Equality Saturation Theory Exploration á la Carte

Anjali Pal, Brett Saiki, Ryan Tjoa*, Cynthia Richey*, Amy Zhu, Oliver Flatt, Max Willsey, Zachary Tatlock, Chandrakana Nandi







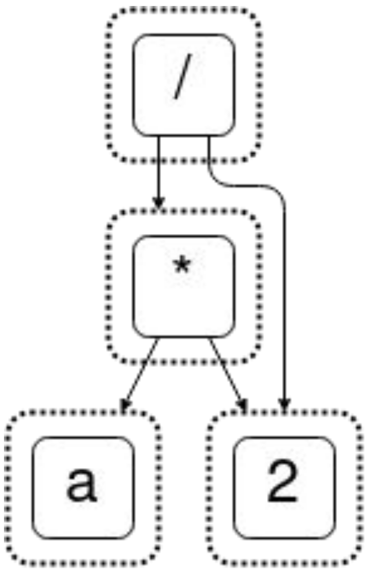


PLSE

* Equal Contribution

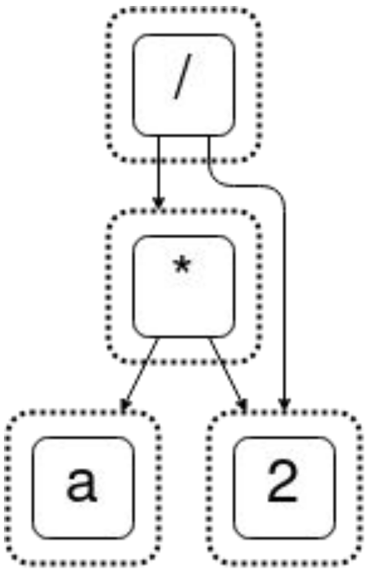
PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Equality Saturation



Tate et al. 2009; Willsey et al. 2021

Equality Saturation

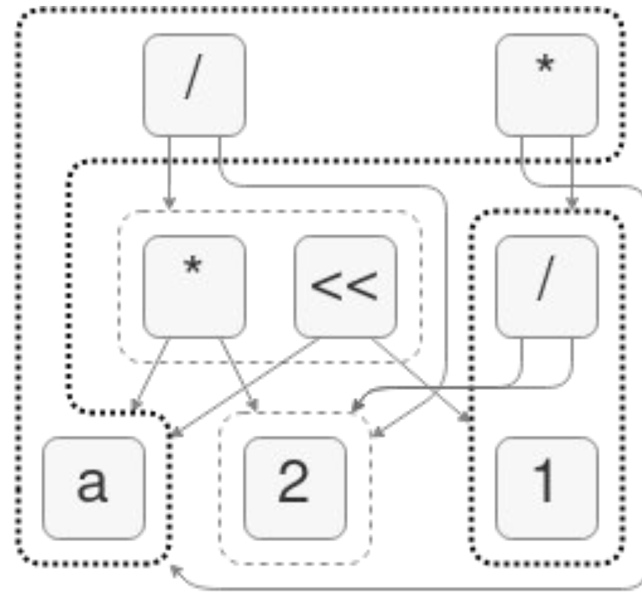


x * 2 ⇒ x << 1

(x * y) / z ⇒ x * (y / z)

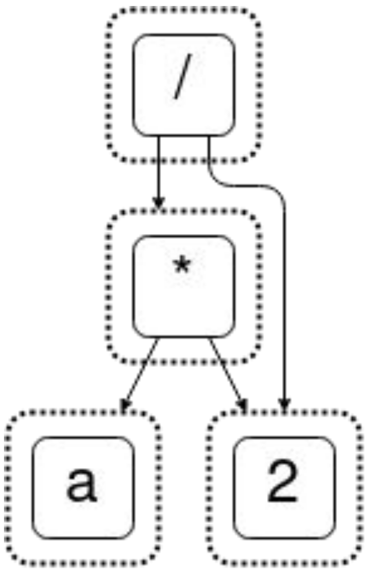
x / x ⇒ 1

x * 1 ⇒ x

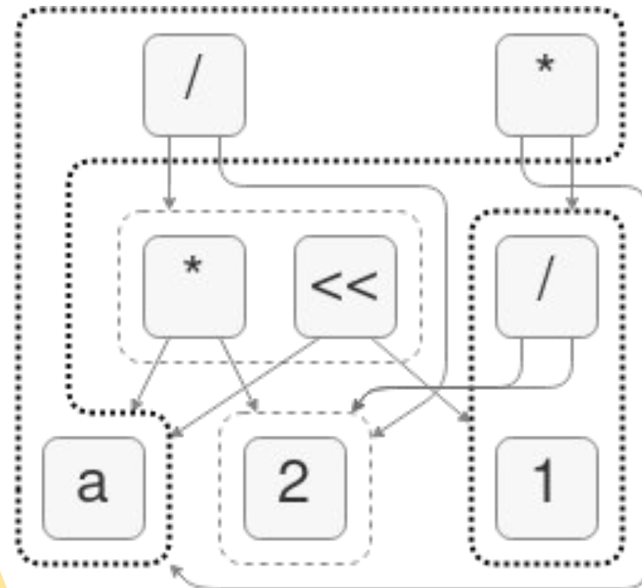


Tate et al. 2009; Willsey et al. 2021

Equality Saturation



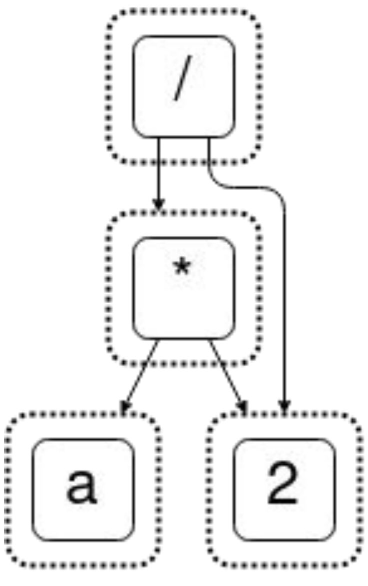
x * 2 ⇒ x << 1
(x * y) / z ⇒ x * (y / z)
x / x ⇒ 1
x * 1 ⇒ x



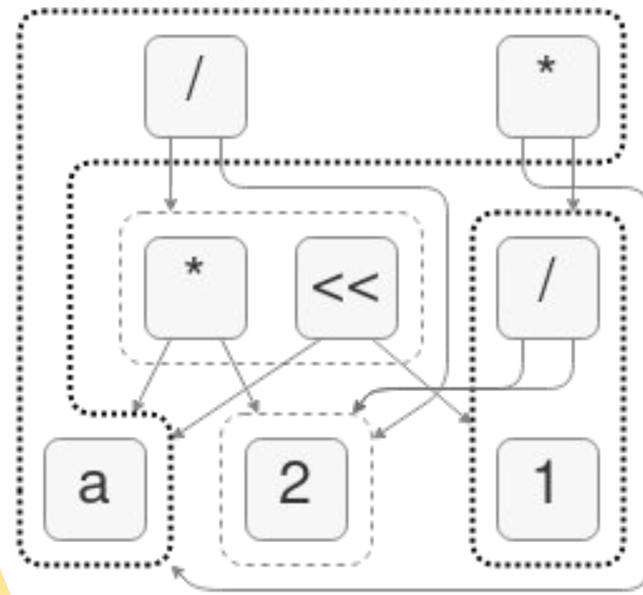
Until saturation

Tate et al. 2009; Willsey et al. 2021

Equality Saturation



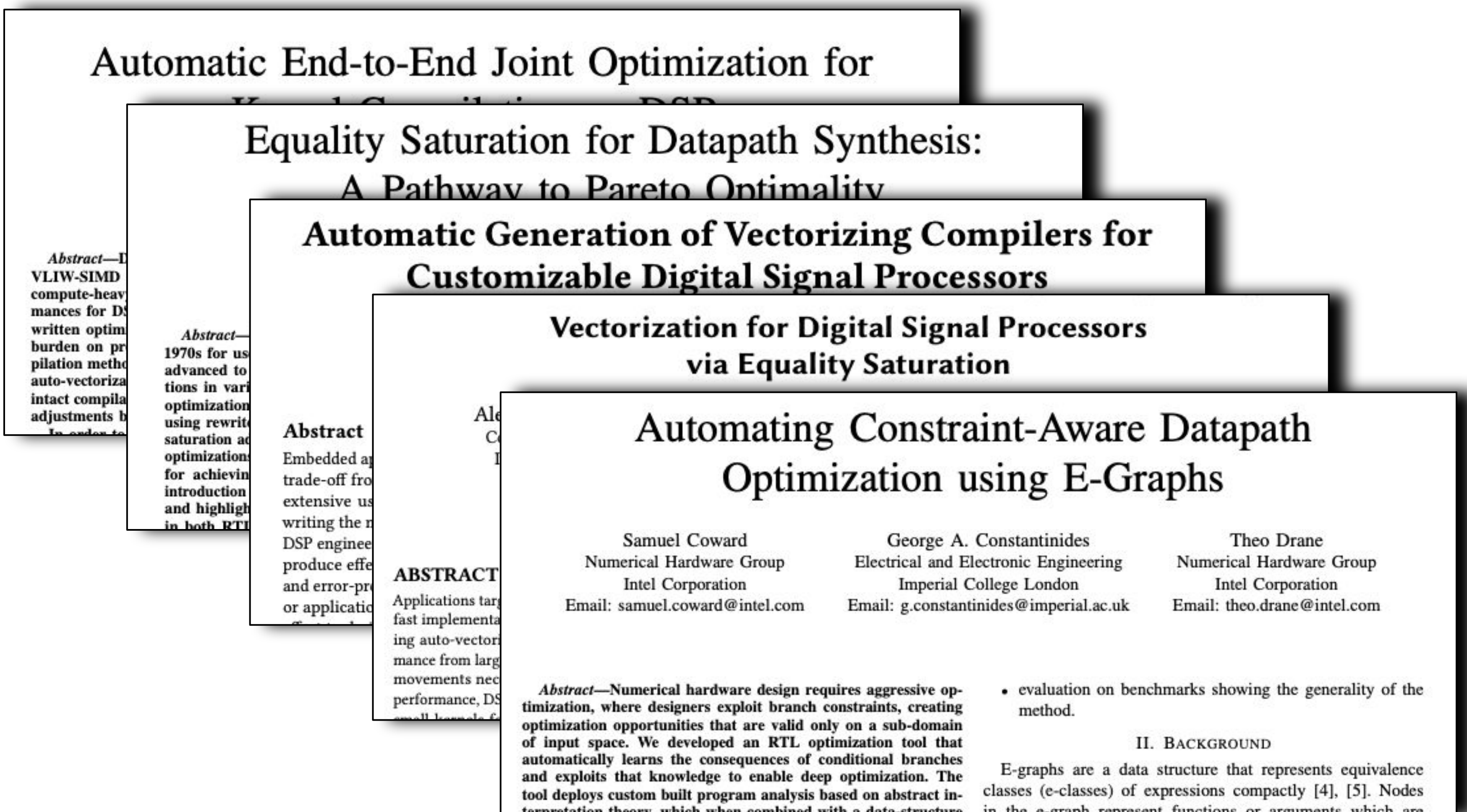
x * 2 ⇒ x << 1
(x * y) / z ⇒ x * (y / z)
x / x ⇒ 1
x * 1 ⇒ x



Until saturation
… or resource limit

Tate et al. 2009; Willsey et al. 2021

Equality Saturation is everywhere!

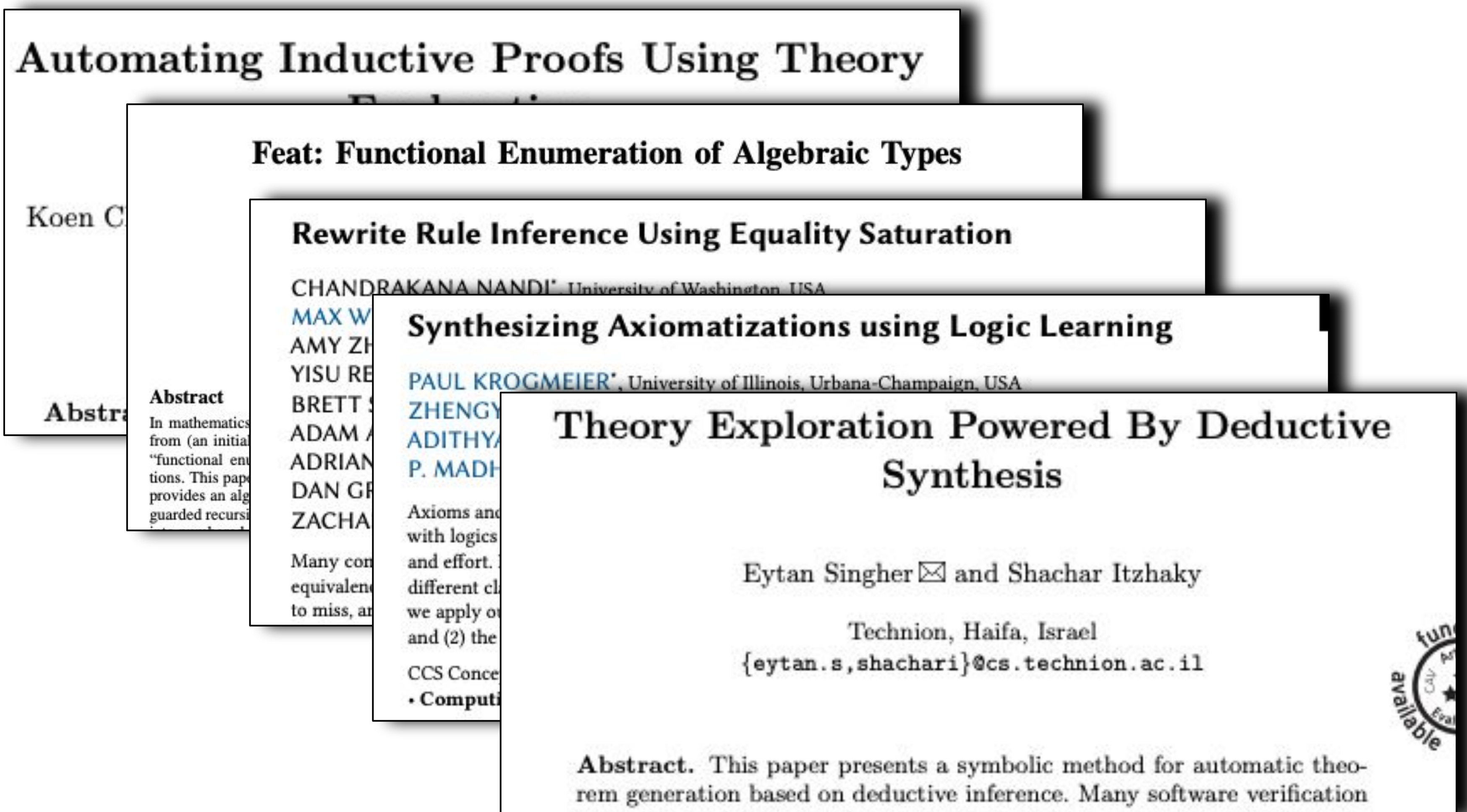


Coward et al. 2023; VanHattum et al. 2021; Thomas and Bornholt 2024; Ustan et al. 2023; Zhao et al. 2023

Equality Saturation is only as powerful as the rules used

Writing rewrite rules manually is hard

Automated Theory Explorers



Singher and Itzhaky 2021; Krogmeier et al. 2022; Nandi et al. 2021; Duregård et al. 2012; Claessen et al. 2013

Problem: Despite recent work in automated theory exploration, building and maintaining rulesets still requires significant engineering effort

Hypothesis: Traditional theory exploration is too inflexible

Hypothesis: Traditional theory exploration is too inflexible

Proposal: Programmable theory exploration using the ENUMO DSL

Hypothesis: Traditional theory exploration is too inflexible

Proposal: Programmable theory exploration using the **ENUMO** DSL

Benefits: Scales better and enables new strategies

1. Traditional theory exploration

2. The ENUMO DSL

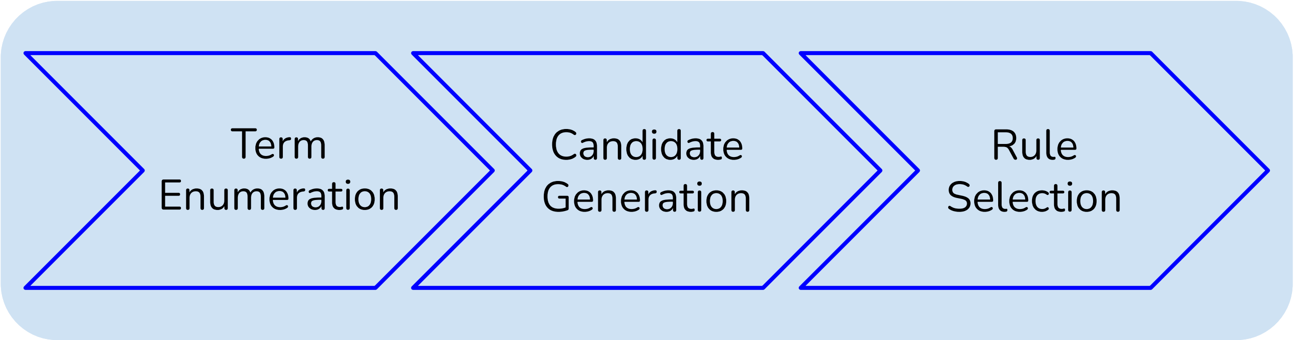
3. Novel scalable rule-finding strategies

1. Traditional theory exploration

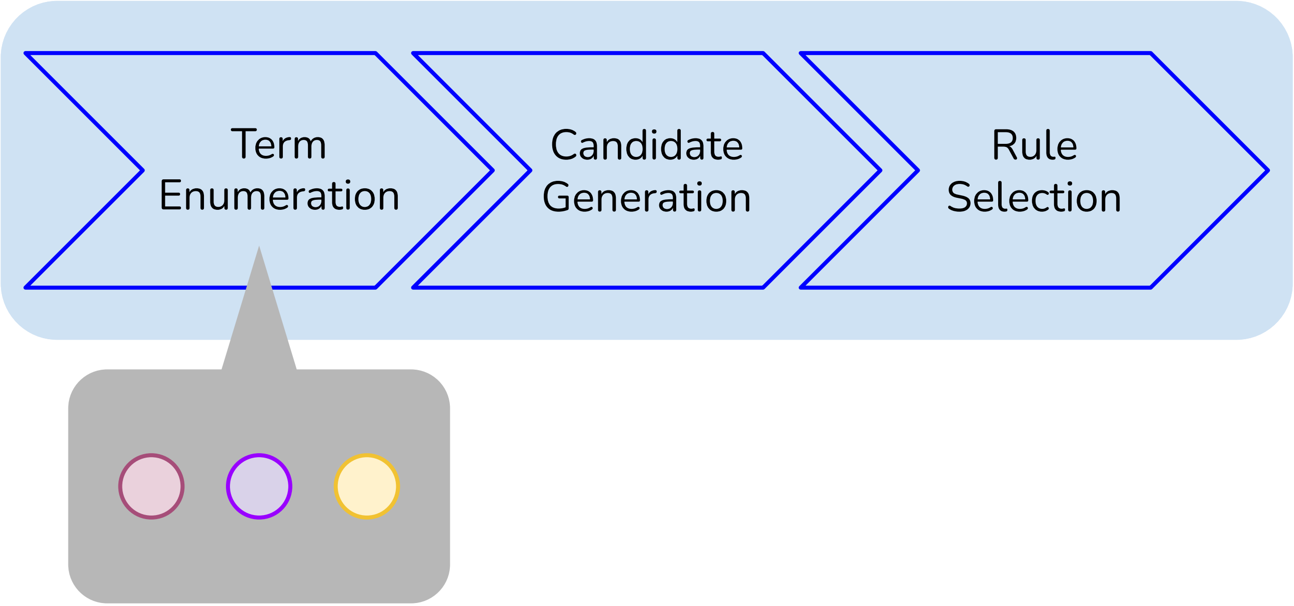
2. The ENUMO DSL

3. Novel scalable rule-finding strategies

Traditional Theory Exploration

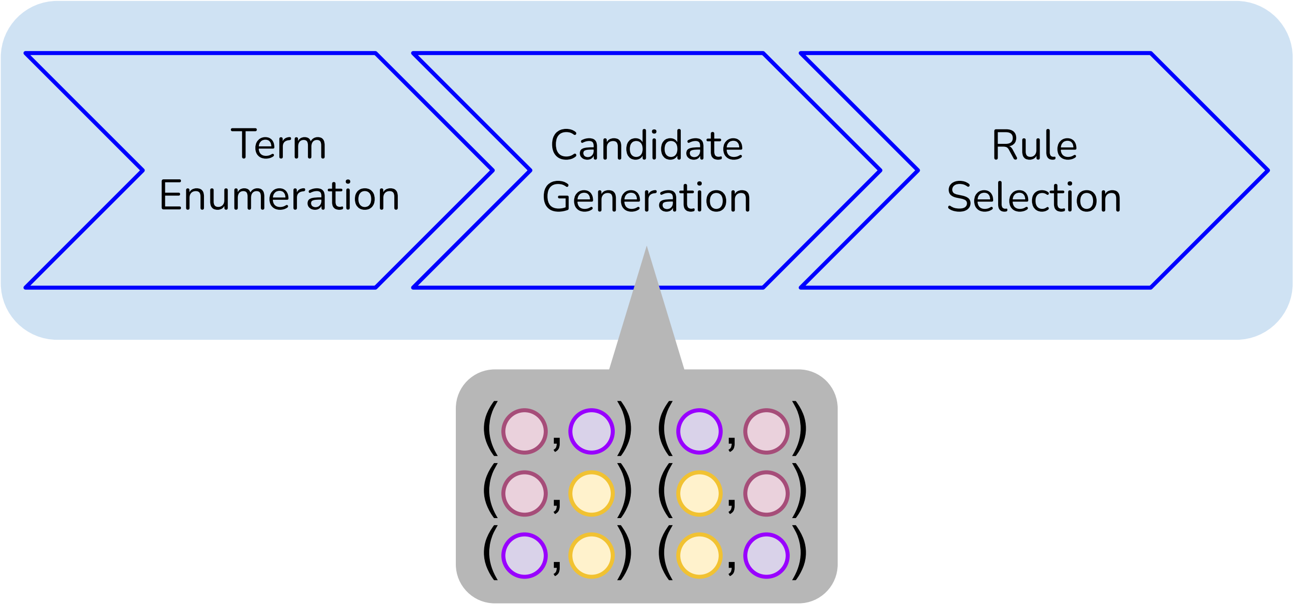


Traditional Theory Exploration



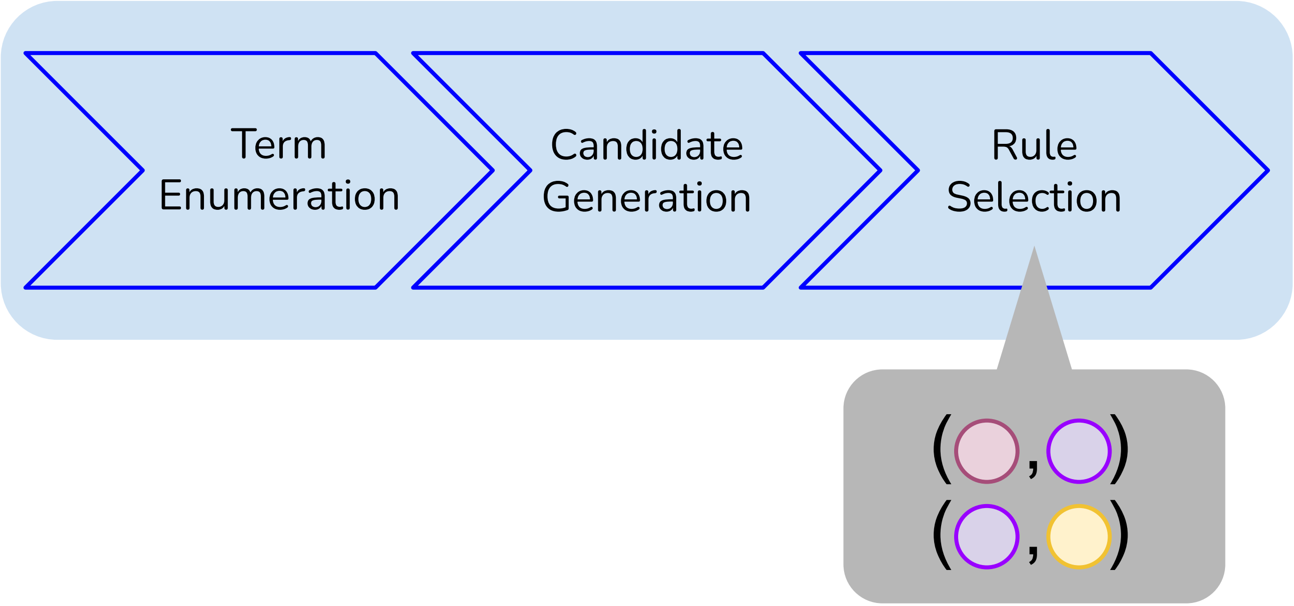
Barrett et al. 2011; Nandi et al. 2021

Traditional Theory Exploration



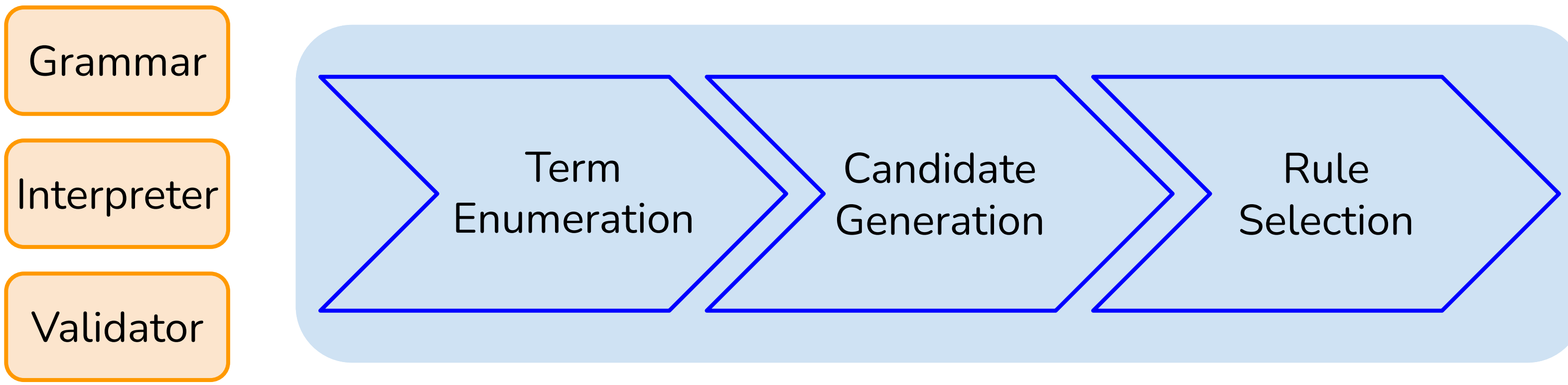
Bansal and Aiken 2006; Nandi et al. 2021; Nötzli et al. 2019; Singher and Itzhaky 2021

Traditional Theory Exploration



Nandi et al. 2021

Traditional Theory Exploration



Traditional Theory Exploration

Grammar

<EXPR> :=
| (*Lit* n)
| (*Var* v)
| (~ *<EXPR>*)
| (+ *<EXPR>* *<EXPR>*)
| (* *<EXPR>* *<EXPR>*)
| (- *<EXPR>* *<EXPR>*)
| (/ *<EXPR>* *<EXPR>*)

Traditional Theory Exploration

Grammar

Interpreter

```
def eval(expr):
  match expr
    |(Const n) => n
    |(Var v)   => lookup(v)
    |(~ e)     => -1 * eval(e)
    |(+ e1 e2) => eval(e1) + eval(e2)
    |(* e1 e2) => eval(e1) * eval(e2)
    |(- e1 e2) => eval(e1) - eval(e2)
    |(/ e1 e2) => eval(e1) / eval(e2)
```

Traditional Theory Exploration

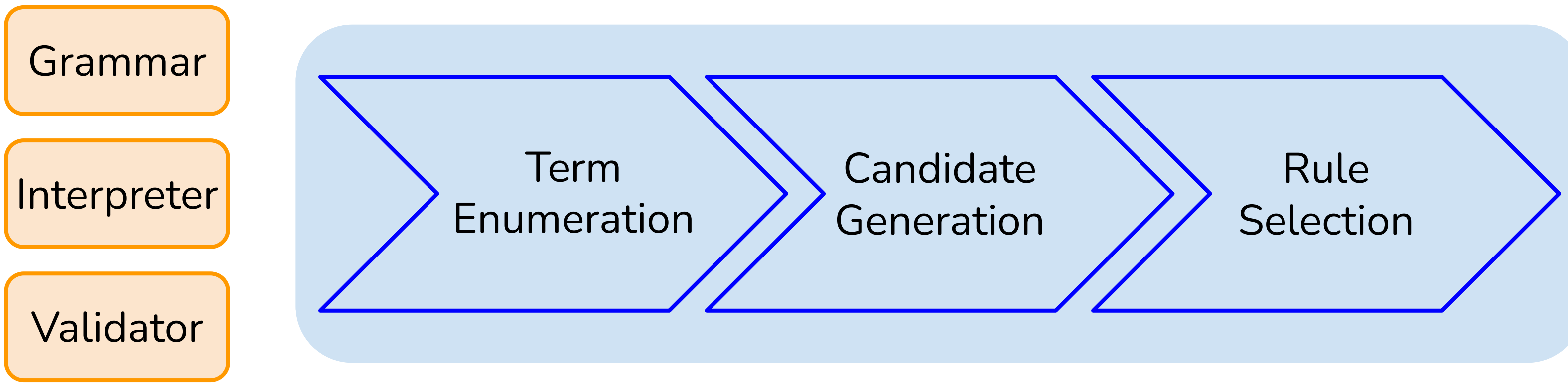
Grammar

Interpreter

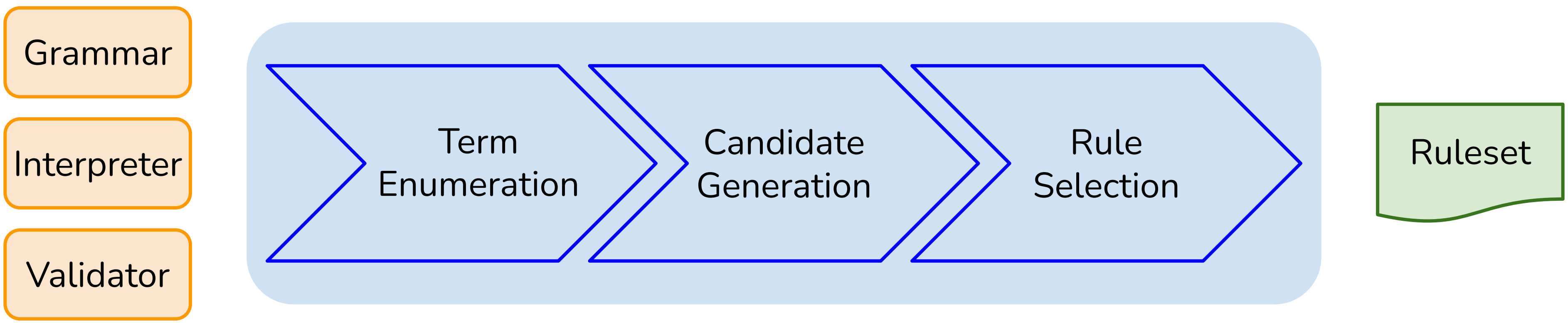
Validator

```
def is_valid(lhs, rhs):
  l = lhs.to_z3()
  r = rhs.to_z3()
  z3.assert(l.eq(r).not())
  match solver.check()
    |Unsat => true
    |Sat |Unknown => false
```

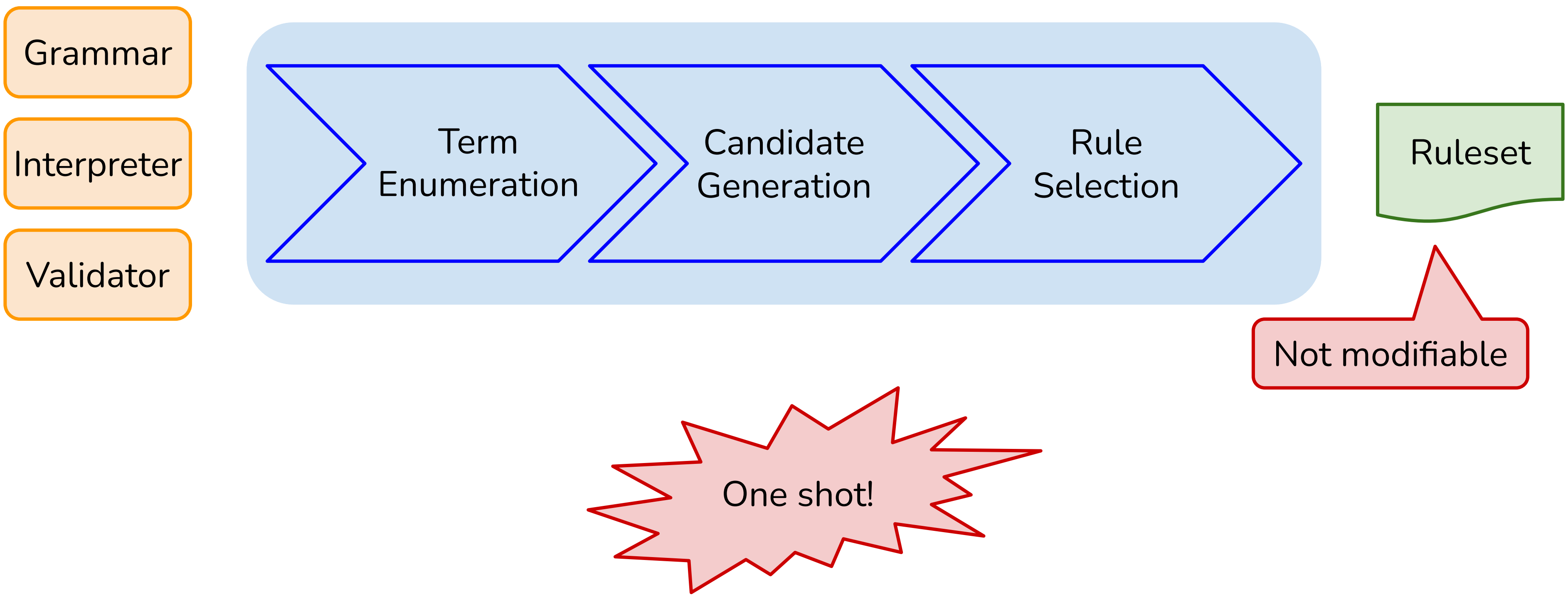
Traditional Theory Exploration



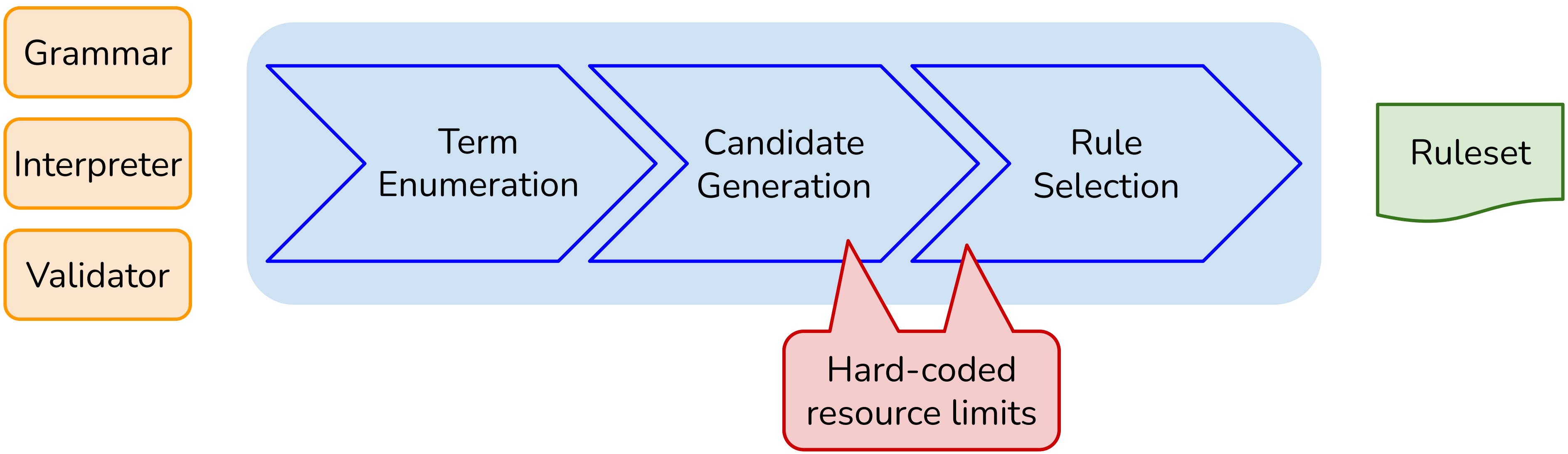
Traditional Theory Exploration



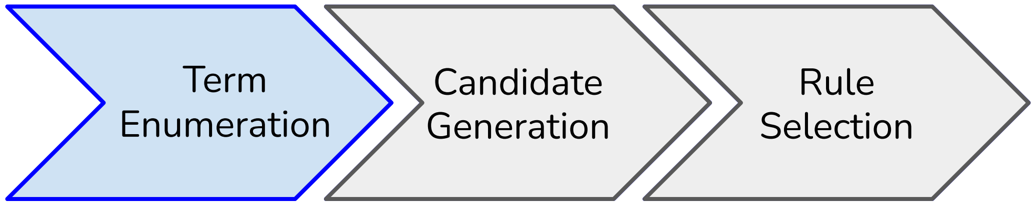
Traditional Theory Exploration



Traditional Theory Exploration



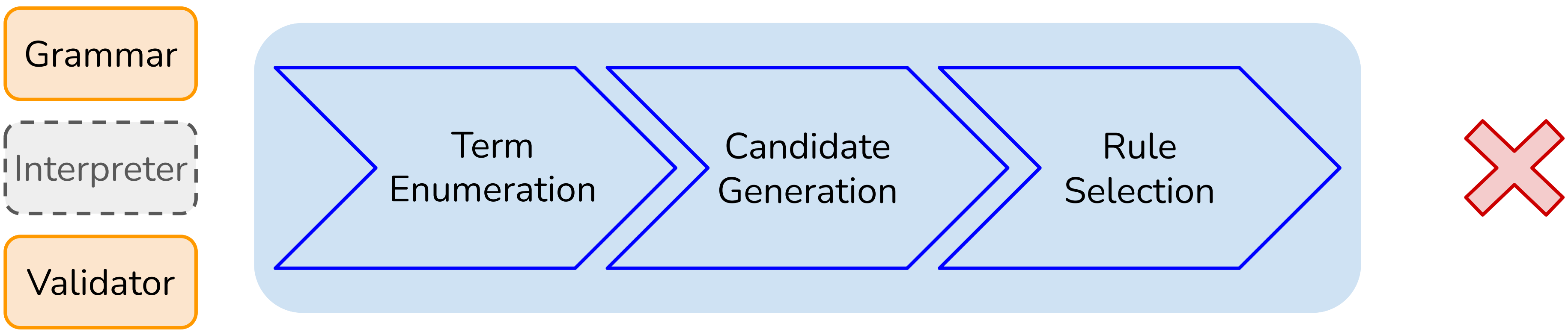
Traditional Theory Exploration



<*EXPR*> := (*Lit* n) | (*Var* v) | (~ <*EXPR*>) | (+ <*EXPR*>, <*EXPR*>) | (* <*EXPR*>, <*EXPR*>)

Size 1	Size 2	Size 3	Size 4	Size 5	Size 6
a	(~ a)	(~ (~ 0))	(+ a (~ a))	(~ (~ (~ (~ 0))))	(~ (~ (~ (~ (~ 0)))))
b	(~ b)	(~ (~ 1))	(+ a (~ b))	(~ (~ (~ (~ 1))))	(~ (~ (~ (~ (~ 1)))))
c	(~ c)	(+ a a)	(+ (~ b) a)	(~ (~ (+ a a)))	(~ (~ (~ (+ a a))))
-1	(~ -1)	(+ a b)	(+ (~ b) b)	(~ (~ (+ a b)))	(~ (~ (~ (+ a b))))
0	(~ 0)	(+ a 0)	(+ b (~ a))	(~ (~ (+ a 0)))	(~ (~ (~ (+ a 0))))
1	(~ 1)	(+ a 1)	(+ b (~ b))	(~ (~ (+ a 1)))	(~ (~ (~ (+ a 1))))
		...	...	...	...
6	6	78	366	4902	52134 😱

Traditional Theory Exploration



1. Traditional theory exploration

2. The ENUMO DSL

3. Novel scalable rule-finding strategies

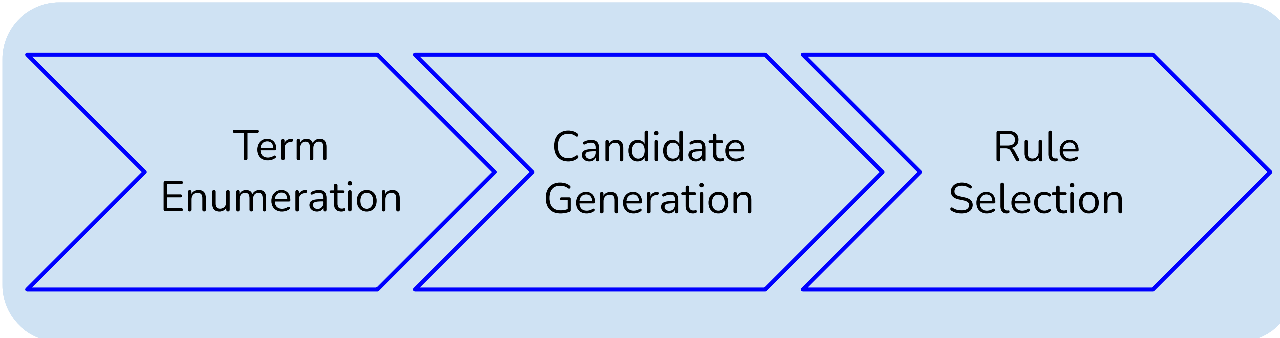
```
(~ (~ (~ (~ (~ a)))))
```

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

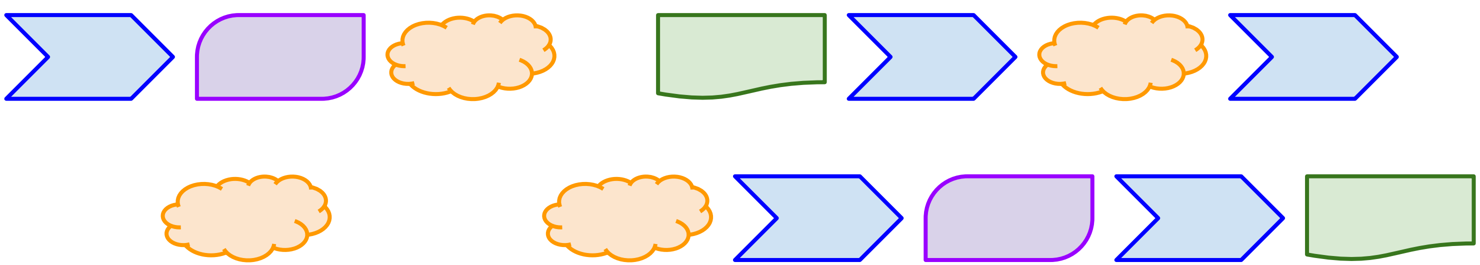
Insight: Users have intuition about which parts of the domain are worth exploring

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

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



We turn theory explorers inside out to expose a small set of useful operators for rule inference



ENUMO DSL

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

ENUMO DSL

```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }
```

<*EXPR*> :=

| (*Lit* n)

| (~ <*EXPR*>)

| (+ <*EXPR*> <*EXPR*>)

ENUMO DSL

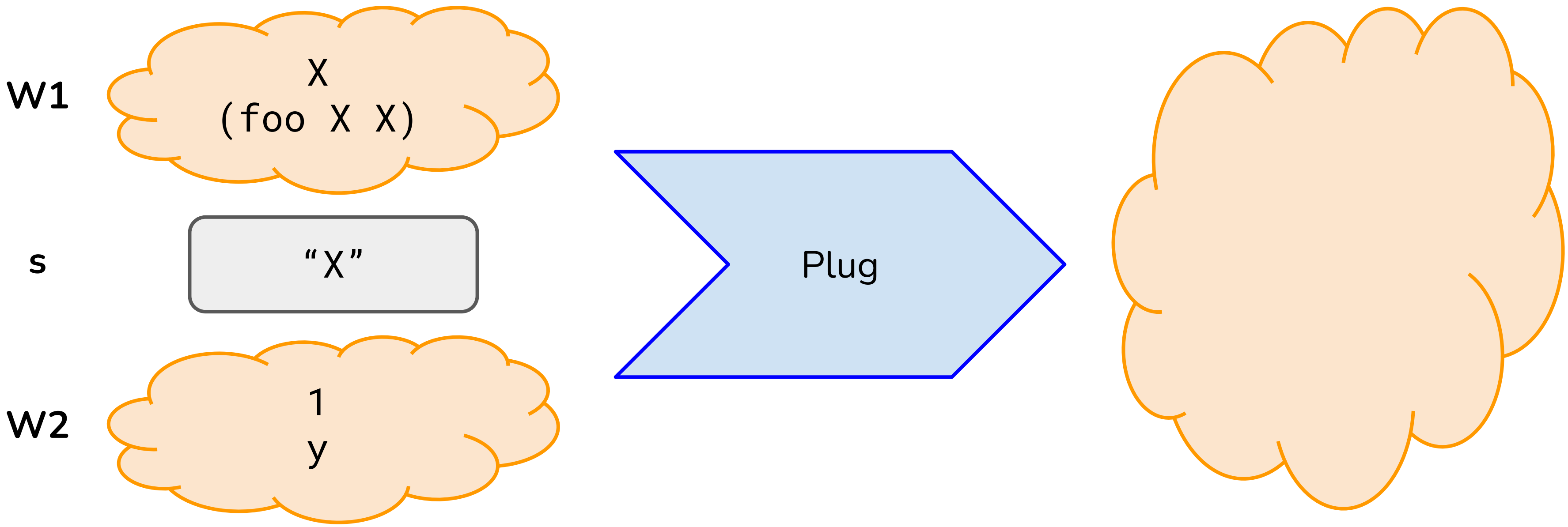
```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug("EXPR", exprs)
```

Enumo DSL

$$\text{Plug } \mathcal{W}_1 \; s \; \mathcal{W}_2$$

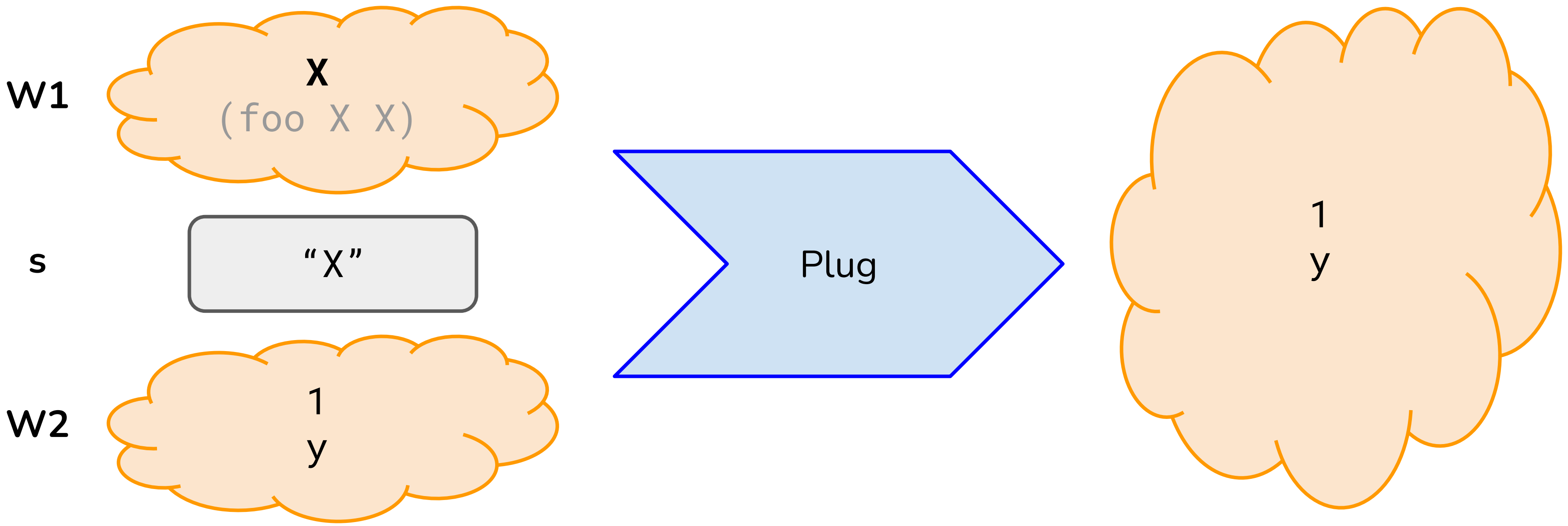
All combinations of replacing s in W_1 with a term from W_2

ENUMO DSL



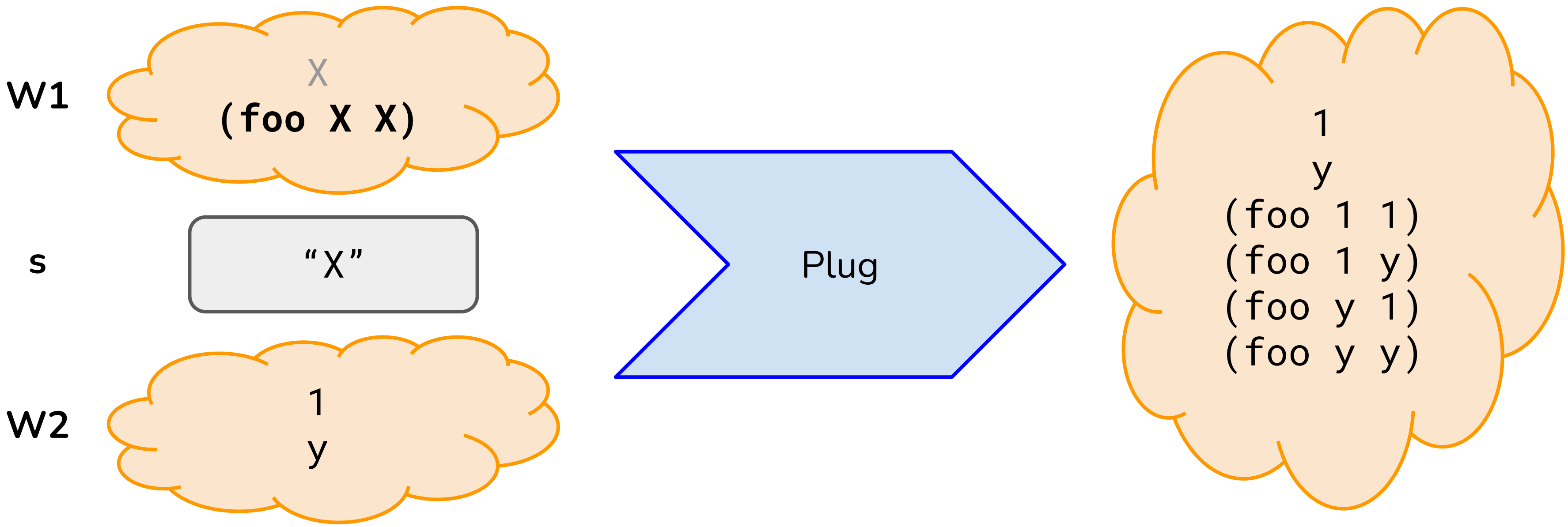
Duregård et al. 2012

ENUMO DSL



Duregård et al. 2012

ENUMO DSL



Duregård et al. 2012

ENUMO DSL

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

ENUMO DSL

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

ENUMO DSL

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

ENUMO DSL

```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug(“EXPR”, exprs)
            .plug(“LIT”, lits)
```

ENUMO DSL

```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug("EXPR", exprs)
            .plug("LIT", lits)
```

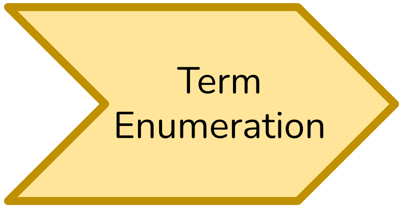
```
a                (~ (+ b b))
(~ a)            (~ (+ a b))
(~ (~ a))        (+ b b)
(~ (+ a a))      c
(+ a a)          (~ c)
b                (~ (~ c))
(~ b)            (~ (+ a c))
(~ (~ b))        ...
```

ENUMO DSL

```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug(“EXPR”, exprs)
            .plug(“LIT”, lits)

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

ENUMO DSL

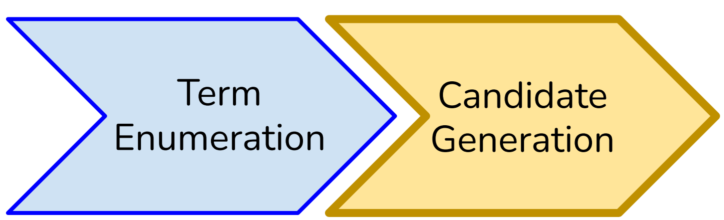
```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug("EXPR", exprs)
            .plug("LIT", lits)

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



Nandi et al. 2021

ENUMO DSL

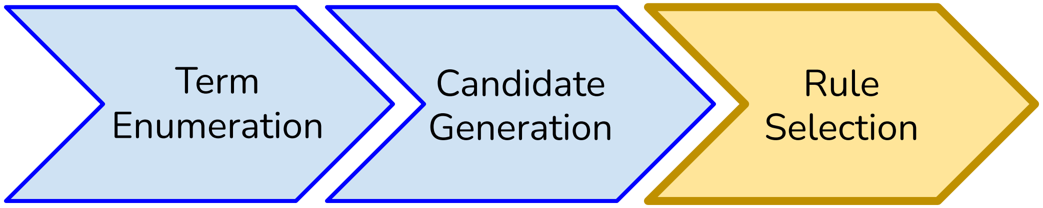
```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug("EXPR", exprs)
            .plug("LIT", lits)

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



Nandi et al. 2021

ENUMO DSL

```
lits  = Workload { a b c 0 1 }
exprs = Workload { LIT (~ EXPR) (+ EXPR EXPR) }

wkld = exprs.plug("EXPR", exprs)
            .plug("LIT", lits)

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



Nandi et al. 2021

Enumo DSL

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

ENUMO DSL

```
e1 = Workload { (~ EXPR) (+ EXPR EXPR) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXPR", 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))
```

ENUMO DSL

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

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

ENUMO DSL

```
e1 = Workload { (~ EXPR) (+ EXPR EXPR) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXPR", 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))
```

ENUMO DSL

```
e1 = Workload { (~ EXPR) (+ EXPR EXPR) }
e2 = Workload { 1 (+ 2 3) (+ (+ 4 5) 6) }
e1.plug("EXPR", 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))~~

Enumo DSL

Optimization: Pushing Filters through Plugs

ENUMO DSL

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

Enumo DSL

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

Requires monotonicity of f

ENUMO DSL

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

ENUMO DSL

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

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

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

ENUMO DSL

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

Monotonic

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

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

ENUMO DSL

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

Monotonic

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

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

Not monotonic

1. Traditional theory exploration

2. The ENUMO DSL

3. Novel scalable rule-finding strategies

Comparison to Ruler

Domain	ENUMO LOC	# ENUMO	# Ruler	ENUMO → Ruler	Ruler → ENUMO
bool	44	64	51	100%	87.5%
bv4	21	180	84	100%	38.3%
bv32	20	120	78	100%	58.3%
rational	51	131	113	100%	62.6%

Comparison to Ruler

Domain	Enumo LOC	# Enumo	# Ruler	Enumo → Ruler	Ruler → Enumo
bool	44	64	51	100%	87.5%
bv4	21	180	84	100%	38.3%
bv32	20	120	78	100%	58.3%
rational	51	131	113	100%	62.6%

Check out the paper for more about derivability!
(TLDR: More rules are not always better)

Large Grammars: Halide-Inspired Case Study

```
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)
}
```

Ragan-Kelley et al. 2013

Large Grammars: Halide-Inspired Case Study

```
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, unsupported operators, or unbound variables

Term Size	# Rules	ENUMO → Halide

Ragan-Kelley et al. 2013

Large Grammars: Halide-Inspired Case Study

```
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

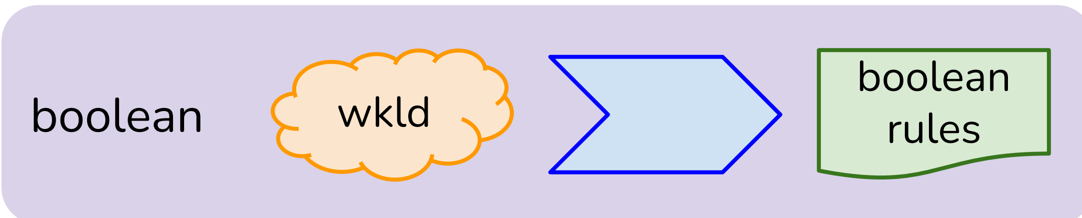
Large Grammars: Halide-Inspired Case Study

```
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)
}
```

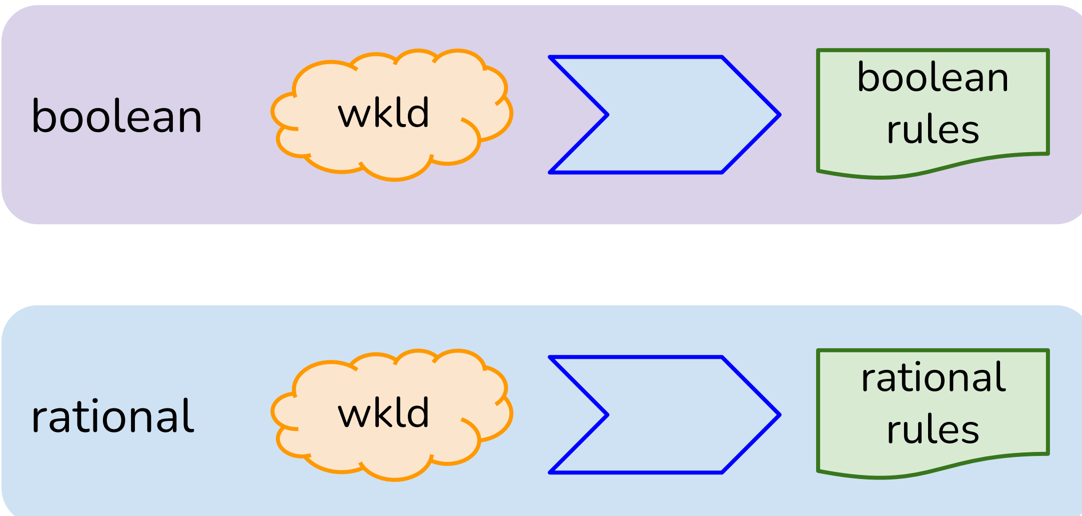
Domain expert know which operators are closely related

Ragan-Kelley et al. 2013

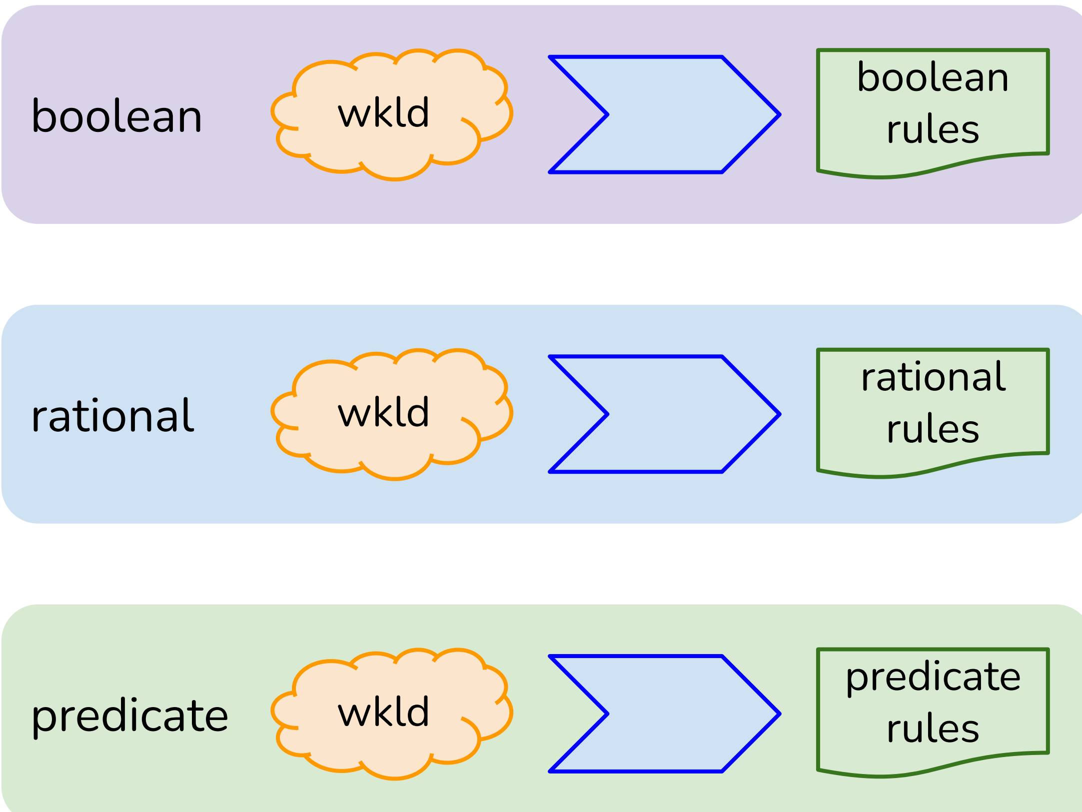
Large Grammars: Halide-Inspired Case Study



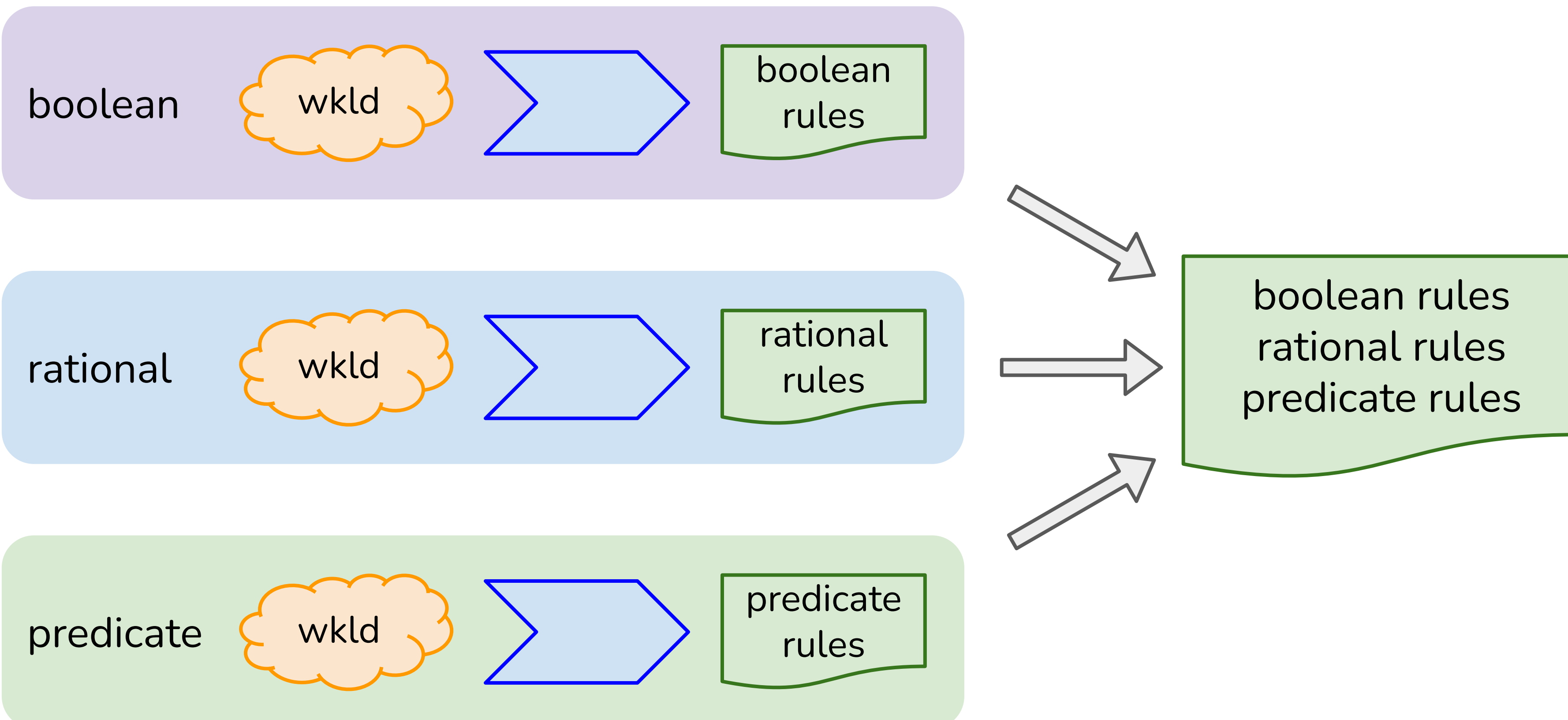
Large Grammars: Halide-Inspired Case Study



Large Grammars: Halide-Inspired Case Study



Large Grammars: Halide-Inspired Case Study



Large Grammars: Halide-Inspired Case Study

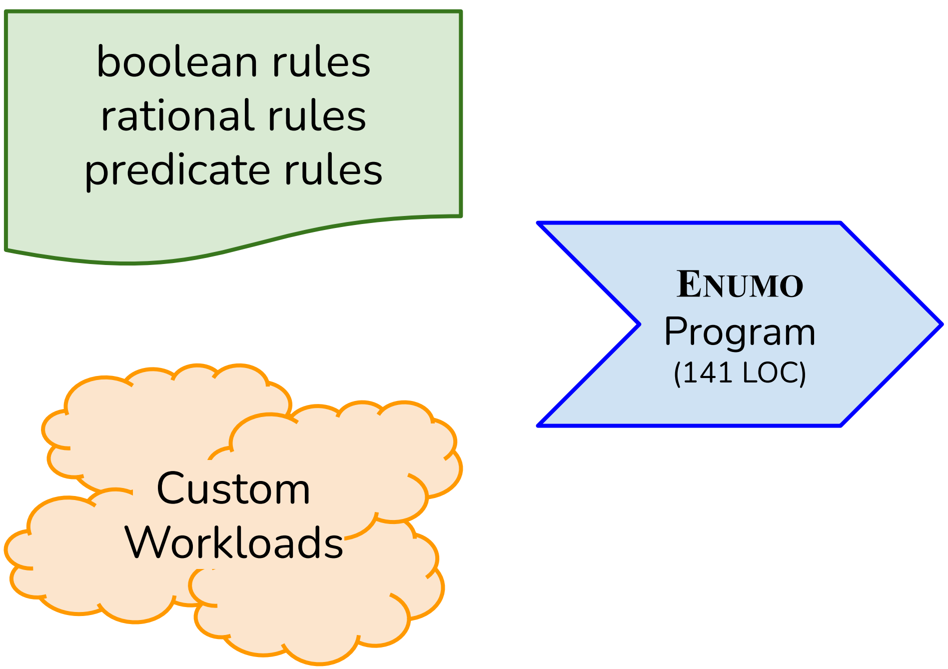
boolean rules
rational rules
predicate rules

Large Grammars: Halide-Inspired Case Study

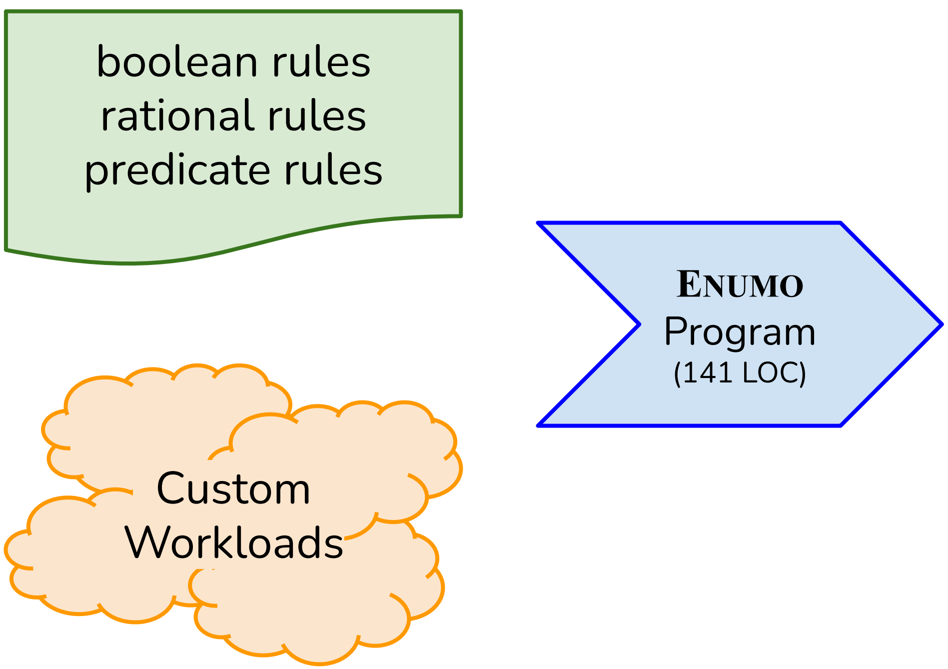
boolean rules
rational rules
predicate rules

Custom
Workloads

Large Grammars: Halide-Inspired Case Study



Large Grammars: Halide-Inspired Case Study



Term Size	# Rules	ENUMO → Halide
Custom	845	90.6%

Large Grammars: Halide-Inspired Case Study

Guided search enables progress past exponential blowup

ENUMO enables new rule inference strategies

ENUMO enables new rule inference strategies

Fast-forwarding is a phased approach for finding "shortcut" rules

ENUMO enables new rule inference strategies

Implemented with <10 lines of ENUMO

Fast-forwarding is a phased approach for finding “shortcut” rules

Doesn’t require an interpreter!

Fast-forwarding

```
def fast_forward(wkld, R, E):
  G = wkld.to_egraph()
  allowed = {r ∈ R | r.is_allowed()}
  G’   = G.compress(allowed, limits)
  G’’  = G’.eqsat(E, limits)
  C    = by_diff(G’, G’’)
  G’’’ = G’’.compress(R, limits)
  C.union(by_diff(G’’, G’’’))
  return C.select_rules(allowed, limits)
```

Herbie: Improve floating point accuracy by rewriting

Carefully crafted rewrite rules over real numbers

$$a + b \rightsquigarrow \frac{a^2 - b^2}{a - b}$$

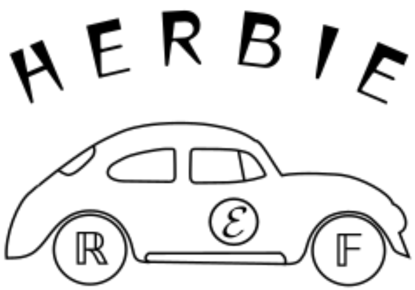
$$\cos^2(a) + \sin^2(a) \rightsquigarrow 1$$

Panchekha et al. 2015

Herbie: Improve floating point accuracy by rewriting

Carefully crafted rewrite rules over real numbers

Learnable with traditional techniques



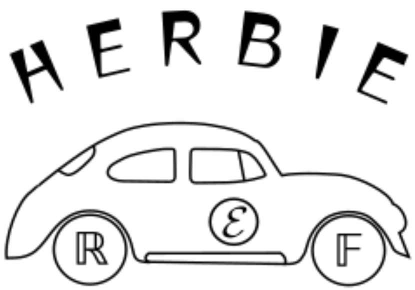
$$a + b \rightsquigarrow \frac{a^2 - b^2}{a - b}$$

$$\cos^2(a) + \sin^2(a) \rightsquigarrow 1$$

Panchekha et al. 2015

Herbie: Improve floating point accuracy by rewriting

Carefully crafted rewrite rules over real numbers



$$a + b \rightsquigarrow \frac{a^2 - b^2}{a - b}$$

Learnable with traditional techniques

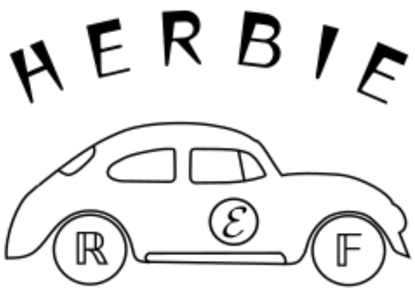
$$\cos^2(a) + \sin^2(a) \rightsquigarrow 1$$

No interpreter ⇒ Can't learn with traditional techniques

Panchekha et al. 2015, Boehm et al. 2020

Herbie: Improve floating point accuracy by rewriting

Carefully crafted rewrite rules over real numbers



Learnable with traditional techniques

$$a + b \rightsquigarrow \frac{a^2 - b^2}{a - b}$$

No interpreter ⇒ Can't learn with traditional techniques

$$\text{cos}^2(a) + \text{sin}^2(a) \rightsquigarrow 1$$

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

$$\text{sin}(x) \rightsquigarrow \frac{\text{cis}(x) - \text{cis}(-x)}{2i}$$

Panchekha et al. 2015, Boehm et al. 2020

Herbie: Improve floating point accuracy by rewriting

Carefully crafted rewrite rules over real numbers

Learnable with traditional techniques

$$a + b \rightsquigarrow \frac{a^2 - b^2}{a - b}$$

No interpreter ⇒ Can't learn with traditional techniques

$$\cos^2(a) + \sin^2(a) \rightsquigarrow 1$$

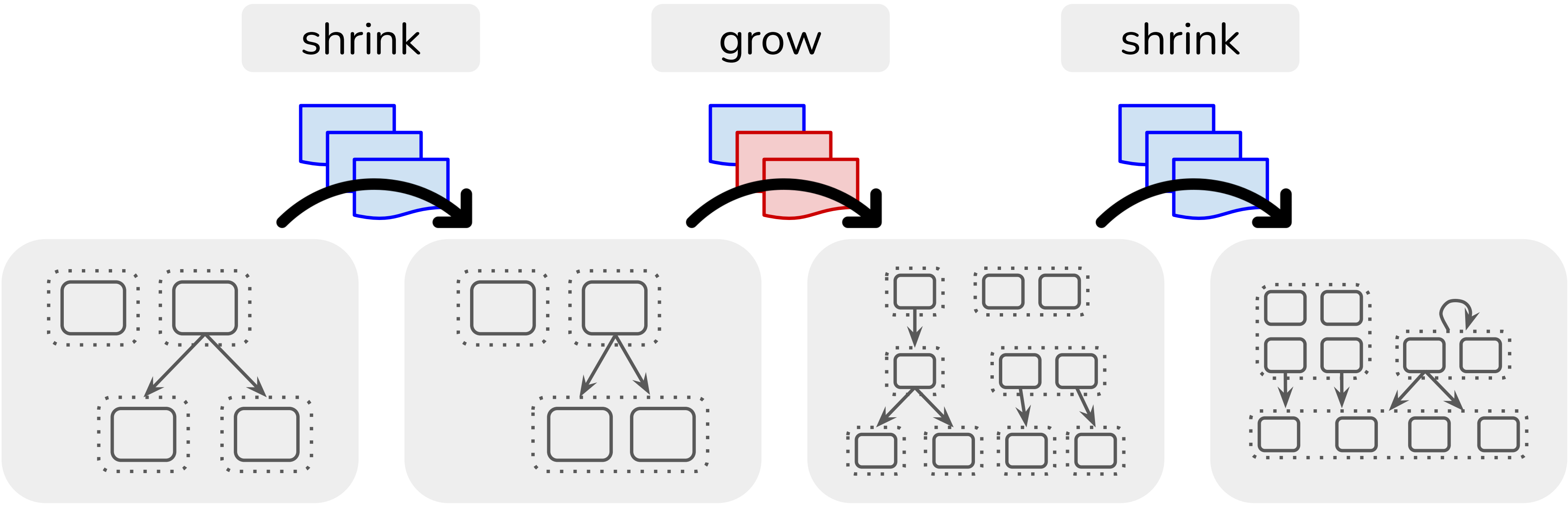
$$\text{cis}(x) = \cos(x) + i \sin(x)$$

$$\sin(x) \rightsquigarrow \frac{\text{cis}(x) - \text{cis}(-x)}{2i}$$

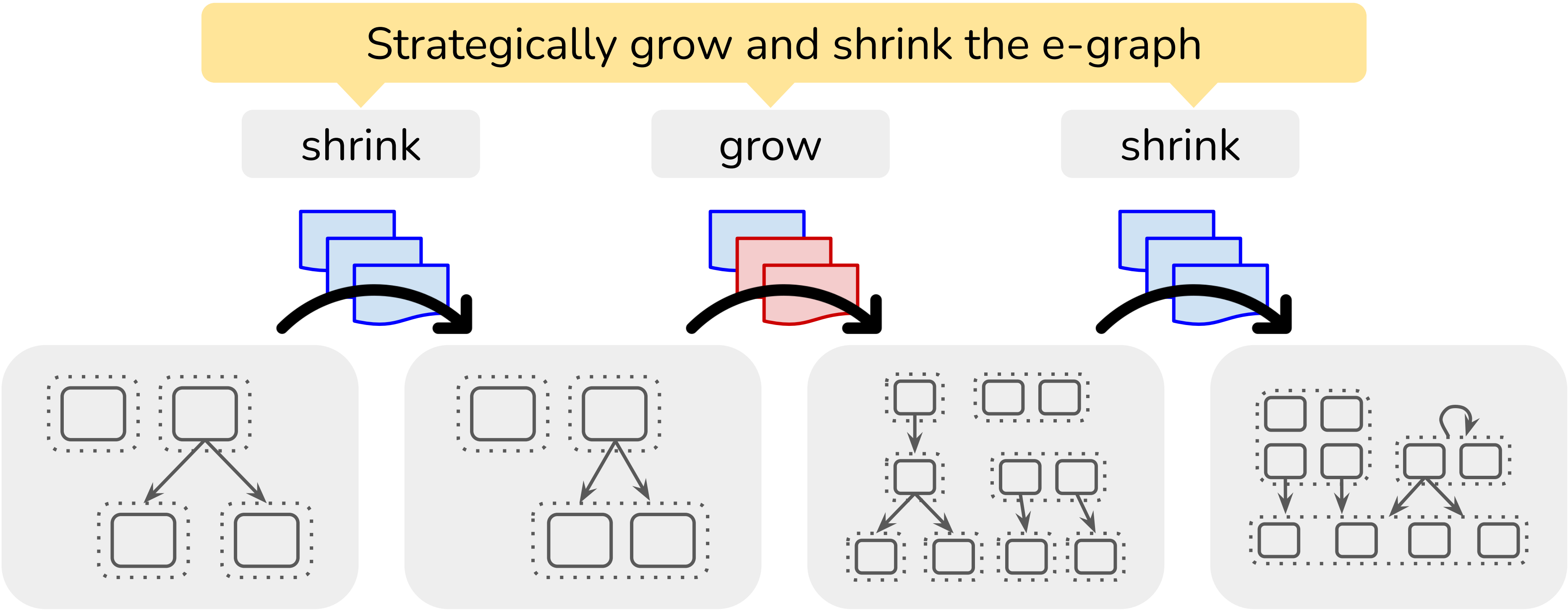
Rewriting through complex terms is not feasible due to resource limits

Panchekha et al. 2015, Boehm et al. 2020

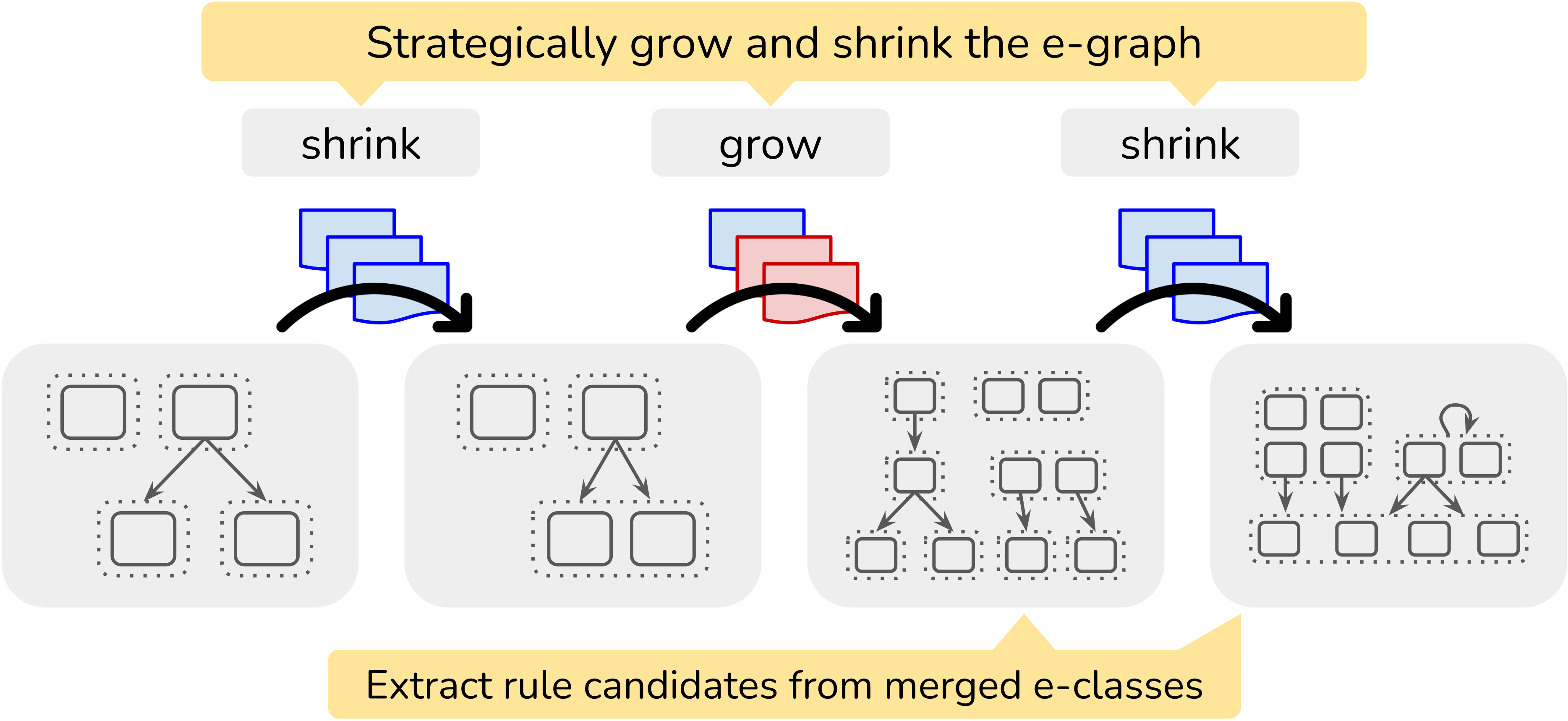
Fast-Forwarding



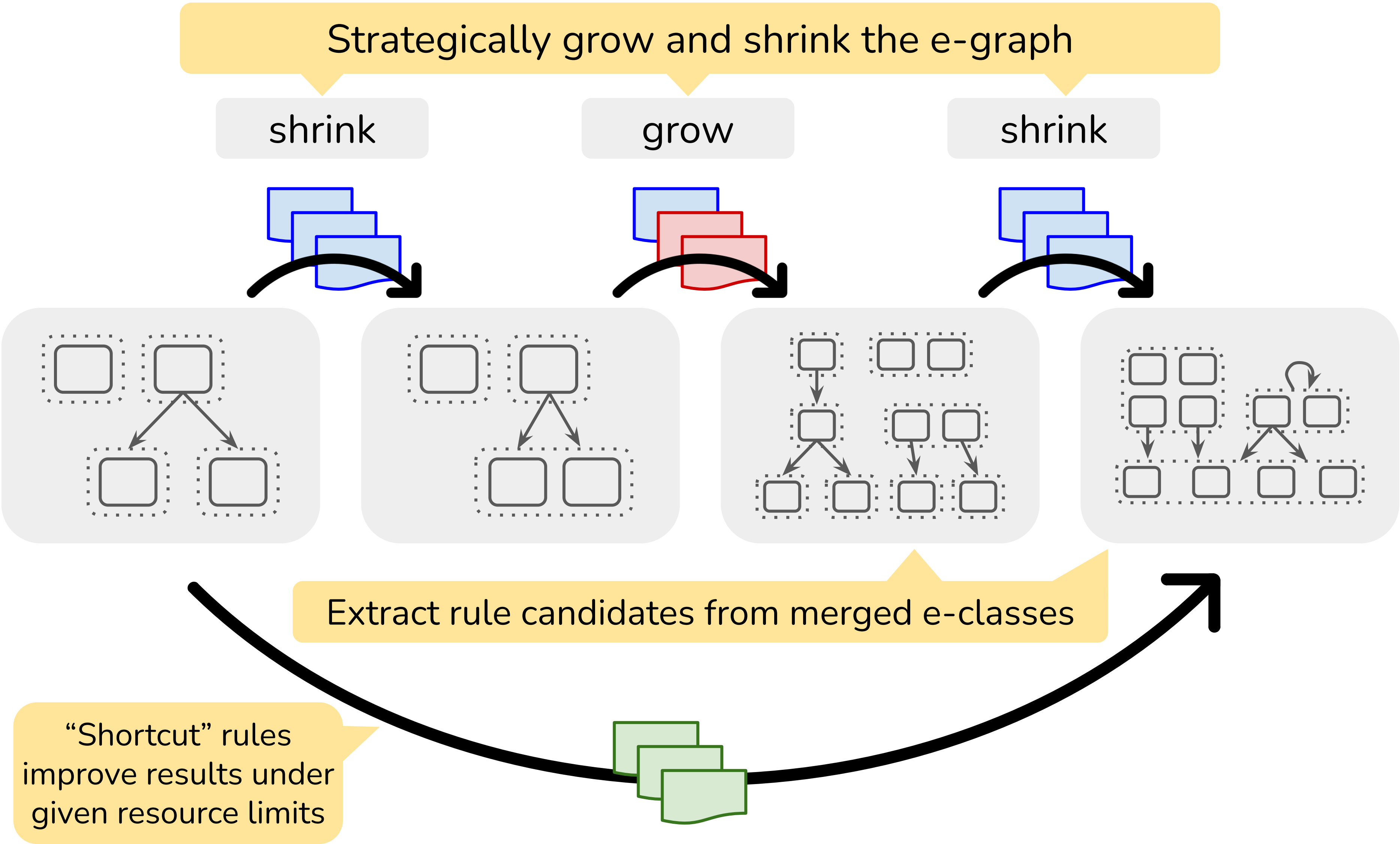
Fast-Forwarding



Fast-Forwarding



Fast-Forwarding



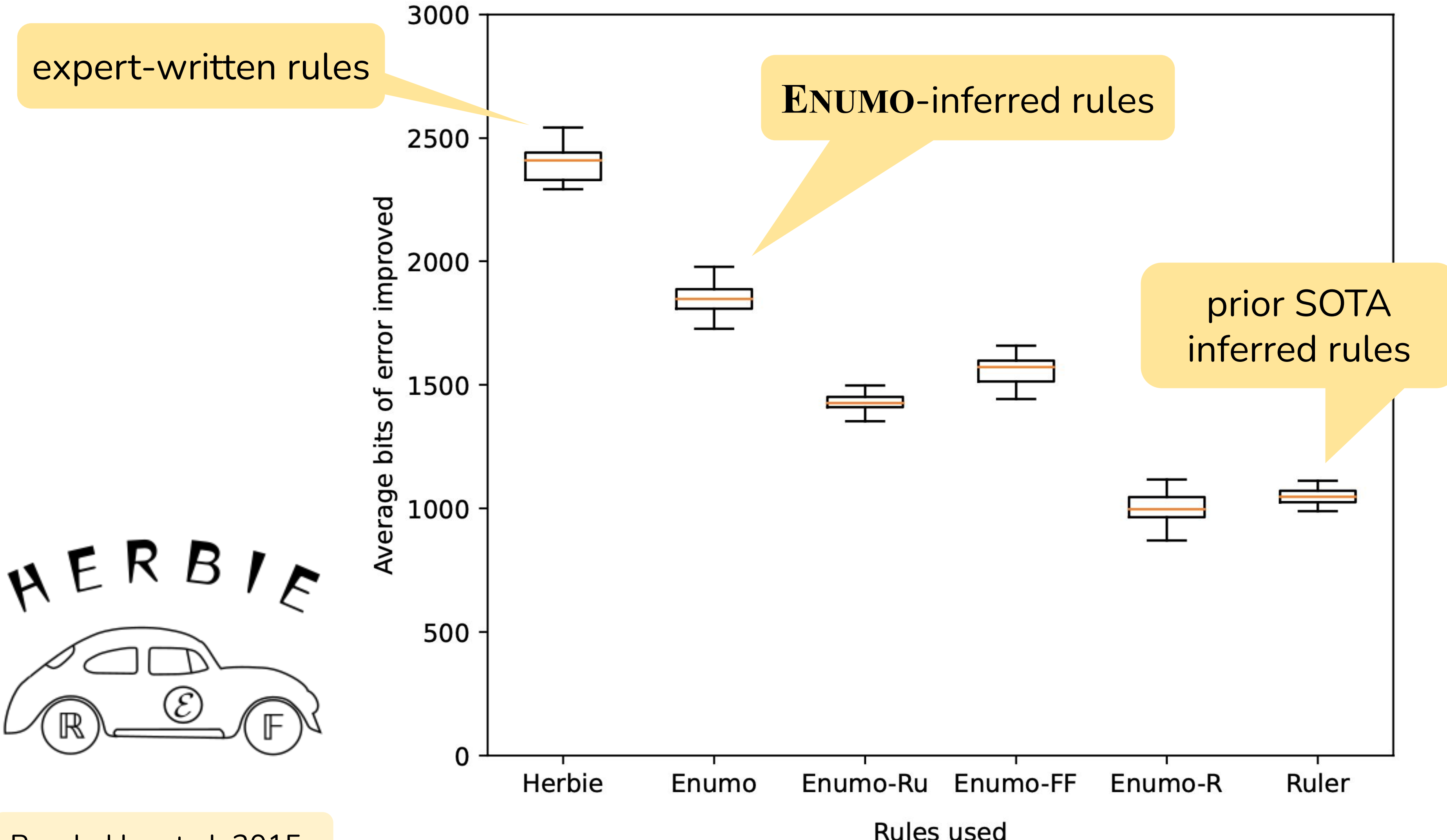
Fast-Forwarding

$$\sin(b+a) \rightsquigarrow \sin(b)\cdot\cos(a)+\sin(a)\cdot\cos(b)$$
$$\sin(b)\cdot\sin(a) \rightsquigarrow \frac{\cos(b-a)-\cos(b+a)}{2}$$

$$c^{ba} \rightsquigarrow (c^a)^b$$
$$(c^b)^{\log(a)} \rightsquigarrow (a^b)^{\log(c)}$$
$$\sqrt{b^a} \rightsquigarrow (\sqrt{b})^a$$

$$\text{Scale}(a,b,c,\text{Trans}(d,e,f,s)) \rightsquigarrow \text{Trans}(da,eb,fc,\text{Scale}(a,b,c,s))$$
$$\text{Cube}(ad,be,cf) \rightsquigarrow \text{Scale}(a,b,c,\text{Cube}(d,e,f))$$

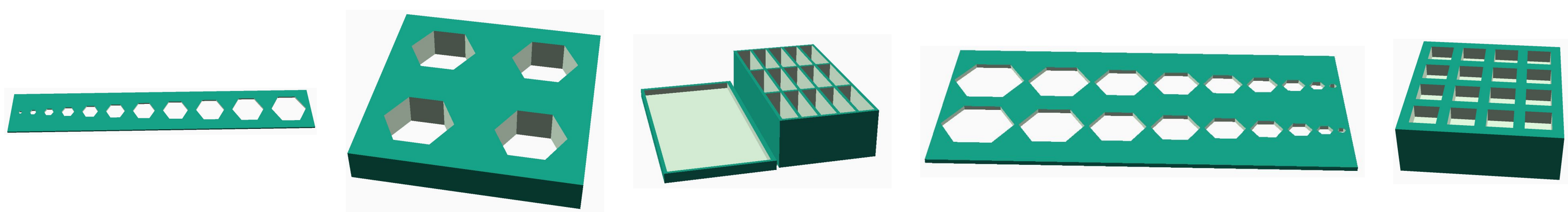
End-to-end: Herbie



Panchekha et al. 2015

End-to-end: Szalinski

Program Id	Percent AST Shrinkage: Handwritten Rules	Percent AST Shrinkage: ENUMO-synthesized Rules
TackleBox	91%	85%
SDCardRack	87%	86%
SingleRowHolder	92%	92%
CircleCell	80%	80%
CNCBitCase	89%	89%
CassetteStorage	89%	89%
RaspberryPiCover	96%	96%
ChargingStation	89%	82%
CardFramer	76%	76%
HexWrenchHolder	95%	84%



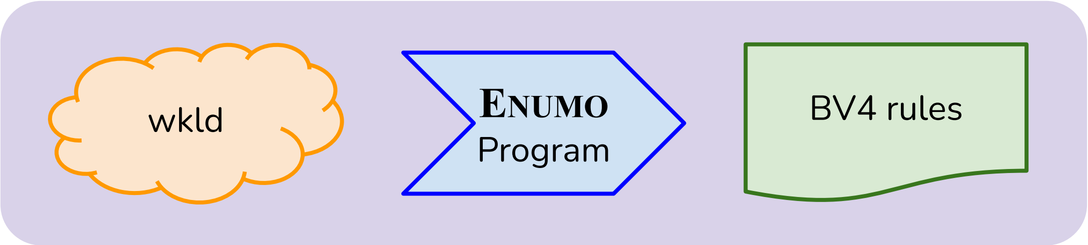
Nandi et al. 2020

Porting rules across domains

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

Porting rules across domains

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



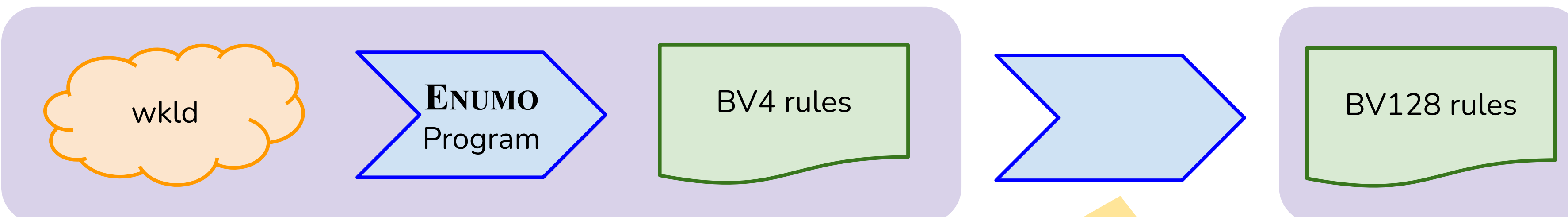
Porting rules across domains

BV 4

✅ Fast to synthesize

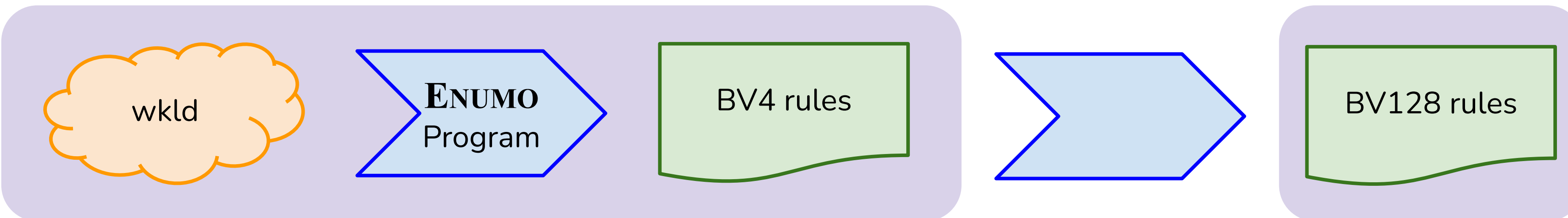
BV 128

❌ Slow to synthesize



Cast rules from BV4 to BV128 and re-validate

Porting rules across domains



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

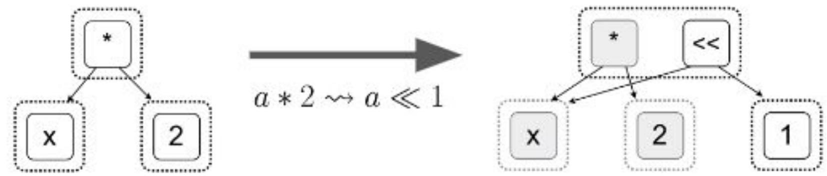
Equality Saturation Theory Exploration à la Carte

Synthesize better rewrite rulesets!

Anjali Pal, Brett Saiki, **Ryan Tjoa***, Cynthia Richey*, Amy Zhu, Oliver Flatt, Max Willsey, Zachary Tatlock, Chandrakana Nandi

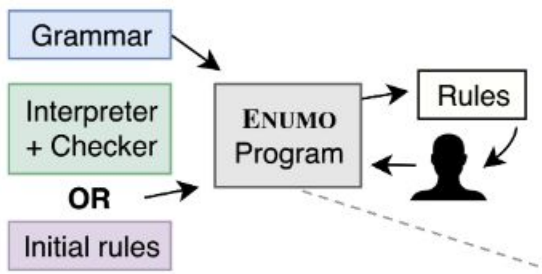
Equality saturation uses rewrite rules for program optimization, verification, synthesis, etc.

How do we discover rewrite rules?



ENUMO: *a theory exploration DSL*

Infer rules *incrementally* using *composable* search operators, even *without an interpreter*.




```
lits = {a b c 0 1}
G = {LIT (~ EXPR) (+ EXPR EXPR)}

wkld = G.plug("EXPR", G)
        .plug("LIT", lits)
        .filter(λt. t.size < 4)

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

Metric for proving power; see §4.3

Derivability vs. **Ruler** (prior SOTA) for common theories:

Domain	ENUMO → Ruler	Ruler → ENUMO
bool	100%	87.5%
bv4	100%	38.3%
bv32	100%	58.3%
rational	100%	62.6%

Evaluation & Case Studies

Herbie: 35% higher accuracy than with Ruler

$$a + b \rightsquigarrow \frac{a \cdot a - b \cdot b}{a - b}$$

$$\cos(b + a) \rightsquigarrow \cos b \cdot \cos a - \sin b \cdot \sin a$$

$$\left(c^b\right)^{\log a} \rightsquigarrow \left(a^b\right)^{\log c}$$

Szalinski: shrink CAD programs by 87% (expert-written identities shrink by 90%)

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

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

PLSE
W PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

https://uwplse.org/ruler/

Composable Operators

Incremental Ruleset Building

Enumo: A DSL for Programmable Theory Exploration

Leverage Domain Expertise

New Inference Techniques

ENUMO DSL

Monotonic

- `Excludes`
- `MetricLt`
- `And`

Not Monotonic

- `Contains`
- `MetricEq`
- `Or`
- `Not`
- `Canon`