Rewrite Rule Inference Using Equality Saturation

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki,
Adam Anderson, Adriana Schulz, Dan Grossman, Zachary Tatlock

OOPSLA 2021

PLSE

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Rewrite Rules Are Ubiquitous!

- Compilers
- Program Synthesizers
- Simplifiers / Optimizers
- SMT Solvers
- ML Frameworks

Rewrite Engines must be Efficient and Reliable!

Compilers

Program Synthesizers

Simplifiers / Optimizers

SMT Solvers

ML Frameworks

Performance and reliability are key for a TRS [Newcomb et al. OOPSLA'20]

But…Designing Rewrite Rules is still Hard!

Who *writes* the *rewrite* rules?

Typically hand written by experts

Time consuming, often takes years

Too few / too many rules

Unsound rules

…ility are key for a
TRS [Newcomb et al. OOPSLA'20]

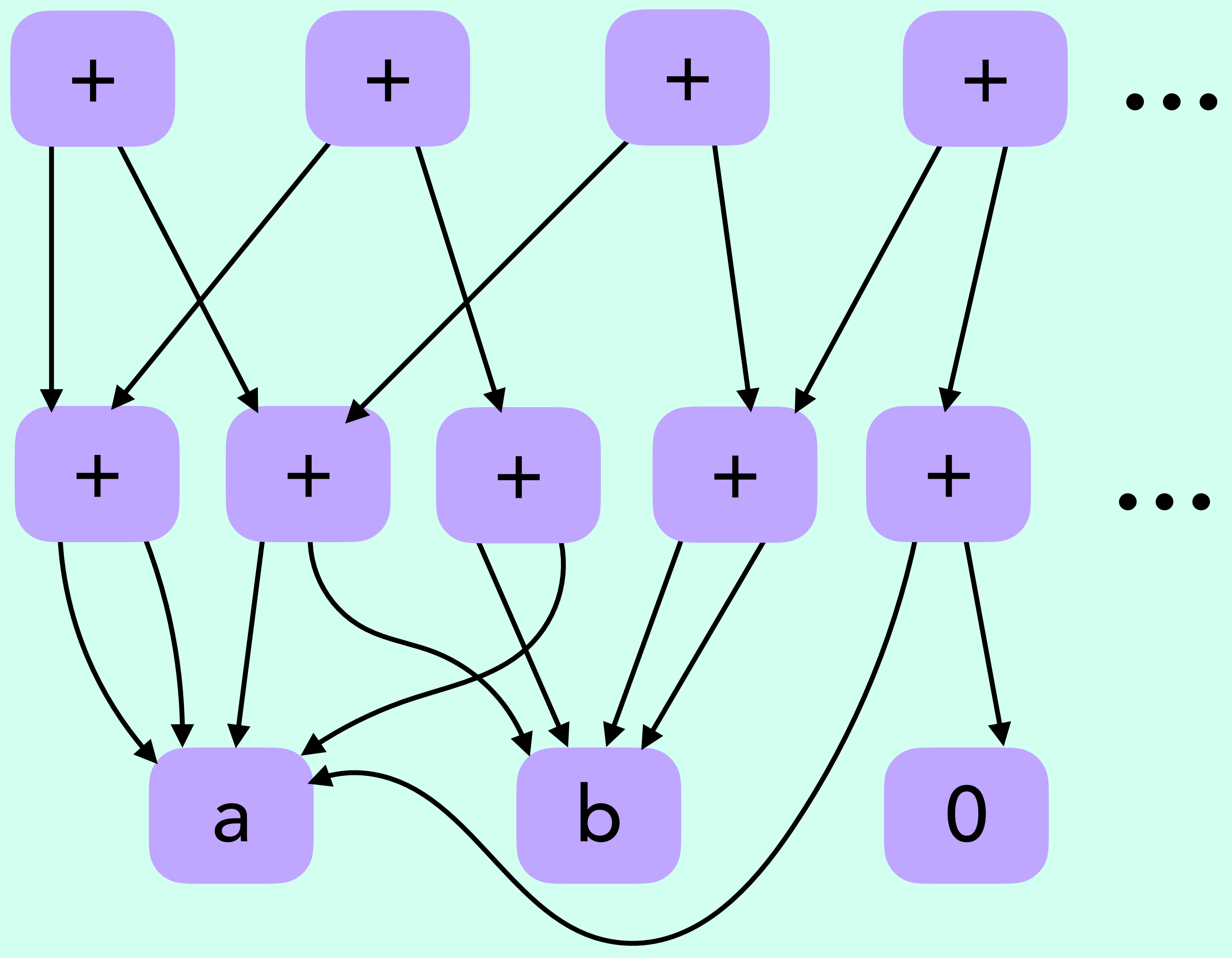
A 3-Step Approach for Inferring Rewrite Rules

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, +, …

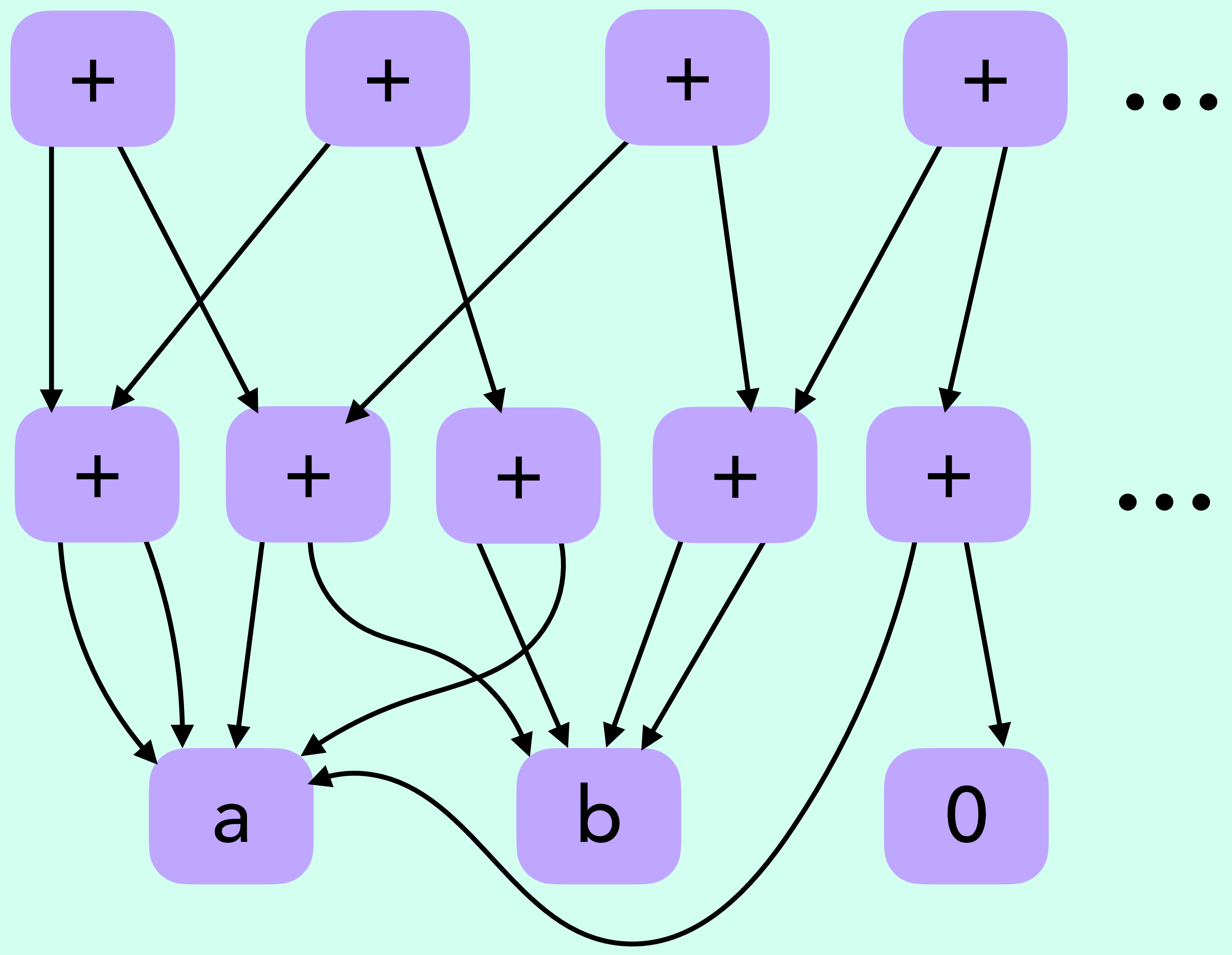


Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-Step Approach for Inferring Rewrite Rules

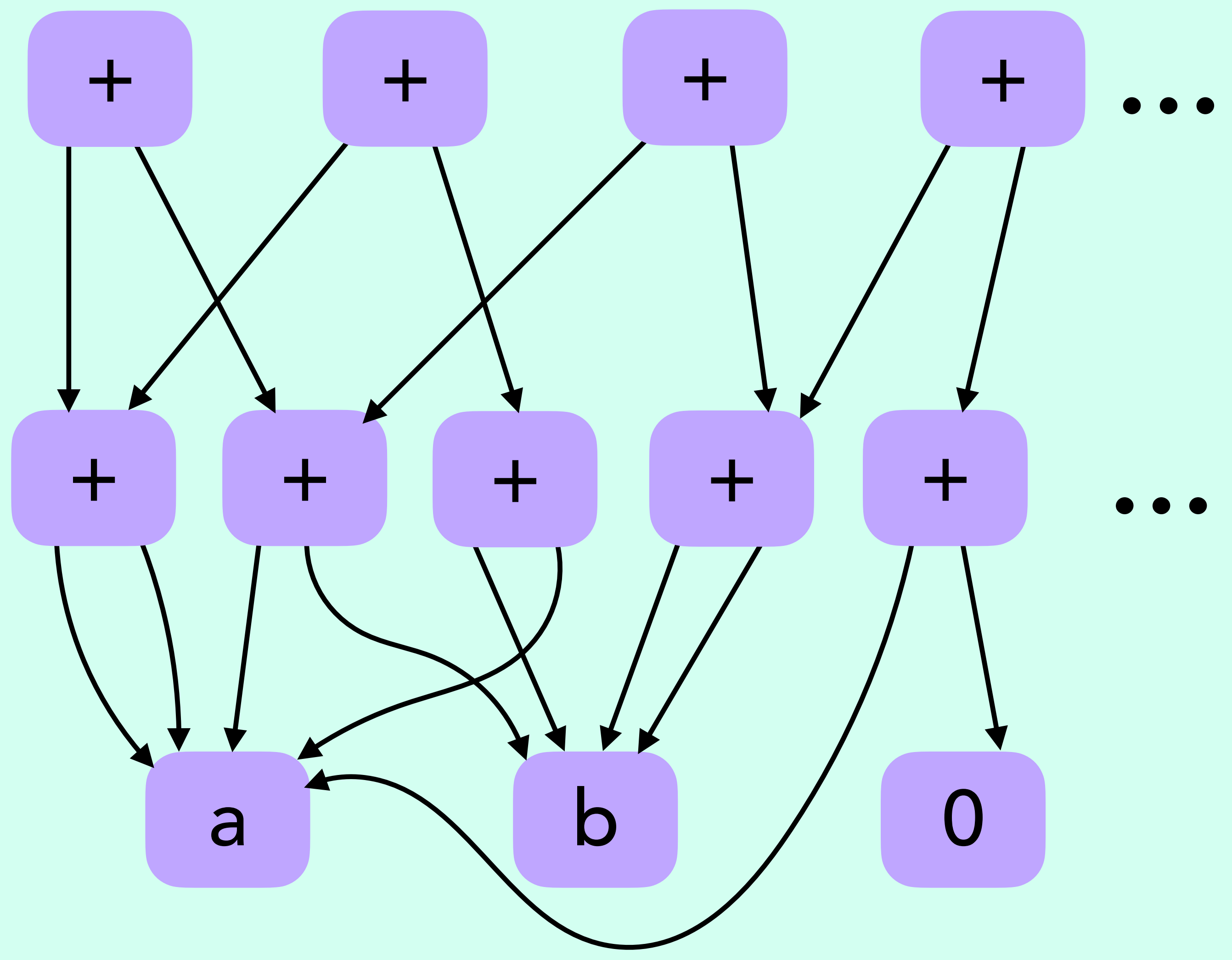
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

"Fingerprints"

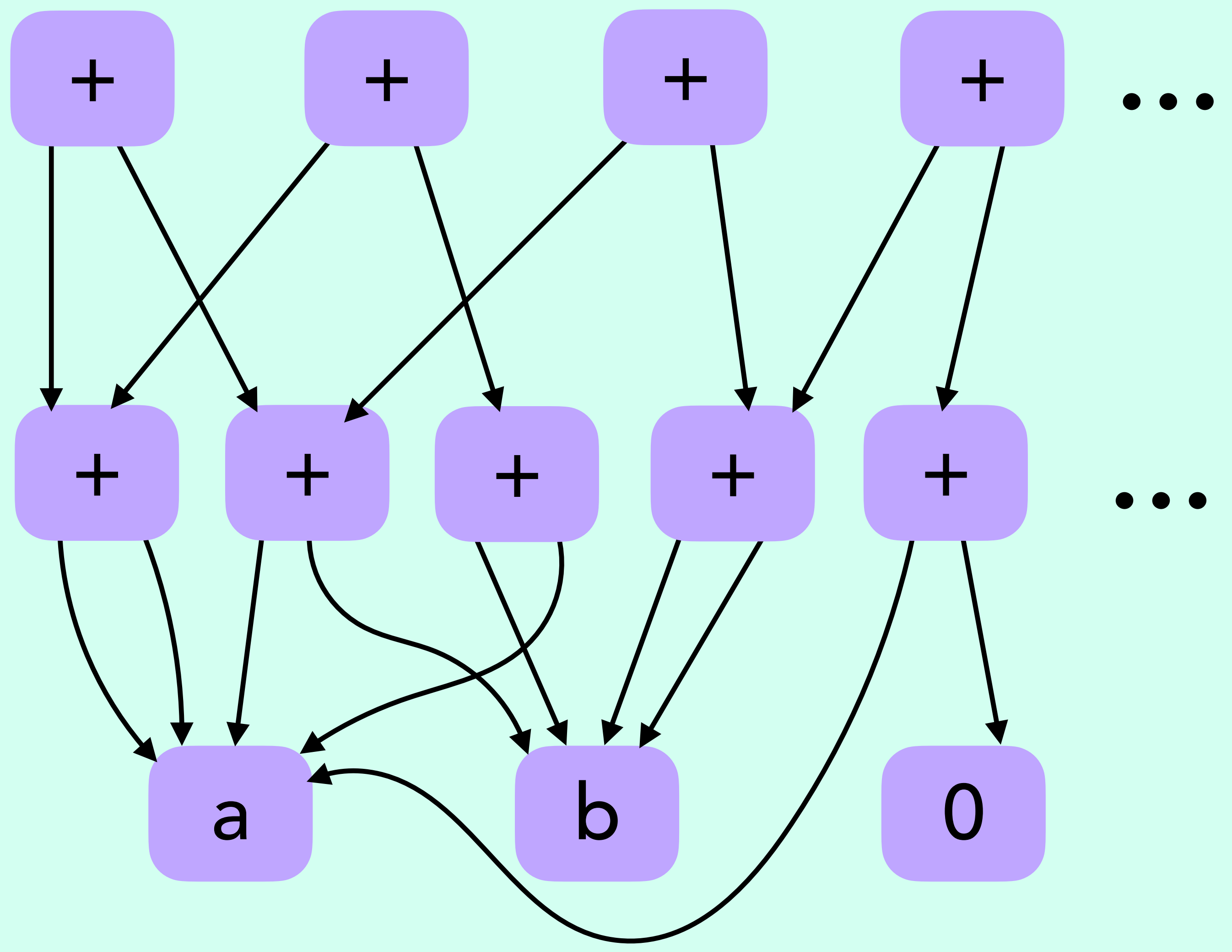


Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, …

A 3-Step Approach for Inferring Rewrite Rules

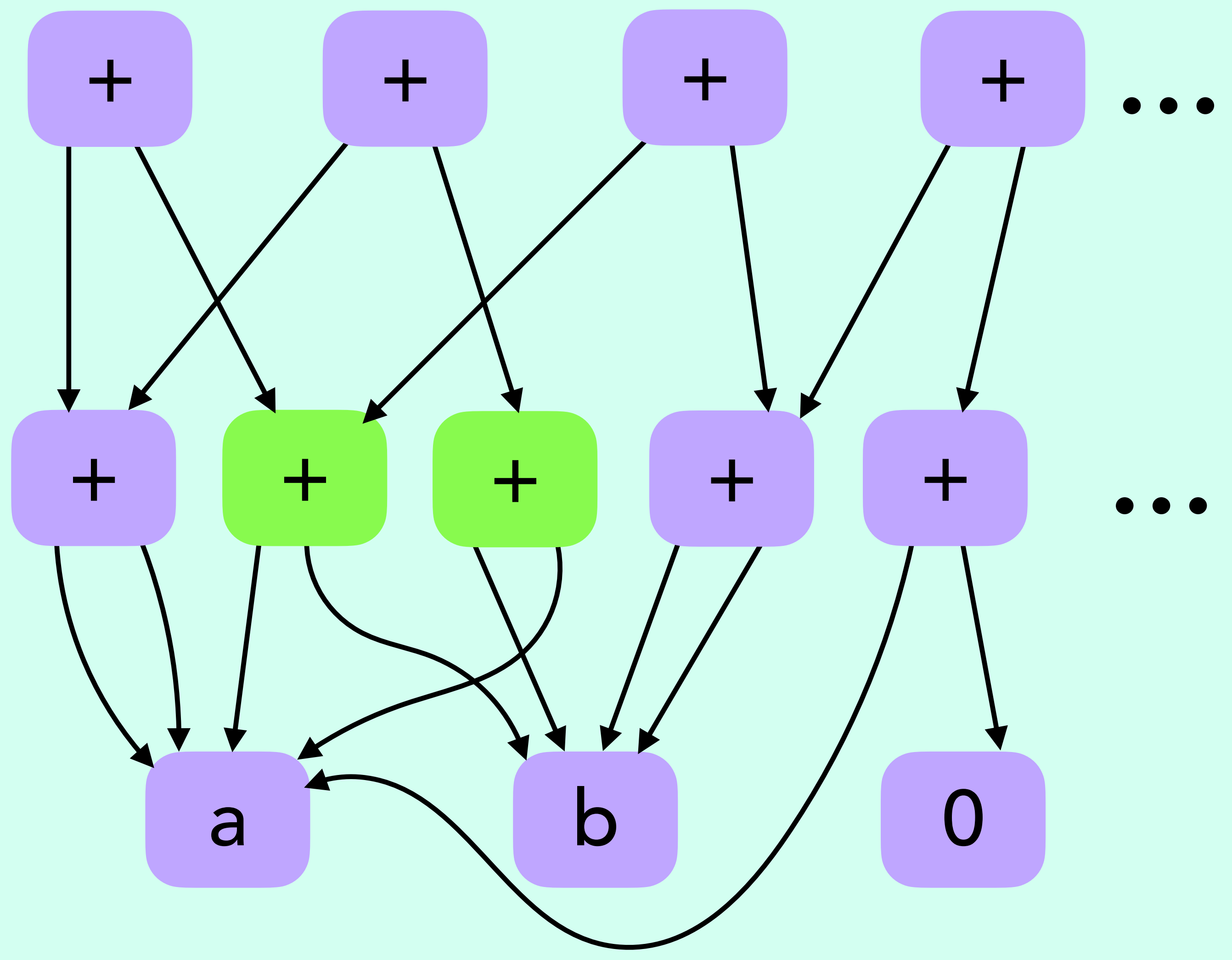
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”



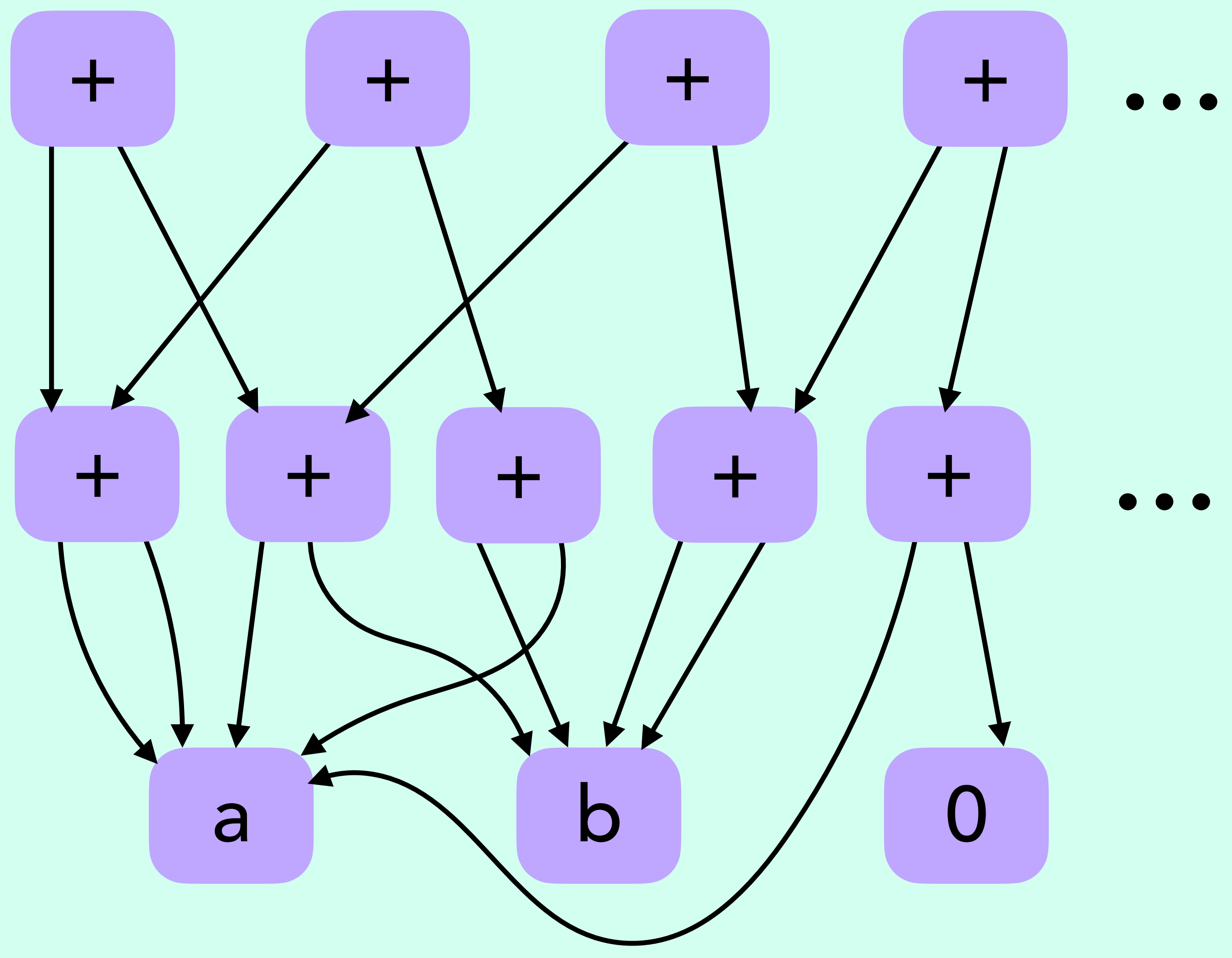
(x + y) ⟷ (y + x)

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

A 3-Step Approach for Inferring Rewrite Rules

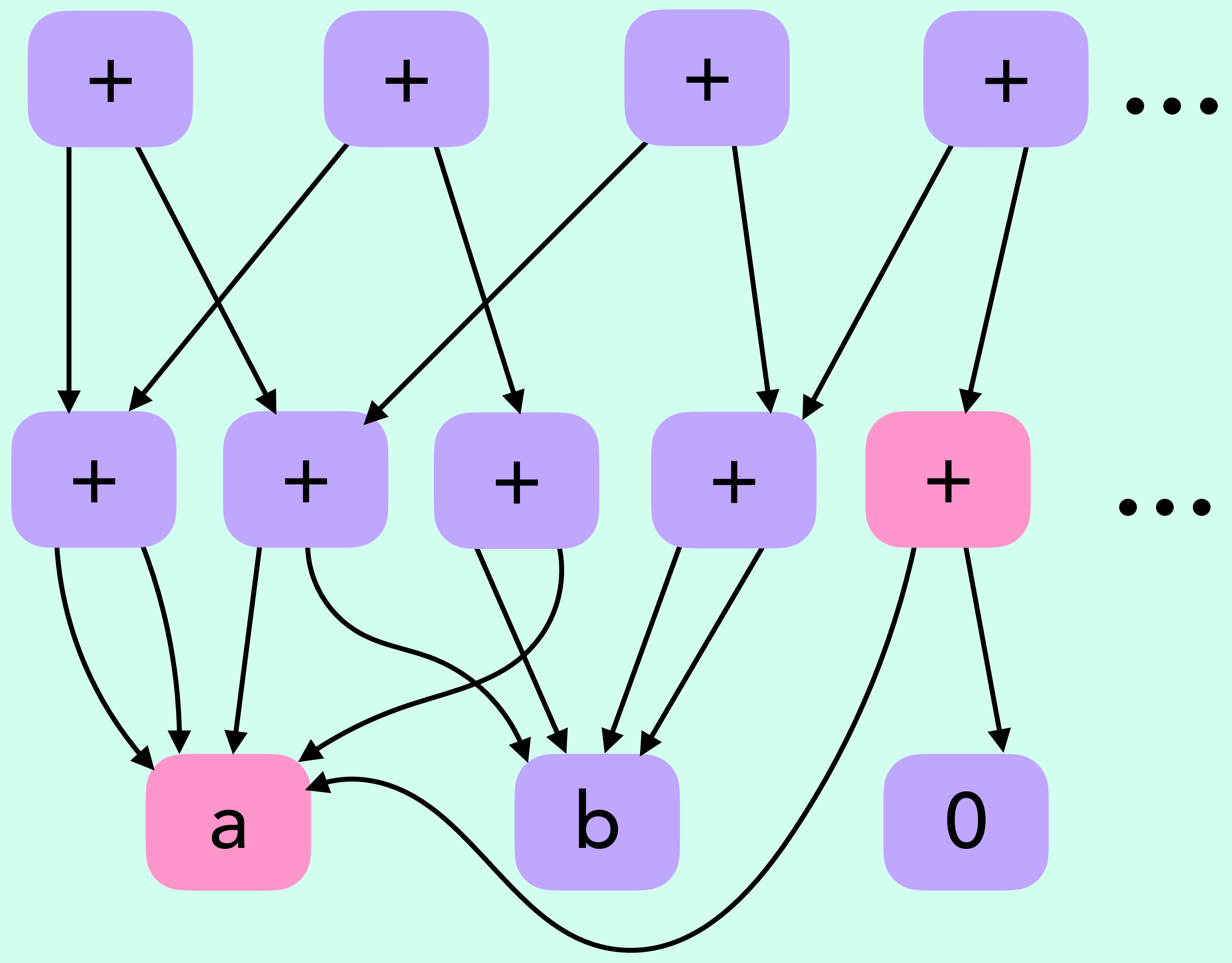
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

"Fingerprints"



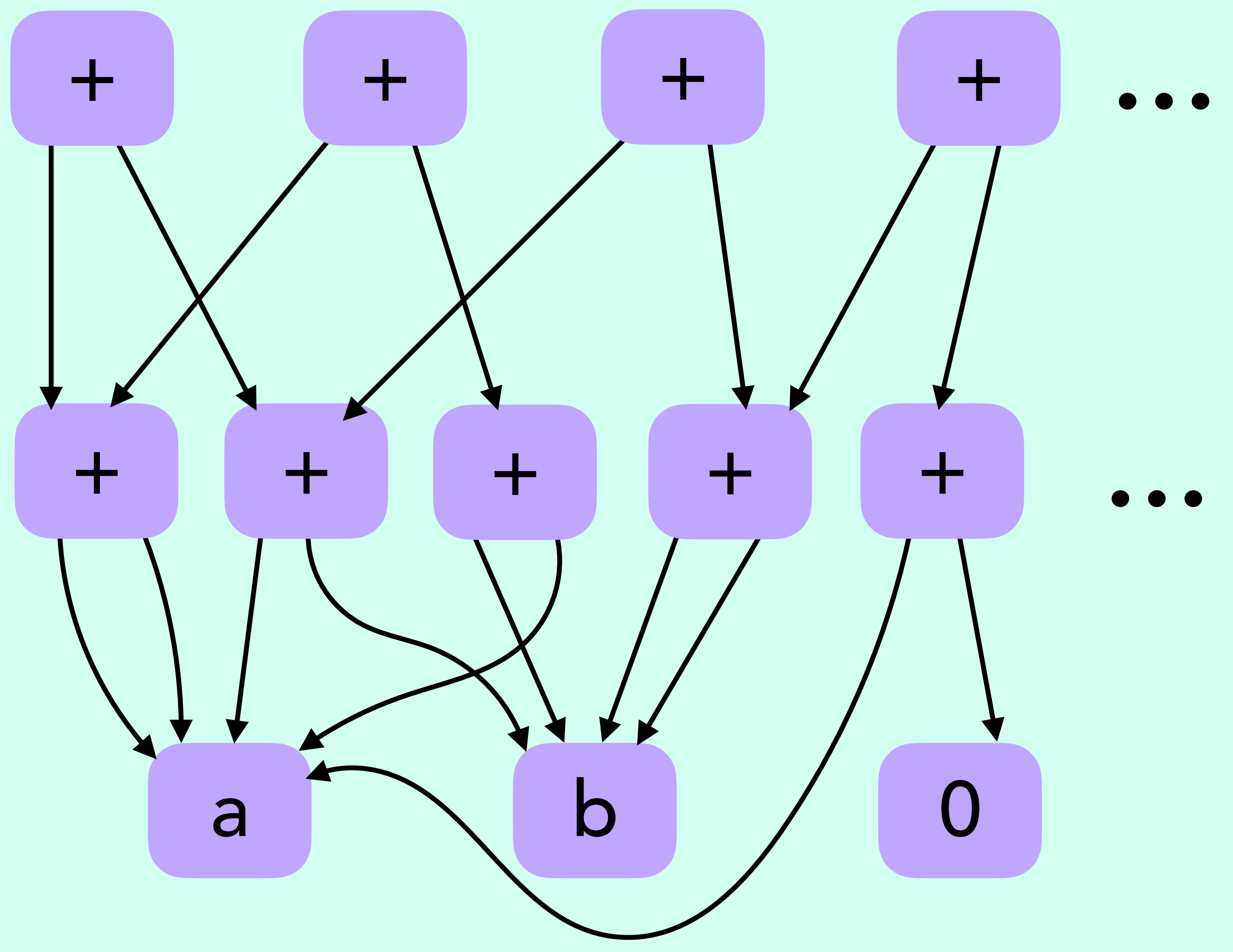
(x + 0) ⟷ x

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

A 3-Step Approach for Inferring Rewrite Rules

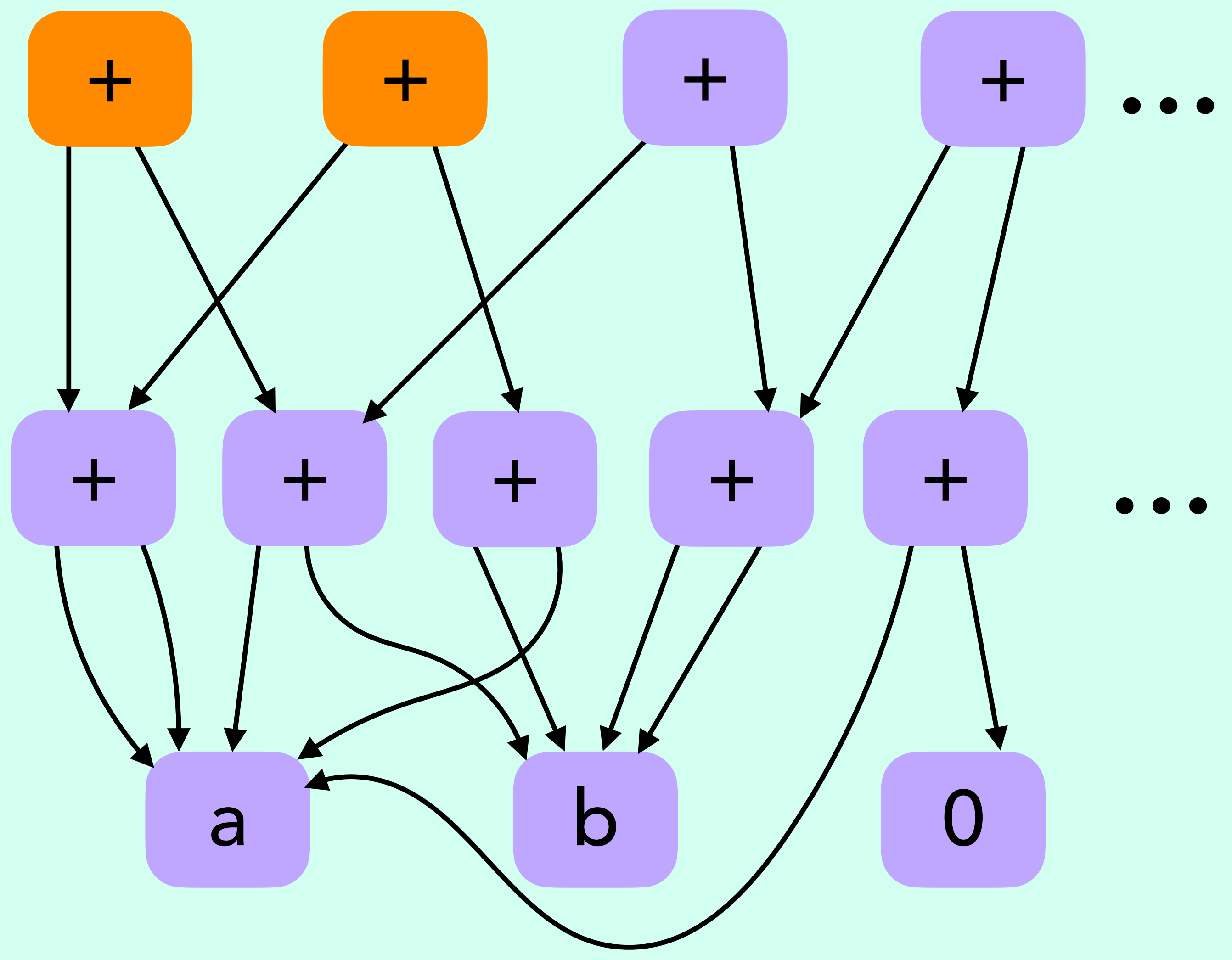
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

“Fingerprints”



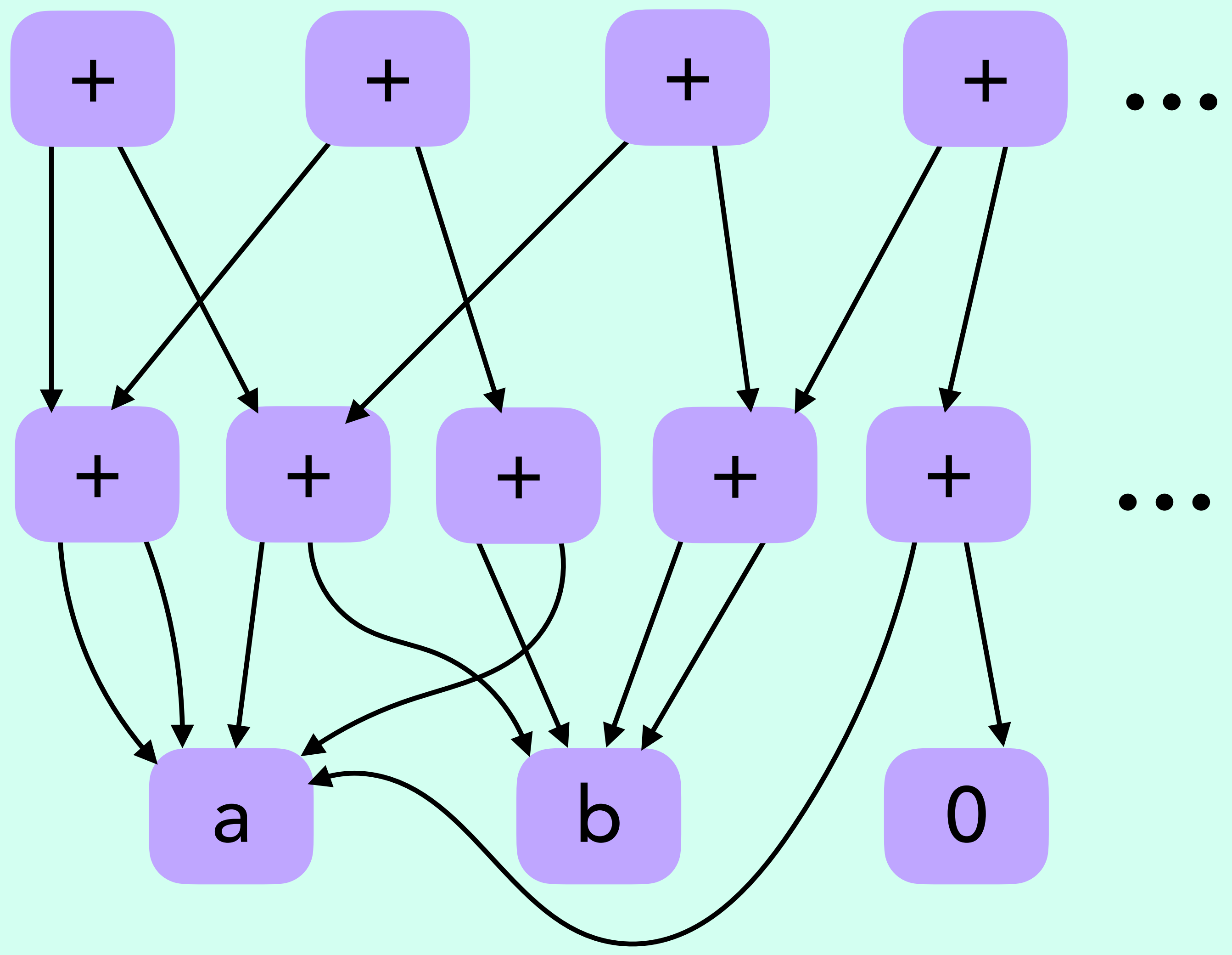
(x + x) + (x + y) ↔ (x + x) + (y + x)

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

A 3-Step Approach for Inferring Rewrite Rules

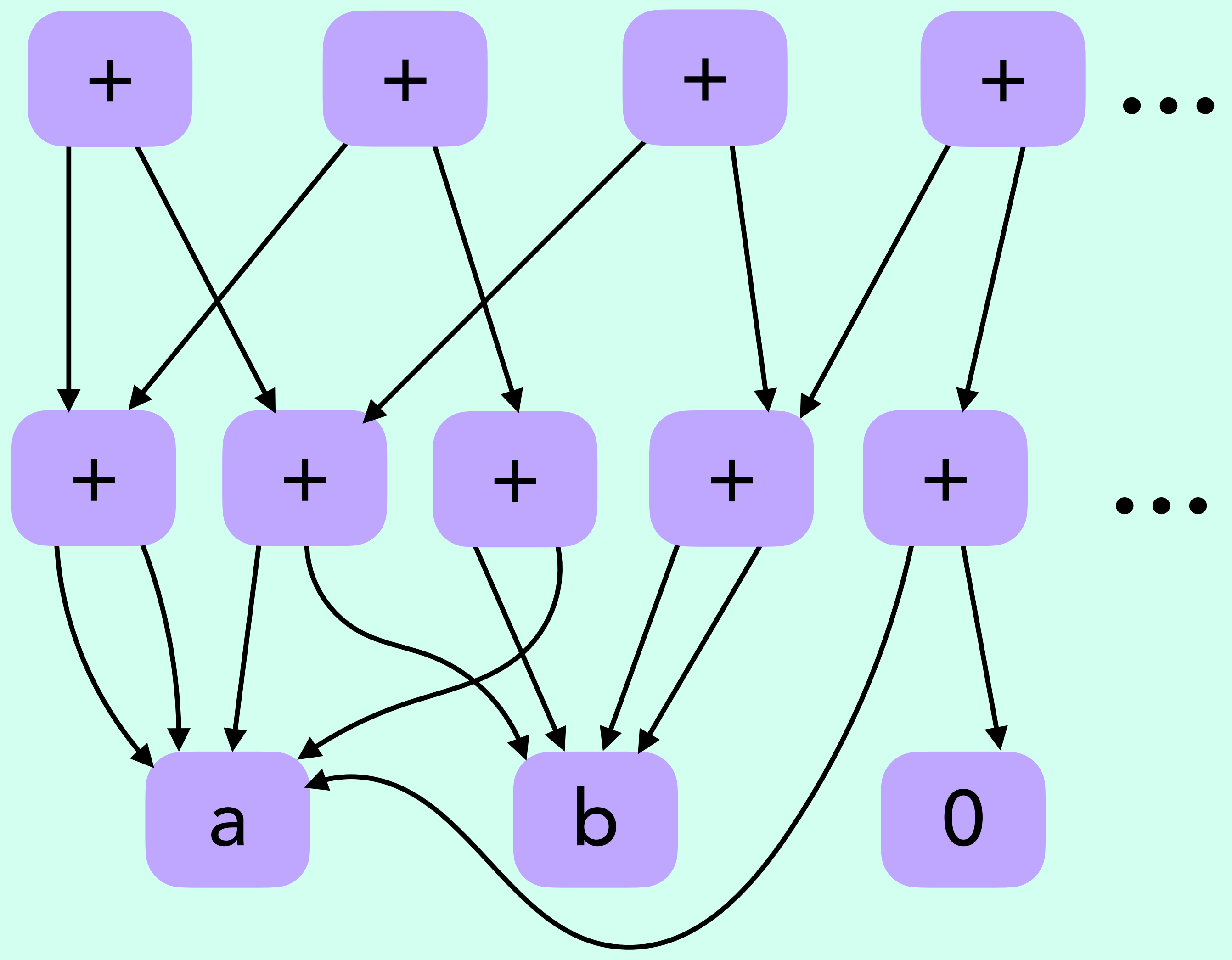
Enumerate terms from a grammar

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, …

Equality Saturation for Inferring Rewrite Rules

This Talk:

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

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

What is *Equality Saturation?*

What is *Equality Saturation*?

$(a * 2) / 2$

What is *Equality Saturation?*

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

What is *Equality Saturation?*

$$(a * 2) / 2 \quad \rightarrow \quad a$$

???

What is *Equality Saturation*?

(a * 2) / 2 → a

Rewrite rules!

(x * y) / z	⟷	x * (y / z)
y / y	⟶	1
x * 1	⟷	x

How to Apply Rewrite Rules?

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

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

How to Apply Rewrite Rules?

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

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

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

$$y / y \longrightarrow 1$$

How to Apply Rewrite Rules?

(a * 2) / 2 ⇒ a * (2 / 2)

(x * y) / z ⟷ x * (y / z)

a * (2 / 2) ⇒ a * 1

y / y ⟶ 1

a * 1 ⇒ a

x * 1 ⟷ x

Destructively, In a Specific Order

(a * 2 … 2 / 2)

a * (2 … * 1

Order of rule application affects result

Missed opportunities for optimizations

Same order may not work for all inputs

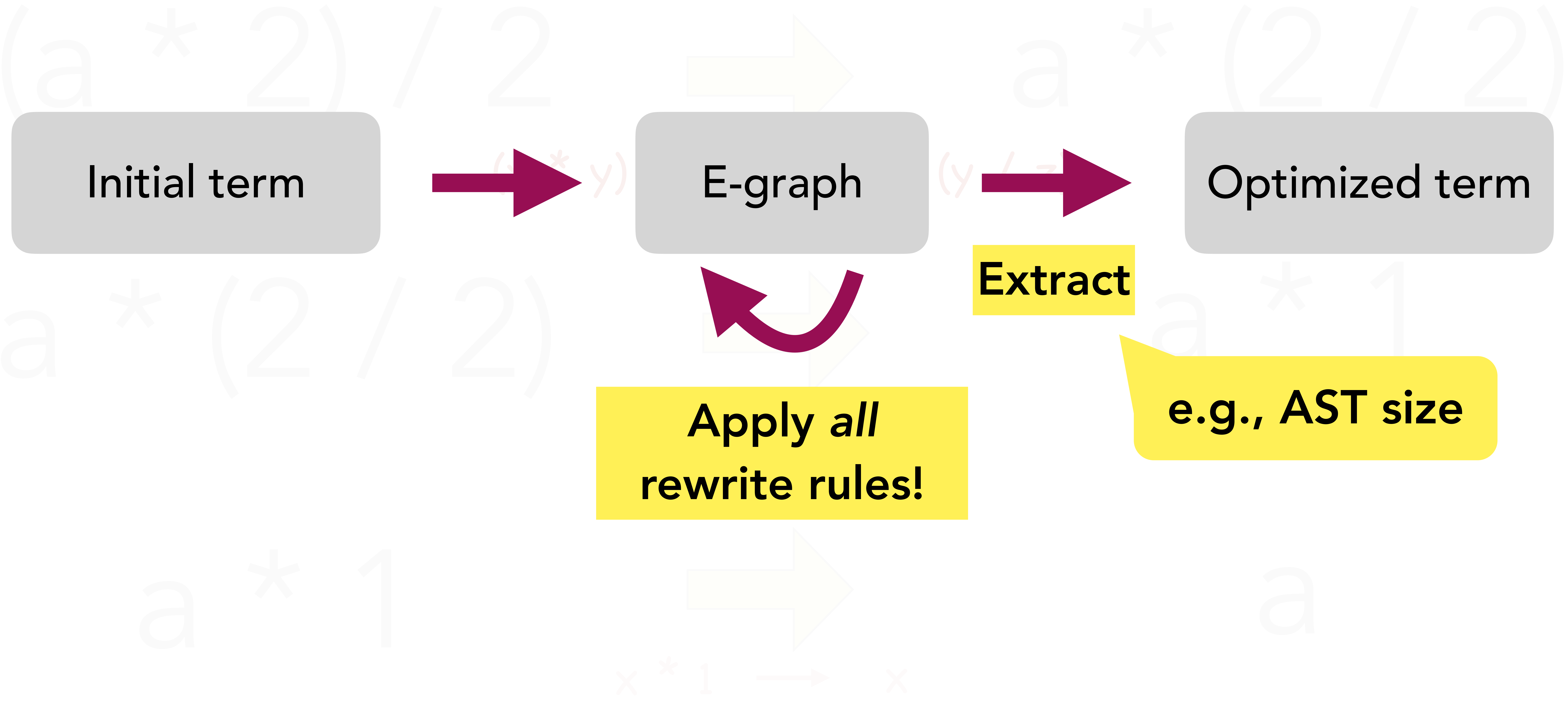
Old expression is lost

a

e.g., supporting commutativity is hard without additional tricks to ensure termination!

x * 1 ⟷ x

Equality Saturation Mitigates Phase Ordering!

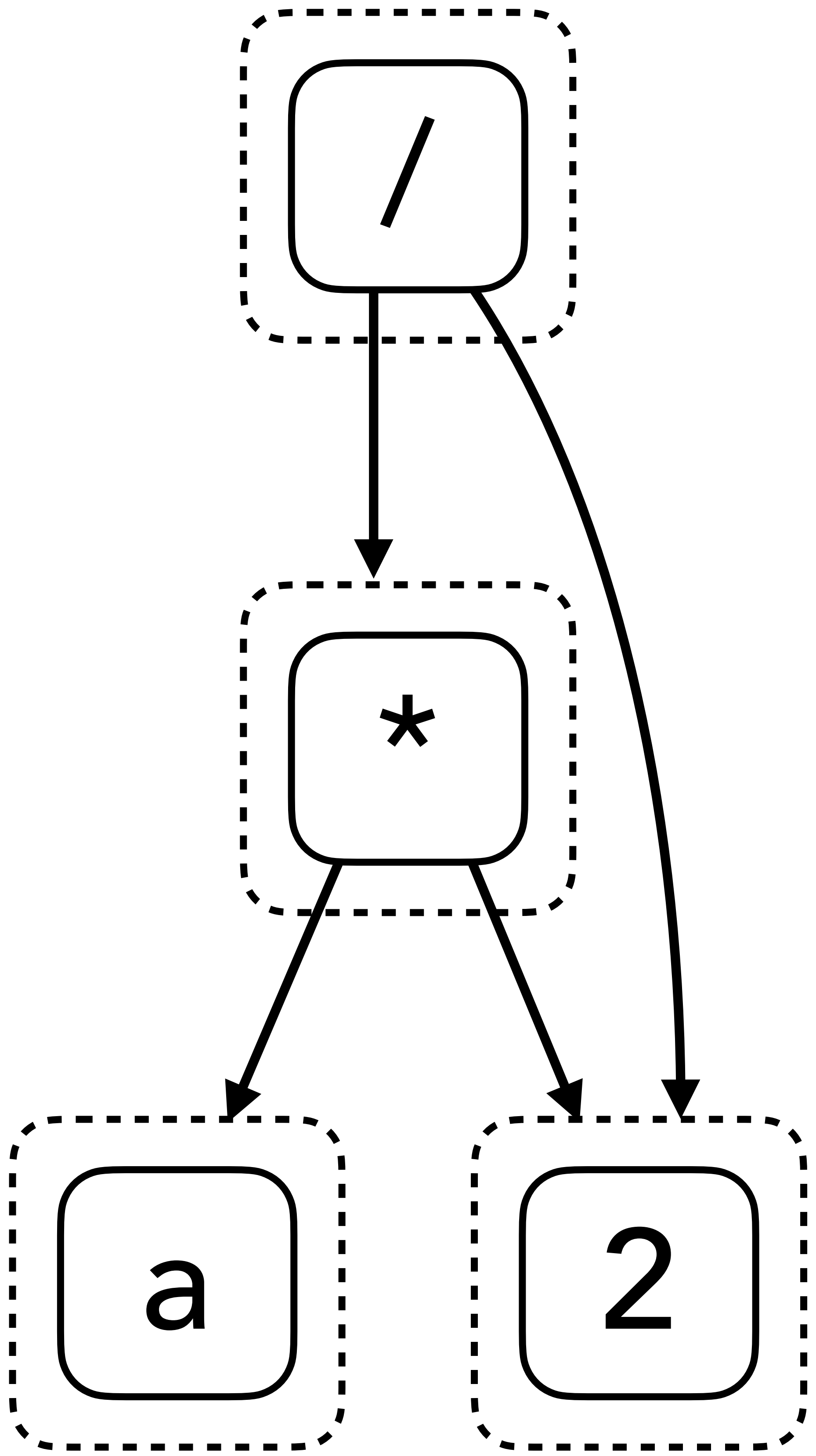


How Does Equality Saturation Work?

(a * 2) / 2

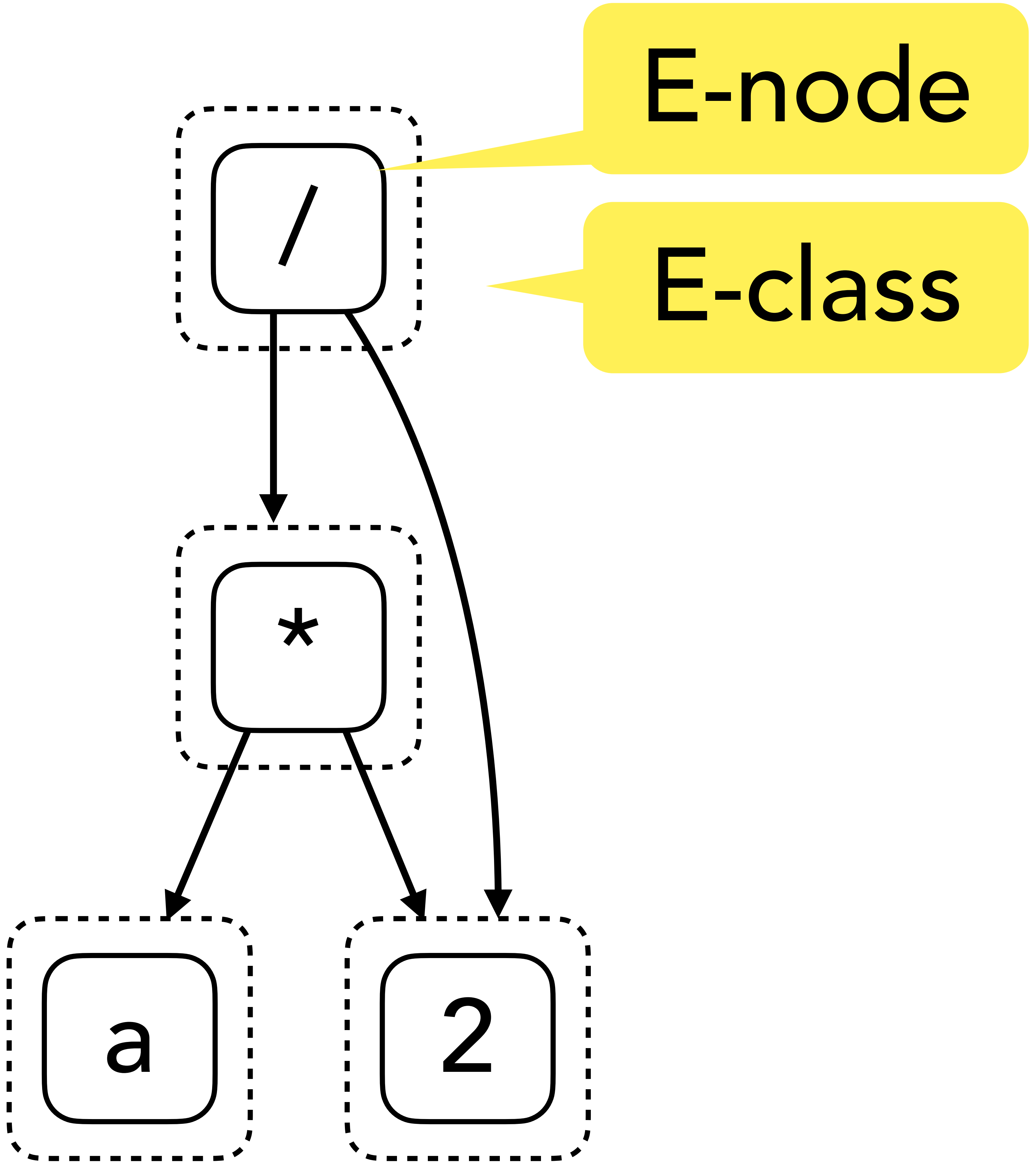
How Does Equality Saturation Work?

(a * 2) / 2



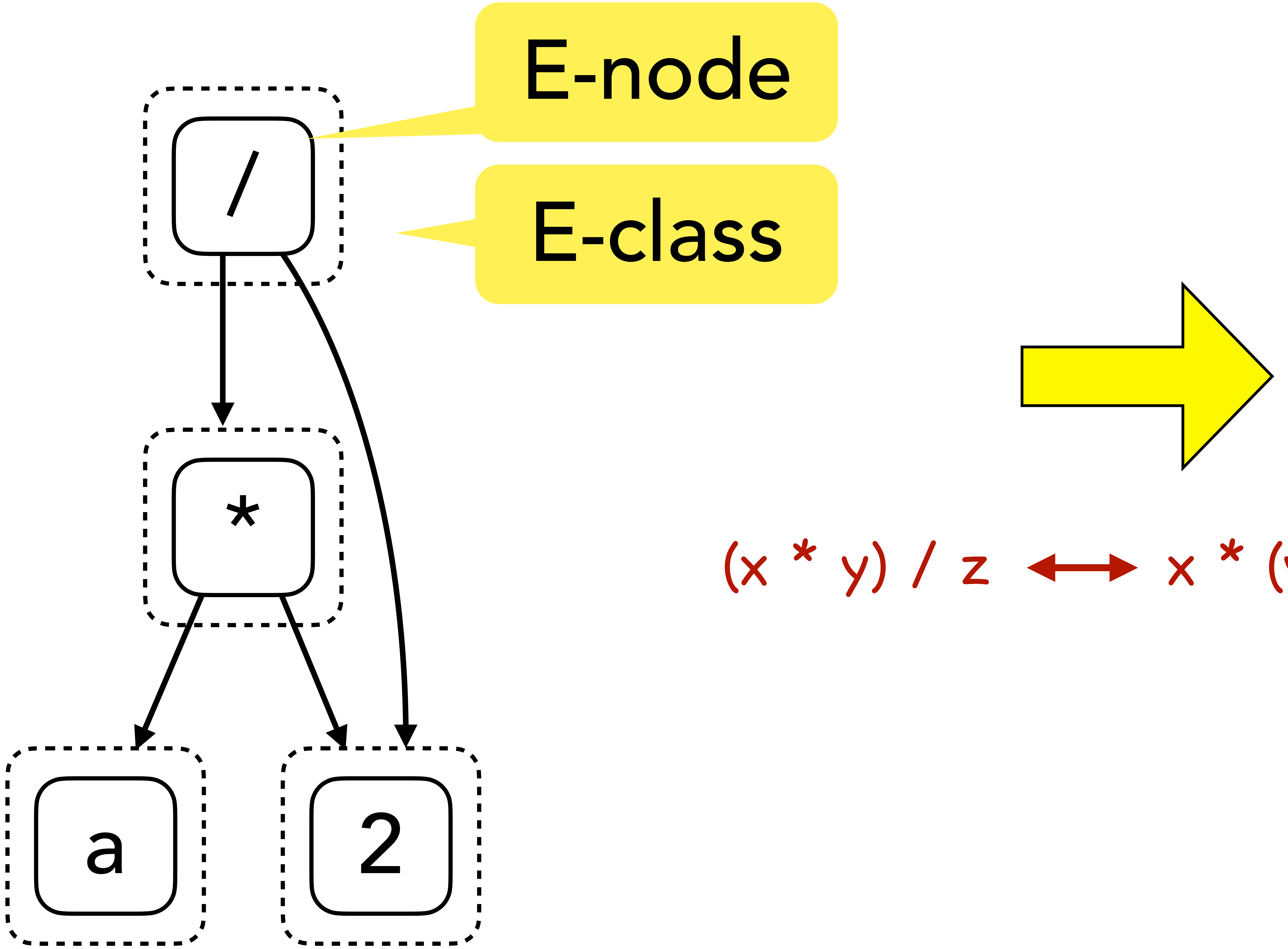
How Does Equality Saturation Work?

(a * 2) / 2



How Does Equality Saturation Work?

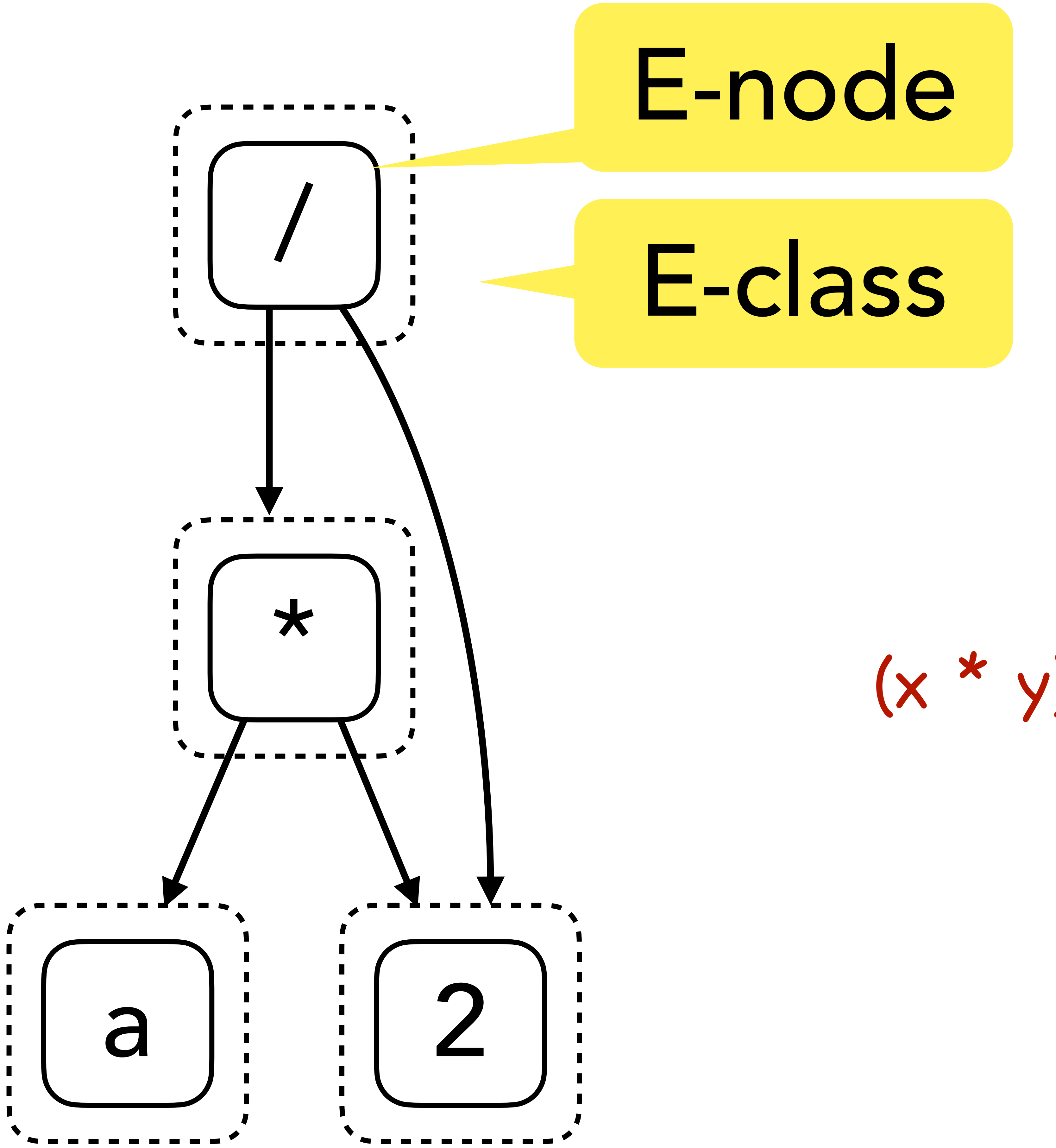
(a * 2) / 2



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

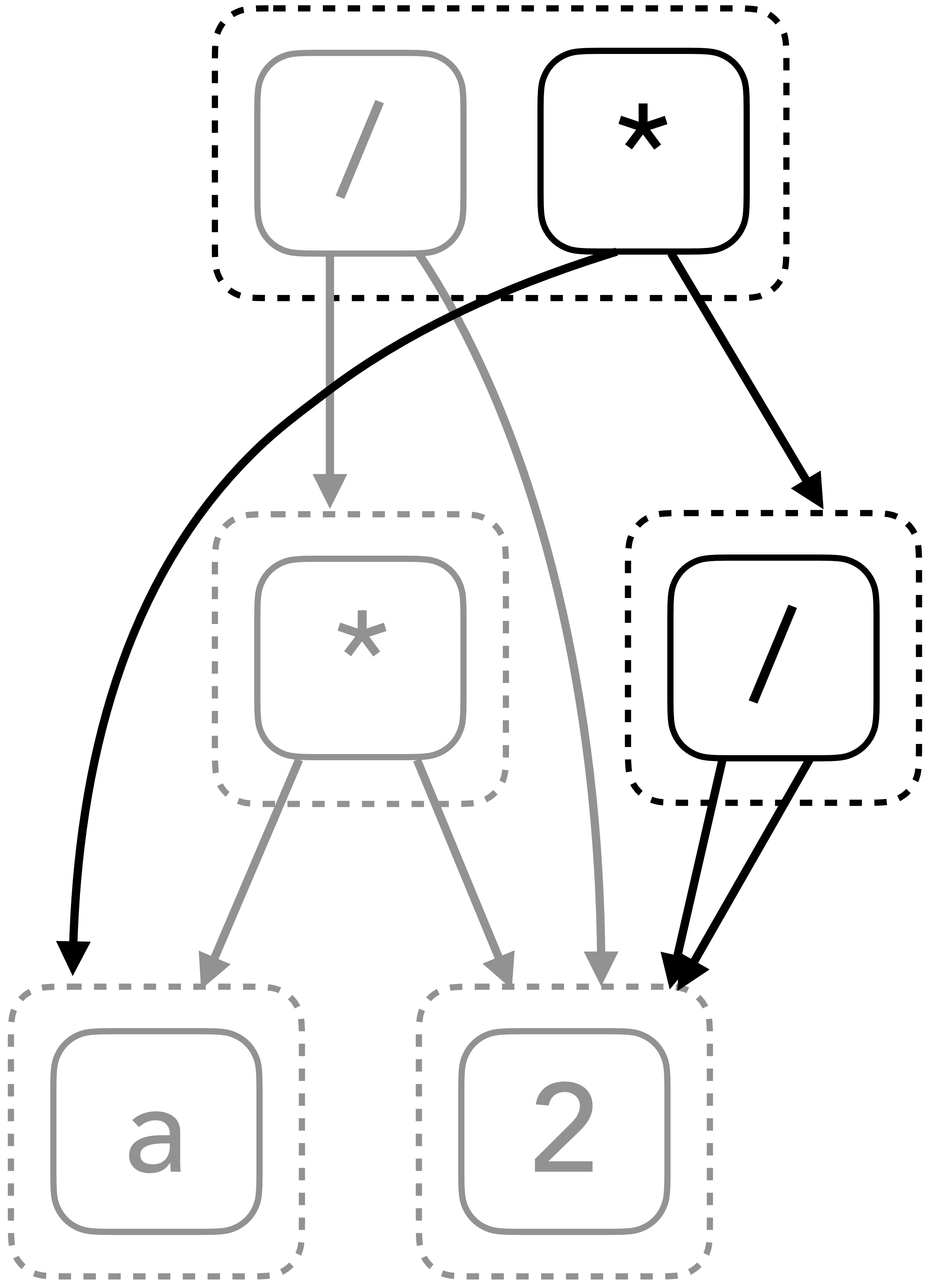
How Does Equality Saturation Work?

(a * 2) / 2

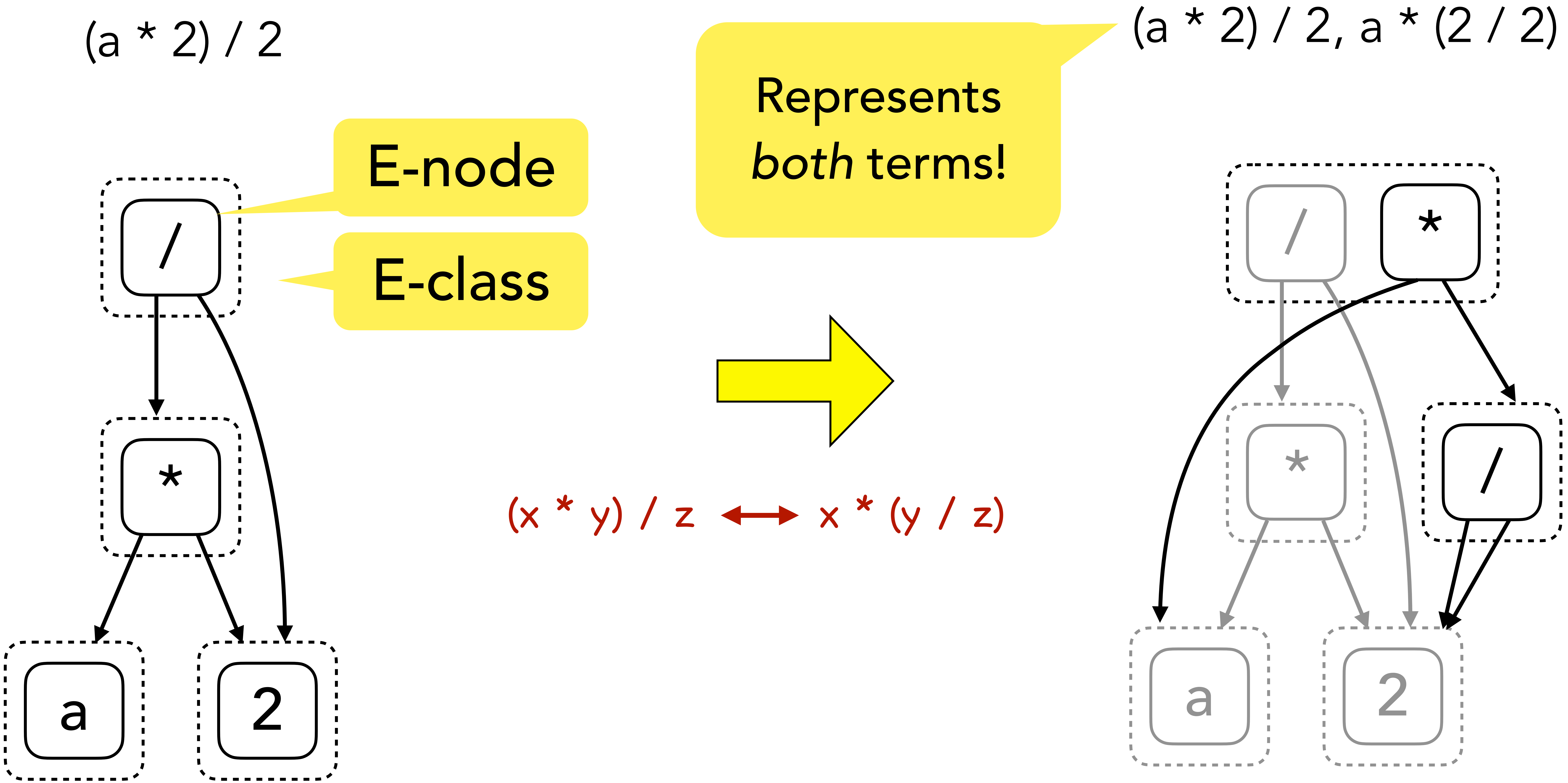


(x * y) / z ⟷ x * (y / z)

(a * 2) / 2, a * (2 / 2)



How Does Equality Saturation Work?

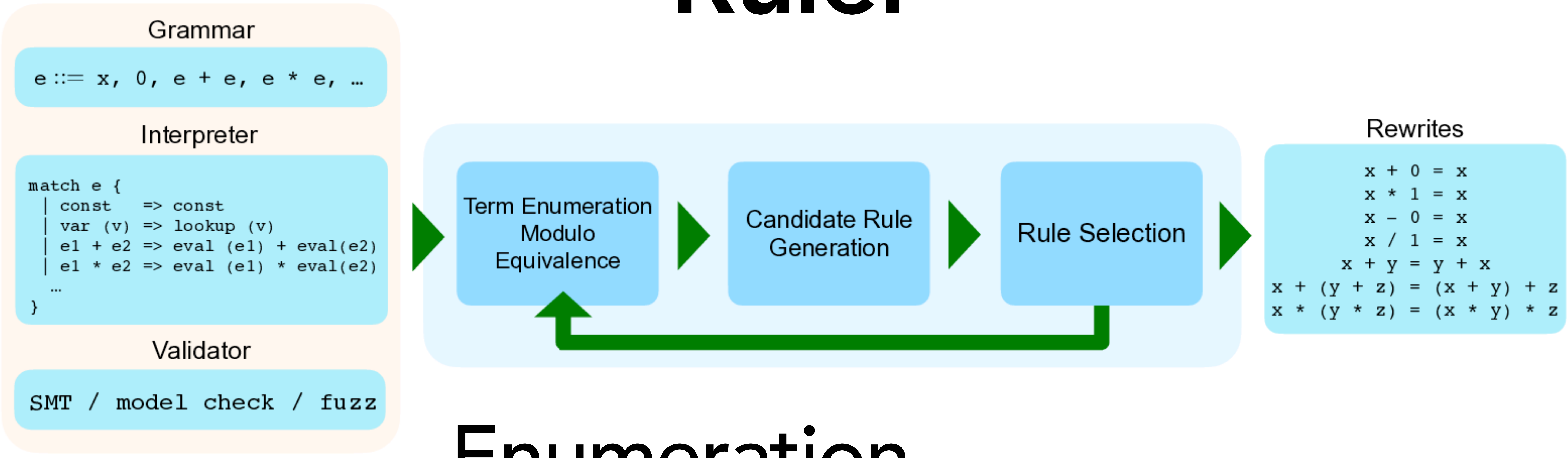


Equality Saturation for Inferring Rewrite Rules

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

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

Ruler

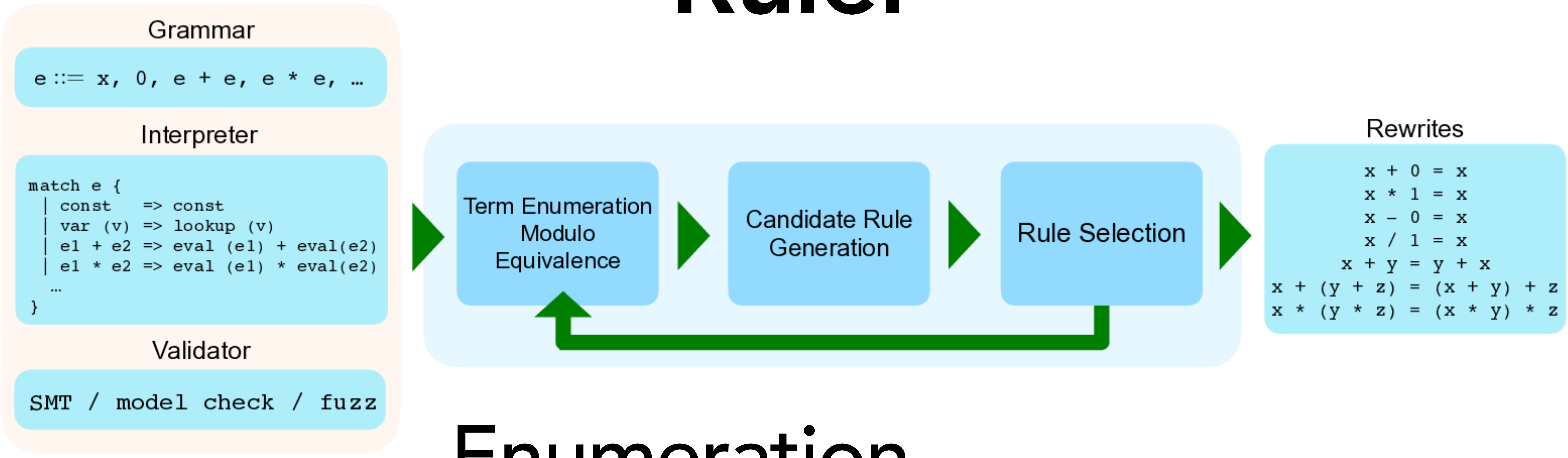


Enumeration

Candidate Generation

Rule Selection

Ruler



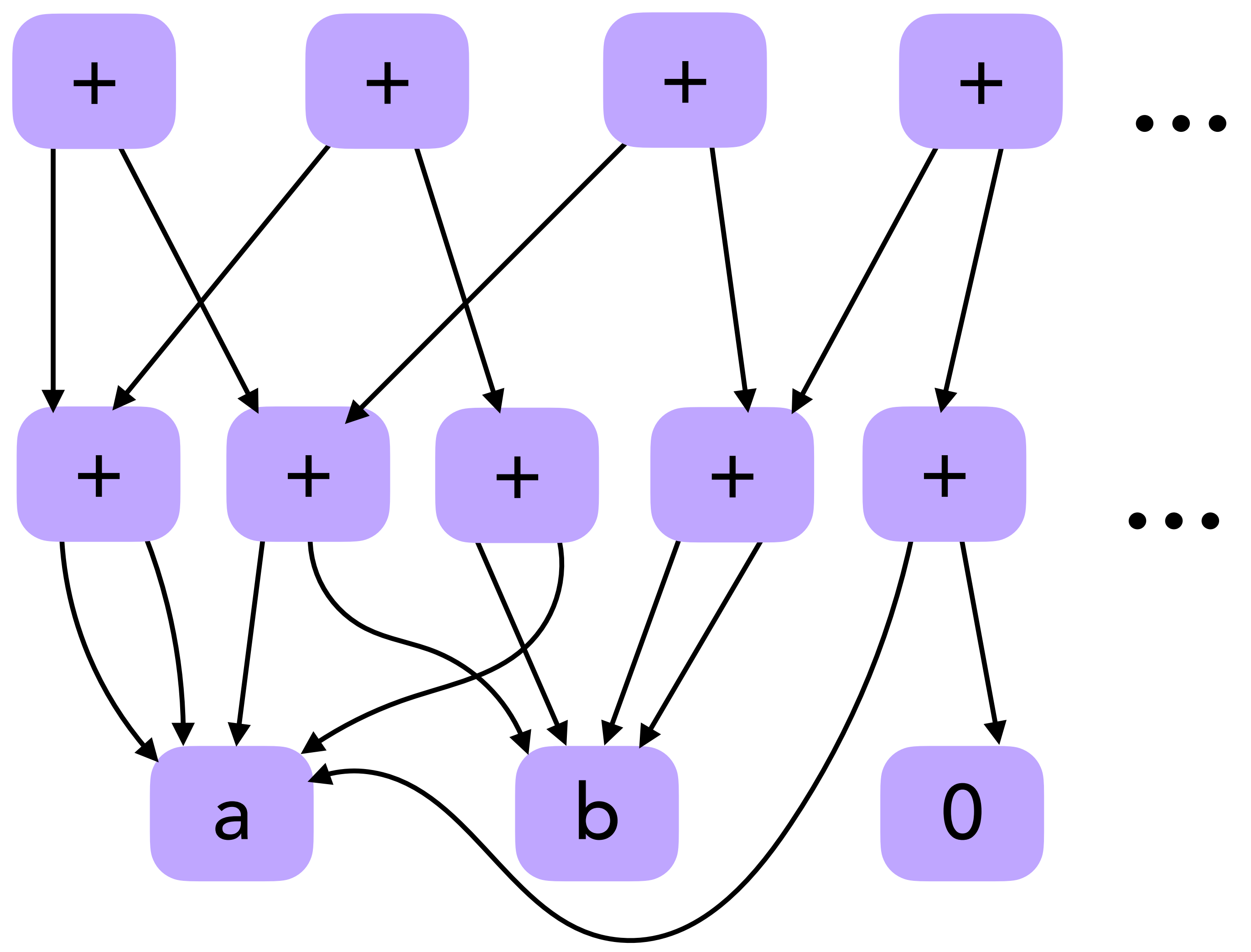
Enumeration

Candidate Generation

Rule Selection

Enumeration Modulo Equality Saturation

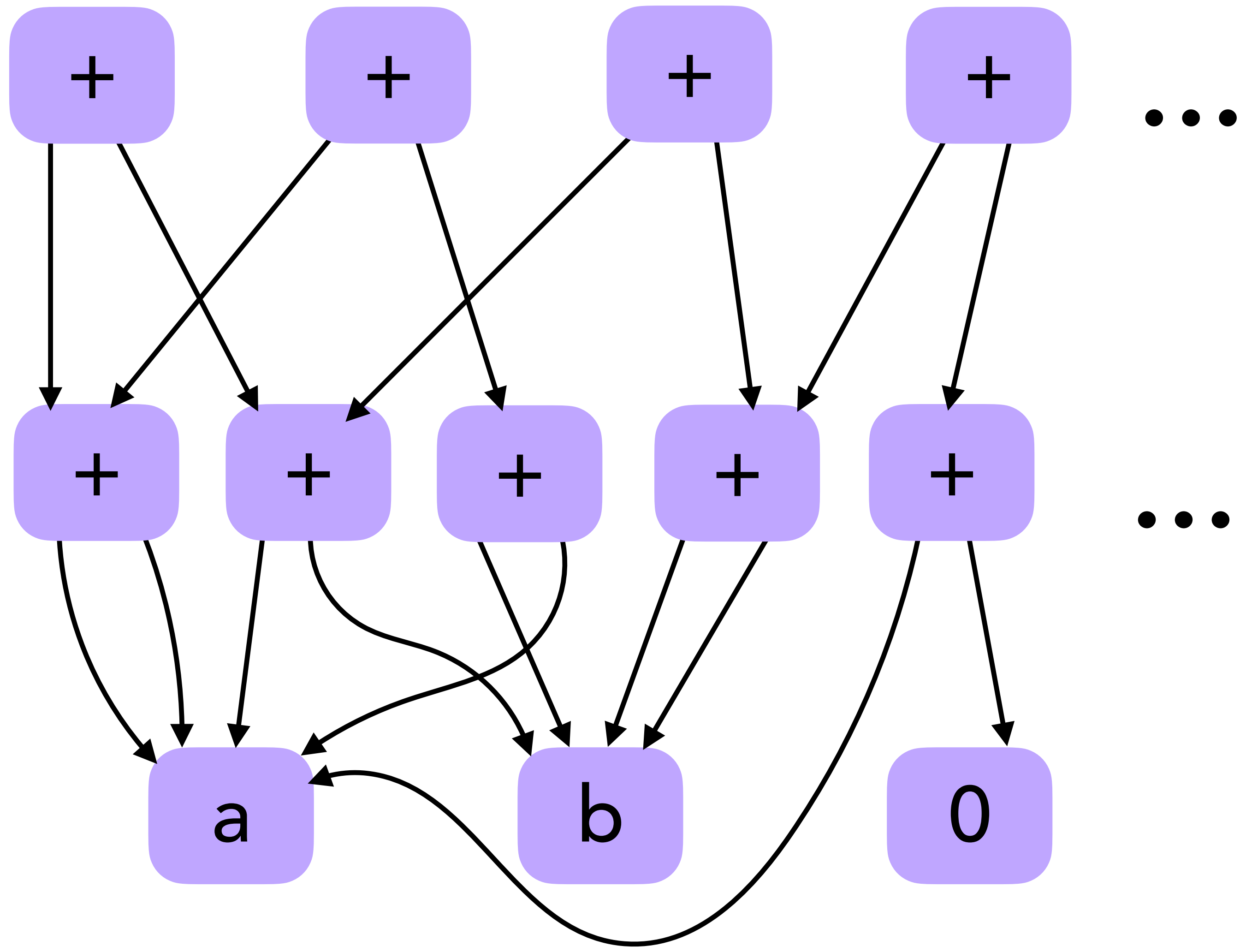
a, b, 0, +, ...

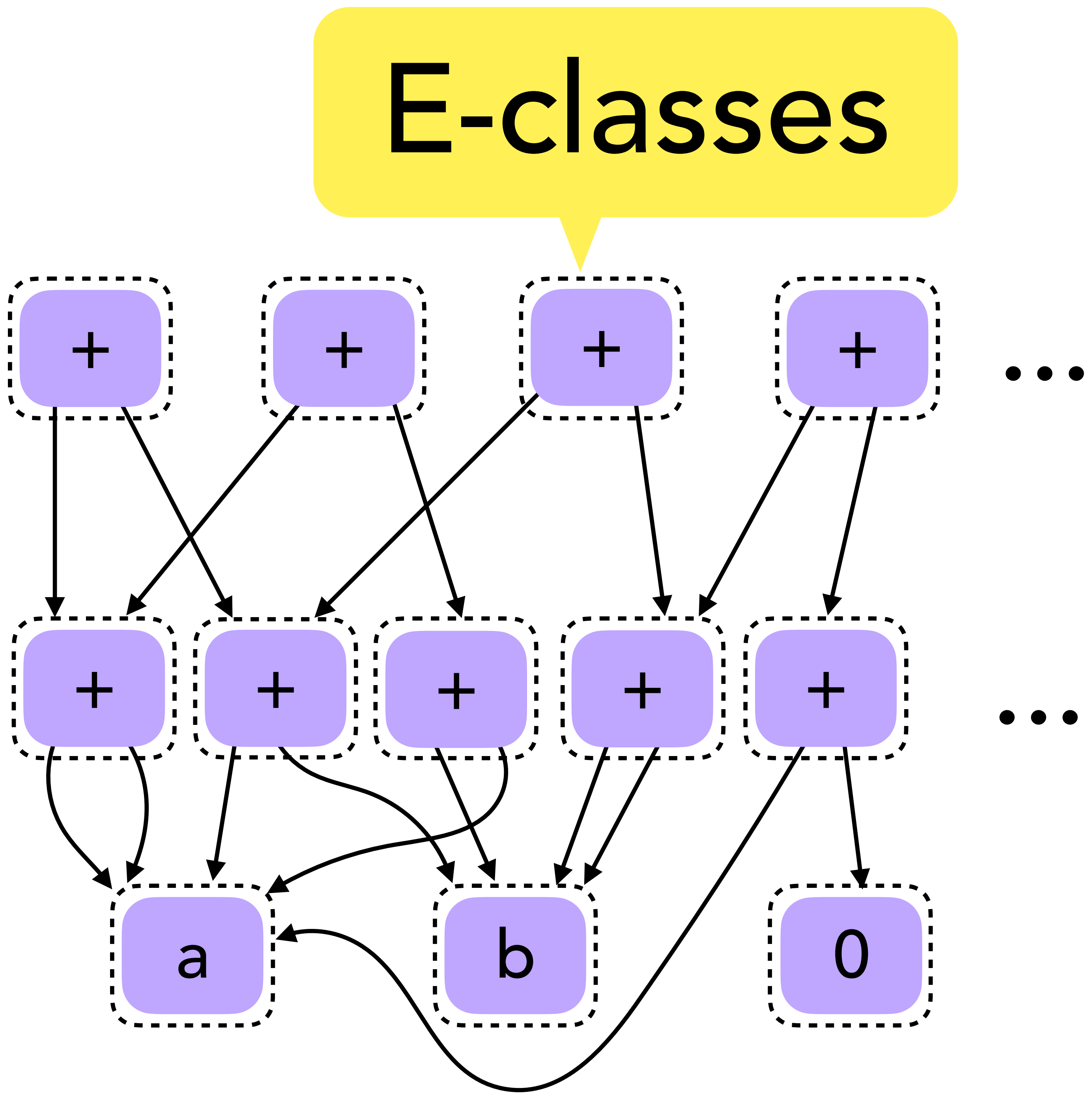


Exponentially many terms!

Enumeration Modulo Equality Saturation

a, b, 0, +, …



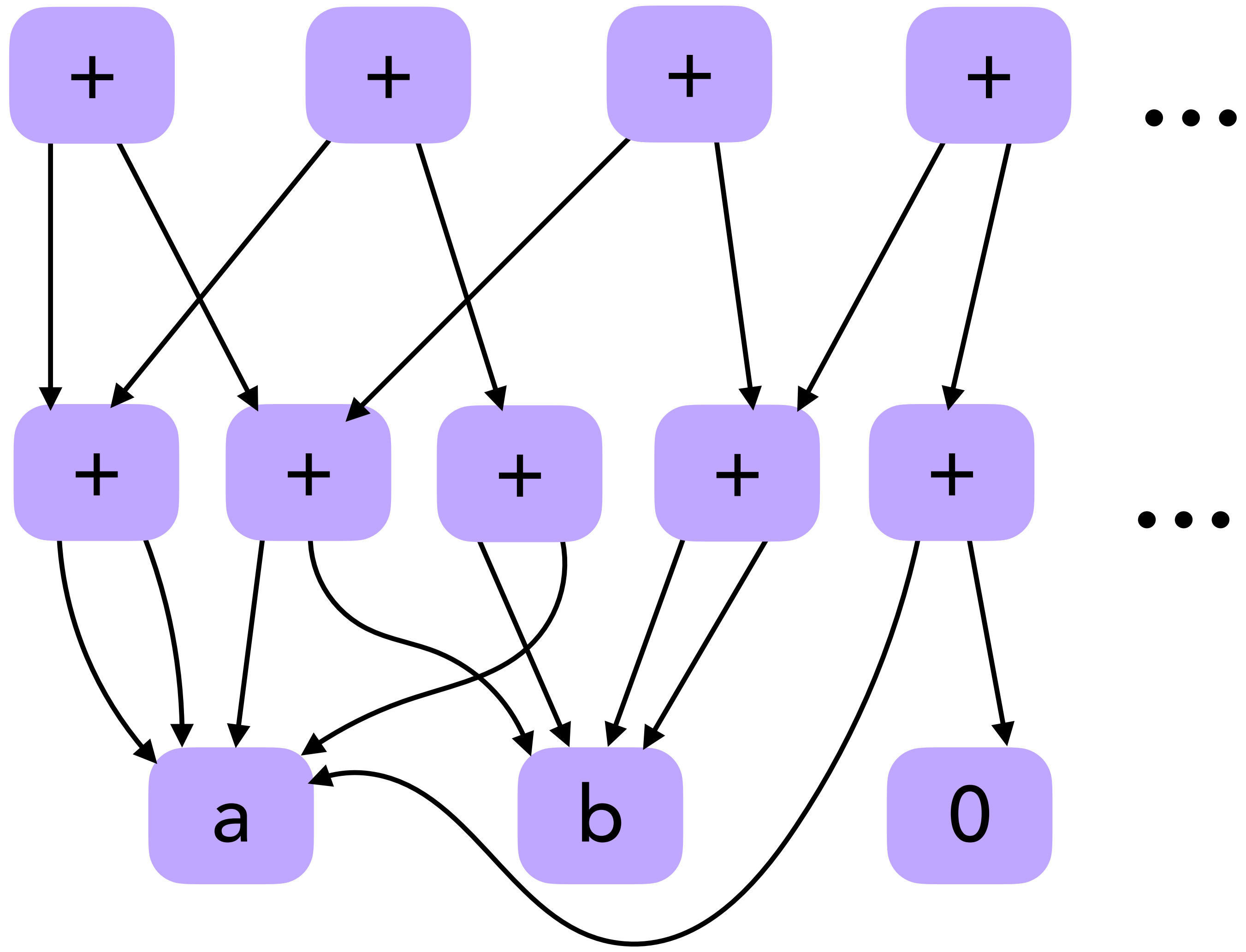


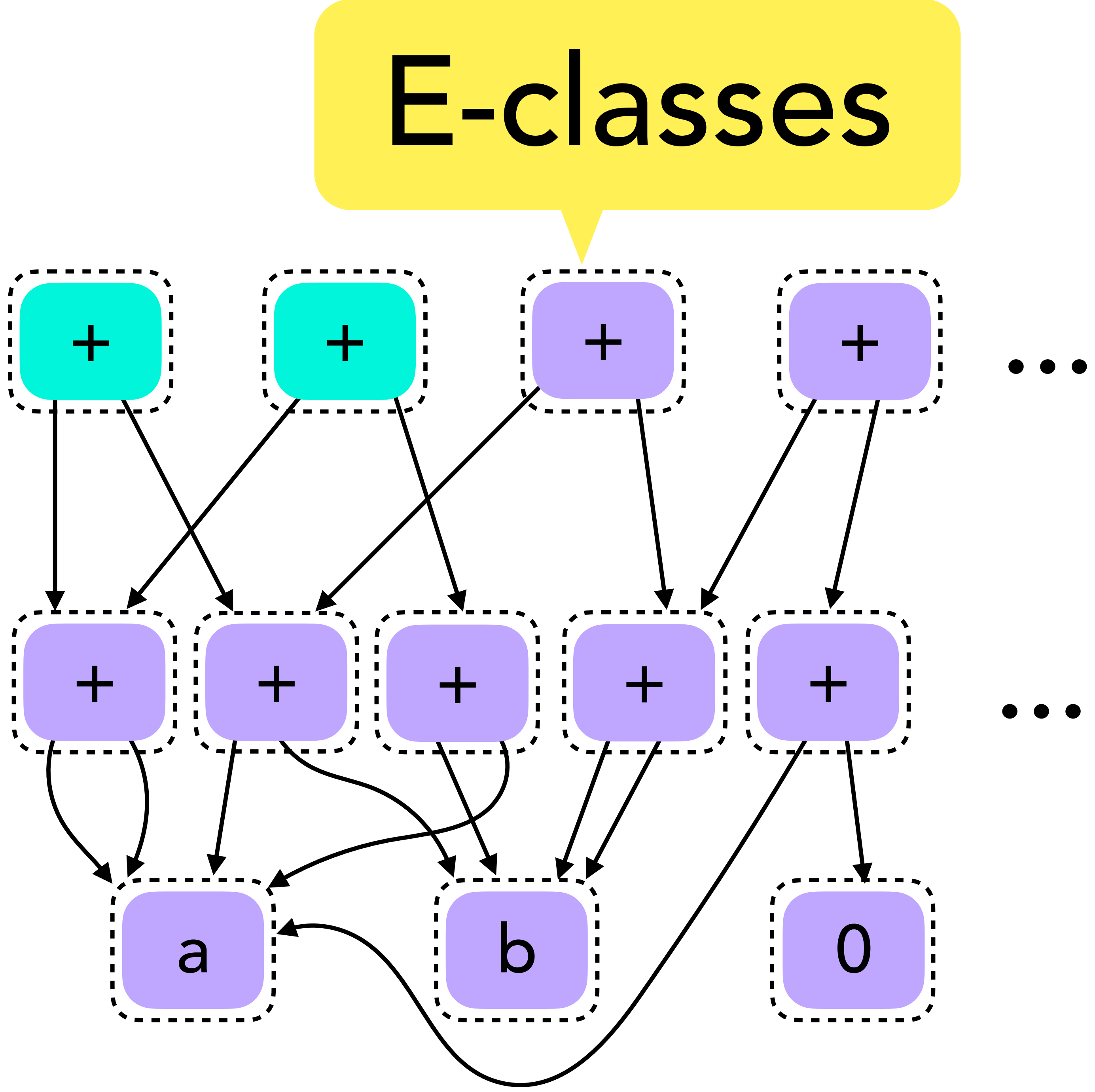
Exponentially many terms!

Enumerate *over* an E-graph

Enumeration Modulo Equality Saturation

a, b, 0, +, …





(x + x) + (x + y)

=

(x + x) + (y + x)

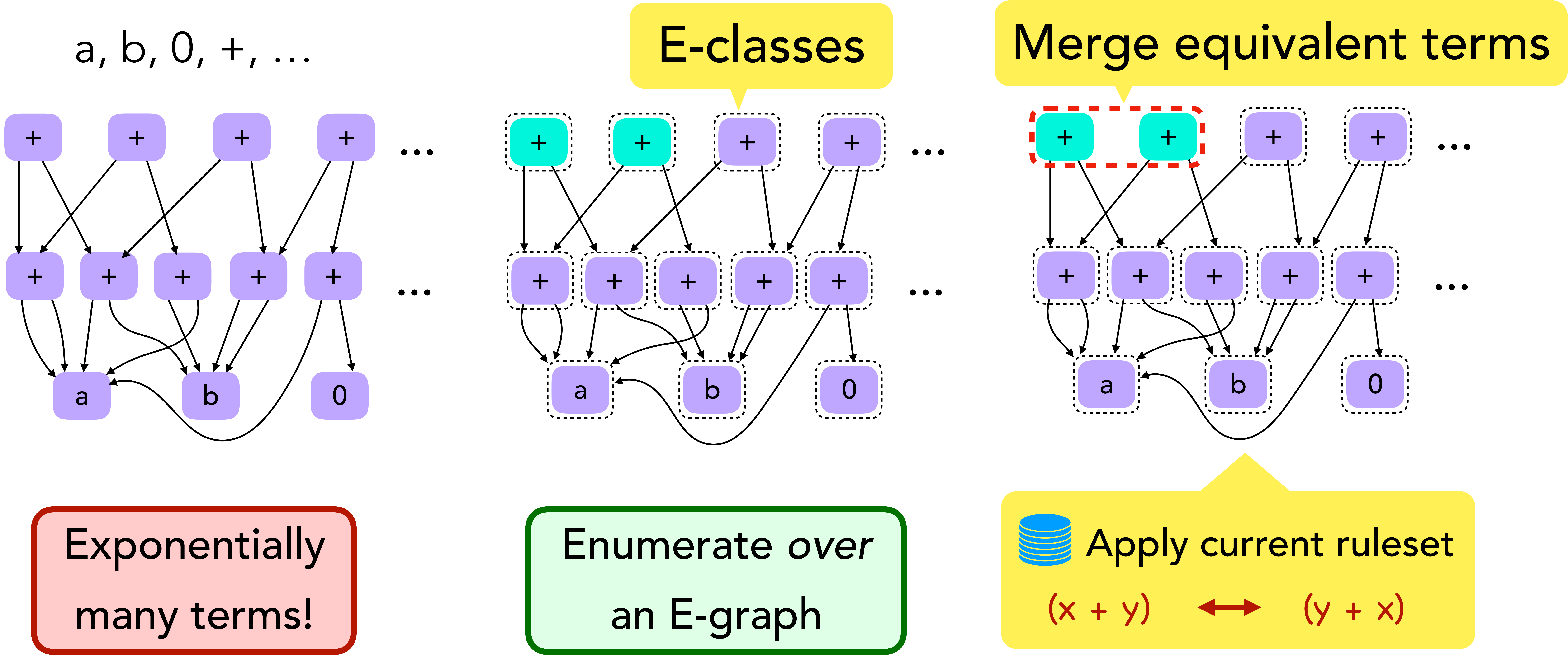
Exponentially many terms!

Enumerate *over* an E-graph

Apply current ruleset

(x + y) ⟷ (y + x)

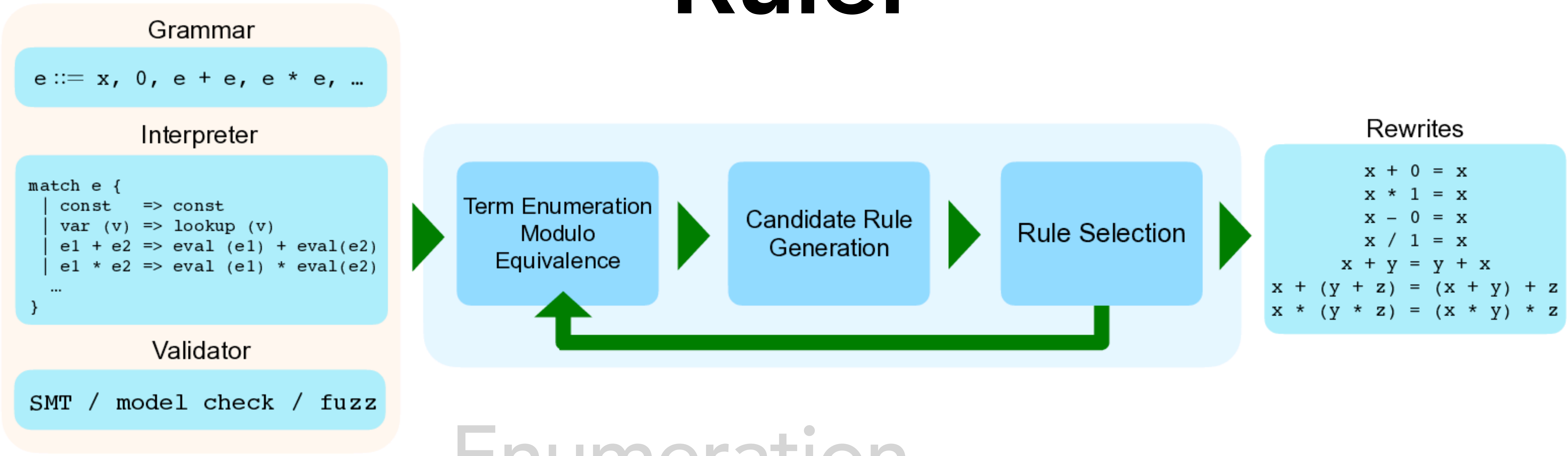
Enumeration Modulo Equality Saturation



Enumeration Modulo Equality Saturation

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

Ruler

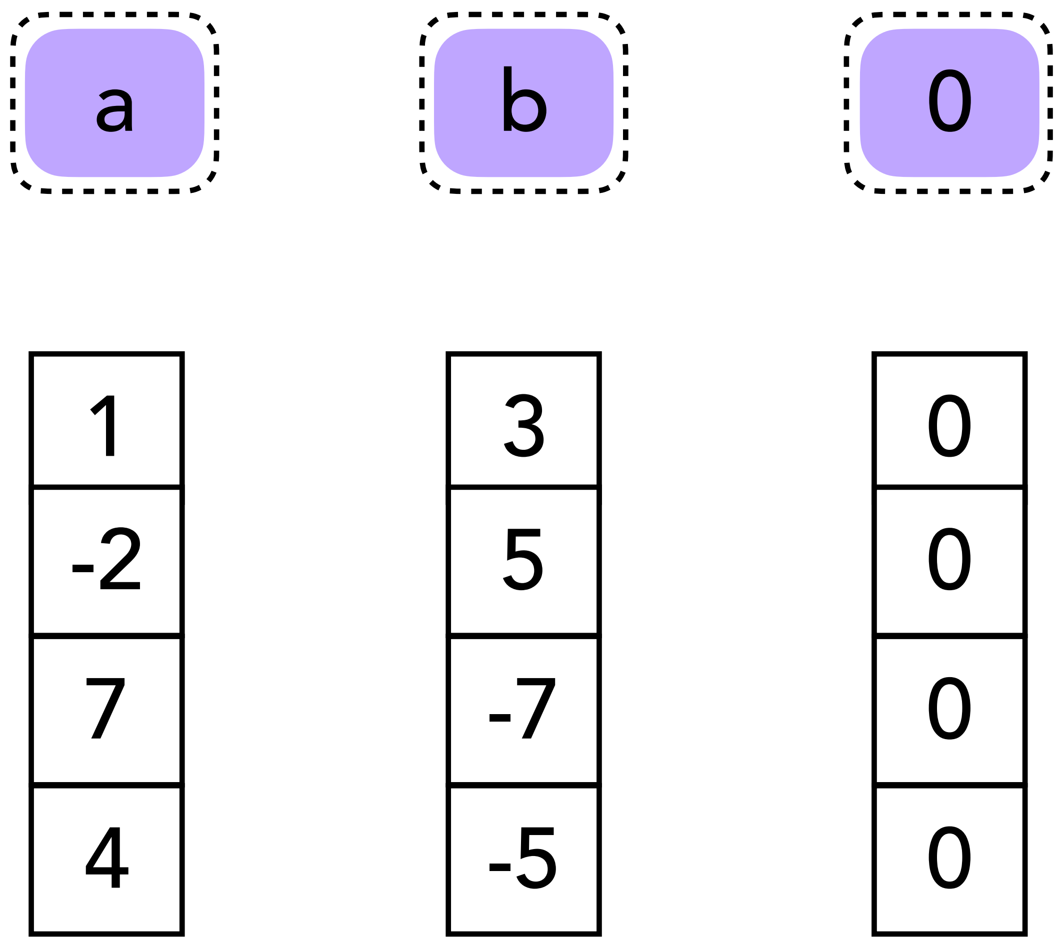


Enumeration

Candidate Generation

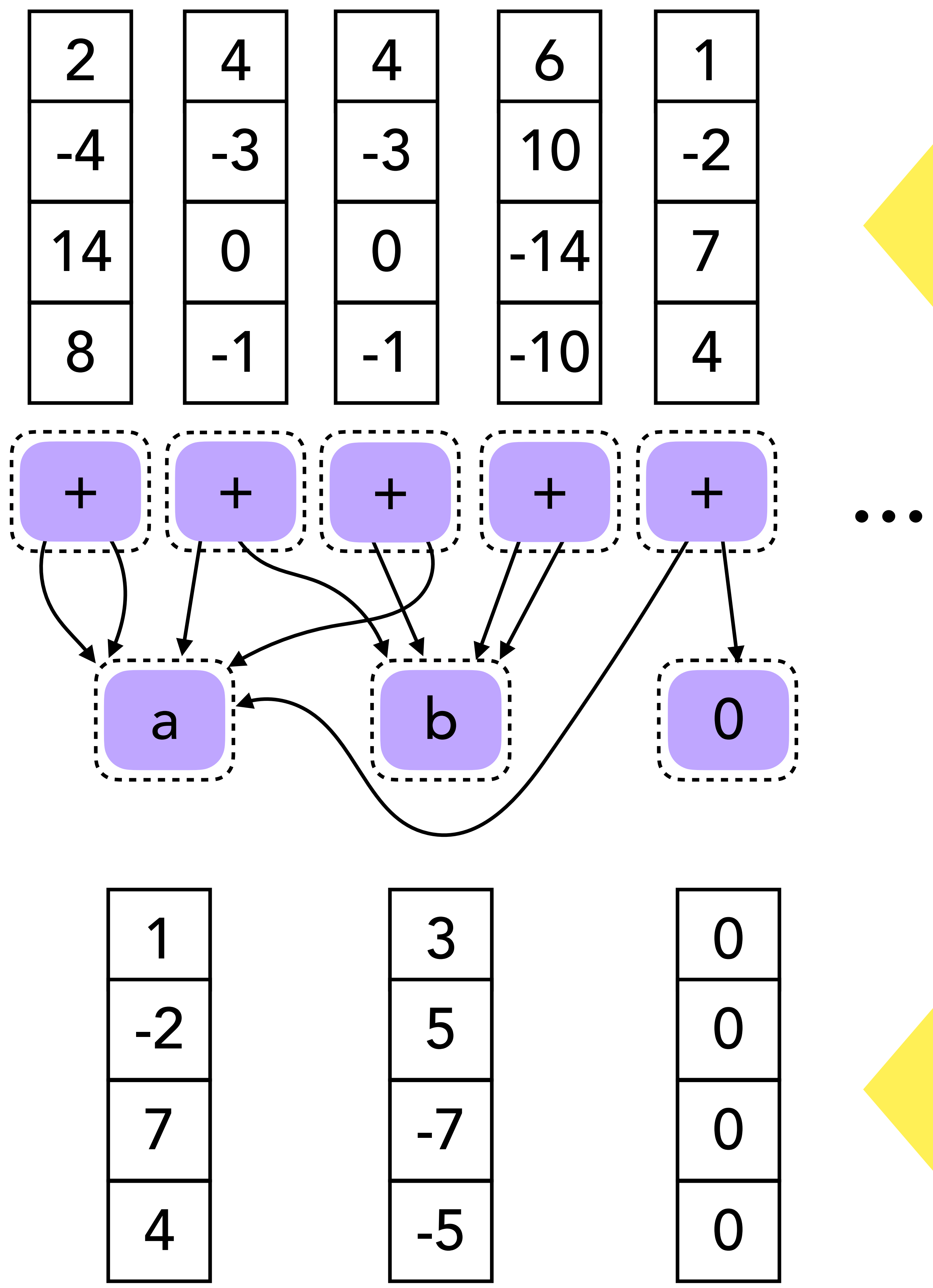
Rule Selection

Candidate Generation by Characteristic Vector Matching



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

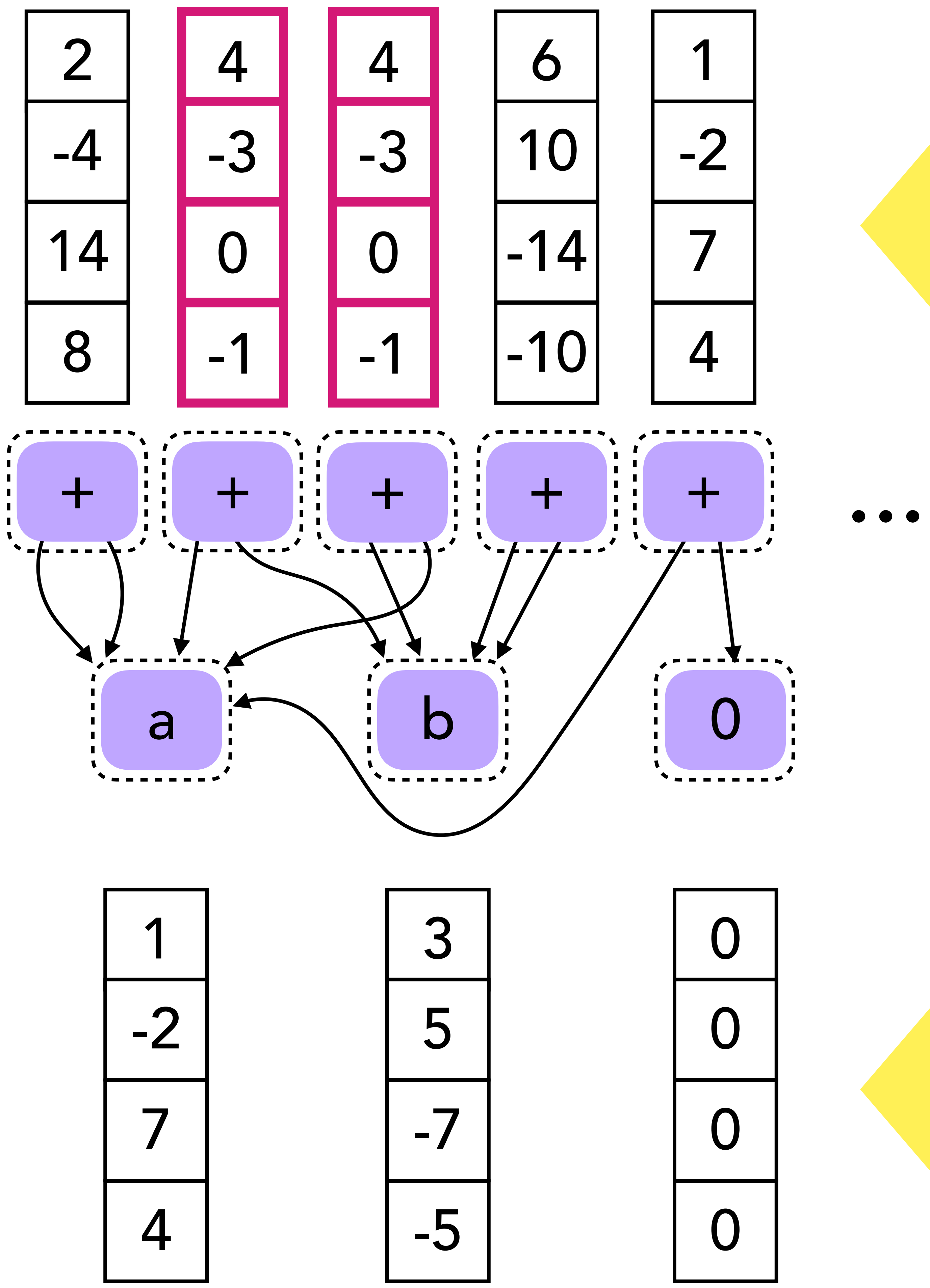
Candidate Generation by Characteristic Vector Matching



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

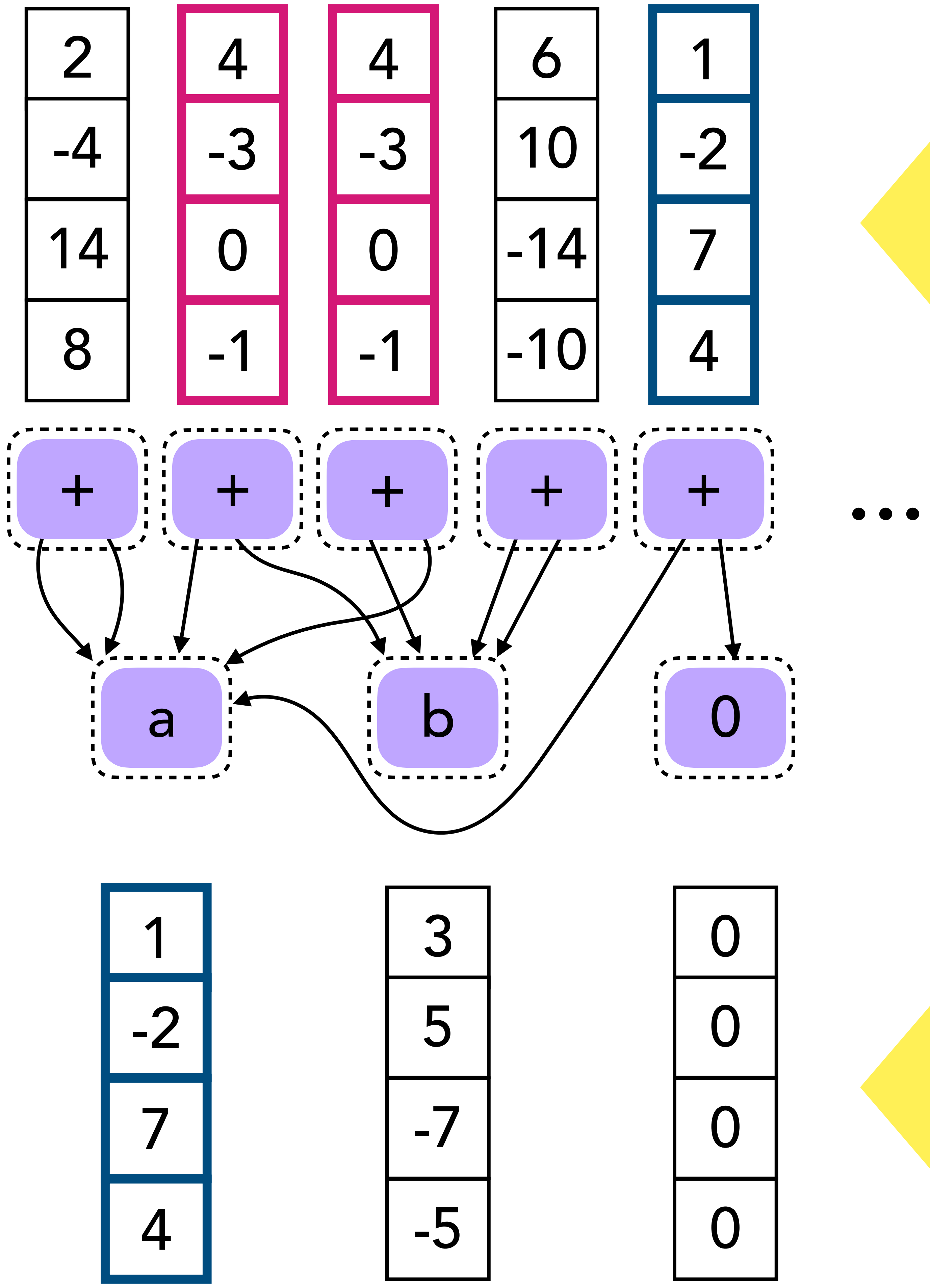


Compute the cvecs for newly enumerated E-classes

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

(x + y) ⟷ (y + x)

Candidate Generation by Characteristic Vector Matching



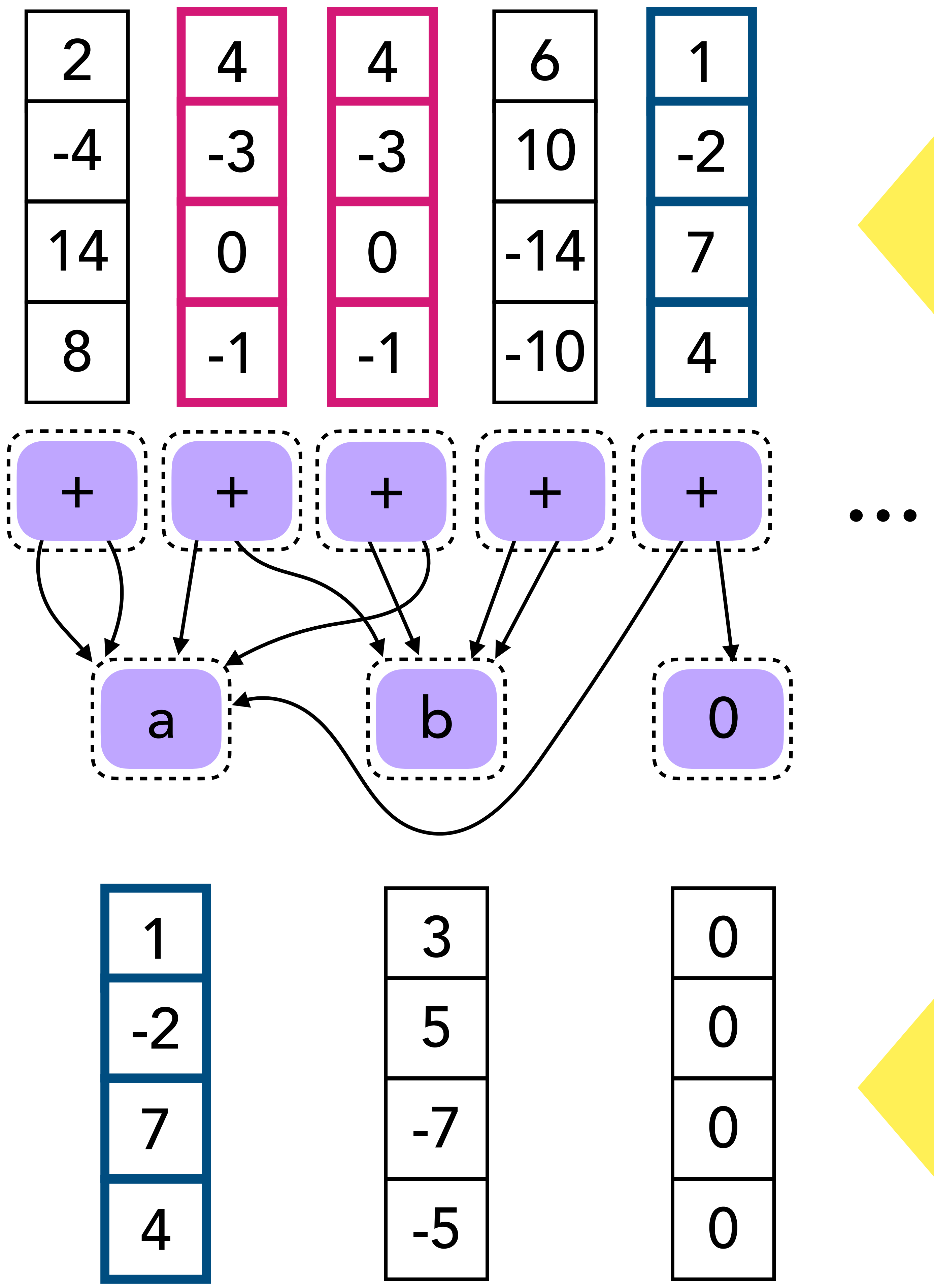
Compute the cvecs for newly enumerated E-classes

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

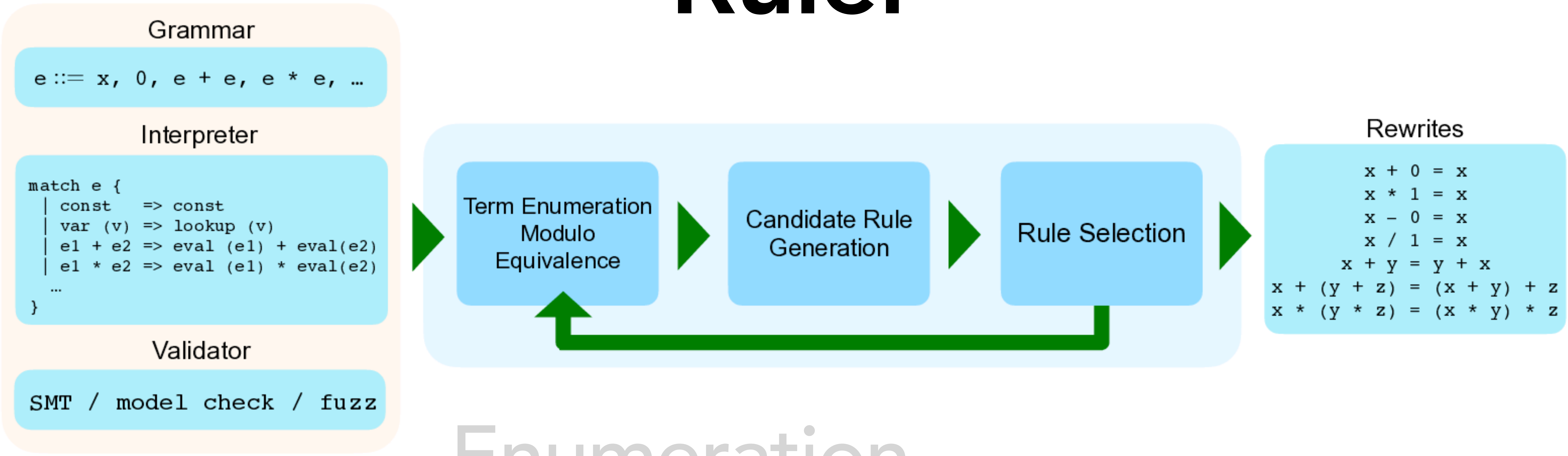
Seed initial E-classes with concrete values (cvecs) from the domain

(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

$$C = \begin{array}{ccc} (x + y) & \longleftrightarrow & (y + x) \\ (x + 0) & \longleftrightarrow & (0 + x) \\ (y + 0) & \longleftrightarrow & (0 + y) \\ (x * y) & \longleftrightarrow & (y * x) \\ (x * 1) & \longleftrightarrow & (1 * x) \\ (y * 1) & \longleftrightarrow & (1 * y) \end{array}$$

Rule Selection with Equality Saturation

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

C =

(x + y) ⟷ (y + x)

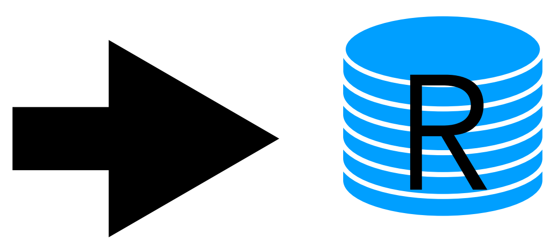
(x * y) ⟷ (y * x)

(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)



Rule Selection with Equality Saturation

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

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)

(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

R

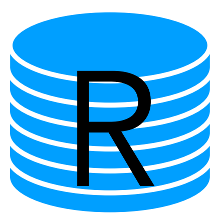
Instantiate and add to rule E-graph

Rule Selection with Equality Saturation

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

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)



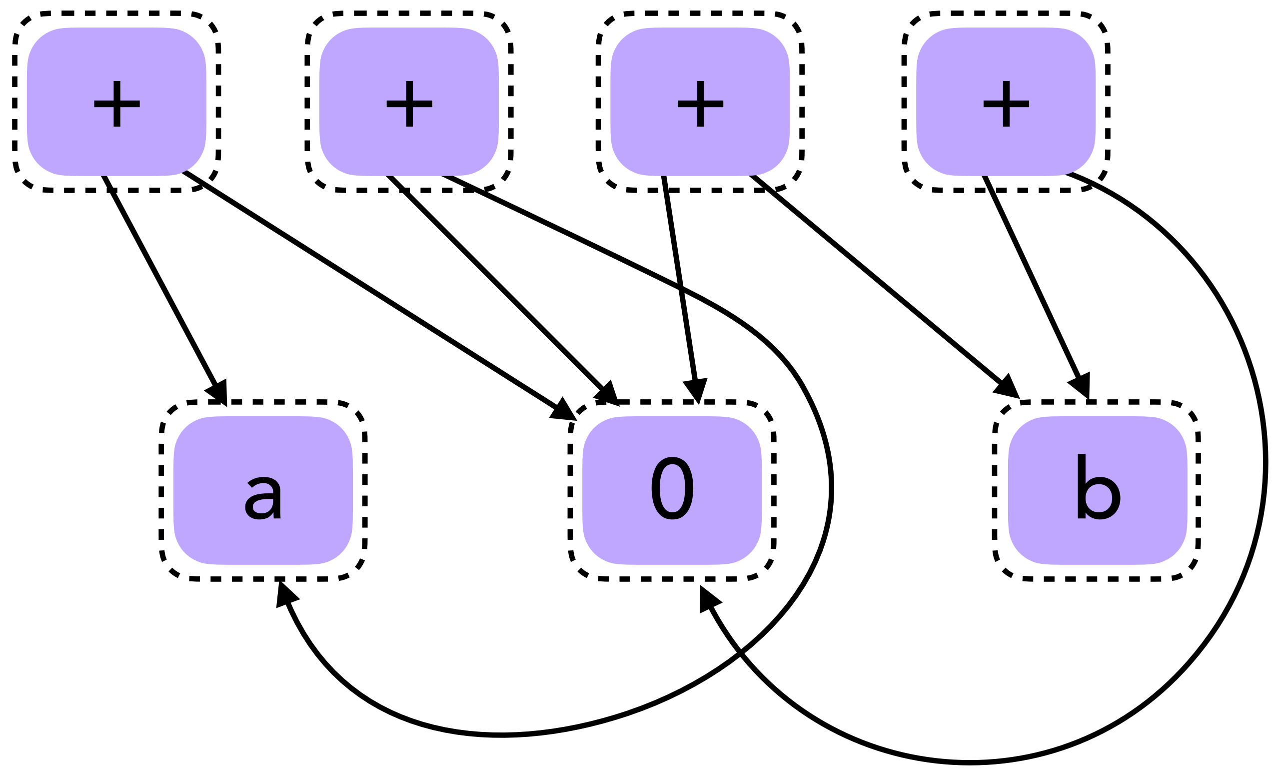
(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

Instantiate and add to rule E-graph



Rule Selection with Equality Saturation

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

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)

(x + 0) ⟷ (0 + x)

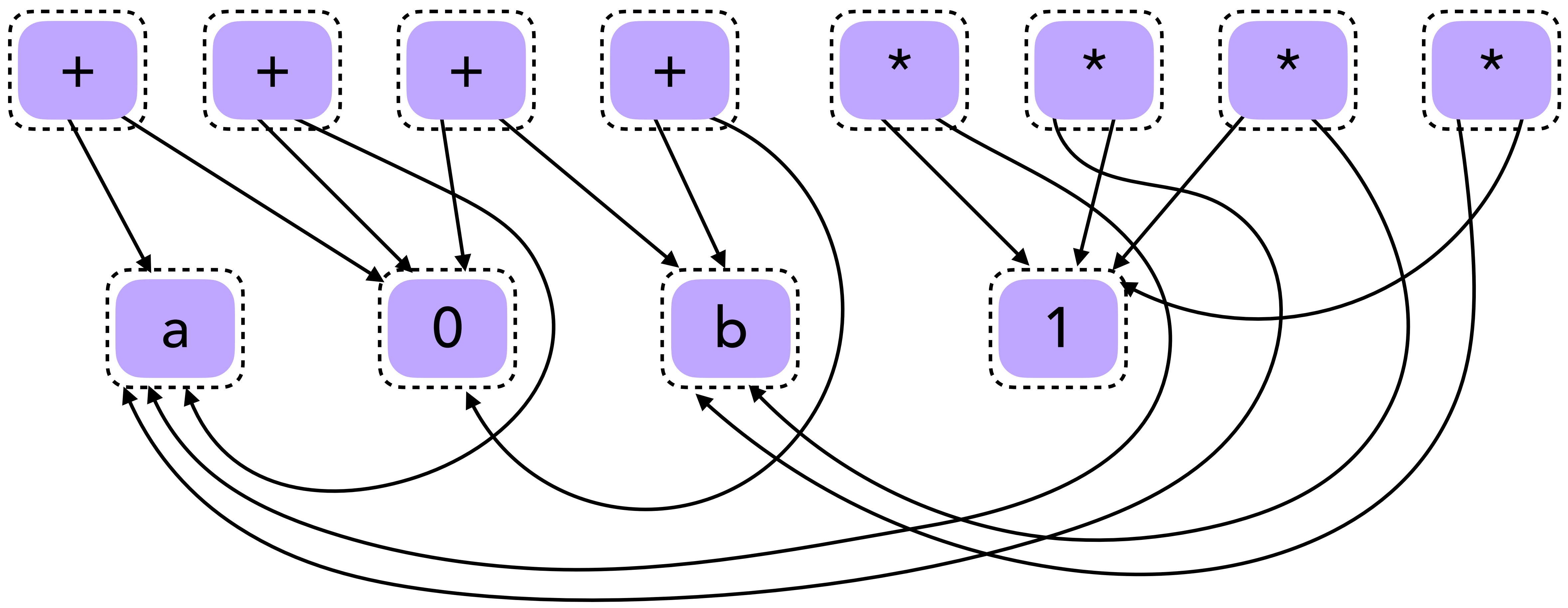
(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

R

Instantiate and add to rule E-graph



Rule Selection with Equality Saturation

R

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)

Run equality saturation

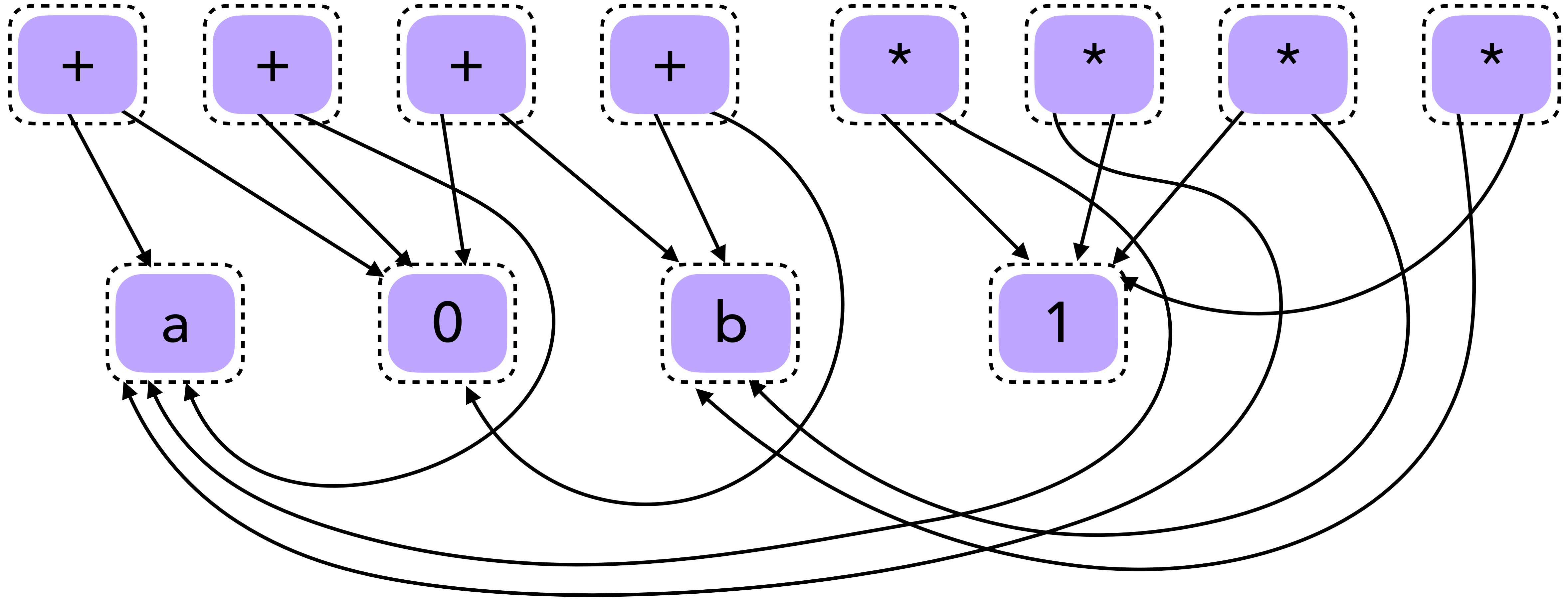
(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

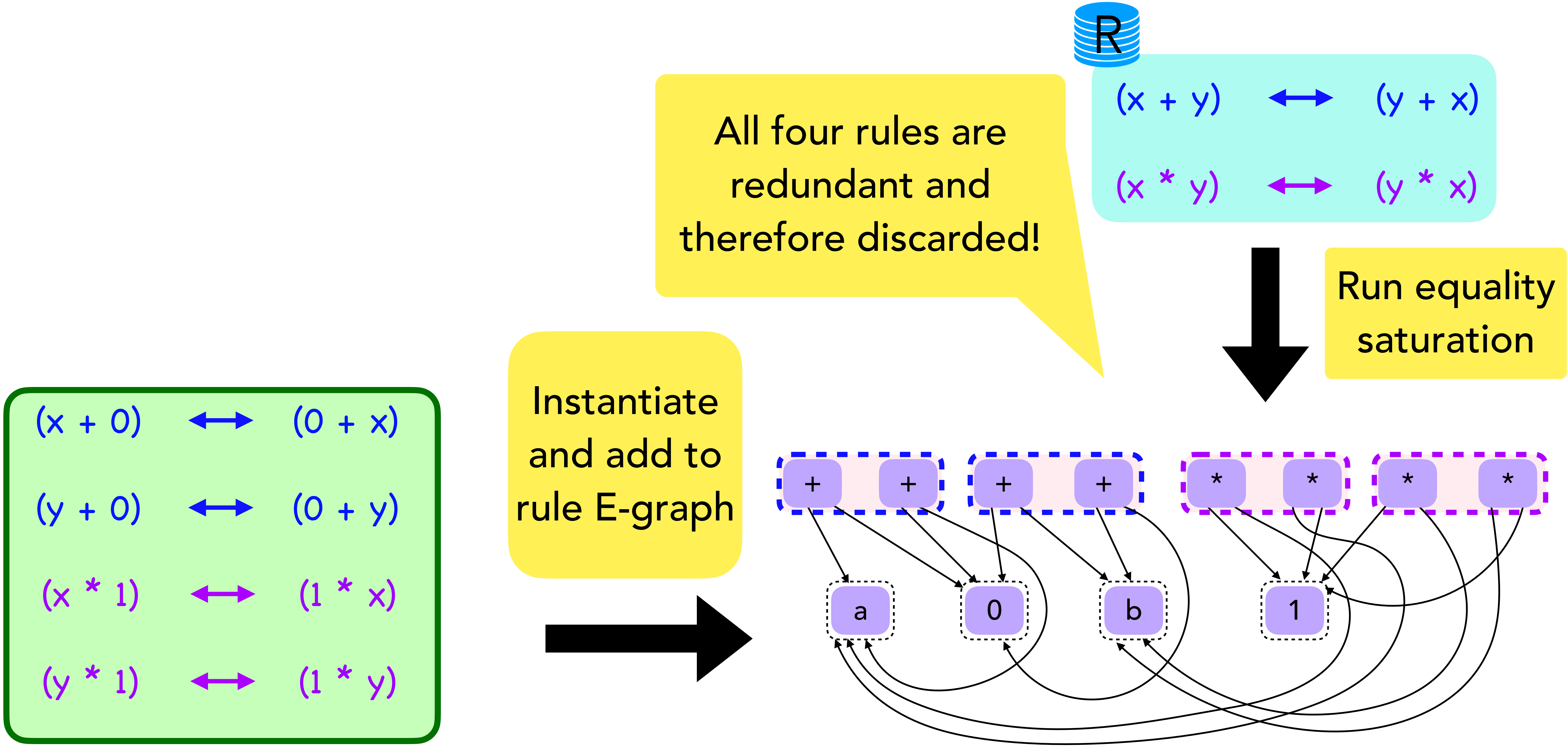
(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

Instantiate and add to rule E-graph



Rule Selection with Equality Saturation



Rule Selection with Equality Saturation

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

(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

Instantiate and add to rule E-graph

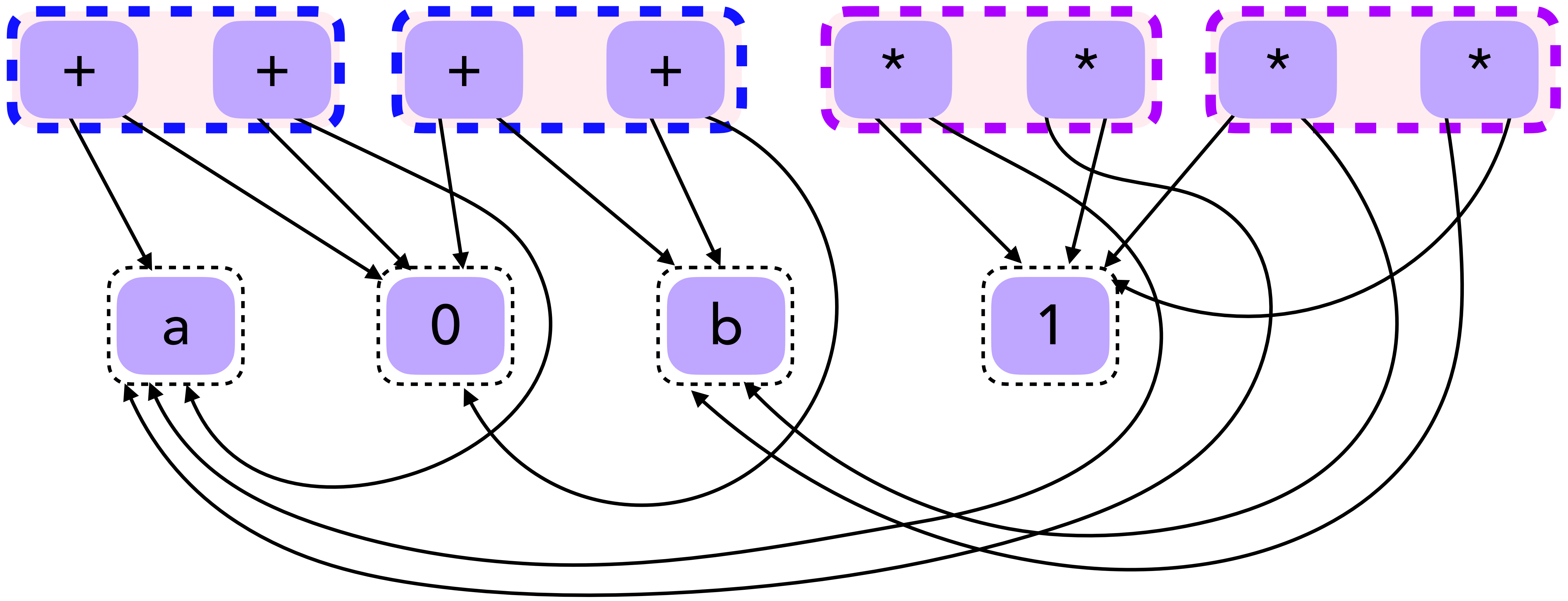
All four rules are redundant and therefore discarded!

R

(x + y) ⟷ (y + x)

(x * y) ⟷ (y * x)

Run equality saturation



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

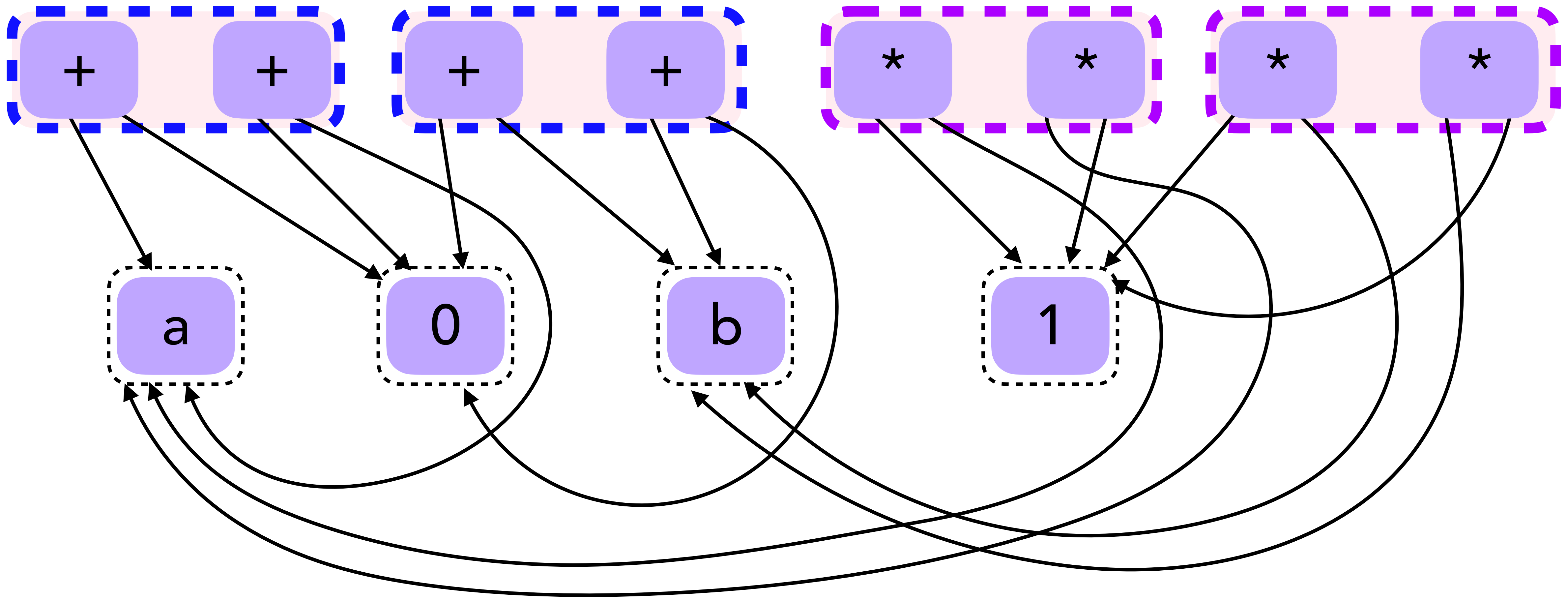
(x + 0) ⟷ (0 + x)

(y + 0) ⟷ (0 + y)

(x * 1) ⟷ (1 * x)

(y * 1) ⟷ (1 * y)

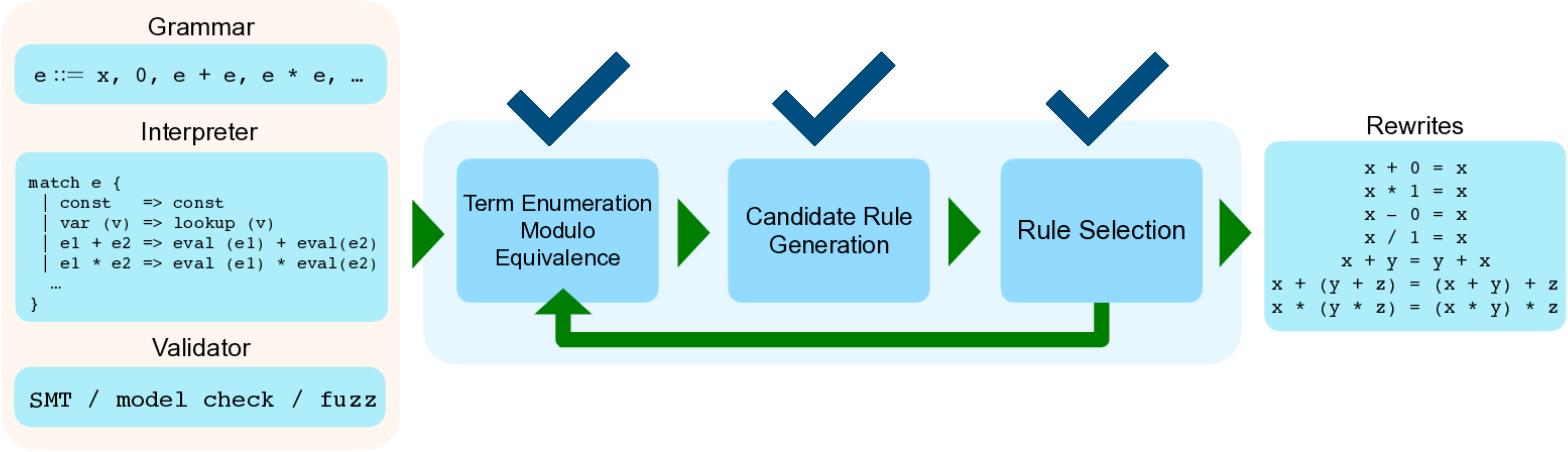
Instantiate and add to rule E-graph



Rule Selection with Equality Saturation

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

Ruler

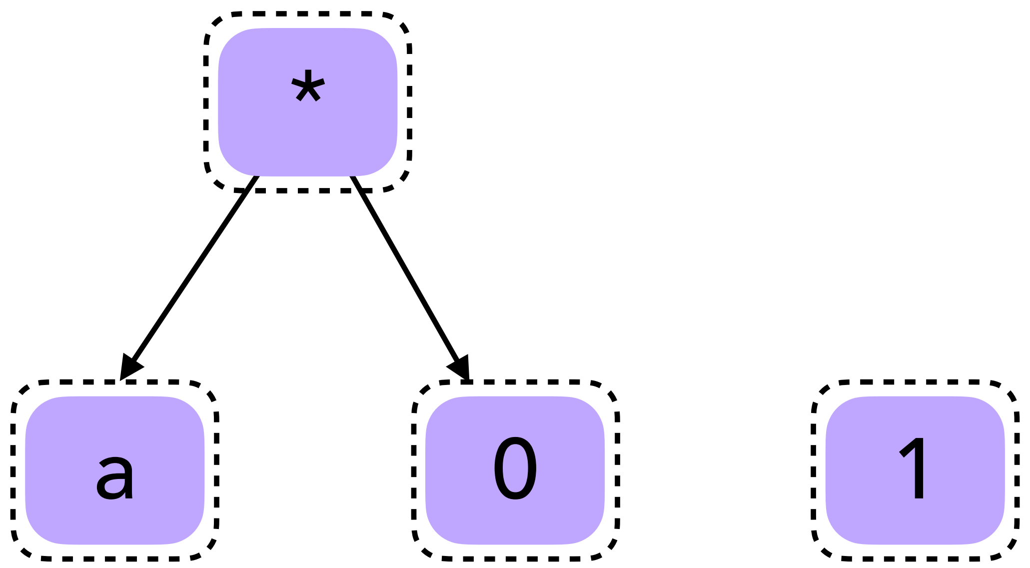

Grammar
e ::= x, 0, e + e, e * e, …
Interpreter
match e {
| const => const
| var (v) => lookup (v)
| e1 + e2 => eval (e1) + eval(e2)
| e1 * e2 => eval (e1) * eval(e2)
…
}
Validator
SMT / model check / fuzz
Term Enumeration Modulo Equivalence
Candidate Rule Generation
Rule Selection
Rewrites
x + 0 = x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



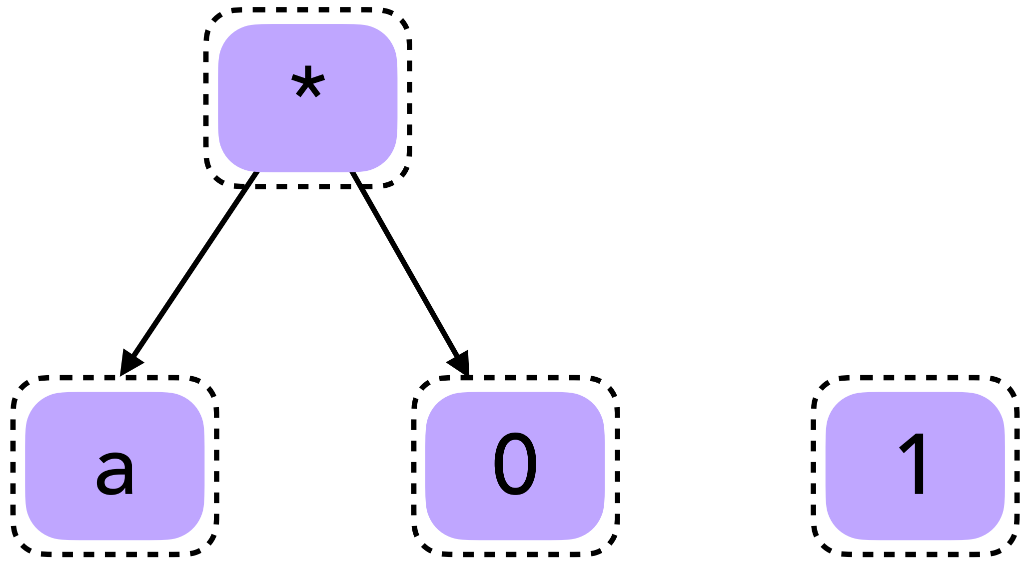
Equality Saturation "*Soundiness*"

Equality Saturation *amplifies* unsoundness!

current ruleset

(y * 0) ⟷ 0

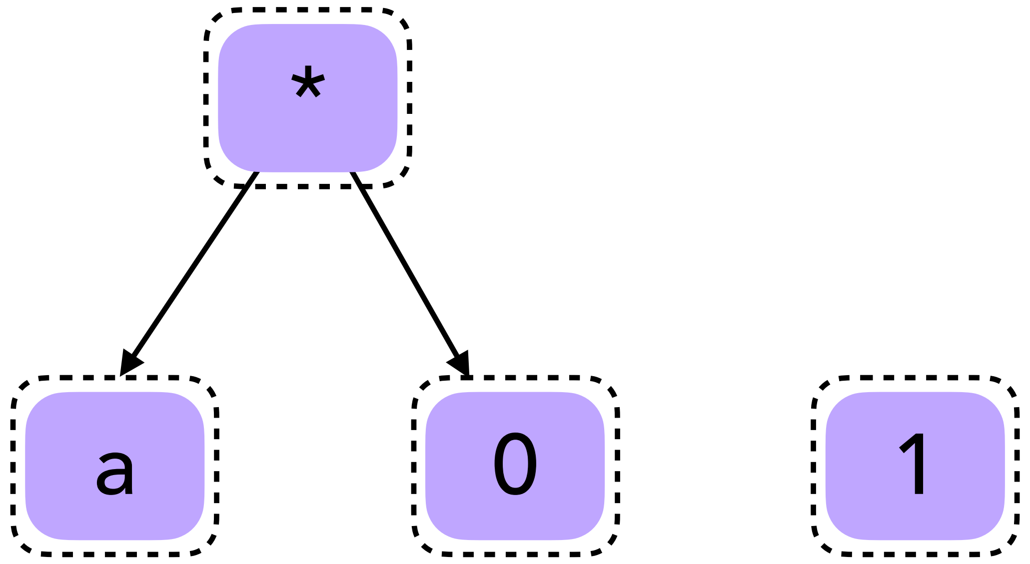
(y * 0) ⟷ 1



Equality Saturation *"Soundiness"*

Equality Saturation *amplifies* unsoundness!

current ruleset



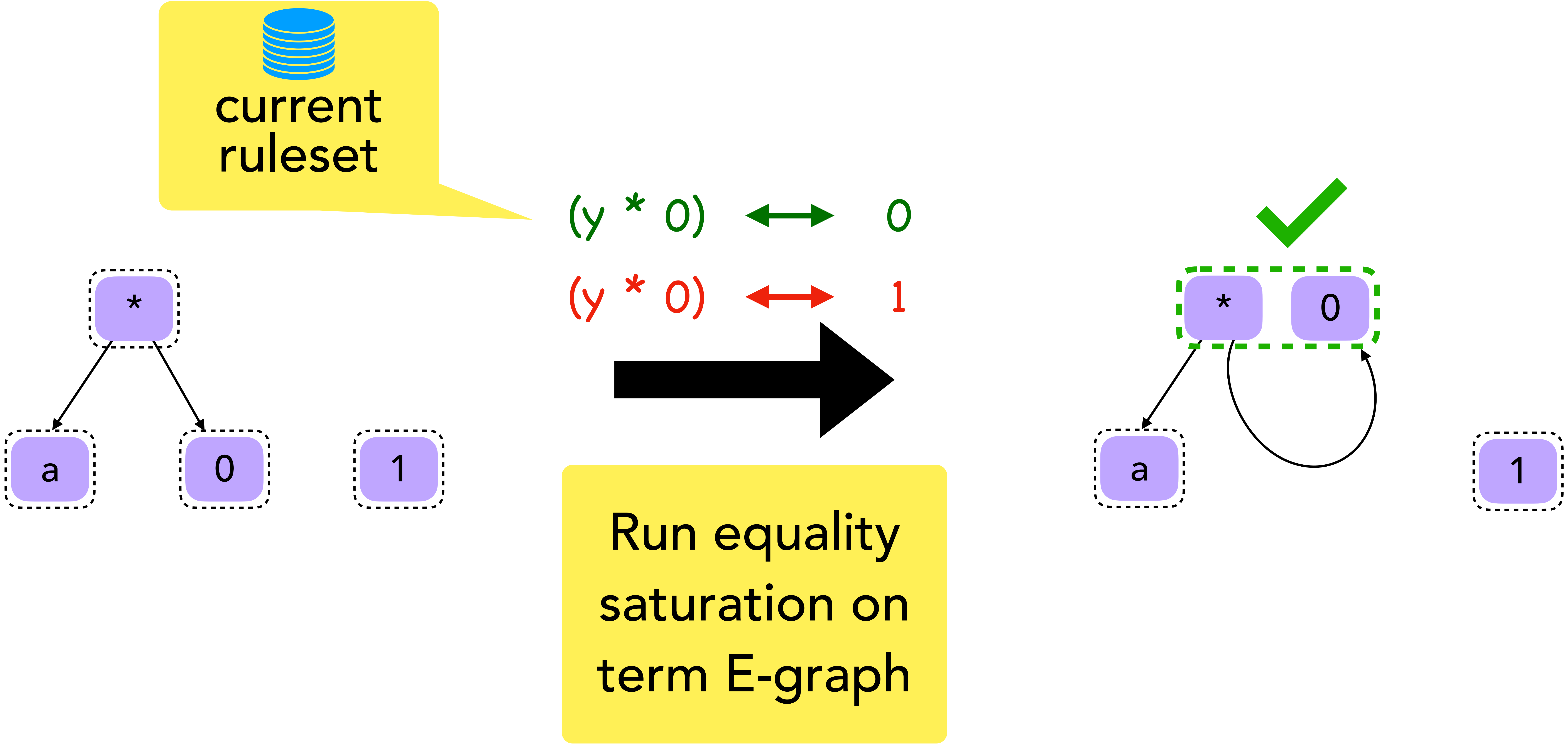
(y * 0) ⟷ 0

(y * 0) ⟷ 1

Run equality saturation on term E-graph

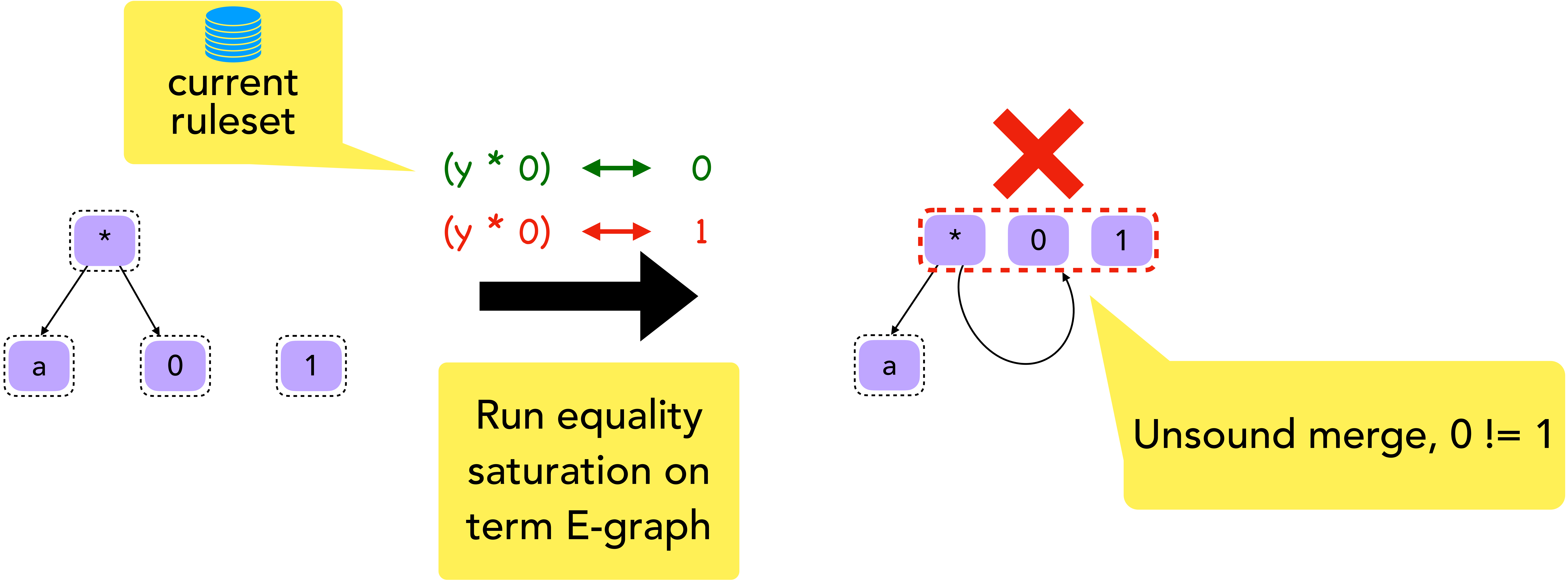
Equality Saturation "*Soundiness*"

Equality Saturation *amplifies* unsoundness!

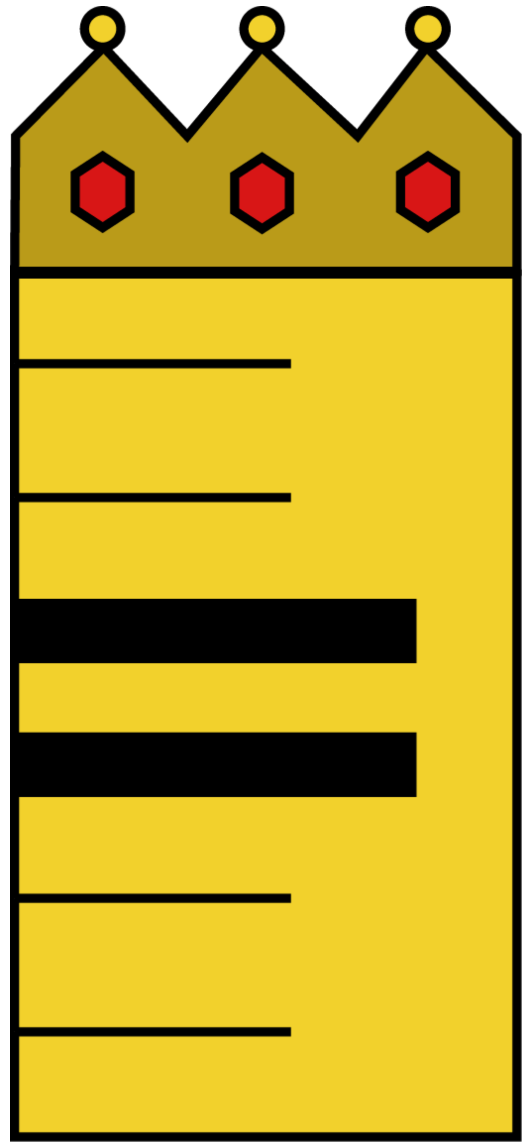


Equality Saturation *"Soundiness"*

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

```
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

`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

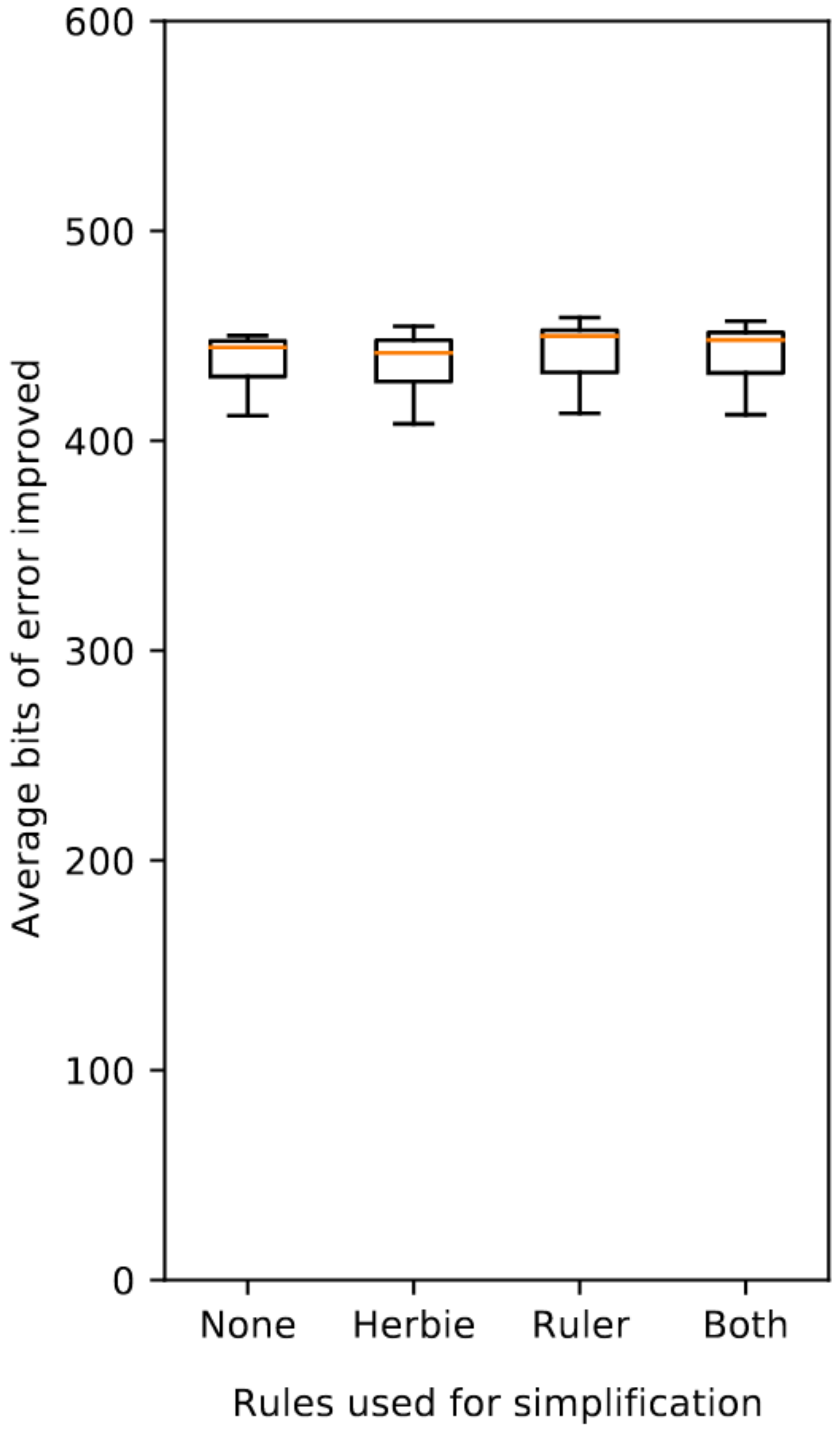
None: Remove all rules

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

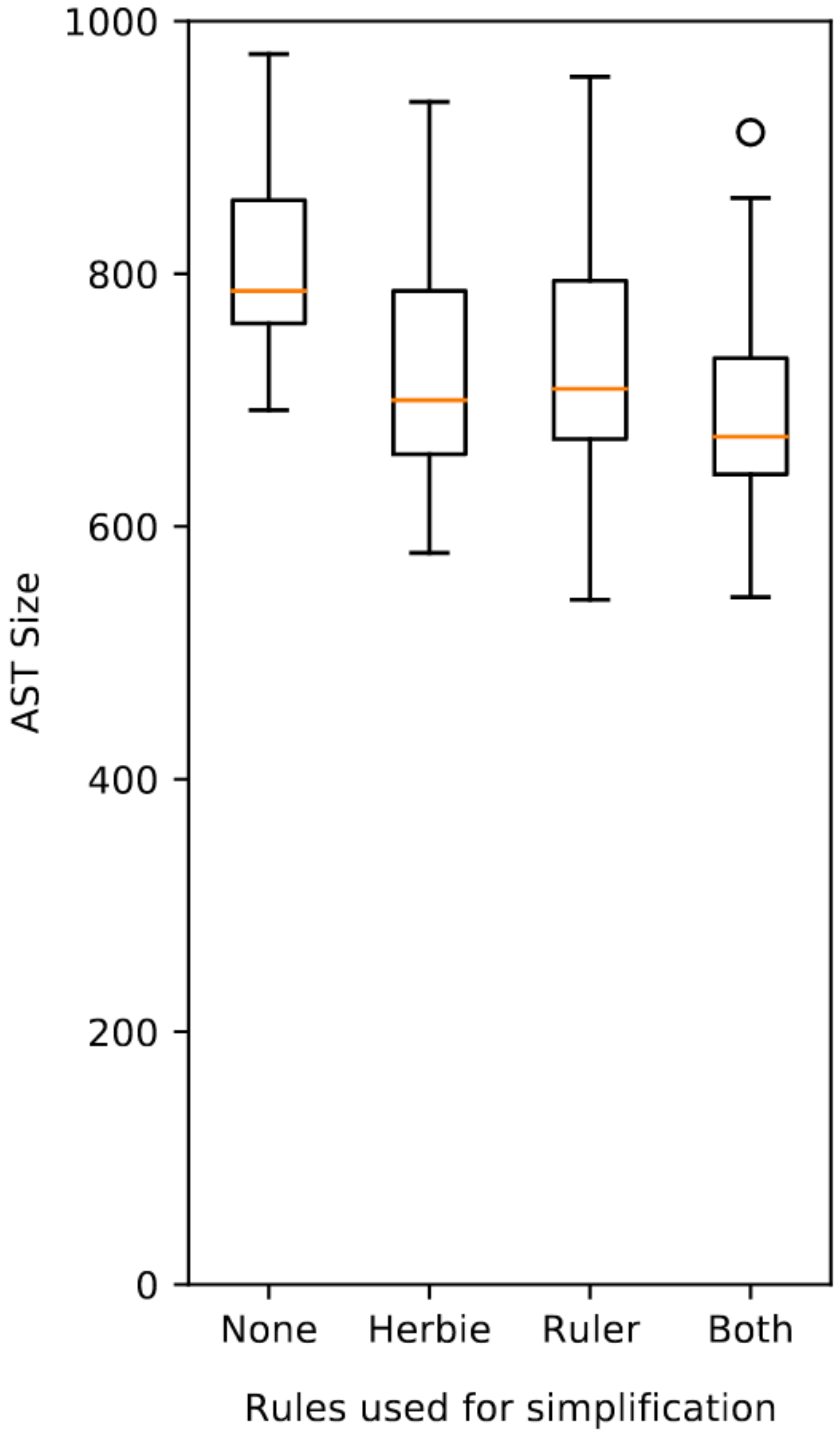
Herbie: Herbie without any changes

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

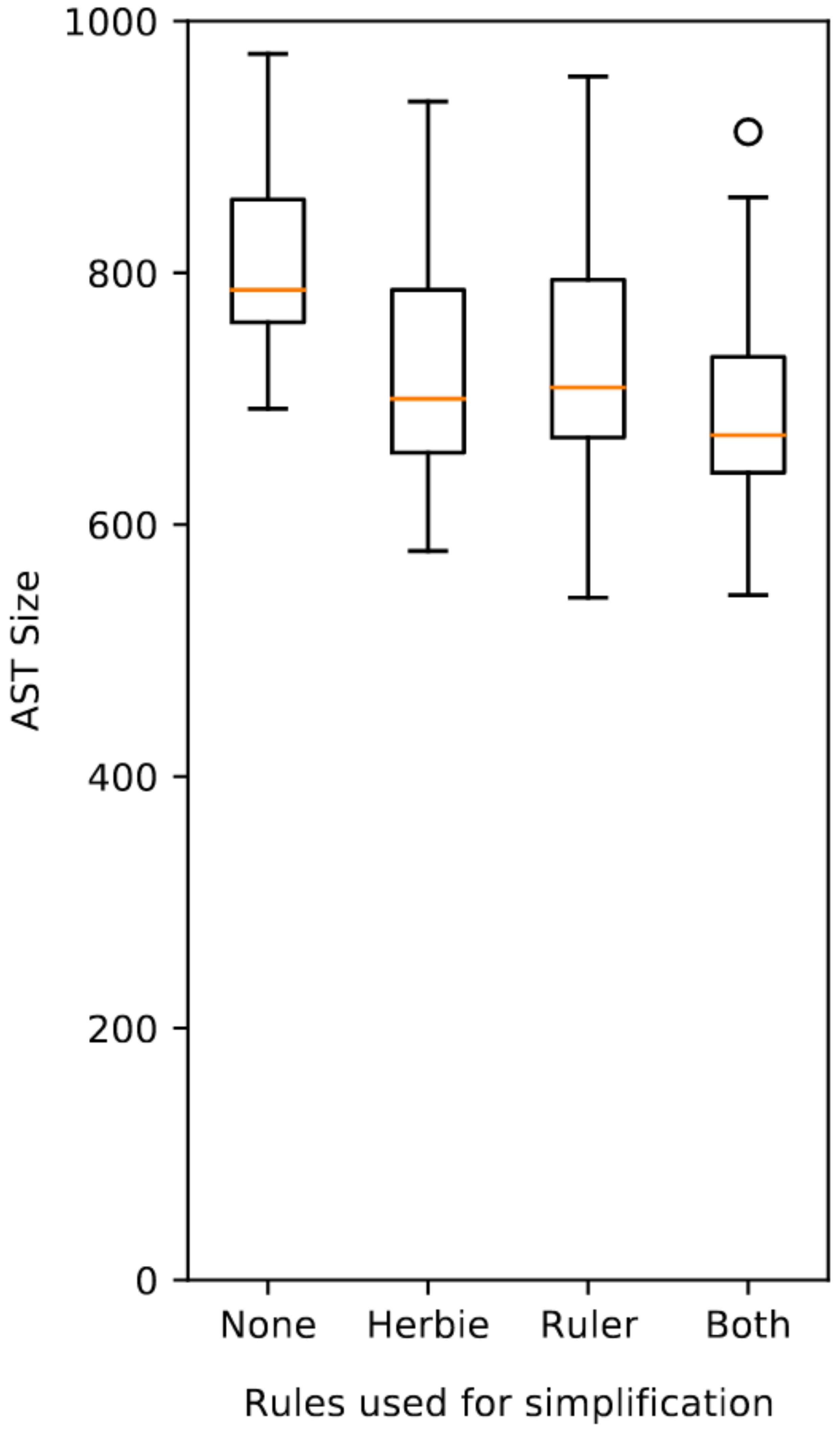
Herbie: Herbie without any changes

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

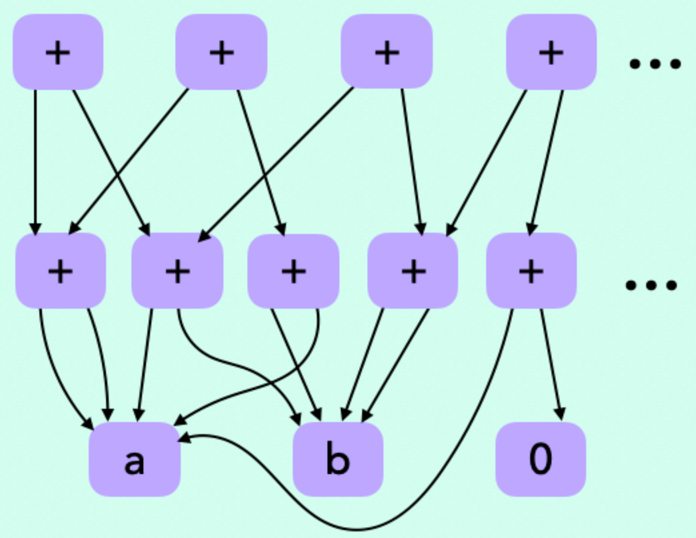
Both: Herbie with both original and Ruler's rules

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

Rewrite Rule Inference Using Equality Saturation

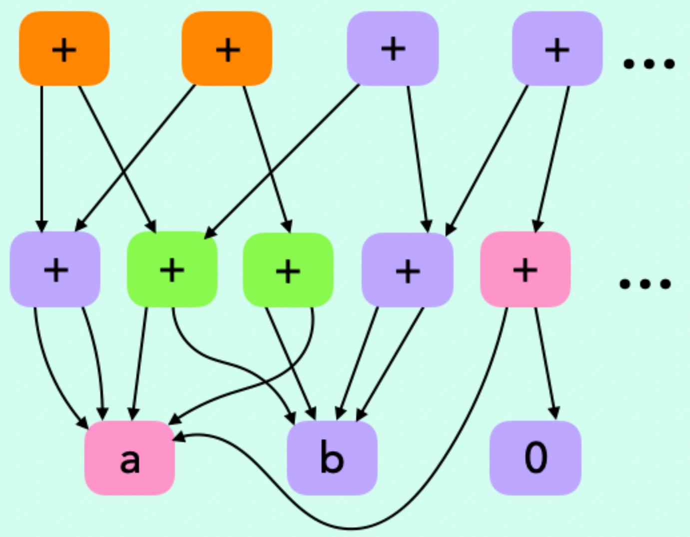
Enumerate terms from a grammar

a, b, 0, +, …



Find candidates: interpret over concrete inputs

"Fingerprints"



Filter candidates to get final ruleset

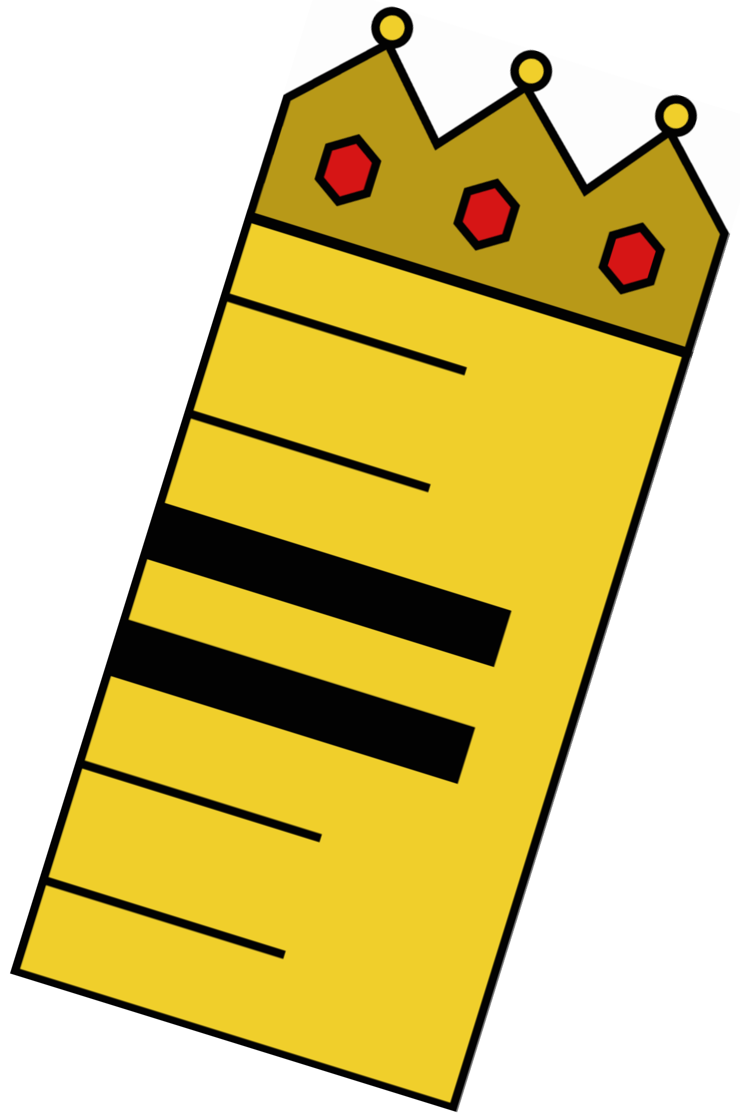
Remove redundant rules

x + 0 ⟷ 0 + x

y + 0 ⟷ 0 + y

x + y ⟷ y + x

Equality Saturation improves all three steps!



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









PLSE

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING