#10: Finite Tree Automata and e-graphs

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Representation-based search

and/or graph

Version Space Algebras (VSA)

Finite Tree Automata (FTA)

- **Equivalence Graphs (e-graphs)**

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

Finite Tree Automata

states

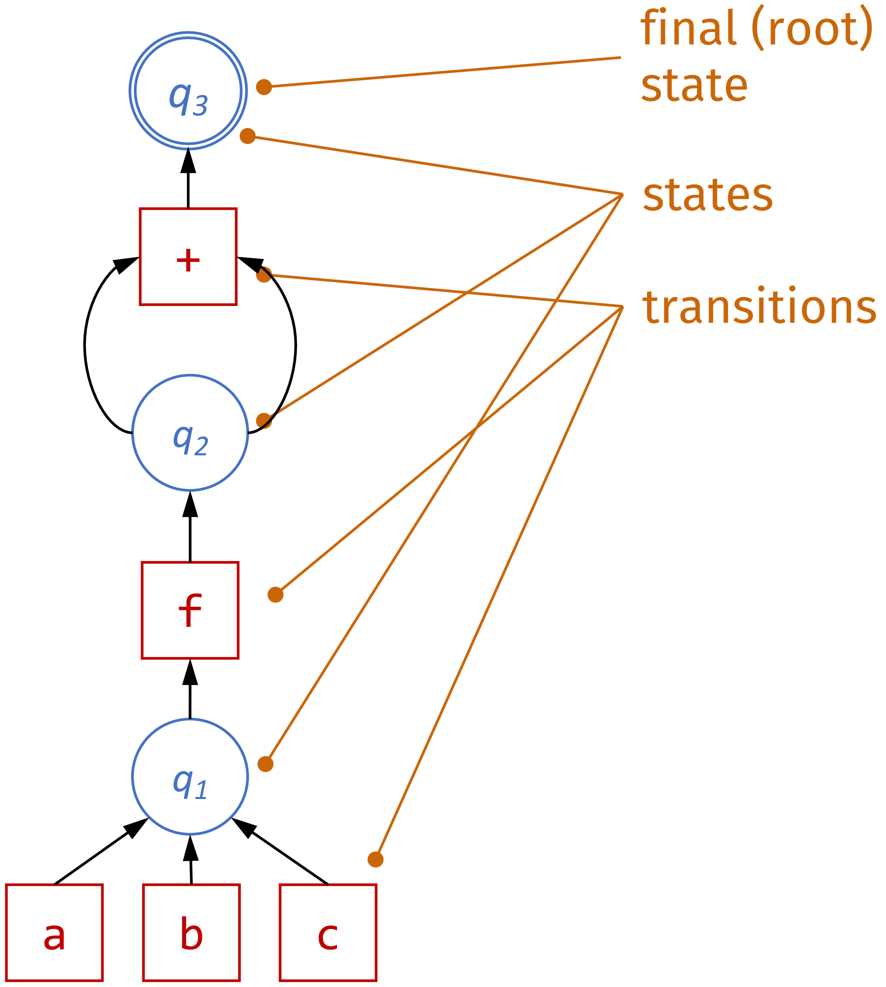
final state(s)

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

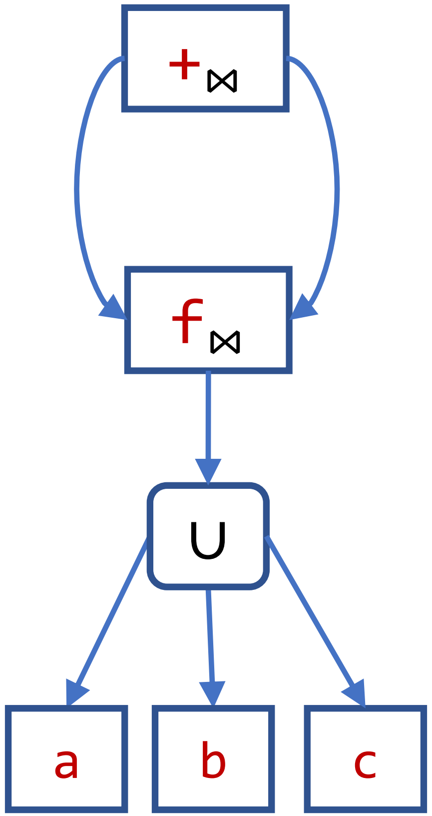
ranked alphabet

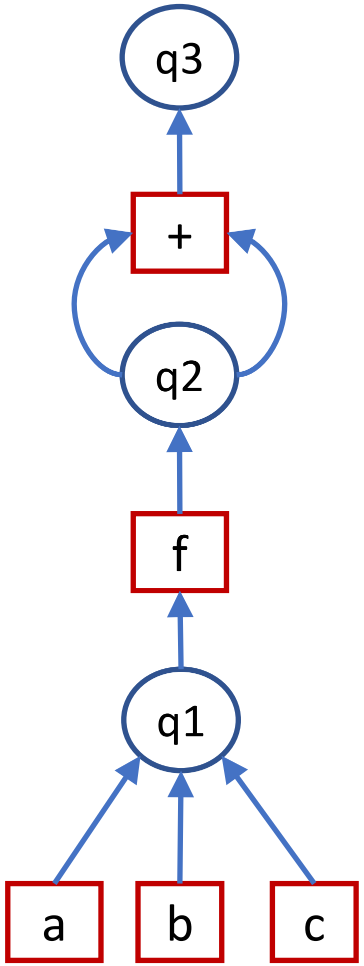
transitions

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



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

Spec

$1 \rightarrow 9$

PBE with Finite Tree Automata

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

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

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

states

final states

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

alphabet

transitions

`id`, `+`, `*`

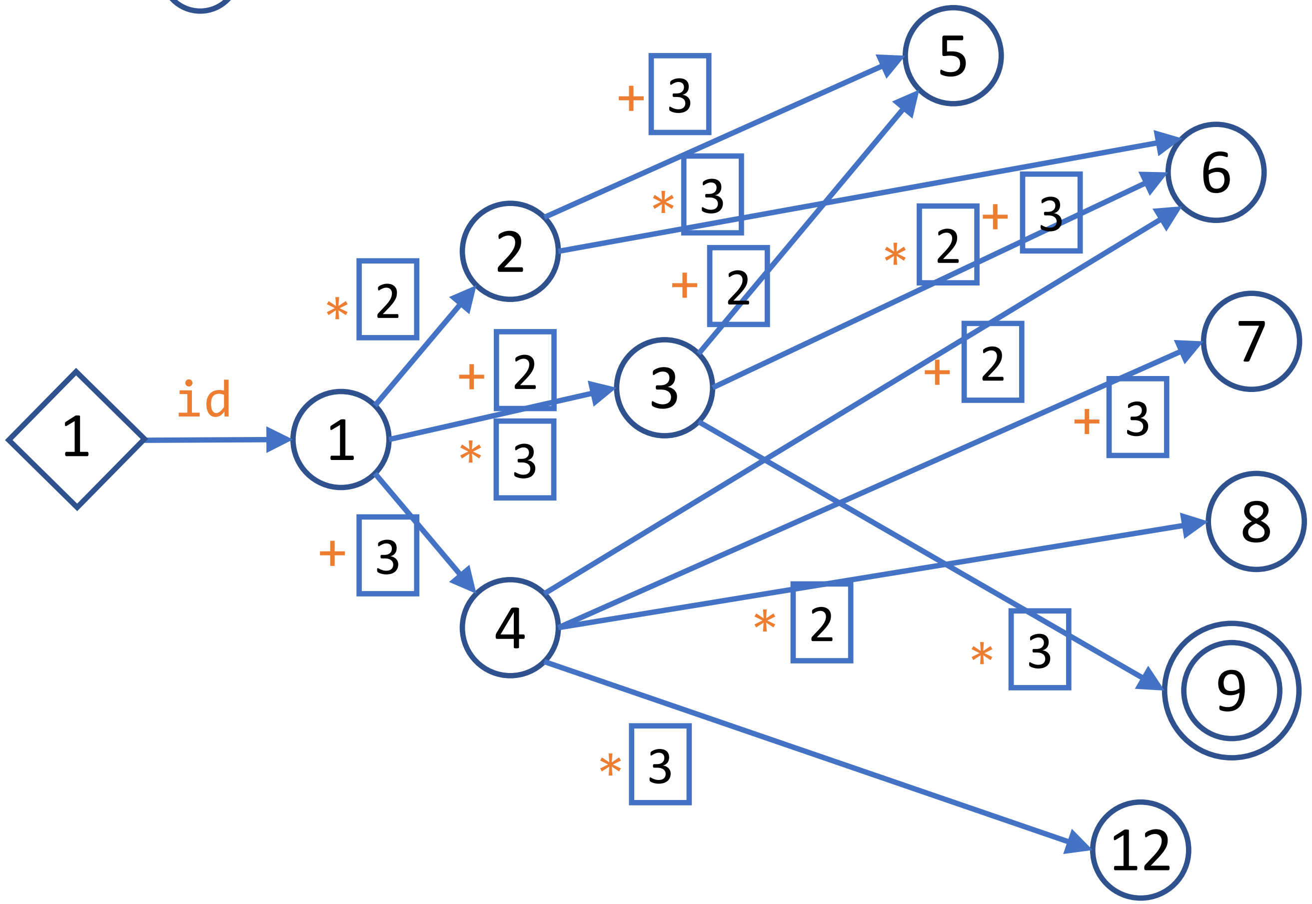
$$f(q_1, \ldots, q_n) \to q$$

```
+(<N,1>,<T,2>) → <N,3>
...
```

PBE with Finite Tree Automata

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

1 → 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

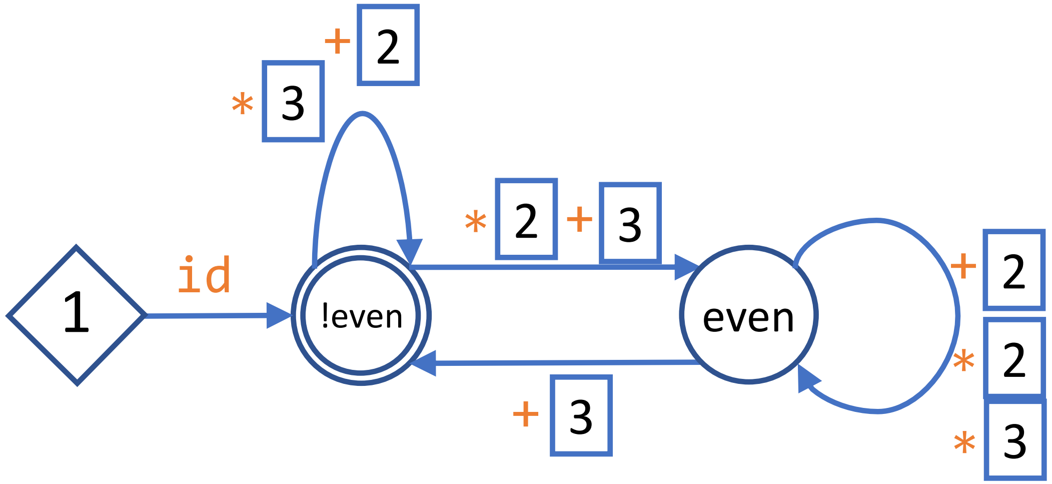
- **Challenge:** FTA still has too many states
- Idea:
  - instead of one state = one value
  - we can do one state = set of values (= abstract value)

Abstract FTA

[Wang, Dillig, Singh POPL‘18]

```
N ::= 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!

Abstract FTA

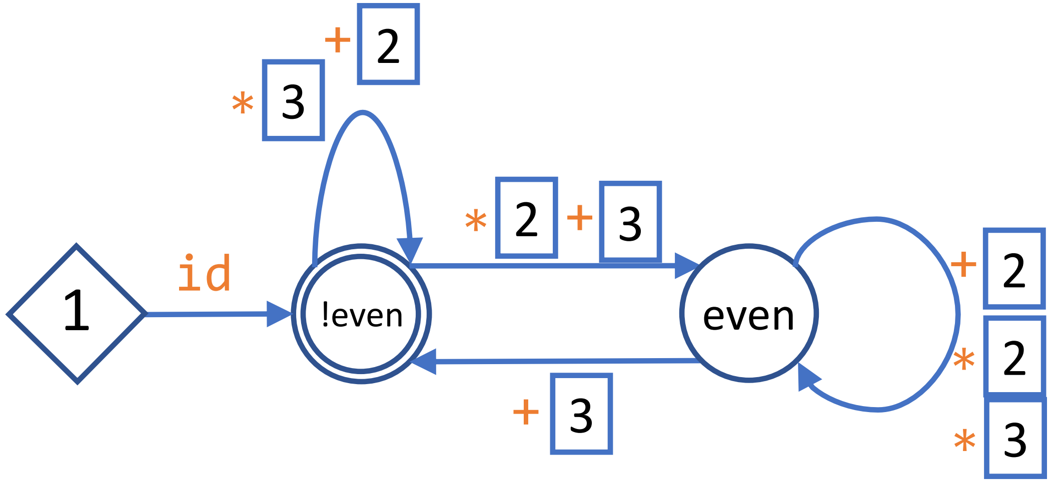
N ::= id(V) | N + T | N * T

T ::= 2 | 3

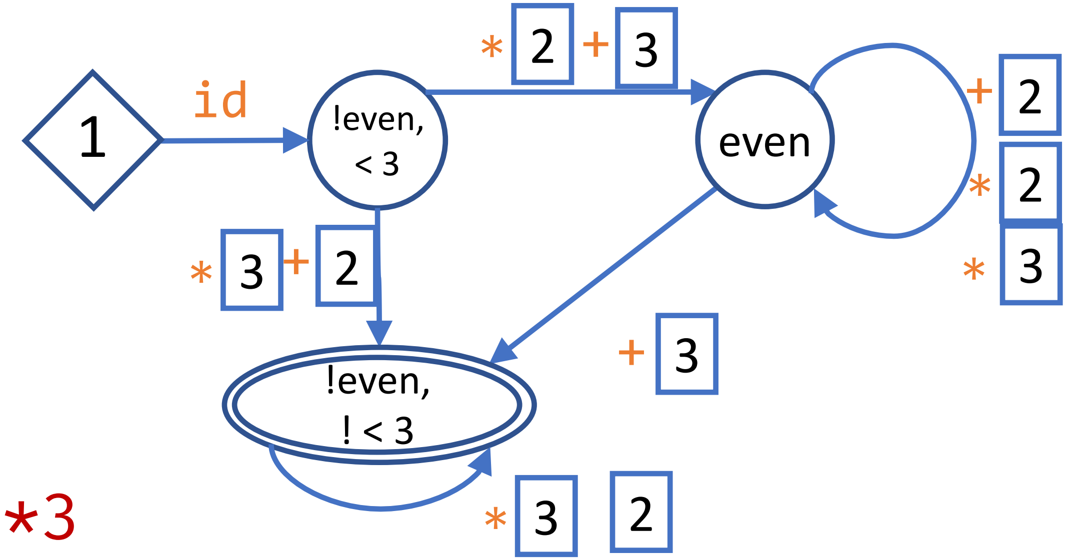
V ::= x

1 → 9

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



solution: id(x)



solution: id(x)*3

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

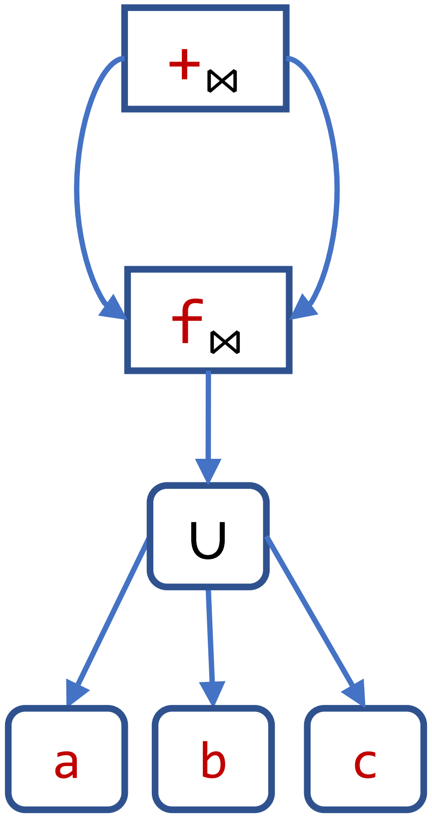
Finite Tree Automata (FTA)

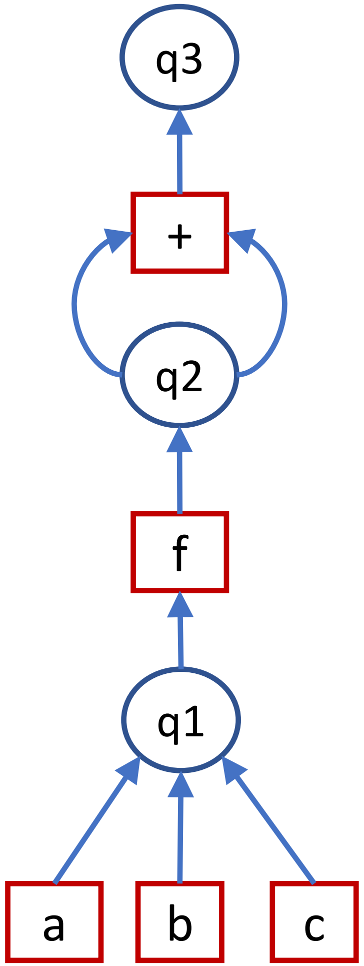
ops: learn-1, intersect, extract
DSL: efficiently enumerable
similar to: bottom-up with OE, but can store all programs (and add examples incrementally)

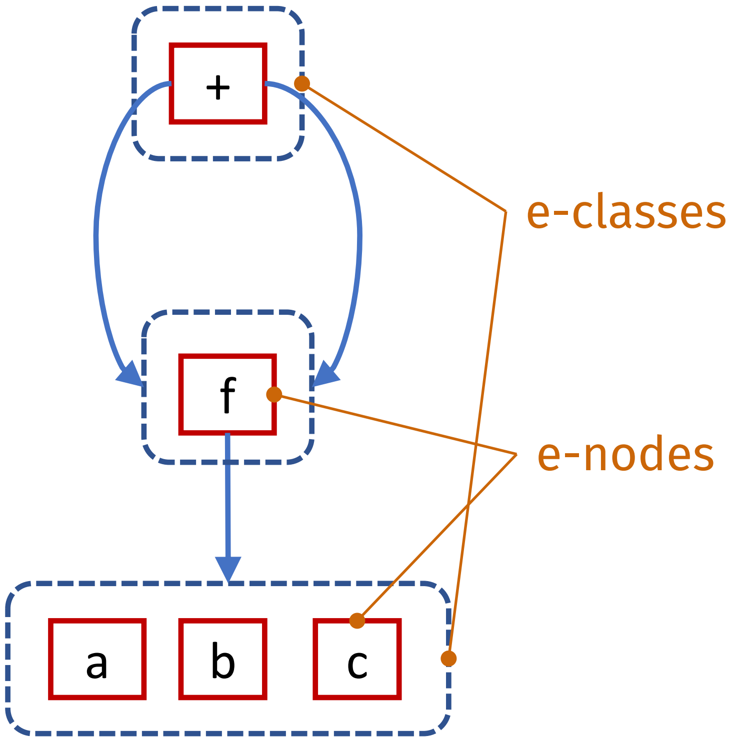
- **Equivalence Graphs (e-grpahs)**

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:

```
   (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?

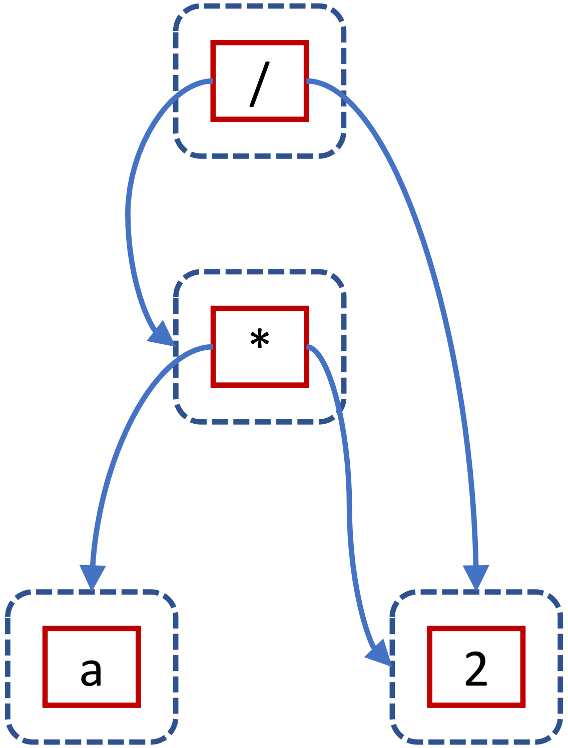
Idea: all of them all the time!

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
```

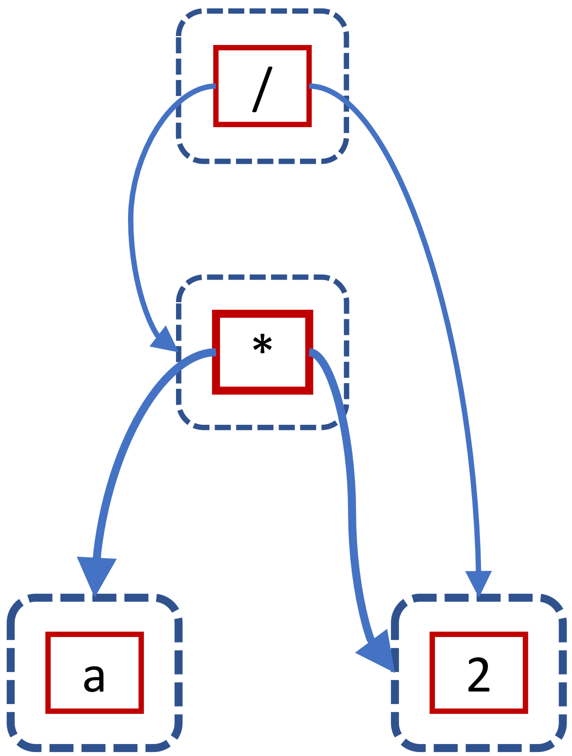


Equality saturation

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
```

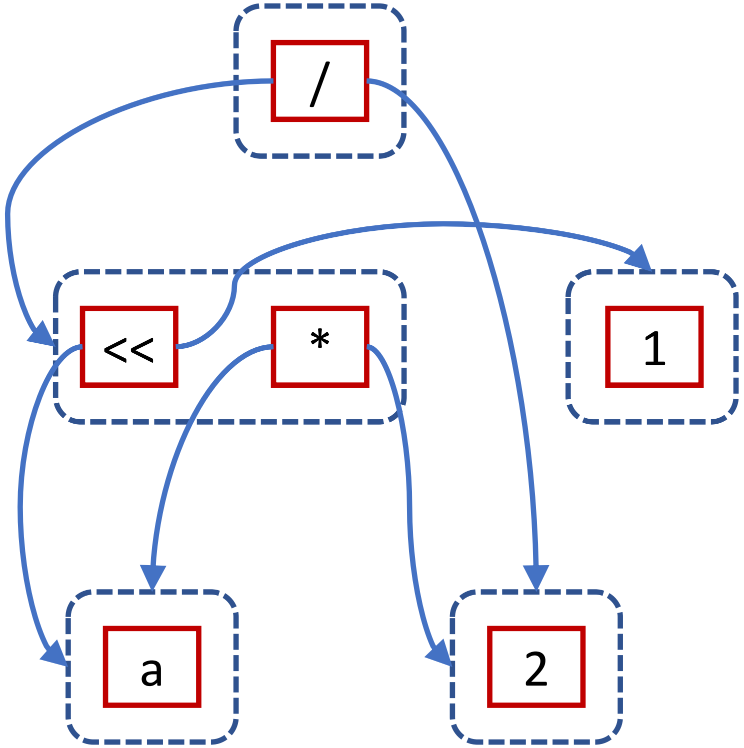


Equality saturation

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
```

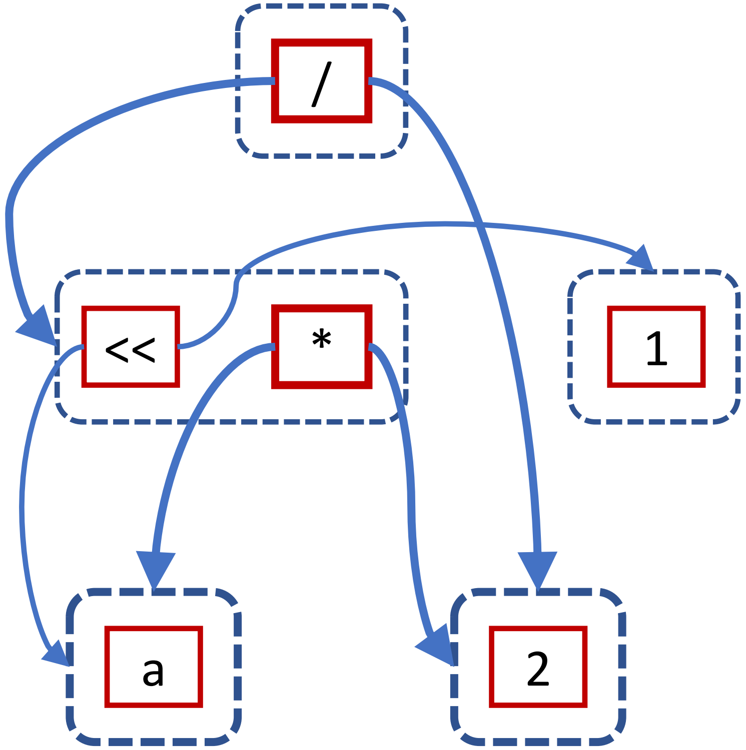


Equality saturation

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
```

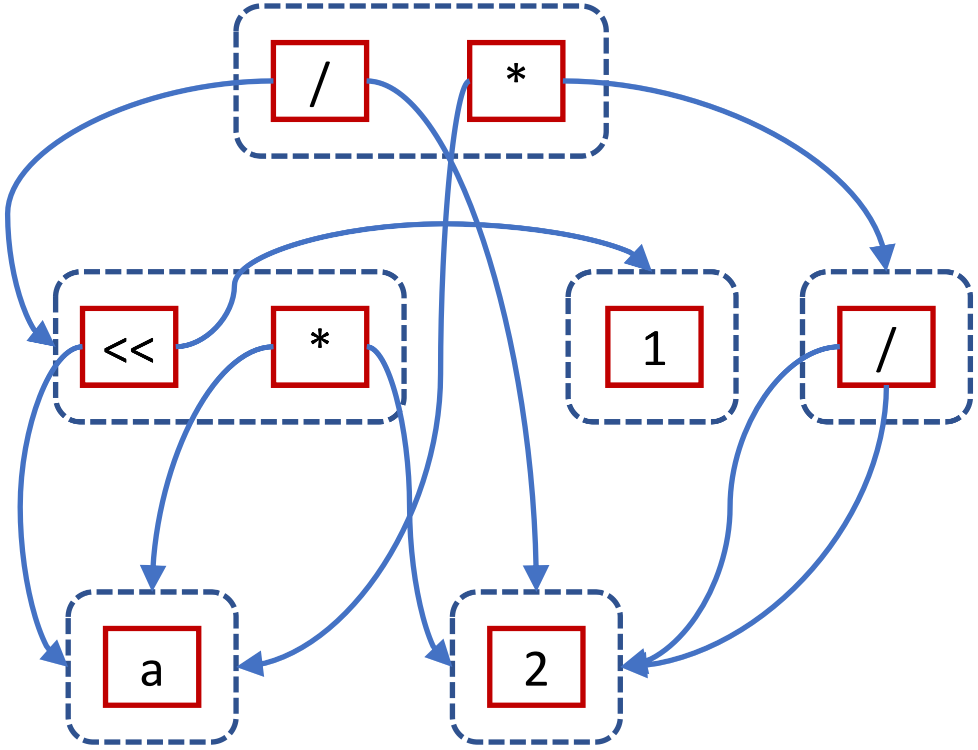


Equality saturation

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
```

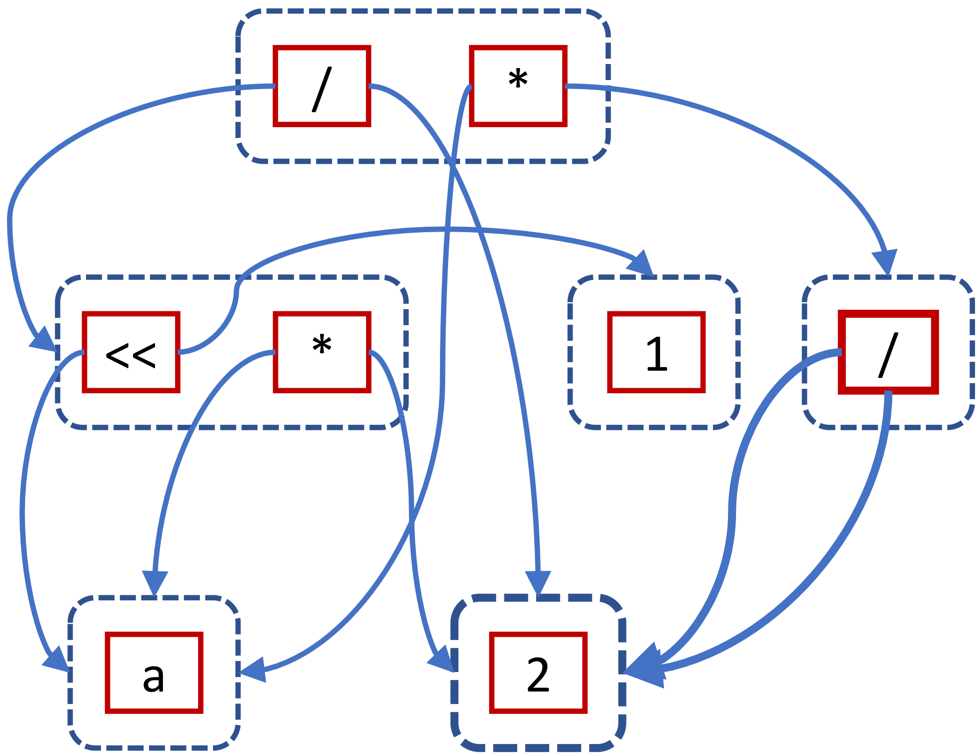


Equality saturation

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:

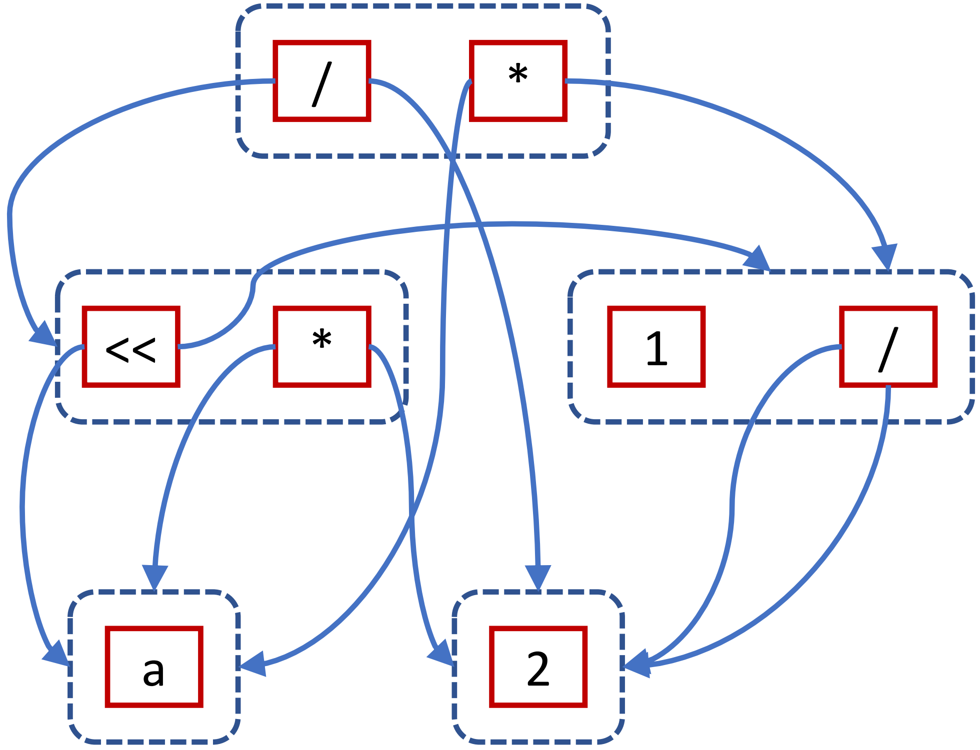
```
(x * y) / z = x * (y / z)

        x / x = 1

        x * 1 = x

      x * 2 = x << 1

       x * y = y * x
```

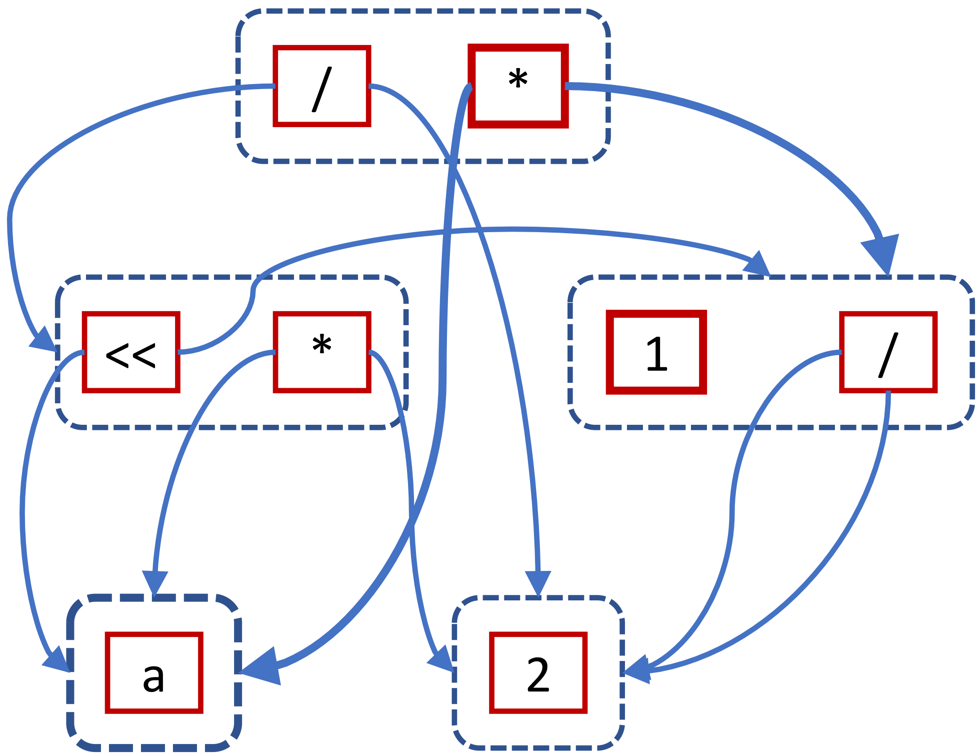


Equality saturation

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
```

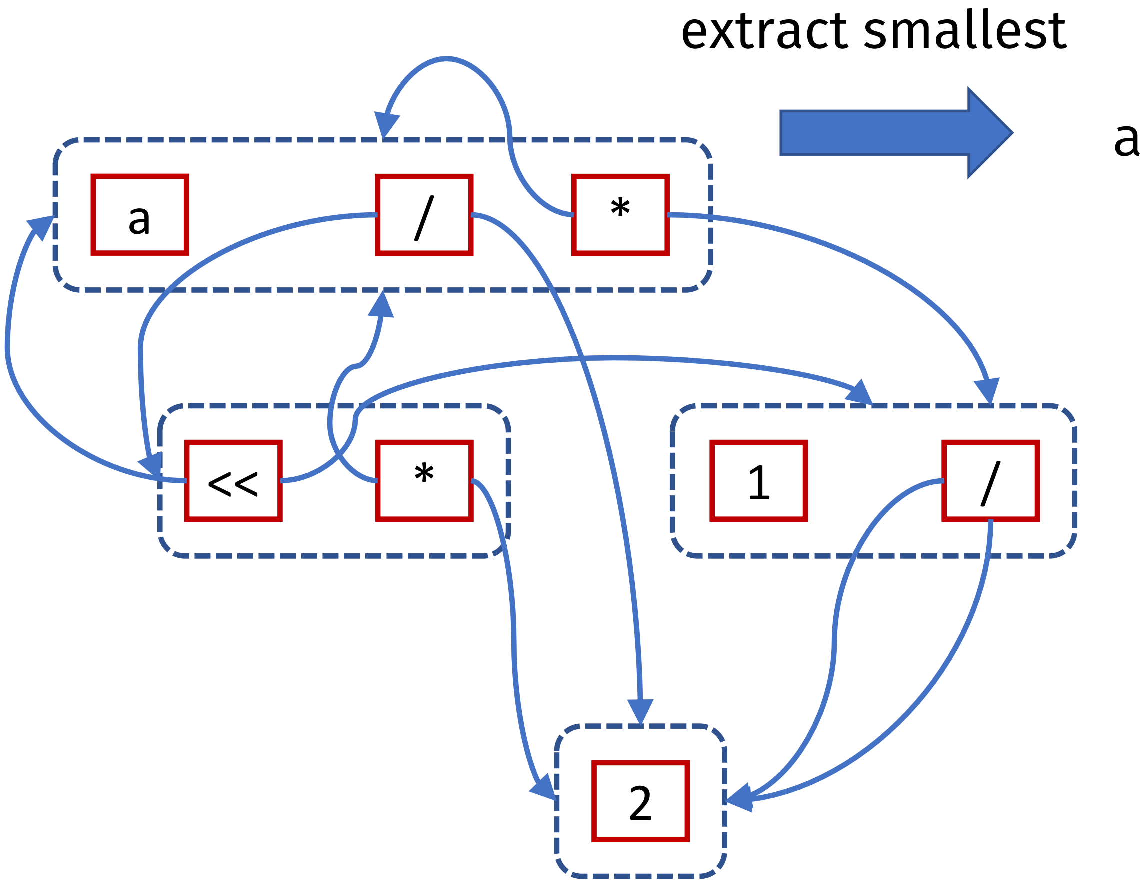


Equality saturation

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