

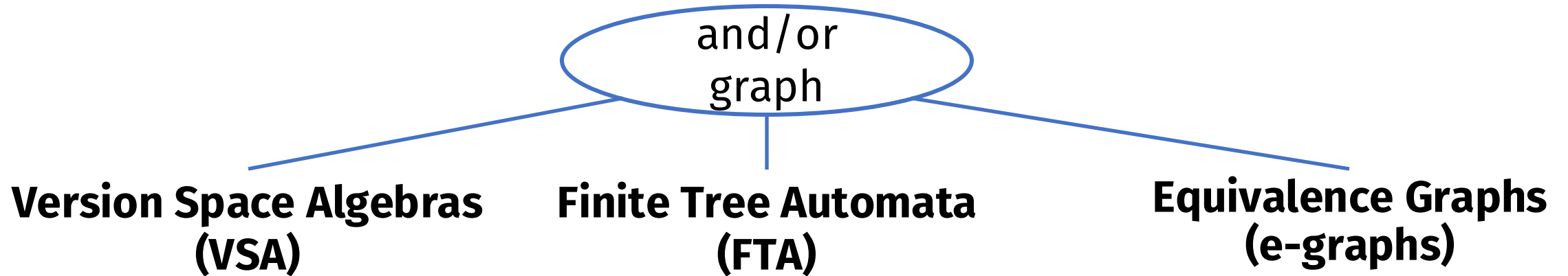
#10: Finite Tree Automata and e-graphs

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis



Representation-based search

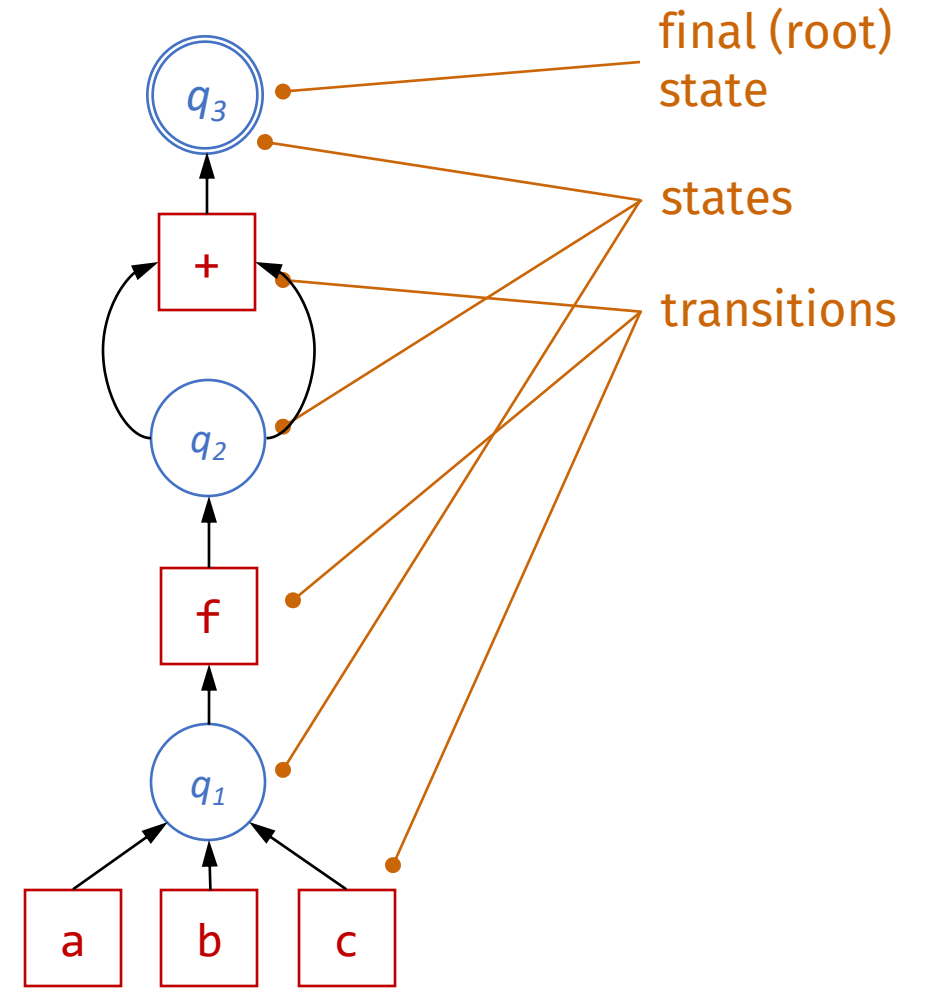
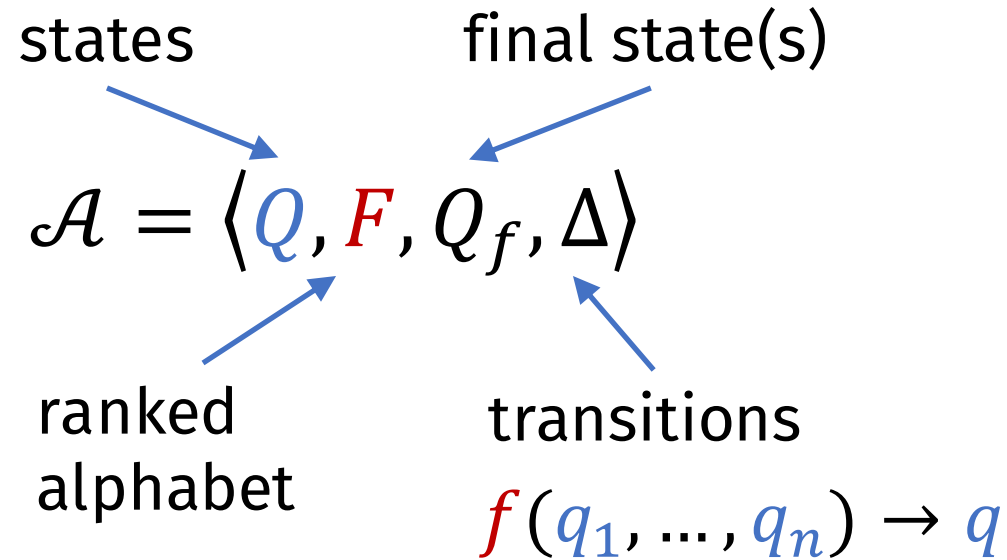


ops: learn-1, intersect, extract

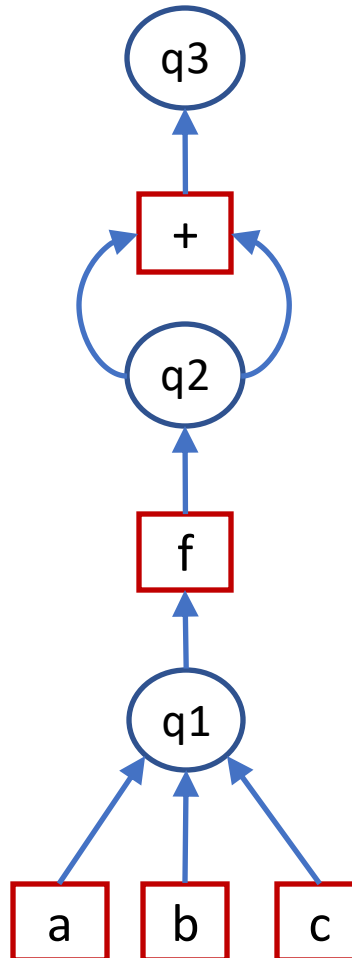
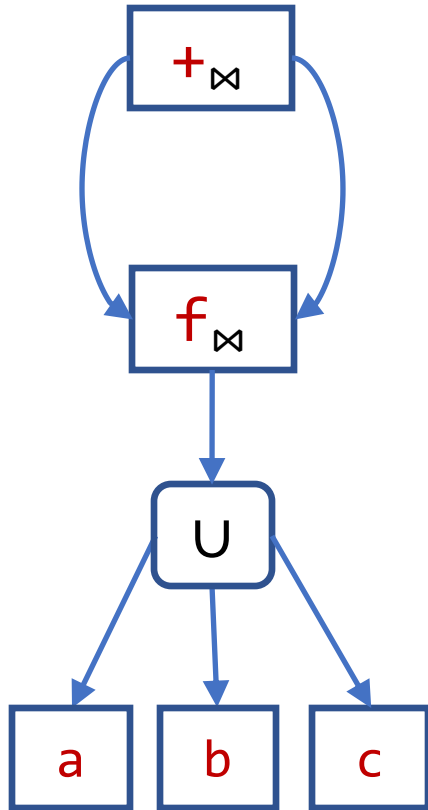
DSL: efficiently invertible

similar to: top-down prop,
but can infer constants

Finite Tree Automata



VSA vs FTA



- Both are and-or graphs
- FTA state = VSA union node
 - in VSAs singleton joins are omitted
- FTA transition = VSA join node

FTA-based search

- **Synthesis of Data Completion Scripts using Finite Tree Automata**

Xinyu Wang, Isil Dillig, Rishabh Singh, *OOPSLA'17*

- **Program Synthesis using Abstraction Refinement**

Xinyu Wang, Isil Dillig, Rishabh Singh, *POPL'18*

- **Searching Entangled Program Spaces**

James Koppel, Zheng Guo, Edsko de Vries, Armando Solar-Lezama, Nadia Polikarpova. *ICFP'22*

FTA-based search

- **Synthesis of Data Completion Scripts using Finite Tree Automata**
Xinyu Wang, Isil Dillig, Rishabh Singh, *OOPSLA'17*
- **Program Synthesis using Abstraction Refinement**
Xinyu Wang, Isil Dillig, Rishabh Singh, *POPL'18*
- **Searching Entangled Program Spaces**
James Koppel, Zheng Guo, Edsko de Vries, Armando Solar-Lezama, Nadia Polikarpova. *ICFP'22*

Example

Grammar

$N ::= \text{id}(V) \mid N + T \mid N * T$

$T ::= 2 \mid 3$

$V ::= x$

Spec

$1 \rightarrow 9$

PBE with Finite Tree Automata

$\langle A, \mathbb{Z} \rangle$

$A \in \{\text{N}, \text{T}, \text{X}\}$

$\{\langle \text{N}, 9 \rangle\}$

state

final states

$\mathcal{A} = \langle Q, F, Q_f, \Delta \rangle$

alphabet

transitions

$\text{id}, +, *$

$f(q_1, \dots, q_n) \rightarrow q$

$+(\langle \text{N}, 1 \rangle, \langle \text{T}, 2 \rangle) \rightarrow \langle \text{N}, 3 \rangle$

...

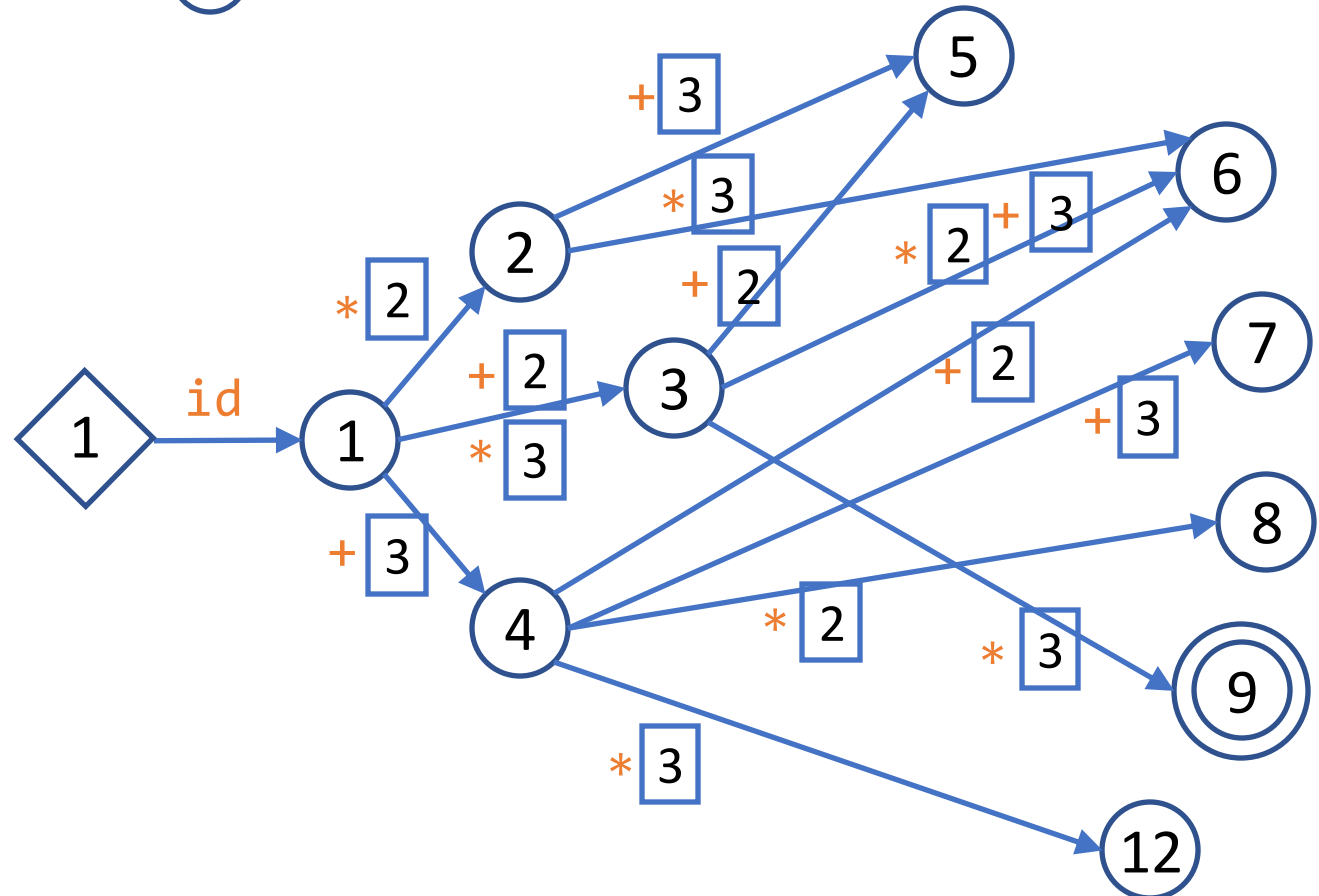
PBE with Finite Tree Automata

$N ::= \text{id}(V) \mid N + T \mid N * T \quad \bigcirc$

$T ::= 2 \mid 3 \quad \square$

$V ::= x \quad \diamond$

$1 \rightarrow 9$



Discussion

- What do FTAs remind you of in the enumerative world?
 - FTA ~ bottom-up search with OE
- How are they different?
 - More size-efficient: sub-terms in the bank are replicated, while in the FTA they are shared
 - Hence, can store all terms, not just one representative per class
 - Can construct one FTA per example and intersect
 - More incremental in the CEGIS context!

FTA-based search

- **Synthesis of Data Completion Scripts using Finite Tree Automata**
Xinyu Wang, Isil Dillig, Rishabh Singh, *OOPSLA'17*
- **Program Synthesis using Abstraction Refinement**
Xinyu Wang, Isil Dillig, Rishabh Singh, *POPL'18*
- **Searching Entangled Program Spaces**
James Koppel, Zheng Guo, Edsko de Vries, Armando Solar-Lezama, Nadia Polikarpova. *ICFP'22*

Abstract FTA

- **Challenge:** FTA still has too many states
- Idea:
 - instead of one state = one value
 - we can do one state = set of values (= abstract value)

Abstract FTA

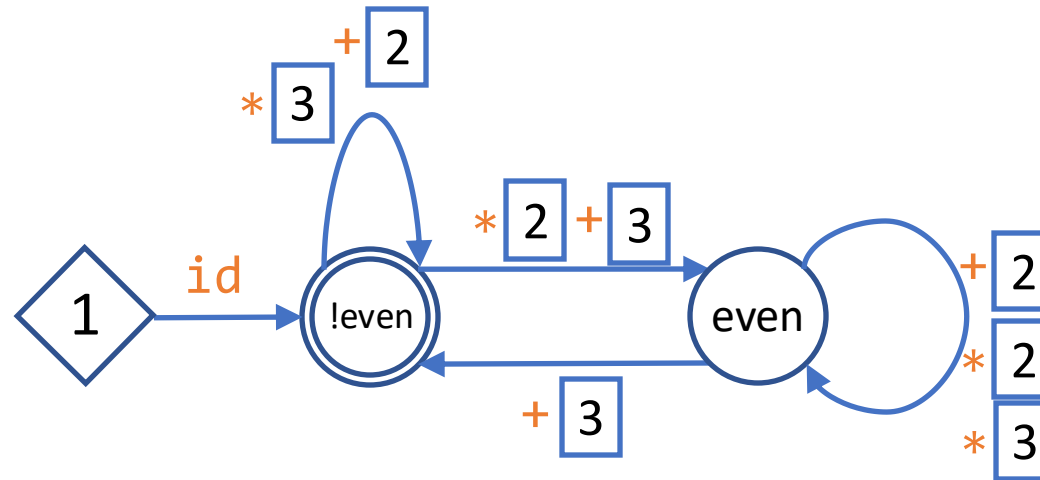
[Wang, Dillig, Singh POPL'18]

$N ::= \text{id}(V) \mid N + T \mid N * T \quad \bigcirc$

$T ::= 2 \mid 3 \quad \square$

$V ::= x \quad \diamond$

$1 \rightarrow 9$



What now?

- idea 1: enumerate from reduced space
- idea 2: refine abstraction!

Abstract FTA

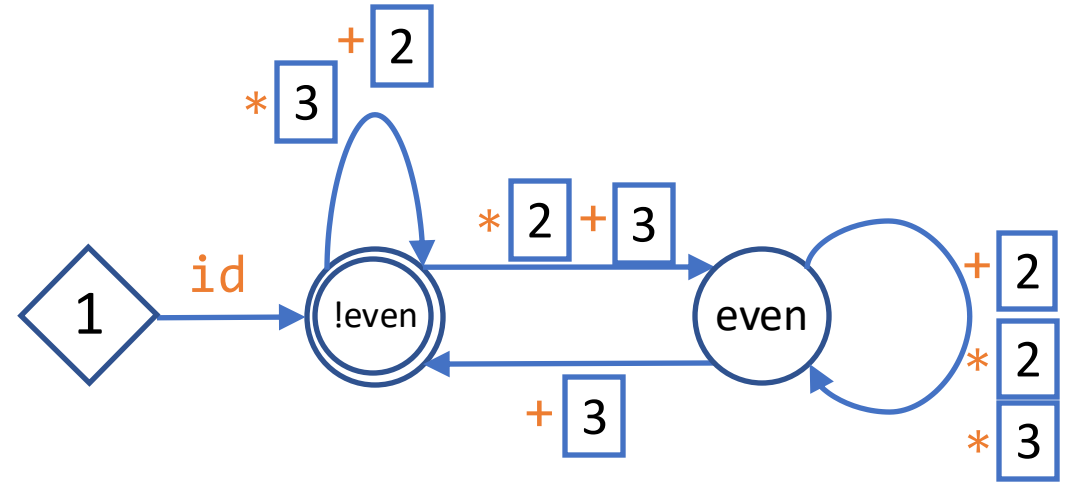
$N ::= \text{id}(V) \mid N + T \mid N * T \quad \bigcirc$

$T ::= 2 \mid 3 \quad \square$

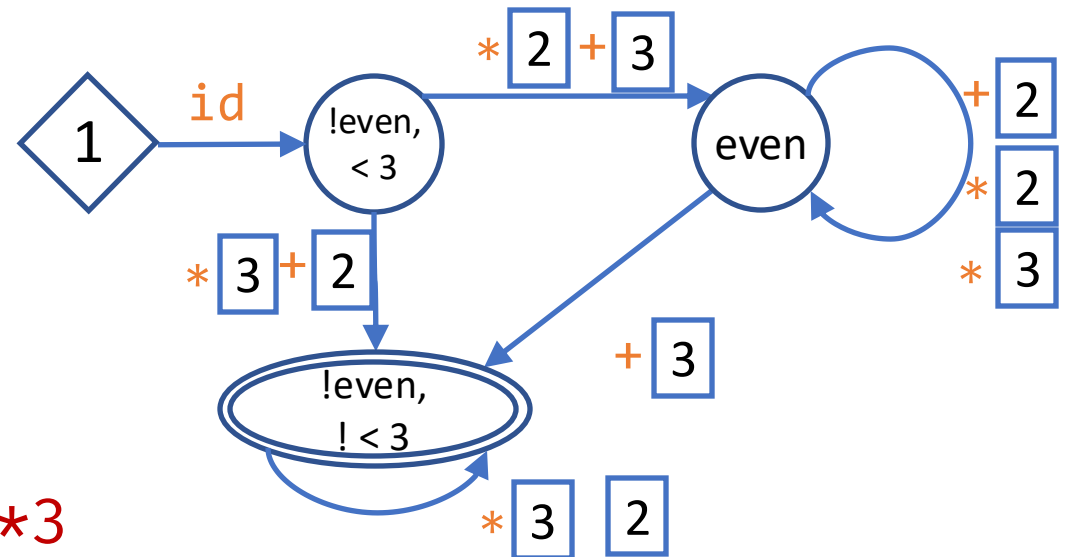
$V ::= x \quad \diamond$

$1 \rightarrow 9$

Predicates: {even, < 3, ...}

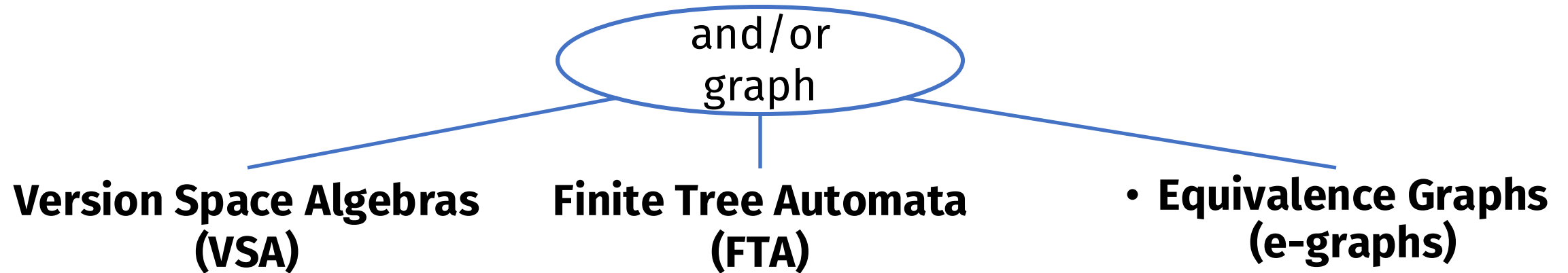


solution: $\text{id}(x)$



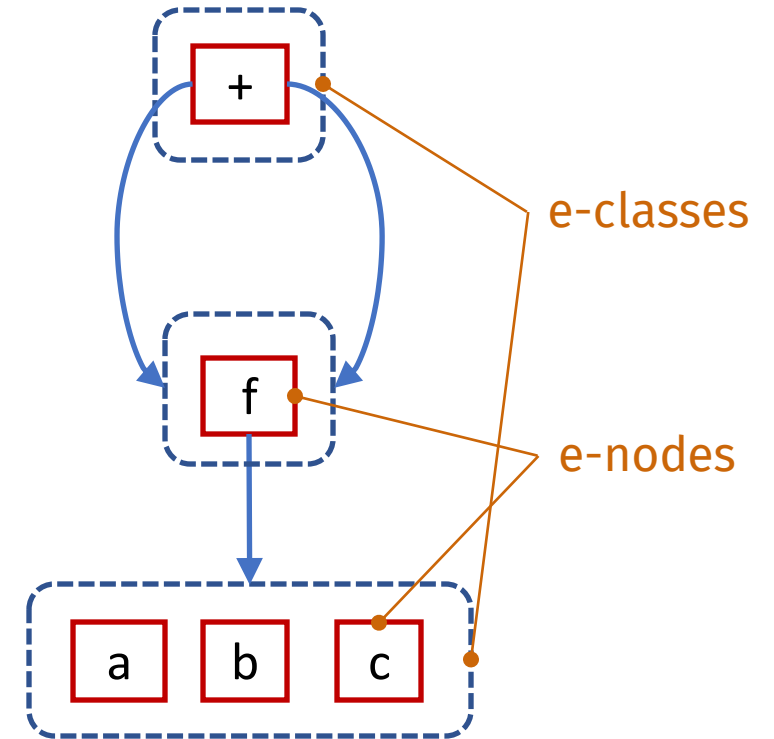
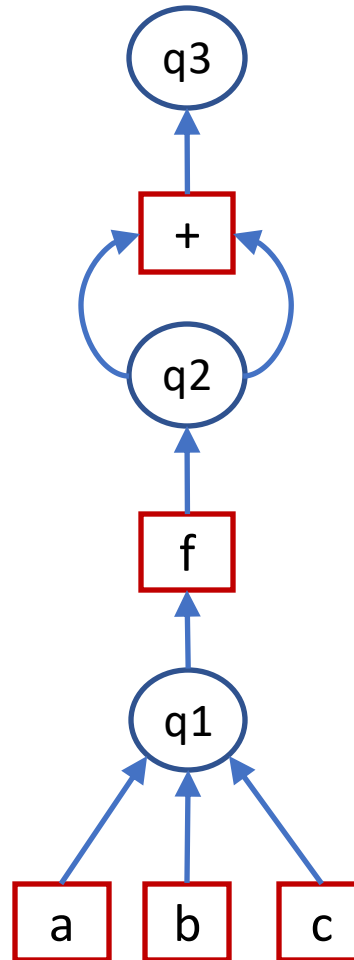
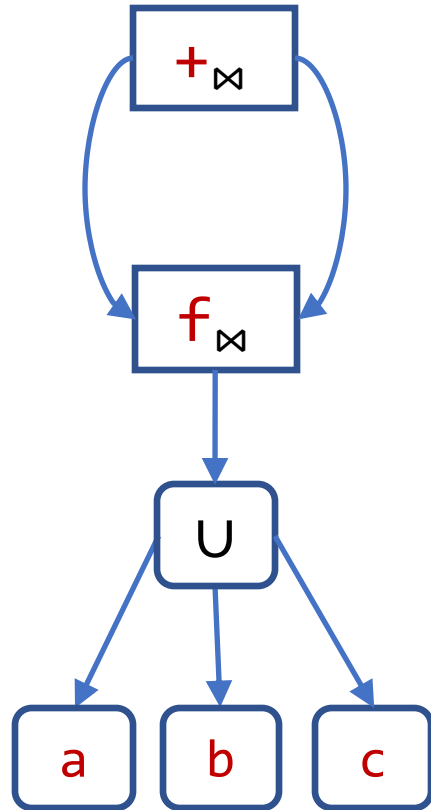
solution: $\text{id}(x) * 3$

Representation-based search



ops: learn-1, intersect, extract	ops: learn-1, intersect, extract
DSL: efficiently invertible	DSL: efficiently enumerable
similar to: top-down prop, but can infer constants	similar to: bottom-up with OE, but can store all programs (and add examples incrementally)
state: represents a set of observationally-equivalent programs	

VSA vs FTA vs E-Graphs



Program search with e-grpahs

- **Equality saturation: a new approach to optimization**
Ross Tate, Michael Stepp, Zachary Tatlock, Sorin Lerner, *POPL'09*
- **egg: Fast and Extensible Equality Saturation**
Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, Pavel Panchekha, *POPL'21*
- **Semantic Code Search via Equational Reasoning**
Varot Premtoon, James Koppel, Armando Solar-Lezama. *PLDI'20*

Equality saturation

Program optimization via
rewriting:

$$\begin{aligned} & (a * 2) / 2 \\ \Rightarrow & a * (2 / 2) \\ \Rightarrow & a * 1 \\ \Rightarrow & a \end{aligned}$$

useful rules:

$$\begin{aligned} (x * y) / z &= x * (y / z) \\ x / x &= 1 \\ x * 1 &= x \end{aligned}$$

not so useful:

$$\begin{aligned} x * 2 &= x \ll 1 \\ x * y &= y * x \end{aligned}$$

Challenge: which ones to apply and in what order?

Idea: all of them all the time!

Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

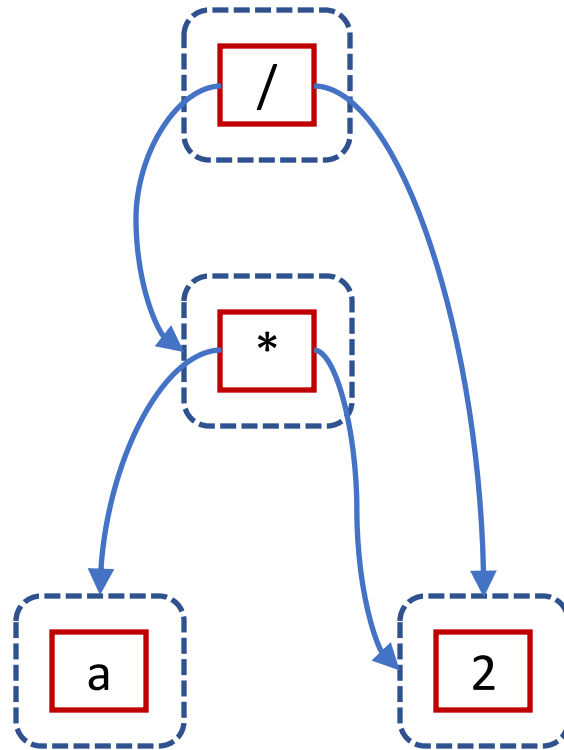
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

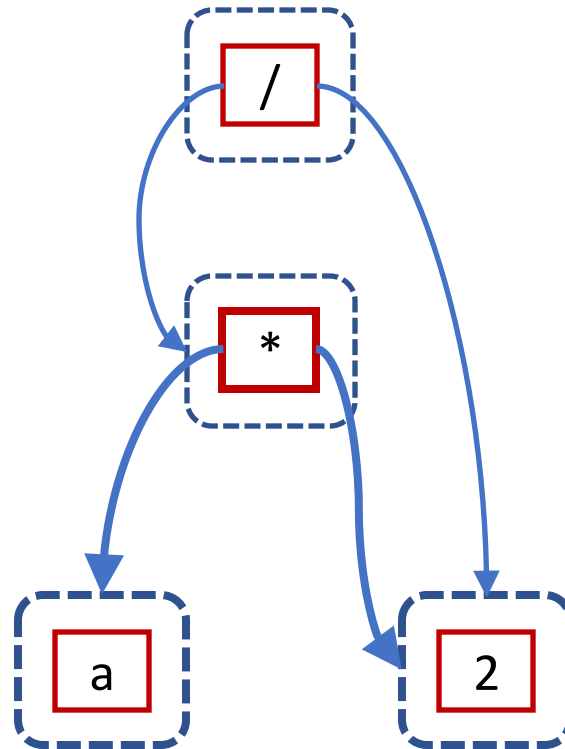
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

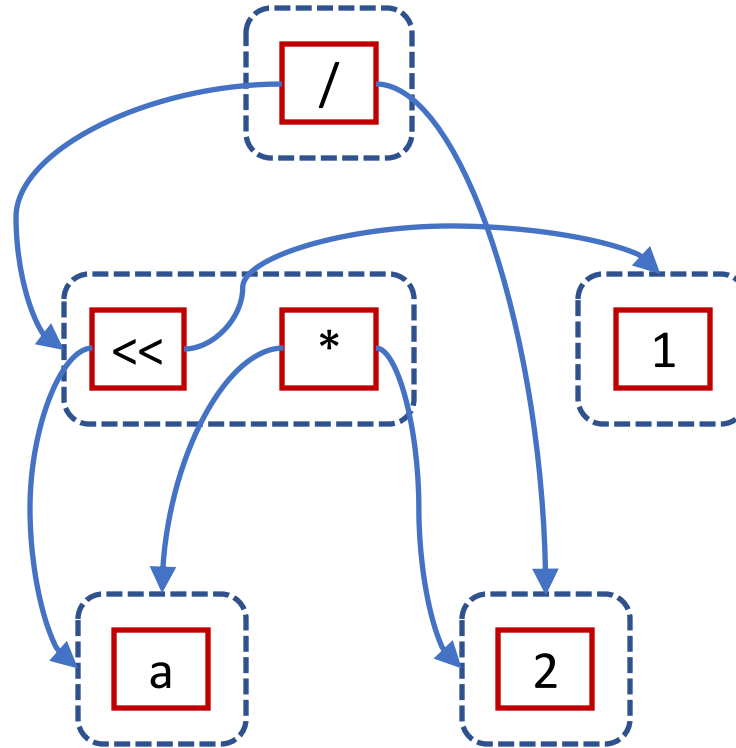
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

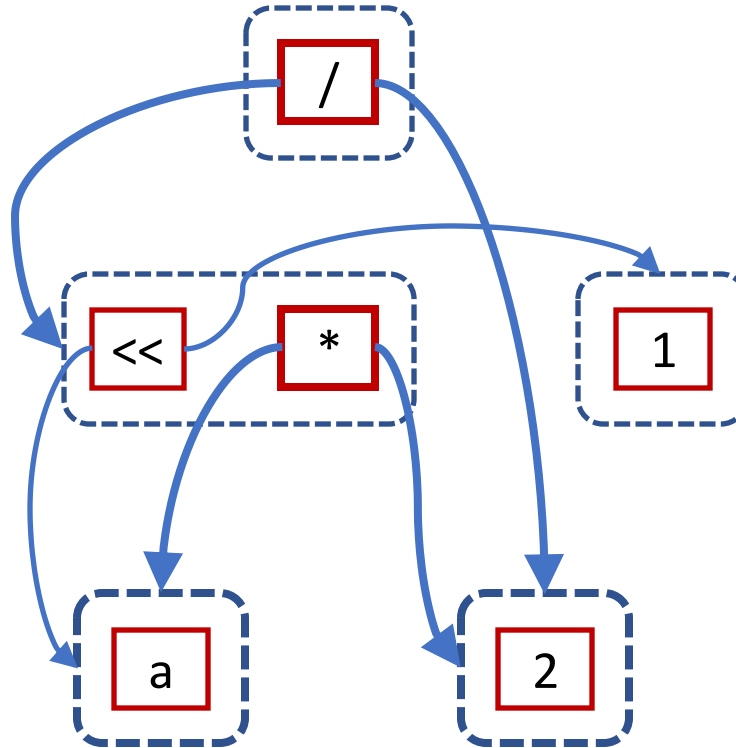
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

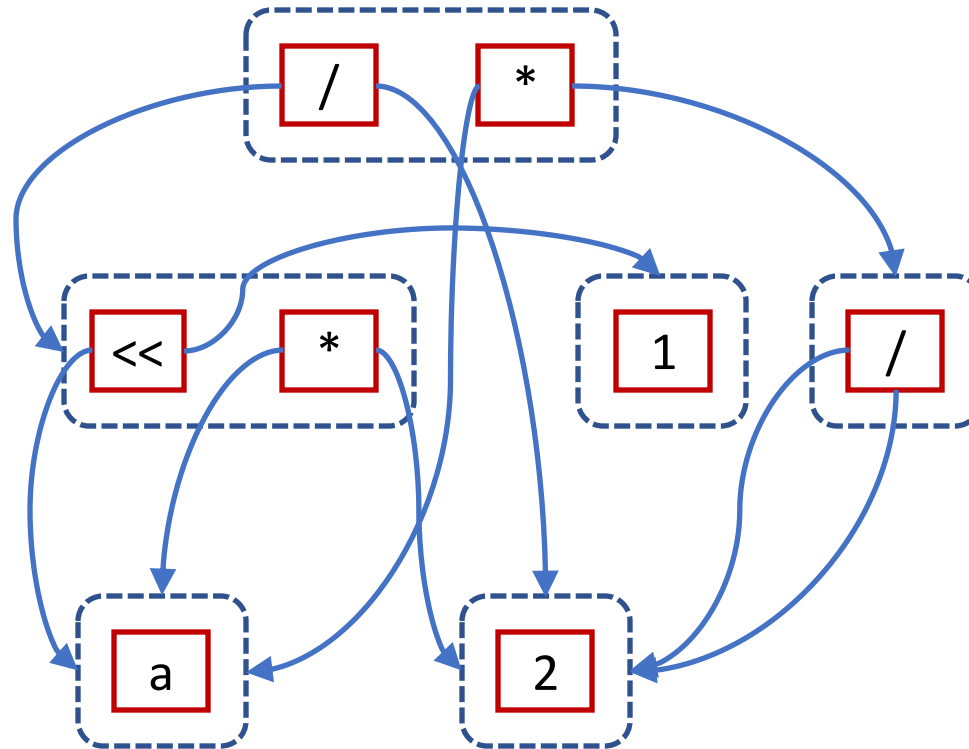
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

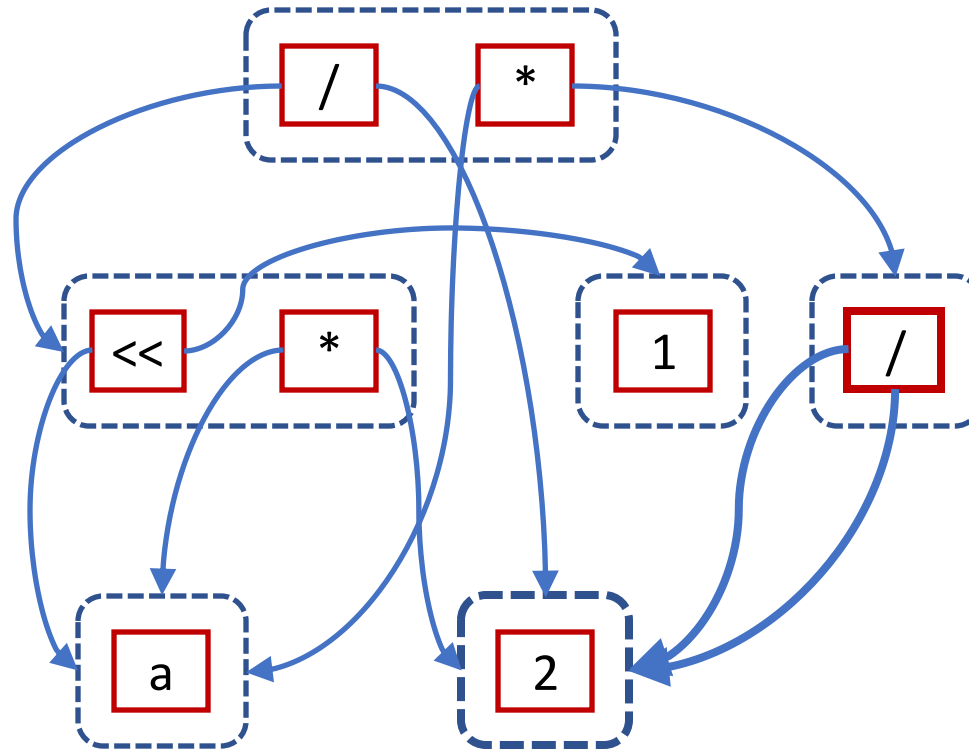
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

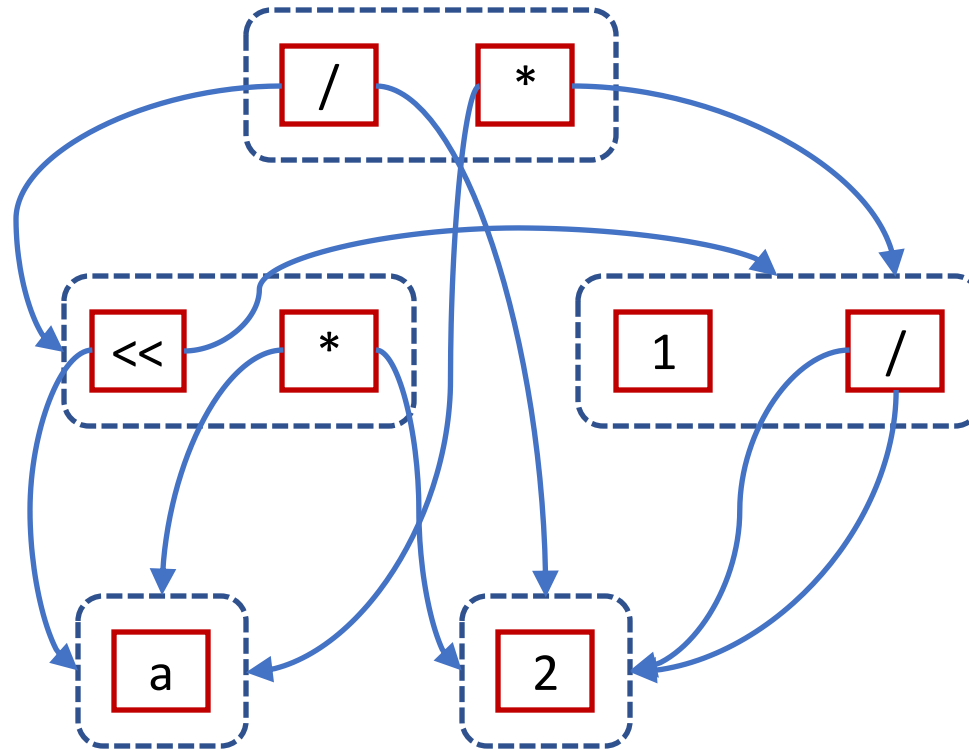
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

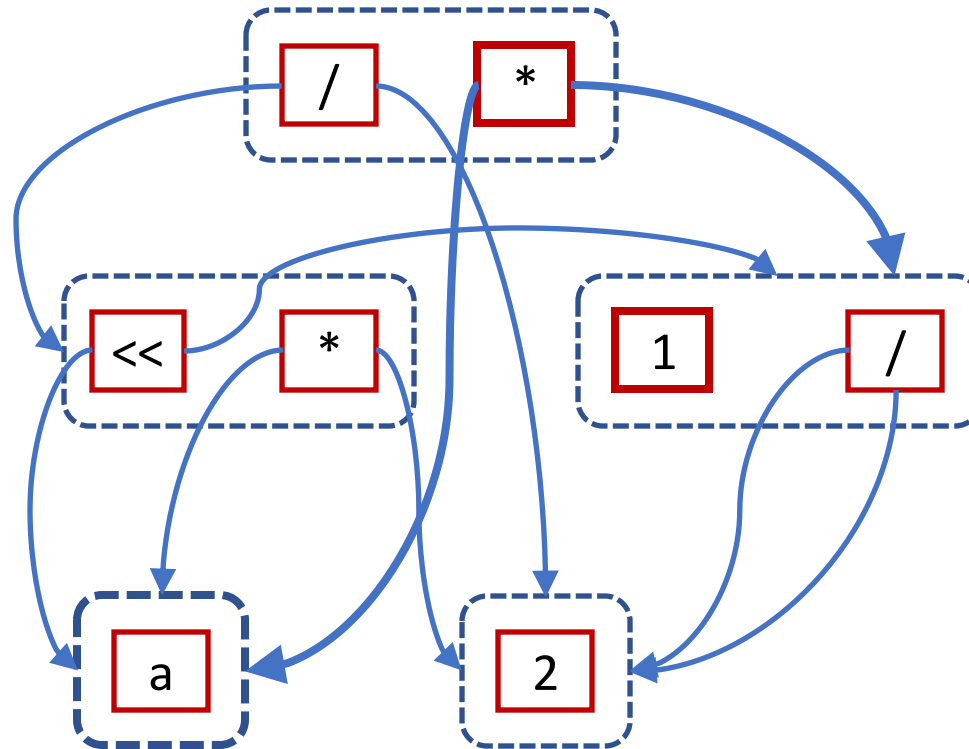
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Equality saturation

Initial term: $(a * 2) / 2$

Rewrite rules:

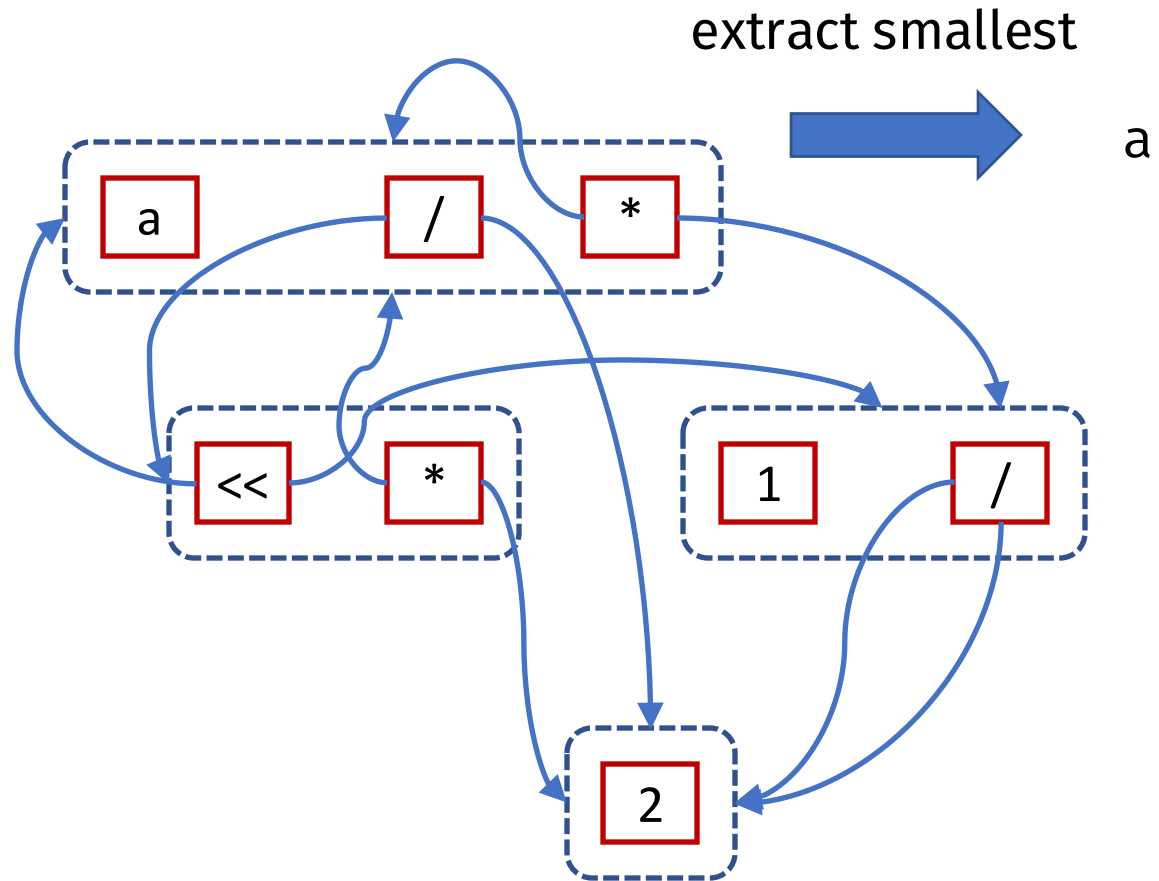
$$(x * y) / z = x * (y / z)$$

$$x / x = 1$$

$$x * 1 = x$$

$$x * 2 = x \ll 1$$

$$x * y = y * x$$



Representation-based search

