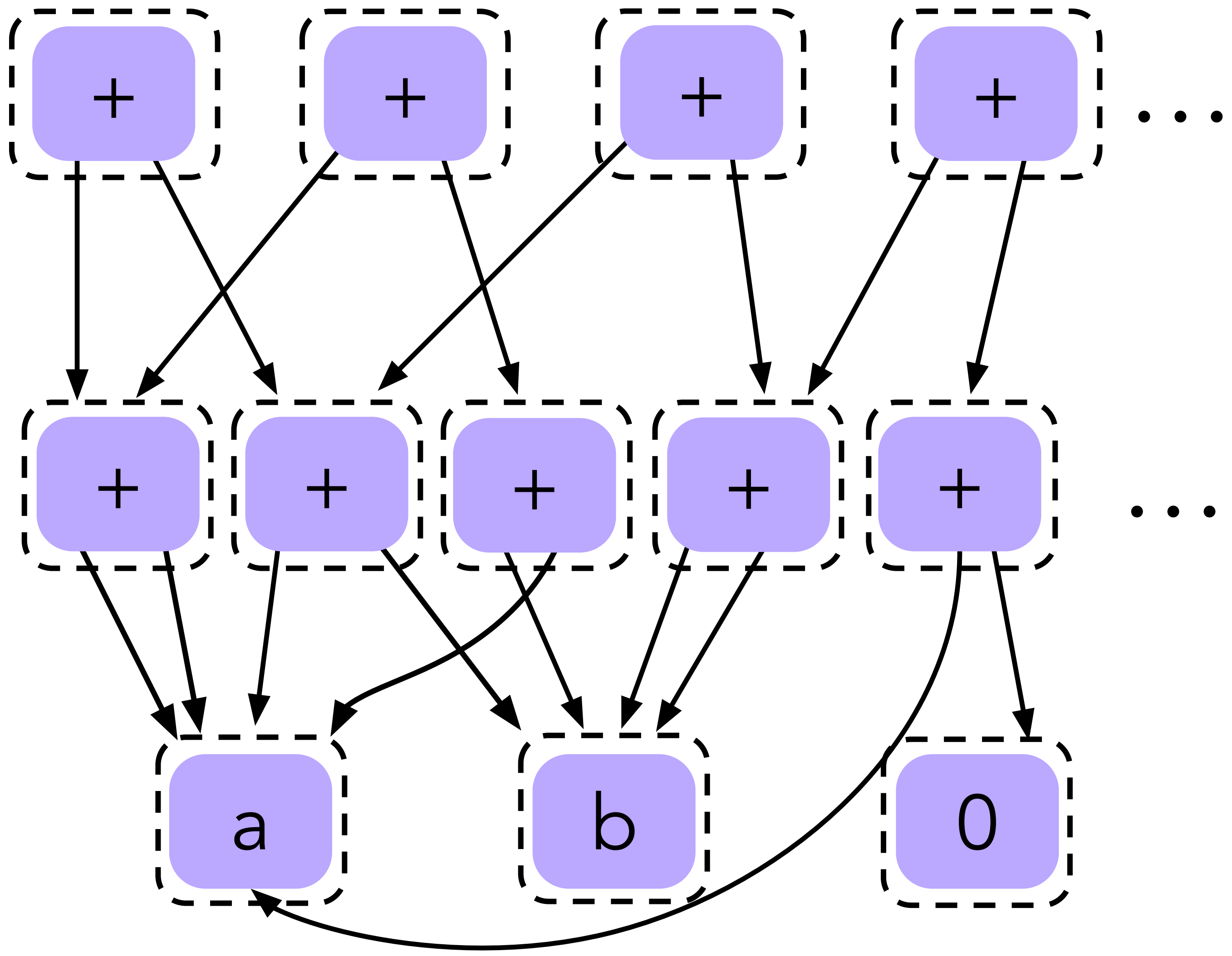
Enumeration modulo equality saturation

a, b, 0, +, …



Exponentially many terms!

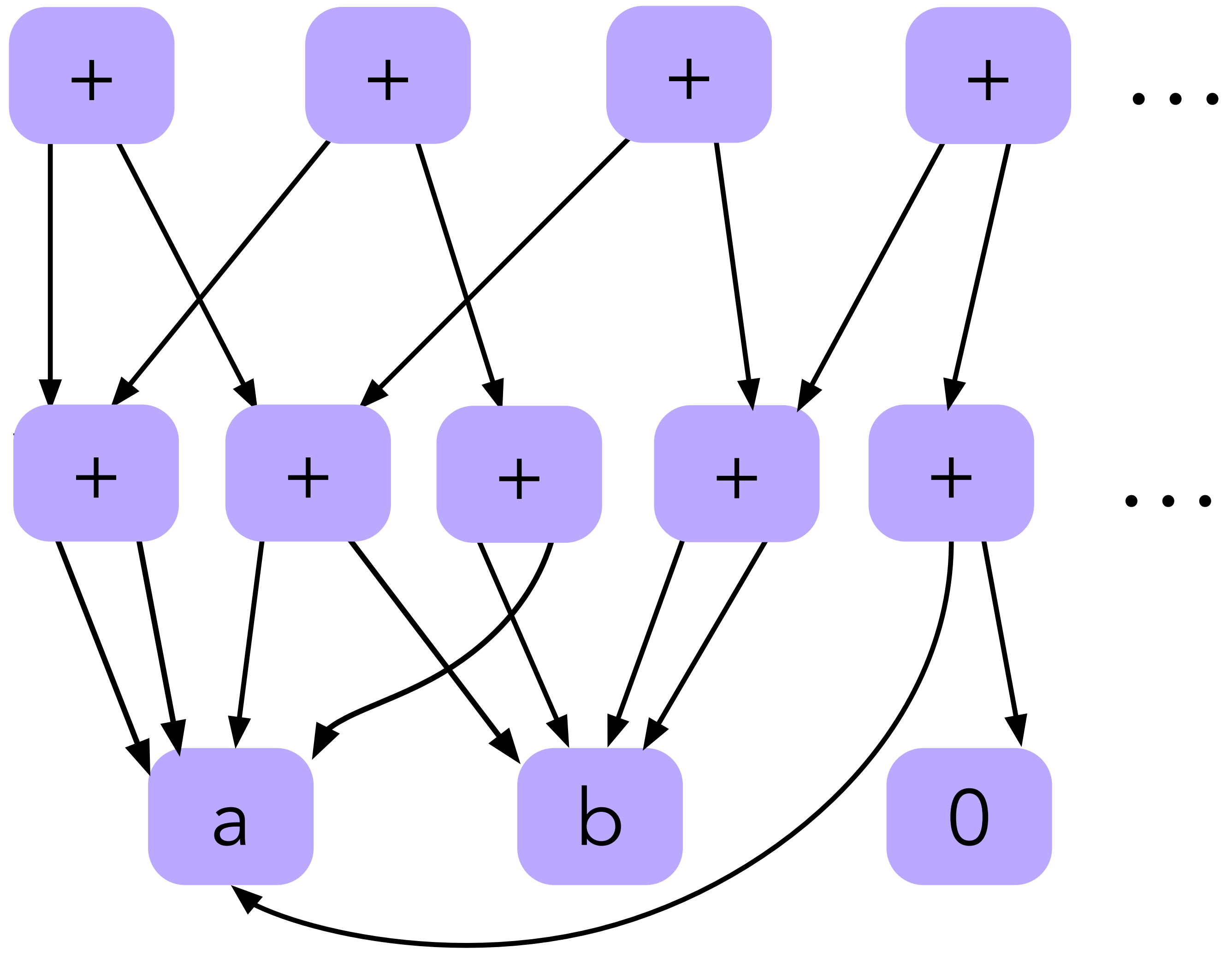
E-classes



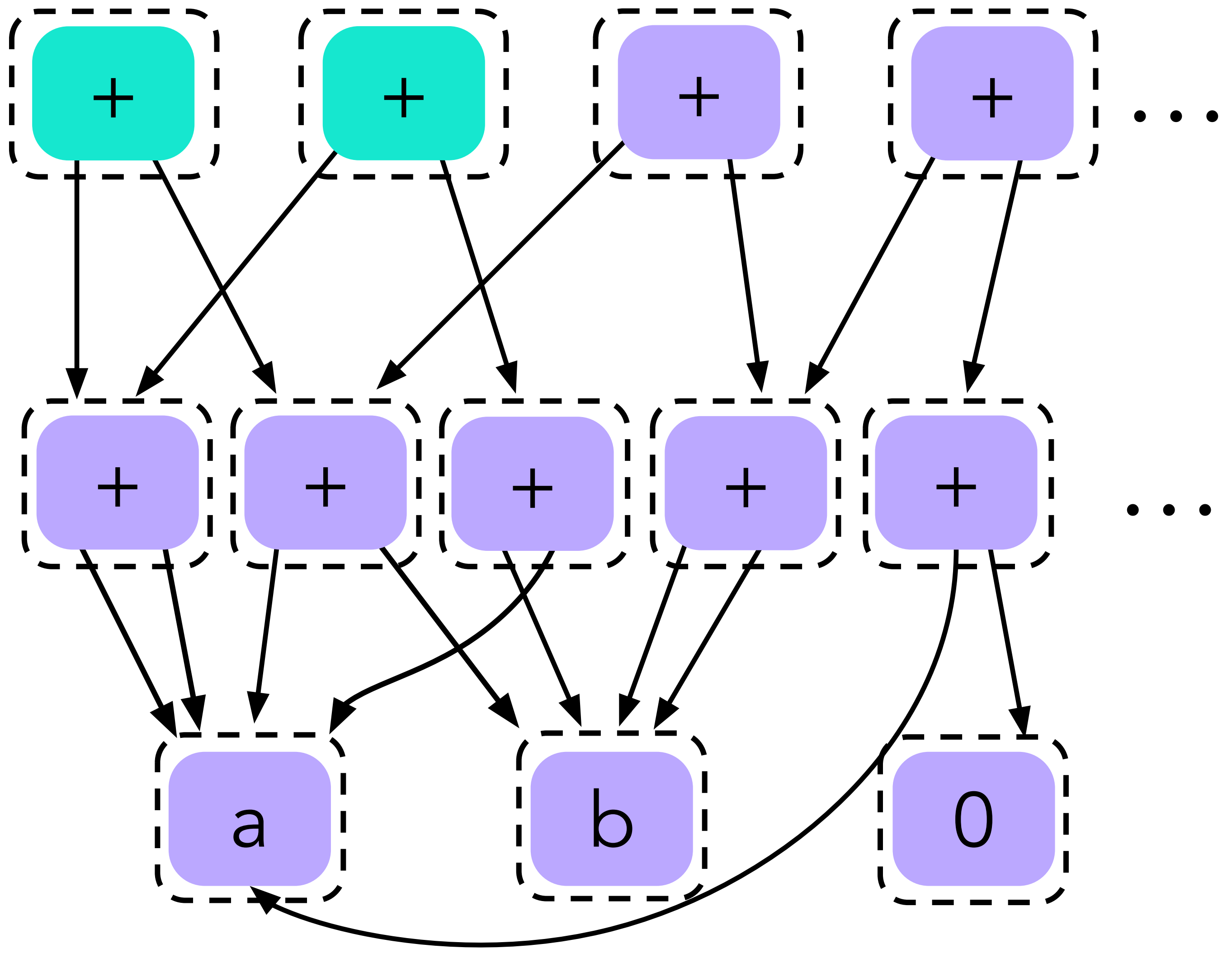
Enumerate over an E-graph

Enumeration modulo equality saturation

a, b, 0, +, …



E-classes



(x + x) + (x + y)

=

(x + x) + (y + x)

Exponentially many terms!

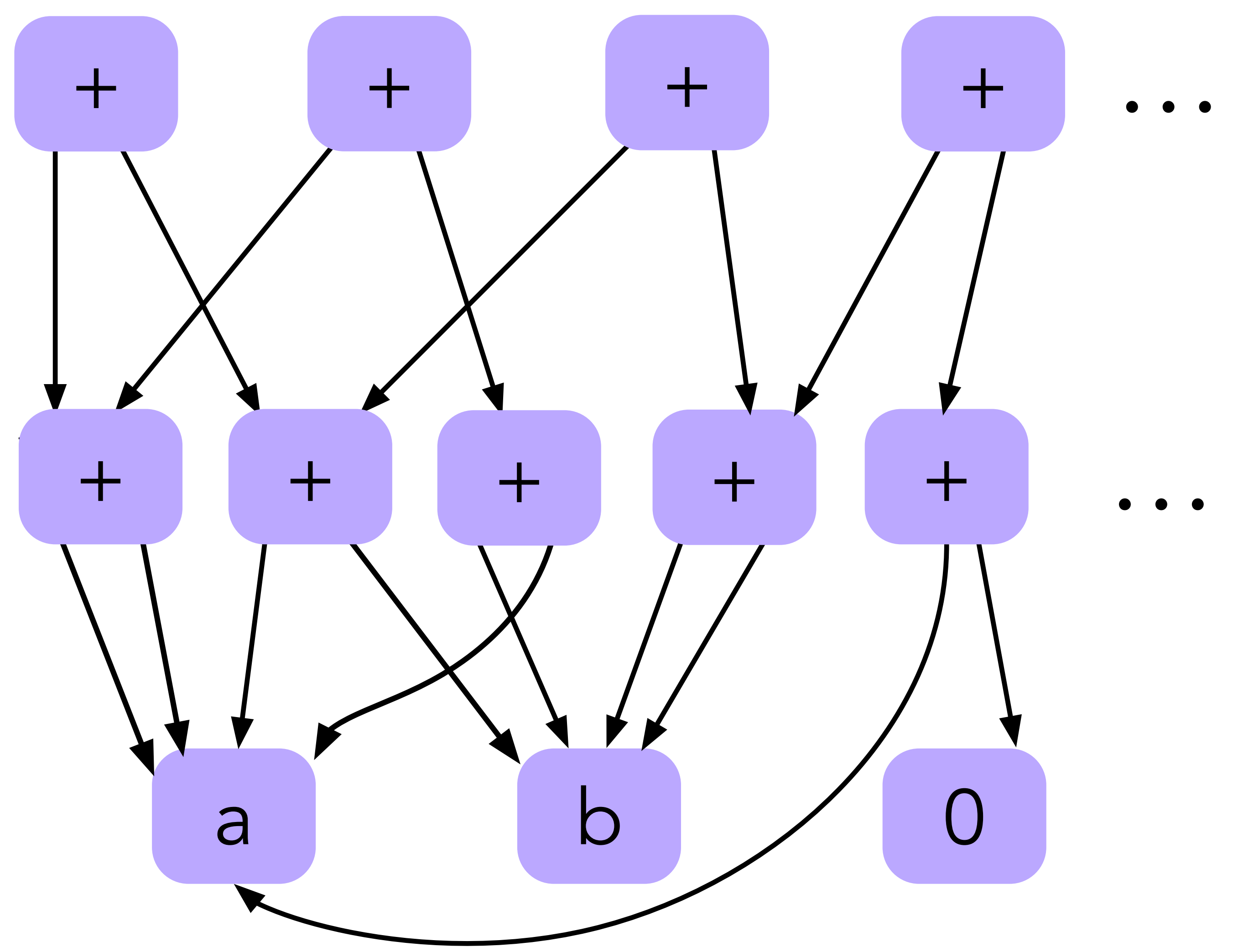
Enumerate over an E-graph

Apply current ruleset

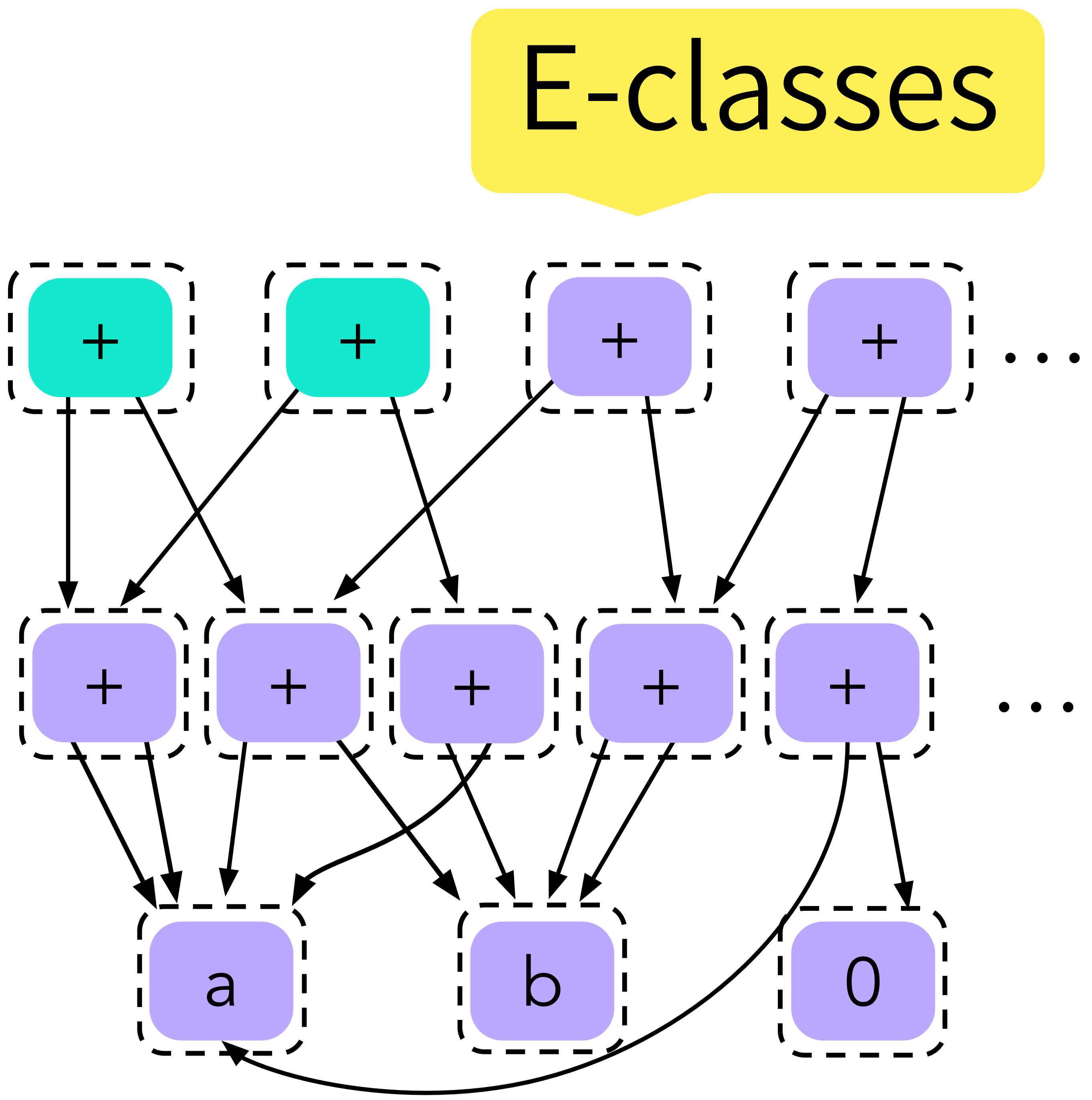
(x + y) ⟷ (y + x)

Enumeration modulo equality saturation

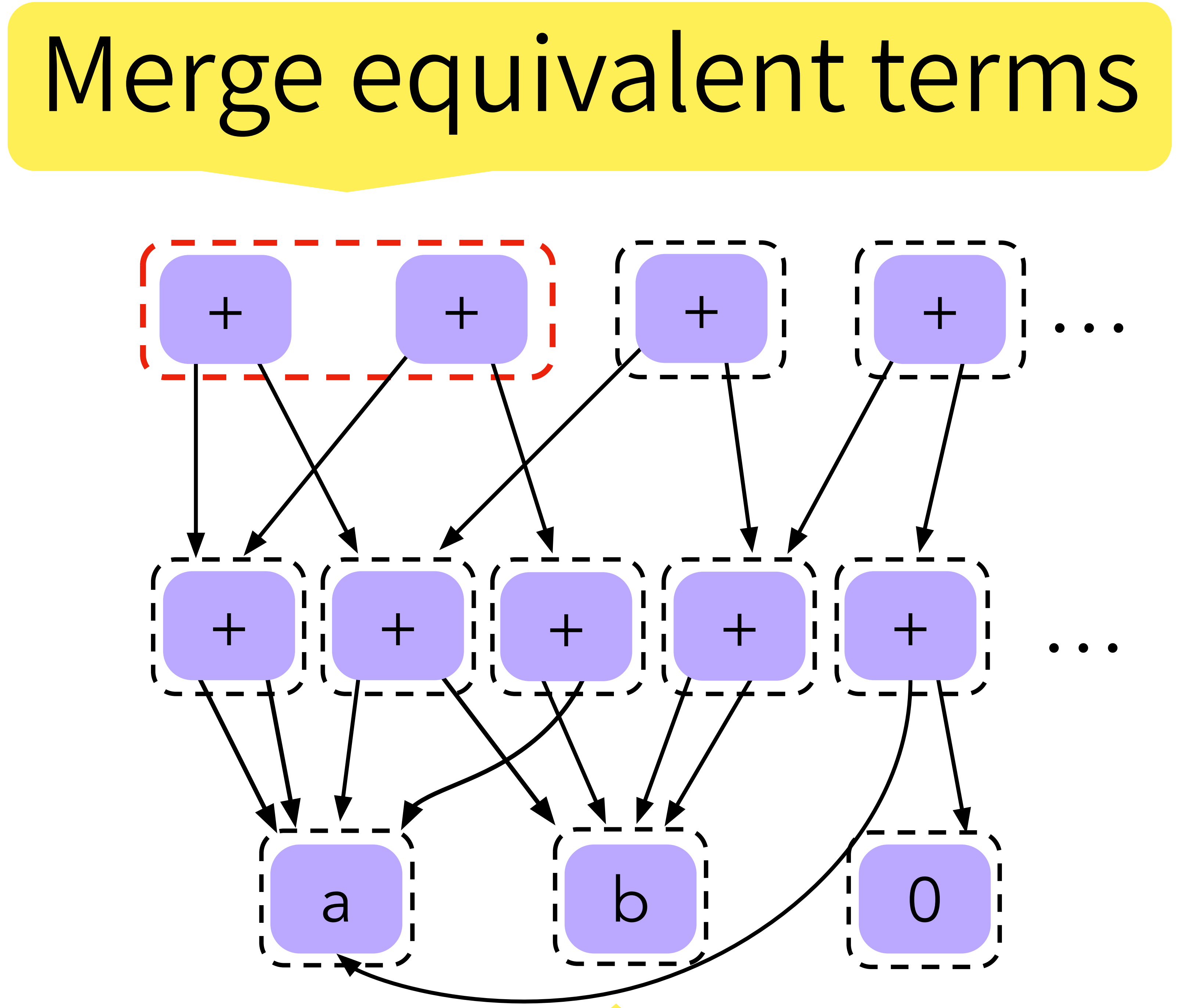
a, b, 0, +, …



Exponentially many terms!



Enumerate over an E-graph



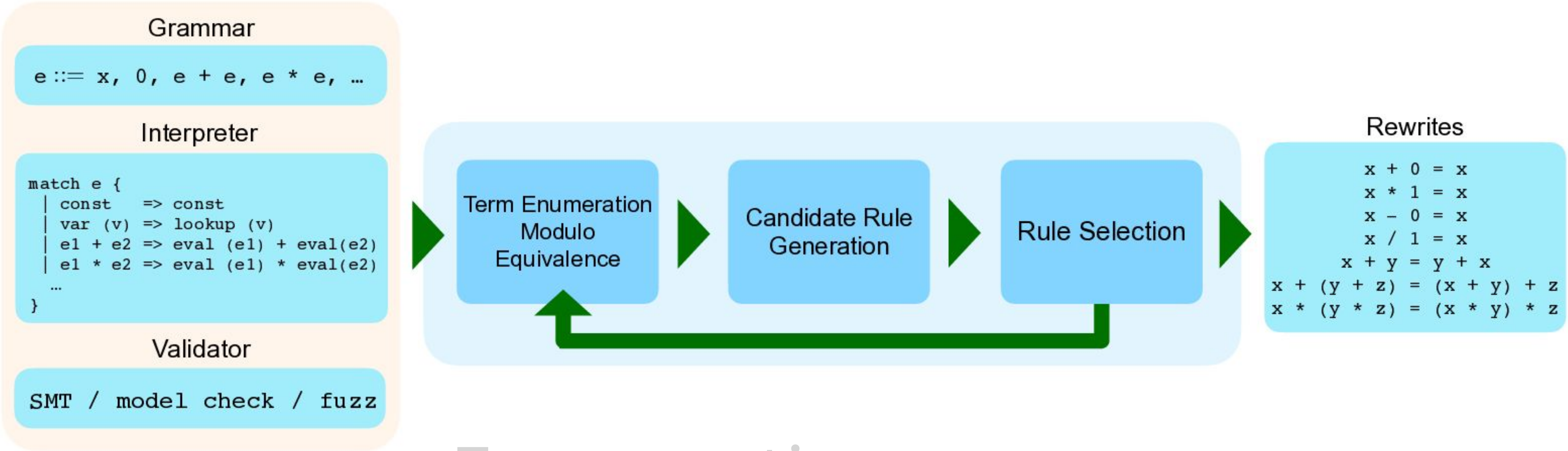
Apply current ruleset

$(x + y) \leftrightarrow (y + x)$

Enumeration modulo equality saturation

Shrinks the term space by applying rewrites as they are learned!

Ruler

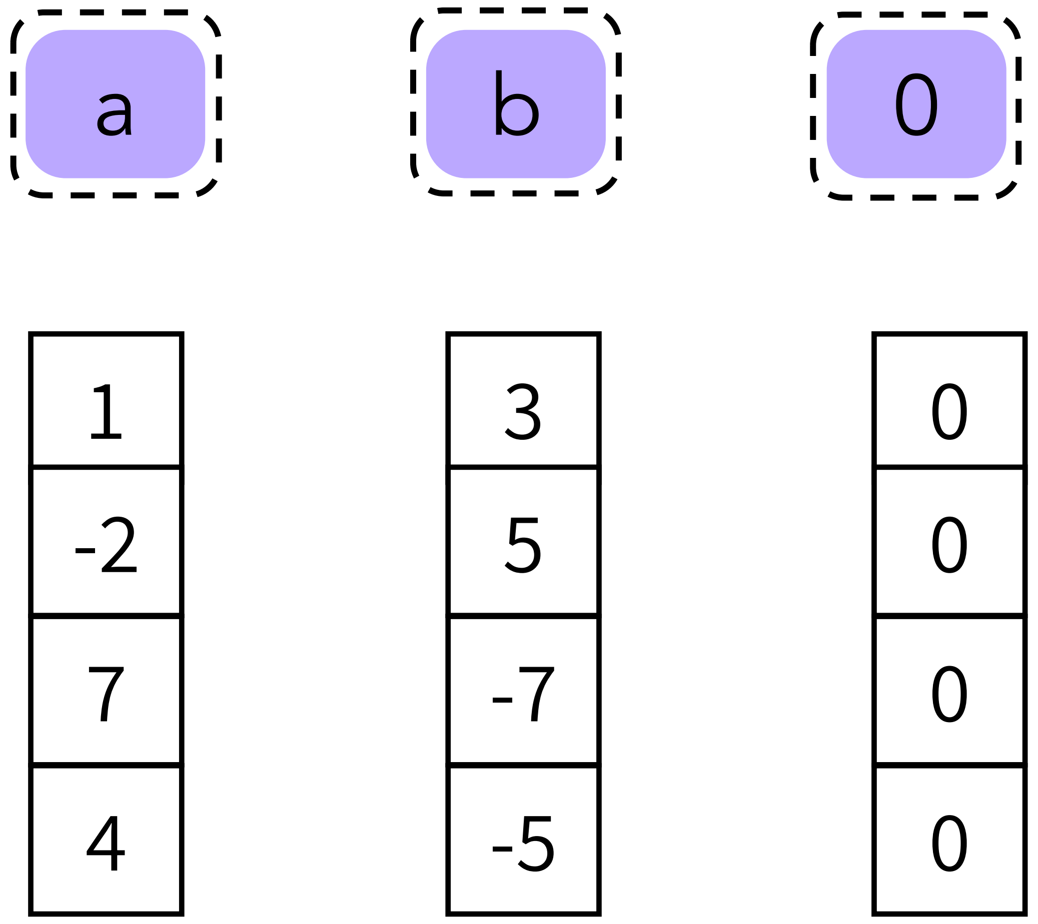


Enumeration

Candidate Generation

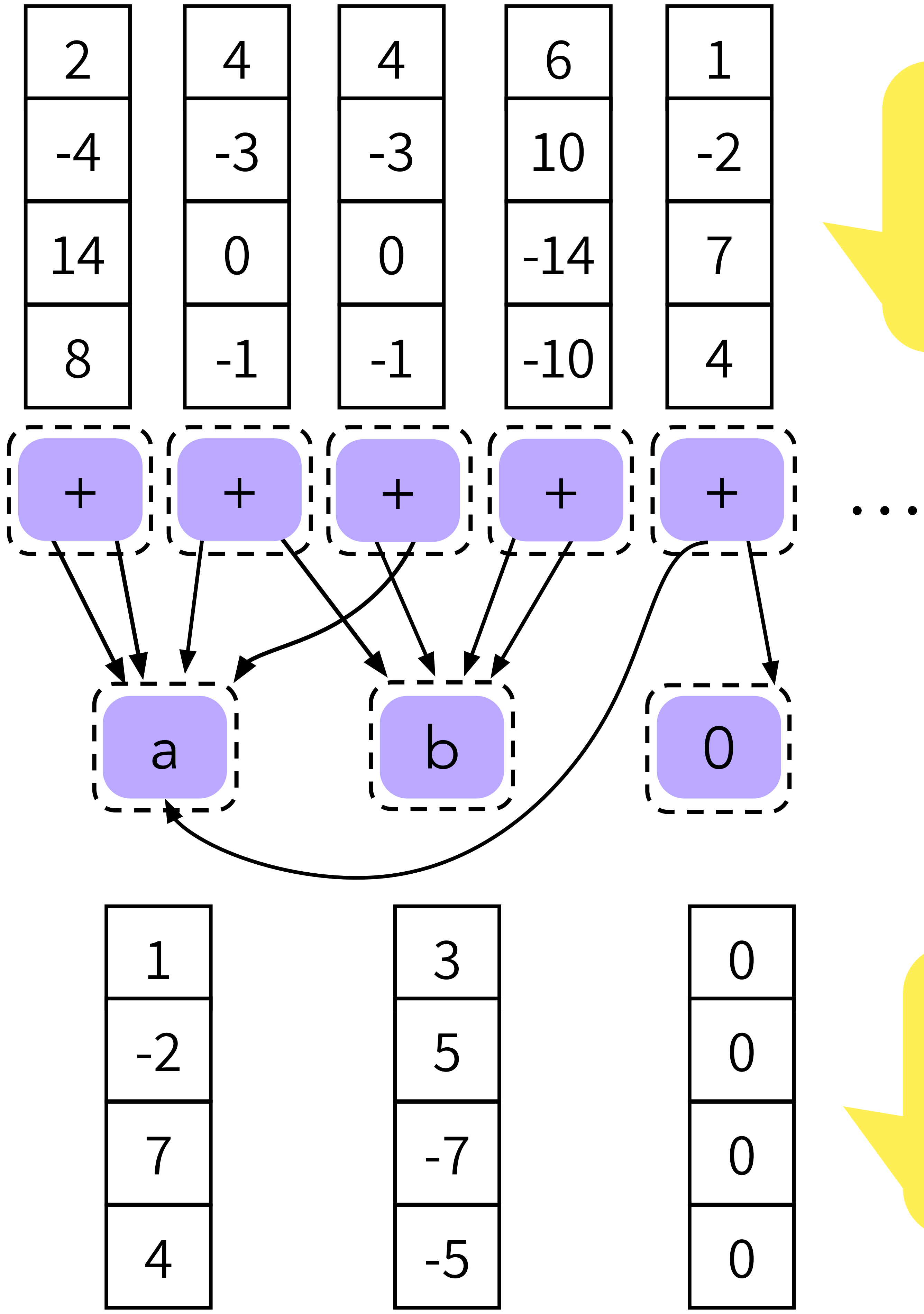
Rule Selection

Candidate generation by characteristic vector matching



Seed initial E-classes with concrete values (cvecs) from the domain

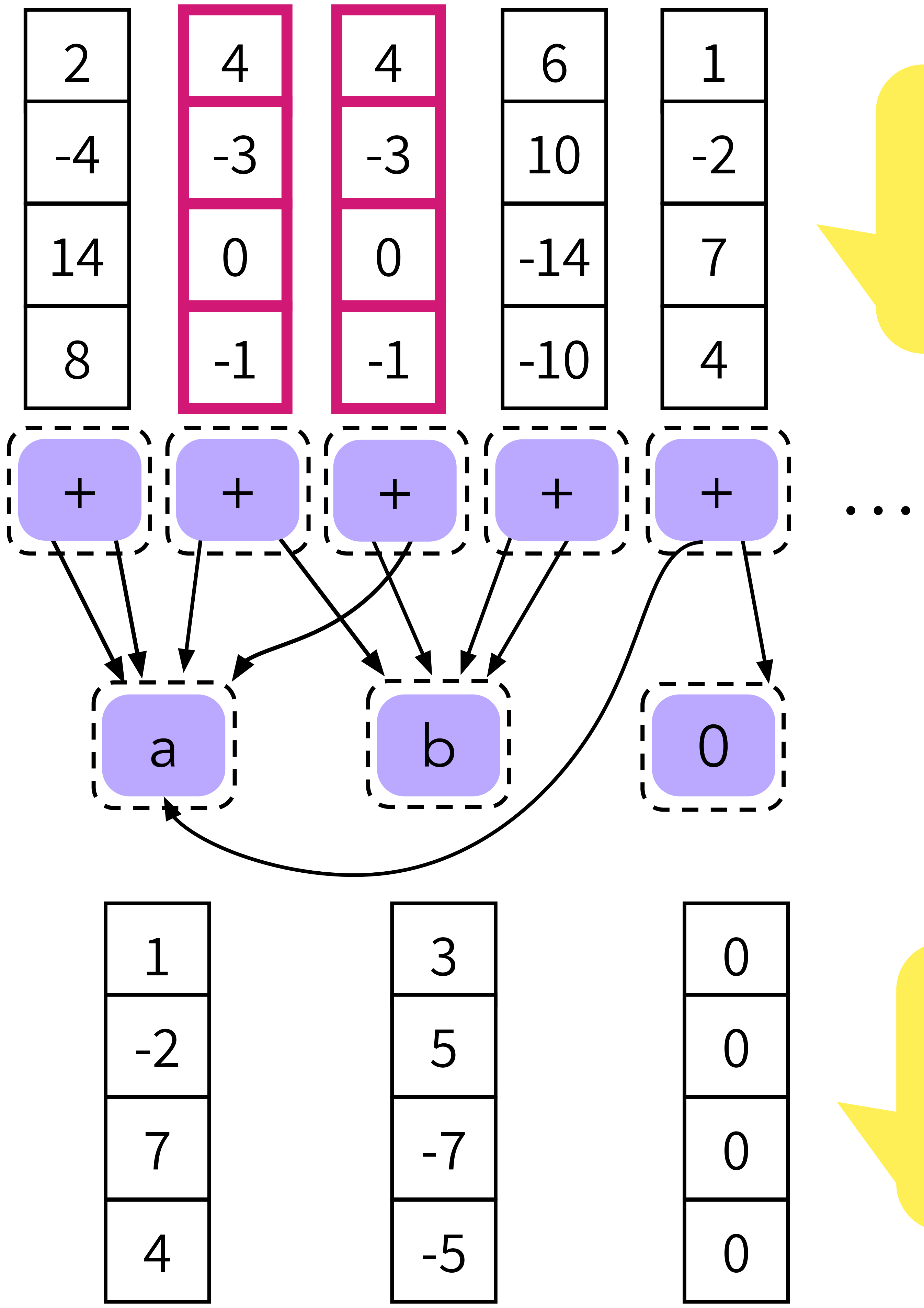
Candidate generation by characteristic vector matching



Compute the cvecs for newly enumerated E-classes

Seed initial E-classes with concrete values (cvecs) from the domain

Candidate generation by characteristic vector matching

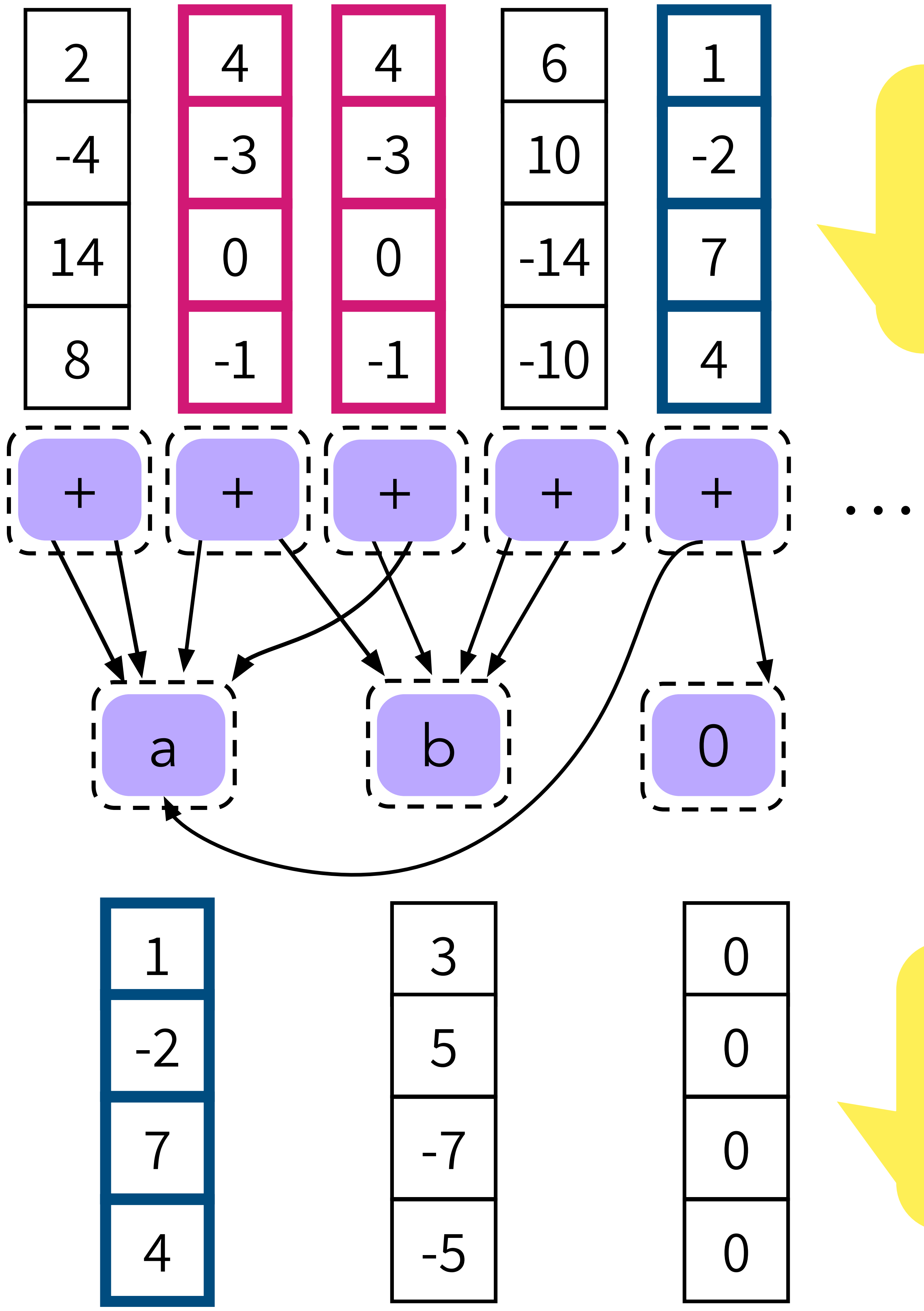


Compute the cvecs for newly enumerated E-classes

Seed initial E-classes with concrete values (cvecs) from the domain

(x + y) ↔ (y + x)

Candidate generation by characteristic vector matching



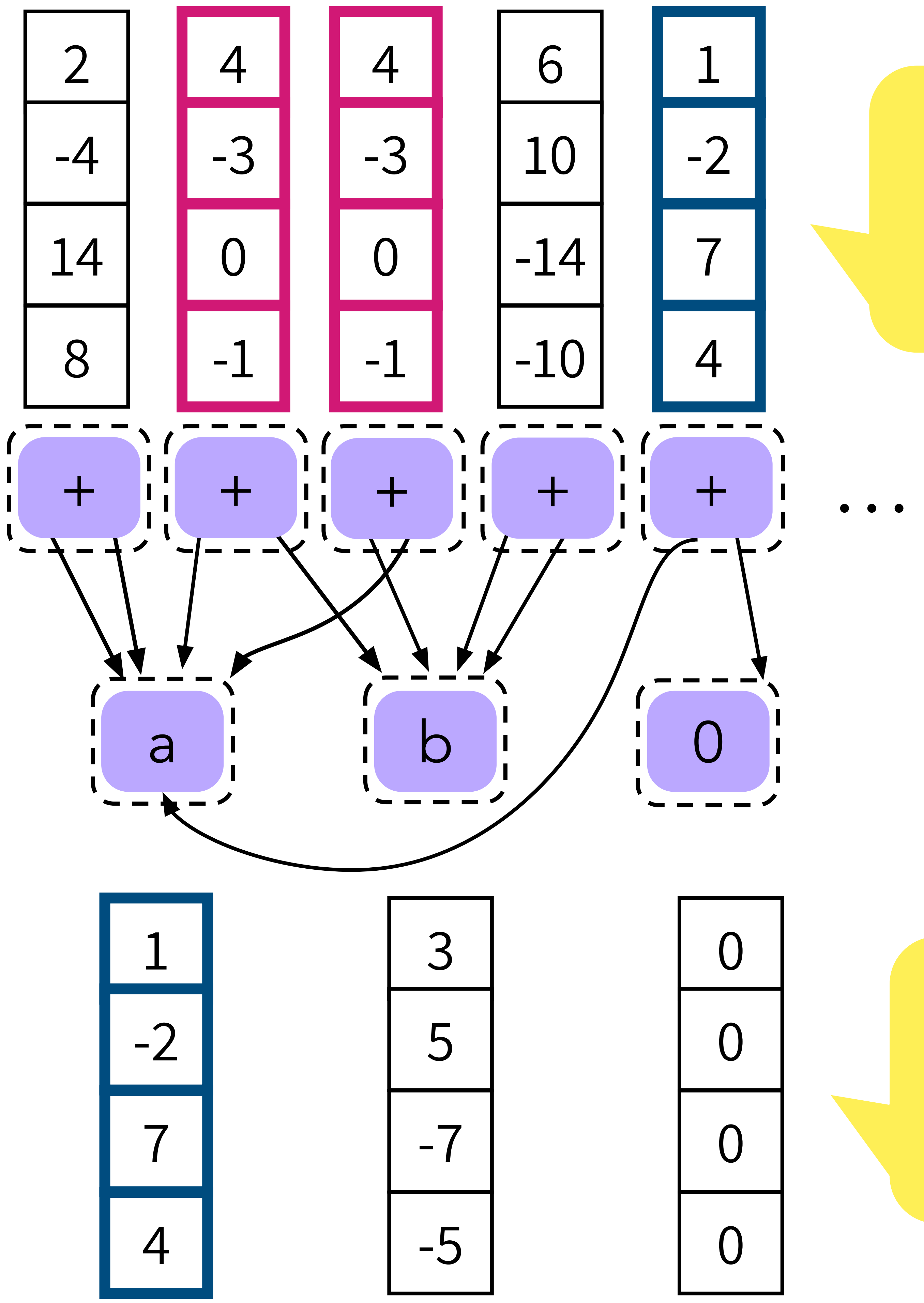
Compute the cvecs for newly enumerated E-classes

Seed initial E-classes with concrete values (cvecs) from the domain

(x + y) ↔ (y + x)

(x + 0) ↔ x

Candidate generation by characteristic vector matching



Compute the cvecs for newly enumerated E-classes

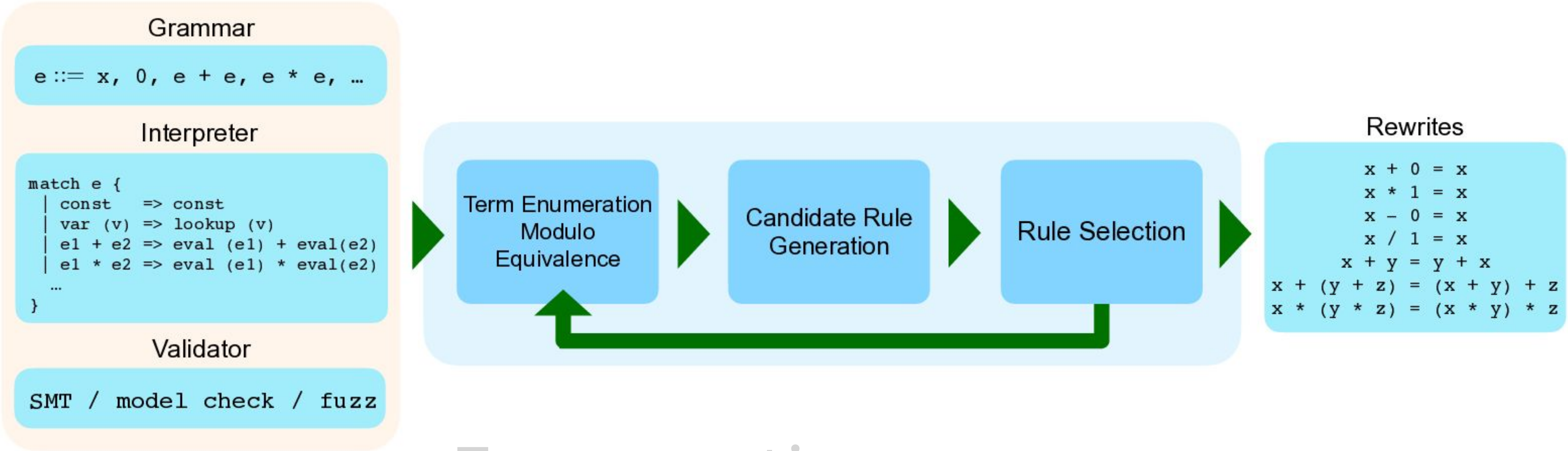
Seed initial E-classes with concrete values (cvecs) from the domain

(x + y) ⟷ (y + x)

(x + 0) ⟷ x

Validate candidates using SMT, fuzzing, model checking

Ruler



Enumeration

Candidate Generation

Rule Selection

Rule selection with equality saturation

C =

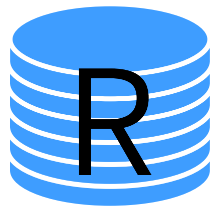
(x + y)	⟷	(y + x)
(x + 0)	⟷	(0 + x)
(y + 0)	⟷	(0 + y)
(x * y)	⟷	(y * x)
(x * 1)	⟷	(1 * x)
(y * 1)	⟷	(1 * y)

Rule selection with equality saturation

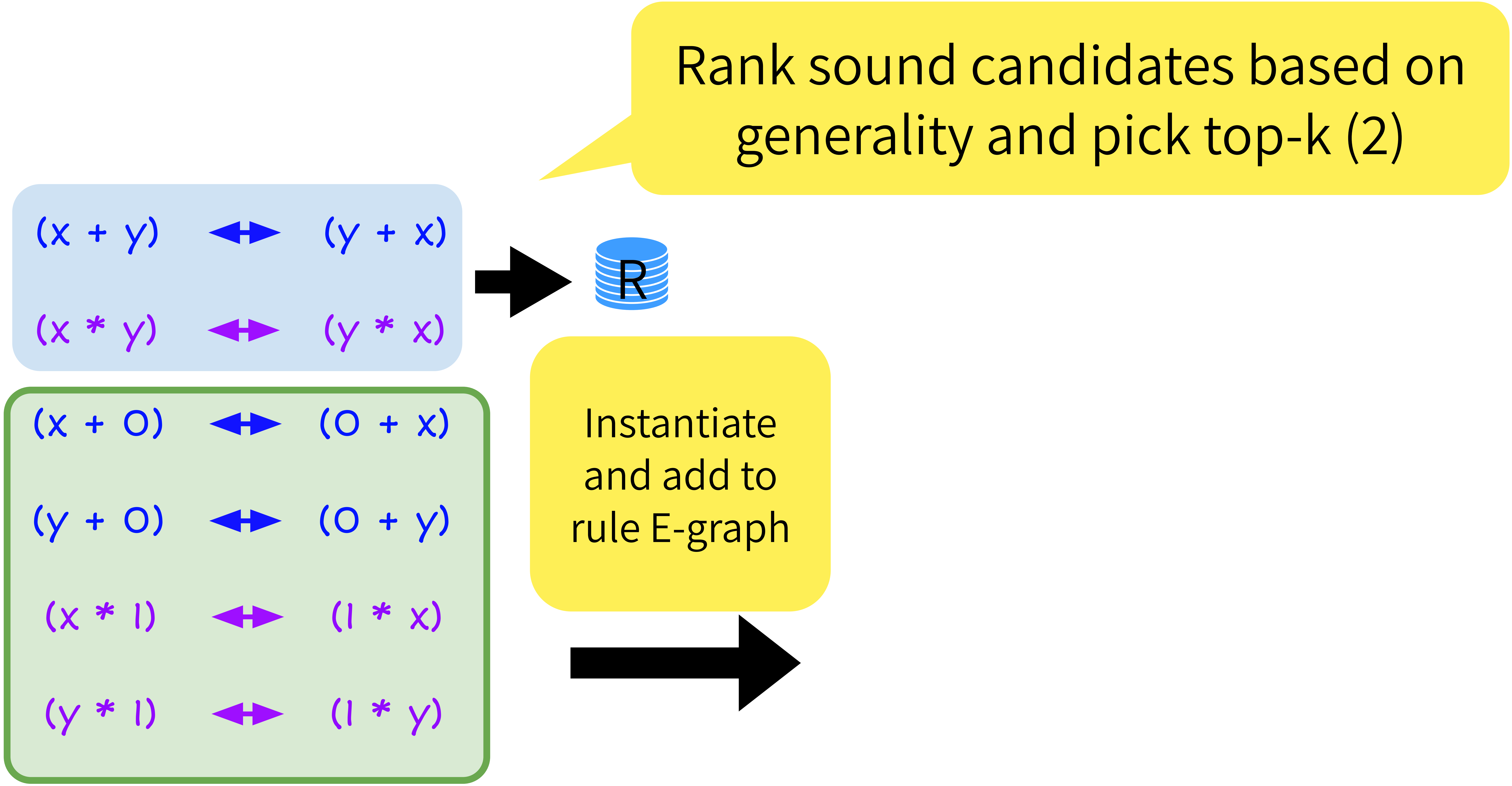
Rank sound candidates based on generality and pick top-k (2)

C =

- (x + y) ↔ (y + x)
- (x * y) ↔ (y * x)
- (x + 0) ↔ (0 + x)
- (y + 0) ↔ (0 + y)
- (x * 1) ↔ (1 * x)
- (y * 1) ↔ (1 * y)

→ 

Rule selection with equality saturation



Rule selection with equality saturation

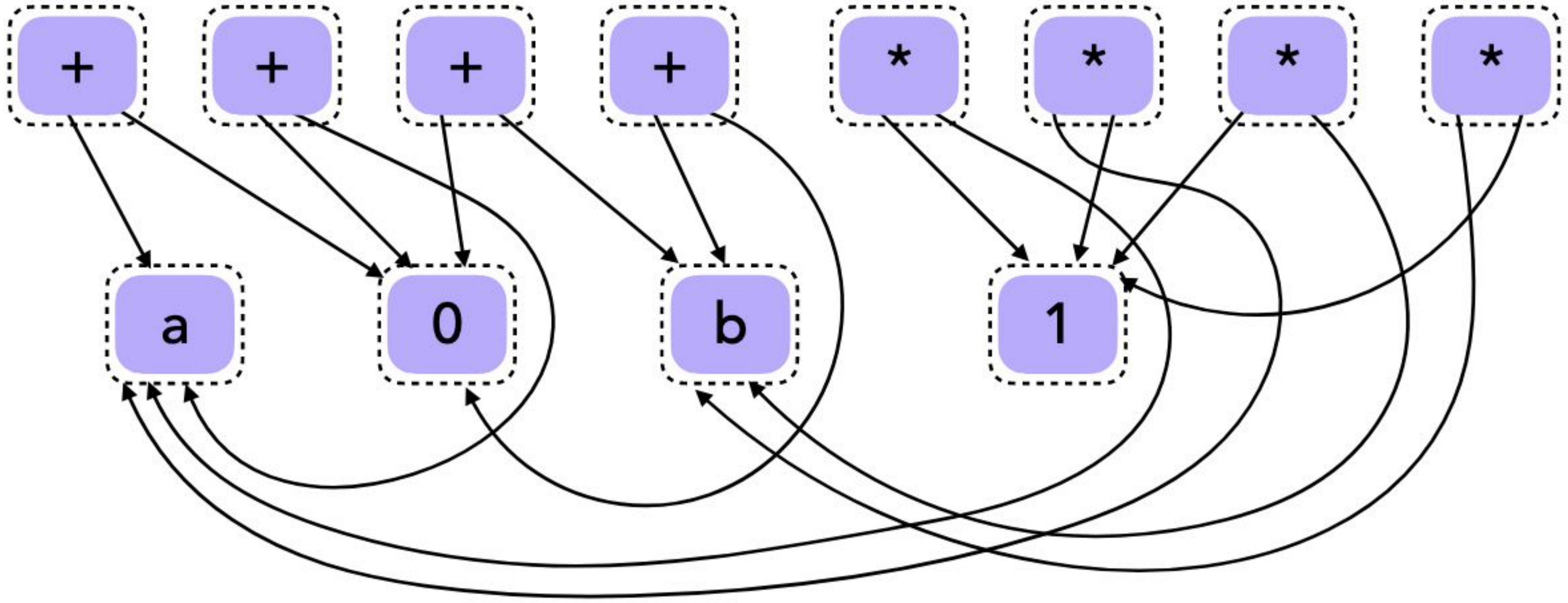
Rank sound candidates based on generality and pick top-k (2)

(x + y) ↔ (y + x)
(x * y) ↔ (y * x)

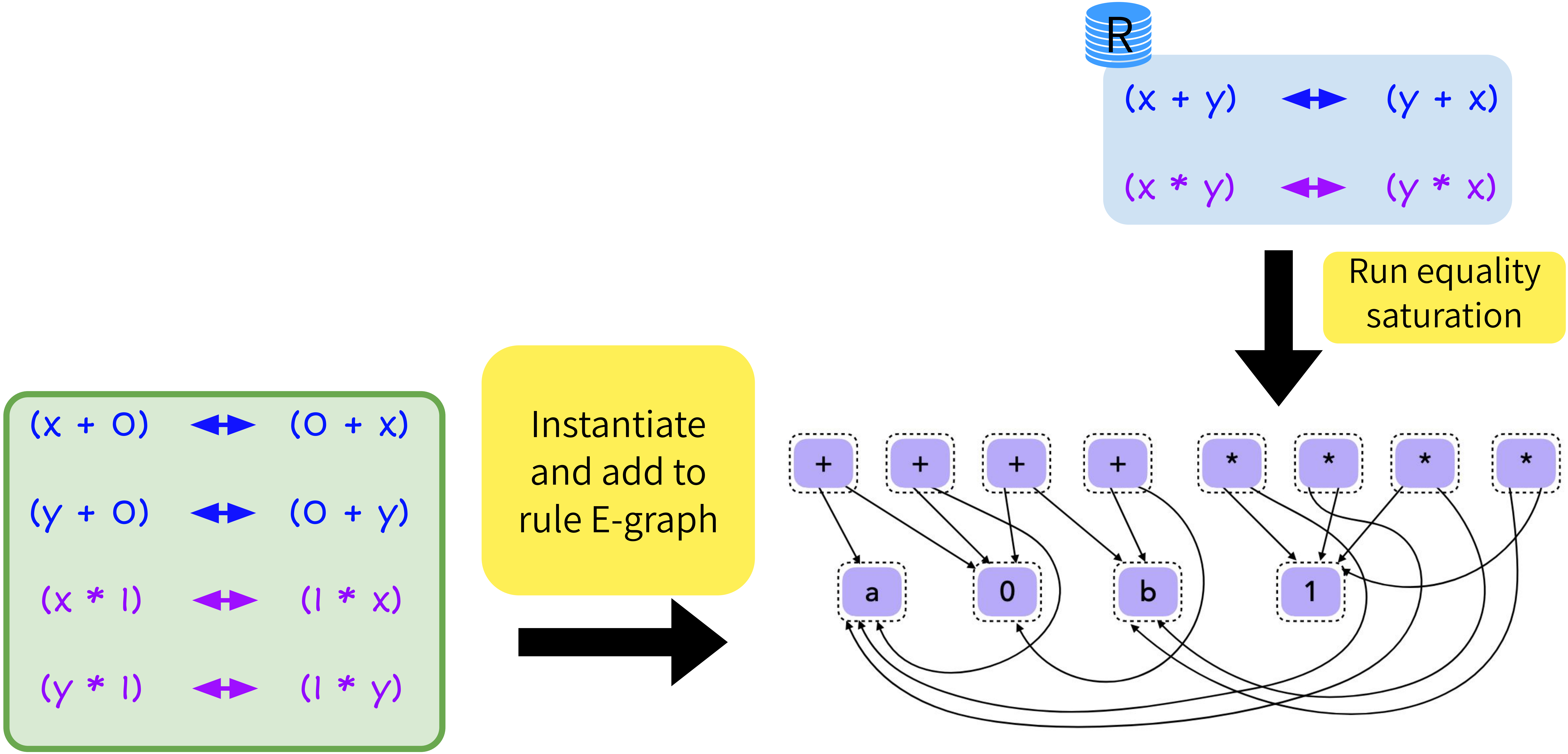
R

(x + 0) ↔ (0 + x)
(y + 0) ↔ (0 + y)
(x * 1) ↔ (1 * x)
(y * 1) ↔ (1 * y)

Instantiate and add to rule E-graph



Rule selection with equality saturation



Rule selection with equality saturation

R

(x + y) ↔ (y + x)

(x * y) ↔ (y * x)

All four rules are redundant and therefore discarded!

Run equality saturation

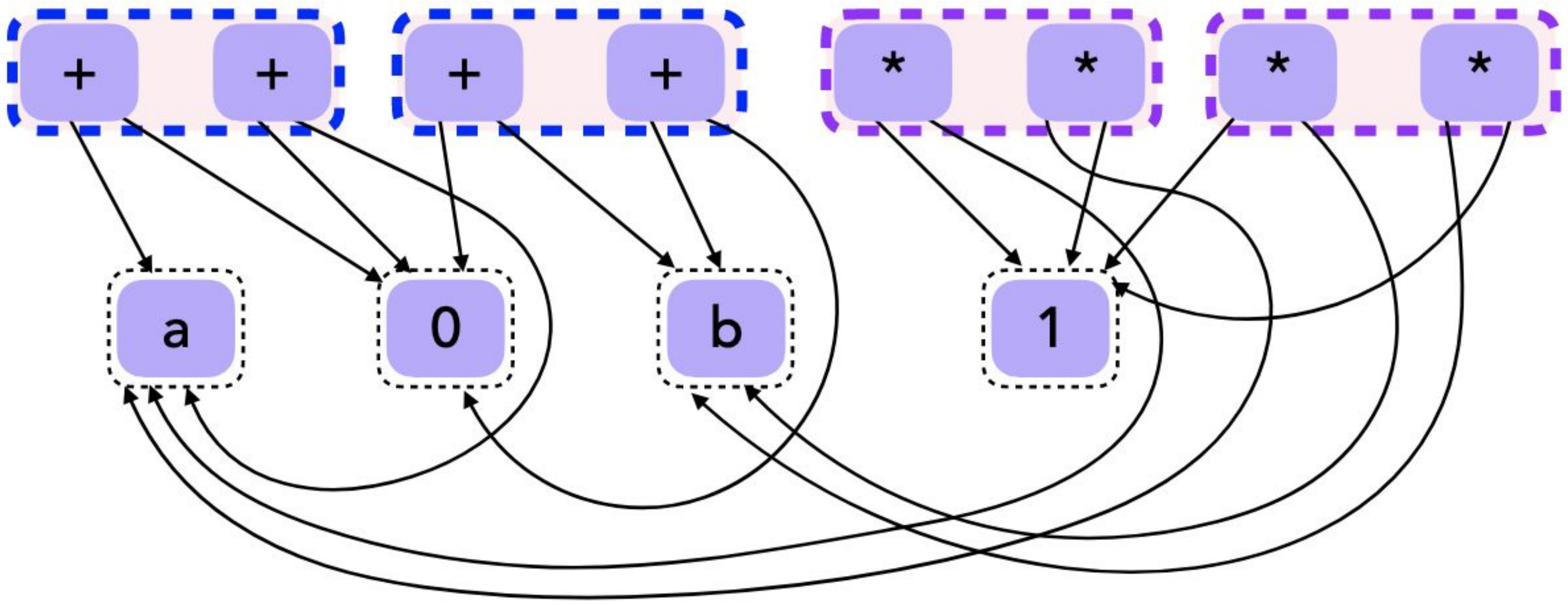
(x + 0) ↔ (0 + x)

(y + 0) ↔ (0 + y)

(x * 1) ↔ (1 * x)

(y * 1) ↔ (1 * y)

Instantiate and add to rule E-graph



Rule selection with equality saturation

Continue processing until candidate set is empty or has only unsound ones left!

R

(x + y) ↔ (y + x)

(x * y) ↔ (y * x)

All four rules are redundant and therefore discarded!

Run equality saturation

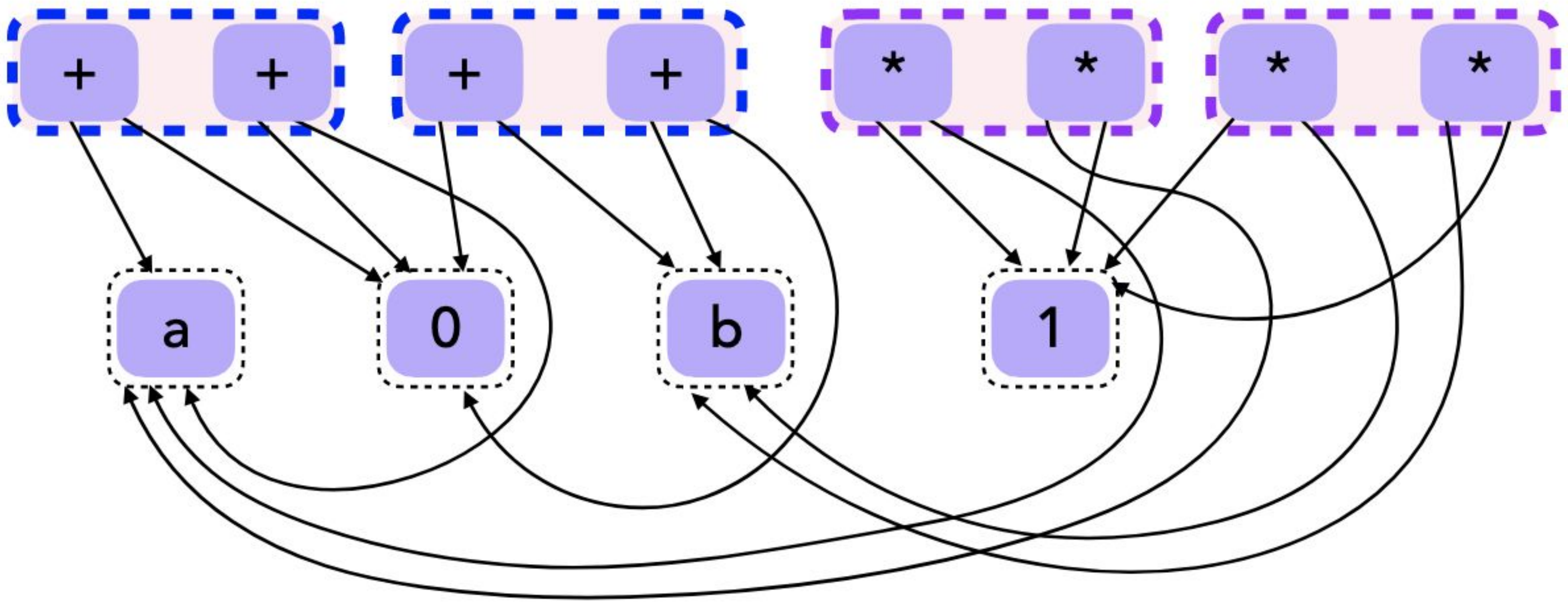
(x + 0) ↔ (0 + x)

(y + 0) ↔ (0 + y)

(x * 1) ↔ (1 * x)

(y * 1) ↔ (1 * y)

Instantiate and add to rule E-graph



Rule selection with equality saturation

Larger top-k makes Ruler faster

Smaller top-k gives smaller rulesets

See paper for detailed comparison!

R

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)

Run equality saturation

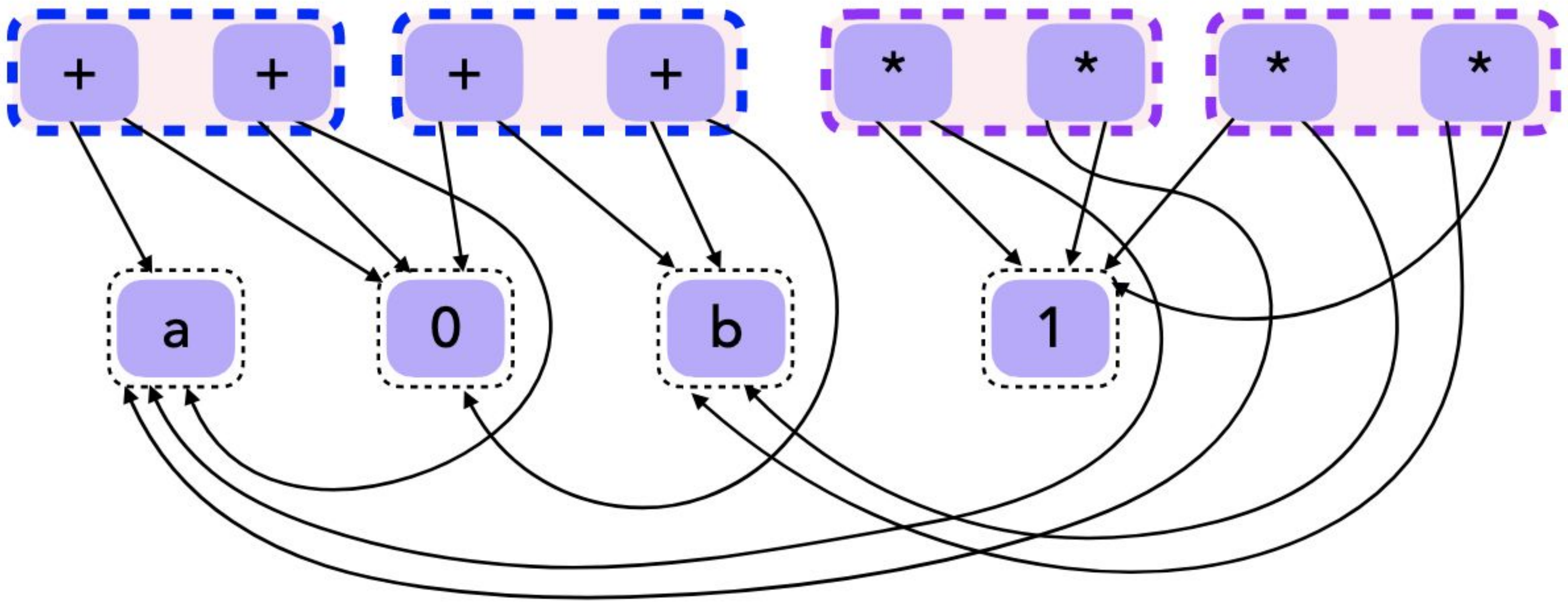
(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

Instantiate and add to rule E-graph

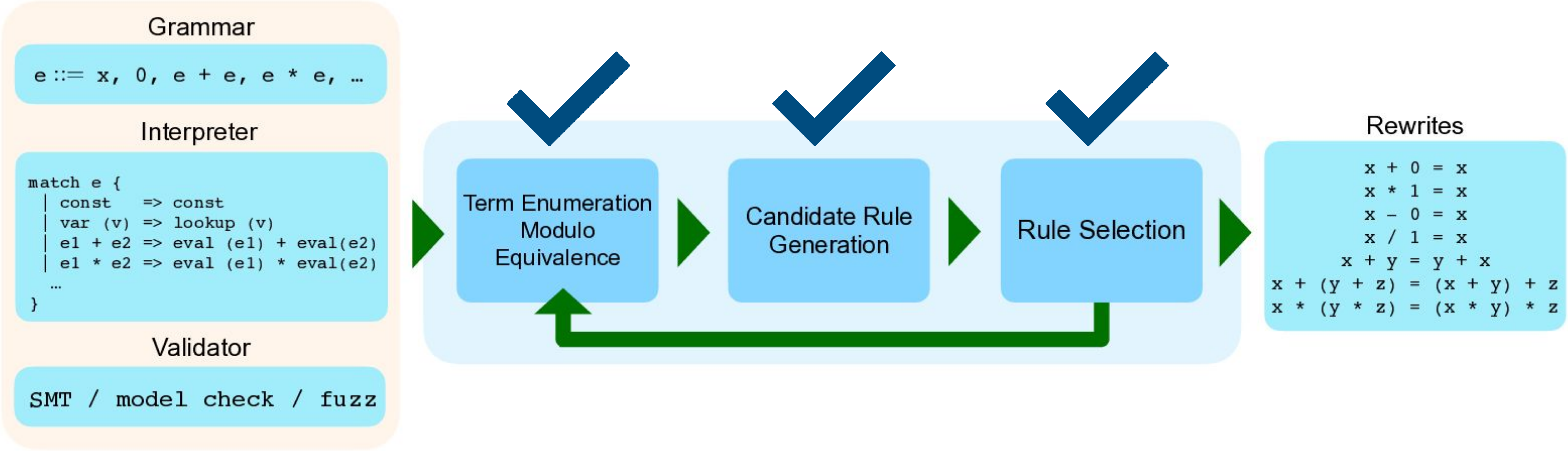


Rule selection with equality saturation

Larger top-k makes Ruler faster

Shrinks the *candidate space* by applying rewrites as they are learned!

Ruler

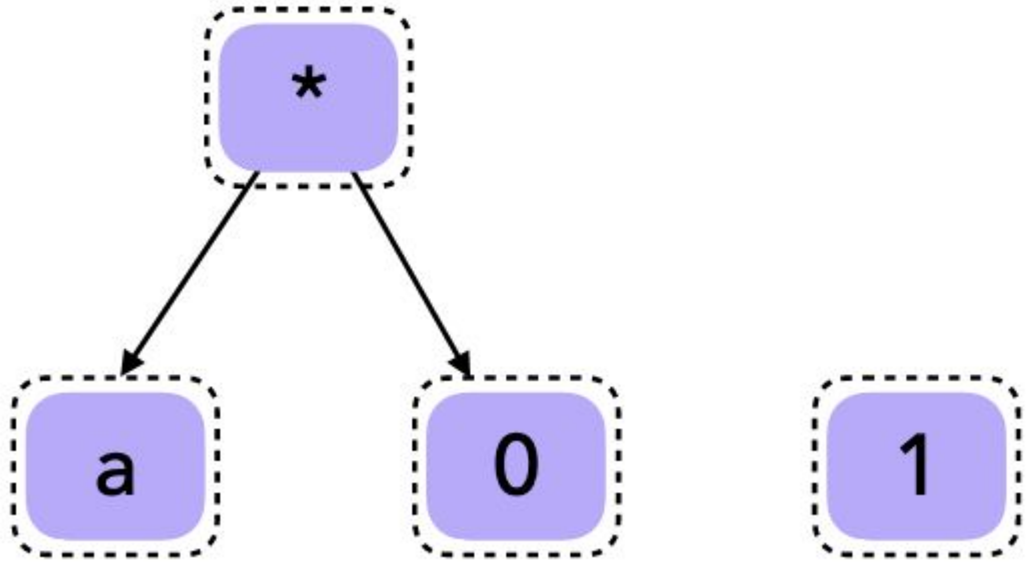

Grammar
e ::= x, 0, e + e, e * e, …
Interpreter
match e {
| const => const
| var (v) => lookup (v)
| e1 + e2 => eval (e1) + eval(e2)
| e1 * e2 => eval (e1) * eval(e2)
…
}
Validator
SMT / model check / fuzz
Term Enumeration Modulo Equivalence
Candidate Rule Generation
Rule Selection
Rewrites
x + 0 = x
x * 1 = x
x – 0 = x
x / 1 = x
x + y = y + x
x + (y + z) = (x + y) + z
x * (y * z) = (x * y) * z

Equality saturation “*soundiness*”

Equality Saturation *amplifies* unsoundness!

Equality saturation “*soundiness*”

Equality Saturation *amplifies* unsoundness!



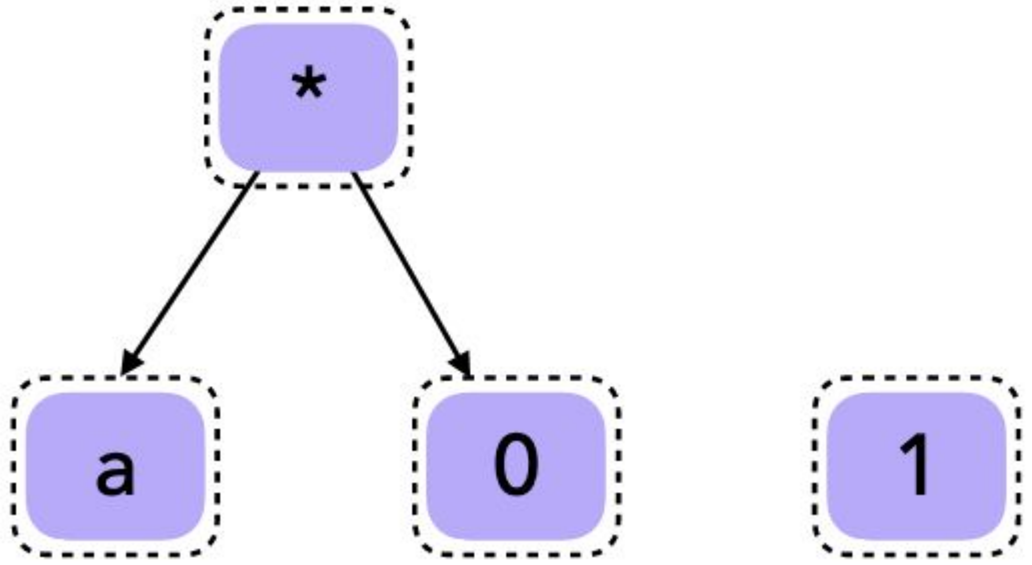
Equality saturation “*soundiness*”

Equality Saturation *amplifies* unsoundness!

current ruleset

(y * 0) ↔ 0

(y * 0) ↔ 1



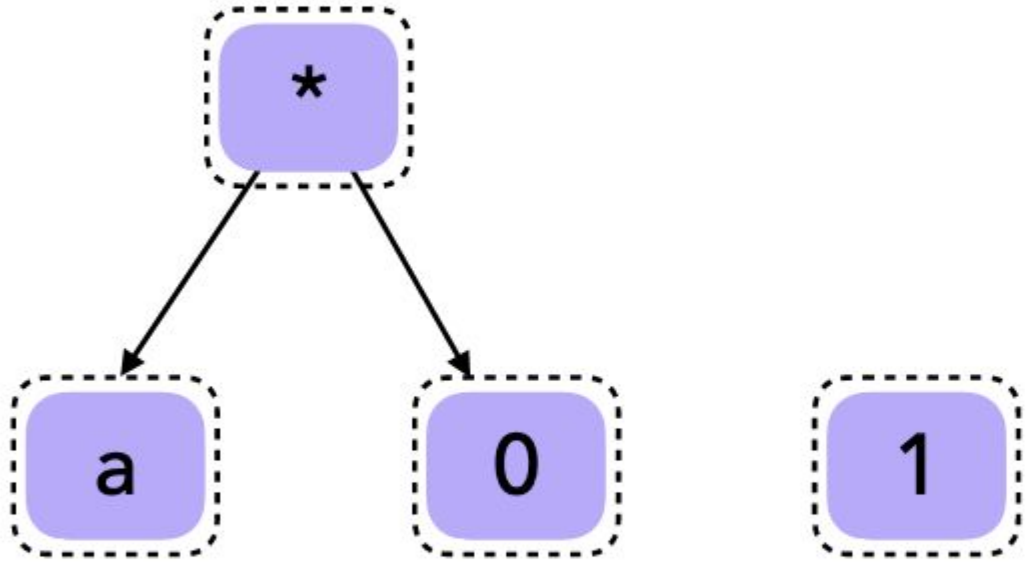
Equality saturation “*soundiness*”

Equality Saturation *amplifies* unsoundness!

current ruleset

(y * 0) ↔ 0

(y * 0) ↔ 1



Run equality saturation on term E-graph

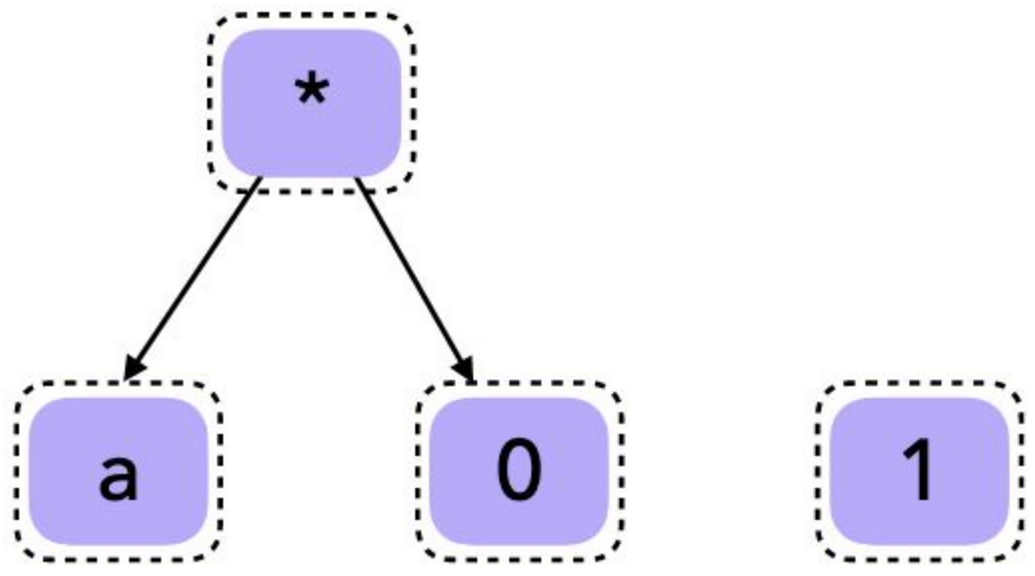
Equality saturation "*soundiness*"

Equality Saturation *amplifies* unsoundness!

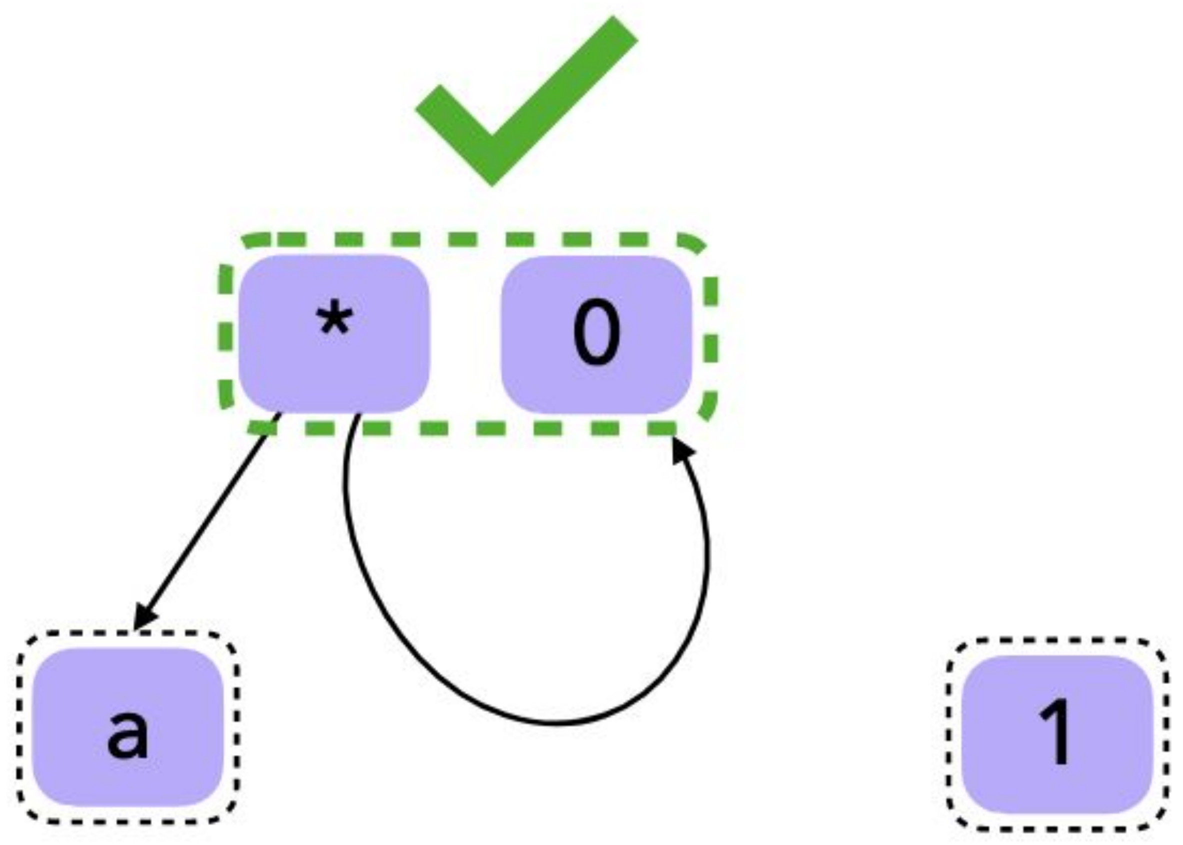
current ruleset

(y * 0) ⟷ 0

(y * 0) ⟷ 1

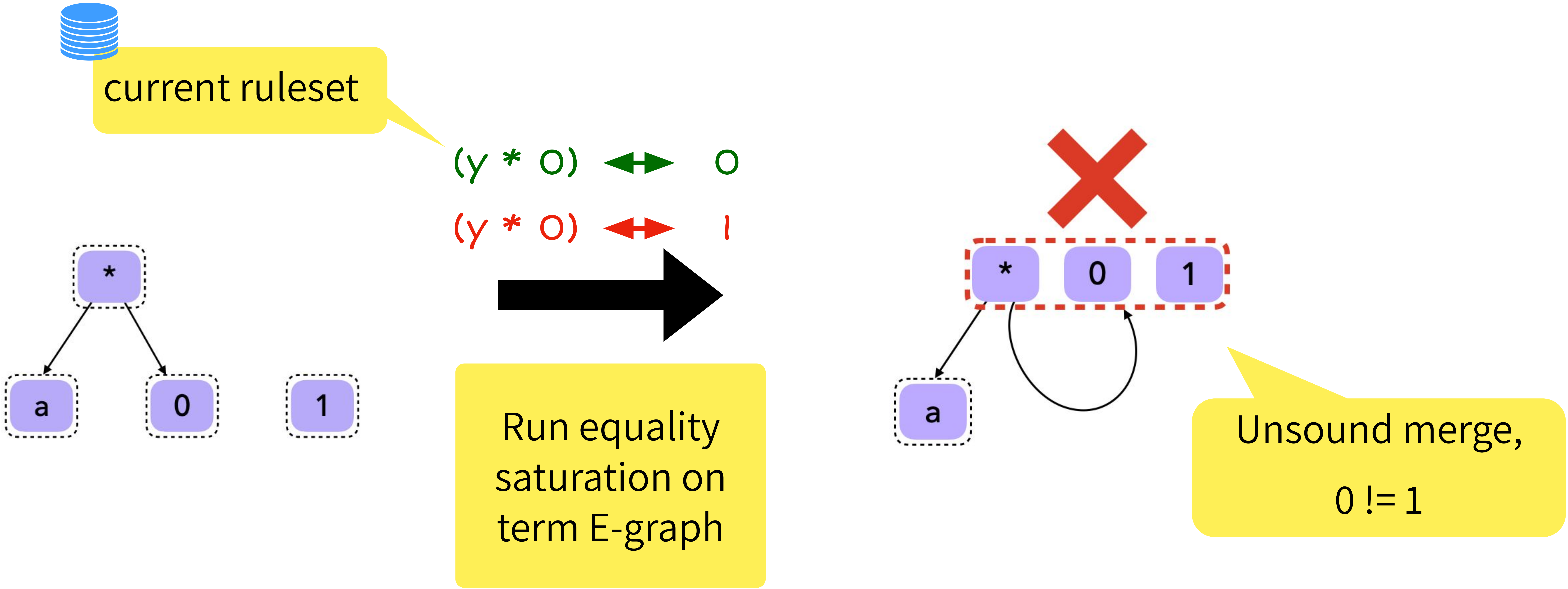


Run equality saturation on term E-graph

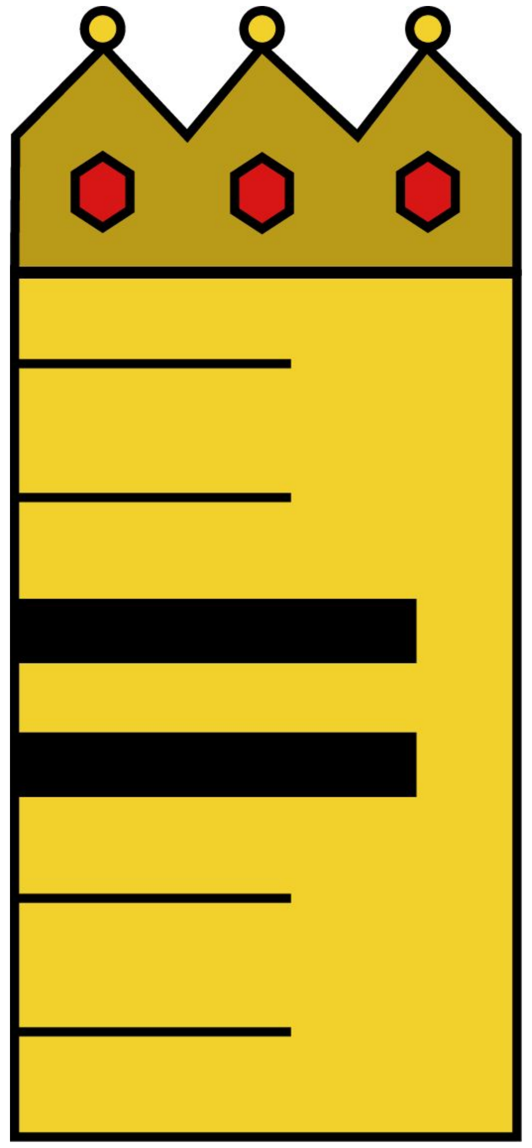


Equality saturation “*soundiness*”

Equality Saturation *amplifies* unsoundness!



Implementation



https://github.com/uwplse/ruler

Implemented in Rust

Uses egg for equality saturation

Evaluation

Ruler vs Other tools (CVC4)

How do the rulesets compare?

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Fraction of the 1782 rules from CVC4 that the 188 rules from Ruler can derive via equality saturation

Comparison with CVC4

Parameters		Ruler			CVC4			Ruler / CVC4	
Domain	# Conn	Time (s)	# Rules	Drv	Time (s)	# Rules	Drv	Time	Rules
bool	2	0.01	20	1	0.13	53	1	0.06	0.38
bool	3	0.06	28	1	0.82	293	1	0.07	0.10
bv4	2	0.14	49	1	4.47	135	0.98	0.03	0.36
bv4	3	4.30	272	1	372.26	1978	1	0.01	0.14
bv32	2	13.00	46	0.97	18.53	126	0.93	0.70	0.37
bv32	3	630.09	188	0.98	1199.53	1782	0.91	0.53	0.11

Ruler infers a smaller, useful ruleset faster

Evaluation

Ruler vs Other tools (CVC4)

How do the rulesets compare?

Ruler vs Humans (Herbie)

Can Ruler compete with experts?

Comparison with human-written rules

```
sqrt(x+1) - sqrt(x)  →  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.

Comparison with human-written rules

HERBIE

```
sqrt(x+1) - sqrt(x)  →  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*.

52 *rational* rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Comparison with human-written rules

HERBIE

`sqrt(x+1) - sqrt(x)` → `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.

52 *rational* rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise #261 Edit New issue

Closed nbraud opened this issue on Aug 31, 2019 · 4 comments

Comparison with human-written rules

HERBIE

`sqrt(x+1) - sqrt(x)` → `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*.

52 *rational* rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise #261 Edit New issue

Closed **nbraud** opened this issue on Aug 31, 2019 · 4 comments

$|x * y| \leftrightarrow |x| * |y|$

$|x * x| \leftrightarrow x * x$

Discovered by Ruler, resolved the GitHub issue!

End-to-end: rational Herbie

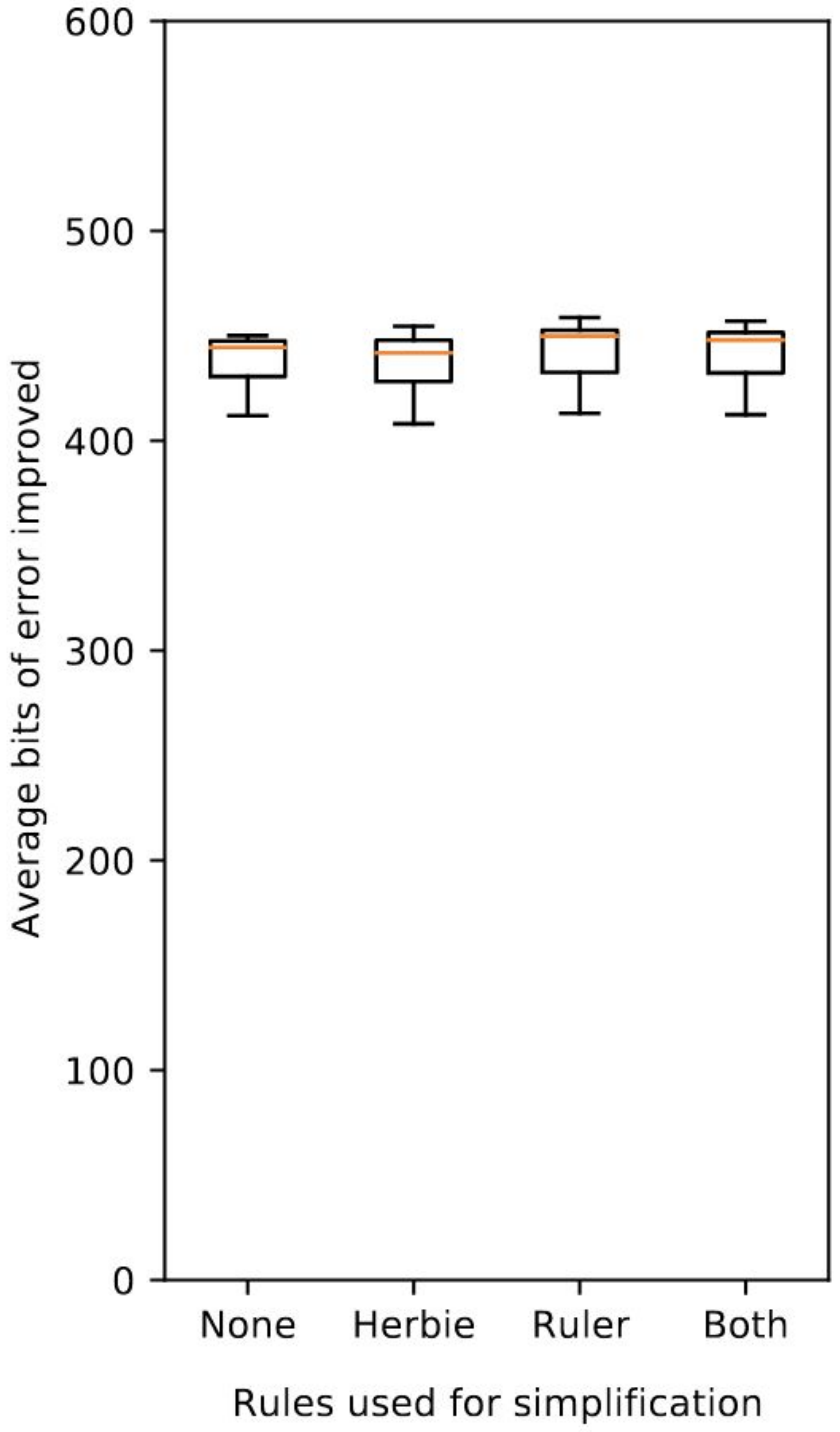
None: Remove all rules

Herbie: Herbie without any changes

Ruler: Herbie with Ruler's rules

Both: Herbie with both original and Ruler's rules

Rational Herbie: comparing accuracy



None: Remove all rules

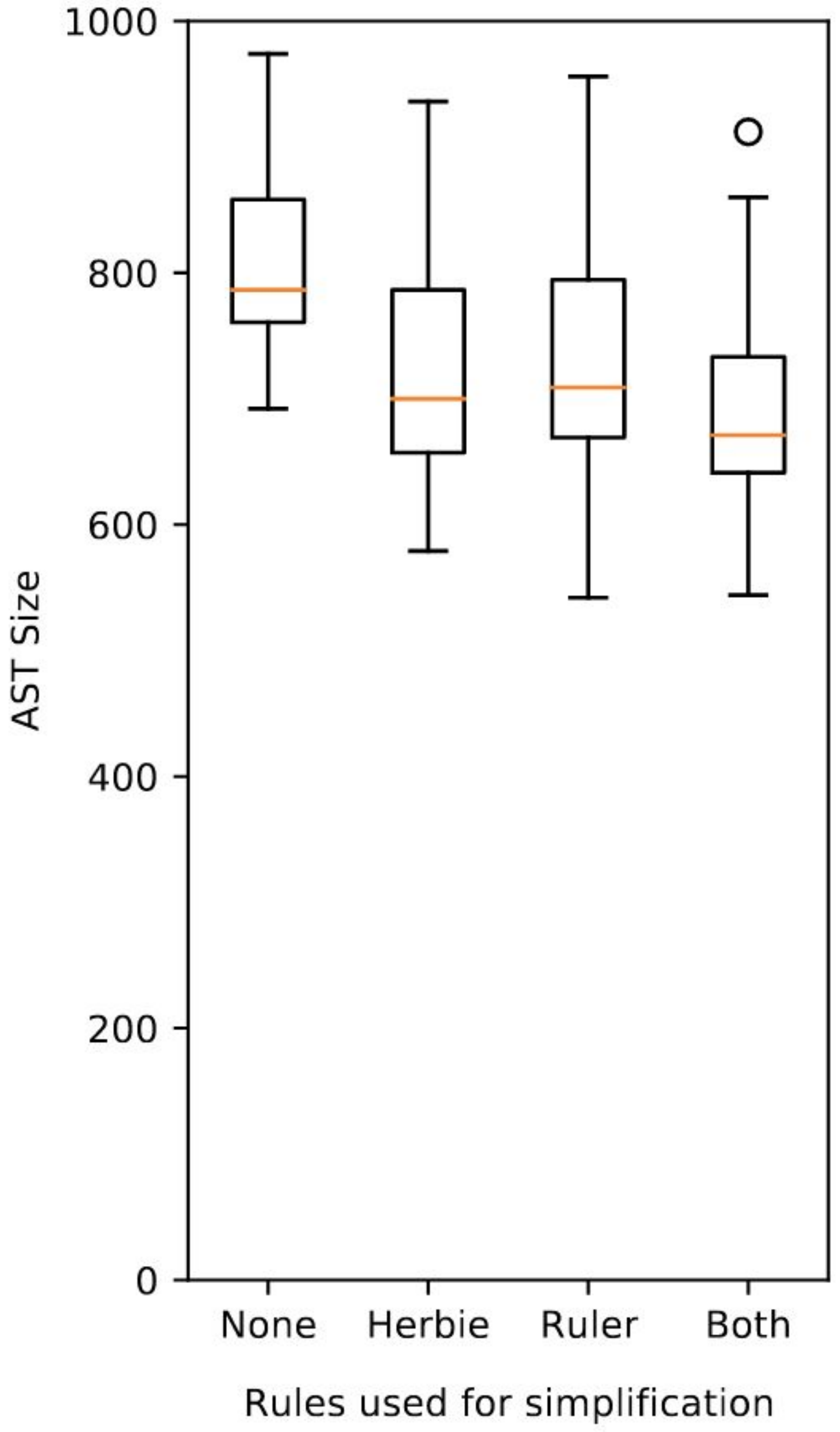
Herbie: Herbie without any changes

Ruler: Herbie with Ruler's rules

Both: Herbie with both original and Ruler's rules

Ruler's rules are at least as good as the original Herbie rules

Rational Herbie: comparing AST size



None: Remove all rules

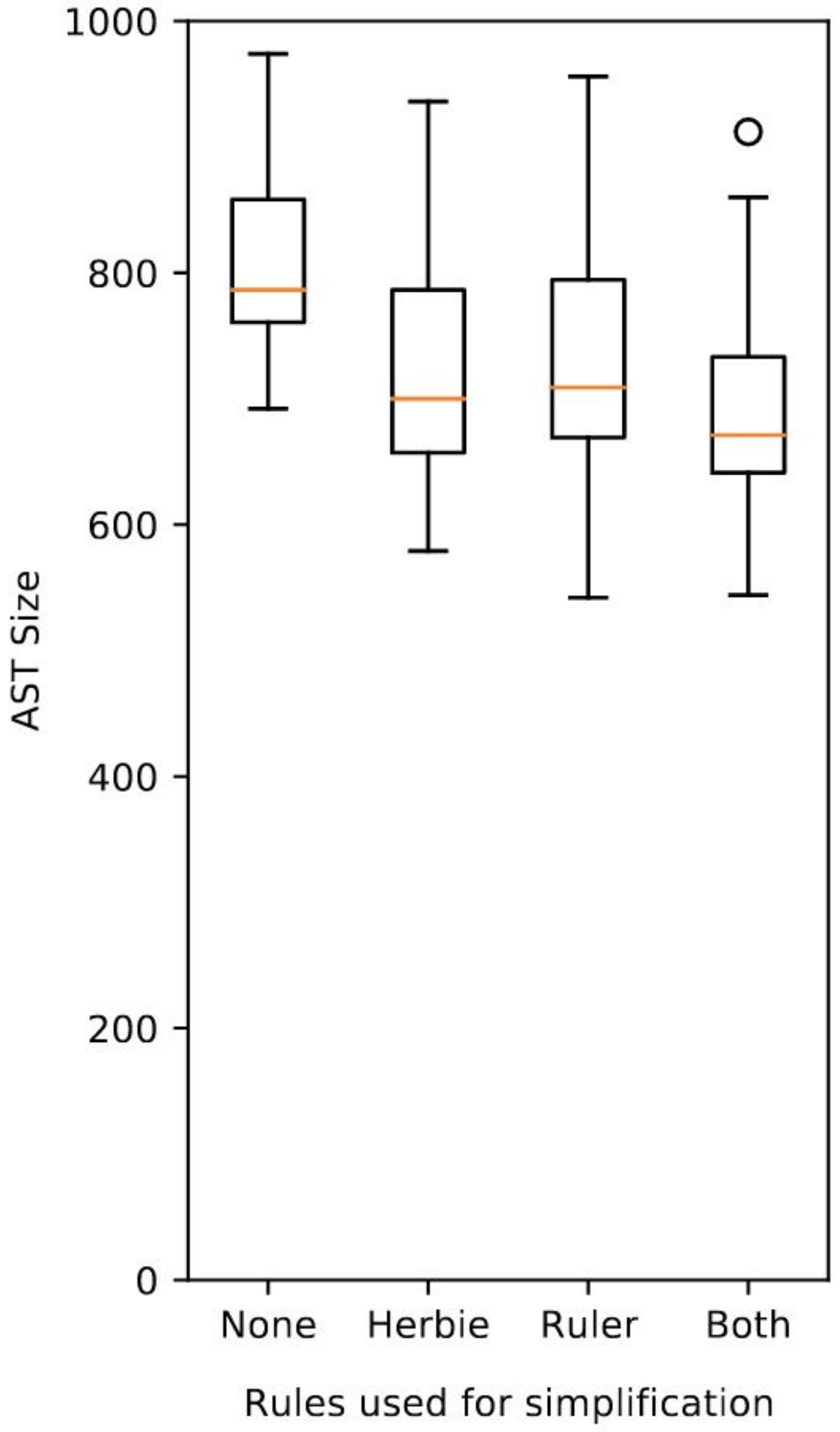
Herbie: Herbie without any changes

Ruler: Herbie with Ruler's rules

Both: Herbie with both original and Ruler's rules

Ruler's rules are at least as good as the original Herbie rules

Rational Herbie: comparing AST size



See paper for more results!

None: Remove all rules

Herbie: Herbie without any changes

Ruler: Herbie with Ruler's rules

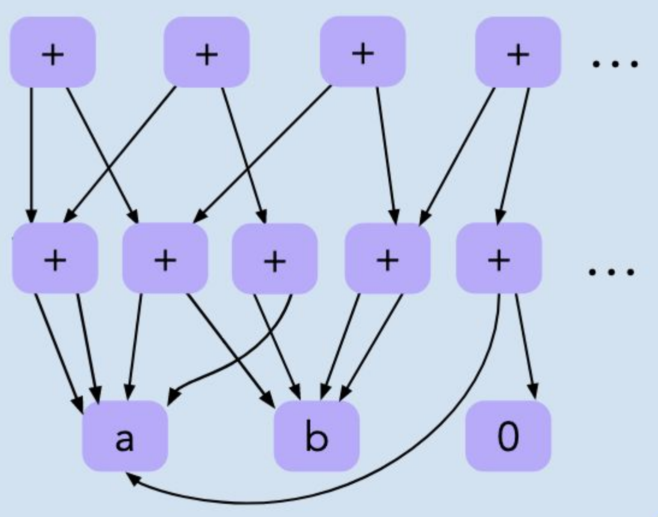
Both: Herbie with both original and Ruler's rules

Ruler's rules are at least as good as the original Herbie rules

Rewrite Rule Inference Using Equality Saturation

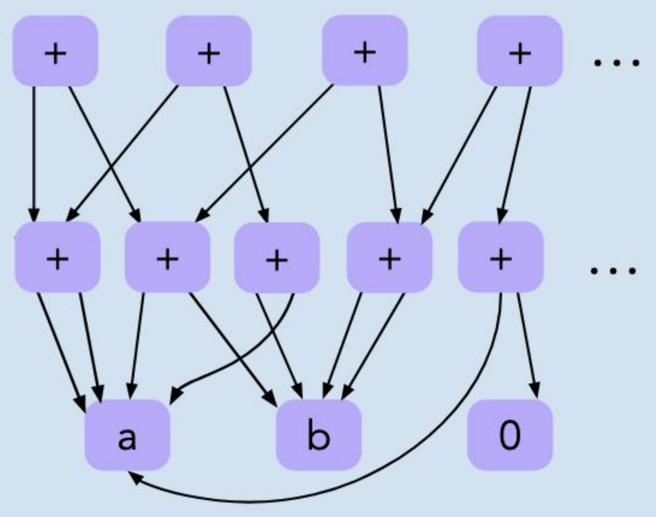
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”



Filter candidates to get final ruleset

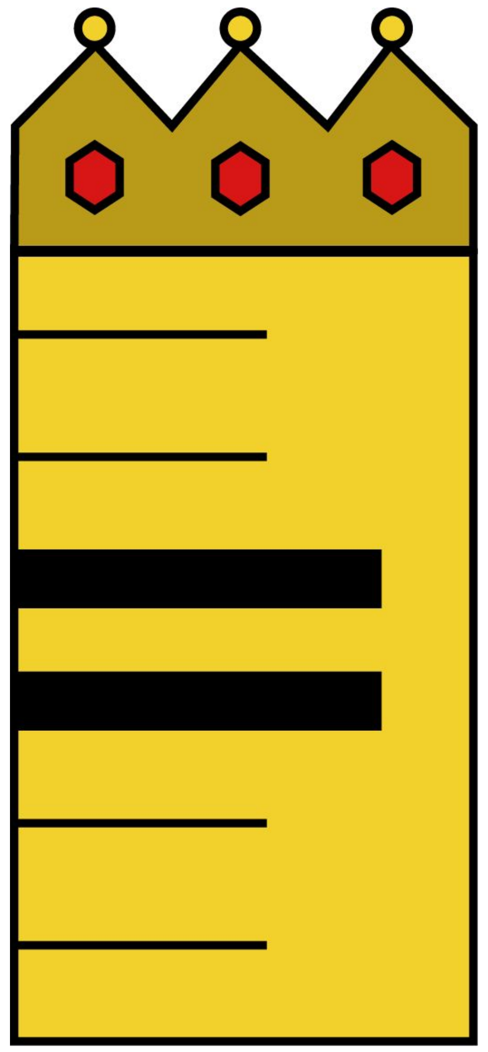
Remove redundant rules

x + 0 ↔ 0 + x

y + 0 ↔ 0 + y

x + y ↔ y + x

Equality Saturation improves all three steps!

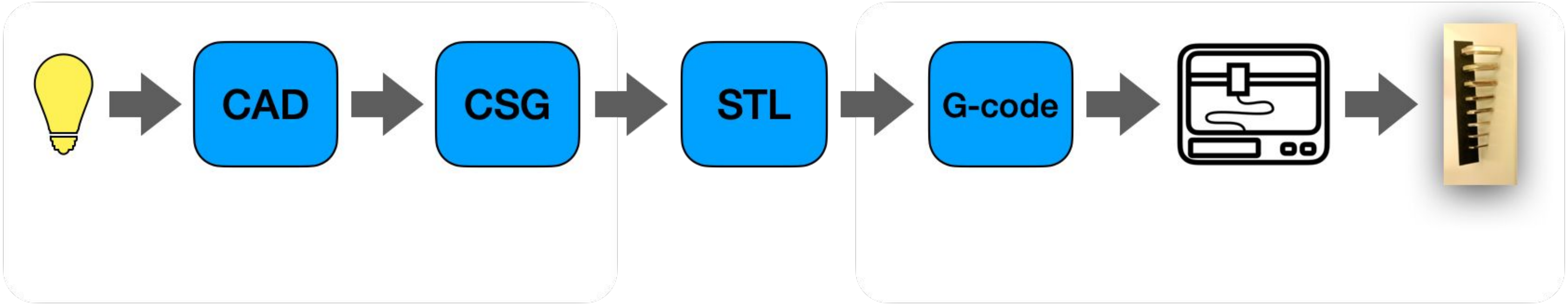


Ruler : https://github.com/uwplse/ruler

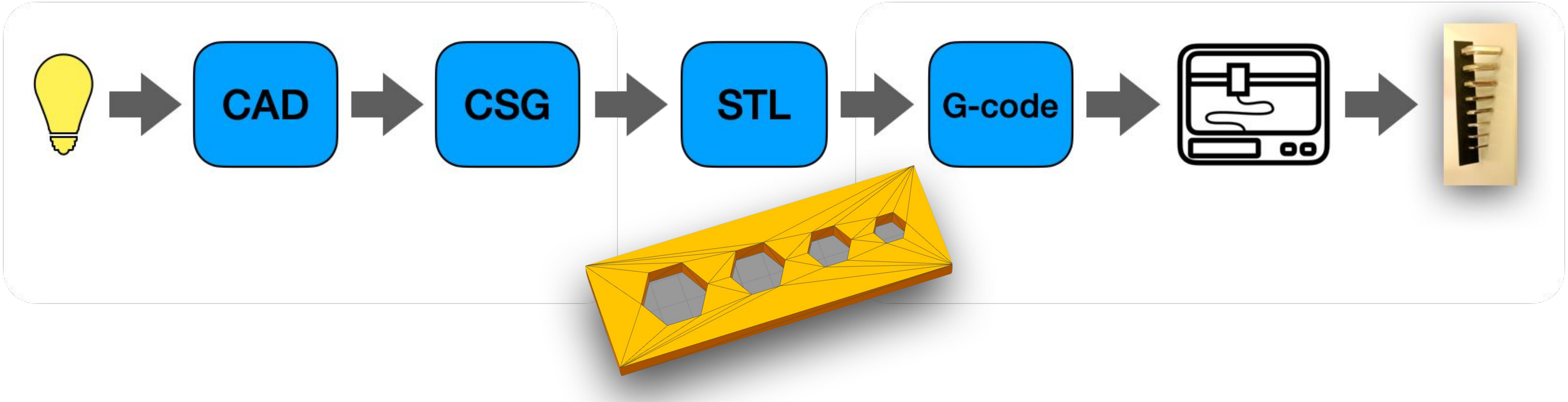
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- ✓ Deferred invariant maintenance & batching
- ✓ Relational e-matching [POPL 2022]
- ✓ E-class analyses
- ✓ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ❏ Applications
  - ❏ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ❏ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …

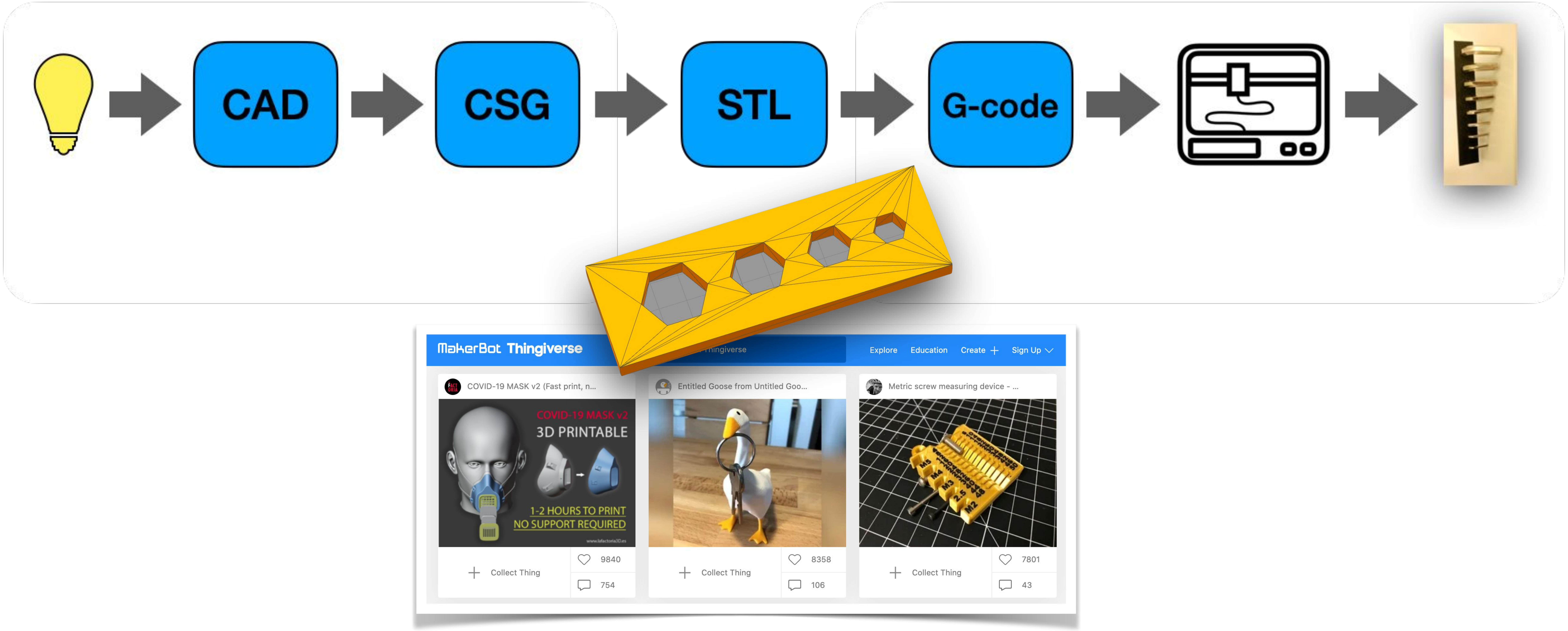
Manufacturing is compilation!



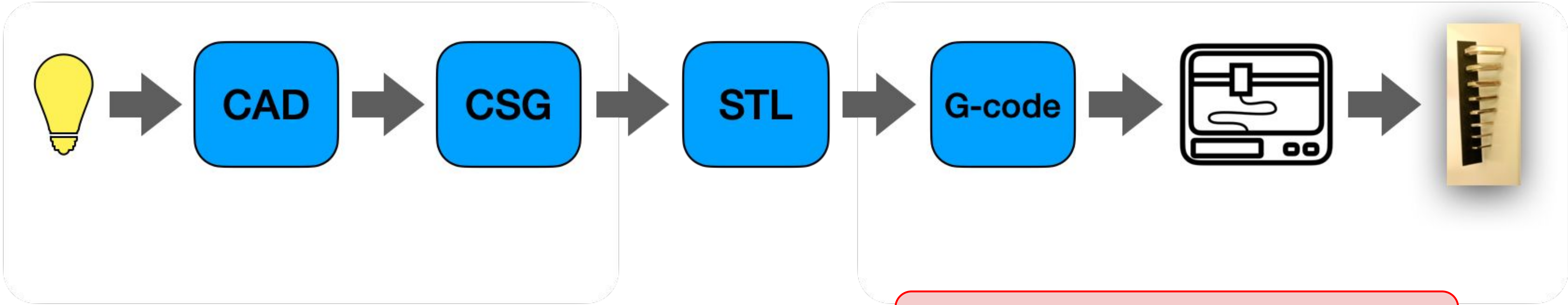
Manufacturing is compilation!

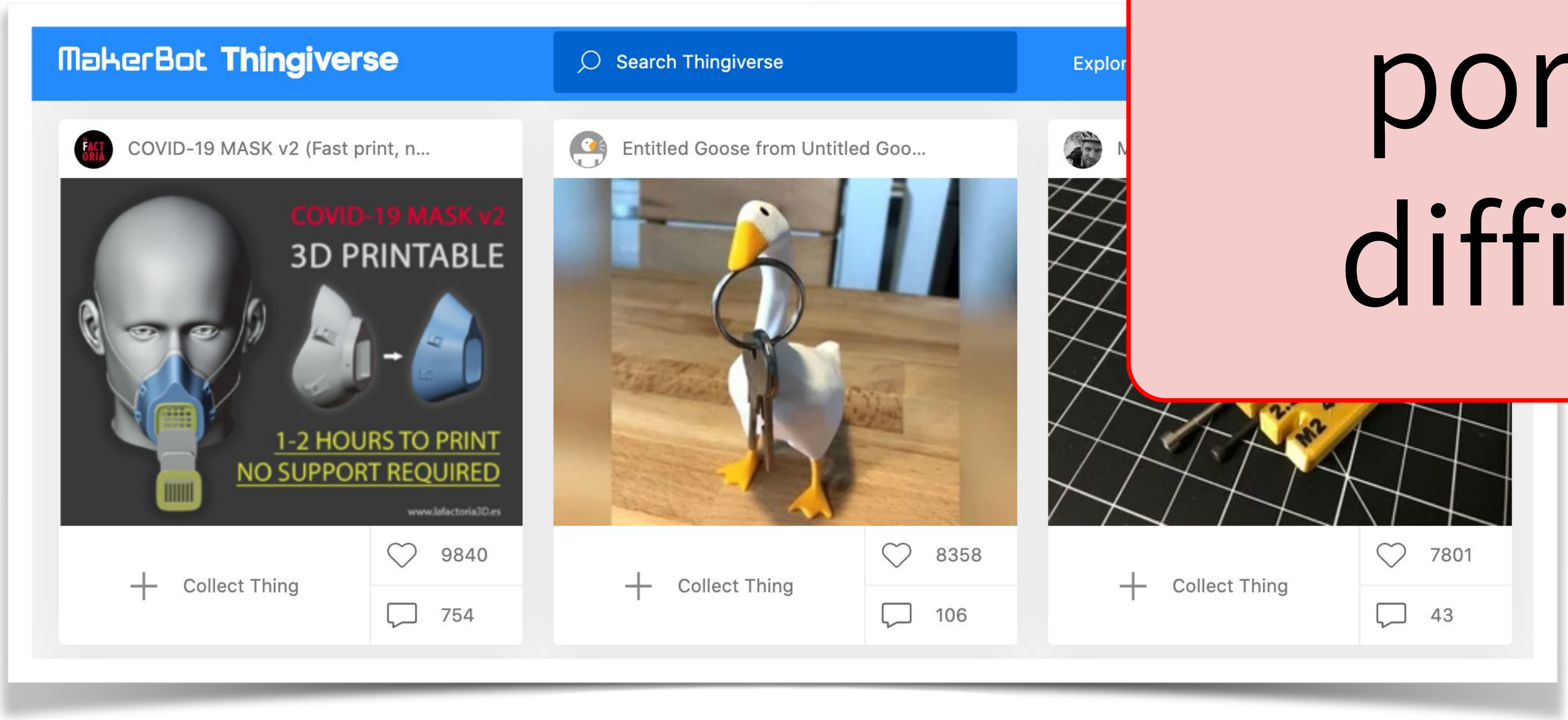


Design is programming!



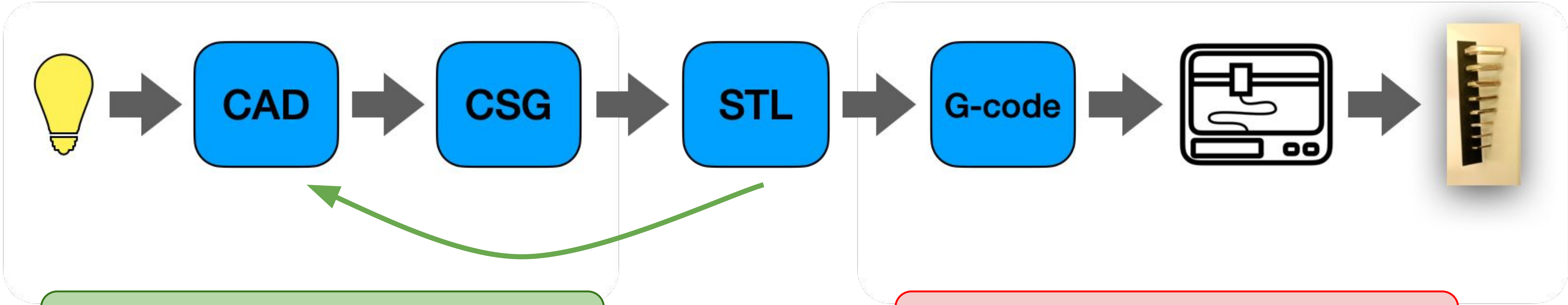
Design is programming!



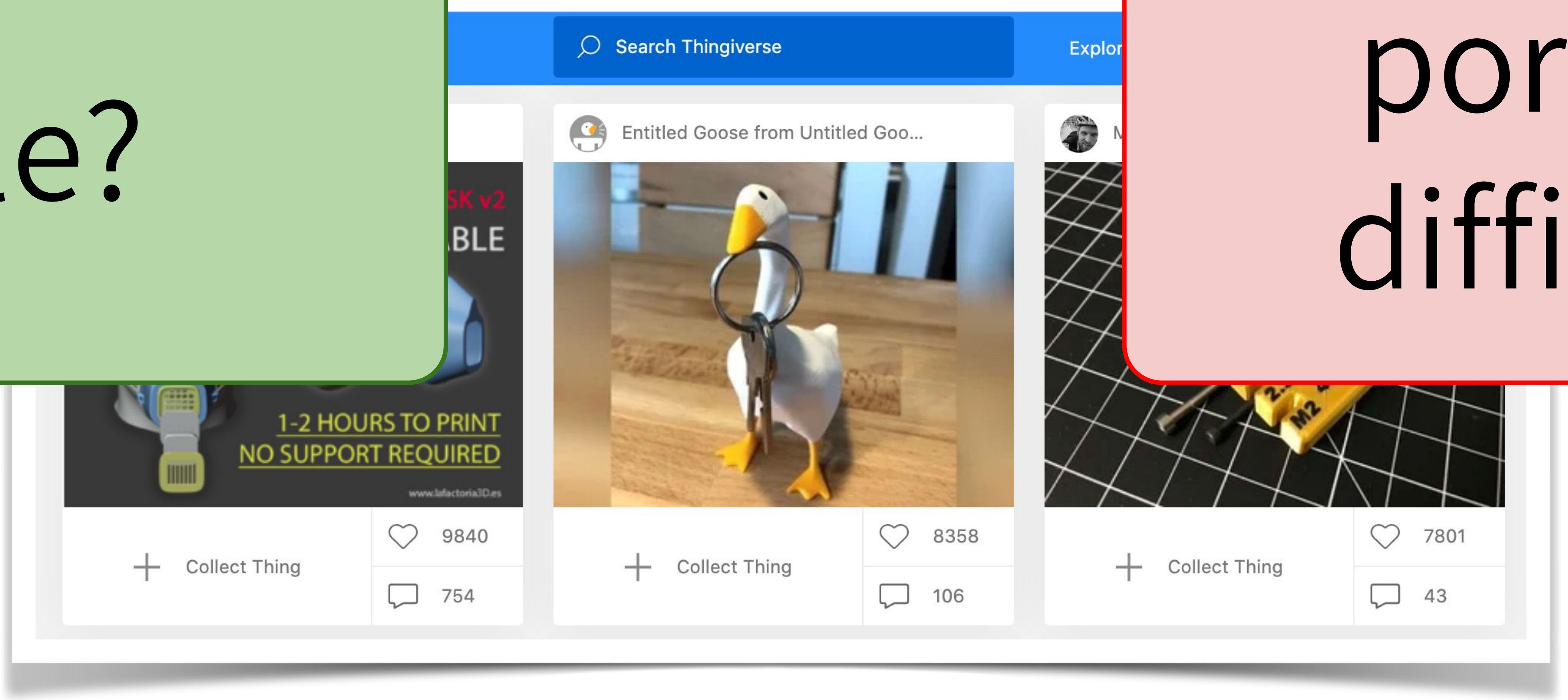


portable, BUT difficult to edit

Design is programming!



decompile?



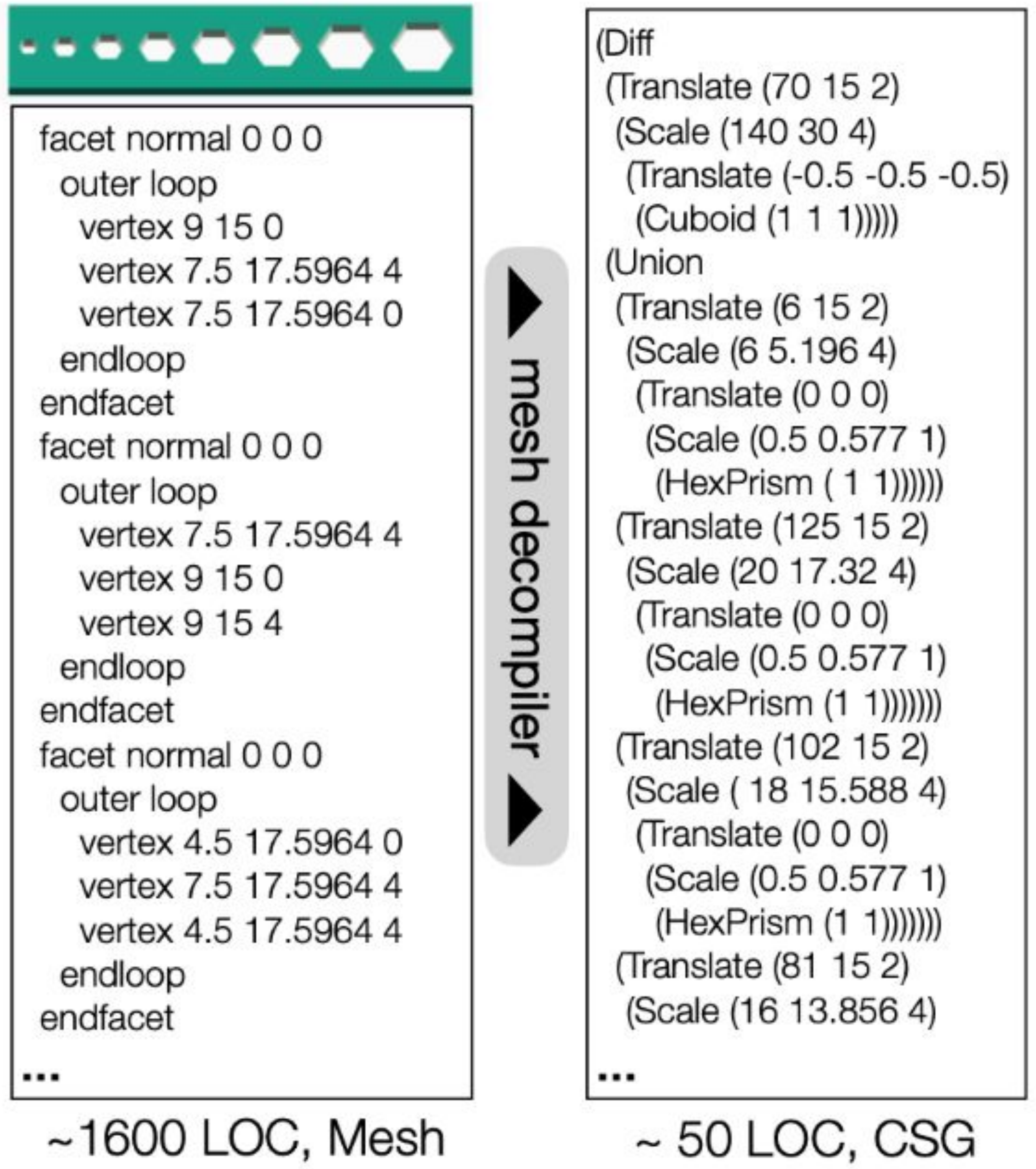
portable, BUT difficult to edit

Szalinski

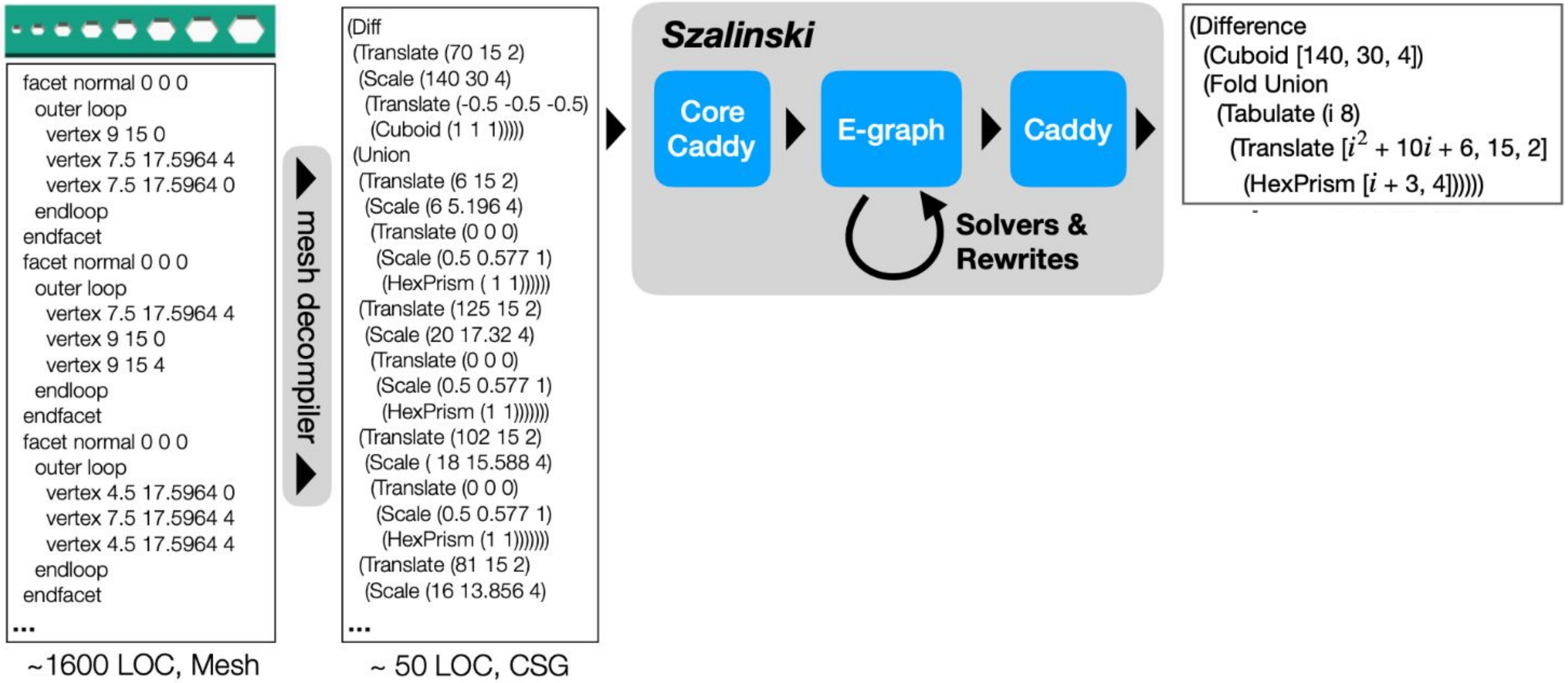
```
facet normal 0 0 0
  outer loop
    vertex 9 15 0
    vertex 7.5 17.5964 4
    vertex 7.5 17.5964 0
  endloop
endfacet
facet normal 0 0 0
  outer loop
    vertex 7.5 17.5964 4
    vertex 9 15 0
    vertex 9 15 4
  endloop
endfacet
facet normal 0 0 0
  outer loop
    vertex 4.5 17.5964 0
    vertex 7.5 17.5964 4
    vertex 4.5 17.5964 4
  endloop
endfacet
...
```

~1600 LOC, Mesh

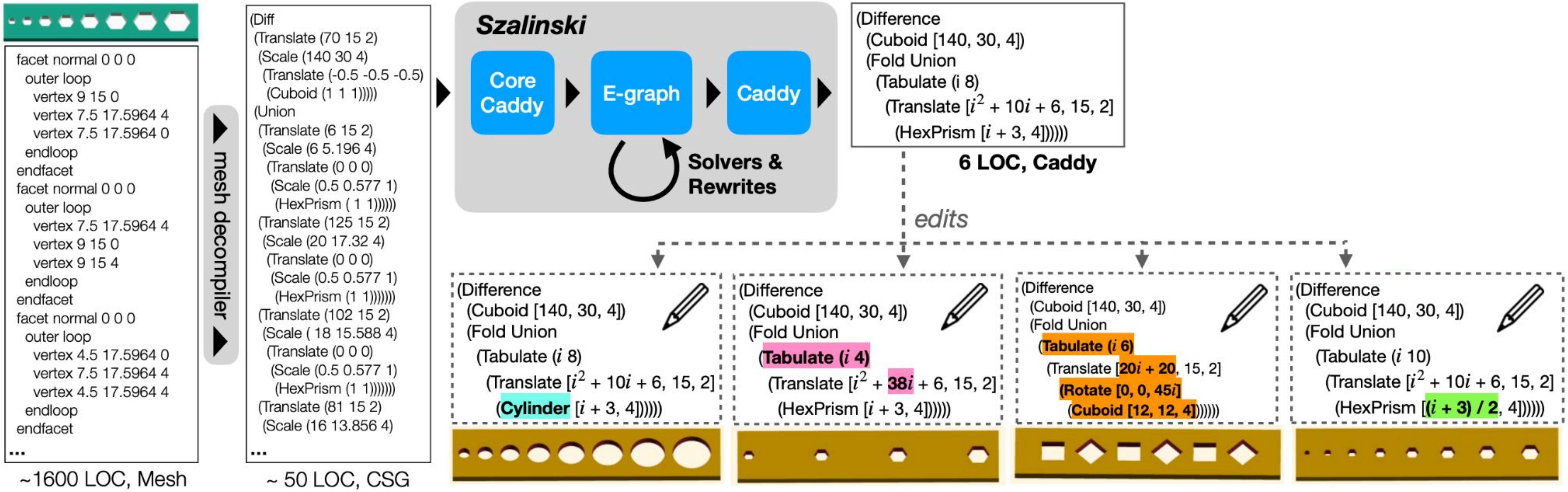
Szalinski


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

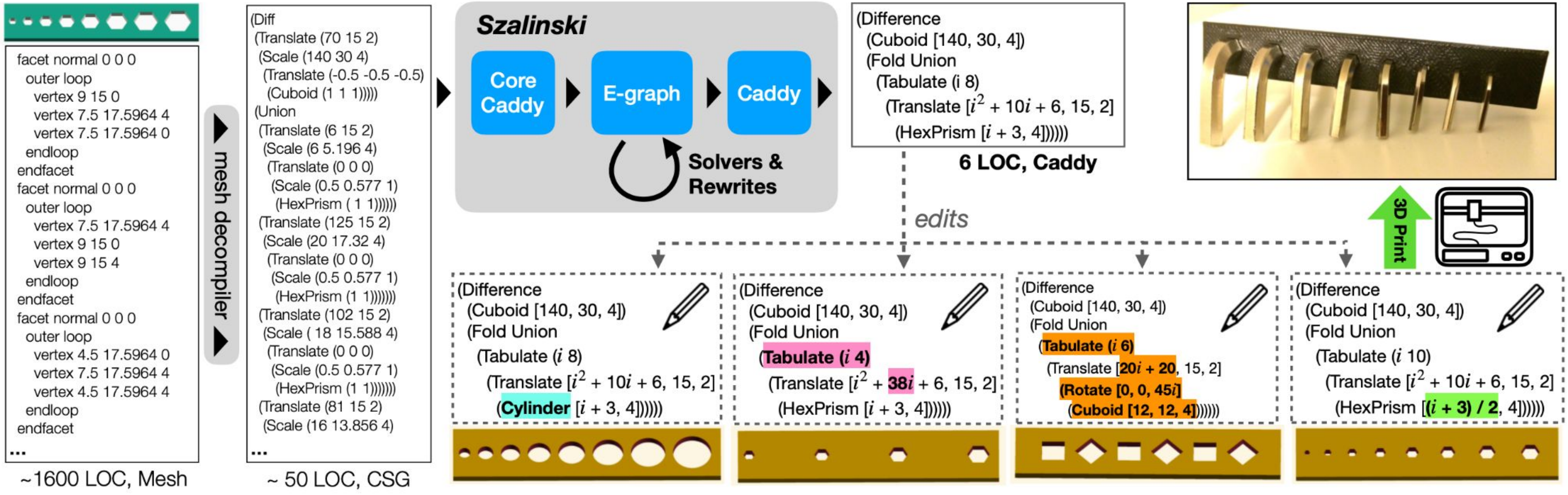

facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~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])))))

Szalinski



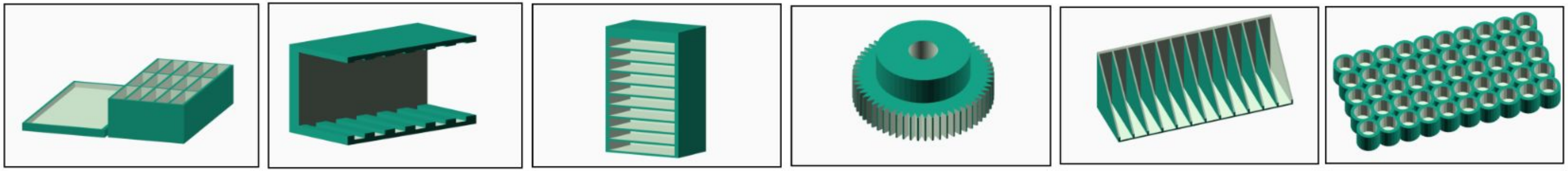
facet normal 0 0 0
outer loop
vertex 9 15 0
vertex 7.5 17.5964 4
vertex 7.5 17.5964 0
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 7.5 17.5964 4
vertex 9 15 0
vertex 9 15 4
endloop
endfacet
facet normal 0 0 0
outer loop
vertex 4.5 17.5964 0
vertex 7.5 17.5964 4
vertex 4.5 17.5964 4
endloop
endfacet
...
~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
(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 [PLDI 2020]



Szalinski [PLDI 2020]

- thousands of models decompiled w/ egg, all < 1 second



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

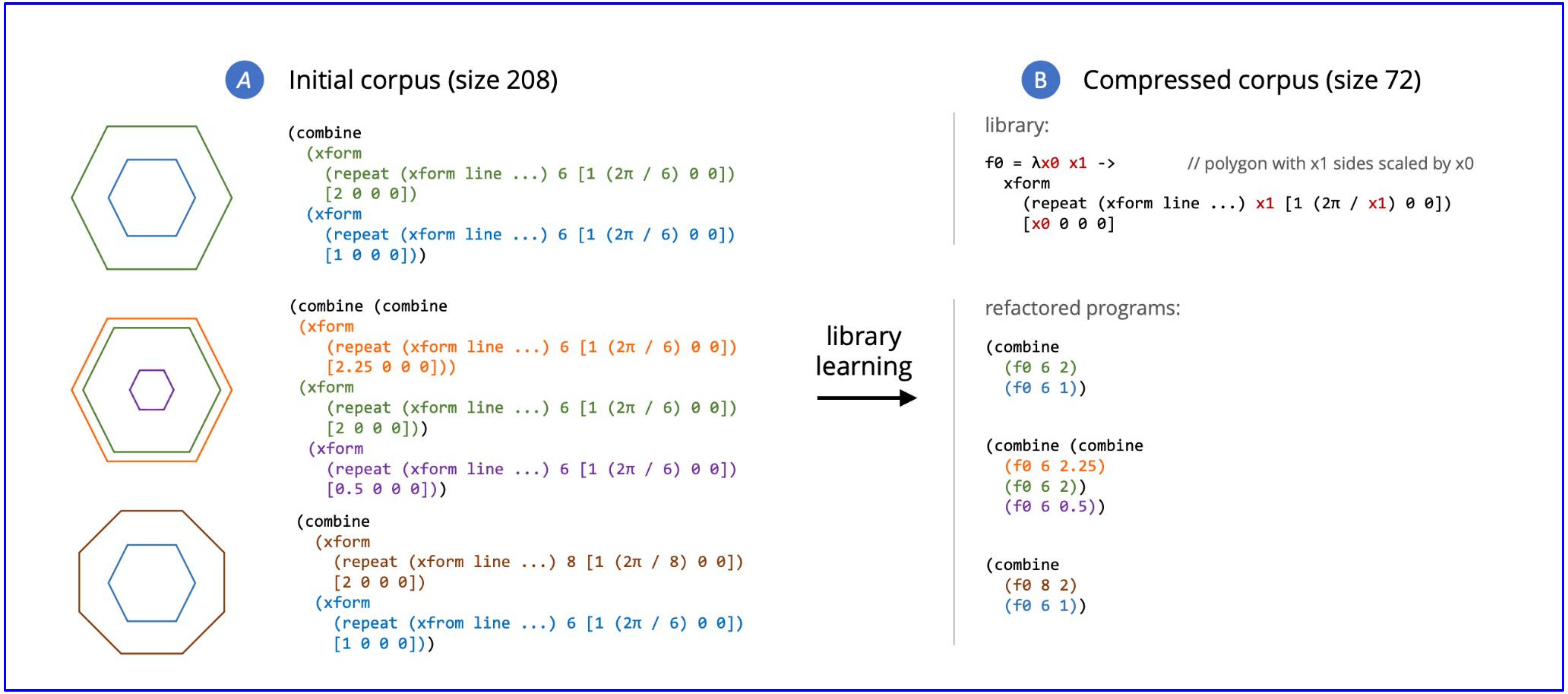
```
(Difference
 (Translate [-57, -3, 3]
  (Cuboid [117, 75, 175]))
 (Fold Union
  (Tabulate (i 10)
   (Translate [-51, -3, 16 * i + 6]
    (Cuboid [105, 58, 13])))))
```

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

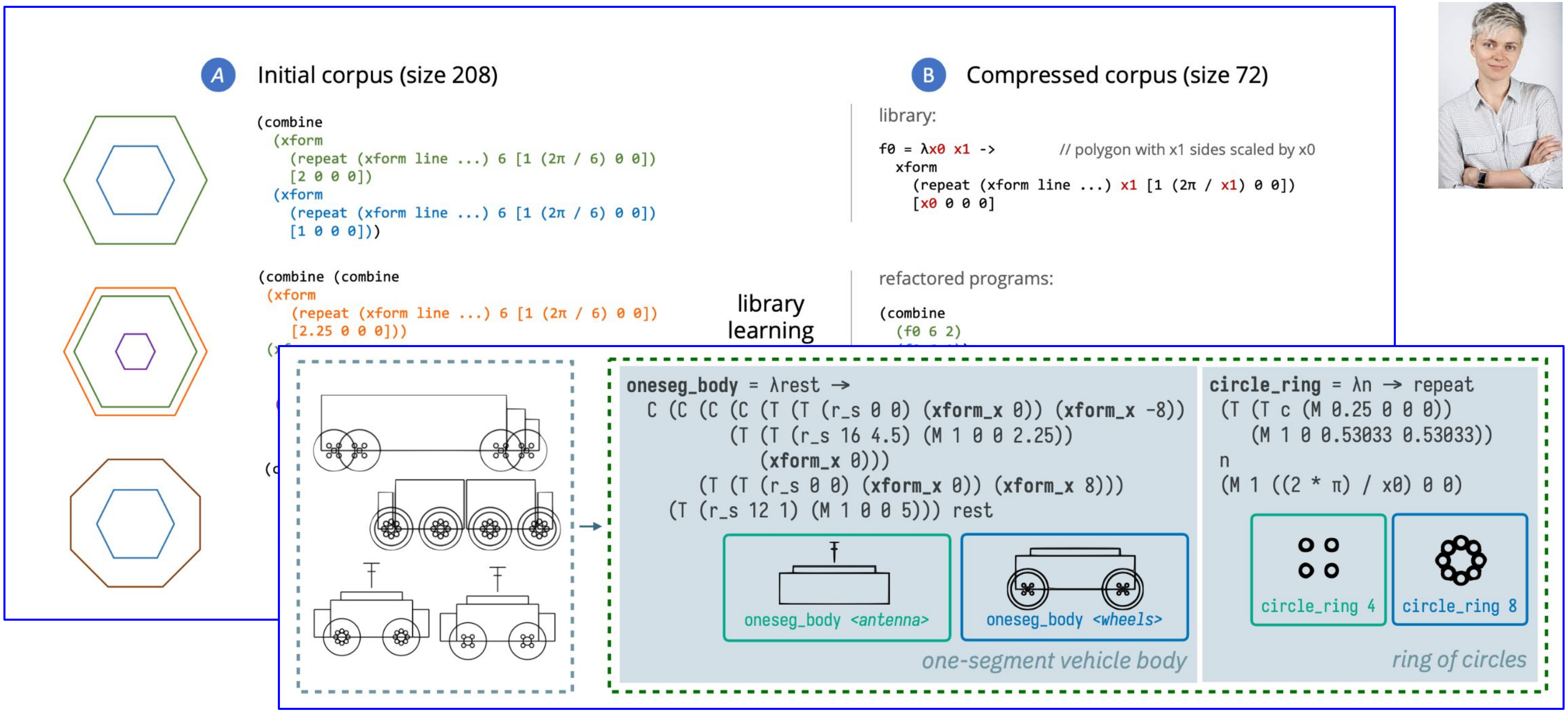
```
(Fold Union
 (Tabulate (i 12)
  (Translate [0, 13* i, 0]
   (Fold Difference
    (List
     (Cuboid [53.1 14.5 58])
     (Translate [1.5, 1.5, 1.5]
       (Cuboid [51.6, 11.5, 56.6]))
     (Translate [0 0 58]
      (Rotate [0, 45, 0]
       (Cuboid [101.5, 14.5, 100])))))))
```

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

Library learning with Babble [POPL 2023]



Library learning with Babble [POPL 2023]



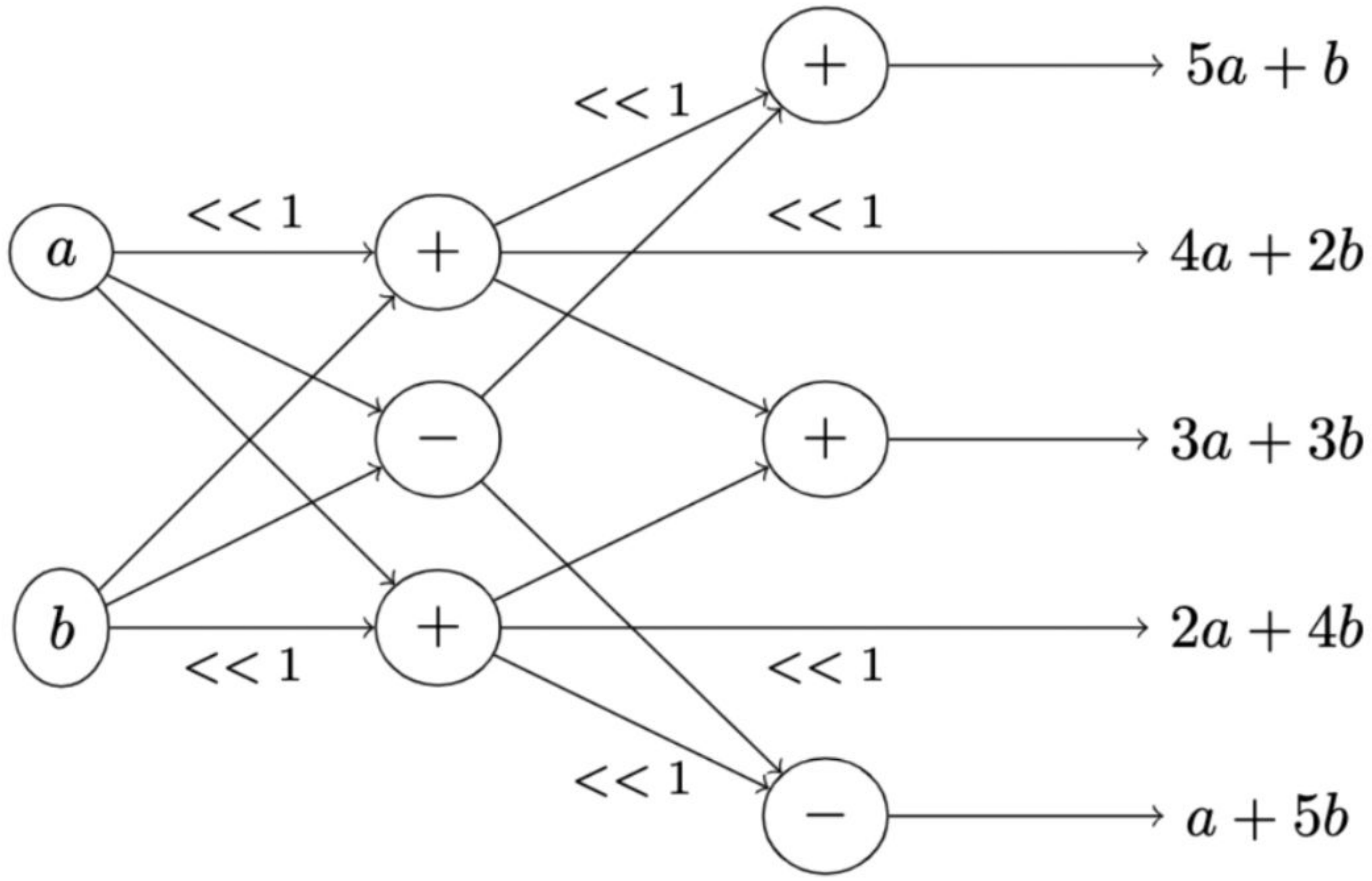
Short Proofs for TV + debugging [FMCAD 2022]

Intel Case Study

Multi-operation circuit optimization and translation validation with egg

4.7 hours -> 2.3 hours

2 inputs



5 outputs

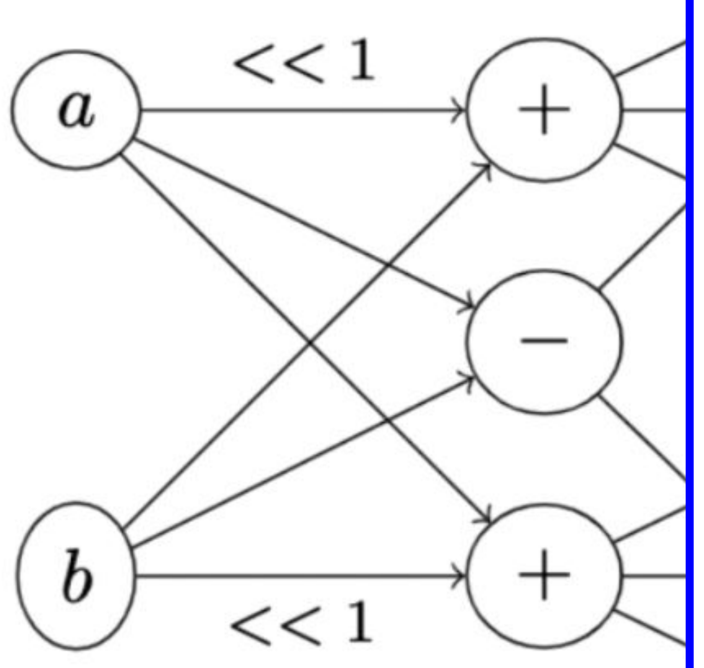
Short Proofs for TV + debugging [FMCAD 2022]

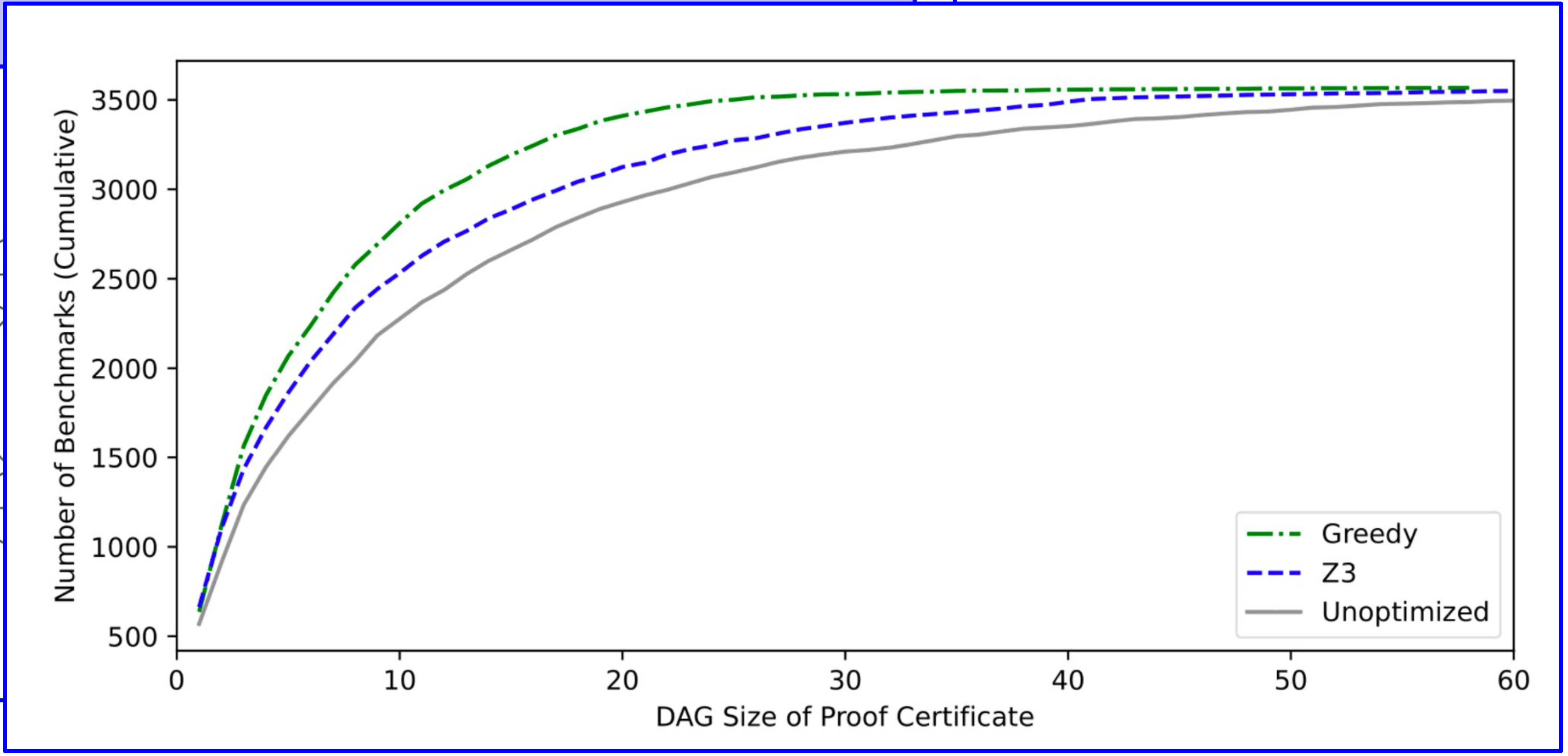
Intel Case Study

Multi-operation circuit optimization and translation validation with egg

4.7 hours -> 2.3 hours

2 inputs





egg case studies

- Herbie: floating point — batching — *3000x faster*
- SPORES: linear algebra kernels — shape analysis — *1.2-5x better*
- Tensat: ML compute graphs — generic library — *23% better, 48x faster*
- Szalinski: CAD synthesis — dynamic rewrites — *12,000 part eval*
  < 1s synthesis
- …, TVM, Java testing, vectorization,
  hw/sw co-design, educational problems, ...

egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- ✓ Deferred invariant maintenance & batching
- ✓ Relational e-matching [POPL 2022]
- ✓ E-class analyses
- ✓ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
- ✓ Applications
  - ✓ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  - ✓ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …