Enabling Robust Equality Saturation Through Flexible Theory Exploration

Anjali Pal

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

PLSE

1. Fast, Flexible, Robust **Term Rewriting** via Equality Saturation

2. Fast, Flexible, Robust **Rule Inference** for Equality Saturation

Term Rewriting

- Synthesis
- Canonicalization
- Equivalence Checking
- SMT Solvers
- Optimization
- Verification
- Code Generation
- Symbolic Evaluation
- And more!

Term Rewriting: Optimization

Task: Compile the program

```
if True:
  y = x + x
  r = y * 2 + 0
else:
  r = x * 1

return r
```

Term Rewriting: Optimization

Task: Compile the program

```
if True:
  y = x + x
  r = y * 2 + 0
else:
  r = x * 1

return r
```

```
return 4 * y
```

```
x + x ⟹ 2 * x
if True then x else y ⟹ x
x + 0 ⟹ x
2 * 2 ⟹ 4
```

Term Rewriting: Equivalence Checking

Task: Determine if refactor is safe

```python
def is_even(n):
 return n % 2 == 0


def foo(x, y):
 if is_even(x):
   return x + y
 else:
   return x - y
```

```python
def is_odd(n):
 return n % 2 ≠ 0


def foo(x, y):
 if is_odd(x):
   return x - y
 else:
   return x + y
```

Term Rewriting: Equivalence Checking

Task: Determine if refactor is safe

```
def is_even(n):
 return n % 2 == 0


def foo(x, y):
 if is_even(x):
   return x + y
 else:
   retur
```

✅

```
def is_odd(n):
 return n % 2 != 0


def foo(x, y):
 if is_odd(x):
   return x - y

   x + y
```

```
is_even(a)  ==> a % 2 == 0
is_odd(a)   ==> a % 2 != 0
x == y      ==> !(x != y)
if A then B else C ==>
     if !A then C else B
```

Term Rewriting: Canonicalization

Task: Put in Polynomial Normal Form

$$2(x + 1) + 3xy + xy + 4$$

Term Rewriting: Canonicalization

Task: Put in Polynomial Normal Form

$$2(x + 1) + 3xy + xy + 4$$

$$4xy + 2x + 6$$

```
a + (b + c)   ⟹ (a + b) + c
a * m + b * m ⟹ (a + b) * m
a + b         ⟹ b + a
c * (a + b)   ⟹ c * a + c * b
              ...
```

(a * 2) / 2 → a

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

Rewrite Rules

```
(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

(a * 2) / 2 ⟹ a * (2 / 2) ⟹ a * 1 ⟹ a

Rewrite Rules

```
(x * y) / z ⟹ x * (y / z)        x * 2 ⟹ x << 1
  x / x       ⟹ 1                 x * y ⟹ y * x
  x * 1       ⟹ x                 x     ⟹ x * 1
```

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

$$(a * 2) / 2 \Longrightarrow (a \ll 1) / 2$$

stuck

Rewrite Rules

```
(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

(a * 2) / 2 ⟹ (2 * a) / 2 ⟹ (a * 2) / 2 ⟹ ...

diverge

Rewrite Rules

(x * y) / z	⟹	x * (y / z)	x * 2	⟹	x << 1
x / x	⟹	1	x * y	⟹	y * x
x * 1	⟹	x	x	⟹	x * 1

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

$$a \Longrightarrow a * 1 \Longrightarrow (a * 1) * 1 \Longrightarrow \ldots$$

infinite size

Rewrite Rules

```
(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

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
```

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

But critical for other inputs!

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 → 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
```

(a * 2) / 2 → a

All of them at once!

USEFUL

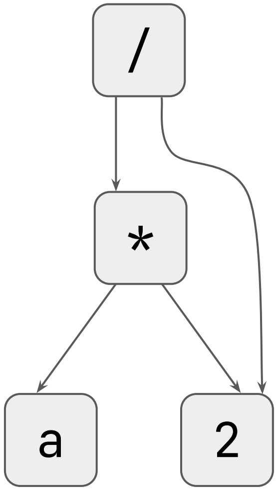
```
(x * y) / z ⟹ x * (y / z)
 x / x      ⟹ 1
 x * 1      ⟹ x
```

NOT SO USEFUL

```
x * 2 ⟹ x << 1
x * y ⟹ y * x
x     ⟹ x * 1
```

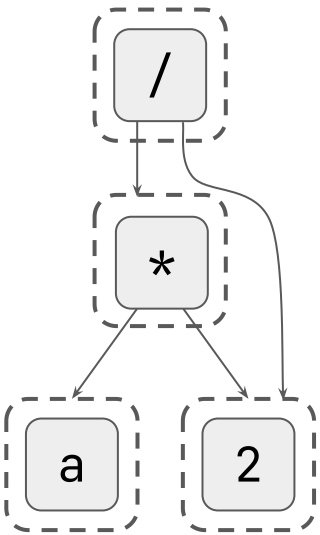
Equivalence Graphs (e-graphs)

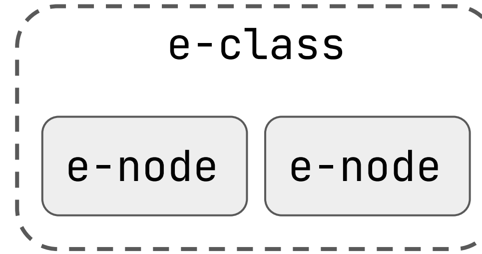
```
(a * 2) / 2
```



Equivalence Graphs (e-graphs)

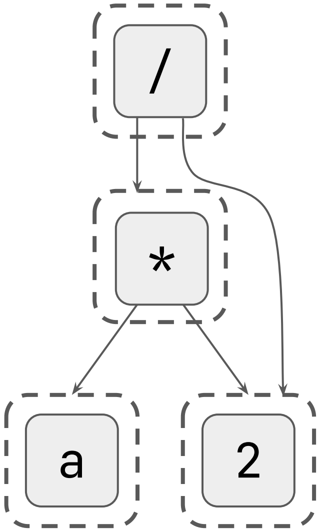
`(a * 2) / 2`





Nelson 1980; Tate et al. 2009; Willsey et al. 2021

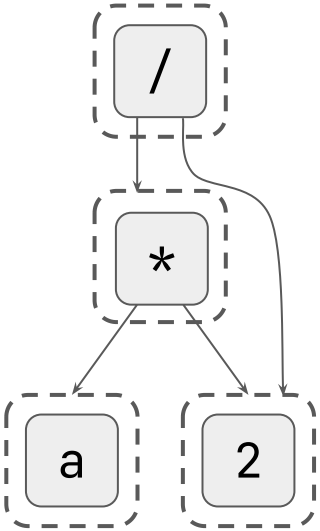
Equality Saturation



x * 2 ⟹ x << 1

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation

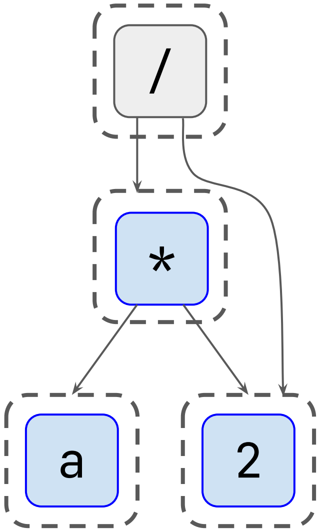


x * 2 ⟹ x << 1

Find a term that looks like the left,
Add a term that looks like the right,
And mark them equivalent

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation

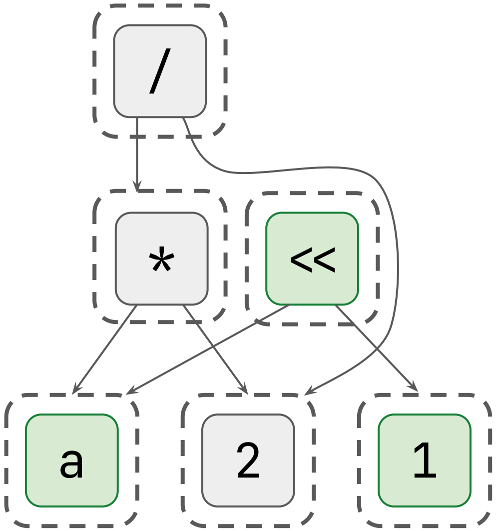


x * 2 ⟹ x << 1

Find a term that looks like the left,
Add a term that looks like the right,
And mark them equivalent

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation

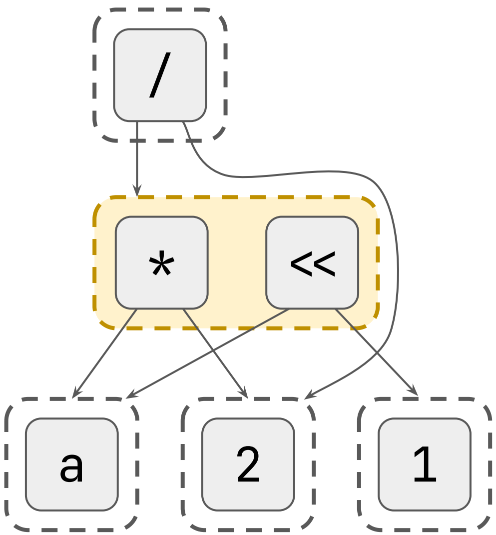


$x * 2 \Longrightarrow x \ll 1$

Find a term that looks like the left,
Add a term that looks like the right,
And mark them equivalent

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation

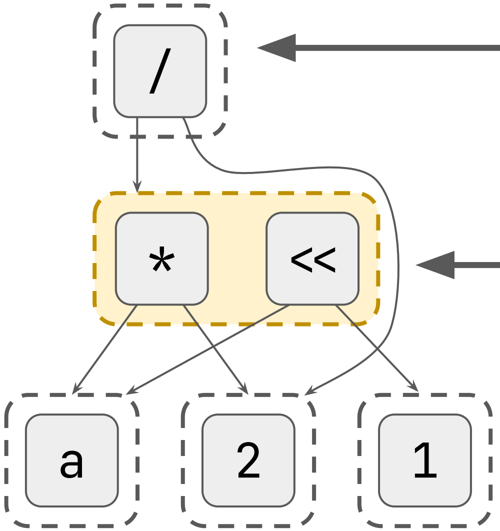


$x * 2 \Longrightarrow x << 1$

Find a term that looks like the left,
Add a term that looks like the right,
And mark them equivalent

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



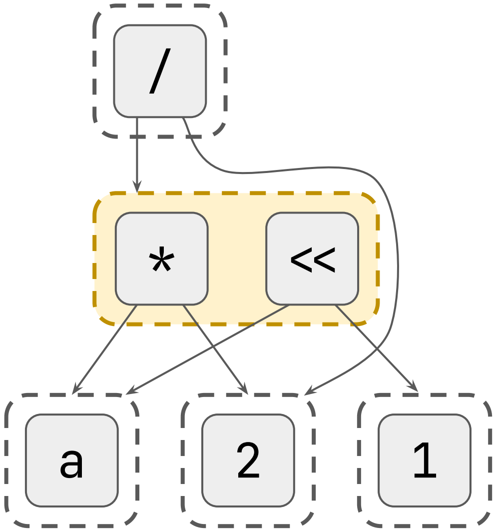
This e-class represents
`(a * 2)/2` **and** `(a << 1)/2`

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

$x * 2 \Longrightarrow x << 1$

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

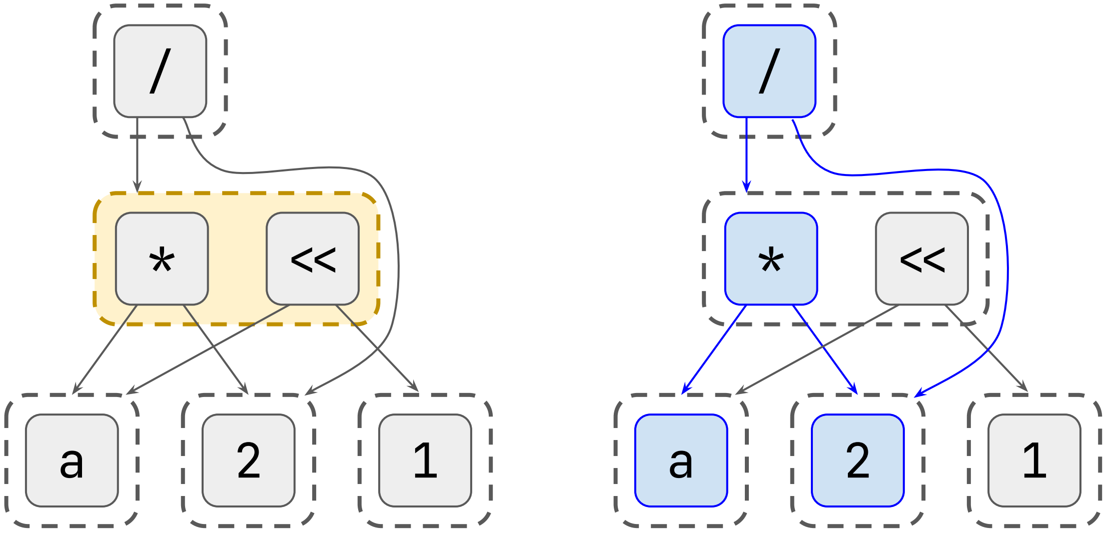
Equality Saturation



`x * 2 ⟹ x << 1`   `(x * y)/z ⟹ x * (y/z)`

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

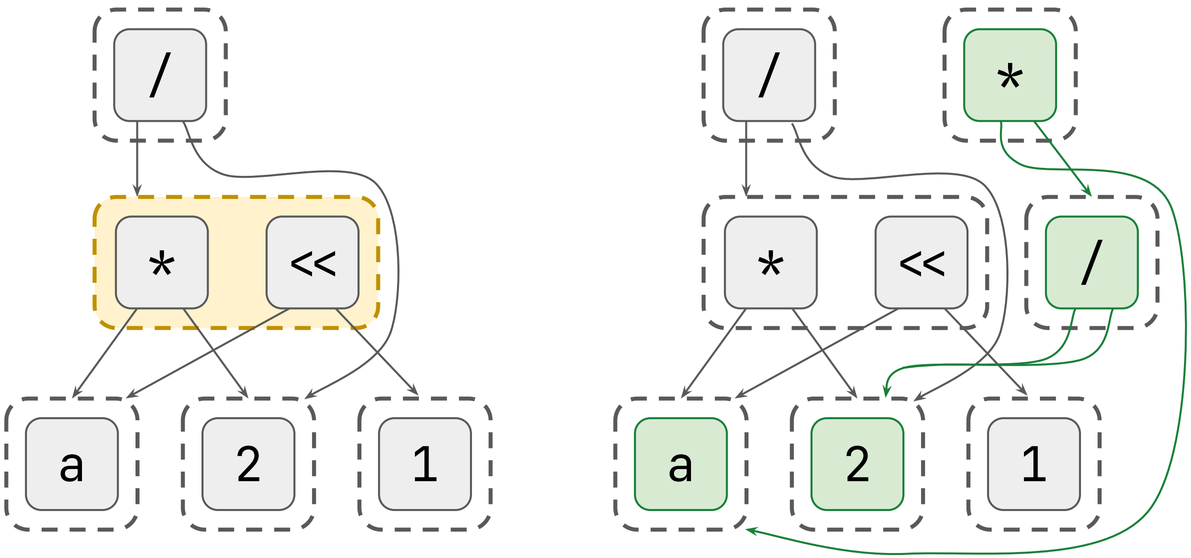
Equality Saturation



x * 2 ⟹ x << 1 (x * y)/z ⟹ x * (y/z)

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

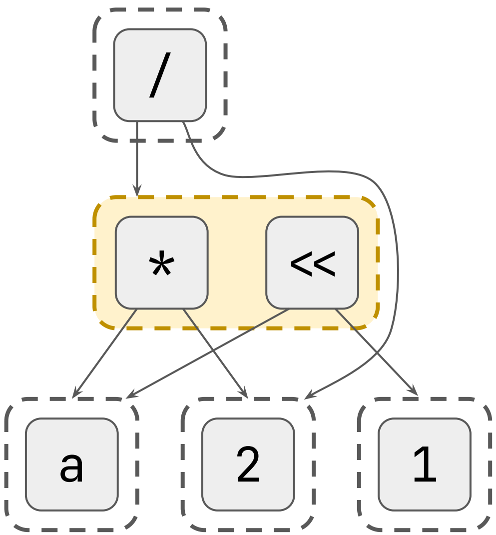
Equality Saturation



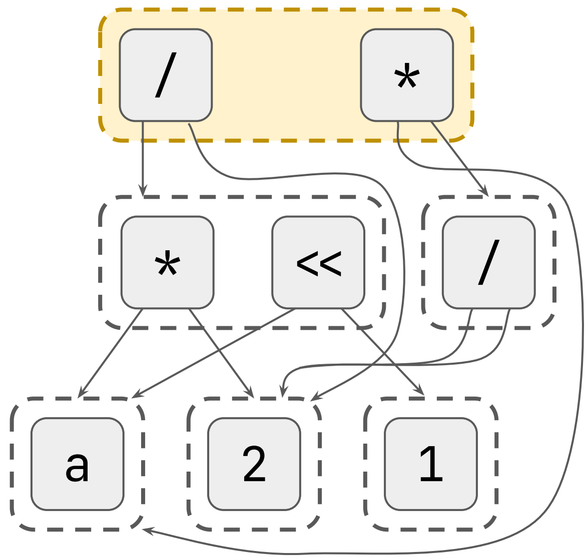
`x * 2 ⟹ x << 1`   `(x * y)/z ⟹ x * (y/z)`

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



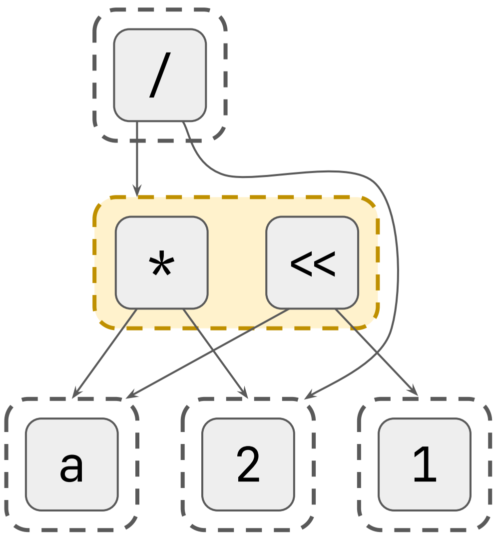
x * 2 ⟹ x << 1



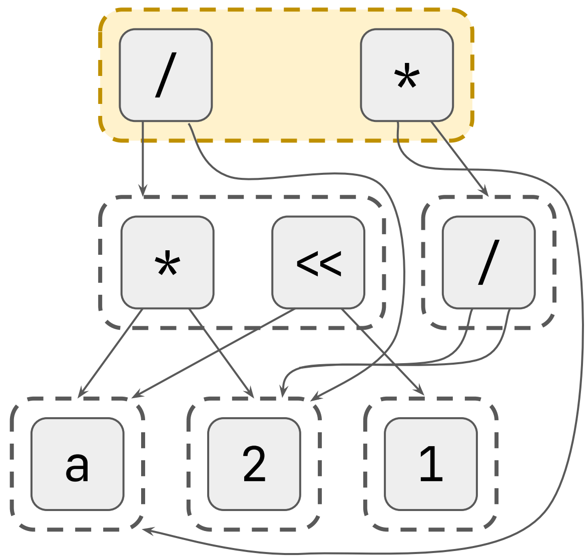
(x * y)/z ⟹ x * (y/z)

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



x * 2 ⟹ x << 1



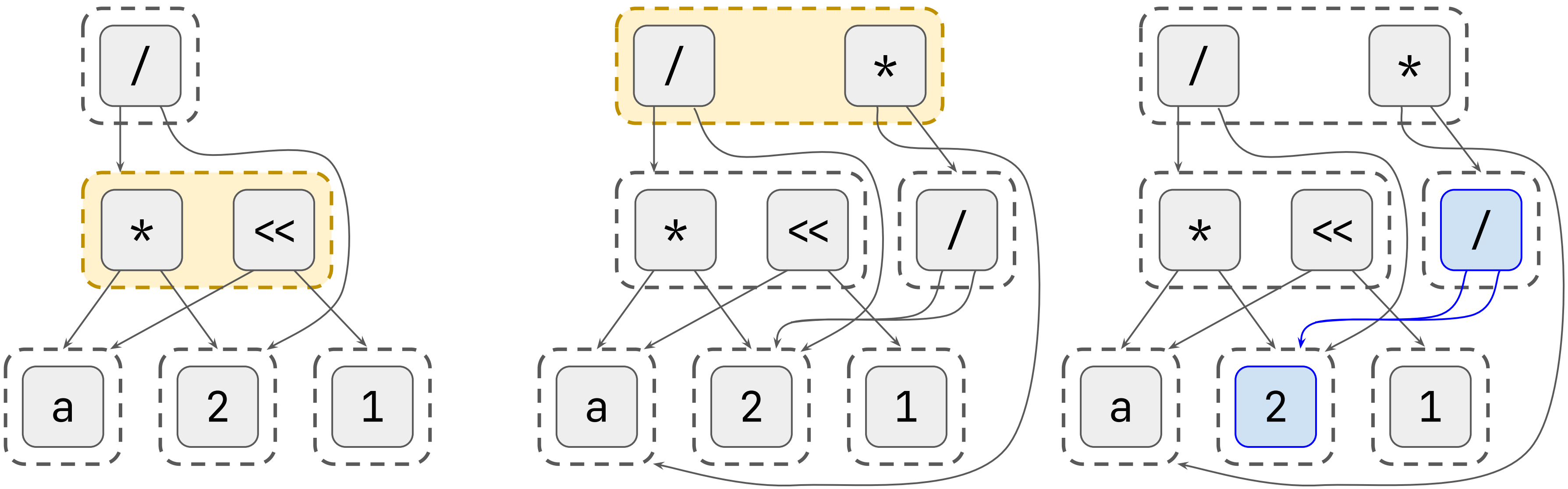
(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1

x * 1 ⟹ x

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



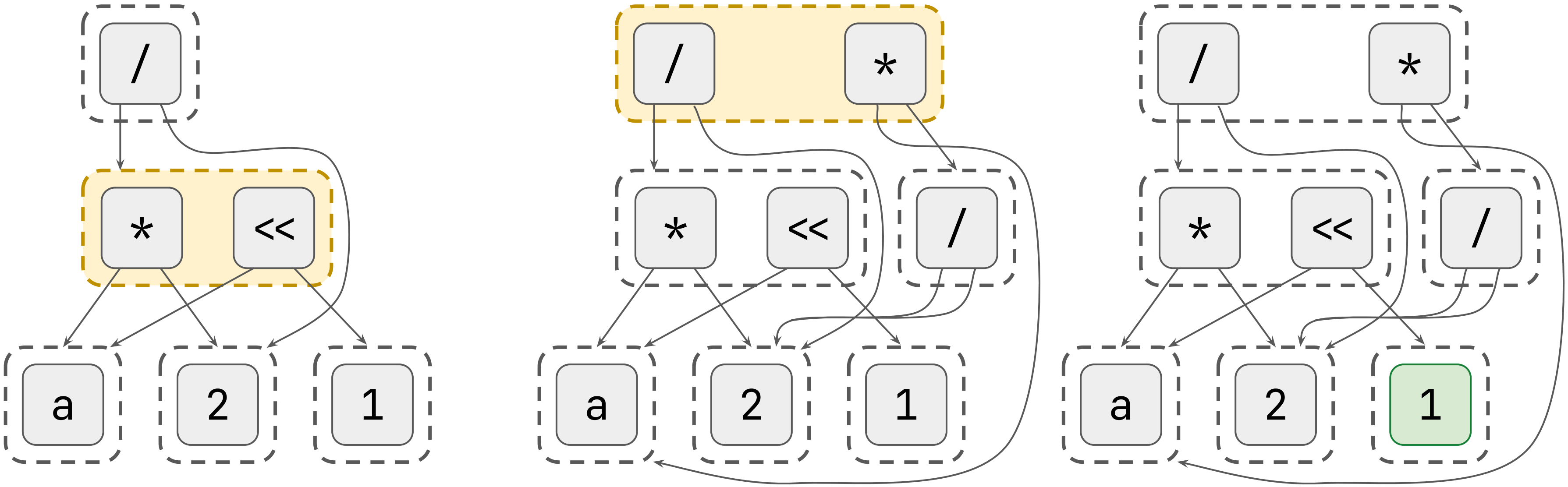
x * 2 ⟹ x << 1

(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1
x * 1 ⟹ x

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



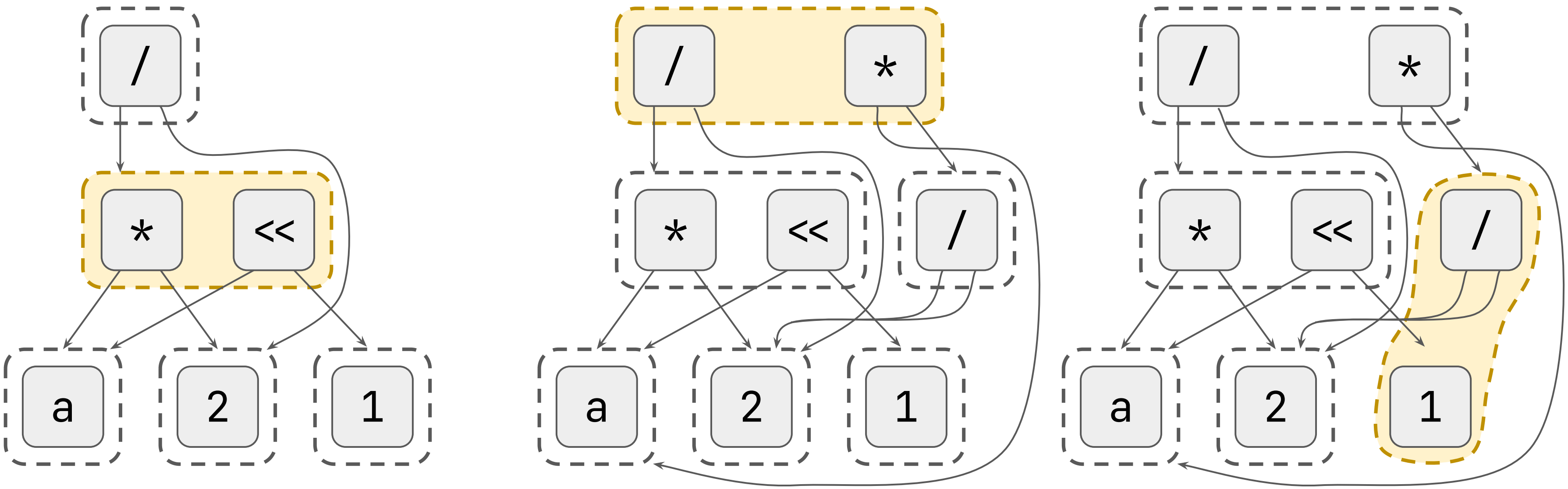
`x * 2 ⟹ x << 1`

`(x * y)/z ⟹ x * (y/z)`

`x / x ⟹ 1`
`x * 1 ⟹ x`

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



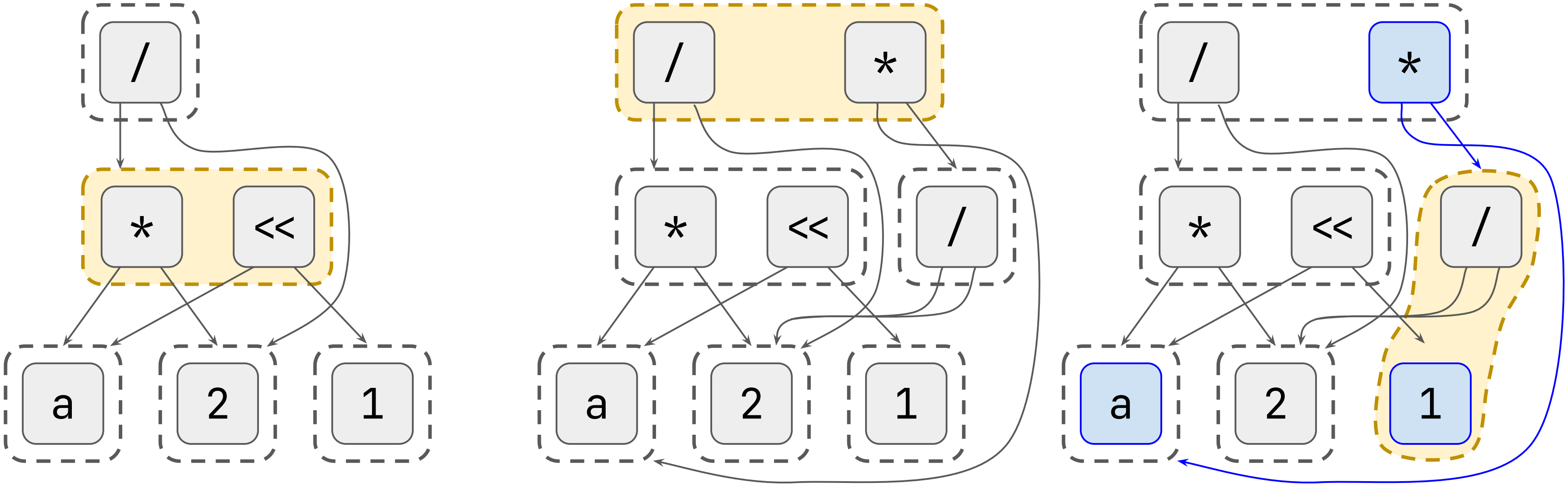
`x * 2 ⟹ x << 1`

`(x * y)/z ⟹ x * (y/z)`

`x / x ⟹ 1`
`x * 1 ⟹ x`

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



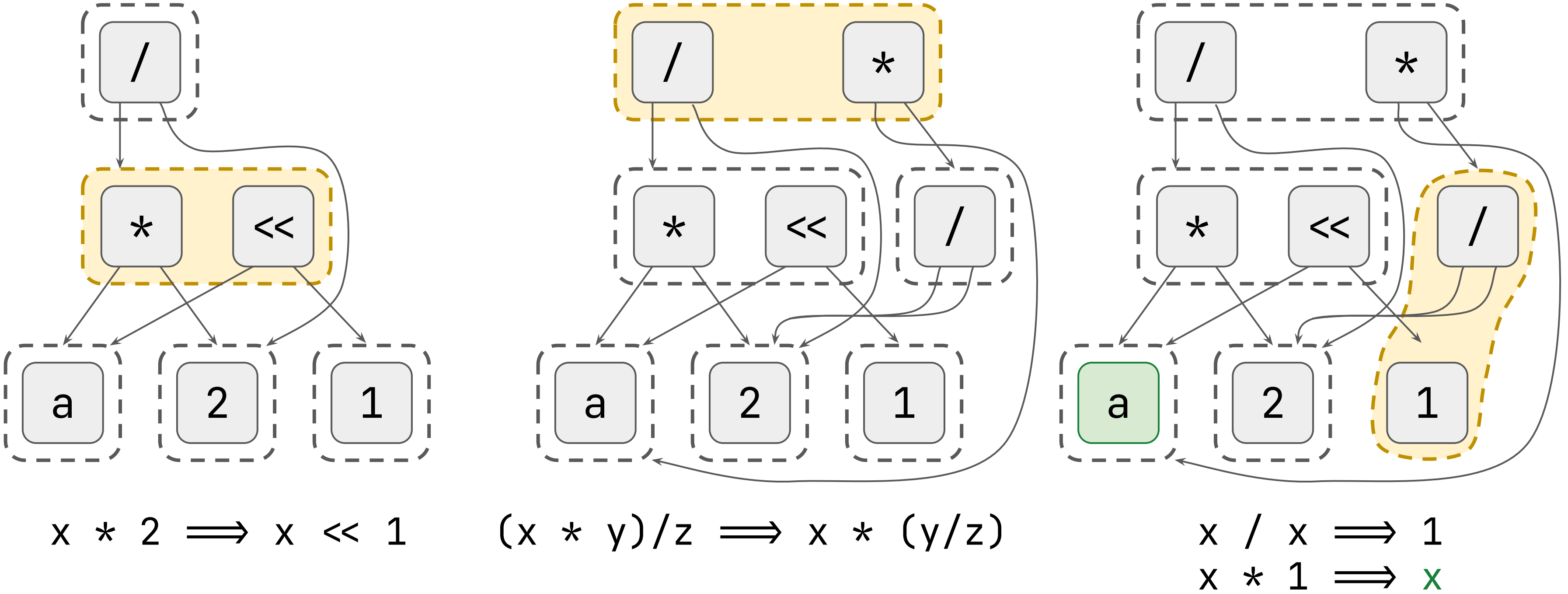
x * 2 ⟹ x << 1

(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1
x * 1 ⟹ x

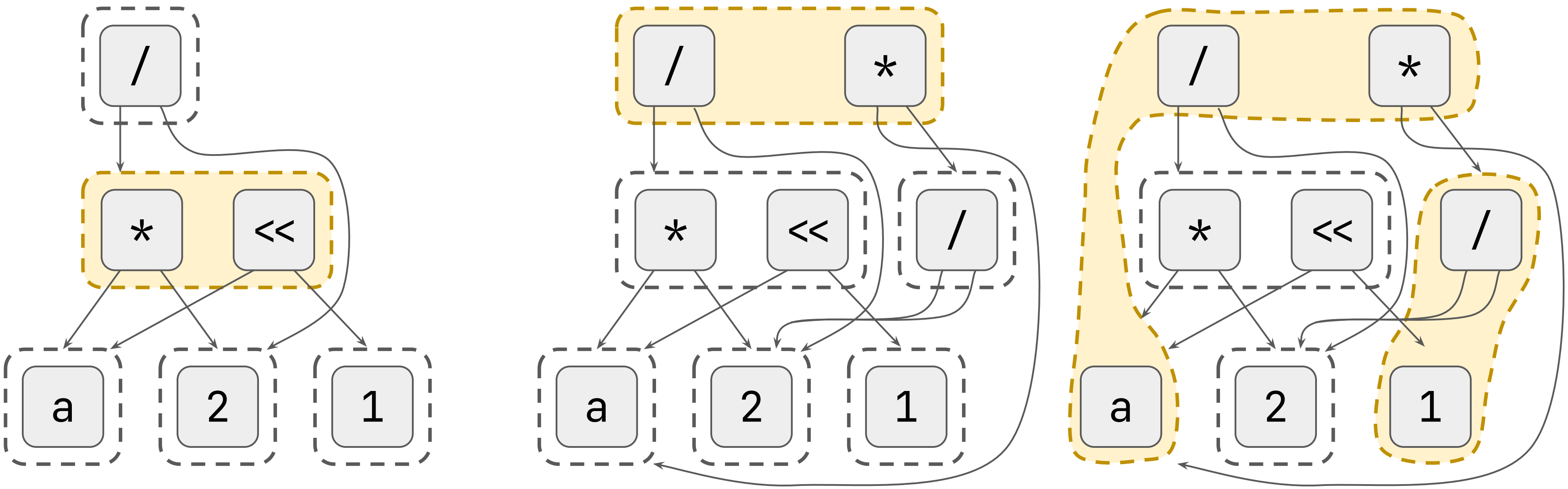
Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



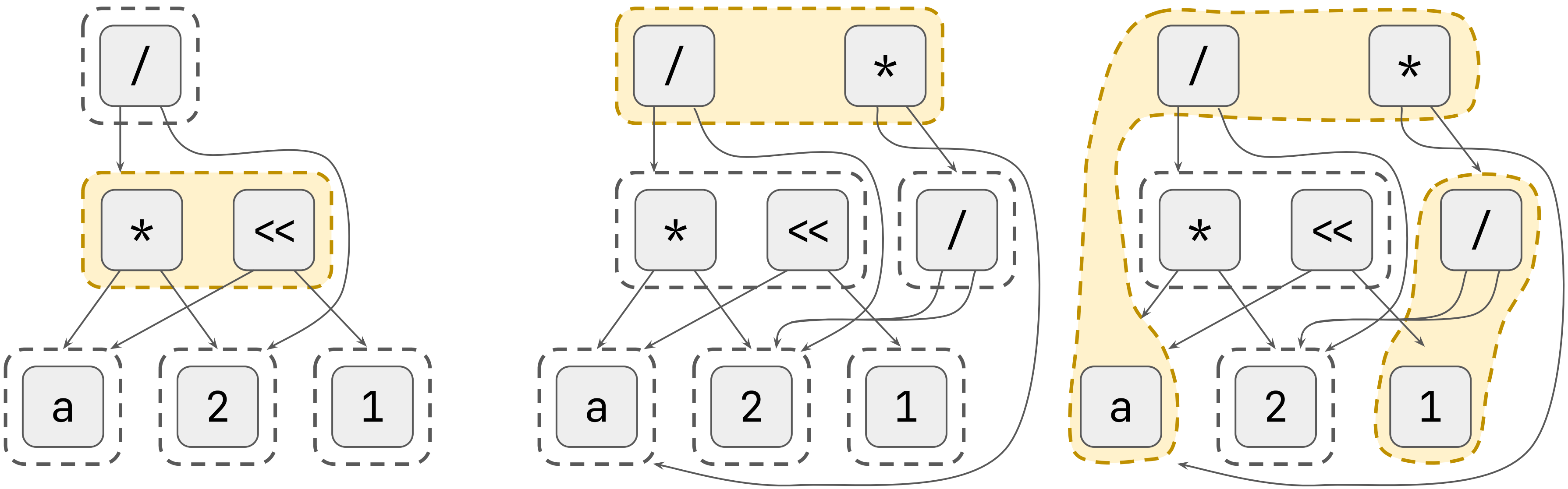
x * 2 ⟹ x << 1

(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1
x * 1 ⟹ x

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



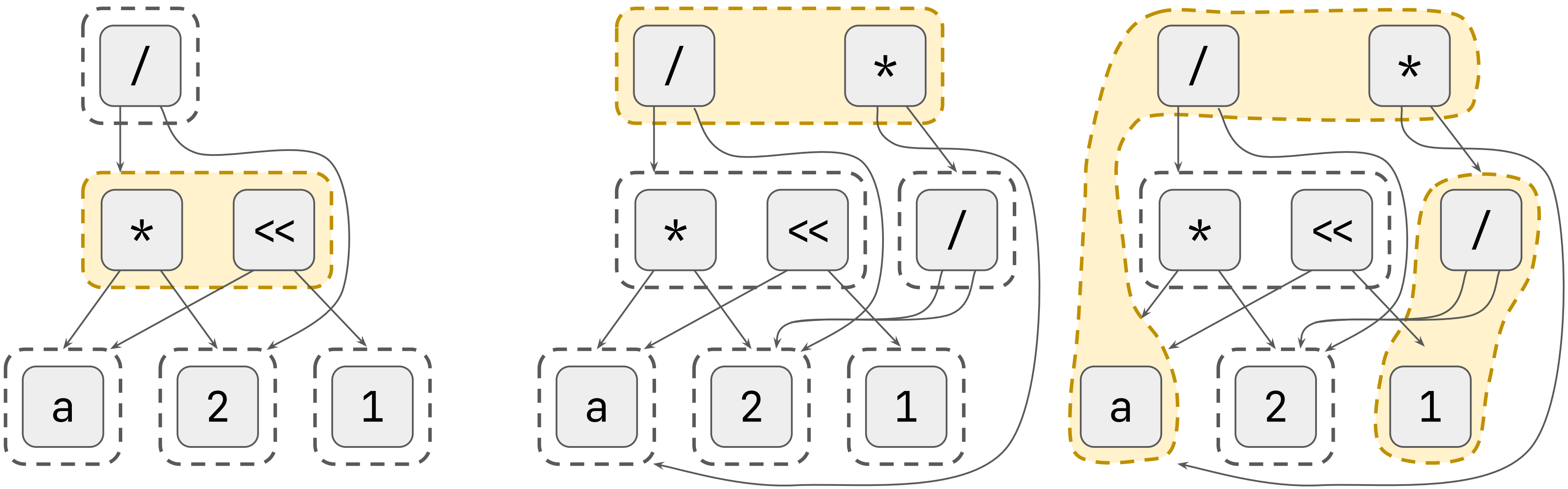
x * 2 ⟹ x << 1

(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1
x * 1 ⟹ x

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation



x * 2 ⟹ x << 1

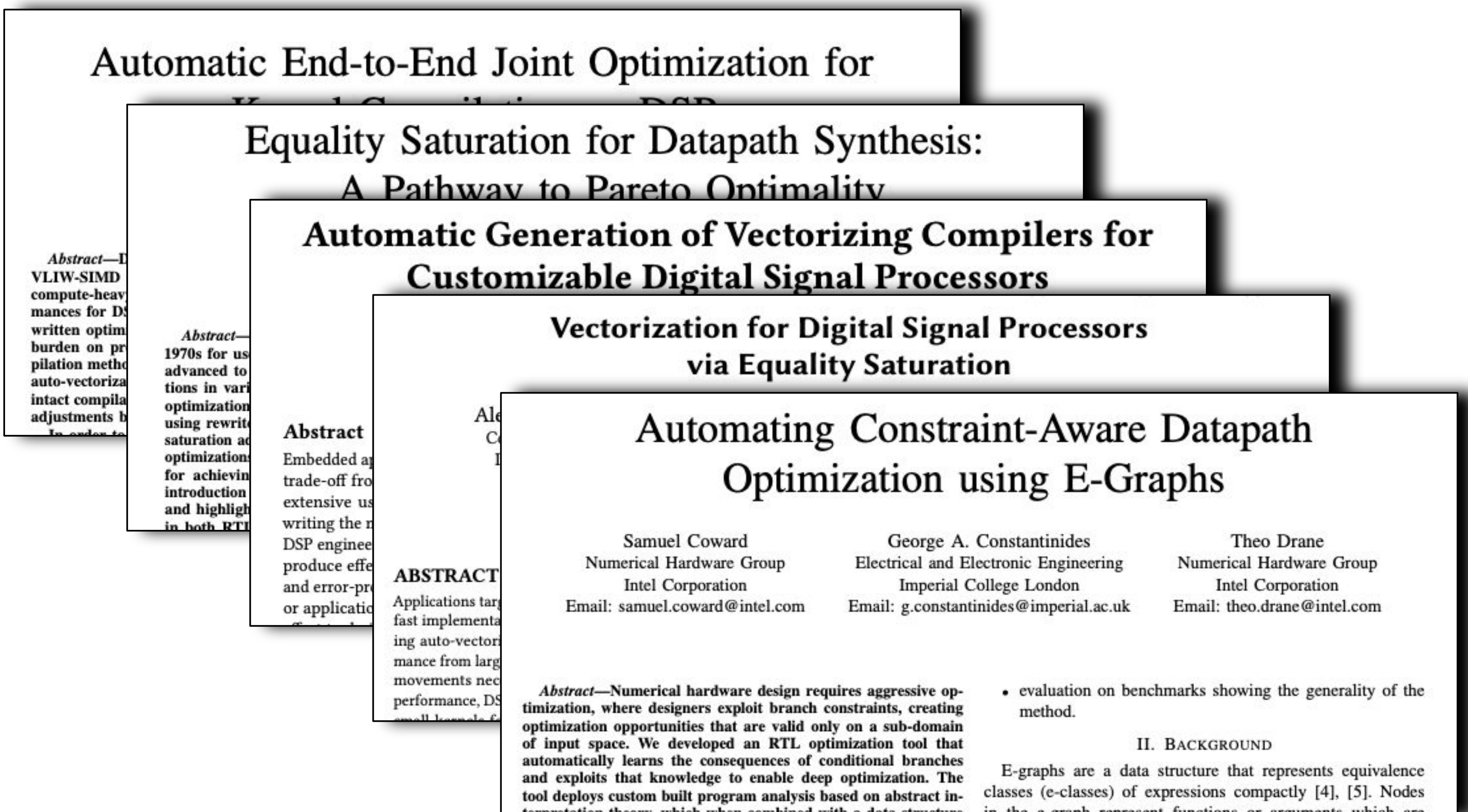
(x * y)/z ⟹ x * (y/z)

x / x ⟹ 1
x * 1 ⟹ x

Keep going until saturation or timeout

Nelson 1980; Tate et al. 2009; Willsey et al. 2021

Equality Saturation is everywhere!

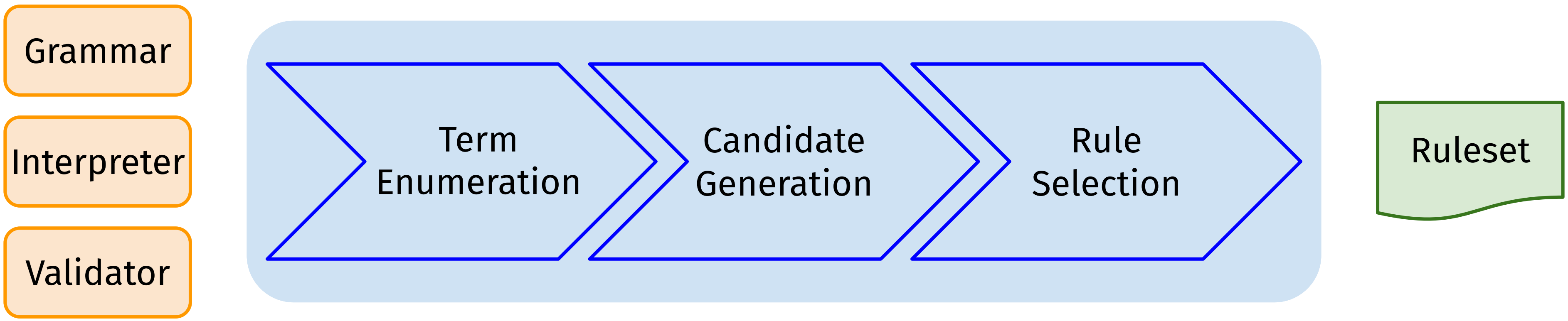


Coward et al. 2023; VanHattum et al. 2021; Thomas and Bornholt 2024; Ustan et al. 2023; Zhao et al. 2023

Equality Saturation is only as powerful as the rules used

Writing rewrite rules manually is hard

Automated Theory Exploration



Theory Exploration Inputs

Grammar

```
EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

Interpreter

```
def eval(expr):
  match expr
    |Num(n)   => n
    |Var(v)   => lookup(v)
    |Add(e1, e2) =>
    eval(e1) + eval(e2)
```

Validator

```
def is_valid(lhs, rhs):
  l = lhs.to_z3()
  r = rhs.to_z3()
  z3.assert(l.eq(r).not())
  return
    z3.check() == Unsat
```

What terms in the language **look like**

Num(5) Add(Num(1), Num(2)) Add(Add(Var(“x”), Num(1)), Num(2))

Theory Exploration Inputs

Grammar

```
EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

Interpreter

```
def eval(expr):
  match expr
    |Num(n)  => n
    |Var(v)  => lookup(v)
    |Add(e1, e2) =>
    eval(e1) + eval(e2)
```

Validator

```
def is_valid(lhs, rhs):
  l = lhs.to_z3()
  r = rhs.to_z3()
  z3.assert(l.eq(r).not())
  return
    z3.check() == Unsat
```

What terms in the language **mean**

eval(Num(5)) = 5 eval(Add(Num(1), Num(2))) = 3 eval(Add(Num(1), Add(Num(2), Num(3))) = 6

Theory Exploration Inputs

Grammar

```
EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

Interpreter

```
def eval(expr):
  match expr
    |Num(n)   => n
    |Var(v)   => lookup(v)
    |Add(e1, e2) =>
    eval(e1) + eval(e2)
```

Validator

```
def is_valid(lhs, rhs):
  l = lhs.to_z3()
  r = rhs.to_z3()
  z3.assert(l.eq(r).not())
  return
    z3.check() == Unsat
```

Whether a candidate rewrite rule is **correct**

Add(x, y) ⇒ Add(y, x) ✅   Add(x, Num(1)) ⇒ x ❌   Add(x, Add(y, z)) ⇒ Add(Add(x, y), z) ✅

Term Enumeration

```
Grammar

EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

Term Enumeration

```
Grammar

EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

a

b

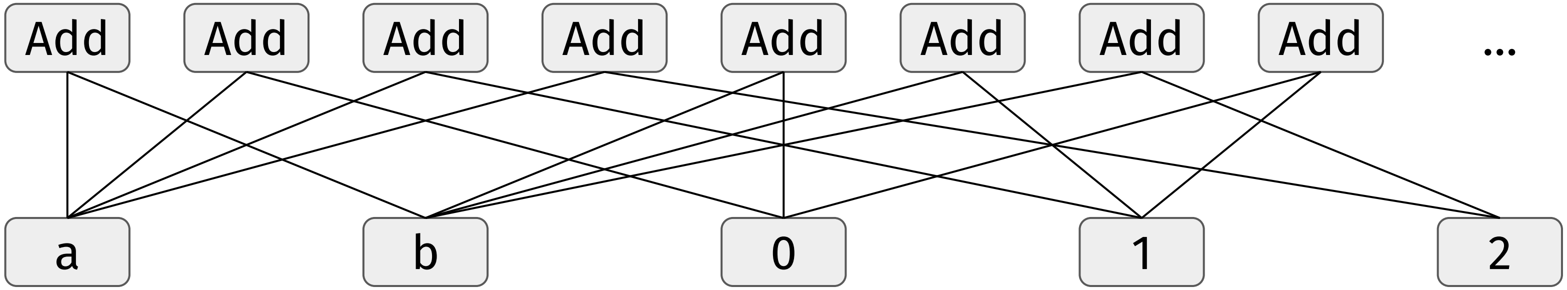
0

1

2

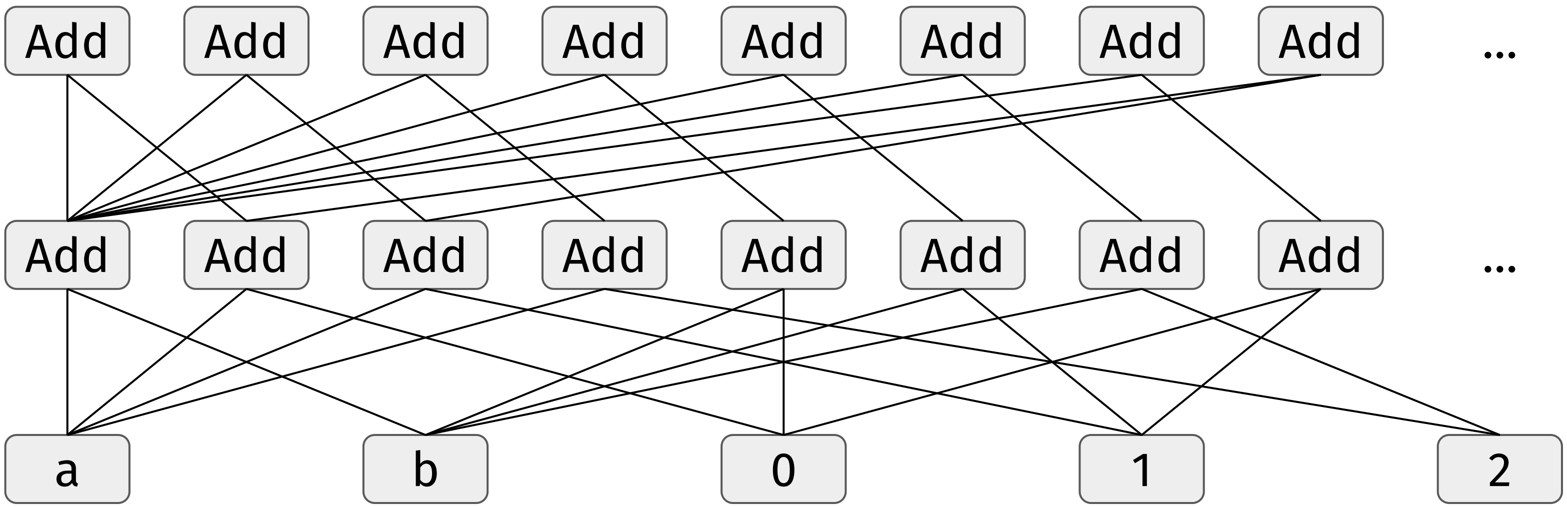
Term Enumeration

```
Grammar

EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```



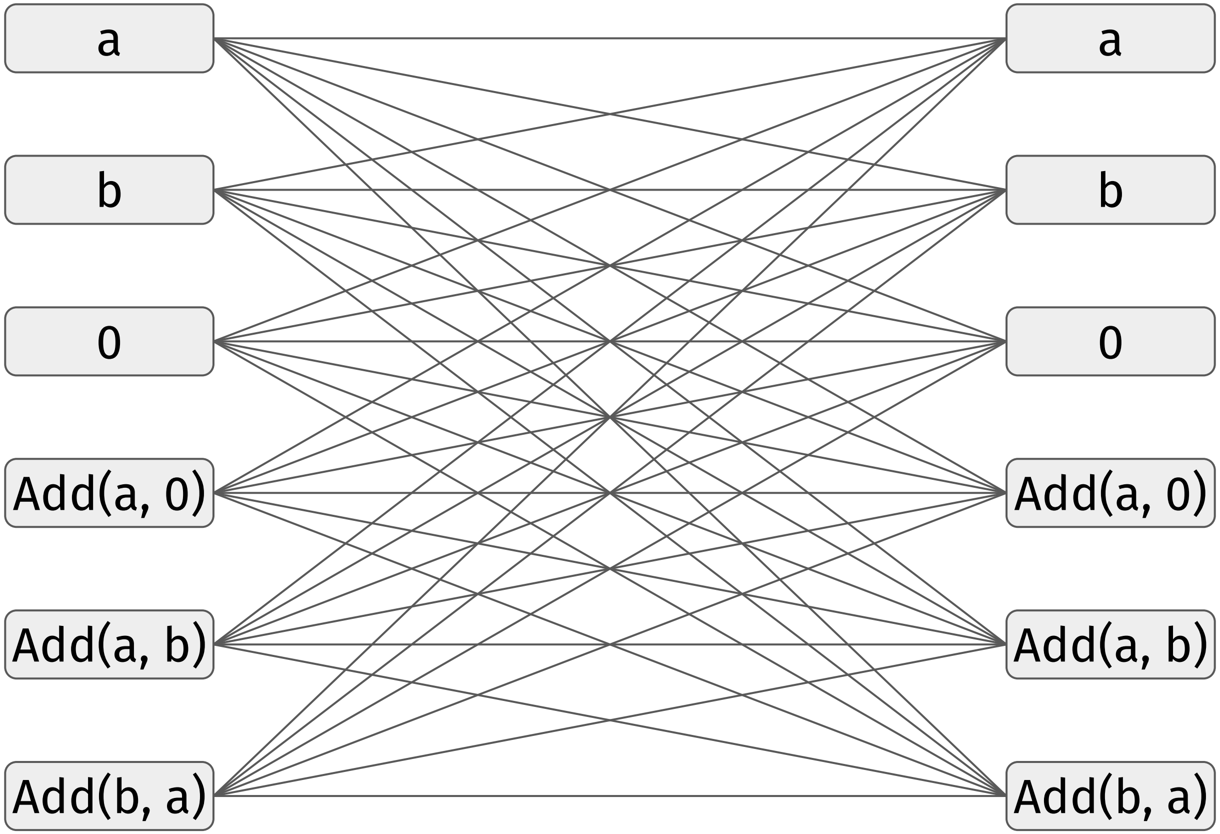
Term Enumeration

```
Grammar

EXPR :=
| Num(n)
| Var(v)
| Add(EXPR, EXPR)
```

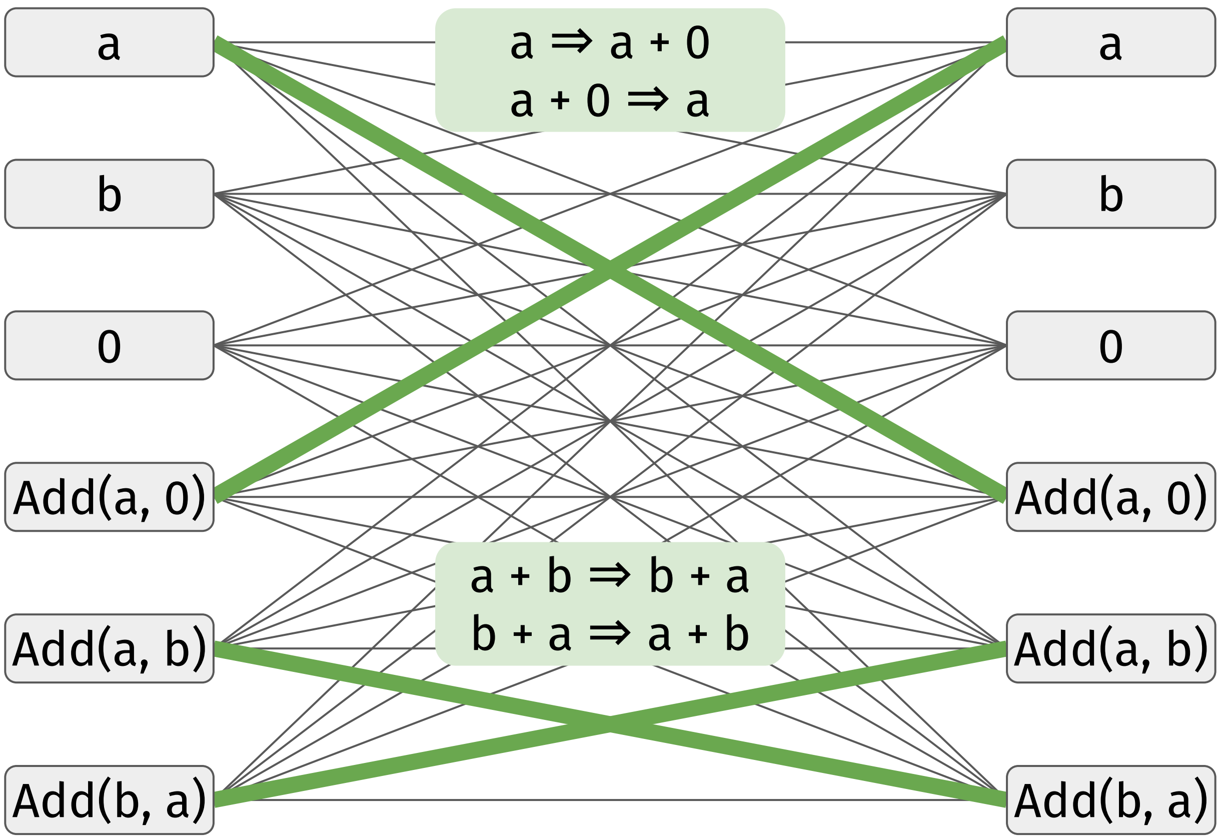


Candidate Generation

Candidate Generation



Candidate Generation



Candidate Generation

```
Interpreter

def eval(expr):
  match expr
  | ...
```

Tag each e-class with an array of possible values

a	b	0
1	3	0
-2	5	0
6	-7	0
4	0	0

Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

```
Interpreter

def eval(expr):
  match expr
  | ...
```

Tag each e-class with an array of possible values

a	b	0
1	3	0
-2	5	0
6	-7	0
4	0	0

Sample values from the domain for variables

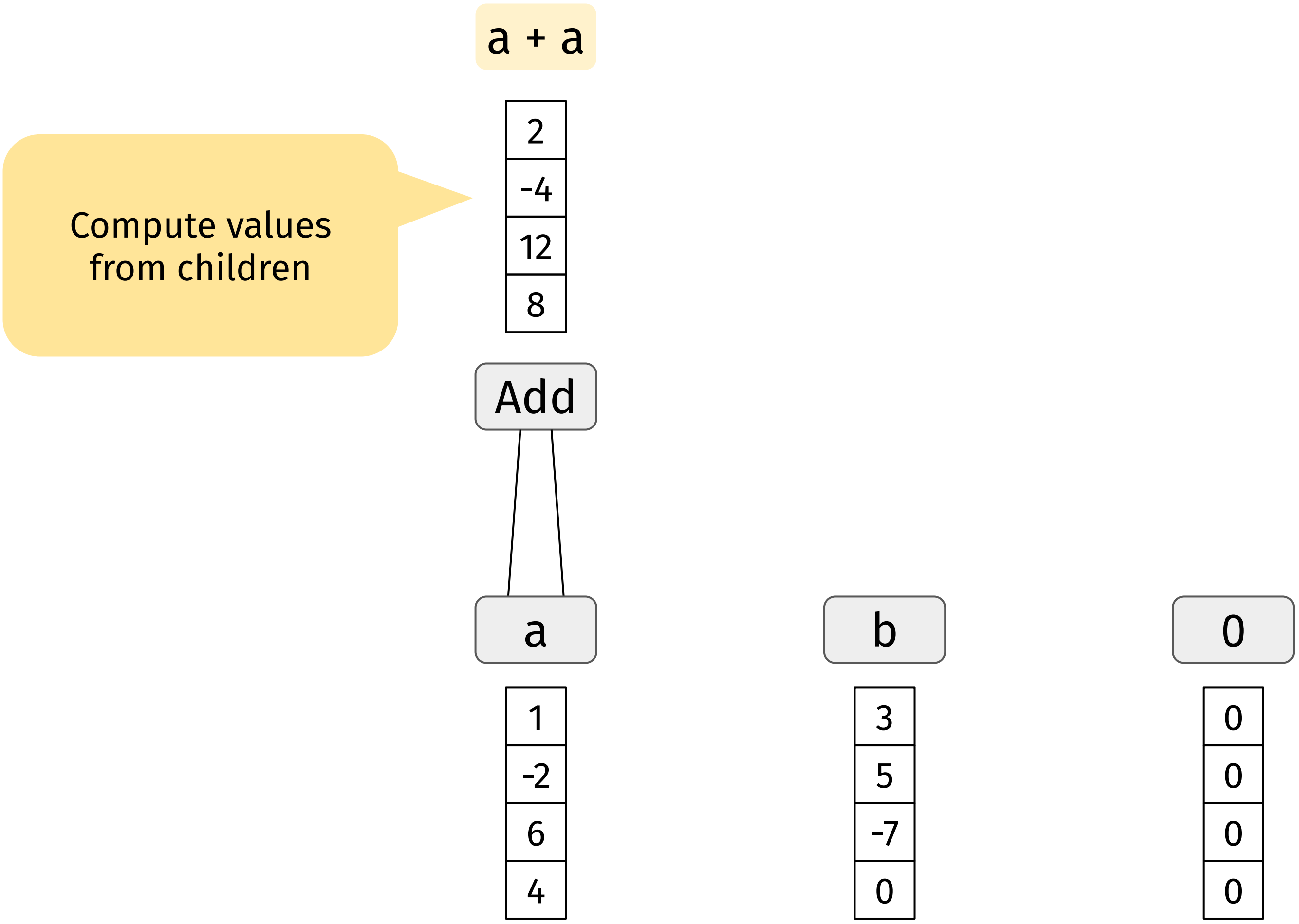
Constants always have the same value

Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

Interpreter

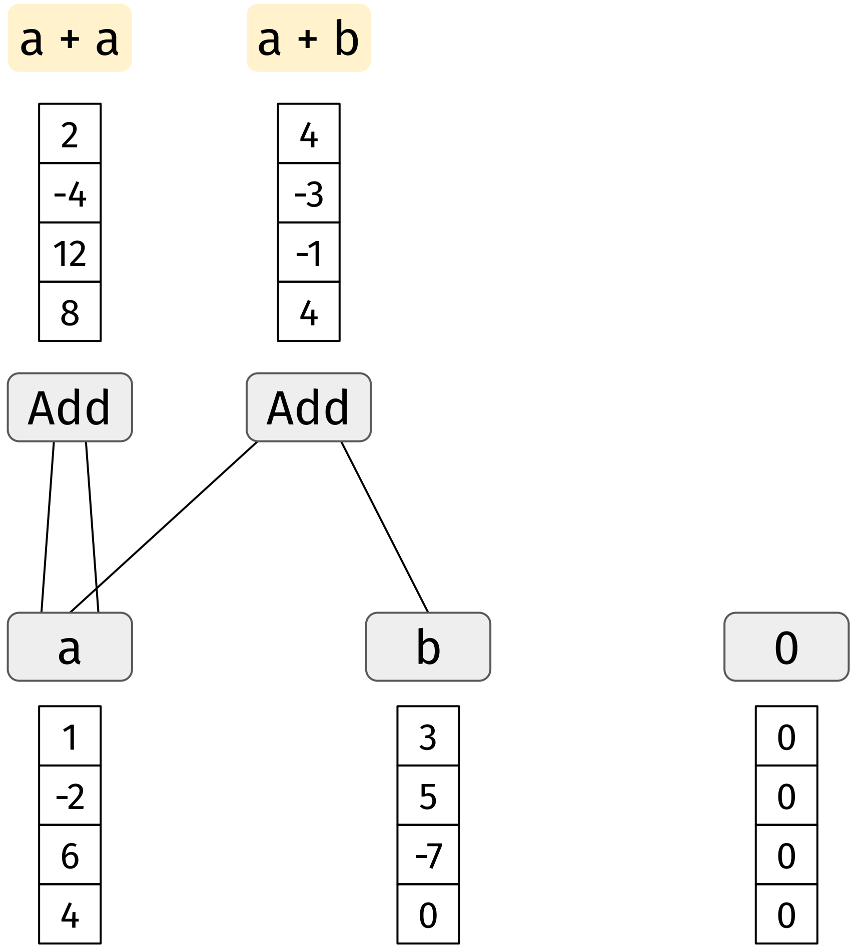
```
def eval(expr):
  match expr
  | ...
```



Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

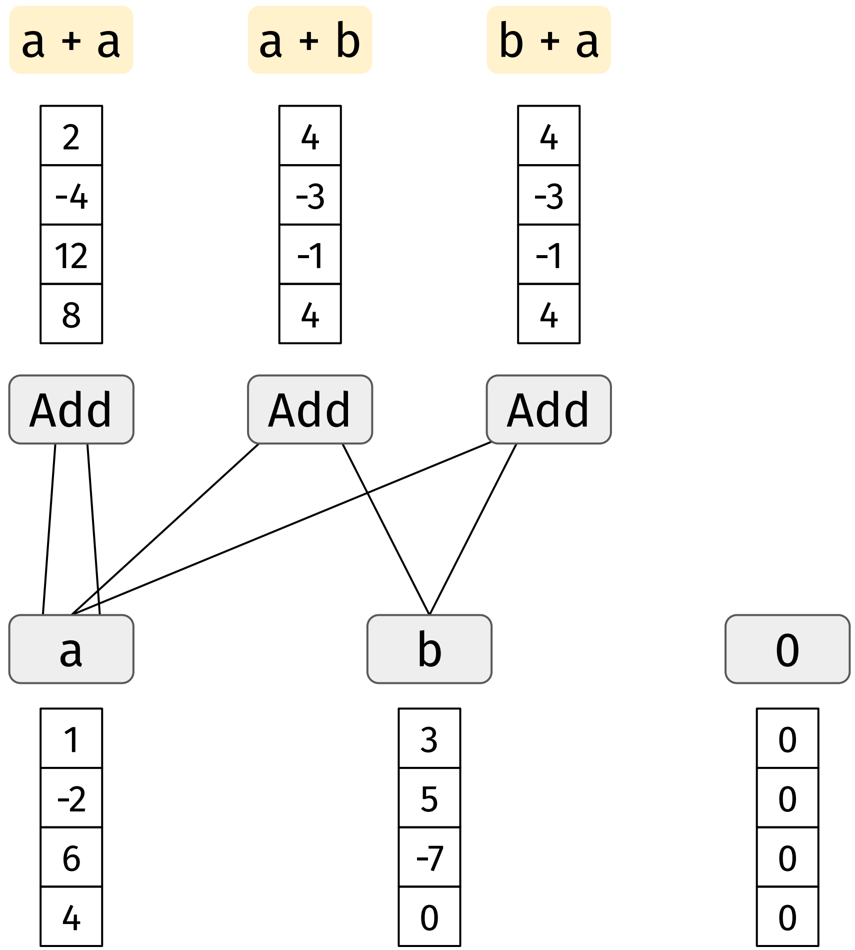
```
Interpreter

def eval(expr):
  match expr
  | ...
```



Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

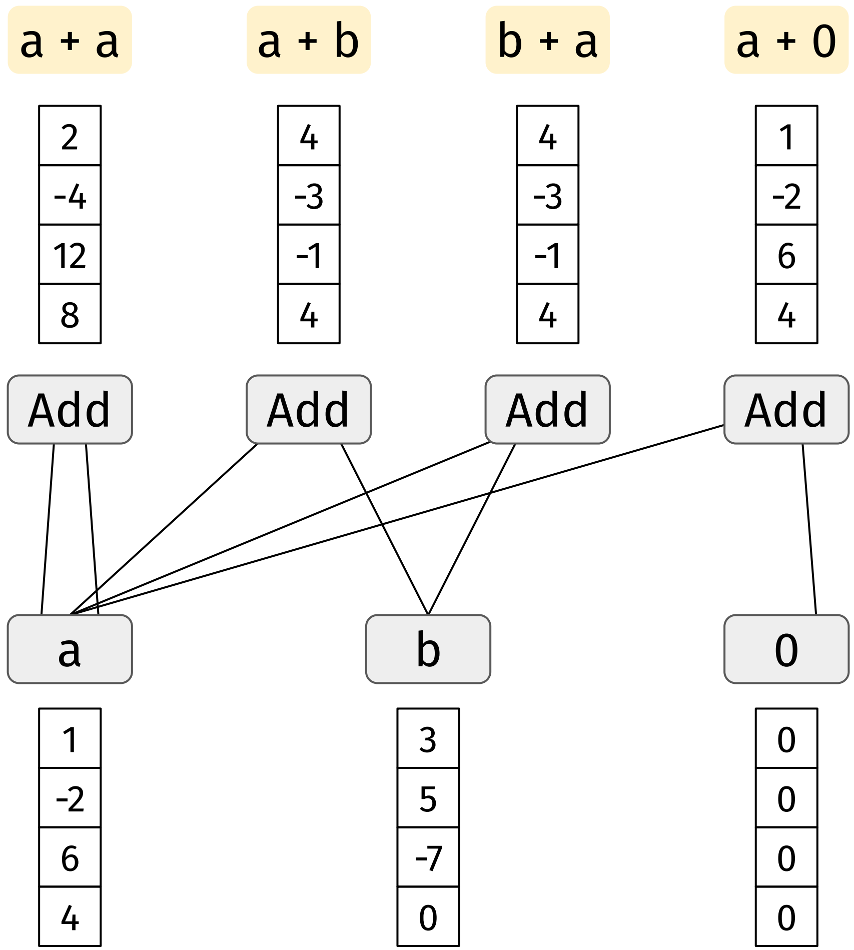
```
Interpreter

def eval(expr):
  match expr
  | ...
```



Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

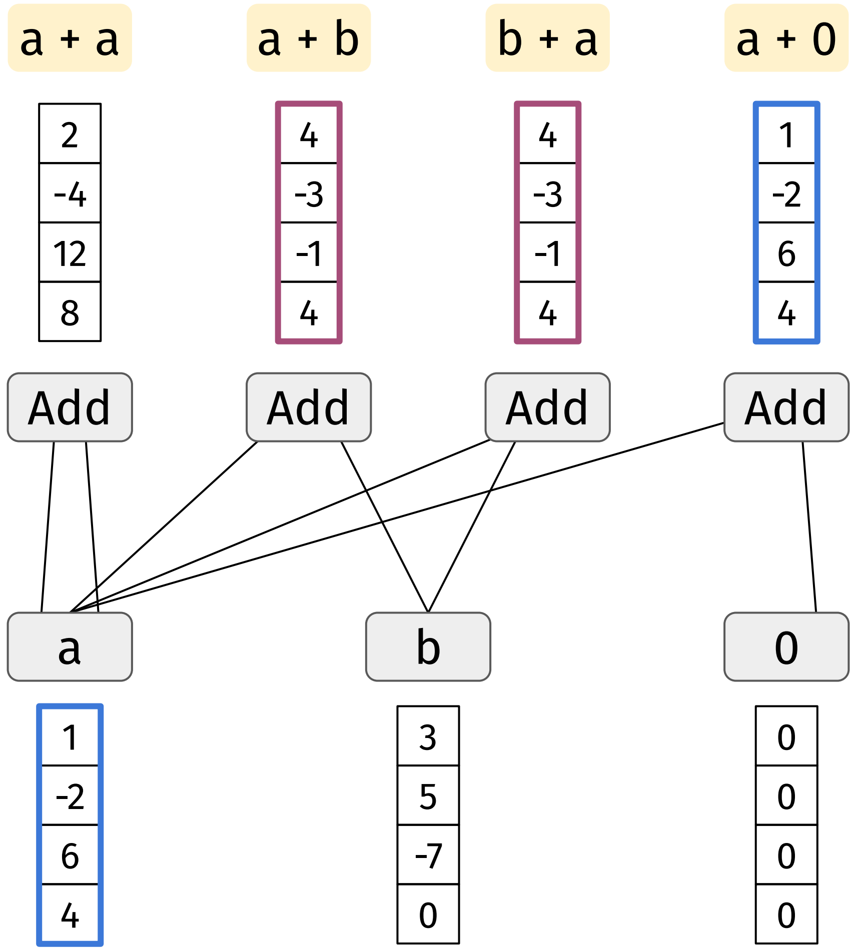
```
Interpreter

def eval(expr):
  match expr
  | ...
```



Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

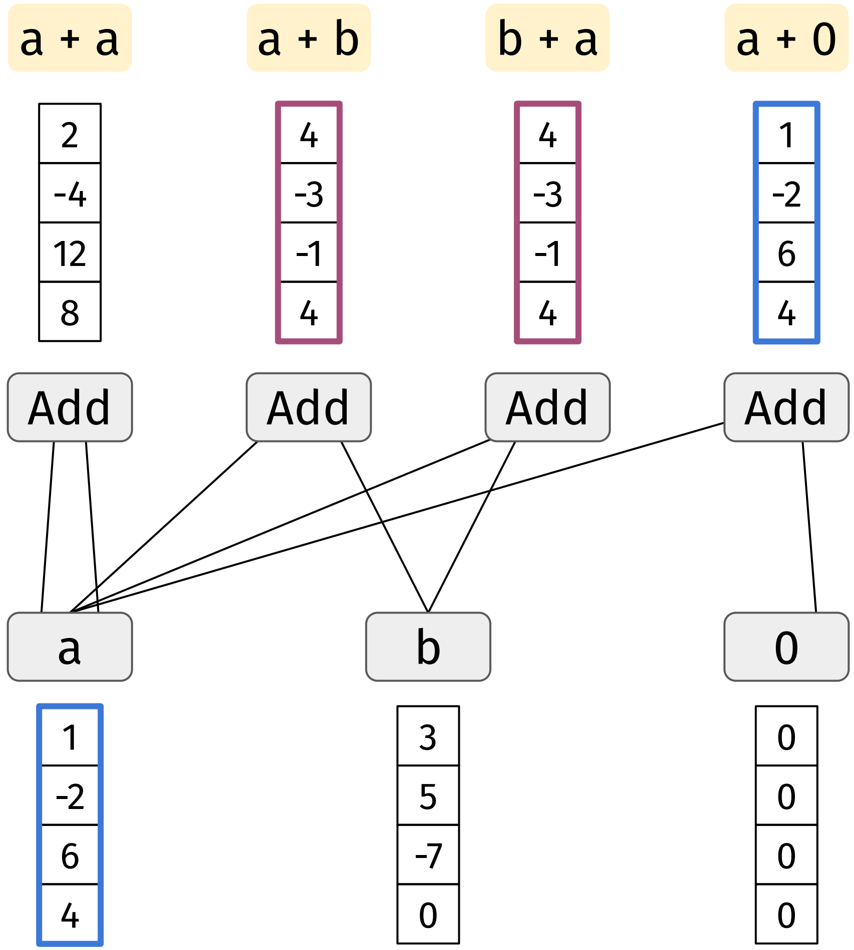
```
Interpreter

def eval(expr):
  match expr
  | ...
```



Bansal and Aiken 2006; Nandi et al. 2021

Candidate Generation

```
Interpreter

def eval(expr):
  match expr
  | ...
```

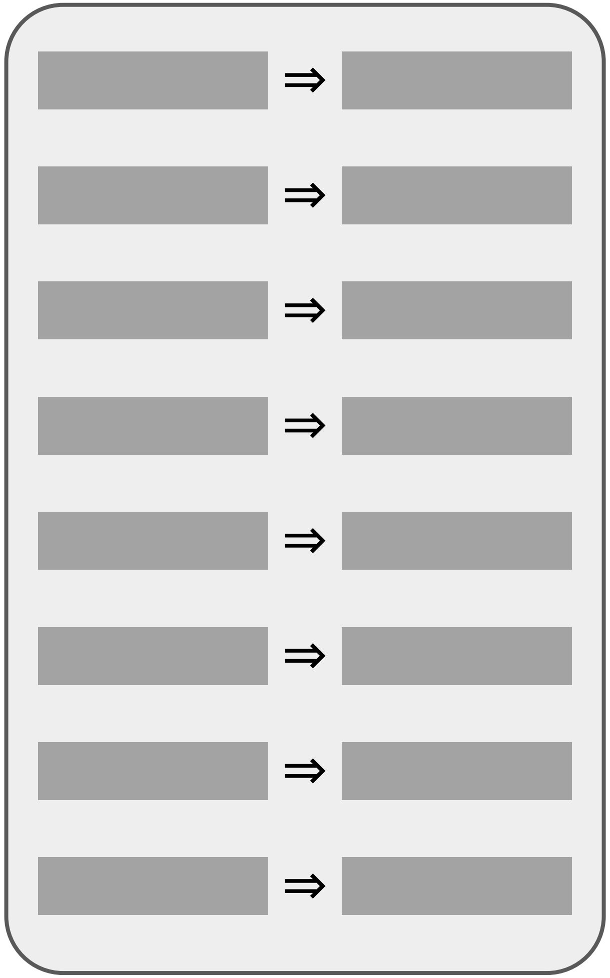


a + b ⇒ b + a
b + a ⇒ a + b

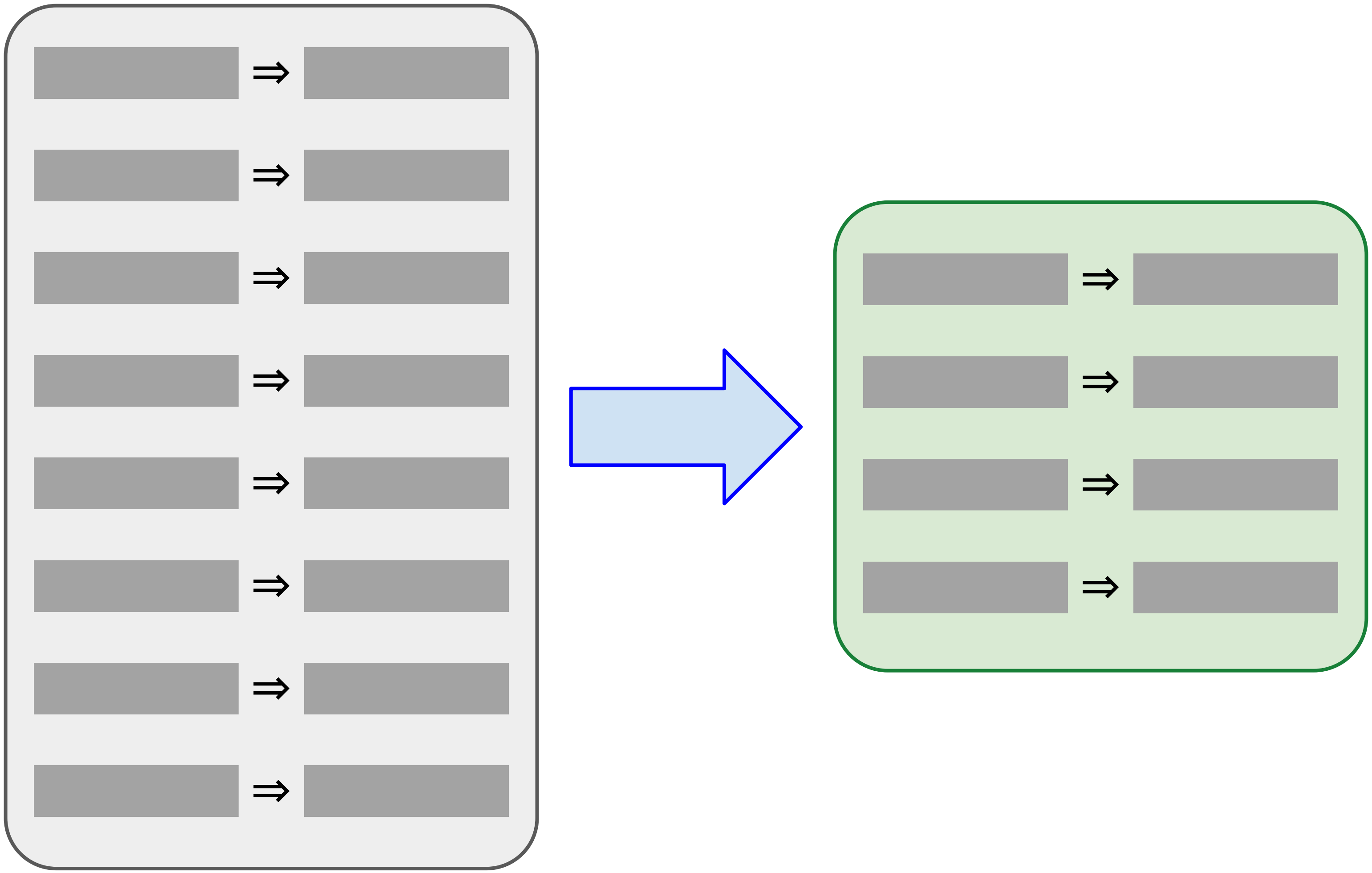
a + 0 ⇒ a
a ⇒ a + 0

Bansal and Aiken 2006; Nandi et al. 2021

Rule Selection

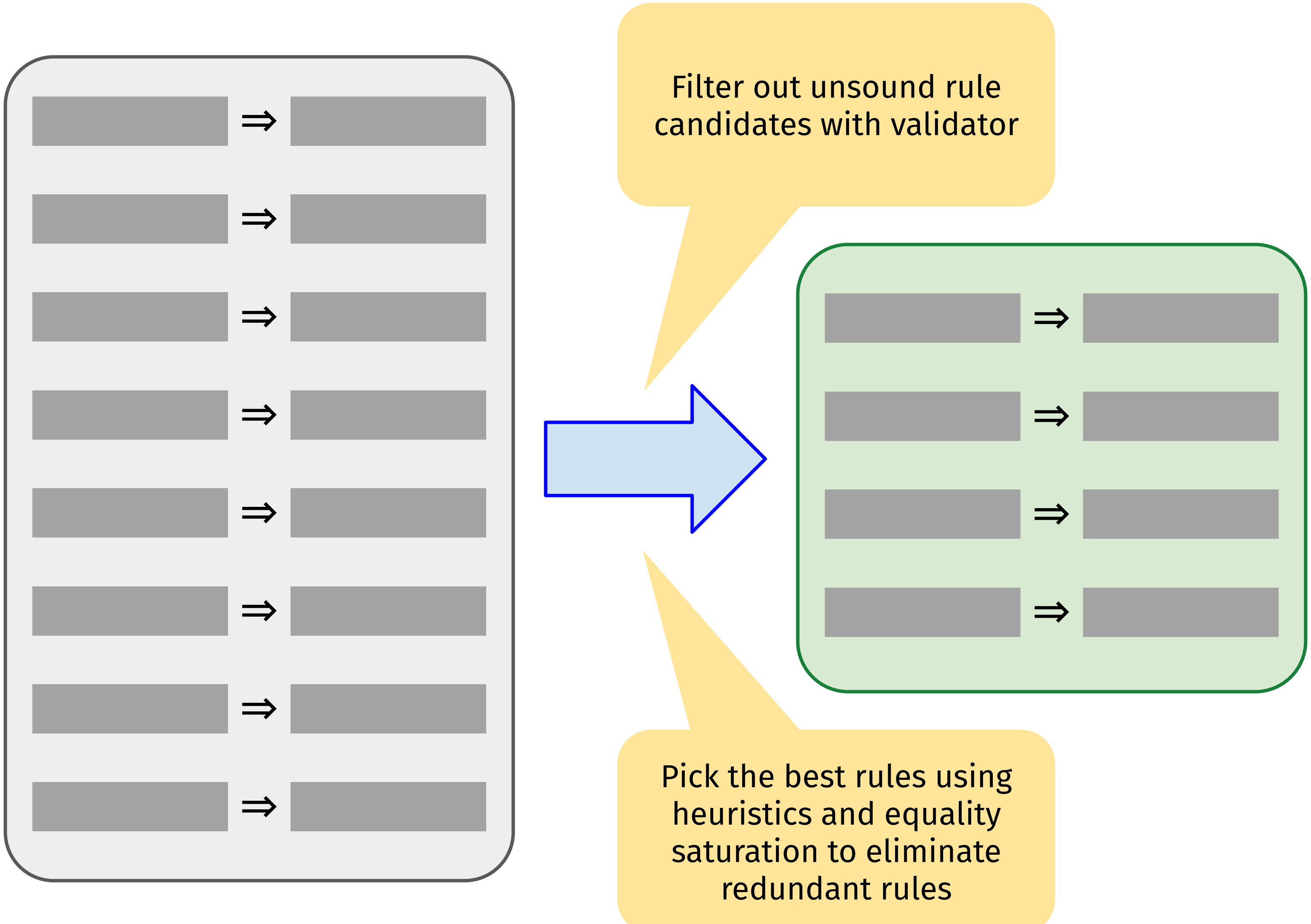


Rule Selection



Rule Selection

```
Validator

def is_valid(l, r):
  ...
```



Rule Selection

Chosen Rules:

Candidates:

```
x + y ⇒ y + x
x * y ⇒ y * x
x + 0 ⇒ 0 + x
y + 0 ⇒ 0 + y
x * 1 ⇒ 1 * x
y * 1 ⇒ 1 * y
```

Sort rule candidates using heuristics:
More general is better

Rule Selection

Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

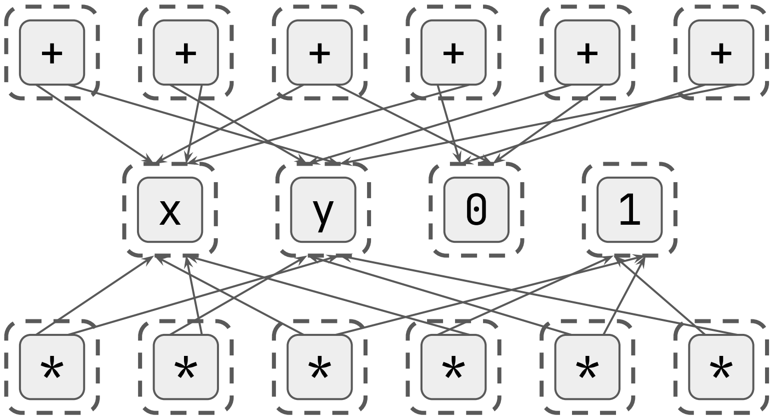
x + 0 ⇒ 0 + x

y + 0 ⇒ 0 + y

x * 1 ⇒ 1 * x

y * 1 ⇒ 1 * y

Chosen Rules:



Initialize e-graph with all candidates

Rule Selection

Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

x + 0 ⇒ 0 + x

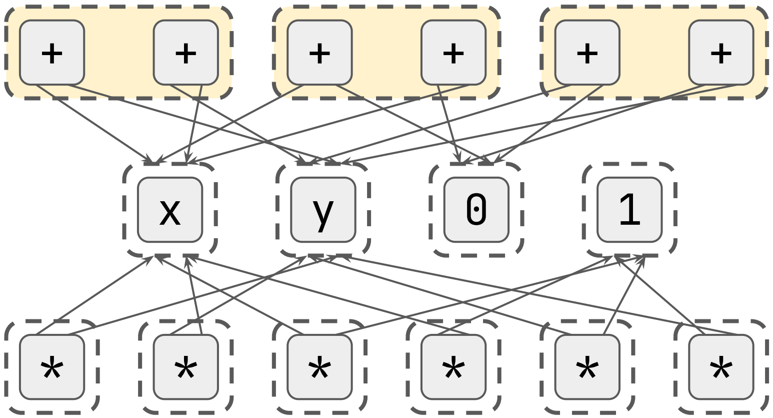
y + 0 ⇒ 0 + y

x * 1 ⇒ 1 * x

y * 1 ⇒ 1 * y

Chosen Rules:

x + y ⇒ y + x



Pick a rule; Run equality saturation

Rule Selection

Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

~~x + 0 ⇒ 0 + x~~

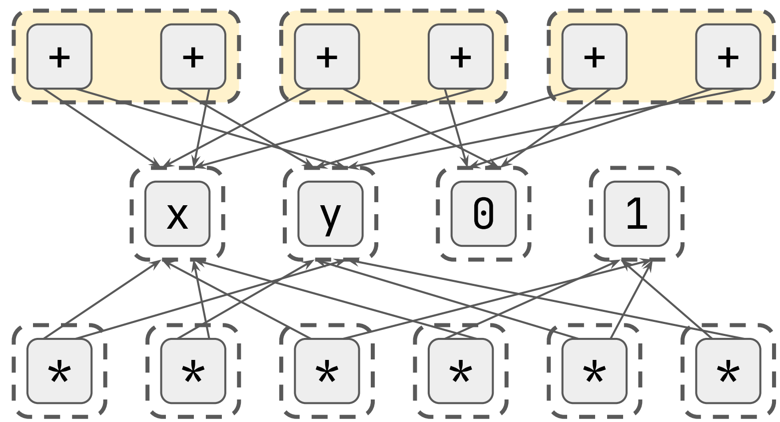
~~y + 0 ⇒ 0 + y~~

x * 1 ⇒ 1 * x

y * 1 ⇒ 1 * y

Chosen Rules:

x + y ⇒ y + x



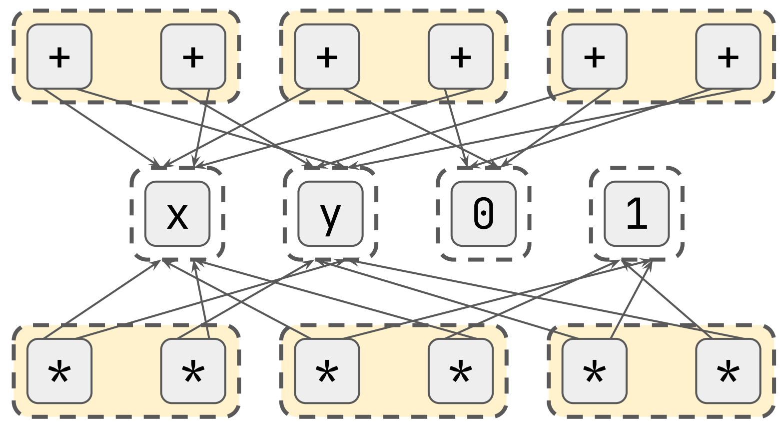
Eliminate redundant candidates

Rule Selection

Chosen Rules:

x + y ⇒ y + x

x * y ⇒ y * x



Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

~~x + 0 ⇒ 0 + x~~

~~y + 0 ⇒ 0 + y~~

x * 1 ⇒ 1 * x

y * 1 ⇒ 1 * y

Pick a rule; Run equality saturation

Rule Selection

Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

~~x + 0 ⇒ 0 + x~~

~~y + 0 ⇒ 0 + y~~

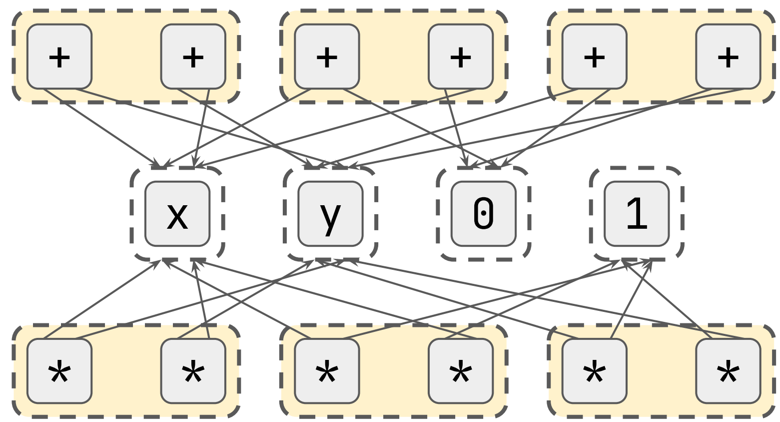
~~x * 1 ⇒ 1 * x~~

~~y * 1 ⇒ 1 * y~~

Chosen Rules:

x + y ⇒ y + x

x * y ⇒ y * x



Eliminate redundant candidates

Rule Selection

Candidates:

x + y ⇒ y + x

x * y ⇒ y * x

~~x + 0 ⇒ 0 + x~~

~~y + 0 ⇒ 0 + y~~

~~x * 1 ⇒ 1 * x~~

~~y * 1 ⇒ 1 * y~~

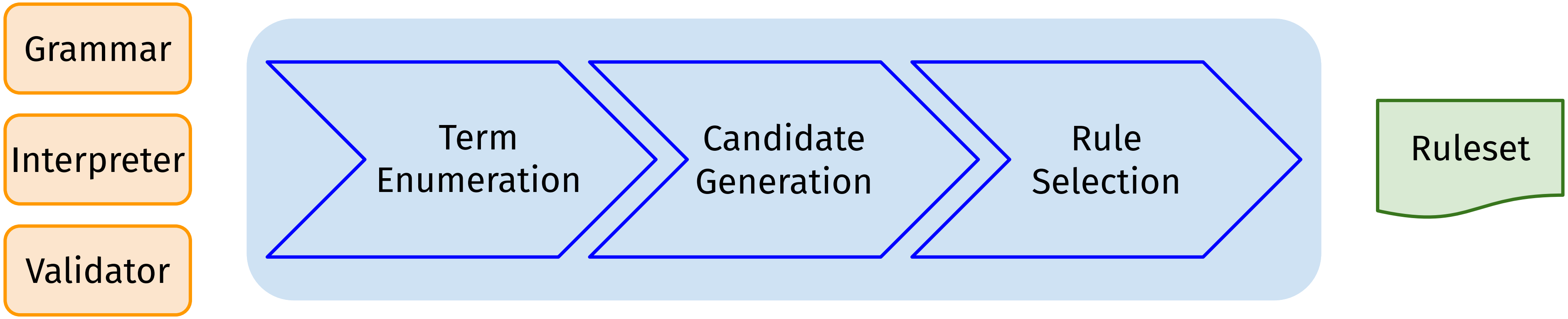
Final Ruleset:

x + y ⇒ y + x

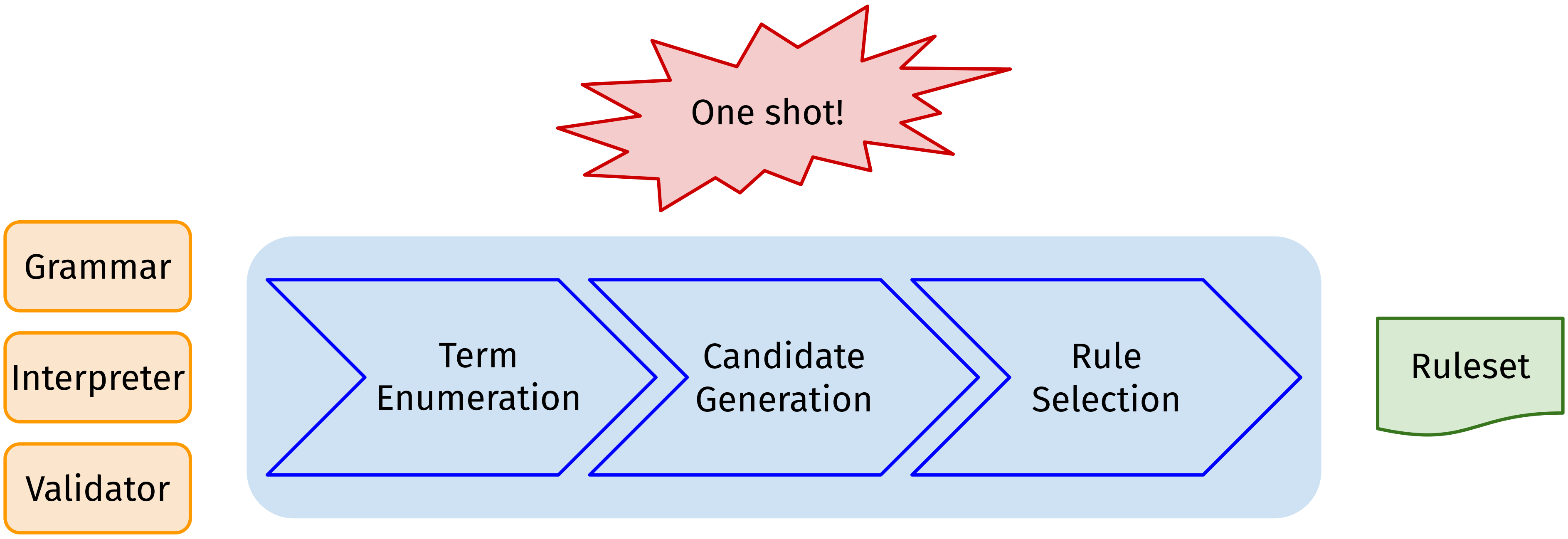
x * y ⇒ y * x

Repeat until no more candidates

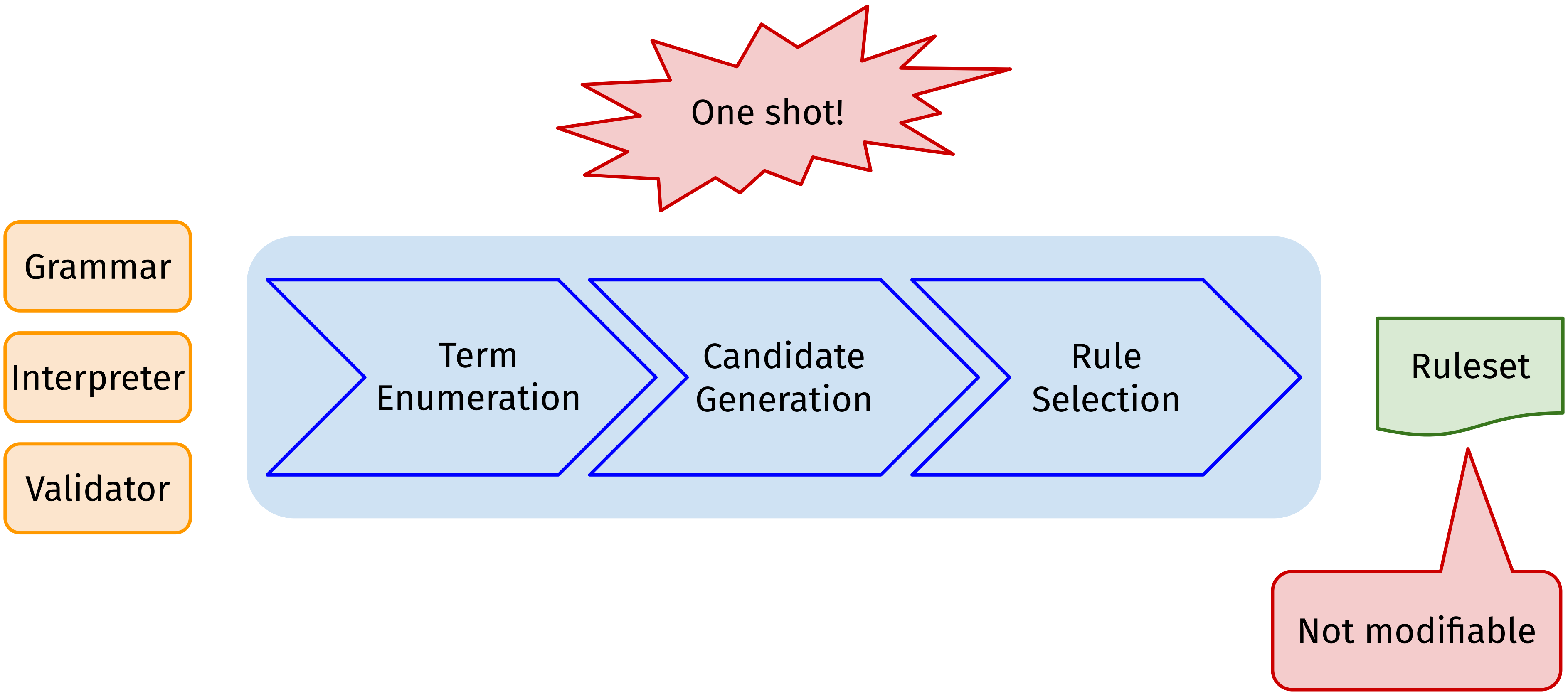
Traditional Theory Exploration



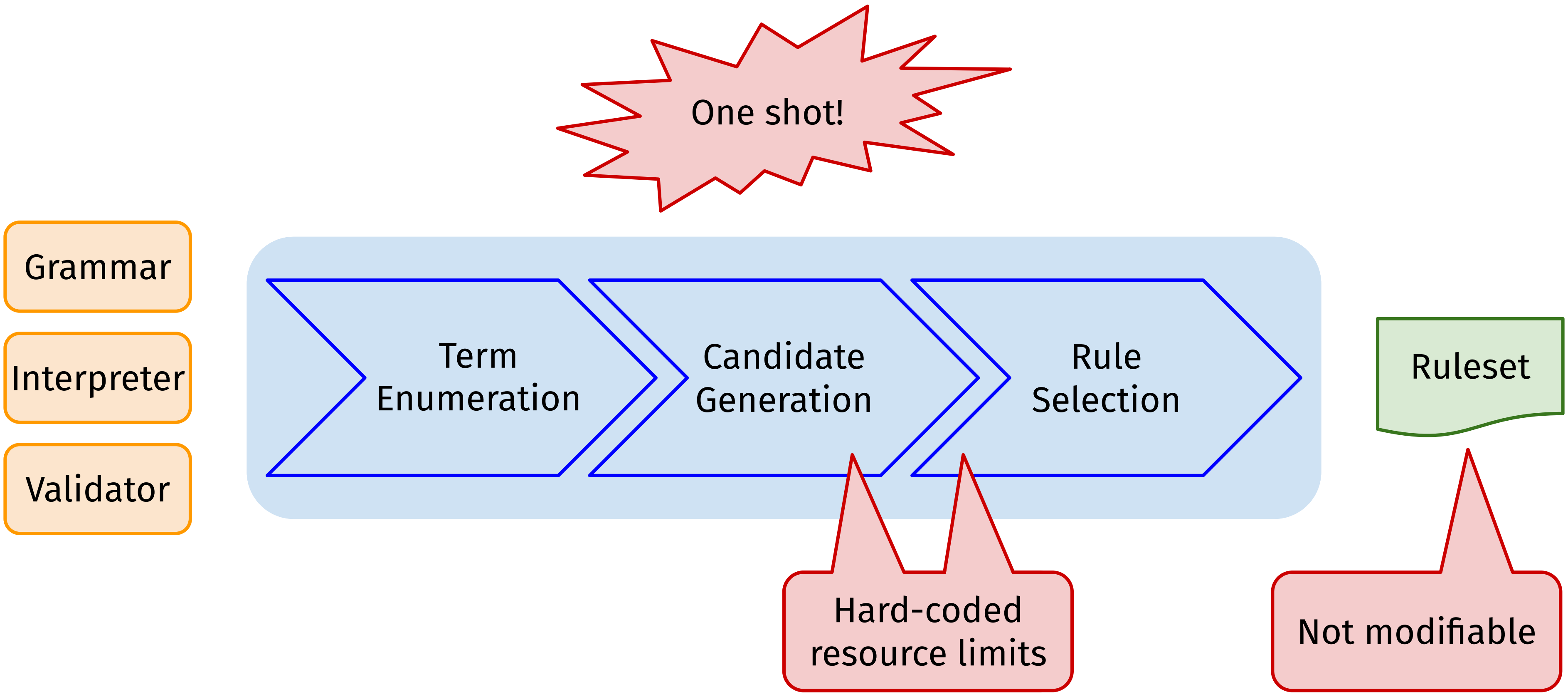
Traditional Theory Exploration



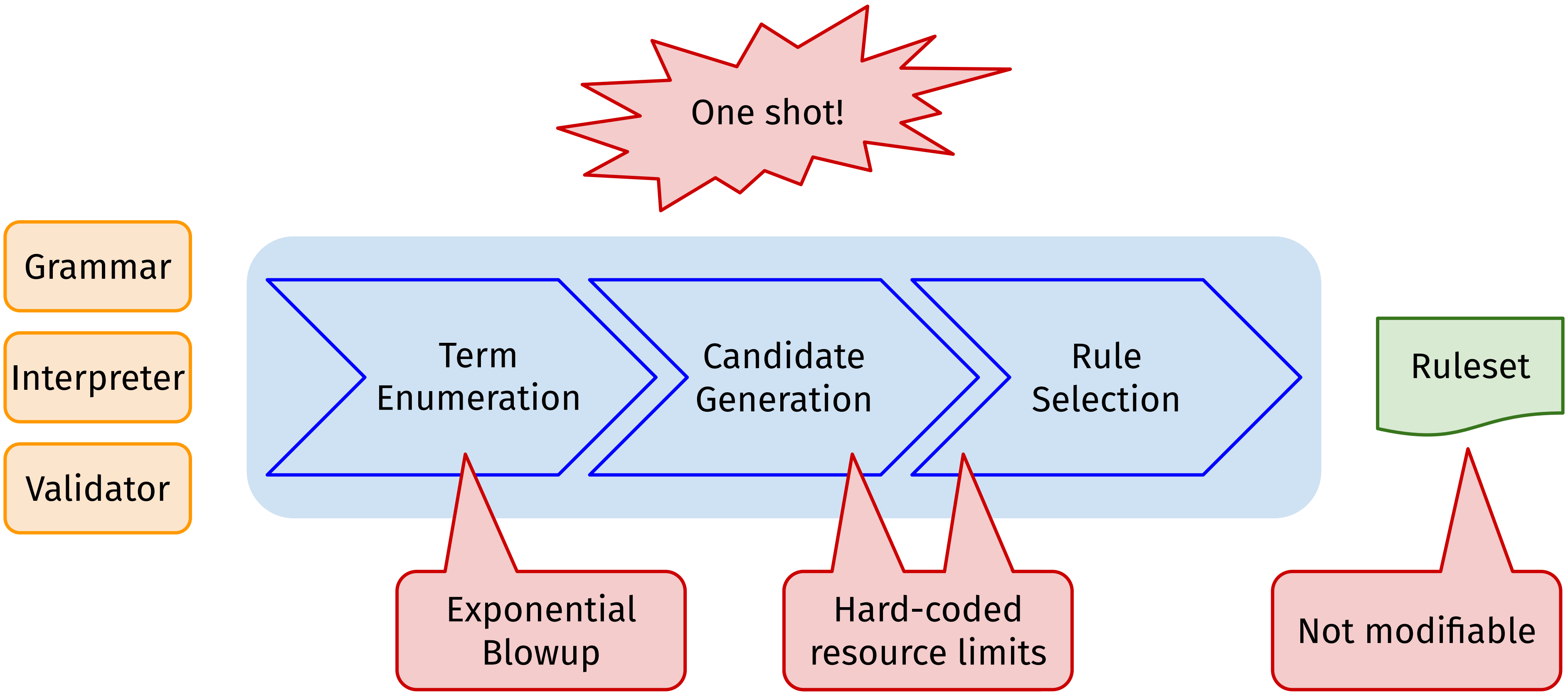
Traditional Theory Exploration



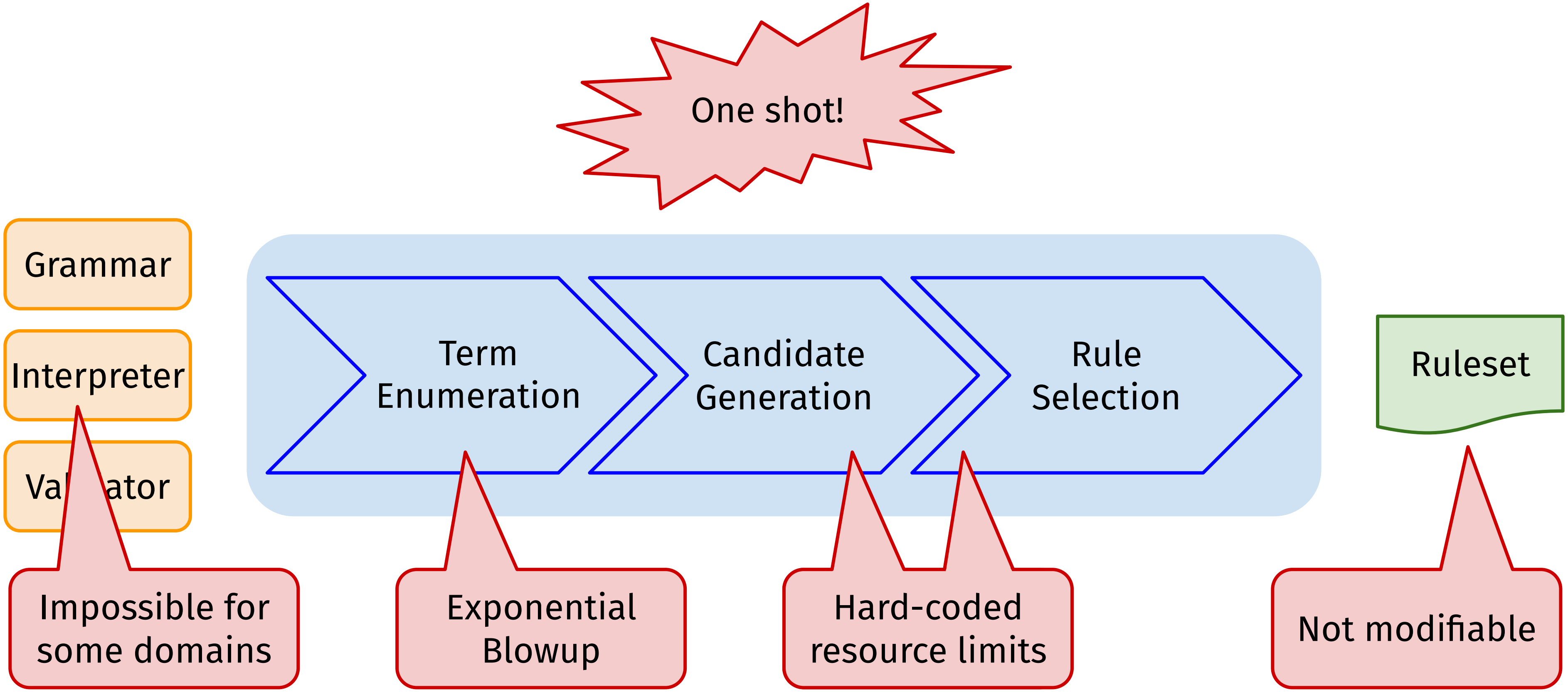
Traditional Theory Exploration



Traditional Theory Exploration



Traditional Theory Exploration



The ENUMO DSL:

A more flexible and extensible approach to rule inference

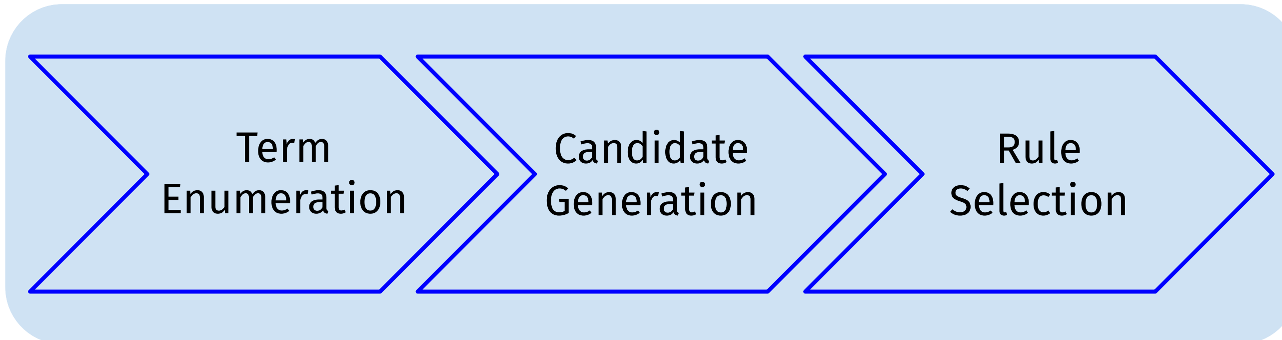
Pal et al. 2023

Insight: Users have intuition about which parts of the domain are worth exploring

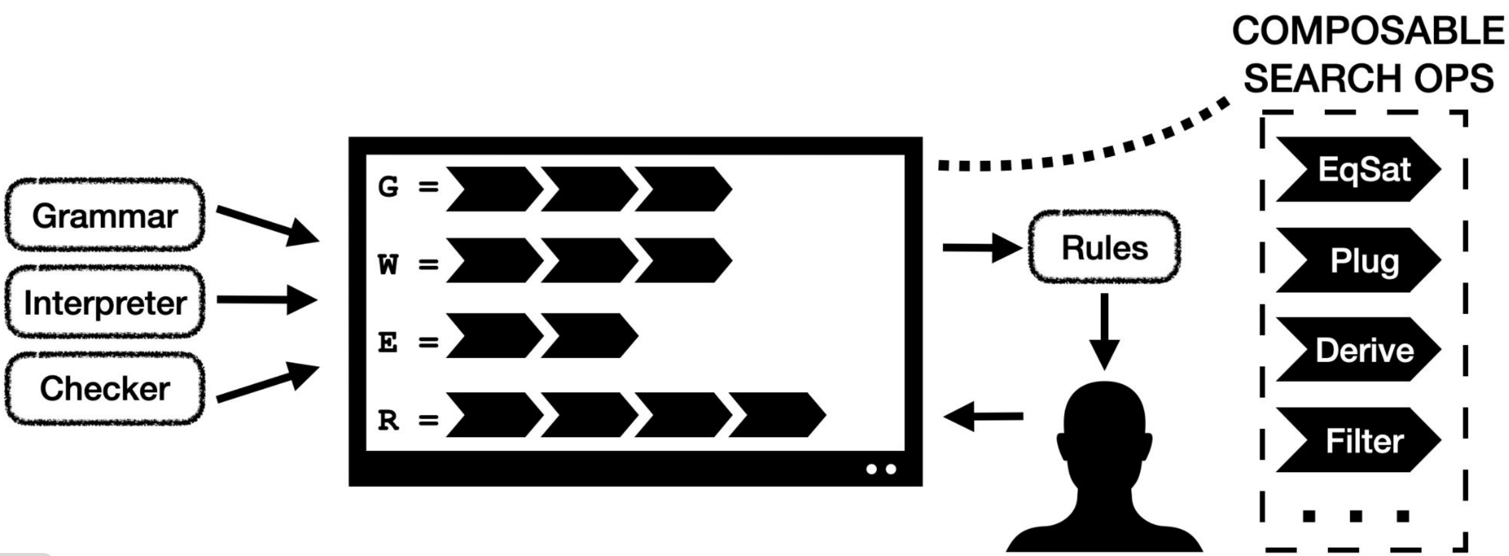
```
(~ (~ (~ (~ (~ a)))))
```

```
(- (* a a) (* b b))
```

Pal et al. 2023



We enable experts to leverage their expertise by exposing a small set of useful operators



Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
```

Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }
```

```
EXP :=
| Num(n)
| Var(v)
| Neg(EXP)
| Add(EXP, EXP)
```

Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
```

Pal et al. 2023

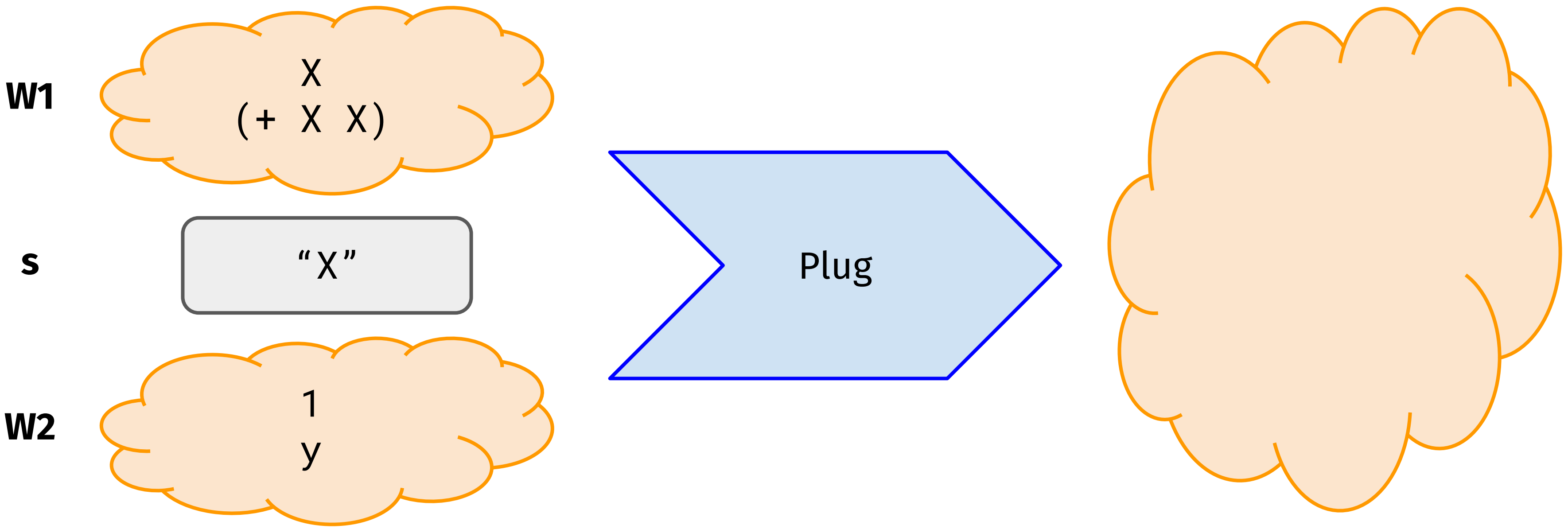
ENUMO DSL

$$\mathsf{Plug}\ \mathcal{W}_1\ s\ \mathcal{W}_2$$

All combinations of replacing s in W_1 with a term from W_2

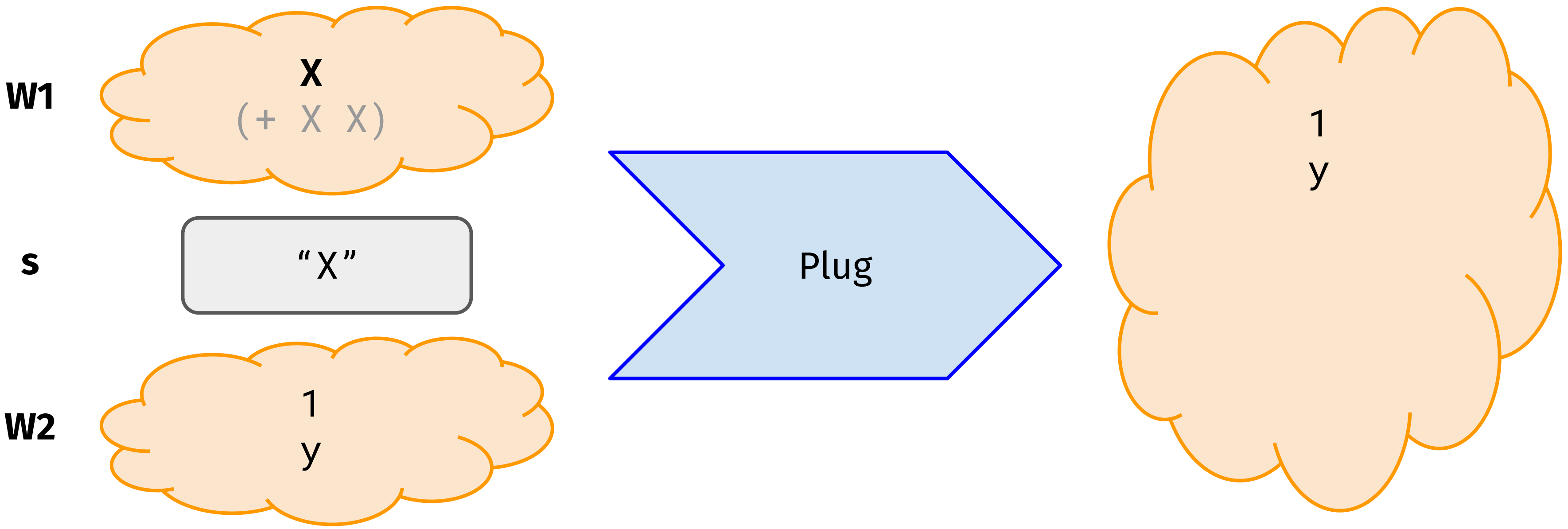
Pal et al. 2023

ENUMO DSL



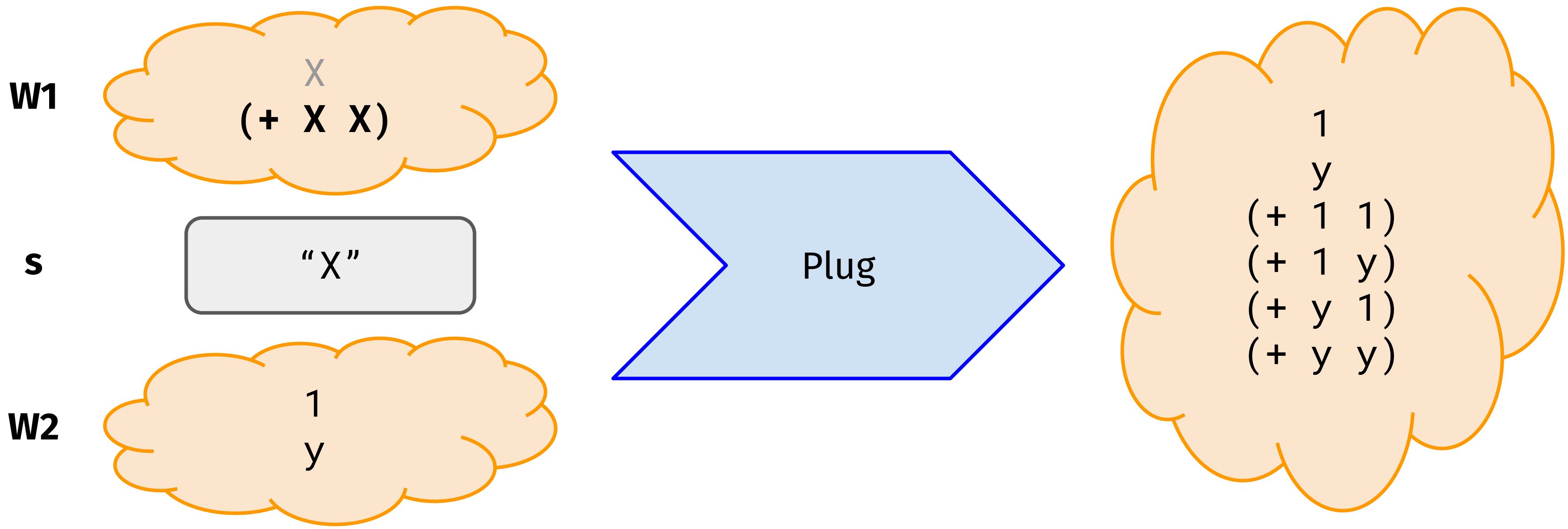
Duregård et al. 2012; Pal et al. 2023

ENUMO DSL



Duregård et al. 2012; Pal et al. 2023

ENUMO DSL



Duregård et al. 2012; Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
           .plug(“EXP”, lits)
```

Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
           .plug(“EXP”, lits)
```

```
EXP                  (+ (~ EXP) EXP)
(~ EXP)              (+ EXP (+ EXP EXP))
(+ EXP EXP)          (+ (~ EXP) (~ EXP))
(~ (~ EXP))          (+ (+ EXP EXP) EXP)
(+ EXP EXP)          (+ (~ EXP) (+ EXP EXP))
(~ (+ EXP EXP))      (+ (+ EXP EXP) (~ EXP))
(+ EXP (~ EXP))      (+ (+ EXP EXP) (+ EXP EXP))
```

Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug("EXP", exps)
           .plug("EXP", lits)
```

```
a         (~ (~ a))       (~ (+ b a))
b         (~ (~ b))       (~ (+ b b))
0         (~ (~ 0))       (~ (+ b 0))
1         (~ (~ 1))       (~ (+ b 1))
(~ a)     (~ (+ a a))     (~ (+ 0 a))
(~ b)     (~ (+ a b))     (~ (+ 0 b))
(~ 0)     (~ (+ a 0))     ...
(~ 1)     (~ (+ a 1))     (+ (+ 1 1) (+ 1 1))
```

Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
           .plug(“EXP”, lits)

rules =
  wkld
    .to_egraph()
    .find_candidates()
    .select_rules(limits)
```

Pal et al. 2023

ENUMO DSL

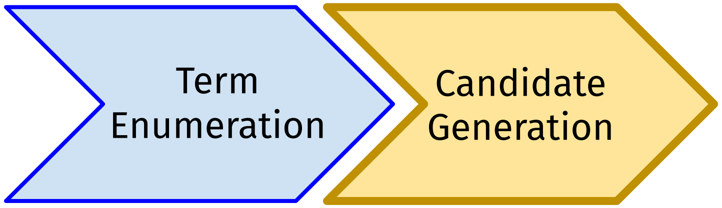
```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
           .plug(“EXP”, lits)

rules =
  wkld
    .to_egraph()
    .find_candidates()
    .select_rules(limits)
```

Term
Enumeration

Nandi et al. 2021; Pal et al. 2023

ENUMO DSL

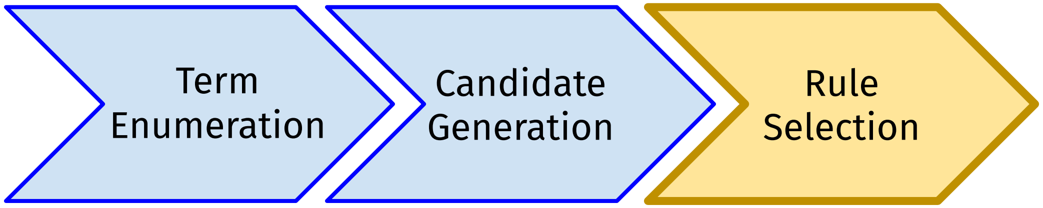
```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug(“EXP”, exps)
           .plug(“EXP”, lits)

rules =
  wkld
    .to_egraph()
    .find_candidates()
    .select_rules(limits)
```



Nandi et al. 2021; Pal et al. 2023

ENUMO DSL

```
lits = Workload { a b 0 1 }
exps = Workload { EXP (~ EXP) (+ EXP EXP) }

wkld = exps.plug("EXP", exps)
           .plug("EXP", lits)

rules =
  wkld
    .to_egraph()
    .find_candidates()
    .select_rules(limits)
```



Nandi et al. 2021; Pal et al. 2023

ENUMO DSL

```
lits     = Workload { a b 0 1 }
sums     = Workload { (+ EXP EXP) }
products = Workload { (* EXP EXP) }
sums_of_products =
   sums.plug(“EXP”, products.plug(“EXP”, lits))
```

Pal et al. 2023

ENUMO DSL

```
lits     = Workload { a b 0 1 }
sums     = Workload { (+ EXP EXP) }
products = Workload { (* EXP EXP) }
sums_of_products =
   sums.plug("EXP", products.plug("EXP", lits))
```

```
(* a a)    (* b a)    (* c a)    (* 0 a)    (* 1 a)
(* a b)    (* b b)    (* c b)    (* 0 b)    (* 1 b)
(* a c)    (* b c)    (* c c)    (* 0 c)    (* 1 c)
(* a 0)    (* b 0)    (* c 0)    (* 0 0)    (* 1 0)
(* a 1)    (* b 1)    (* c 1)    (* 0 1)    (* 1 1)
```

Pal et al. 2023

ENUMO DSL

```
lits     = Workload { a b 0 1 }
sums     = Workload { (+ EXP EXP) }
products = Workload { (* EXP EXP) }
sums_of_products =
   sums.plug(“EXP”, products.plug(“EXP”, lits))
```

```
(+ (* a a) (* a a))  (+ (* a a) (* b a))  (+ (* a a) (* c a))
(+ (* a a) (* a b))  (+ (* a a) (* b b))  (+ (* a a) (* c b))
(+ (* a a) (* a c))  (+ (* a a) (* b c))  (+ (* a a) (* c c))
(+ (* a a) (* a 0))  (+ (* a a) (* b 0))         . . .
(+ (* a a) (* a 1))  (+ (* a a) (* b 1))  (+ (* 1 1) (* 1 1))
```

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug(“EXP”, e2)
  .filter(λt. size t < 4)
```

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2)
  .filter(λt. size t < 4)
```

```
(~ 1)                   (+ (+ 2 3) 1)
(~ (+ 2 3))             (+ (+ 2 3) (+ 2 3))
(~ (+ (+ 3 4) 5))       (+ (+ 2 3) (+ (+ 4 5) 6))
(+ 1 1)                 (+ (+ (+ 4 5) 6) 1)
(+ 1 (+ 2 3))           (+ (+ (+ 4 5) 6) (+ 2 3))
(+ 1 (+ (+ 4 5) 6))     (+ (+ (+ 4 5) 6) (+ (+ 4 5) 6))
```

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2)
  .filter(λt. size t < 4)
```

(~ 1)
~~(~ (+ 2 3))~~
~~(~ (+ (+ 3 4) 5))~~
(+ 1 1)
~~(+ 1 (+ 2 3))~~
~~(+ 1 (+ (+ 4 5) 6))~~
~~(+ (+ 2 3) 1)~~
~~(+ (+ 2 3) (+ 2 3))~~
~~(+ (+ 2 3) (+ (+ 4 5) 6))~~
~~(+ (+ (+ 4 5) 6) 1)~~
~~(+ (+ (+ 4 5) 6) (+ 2 3))~~
~~(+ (+ (+ 4 5) 6) (+ (+ 4 5) 6))~~

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2)
  .filter(λt. size t < 4)
```

(~ 1)
~~(~ (+ 2 3))~~
~~(~ (+ (+ 3 4) 5))~~
(+ 1 1)
~~(+ 1 (+ 2 3))~~
~~(+ 1 (+ (+ 4 5) 6))~~
~~(+ (+ 2 3) 1)~~
~~(+ (+ 2 3) (+ 2 3))~~
~~(+ (+ 2 3) (+ (+ 4 5) 6))~~
~~(+ (+ (+ 4 5) 6) 1)~~
~~(+ (+ (+ 4 5) 6) (+ 2 3))~~
~~(+ (+ (+ 4 5) 6) (+ (+ 4 5) 6))~~

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2.filter(λt. size t < 4))
  .filter(λt. size t < 4)
```

```
1
(+ 2 3)
```

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2.filter(λt. size t < 4))
  .filter(λt. size t < 4)
```

```
1
(+ 2 3)
```

```
(~ 1)
(~ (+ 2 3))
(+ 1 1)
(+ 1 (+ 2 3))
(+ (+ 2 3) 1)
(+ (+ 2 3) (+ 2 3))
```

Pal et al. 2023

ENUMO DSL

```
e1 = Workload { (~ EXP) (+ EXP EXP) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXP", e2.filter(λt. size t < 4))
  .filter(λt. size t < 4)
```

1
(+ 2 3)

(~ 1)
~~(~ (+ 2 3))~~
(+ 1 1)
~~(+ 1 (+ 2 3))~~
~~(+ (+ 2 3) 1)~~
~~(+ (+ 2 3) (+ 2 3))~~

Pal et al. 2023

ENUMO DSL

Optimization: Pushing Filters through Plugs

$$[\![\text{Filter } f \text{ (Plug W1 s W2) }]\!] = [\![\text{Filter } f \text{ (Plug W1 s (Filter } f \text{ W2))}]\!]$$

Pal et al. 2023

ENUMO DSL

Optimization: Pushing Filters through Plugs

[[Filter *f* (Plug W1 s W2)]] = [[Filter *f* (Plug W1 s (Filter *f* W2))]]

Requires monotonicity of *f*

Pal et al. 2023

ENUMO DSL

A filter f is monotonic if,
for every term t satisfying f,
every subterm $s \in t$
also satisfies f

Pal et al. 2023

ENUMO DSL

Monotonic

```
Excludes((+ (* x x) (* y y)), “z”)
```

A filter f is monotonic if,
for every term t satisfying f,
every subterm $s \in t$
also satisfies f

```
Contains((+ (* x x) (* y y)), “x”)
```

Not monotonic

Pal et al. 2023

Comparison to Ruler

Domain	Enumo LOC	Enumo Time (s)	# Enumo	# Ruler	Enumo → Ruler	Ruler → Enumo
bool	44	0.25	64	51		
bv4	21	7.10	180	84		
bv32	20	48.44	120	78		
rational	51	6.37	131	113		

Pal et al. 2023

Comparison to Ruler

Domain	ENUMO LOC	ENUMO Time (s)	# ENUMO	# Ruler	ENUMO → Ruler	Ruler → ENUMO
bool	44	0.25	64	51		
bv4	21	7.10	180	84		
bv32	20	48.44	120	78		
rational	51	6.37	131	113		

To test whether a ruleset, *R*, can derive a rule, $e_1 \Rightarrow e_2$:

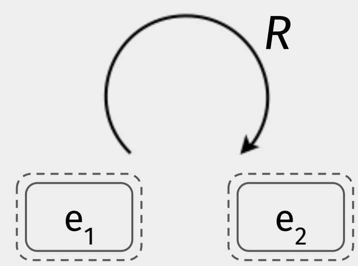
Initialize e-graph with e_1 and e_2

Pal et al. 2023

Comparison to Ruler

Domain	Enumo LOC	Enumo Time (s)	# Enumo	# Ruler	Enumo → Ruler	Ruler → Enumo
bool	44	0.25	64	51		
bv4	21	7.10	180	84		
bv32	20	48.44	120	78		
rational	51	6.37	131	113		

To test whether a ruleset, *R*, can derive a rule, $e_1 \Rightarrow e_2$:



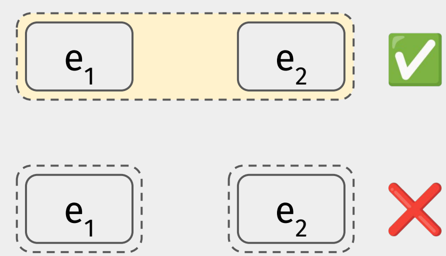
Run equality saturation with *R*

Pal et al. 2023

Comparison to Ruler

Domain	Enumo LOC	Enumo Time (s)	# Enumo	# Ruler	Enumo → Ruler	Ruler → Enumo
bool	44	0.25	64	51		
bv4	21	7.10	180	84		
bv32	20	48.44	120	78		
rational	51	6.37	131	113		

To test whether a ruleset, *R*, can derive a rule, $e_1 \Rightarrow e_2$:



Derivable if the e-classes merge

Pal et al. 2023

Comparison to Ruler

Key Idea:

Enumo outperforms prior work

Domain	Enumo LOC	Enumo Time (s)	# Enumo	# Ruler	Enumo → Ruler	Ruler → Enumo
bool	44	0.25	64	51	100%	87.5%
bv4	21	7.10	180	84	100%	38.3%
bv32	20	48.44	120	78	100%	58.3%
rational	51	6.37	131	113	100%	62.6%

Pal et al. 2023

Comparison to Ruler

Key Idea:

Rule Inference is fast

Domain	ENUMO LOC	ENUMO Time (s)	# ENUMO	# Ruler	ENUMO → Ruler	Ruler → ENUMO
bool	44	0.25	64	51	100%	87.5%
bv4	21	7.10	180	84	100%	38.3%
bv32	20	48.44	120	78	100%	58.3%
rational	51	6.37	131	113	100%	62.6%

Pal et al. 2023

Case Study: Large Grammar

```
G = Workload {
    (<   EXPR EXPR)
    (<=  EXPR EXPR)
    (==  EXPR EXPR)
    (!=  EXPR EXPR)
    (!   EXPR)
    (-   EXPR)
    (&&  EXPR EXPR)
    (||  EXPR EXPR)
    (^   EXPR EXPR)
    (+   EXPR EXPR)
    (-   EXPR EXPR)
    (*   EXPR EXPR)
    (/   EXPR EXPR)
    (min EXPR EXPR)
    (max EXPR EXPR)
    (select EXPR EXPR EXPR)
}
```

725 rules with no side conditions

Term Size	# Rules	ENUMO → Halide

Ragan-Kelley et al. 2013

Case Study: Large Grammar

```
G = Workload {
    (<   EXPR EXPR)
    (<=  EXPR EXPR)
    (==  EXPR EXPR)
    (!=  EXPR EXPR)
    (!   EXPR)
    (-   EXPR)
    (&&  EXPR EXPR)
    (||  EXPR EXPR)
    (^   EXPR EXPR)
    (+   EXPR EXPR)
    (-   EXPR EXPR)
    (*   EXPR EXPR)
    (/   EXPR EXPR)
    (min EXPR EXPR)
    (max EXPR EXPR)
    (select EXPR EXPR EXPR)
}
```

Term Size	# Rules	ENUMO → Halide
3	96	2.9%
4	224	6.9%
5	485	42.6%
6	TIMEOUT	TIMEOUT

Ragan-Kelley et al. 2013; Pal et al. 2023

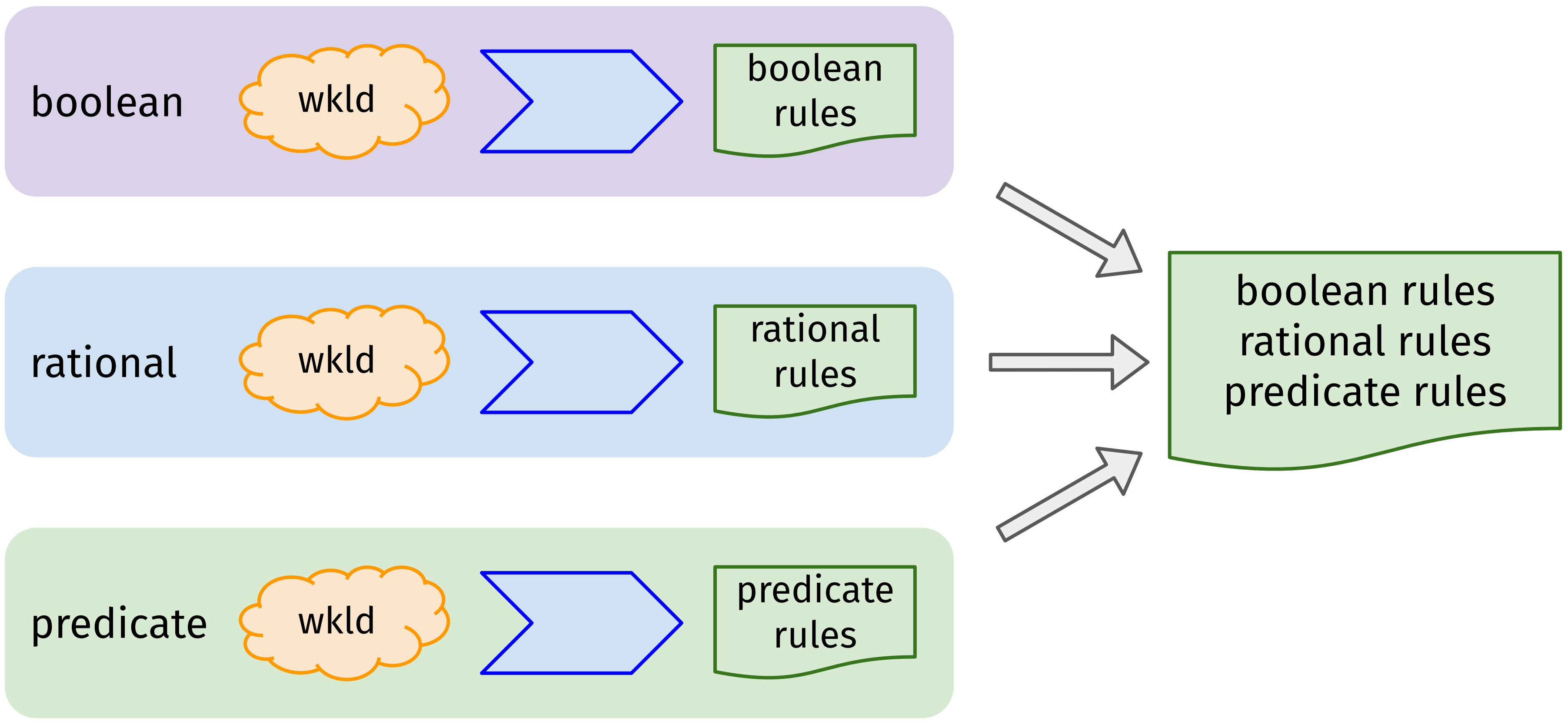
Case Study: Large Grammar

```
G = Workload {
    (<   EXPR EXPR)
    (<=  EXPR EXPR)
    (==  EXPR EXPR)
    (!=  EXPR EXPR)
    (!   EXPR)
    (-   EXPR)
    (&&  EXPR EXPR)
    (||  EXPR EXPR)
    (^   EXPR EXPR)
    (+   EXPR EXPR)
    (-   EXPR EXPR)
    (*   EXPR EXPR)
    (/   EXPR EXPR)
    (min EXPR EXPR)
    (max EXPR EXPR)
    (select EXPR EXPR EXPR)
}
```

Domain experts know which operators are closely related

Ragan-Kelley et al. 2013; Pal et al. 2023

Case Study: Large Grammar



Pal et al. 2023

Case Study: Large Grammar

Key Idea:

Guided search enables progress past exponential blowup

boolean rules
rational rules
predicate rules

Custom Workloads

ENUMO Program
(141 LOC)

Term Size	Time (s)	# Rules	ENUMO → Halide
Custom	51.76	845	90.6%

Pal et al. 2023

Case Study: Cross-Domain Inference

BV 4	BV 128
✅ Fast to synthesize	❌ Slow to synthesize

Pal et al. 2023

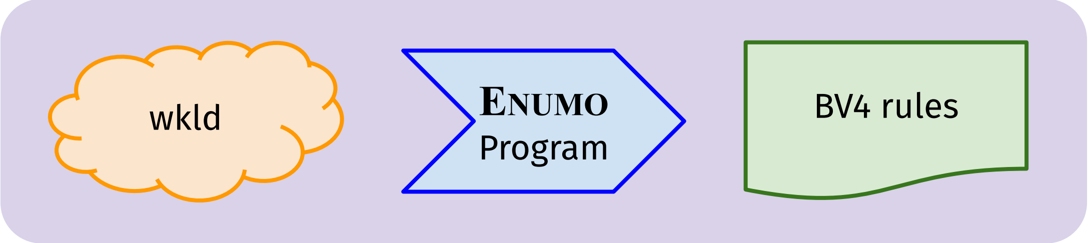
Case Study: Cross-Domain Inference

BV 4

✅ Fast to synthesize

BV 128

❌ Slow to synthesize



Pal et al. 2023

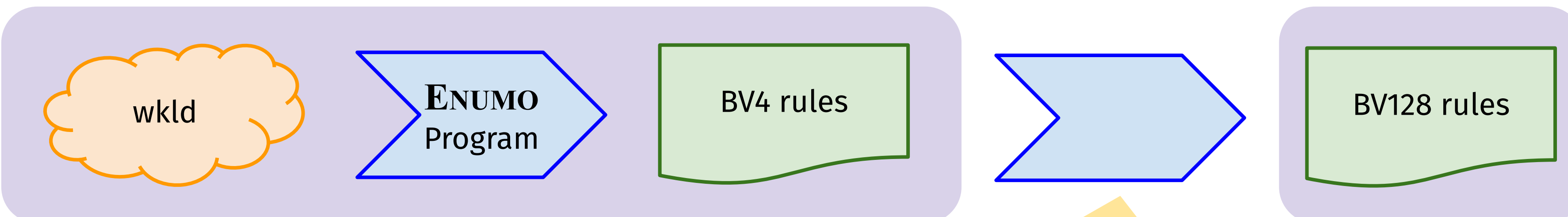
Case Study: Cross-Domain Inference

BV 4

✅ Fast to synthesize

BV 128

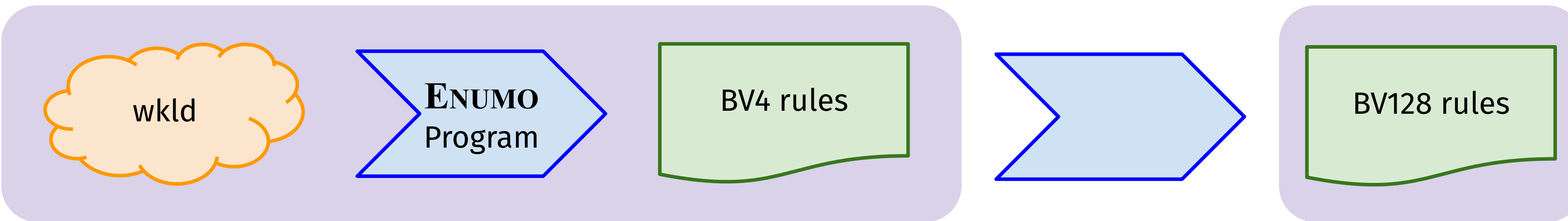
❌ Slow to synthesize



Cast rules from BV4 to BV128 and re-validate

Pal et al. 2023

Case Study: Cross-Domain Inference



Generated Rules (Time)	Ported BV4 Rules (Time)	Ported → Generated
190 (1784.14)	210 (38.68)	91%

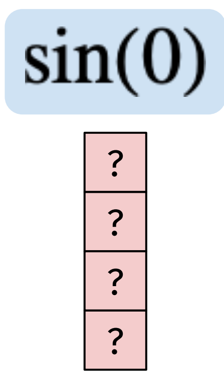
Directly synthesized BV128 rules

Of the 246 BV4 rules, 210 are sound for BV128

The ported rules have almost as much proving power at a fraction of the cost

Pal et al. 2023

Case Study: No Interpreter



Pal et al. 2023

Case Study: No Interpreter

$$\mathrm{cis}(x) = \cos(x) + i\ \sin(x)$$

$$\sin(0) \rightarrow \frac{\mathrm{cis}(0) - \mathrm{cis}(-0)}{2i}$$

Pal et al. 2023

Case Study: No Interpreter

$$\text{cis}(x) = \cos(x) + i\,\sin(x)$$

$$\sin(0) \rightarrow \frac{\text{cis}(0) - \text{cis}(-0)}{2i}$$

$$-x = 0 - x$$
$$x - x = 0$$
$$0/x = 0$$

$$\frac{\text{cis}(0) - \text{cis}(-0)}{2i} \rightarrow 0$$

Pal et al. 2023

Case Study: No Interpreter

$$\mathrm{cis}(x) = \cos(x) + i\ \sin(x)$$

$$-x = 0 - x$$
$$x - x = 0$$
$$0/x = 0$$

$$\sin(0)$$

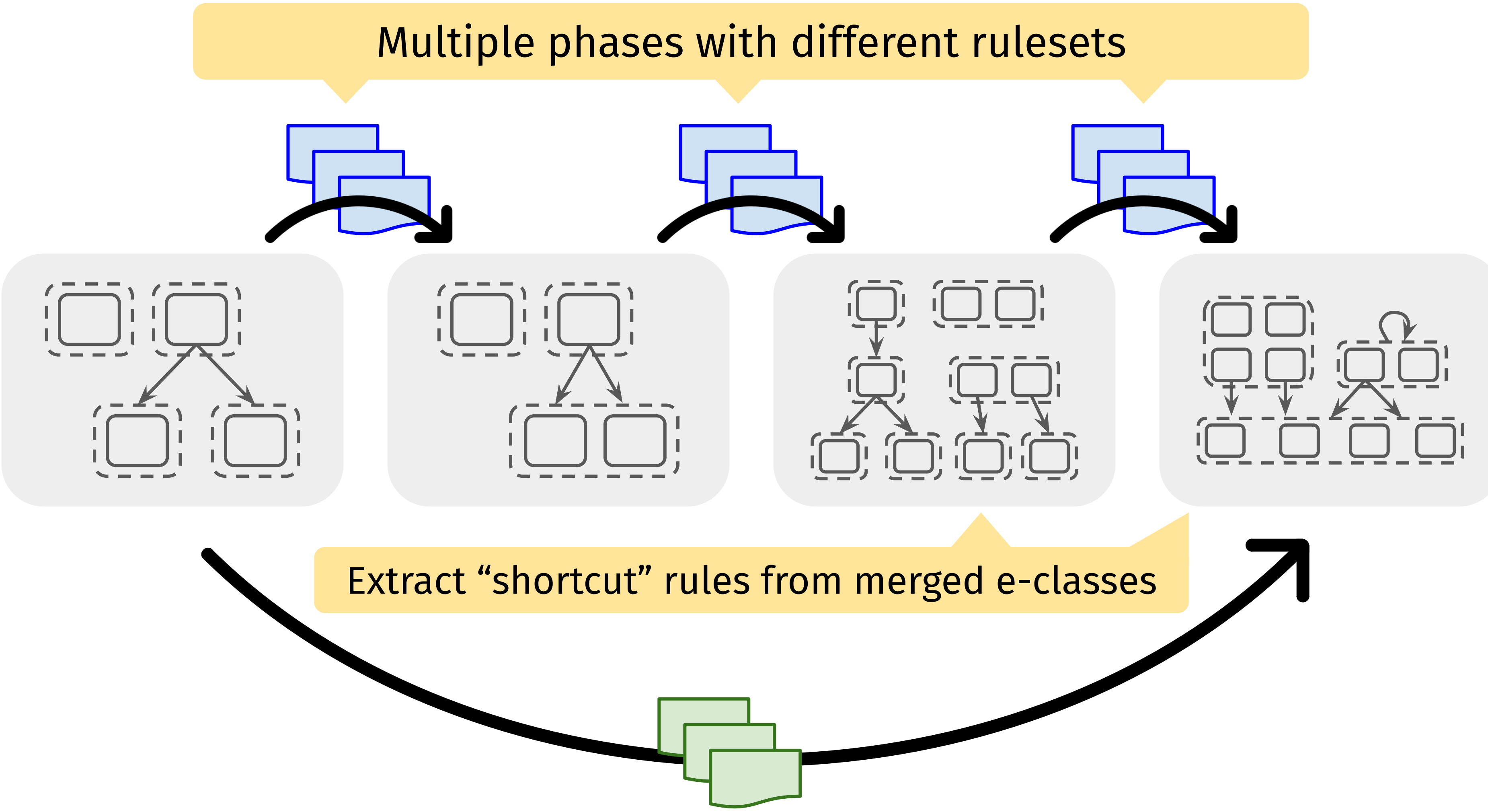
$$\frac{\mathrm{cis}(0) - \mathrm{cis}(-0)}{2i}$$

$$0$$

$$\sin(0) \implies 0$$

Pal et al. 2023

Case Study: No Interpreter



Pal et al. 2023

Case Study: No Interpreter

Key Idea:

New Domains for Rule Inference

$$\sin(b + a) \Rightarrow \sin(b) \cdot \cos(a) + \sin(a) \cdot \cos(b)$$

$$\sin(b) \cdot \sin(a) \Rightarrow (\cos(b - a) - \cos(b + a)) / 2$$

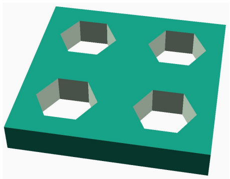
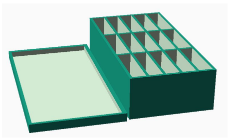
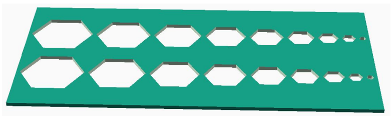
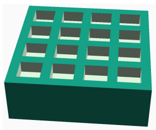
$$c^{ba} \Rightarrow (c^{a})^{b}$$

$$(c^{b})^{\log(a)} \Rightarrow (a^{b})^{\log(c)}$$

$$\sqrt{(b^{a})} \Rightarrow (\sqrt{b})^{a}$$

$$\mathrm{Scale}(a, b, c, \mathrm{Trans}(d, e, f, s)) \Rightarrow \mathrm{Trans}(da, eb, fc, \mathrm{Scale}(a, b, c, s))$$

$$\mathrm{Cube}(ad, be, cf) \Rightarrow \mathrm{Scale}(a, b, c, \mathrm{Cube}(d, e, f))$$

What's next?

Conditional Rule Inference

Verified Optimization

Large Language Models

What’s next?

Conditional Rule Inference

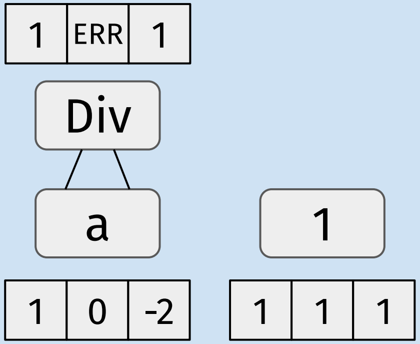
Verified Optimization

Large Language Models

Most rules depend on context

x / x ⇒ 1 **when x ≠ 0**

Candidate generation will miss this because the arrays for the two e-classes won’t match



Can we infer useful, simple side conditions for partial rules?

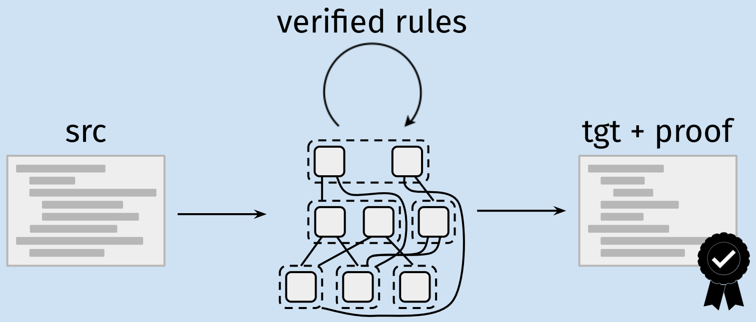
What's next?

Conditional Rule Inference

Verified Optimization

Large Language Models

Can we build a verified compiler using equality saturation?



Can we automatically infer rules for program optimization?

What’s next?

Conditional Rule Inference

Verified Optimization

Large Language Models

Which parts of rule inference can language models help with?

Does it vary by domain?

```
x && true ⟹ x
```

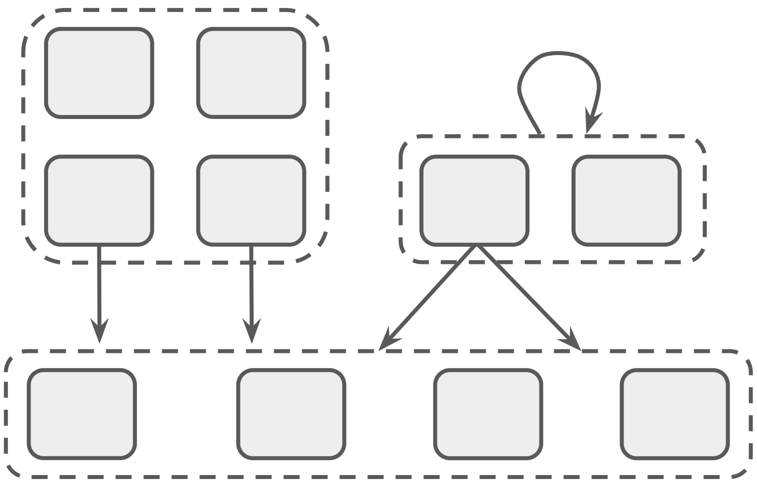
```
sin 0 ⟹ 0
```

$c^{ba} \Longrightarrow (c^{a})^{b}$

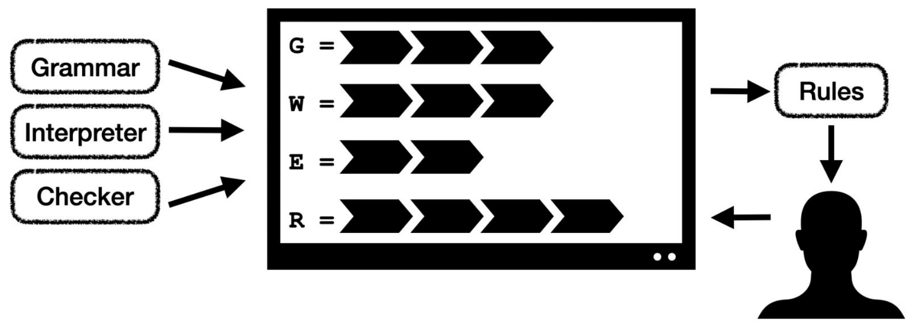
```
Cube(ad, be, cf) ⟹
     Scale(a, b, c, Cube(d, e, f)
```

Thank you!

Fast, Flexible, Robust Term Rewriting via Equality Saturation



Fast, Flexible, Robust Rule Inference via Programmable Theory Exploration



PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

PLSE