CS5733 Program Synthesis

#10.Representation Based Search-II

Ashish Mishra, September 03, 2024

Recap

- Representation-Based Search

Idea:

1. build a data structure that compactly represents good parts of the program space
2. extract solution from that data structure

- Compact representation of the search space:
    - And/Or Graphs
        - Version Spaces and Algebra :
            - Join and Union Nodes
            - FlashFill

VSAs Again

Version Space Formulation

Hypothesis space H

- Space of possible functions $In \rightarrow Out$

Version Space $VS_{H,D} \subseteq H$

- H is the original hypothesis space
- D is a set of examples i_j, o_j
- $h \in VS_{H,D} \Leftrightarrow \forall\ i, o \in D \;\; h(i) = \mathrm{o}$

Hypothesis space provides *restriction bias*

- Defines what functions one is allowed to consider
- *Preference bias* needs to be provided independently

Partial Orders

Set P

Partial order ≤ such that ∀x,y,z∈P

- x ≤ x (reflexive)
- x ≤ y and y ≤ x implies x = y (asymmetric)
- x ≤ y and y ≤ z implies x ≤ z (transitive)

Can use partial order to define

- Upper and lower bounds
- Least upper bound
- Greatest lower bound

Upper Bounds

If $S \subseteq P$ then

- $x \in P$ is an upper bound of S if $\forall y \in S.\ y \leq x$
- $x \in P$ is the least upper bound of S if
  - x is an upper bound of S, and
  - $x \leq y$ for all upper bounds y of S
- $\vee$ - join, least upper bound, lub, supremum, sup
  - $\vee S$ is the least upper bound of S
  - $x \vee y$ is the least upper bound of $\{x,y\}$
- Often written as $\sqcup$ as well

Lower Bounds

If $S \subseteq P$ then

- $x \in P$ is a lower bound of S if $\forall y \in S.\ x \leq y$
- $x \in P$ is the greatest lower bound of S if
  - x is a lower bound of S, and
  - $y \leq x$ for all lower bounds y of S
- $\wedge$ - meet, greatest lower bound, glb, infimum, inf
  - $\wedge S$ is the greatest lower bound of S
  - $x \wedge y$ is the greatest lower bound of $\{x,y\}$

- Often written as $\sqcap$ as well

Lattices

If $x \wedge y$ and $x \vee y$ exist for all $x,y \in P$

then P is a lattice

If $\wedge S$ and $\vee S$ exist for all $S \subseteq P$

then P is a complete lattice

All finite lattices are complete

Example of a lattice that is not complete

- Integers I
- For any $x, y \in I$, $x \vee y = \max(x,y)$, $x \wedge y = \min(x,y)$
- But $\vee$ I and $\wedge$ I do not exist
- $I \cup \{+\infty, -\infty\}$ is a complete lattice

Partial Ordering of hypothesis

Partial order $h_1 \sqsubseteq h_2$

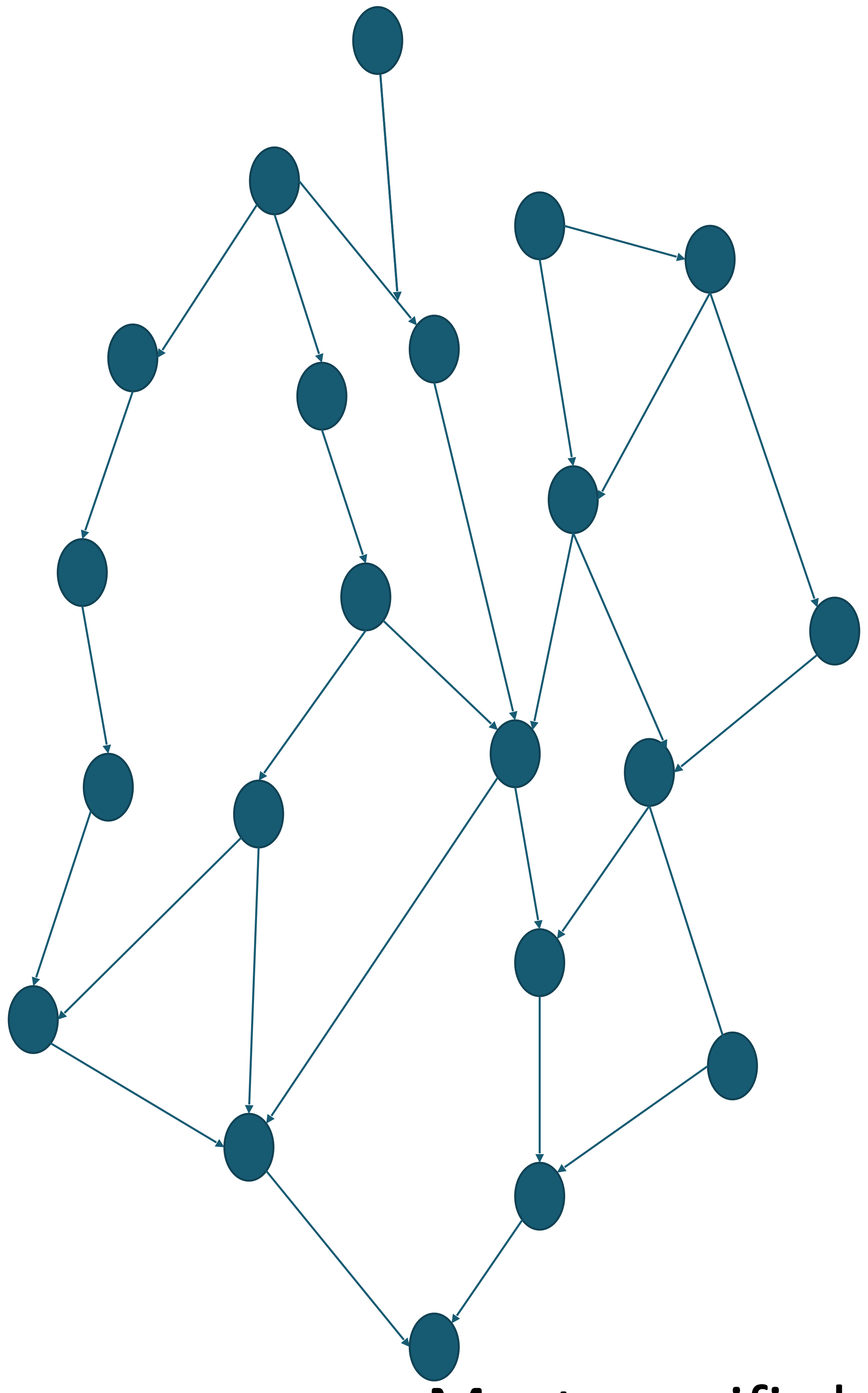
- h_2 is “better” than h_1

Ex: For boolean hypothesis

- “better” == more general
- $h_1 \sqsubseteq h_2 \Leftrightarrow (h_1 \Rightarrow h_2)$

For booleans, VS forms a lattice

- $h_1, h_2 \in VS \Rightarrow h_1 \sqcap h_2 = h_1 \wedge h_2 \in VS$



Most specific hypothesis that satisfies the observations

Boundary set representable

You can represent a VS by the pair (G,S) where

- G is most general hypothesis (i.e. $\top$)
- S is the most specific (i.e. $\bot$)

Applies in general when hypothesis space is partially ordered and version space is a lattice

Example: FindSuffix(T)

FS_T: move to the position right before the next occurrence of T.

We shall go on to the end. We |shall fight in France, we |shall fight on the seas and oceans, we shall fight with growing confidence and growing strength in the air,…

$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,\ we\ shall\ fight\ldots"}$

Example: FindSuffix

FS_T: move to the position right before the next occurrence of T.

$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,\ we\ shall\ fight\ldots"}$

We shall go on to the end. We |shall fight in France, we |shall fight on the seas and oceans, we |shall fight with growing confidence and growing strength in the air,…

Example: FindSuffix

FS_T: move to the position right before the next occurrence of T.

$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,\ we\ shall\ fight\ldots"}$

We shall go on to the end. We |shall fight in France, we |shall fight on the seas and oceans, we |shall fight with growing confidence and growing strength in the air,…

Example: FindSuffix

FS_T: move to the position right before the next occurrence of T.

We shall go on to the end. We | shall fight in France, we | shall fight on the seas and oceans, we | shall fight with growing confidence and growing strength in the air,...

$$
\begin{aligned}
T_1 \leq T_2 &\quad \text{iff } T_1 \text{ prefix } T_2 \\
glb(T_1, T_2) = &\quad \text{longest common prefix of } T_1 \text{ and } T_2 \\
lub(T_1, T_2) = &\quad \text{shortest string that has } T_1 \text{ and } T_2 \text{ as prefix}
\end{aligned}
$$

$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,\ we\ shall\ fight\ldots"}$

Is this a lattice?

VSs for the two movements

- The set of functions consistent with moving the cursor from position 1 to position 2 is concisely represented by the range
  - `["s", "shall fight on the seas and oceans...in the air."]`.
- The set of functions consistent with moving the cursor from position 2 to position 3 is concisely represented by the range
  - `["sh", "shall fight with growing confidence and growing strength in the air."]`
- The set of functions for both the movements:
  - $$[a_l, a_h] \cap [b_l, b_h] = [lub(a_l, b_l), glb(a_h, b_h)]$$
  - `["sh", "shall fight "]`.

Idea

If your hypothesis space is partially ordered and your VS are boundary set representable, you can represent and search very efficiently

If they are not?

Break them down into simpler hypothesis spaces!

Union And Join

$$VS_{H_1 D} \cup VS_{H_2 D} = VS_{H_1 \cup H_2\ D}$$

$$VS_{H_1 D_1} \bowtie VS_{H_2 D_2} = \left\{ \langle h_1, h_2 \rangle \;\middle|\; h_1 \in VS_{H_1 D_1},\ h_2 \in VS_{H_2 D_2},\ C(\langle h_1, h_2 \rangle,\ D) \right\}$$

- Where $D_1 = \{d_1^i\}_{i=0..n}$ and $D_2 = \{d_2^i\}_{i=0..n}$ and $D = \{\langle d_1^i, d_2^i \rangle\}_{i=0..n}$
- $C(\langle h_1, h_2 \rangle,\ D)$ means that $\langle h_1, h_2 \rangle$ is consistent with the input output pairs in D

What does $\langle h_1, h_2 \rangle$ mean? What about $\langle d_1, d_2 \rangle$?

- Pair
- Composition $\langle h_1, h_2 \rangle = h_1 \circ h_2$ and $\langle d_1, d_2 \rangle = (d_1 . in\ ,\ d_2 . out)$

Representation-based search

and/or graph

Version Space Algebras
(VSA)

Finite Tree Automata
(FTA)

Equivalence Graphs
(e-grpahs)

ops: learn-1, intersect, extract
DSL: efficiently invertible
similar to: top-down prop,
but can infer constants

Finite Tree Automata

states

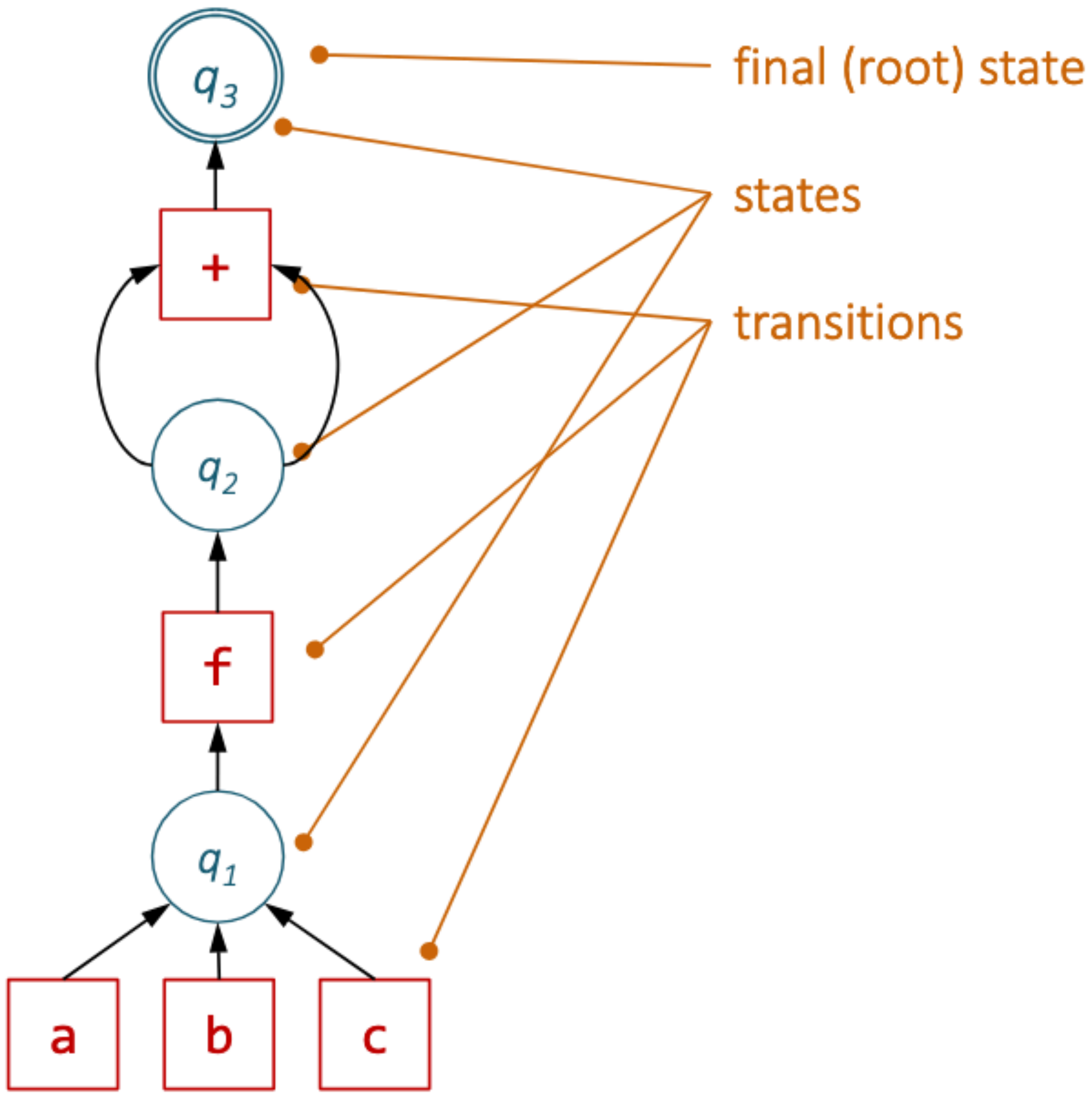
final state(s)

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

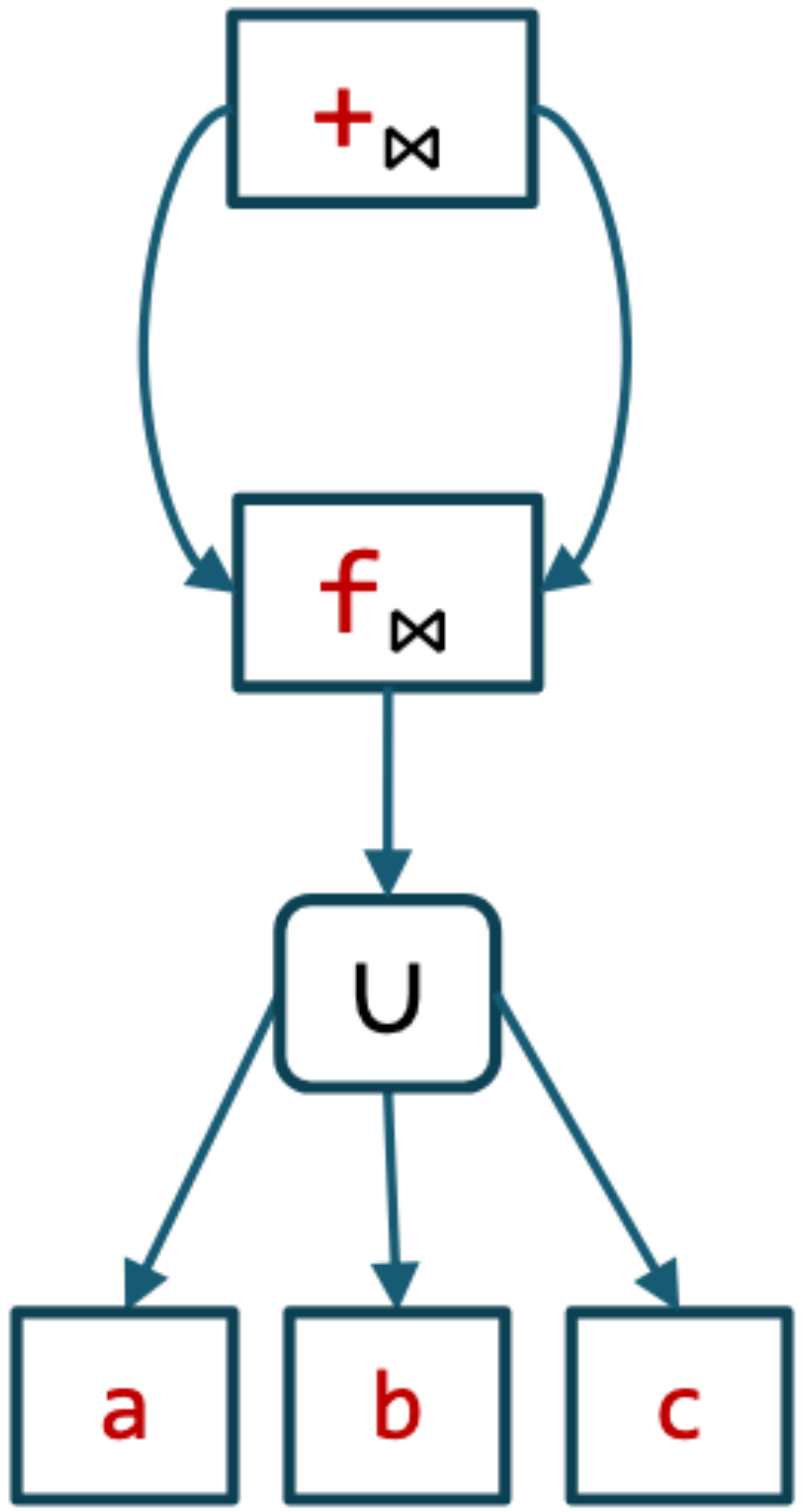
ranked alphabet

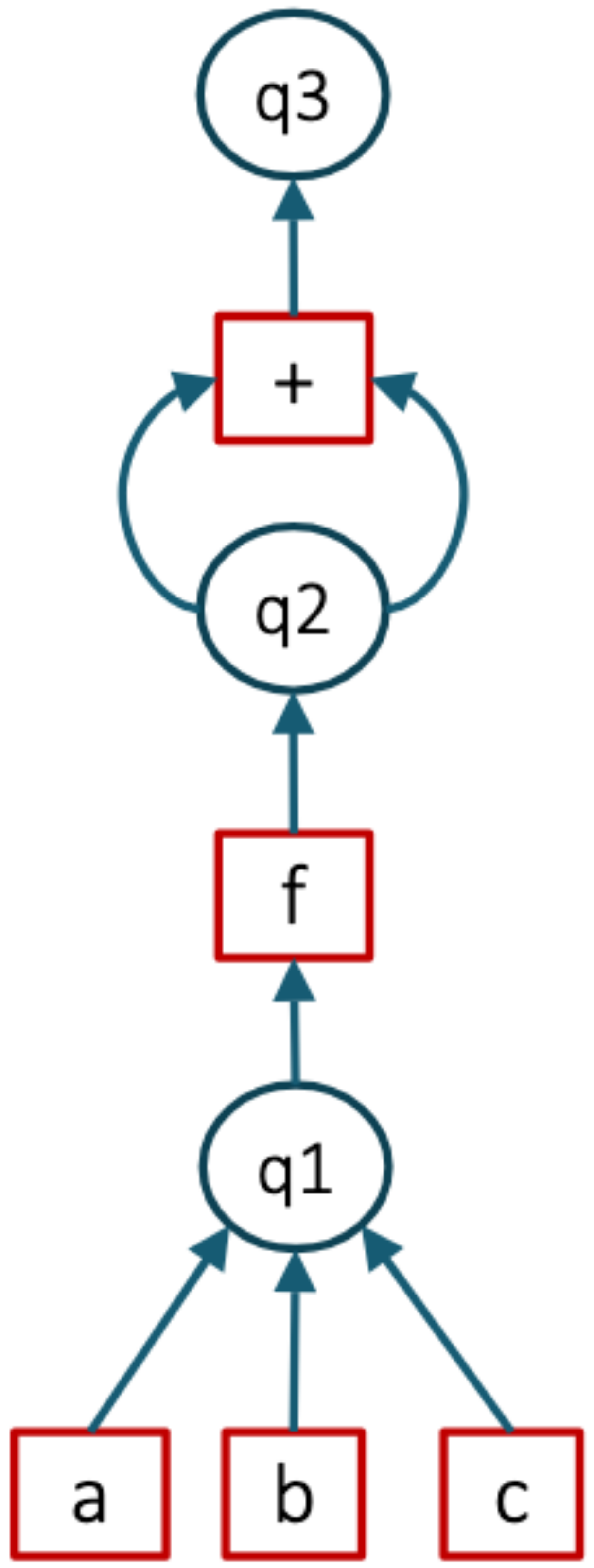
transitions

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



VSA vs FTA





Both are and-or graphs

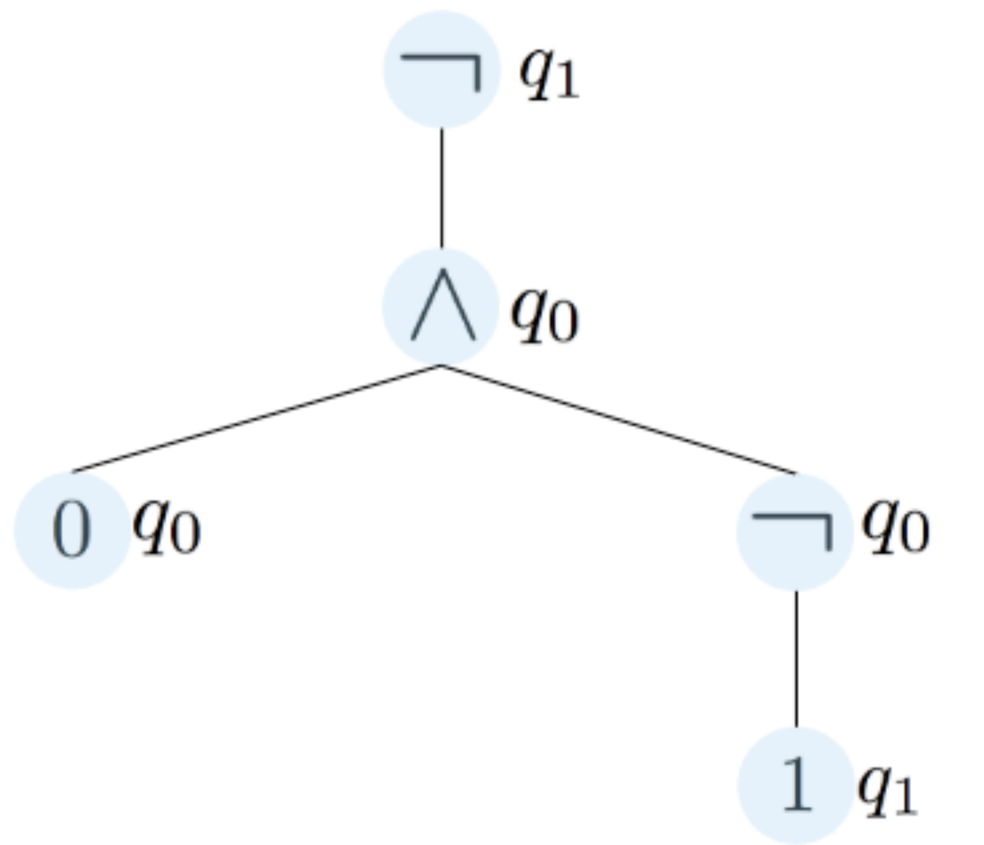
FTA state = VSA union node

- in VSAs singleton unions are omitted

FTA transition = VSA join node

FTA Formally

Definition 2.1. **(FTA)** A (bottom-up) finite tree automato
$\mathcal{A} = (Q, F, Q_f, \Delta)$ where Q is a set of states, $Q_f \subseteq Q$ is a s
transitions (rewrite rules) of the form $f(q_1, \cdots, q_n) \to q$ wher



. Tree for $\neg(0 \wedge \neg 1)$, annotated with states.

Example: **(FTA)** Consider the tree automaton $\mathcal{A}$ defined by states $Q = \{q_0, q_1\}$, $F_0 = \{0, 1\}$, $F_1 = \{\neg\}$, $F_2 = \{\wedge\}$, final states $Q_f = \{q_0\}$, and the following transitions Δ:

$$\begin{array}{llll} 1 \to q_1 & 0 \to q_0 & \wedge(q_0, q_0) \to q_0 & \wedge(q_0, q_1) \to q_0 \\ \neg(q_0) \to q_1 & \neg(q_1) \to q_0 & \wedge(q_1, q_0) \to q_0 & \wedge(q_1, q_1) \to q_1 \end{array}$$

A term t is accepted by an FTA if we can rewrite t to some state q $\in$ Qf using rules in Δ.

Accepts those propositional Logic Formulas which evaluate to false

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 ::= id(V) | N + T | N * T
T ::= 2 | 3
V ::= x
```

Spec

$1 \rightarrow 9$

PBE with Finite Tree Automata

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

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

$q_s^{\bar{c}}$

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

final states

states

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

alphabet

transitions

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

`id, +, *`

`+(<N,1>,<T,2>) → <N,3>`

...

$+(q_N^1, q_T^2) \rightarrow q_N^3$

PBE with Finite Tree Automata

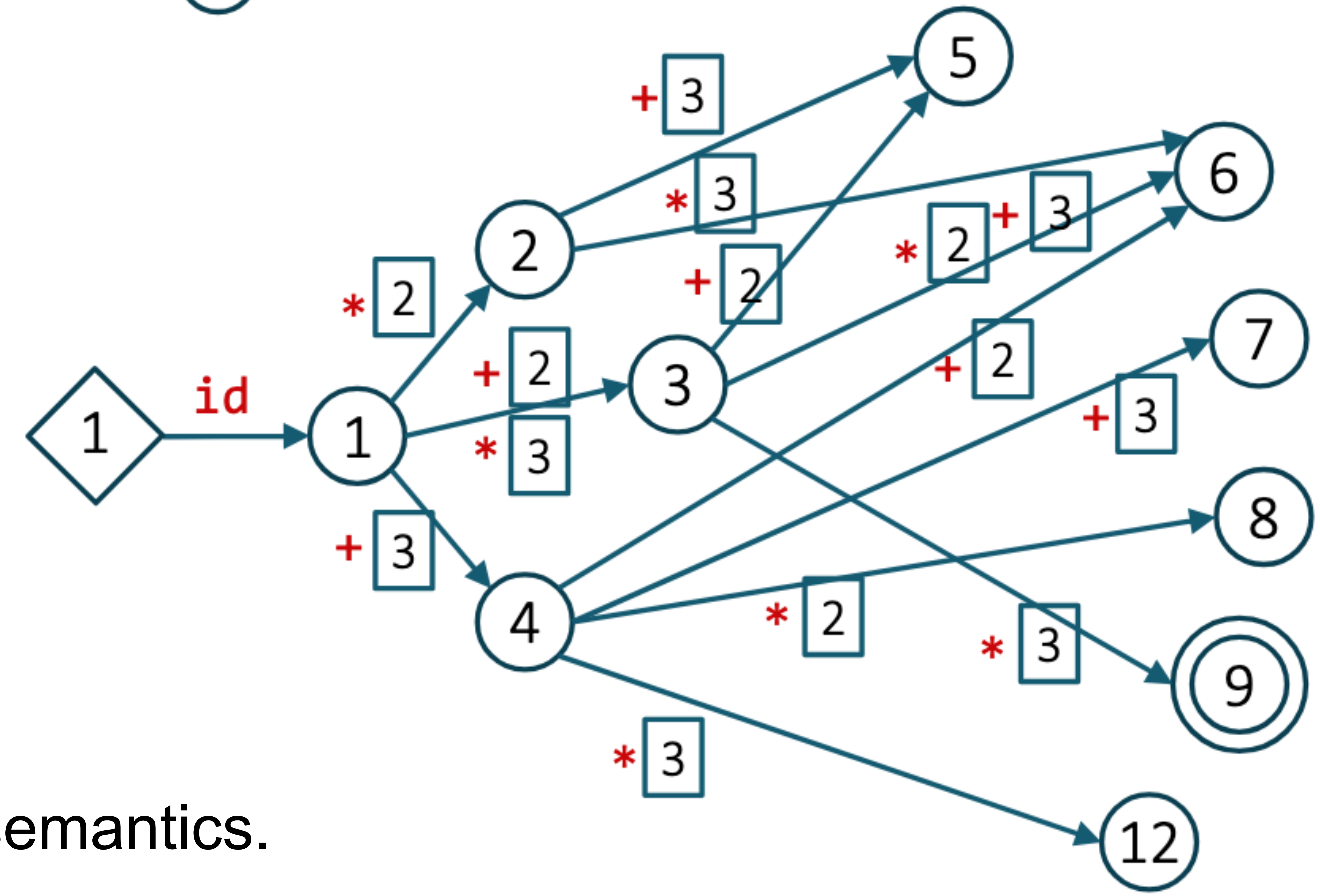
N ::= id(V) | N + T | N * T

T ::= 2 | 3

V ::= x

Circles for N

1 → 9



Q : Set of concrete values

Transitions : Using Concrete semantics.

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:

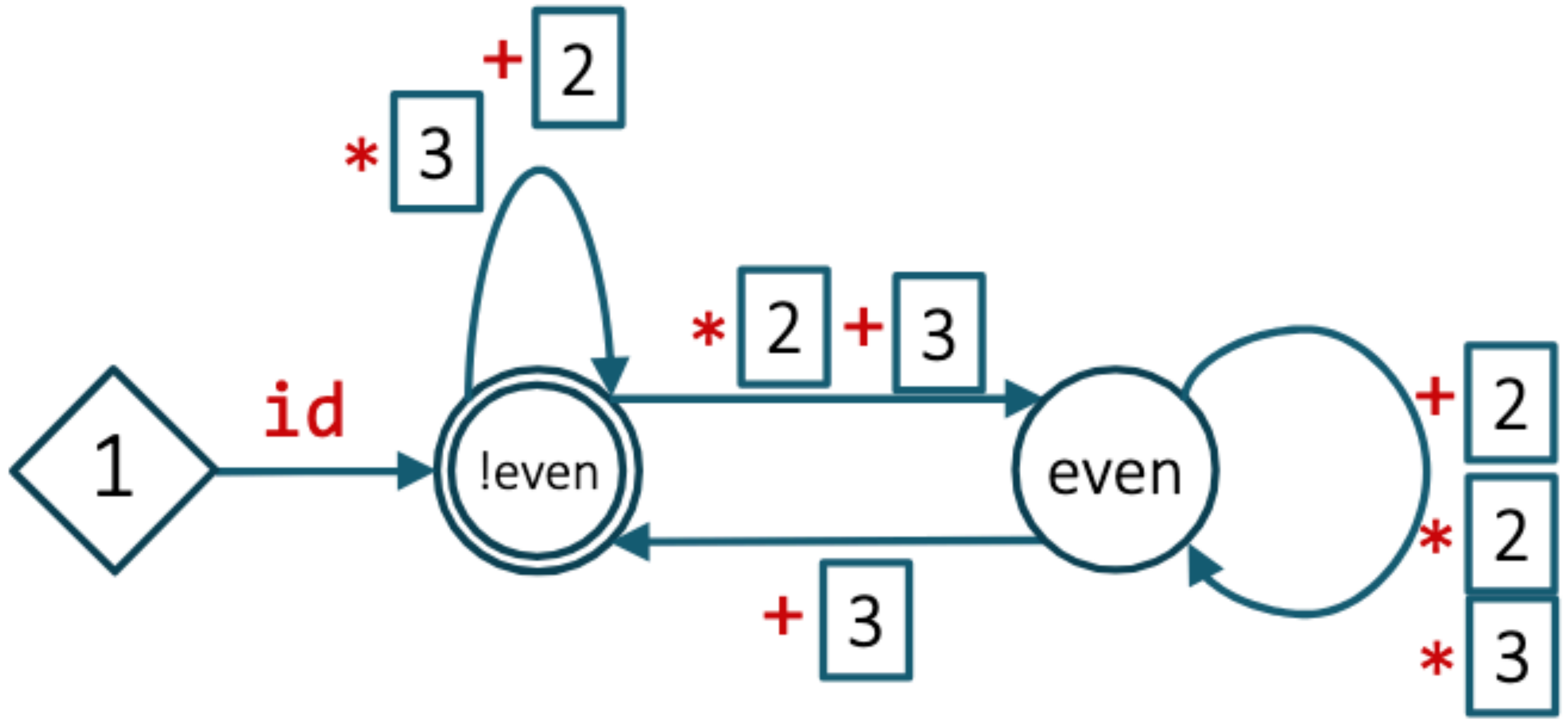
- instead of one state = one value
- we can do one state = set of values (= abstract value)

Abstract FTA

[Wang, Dillig, Singh POPL'18]

```
N ::= id(V) | N + T | N * T
T ::= 2 | 3
V ::= x
```

1 → 9



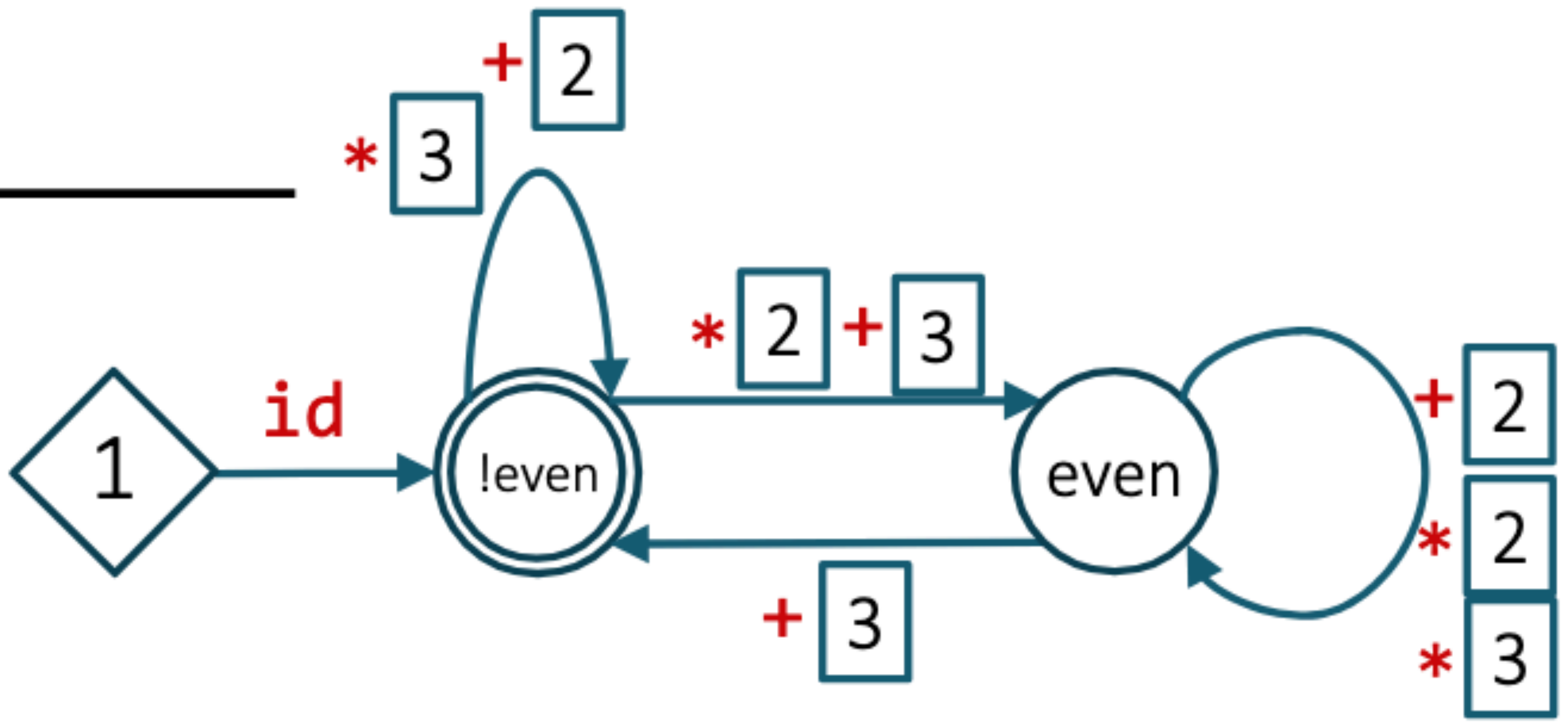
What now?

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

These Predicates are the abstractions

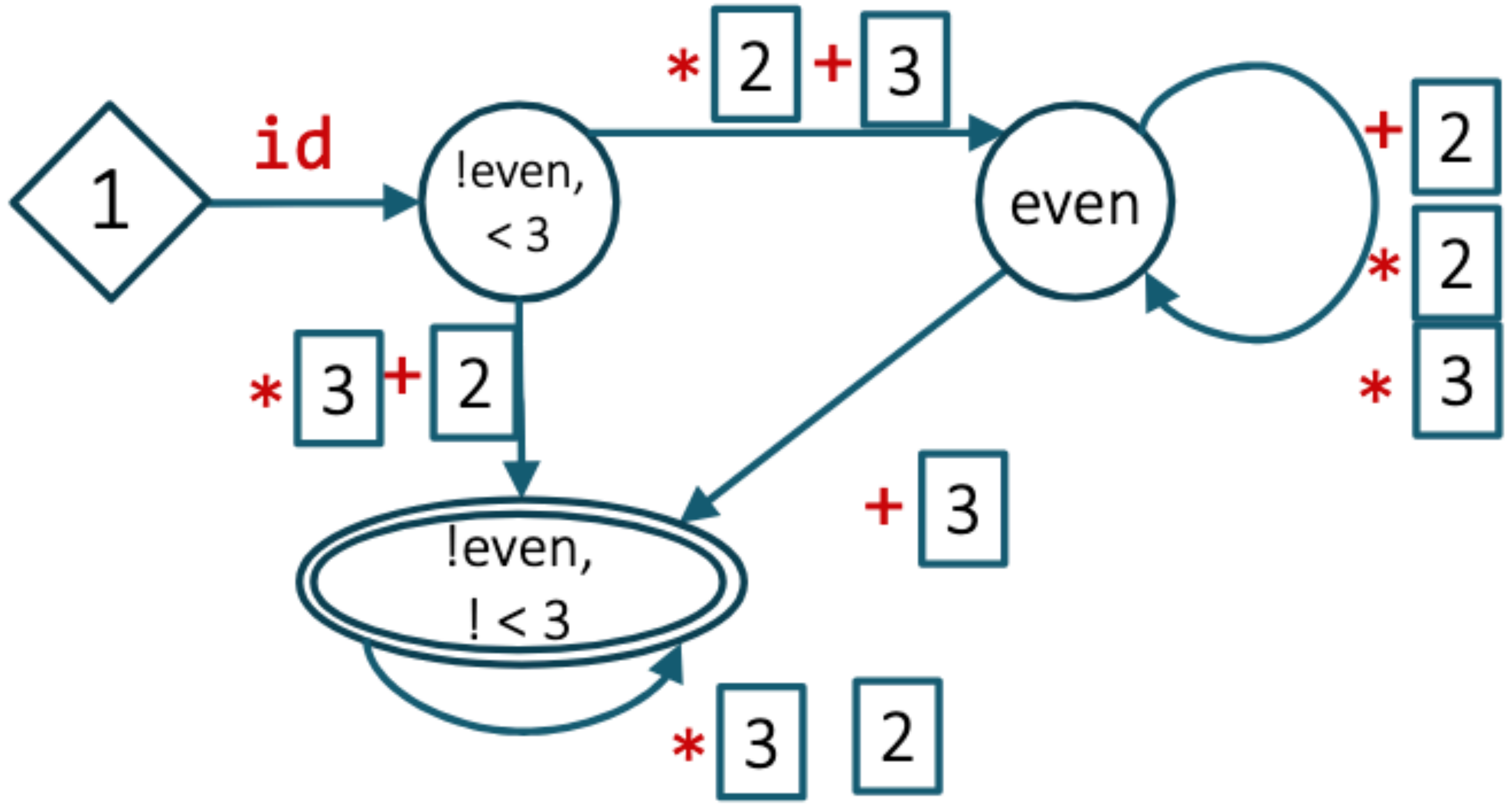
Abstract FTA

```
N ::= id(V) | N + T | N * T
T ::= 2 | 3
V ::= x
```



solution: `id(x)`

1 → 9



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

solution: `id(x)*3`

Representation-based search

and/or graph

Version Space Algebras (VSA)

Finite Tree Automata (FTA)

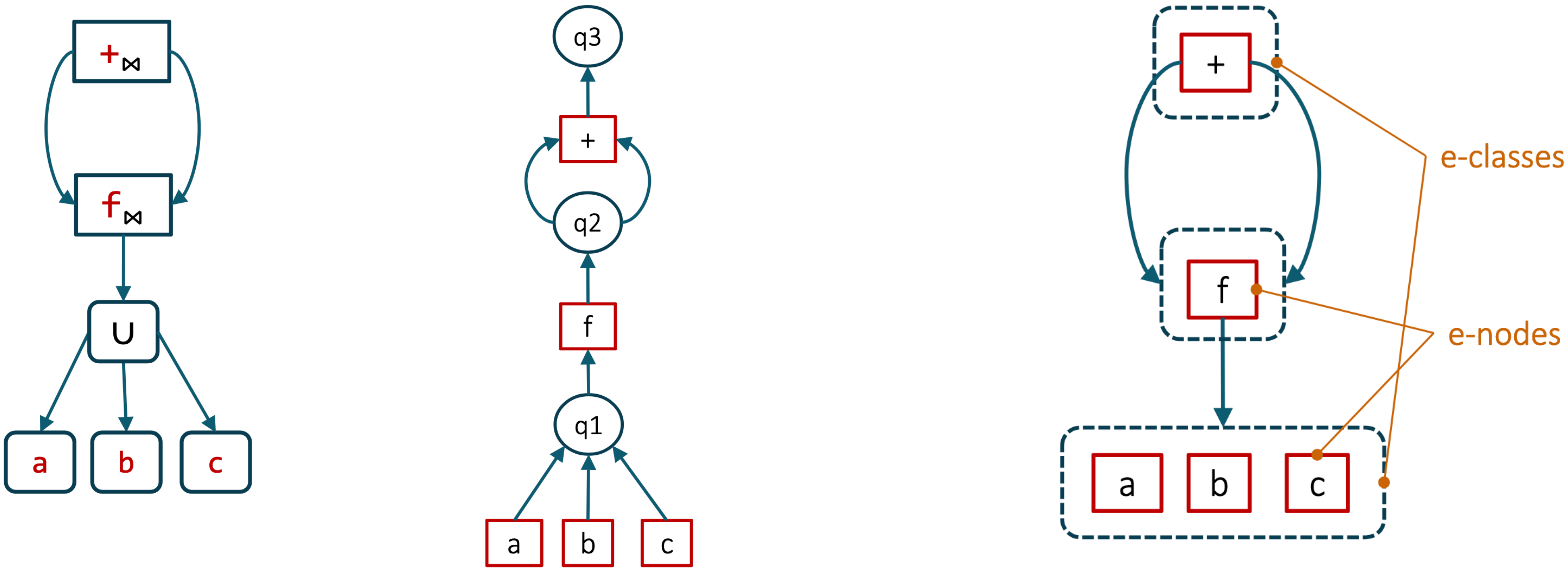
Equivalence Graphs (e-grpahs)

ops: learn-1, intersect, extract
DSL: efficiently invertible
similar to: top-down prop, but can infer constants

ops: learn-1, intersect, extract
DSL: efficiently enumerable
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-graphs

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:

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

useful rules:

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

not so useful:

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

Challenge: which ones to apply and in what order?

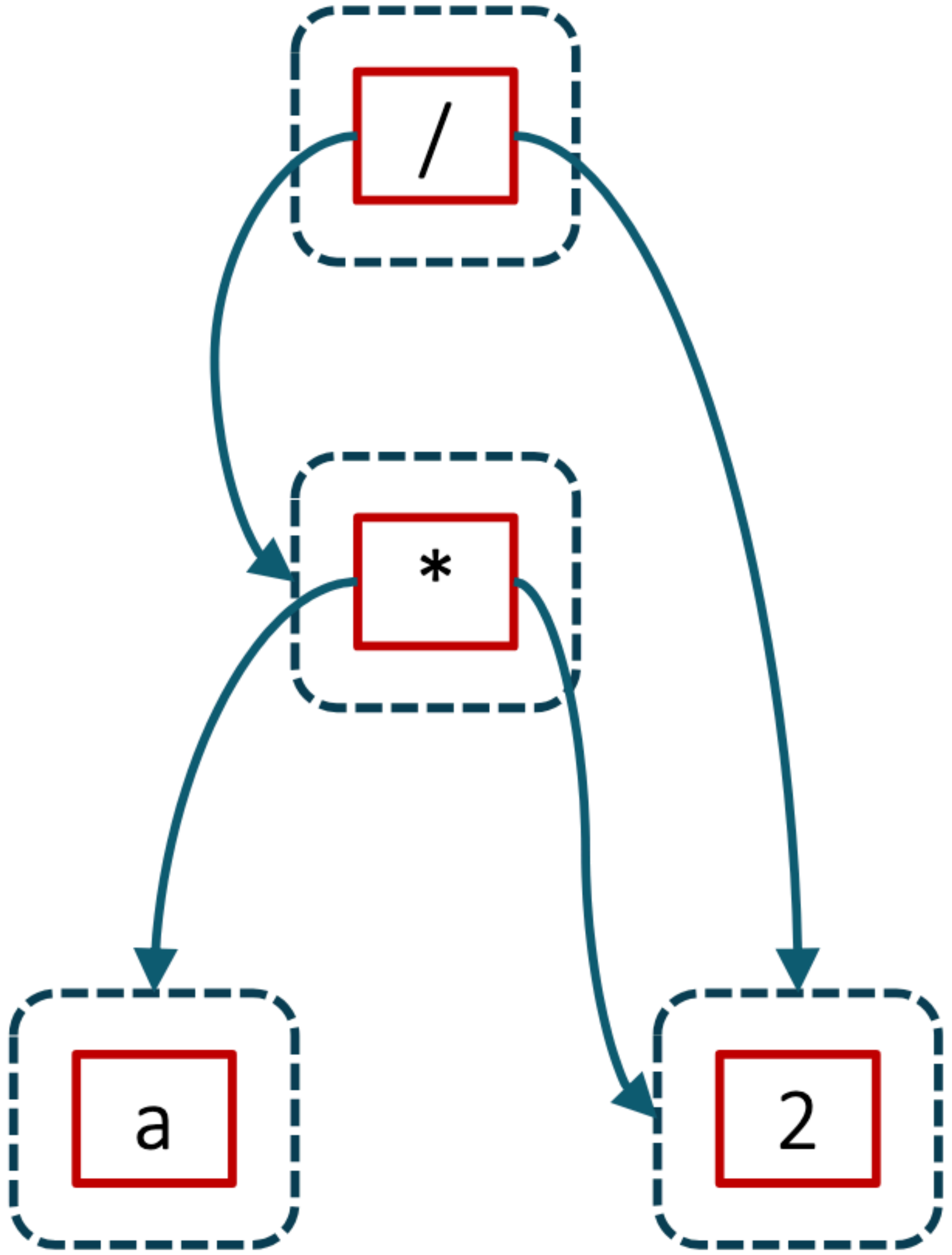
Idea: all of them all the time!

Equality saturation

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

Rewrite rules:

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

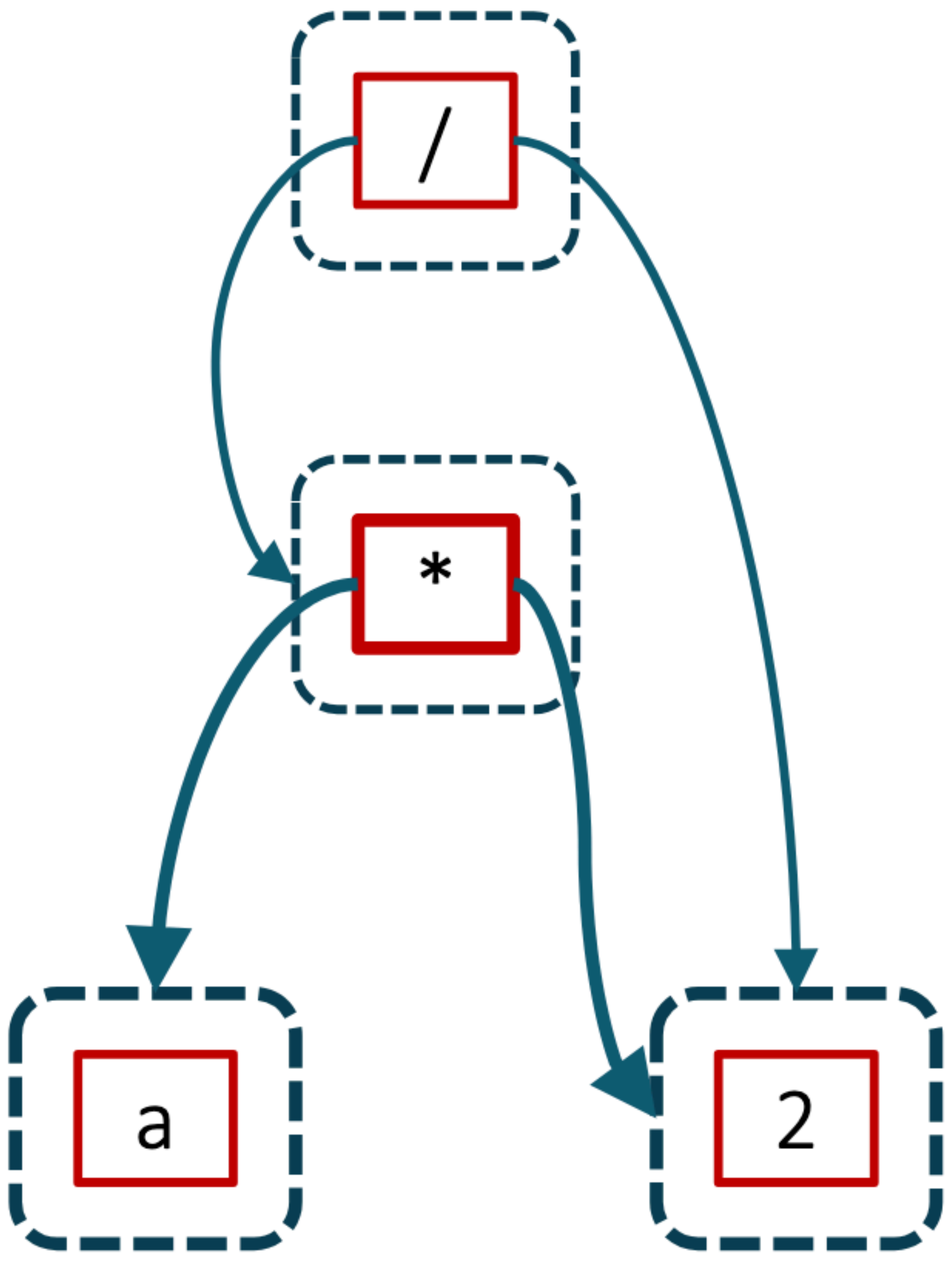


Equality saturation

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

Rewrite rules:

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

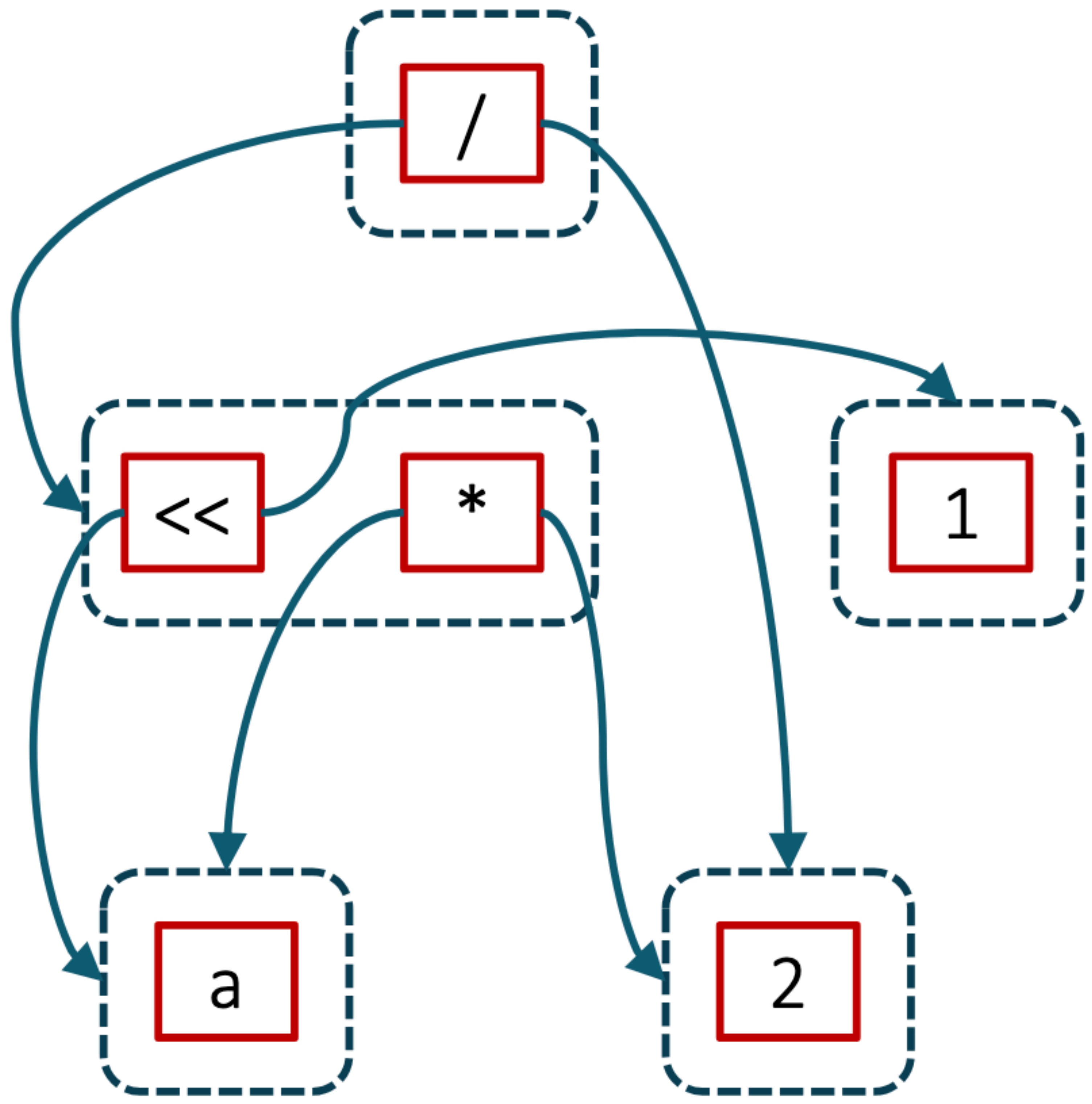


Equality saturation

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

Rewrite rules:

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

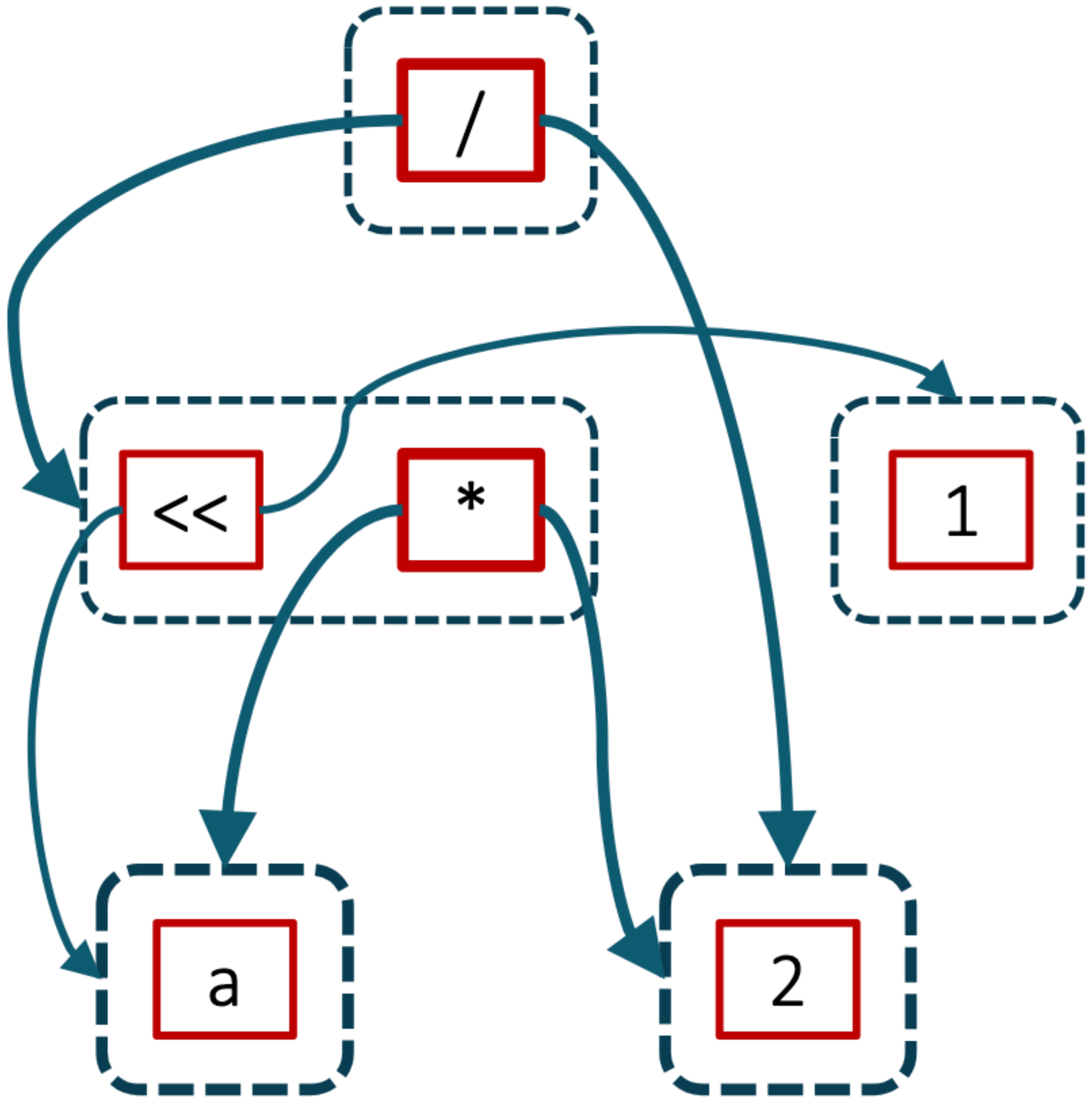


Equality saturation

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

Rewrite rules:

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

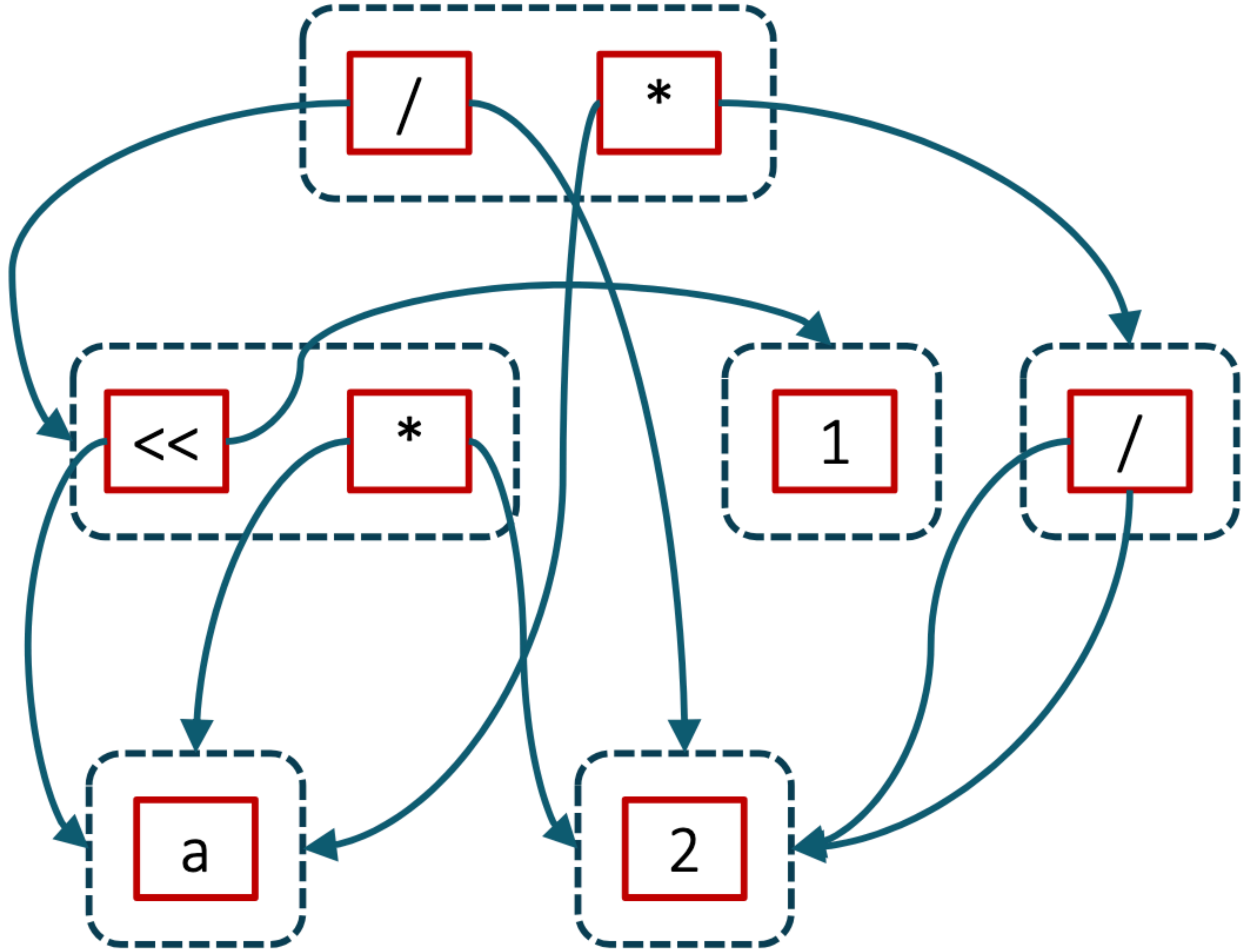


Equality saturation

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

Rewrite rules:

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

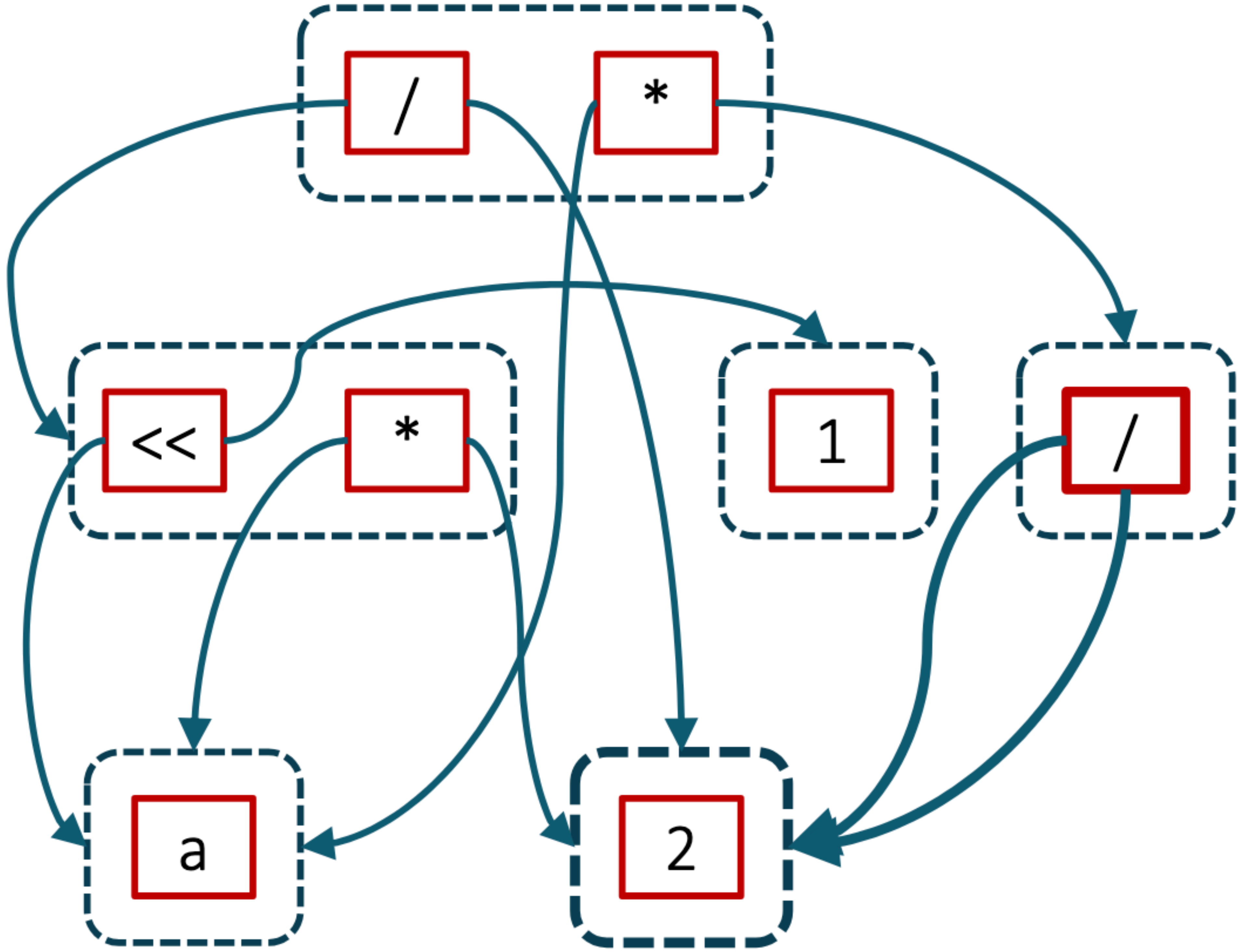


Equality saturation

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

Rewrite rules:

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

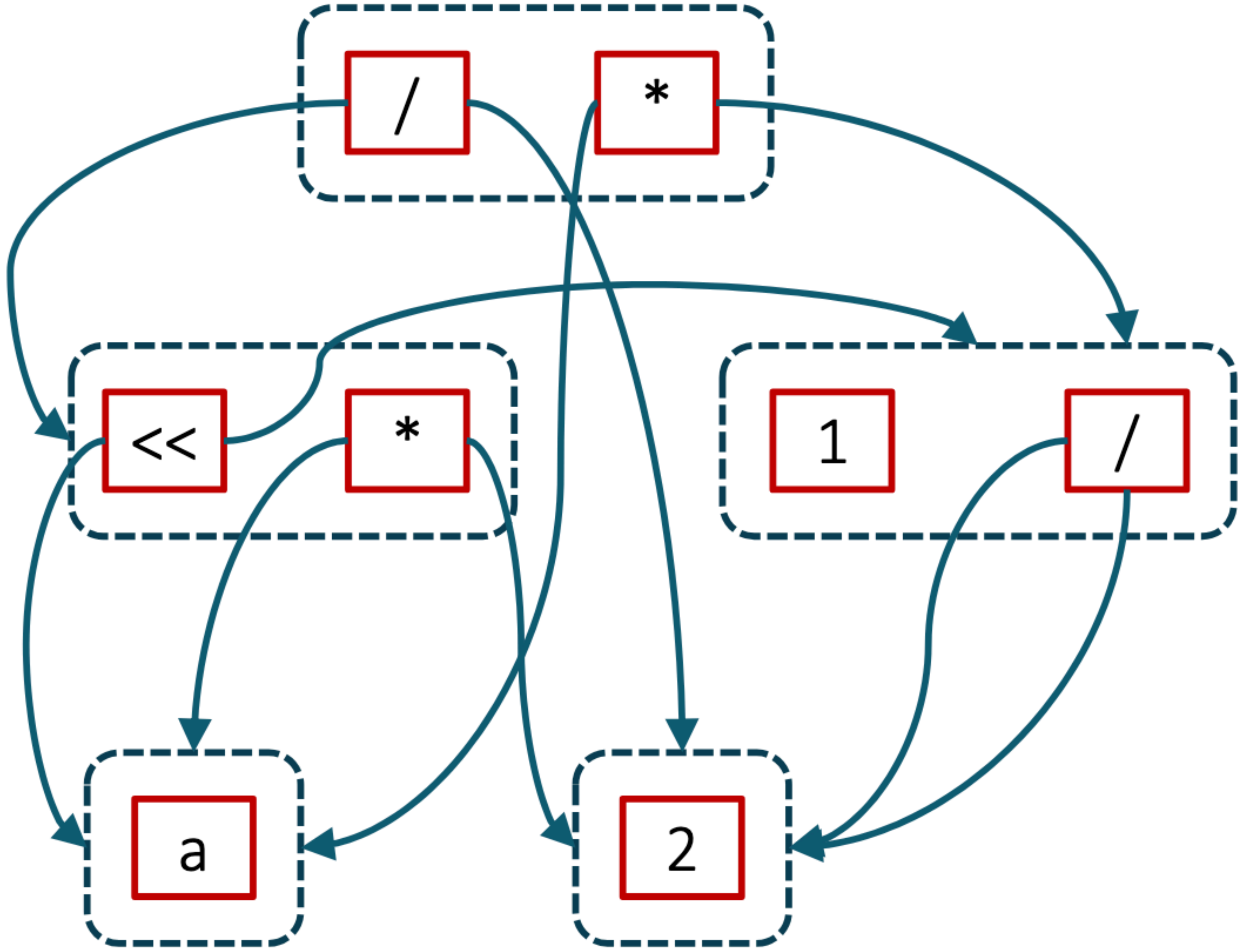


Equality saturation

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

Rewrite rules:

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

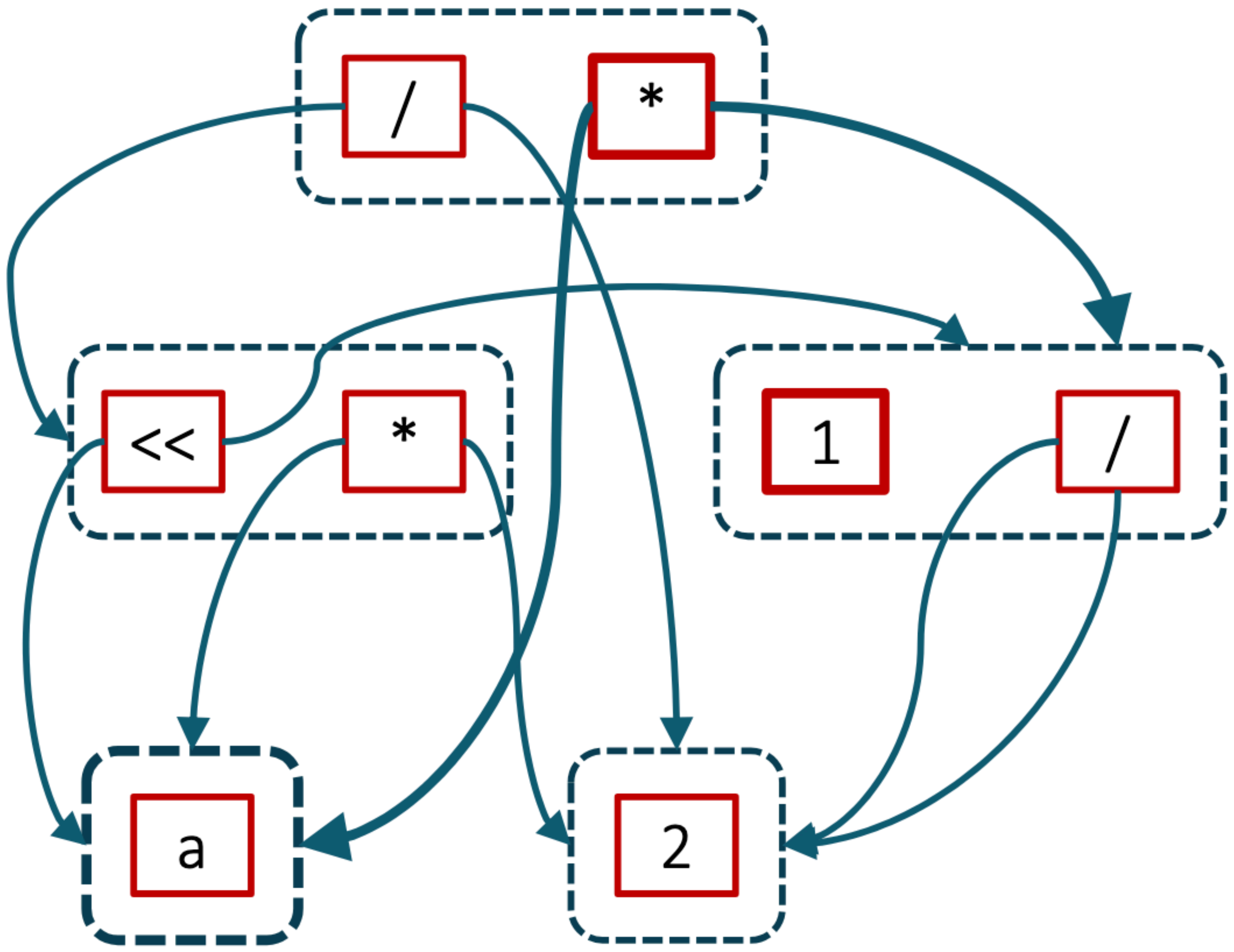


Equality saturation

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

Rewrite rules:

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

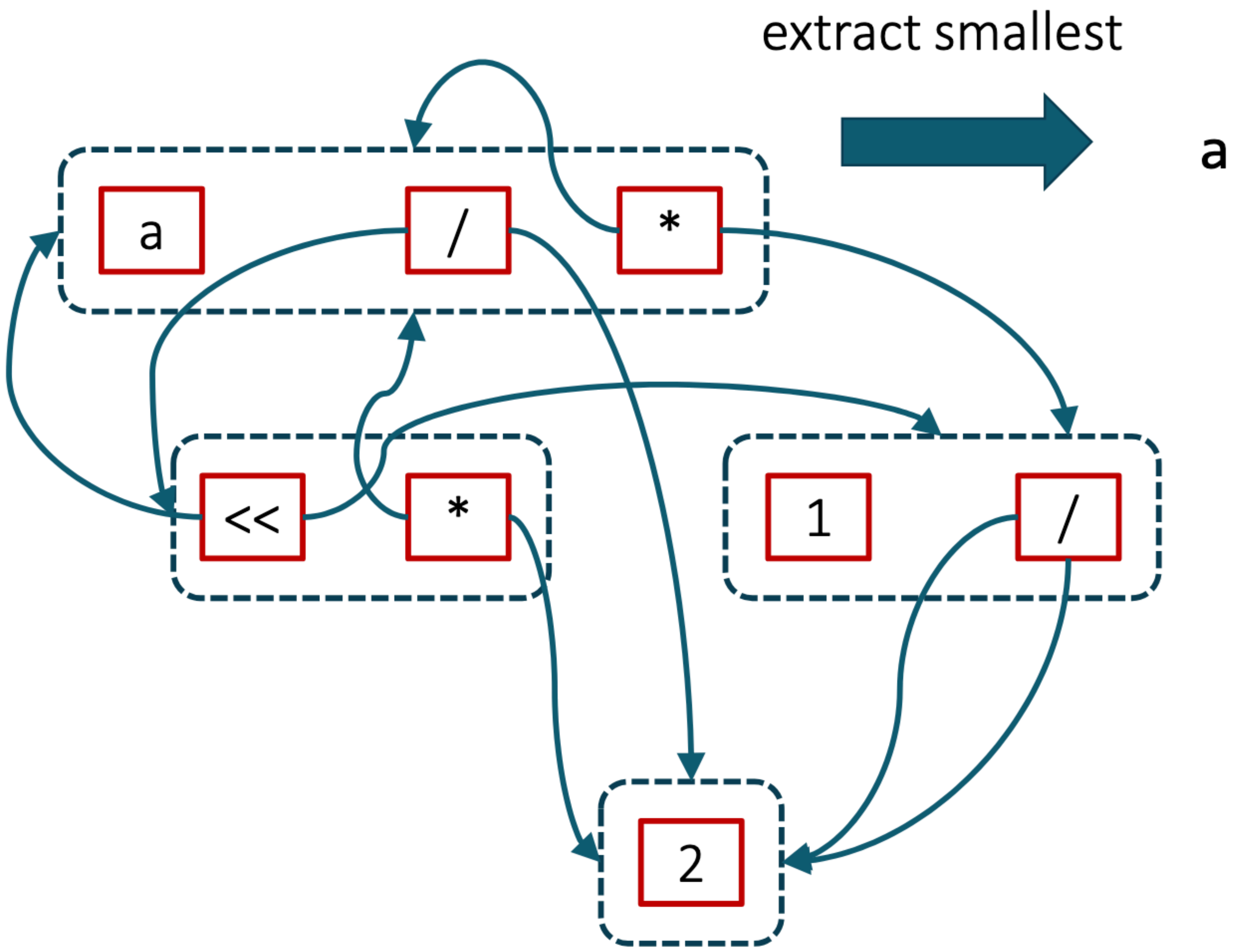


Equality saturation

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

Rewrite rules:

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



Representation-based search

and/or graph

Version Space Algebras (VSA)

ops: learn-1, intersect, extract
DSL: efficiently invertible
similar to: top-down prop, but can infer constants

Finite Tree Automata (FTA)

ops: learn-1, intersect, extract
DSL: efficiently enumerable
similar to: bottom-up with OE, but can store all programs (and add examples incrementally)

state: represents a set of observationally-equivalent programs

Equivalence Graphs (e-grpahs)

ops: rewrite, extract

similar to: term rewriting, but can store all programs

e-class: represents a set of programs equivalent up to rewrites

Projects

Project Goal

- Deeper understanding of the ideas discussed in the class and in the reading assignments.
  - by (re)-implementing some of the papers main algorithm.
  - Extending the behavioral, structural or the search strategy in some of these works.
  - Applying synthesis to some novel domain: e.g. robotics, compilers, networks, databases, logic, hardware designs, etc.
  - List of projects is provided on the course page.
  - Surely, welcome to propose your own idea.

Evaluations and Milestones.

- M1: Discussion about your idea with me, what papers is your idea based on, what do you plan to do. 10%
  - It will allow us to gauge the proposal in terms of if it is doable in the given time.
  - I can suggest some related papers and open-source implementations.
  - We can revise the problem, making it more palpable for the class project.
- M2: Formal 1-2 page proposal, this should include: 15%
  - a concrete example, showing input and output for the synthesizer and how does the Algorithm roughly works on the input.
- E1 : Execution of the Idea!
- M3 : Final
  - Presentation and 10%
  - Report (3-8 pages) in PACML PL format. 15%