Semantic Foundations of Equality Saturation

Dan Suciu[1], Remy Wang[2], Yihong Zhang[1]
[1] University of Washington [2] University of California, Los Angeles

Equality Saturation:
A general framework for program optimization

Equality Saturation:
A general framework for program optimization

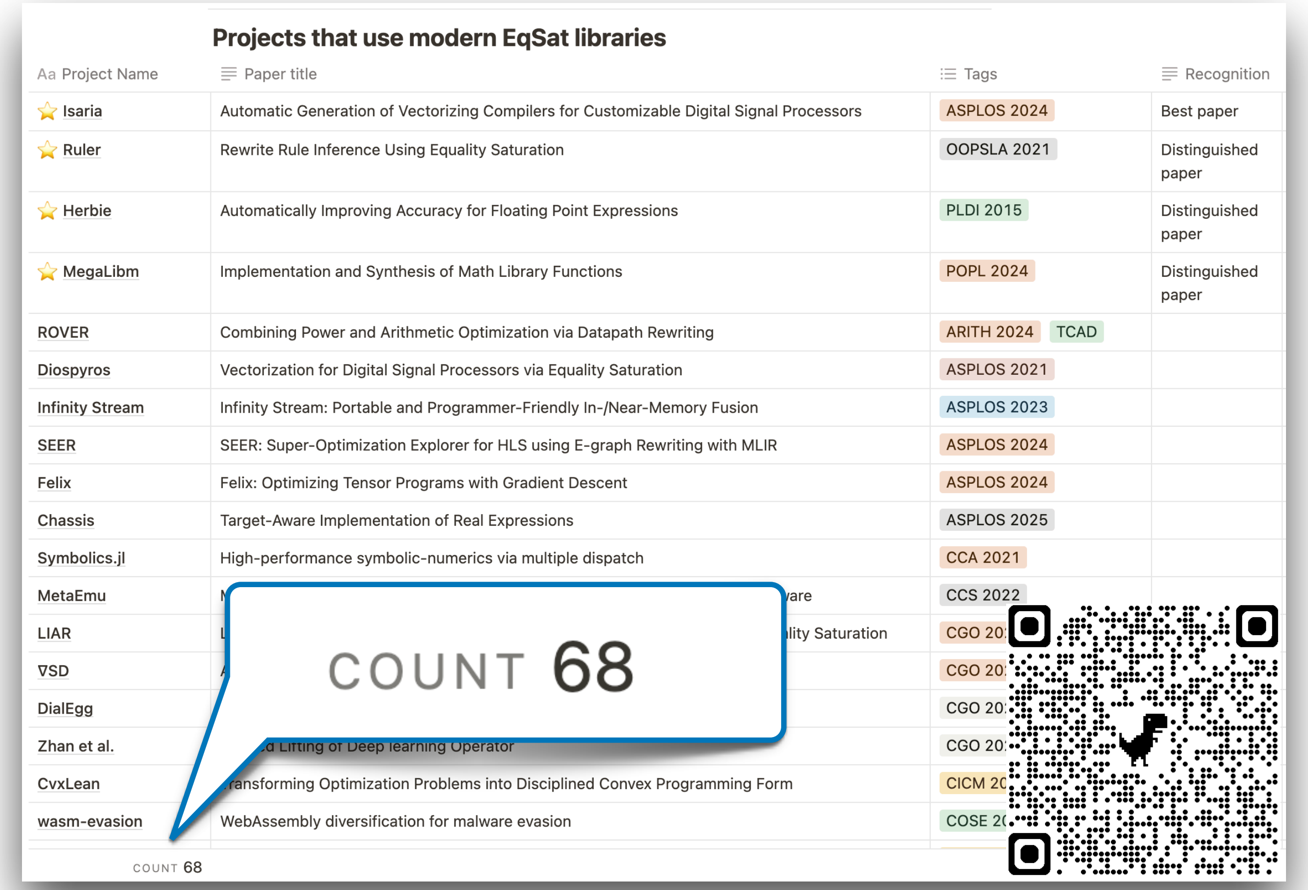
- Invented in 1970s and 2000s and repopularized in 2020.

Equality Saturation: A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.

Equality Saturation:
A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.



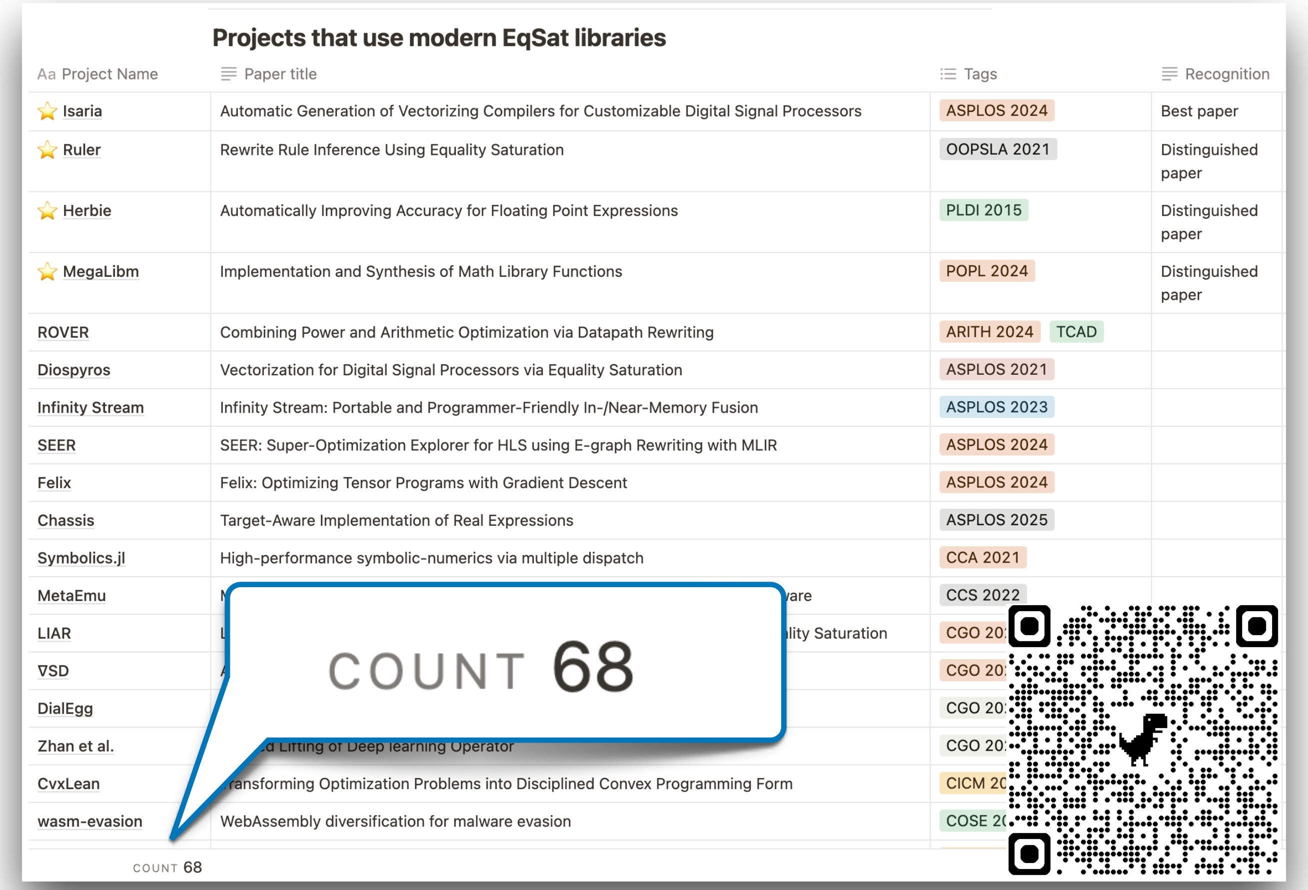
Projects that use modern EqSat libraries

Project Name	Paper title	Tags	Recognition
⭐ Isaria	Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors	ASPLOS 2024	Best paper
⭐ Ruler	Rewrite Rule Inference Using Equality Saturation	OOPSLA 2021	Distinguished paper
⭐ Herbie	Automatically Improving Accuracy for Floating Point Expressions	PLDI 2015	Distinguished paper
⭐ MegaLibm	Implementation and Synthesis of Math Library Functions	POPL 2024	Distinguished paper
ROVER	Combining Power and Arithmetic Optimization via Datapath Rewriting	ARITH 2024 TCAD	
Diospyros	Vectorization for Digital Signal Processors via Equality Saturation	ASPLOS 2021	
Infinity Stream	Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion	ASPLOS 2023	
SEER	SEER: Super-Optimization Explorer for HLS using E-graph Rewriting with MLIR	ASPLOS 2024	
Felix	Felix: Optimizing Tensor Programs with Gradient Descent	ASPLOS 2024	
Chassis	Target-Aware Implementation of Real Expressions	ASPLOS 2025	
Symbolics.jl	High-performance symbolic-numerics via multiple dispatch	CCA 2021	
MetaEmu	M… …are	CCS 2022	
LIAR	L… …lity Saturation	CGO 20…	
∇SD	A…	CGO 20…	
DialEgg		CGO 20…	
Zhan et al.	…d Lifting of Deep learning Operator	CGO 20…	
CvxLean	…ransforming Optimization Problems into Disciplined Convex Programming Form	CICM 20…	
wasm-evasion	WebAssembly diversification for malware evasion	COSE 20…	

COUNT 68

Equality Saturation: A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.
- Thriving community
  - Zulip online chat
  - Monthly community meetings
  - Annual workshops



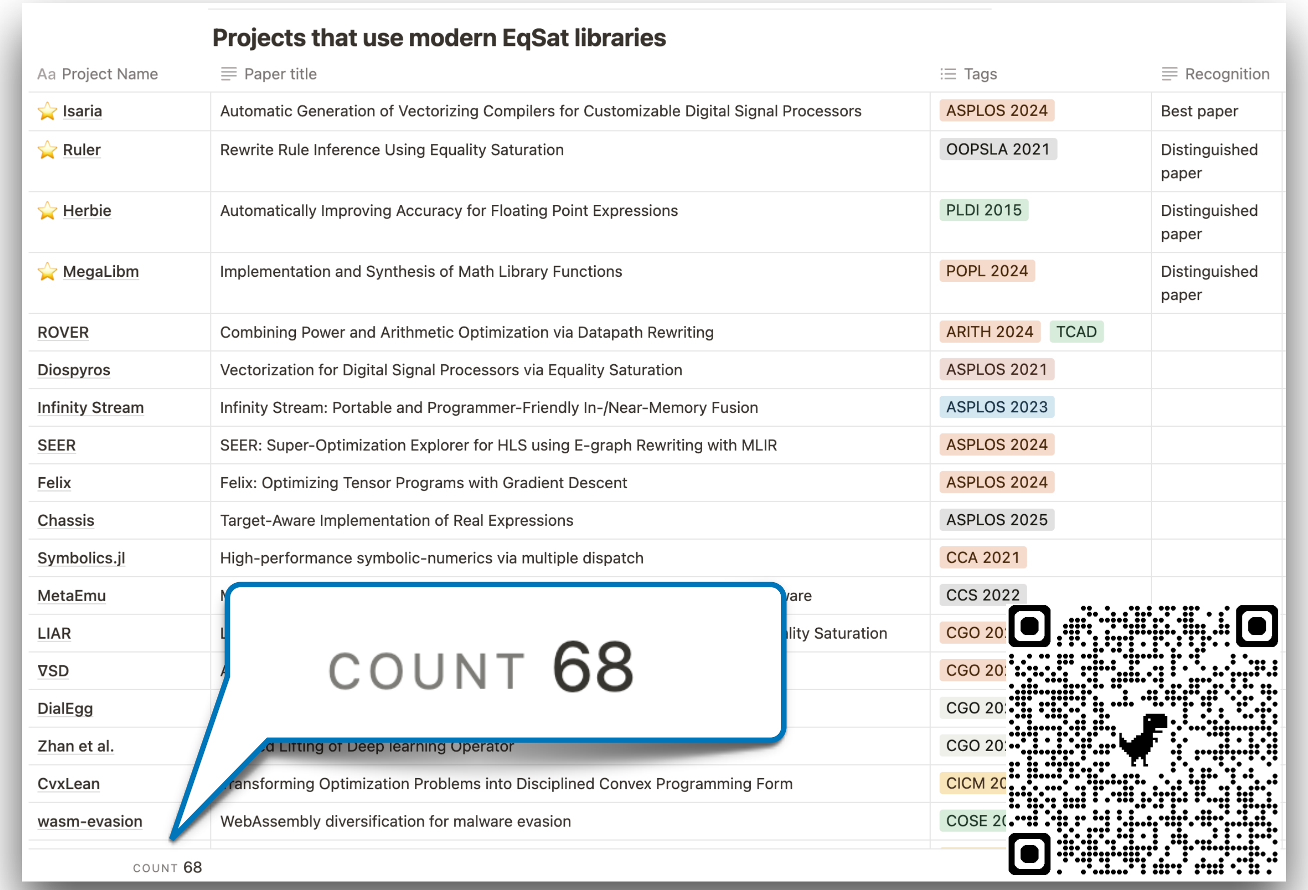
Projects that use modern EqSat libraries

Project Name	Paper title	Tags	Recognition
Isaria	Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors	ASPLOS 2024	Best paper
Ruler	Rewrite Rule Inference Using Equality Saturation	OOPSLA 2021	Distinguished paper
Herbie	Automatically Improving Accuracy for Floating Point Expressions	PLDI 2015	Distinguished paper
MegaLibm	Implementation and Synthesis of Math Library Functions	POPL 2024	Distinguished paper
ROVER	Combining Power and Arithmetic Optimization via Datapath Rewriting	ARITH 2024, TCAD	
Diospyros	Vectorization for Digital Signal Processors via Equality Saturation	ASPLOS 2021	
Infinity Stream	Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion	ASPLOS 2023	
SEER	SEER: Super-Optimization Explorer for HLS using E-graph Rewriting with MLIR	ASPLOS 2024	
Felix	Felix: Optimizing Tensor Programs with Gradient Descent	ASPLOS 2024	
Chassis	Target-Aware Implementation of Real Expressions	ASPLOS 2025	
Symbolics.jl	High-performance symbolic-numerics via multiple dispatch	CCA 2021	
MetaEmu	N… …are	CCS 2022	
LIAR	L… …lity Saturation	CGO 20…	
∇SD	A…	CGO 20…	
DialEgg		CGO 20…	
Zhan et al.	…d Lifting of Deep learning Operator	CGO 20…	
CvxLean	…ransforming Optimization Problems into Disciplined Convex Programming Form	CICM 20…	
wasm-evasion	WebAssembly diversification for malware evasion	COSE 20…	

COUNT 68

Equality Saturation:
A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.
- Thriving community
  - Zulip online chat
  - Monthly community meetings
  - Annual workshops

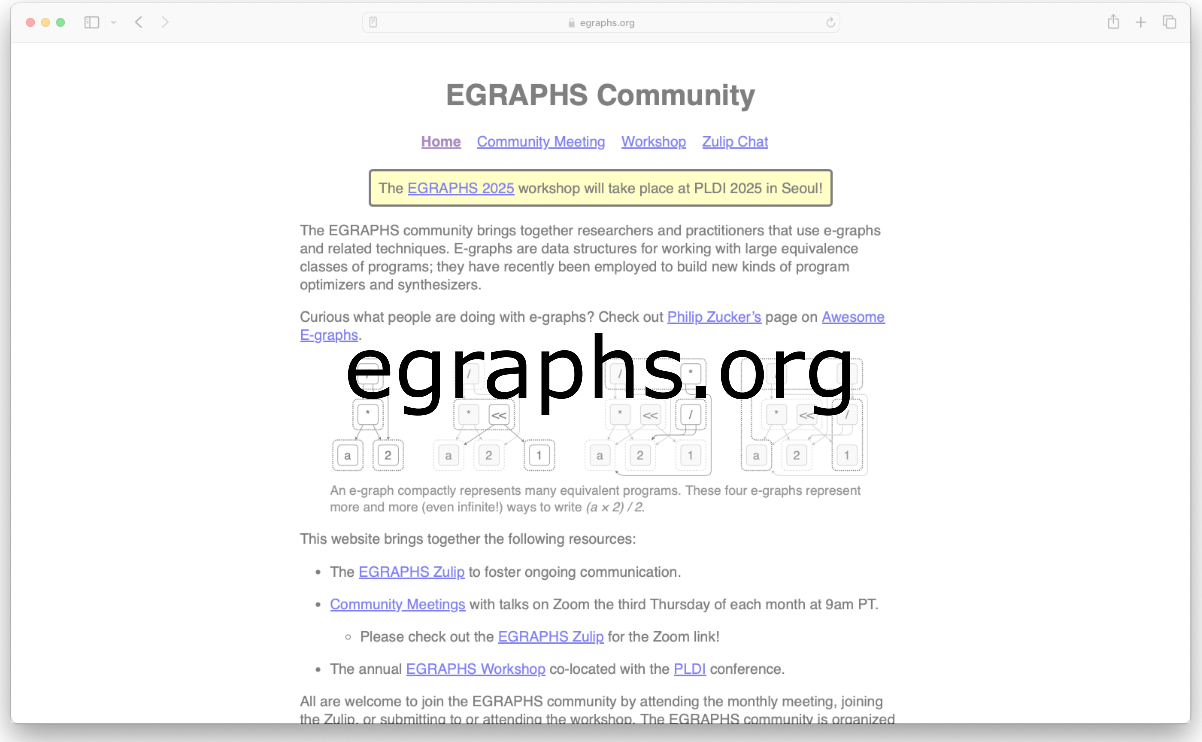


Projects that use modern EqSat libraries

Project Name	Paper title	Tags	Recognition
⭐ Isaria	Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors	ASPLOS 2024	Best paper
⭐ Ruler	Rewrite Rule Inference Using Equality Saturation	OOPSLA 2021	Distinguished paper
⭐ Herbie	Automatically Improving Accuracy for Floating Point Expressions	PLDI 2015	Distinguished paper
⭐ MegaLibm	Implementation and Synthesis of Math Library Functions	POPL 2024	Distinguished paper
ROVER	Combining Power and Arithmetic Optimization via Datapath Rewriting	ARITH 2024, TCAD	
Diospyros	Vectorization for Digital Signal Processors via Equality Saturation	ASPLOS 2021	
Infinity Stream	Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion	ASPLOS 2023	
SEER	SEER: Super-Optimization Explorer for HLS using E-graph Rewriting with MLIR	ASPLOS 2024	
Felix	Felix: Optimizing Tensor Programs with Gradient Descent	ASPLOS 2024	
Chassis	Target-Aware Implementation of Real Expressions	ASPLOS 2025	
Symbolics.jl	High-performance symbolic-numerics via multiple dispatch	CCA 2021	
MetaEmu	M … are	CCS 2022	
LIAR	L … lity Saturation	CGO 20	
∇SD	A	CGO 20	
DialEgg		CGO 20	
Zhan et al.	… d Lifting of Deep learning Operator	CGO 20	
CvxLean	… ransforming Optimization Problems into Disciplined Convex Programming Form	CICM 20	
wasm-evasion	WebAssembly diversification for malware evasion	COSE 20	

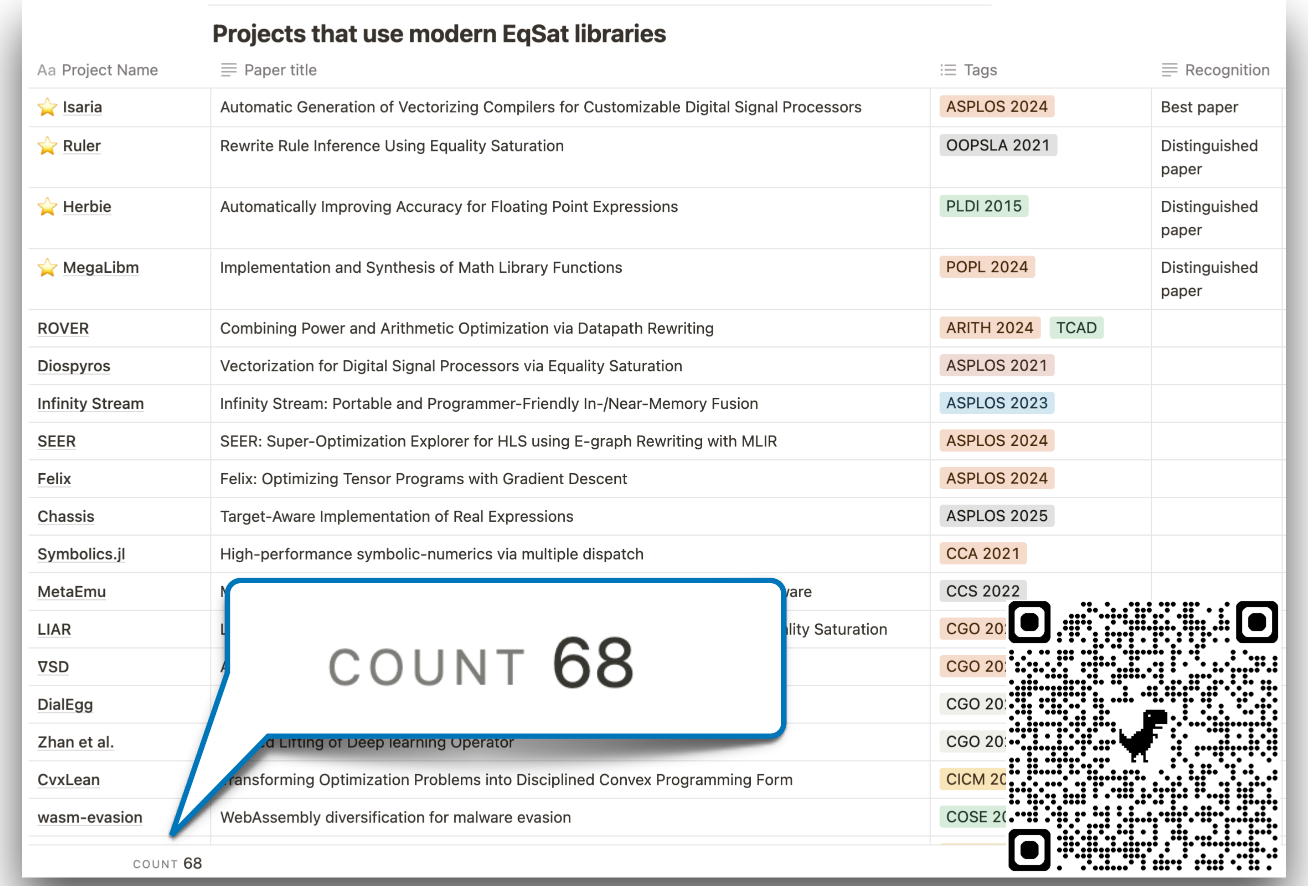
COUNT 68

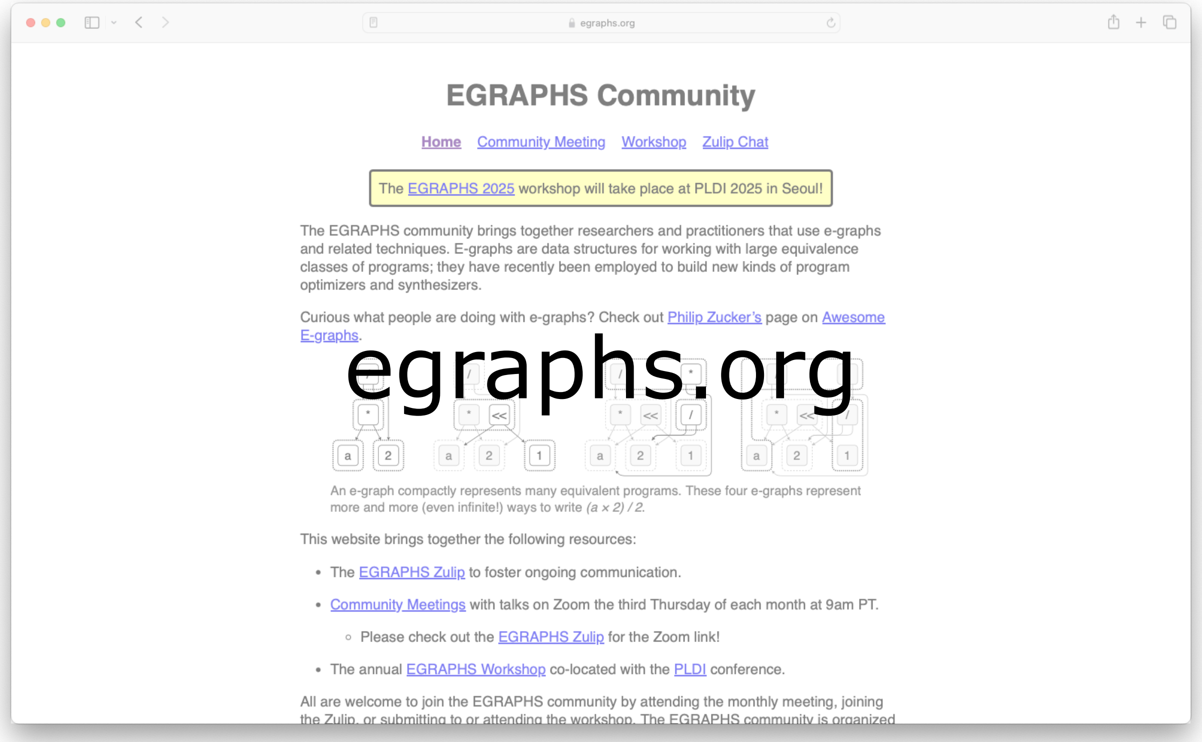
COUNT 68

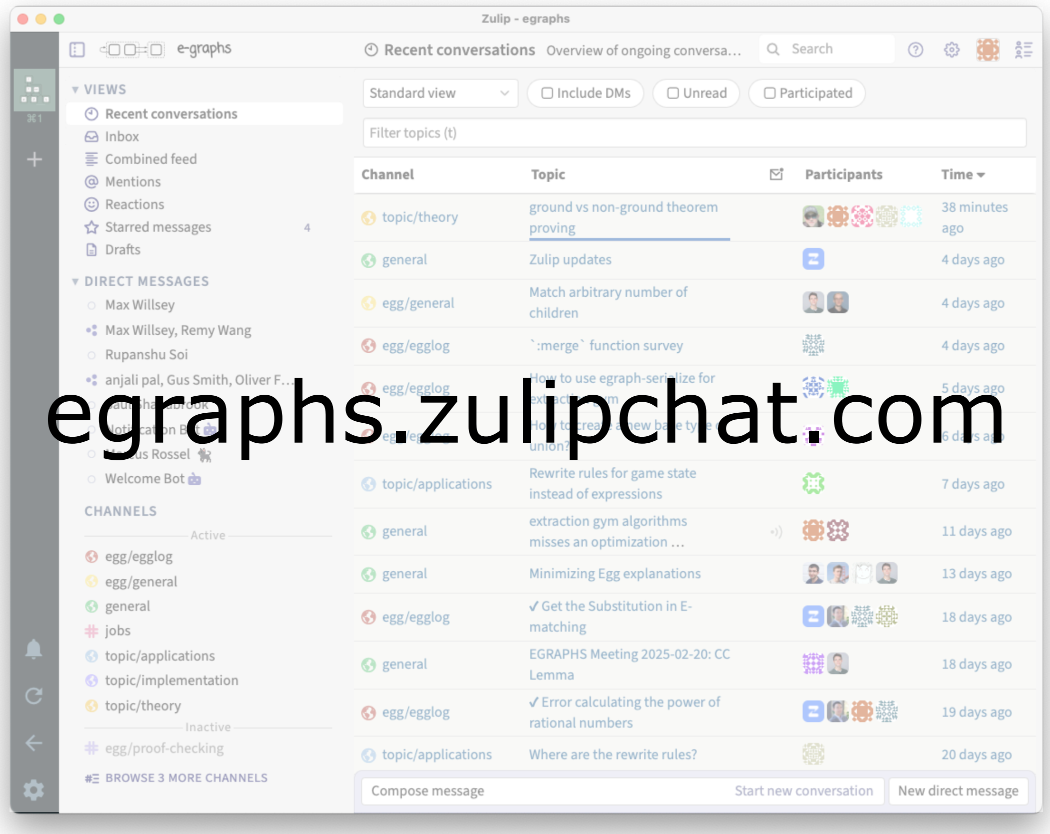


Equality Saturation:
A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.
- Thriving community
  - Zulip online chat
  - Monthly community meetings
  - Annual workshops

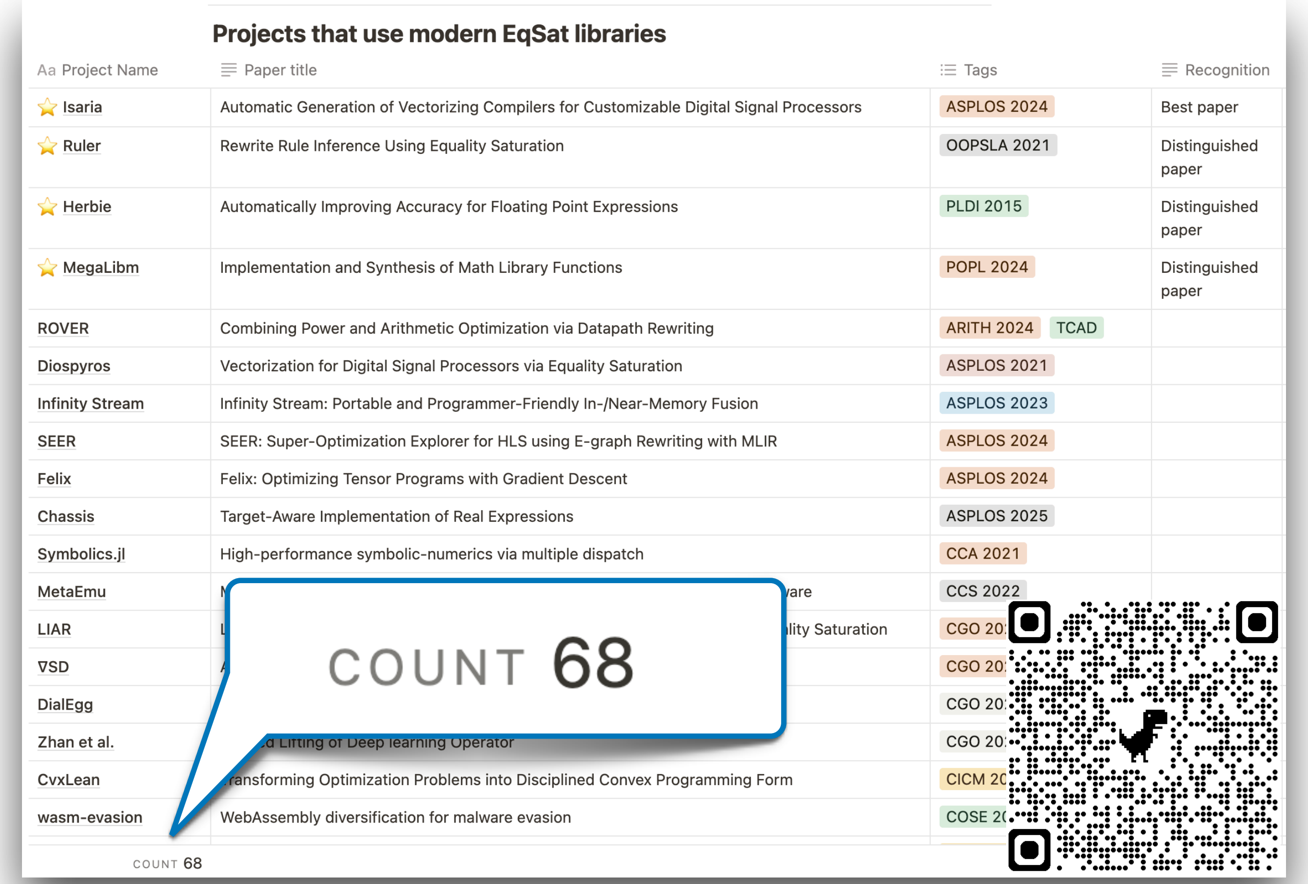






Equality Saturation: A general framework for program optimization

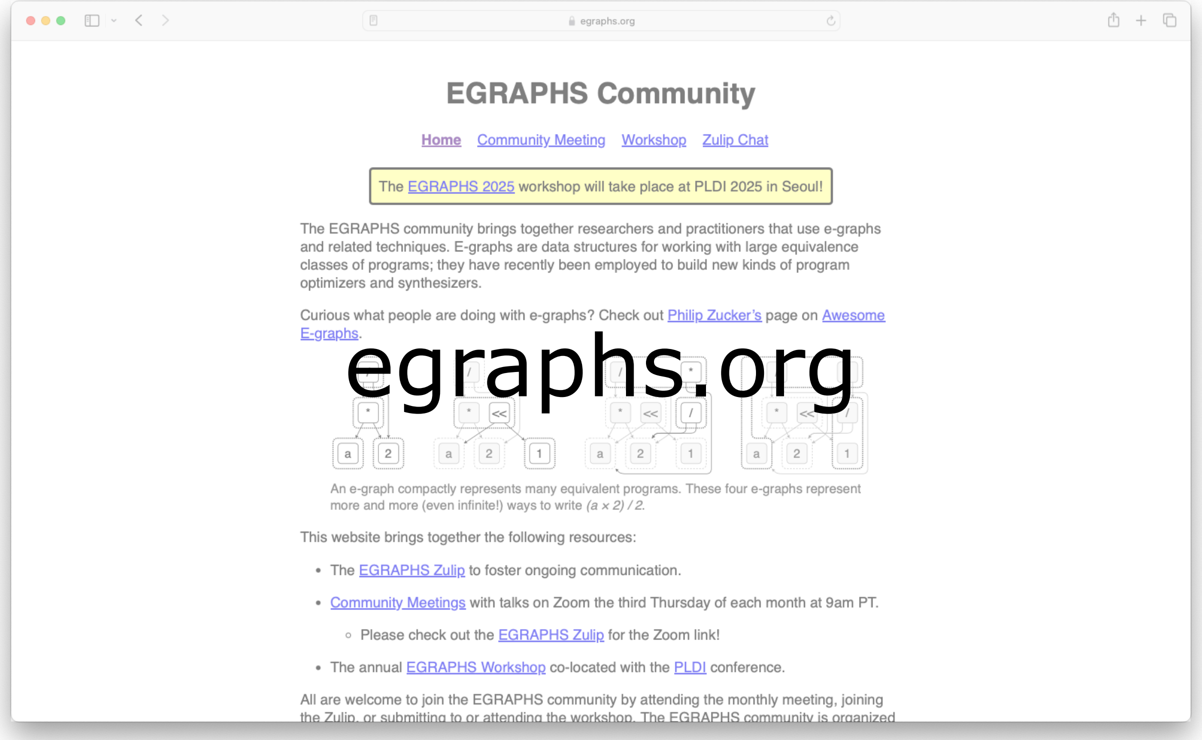
- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.
- Thriving community
  - Zulip online chat
  - Monthly community meetings
  - Annual workshops

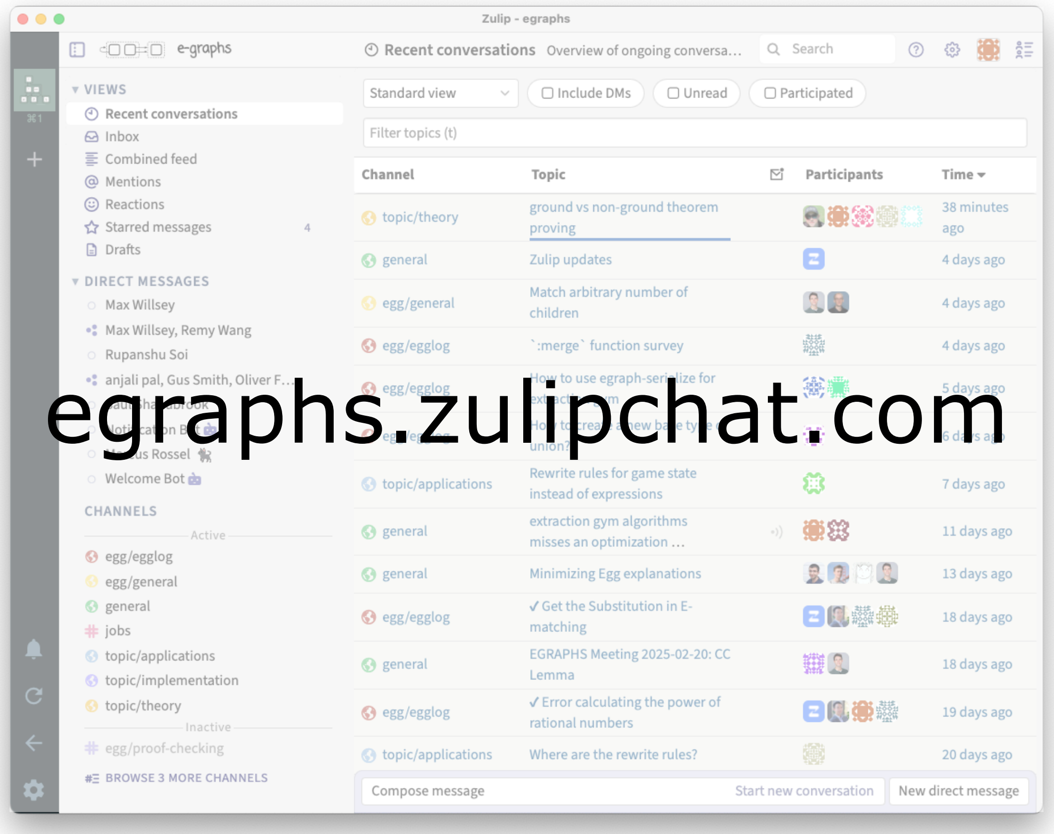


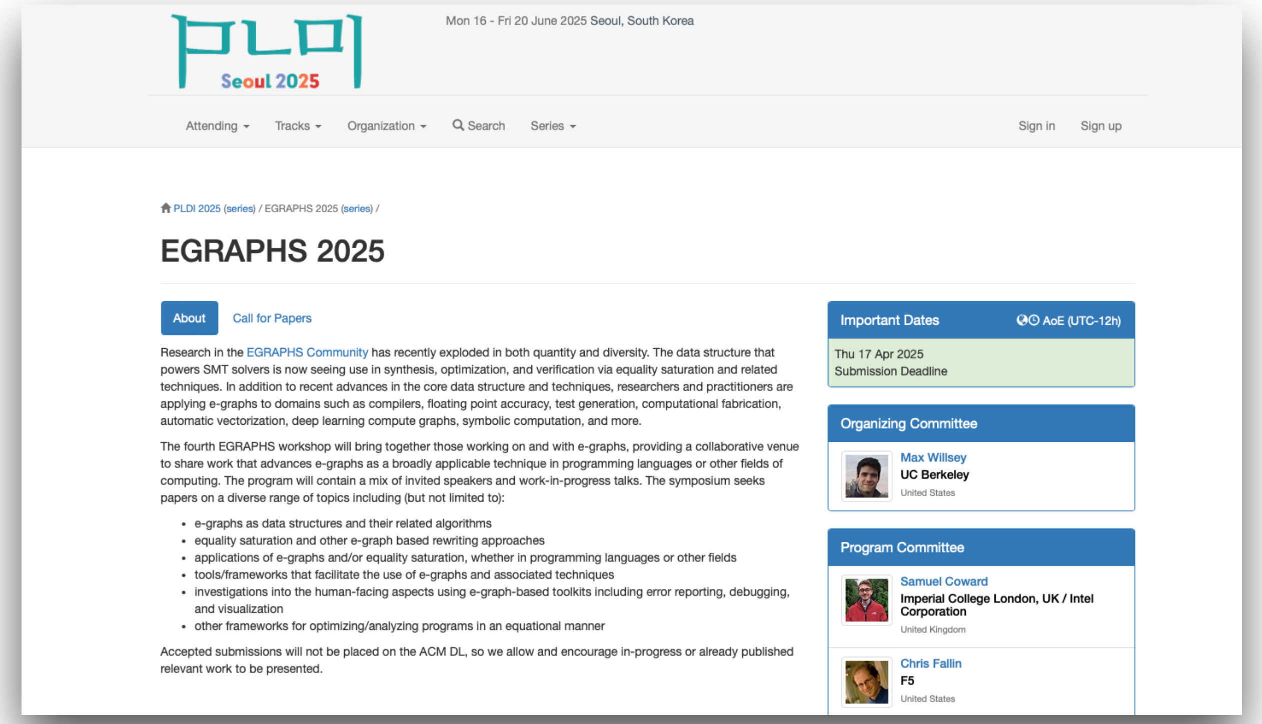
Projects that use modern EqSat libraries

Project Name	Paper title	Tags	Recognition
⭐ Isaria	Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors	ASPLOS 2024	Best paper
⭐ Ruler	Rewrite Rule Inference Using Equality Saturation	OOPSLA 2021	Distinguished paper
⭐ Herbie	Automatically Improving Accuracy for Floating Point Expressions	PLDI 2015	Distinguished paper
⭐ MegaLibm	Implementation and Synthesis of Math Library Functions	POPL 2024	Distinguished paper
ROVER	Combining Power and Arithmetic Optimization via Datapath Rewriting	ARITH 2024 TCAD	
Diospyros	Vectorization for Digital Signal Processors via Equality Saturation	ASPLOS 2021	
Infinity Stream	Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion	ASPLOS 2023	
SEER	SEER: Super-Optimization Explorer for HLS using E-graph Rewriting with MLIR	ASPLOS 2024	
Felix	Felix: Optimizing Tensor Programs with Gradient Descent	ASPLOS 2024	
Chassis	Target-Aware Implementation of Real Expressions	ASPLOS 2025	
Symbolics.jl	High-performance symbolic-numerics via multiple dispatch	CCA 2021	
MetaEmu	M … are	CCS 2022	
LIAR	L … lity Saturation	CGO 20	
∇SD	A	CGO 20	
DialEgg		CGO 20	
Zhan et al.	… d Lifting of Deep learning Operator	CGO 20	
CvxLean	… ransforming Optimization Problems into Disciplined Convex Programming Form	CICM 20	
wasm-evasion	WebAssembly diversification for malware evasion	COSE 20	

COUNT 68



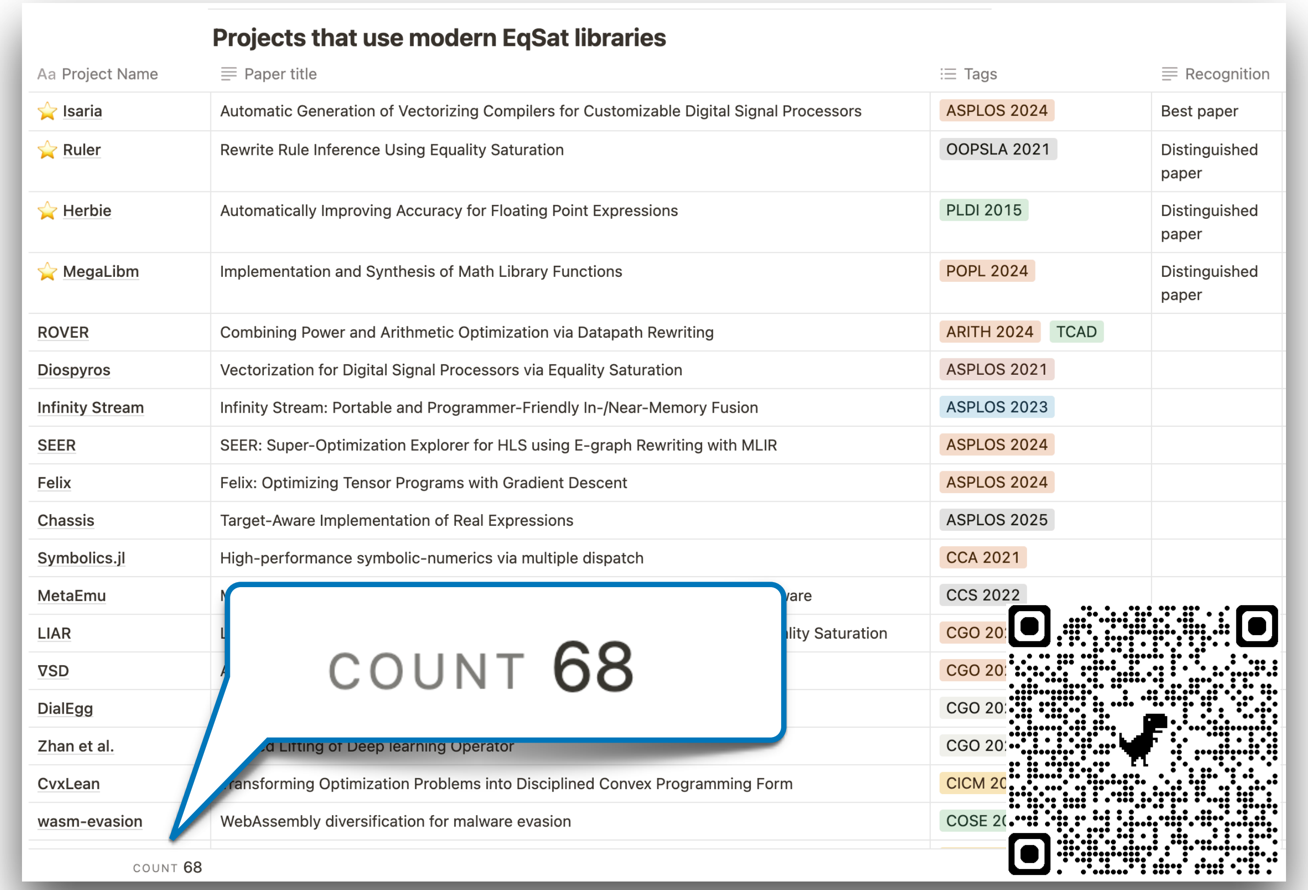




Equality Saturation:
A general framework for program optimization

- Invented in 1970s and 2000s and repopularized in 2020.
- Adopted in 50+ projects since then.
- Thriving community
  - Zulip online chat
  - Monthly community meetings
  - Annual workshops

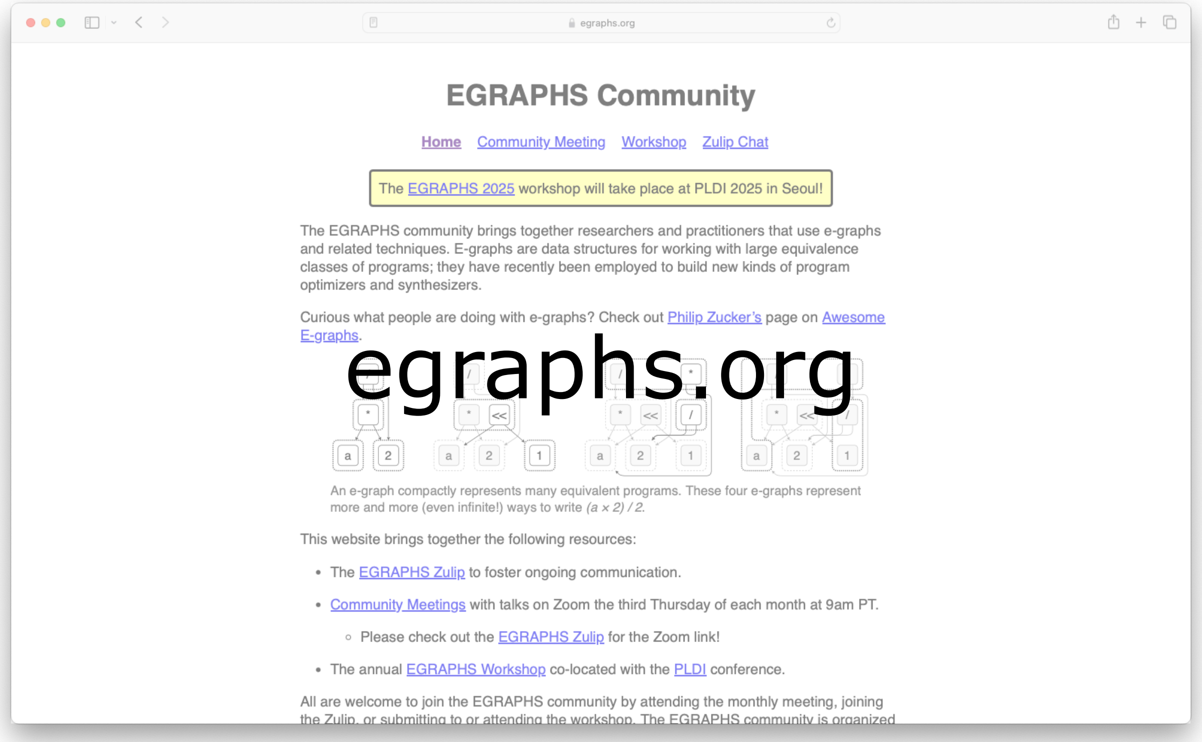
<u>Database theory community can help!</u>

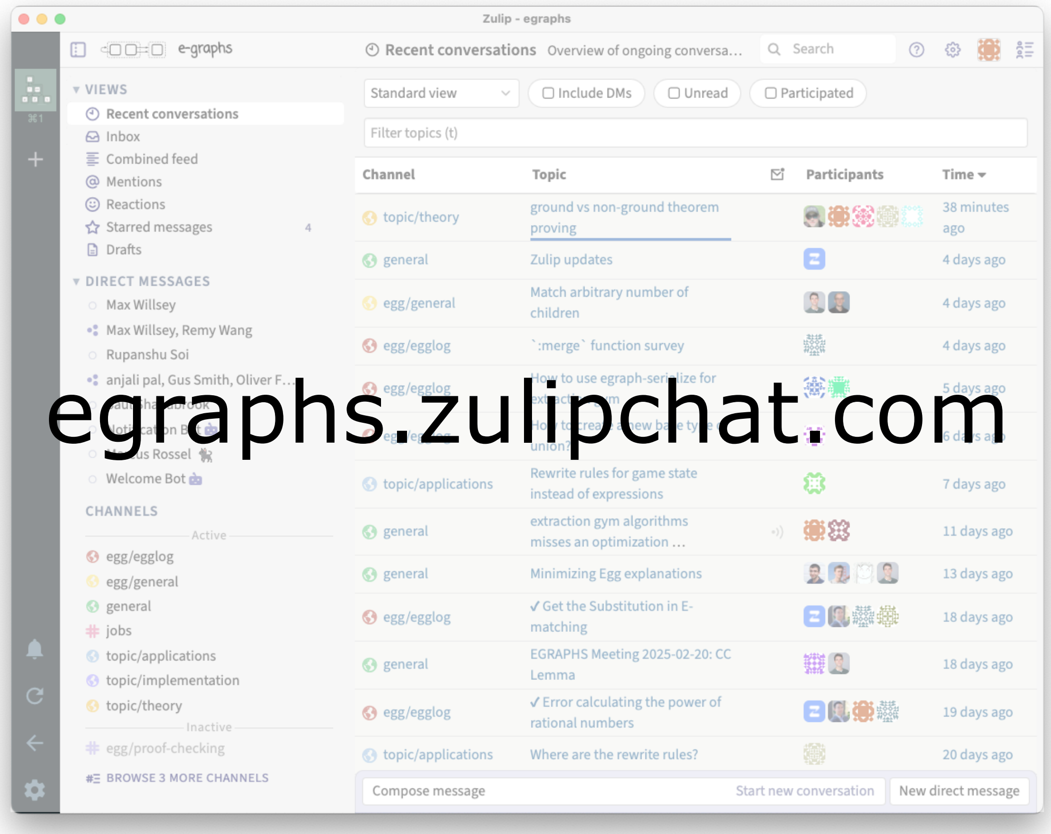


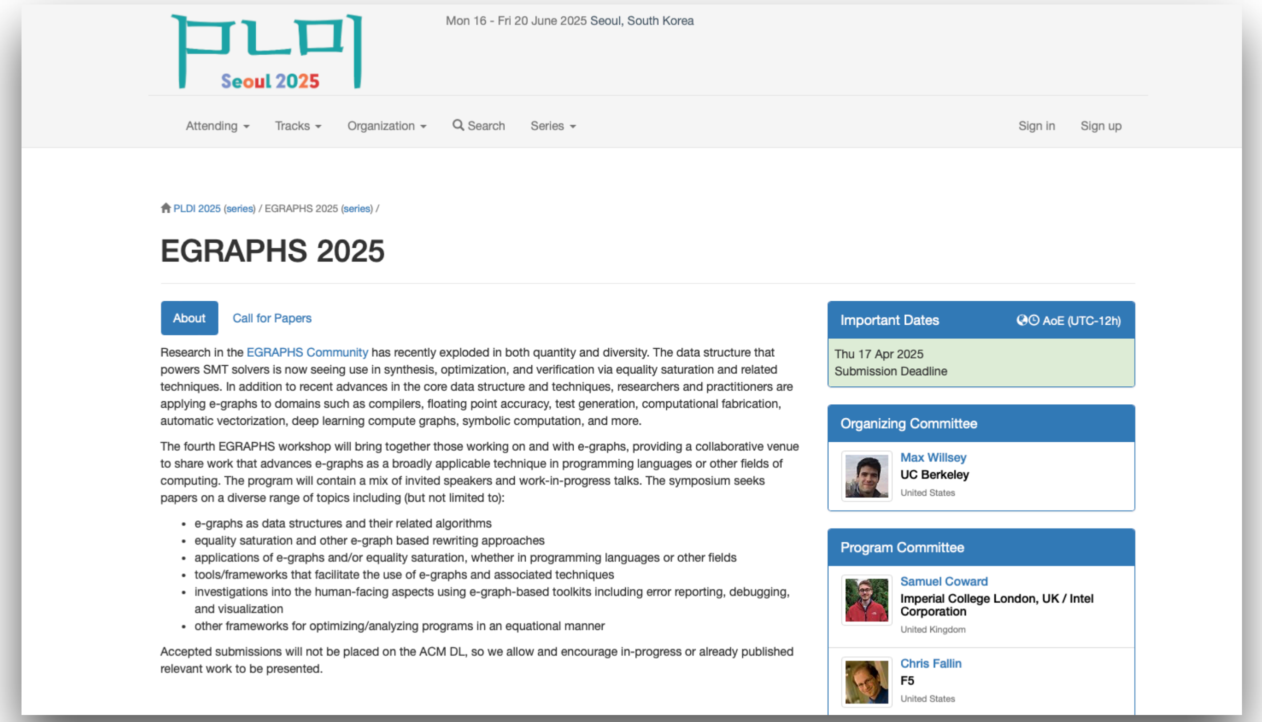
Projects that use modern EqSat libraries

Project Name	Paper title	Tags	Recognition
Isaria	Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors	ASPLOS 2024	Best paper
Ruler	Rewrite Rule Inference Using Equality Saturation	OOPSLA 2021	Distinguished paper
Herbie	Automatically Improving Accuracy for Floating Point Expressions	PLDI 2015	Distinguished paper
MegaLibm	Implementation and Synthesis of Math Library Functions	POPL 2024	Distinguished paper
ROVER	Combining Power and Arithmetic Optimization via Datapath Rewriting	ARITH 2024 TCAD	
Diospyros	Vectorization for Digital Signal Processors via Equality Saturation	ASPLOS 2021	
Infinity Stream	Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion	ASPLOS 2023	
SEER	SEER: Super-Optimization Explorer for HLS using E-graph Rewriting with MLIR	ASPLOS 2024	
Felix	Felix: Optimizing Tensor Programs with Gradient Descent	ASPLOS 2024	
Chassis	Target-Aware Implementation of Real Expressions	ASPLOS 2025	
Symbolics.jl	High-performance symbolic-numerics via multiple dispatch	CCA 2021	
MetaEmu	...are	CCS 2022	
LIAR	...lity Saturation	CGO 20..	
∇SD		CGO 20..	
DialEgg		CGO 20..	
Zhan et al.	...Lifting of Deep learning Operator	CGO 20..	
CvxLean	...nsforming Optimization Problems into Disciplined Convex Programming Form	CICM 20..	
wasm-evasion	WebAssembly diversification for malware evasion	COSE 20..	

COUNT 68







This paper

Defines a rigorous semantics to Equality Saturation (EqSat).

Studies EqSat in relationship to Term Rewriting and the Chase.

Proves the undecidability of EqSat termination in three cases.

The word problem and term rewriting

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \dots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \ldots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \ldots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \dots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \dots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

- Term rewriting for word problem:

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \dots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \dots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

- Term rewriting for word problem:
  - Use a Term Rewriting System (TRS) R capturing axioms E.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\varnothing)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \ldots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

- Term rewriting for word problem:
  - Use a Term Rewriting System (TRS) R capturing axioms E.
  - Apply $\rightarrow_R$ to u and v and check if $\exists w \, . \; u \rightarrow_R^* w \leftarrow_R^* v$.

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \ldots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

- Term rewriting for word problem:
  - Use a Term Rewriting System (TRS) R capturing axioms E.
  - Apply $\to_R$ to u and v and check if $\exists w \,.\, u \to_R^* w \leftarrow_R^* v$.

Problem: $(a^{-1} \cdot b^{-1})^{-1} \overset{?}{\approx} b \cdot a$

The word problem and term rewriting

- Signature $\Sigma := \{f_1, f_2, \ldots\}$.
- Patterns $T_\Sigma(V)$ for set of vars V.
- Ground terms $T_\Sigma \quad \big(:= T_\Sigma(\emptyset)\big)$.
- The (ground) word problem
  - Input: $E = \{s_1 \approx t_1, \ldots\}$ and $u, v \in T_\Sigma$.
  - Ask: $u \overset{?}{\approx}_E v$.
- Undecidable in general.

- Term rewriting for word problem:
  - Use a Term Rewriting System (TRS) R capturing axioms E.
  - Apply $\to_R$ to u and v and check if $\exists w \,.\, u \to_R^* w \leftarrow_R^* v$.

Problem: $(a^{-1} \cdot b^{-1})^{-1} \overset{?}{\approx} b \cdot a$

Yes.

$$\begin{aligned}(a^{-1} \cdot b^{-1})^{-1} &\to (b^{-1})^{-1} \cdot (a^{-1})^{-1} \\ &\to b \cdot (a^{-1})^{-1} \\ &\to b \cdot a\end{aligned}$$

Program optimization with term rewriting

Program optimization with term rewriting

Program Optimization

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$$

$$x \times 1 \to x \qquad x \times 2 \to x \ll 1$$

...

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$$

$$x \times 1 \to x \qquad x \times 2 \to x \ll 1$$

...

$$(a \times 2) \div 2$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$$

$$x \times 1 \to x \qquad x \times 2 \to x \ll 1$$

...

$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2)$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$

$x \times 1 \to x \qquad x \times 2 \to x \ll 1$

...

$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \rightarrow a \times 1$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$

$x \times 1 \to x \qquad x \times 2 \to x \ll 1$

...

$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \to a \times 1 \to a$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$x \div x \to 1$ $\quad$ $(x \times y) \div z \to x \times (y \div z)$

$x \times 1 \to x$ $\quad$ $x \times 2 \to x \ll 1$

...

$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \longrightarrow a \times 1 \longrightarrow a$$

$$(a \times 2) \div 2 \longrightarrow (a \ll 1)/2$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$x \div x \to 1$ $\qquad$ $(x \times y) \div z \to x \times (y \div z)$

$x \times 1 \to x$ $\qquad$ $x \times 2 \to x \ll 1$

...

$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \to a \times 1 \to a$$

$$(a \times 2) \div 2 \searrow (a \ll 1)/2 \to ?$$

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$x \div x \to 1$ $\quad$ $(x \times y) \div z \to x \times (y \div z)$

$x \times 1 \to x$ $\quad$ $x \times 2 \to x \ll 1$

...

$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \to a \times 1 \to a$$

$$\searrow (a \ll 1)/2 \to ?$$

Term rewriting is greedy!

Program optimization with term rewriting

Program Optimization

- Input:
  - A set of axioms E,
  - A program s,
  - Cost function C,
- Output:
  - Optimized program

$$t = \arg\ \min_{t \in [s]_E} C(t)$$

where

$$[s]_E := \{t \mid t \in T_\Sigma \,.\, t \approx_E s\}.$$

Ruled-based program optimization

$$x \div x \to 1 \qquad (x \times y) \div z \to x \times (y \div z)$$

$$x \times 1 \to x \qquad x \times 2 \to x \ll 1$$

...

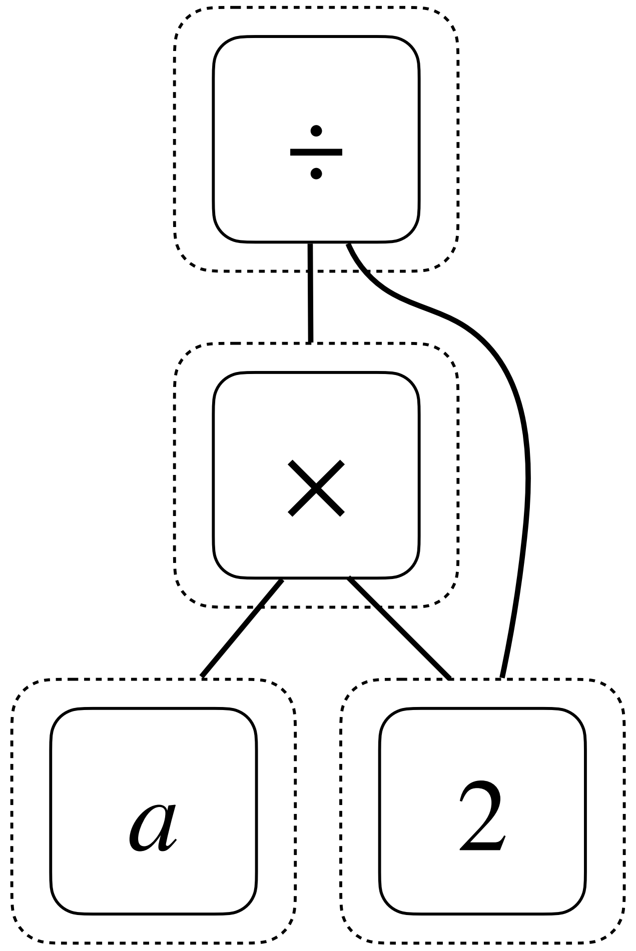
$$(a \times 2) \div 2 \longrightarrow a \times (2 \div 2) \to a \times 1 \to a$$

$$(a \times 2) \div 2 \searrow (a \ll 1)/2 \to ?$$

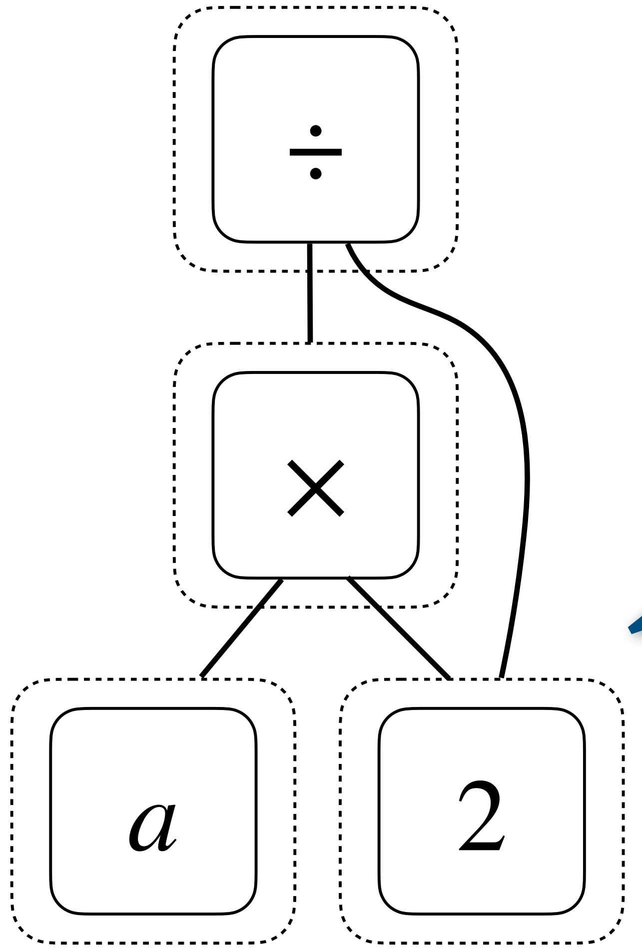
Term rewriting is greedy!

Equality Saturation is an algorithm to efficiently explore the program space defined by rules.

E-graphs and Equality Saturation

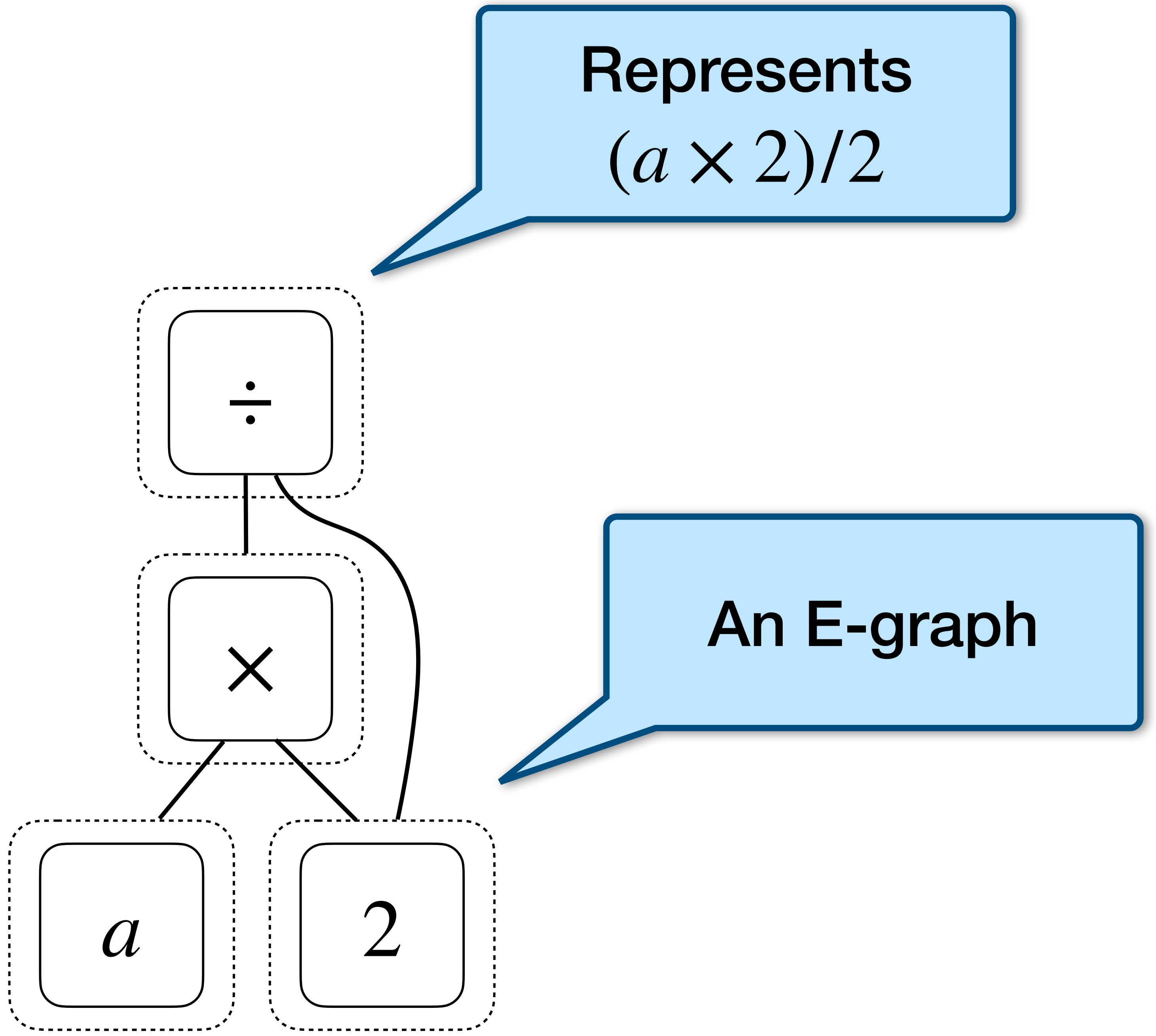


E-graphs and Equality Saturation

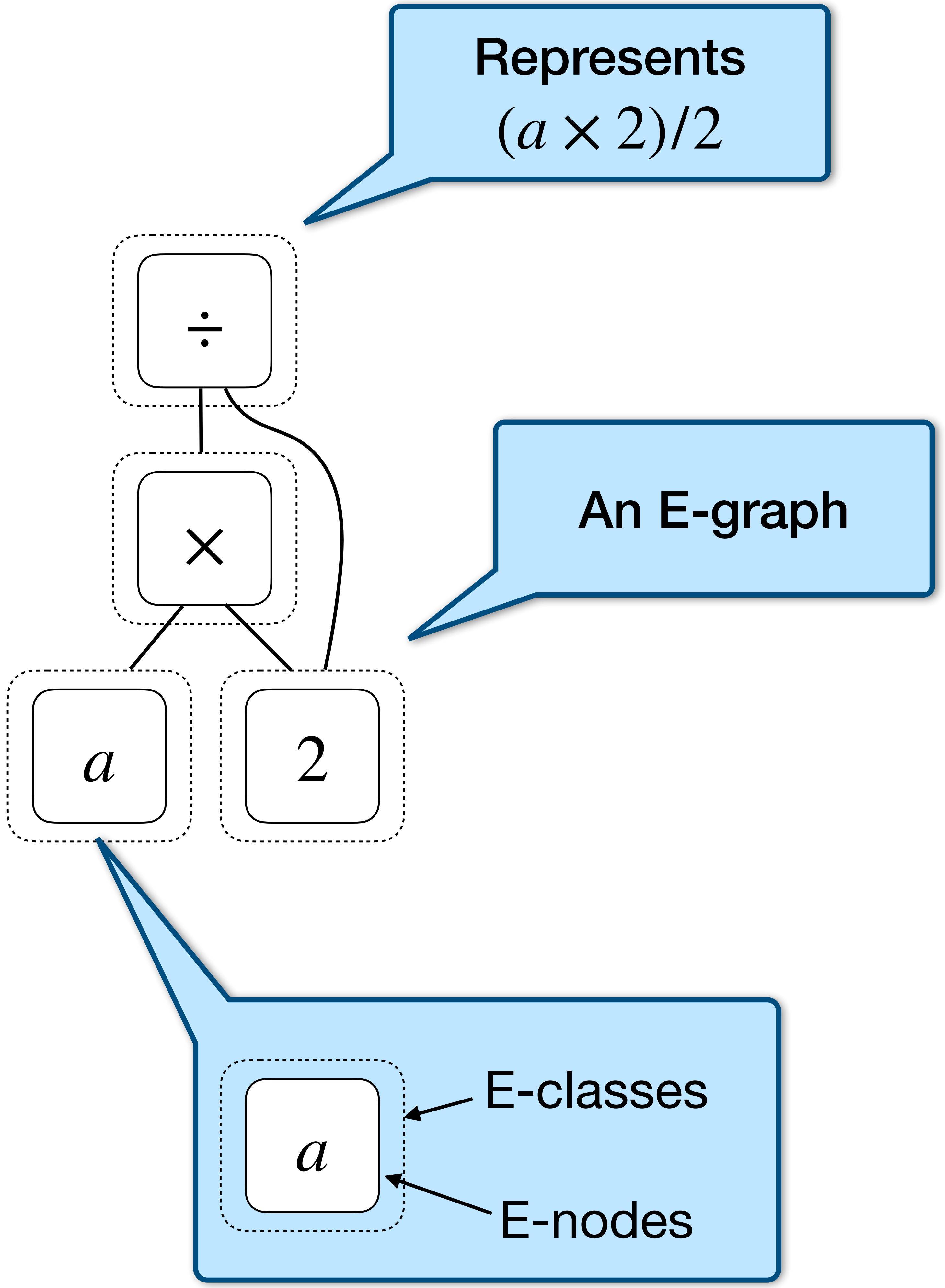


An E-graph

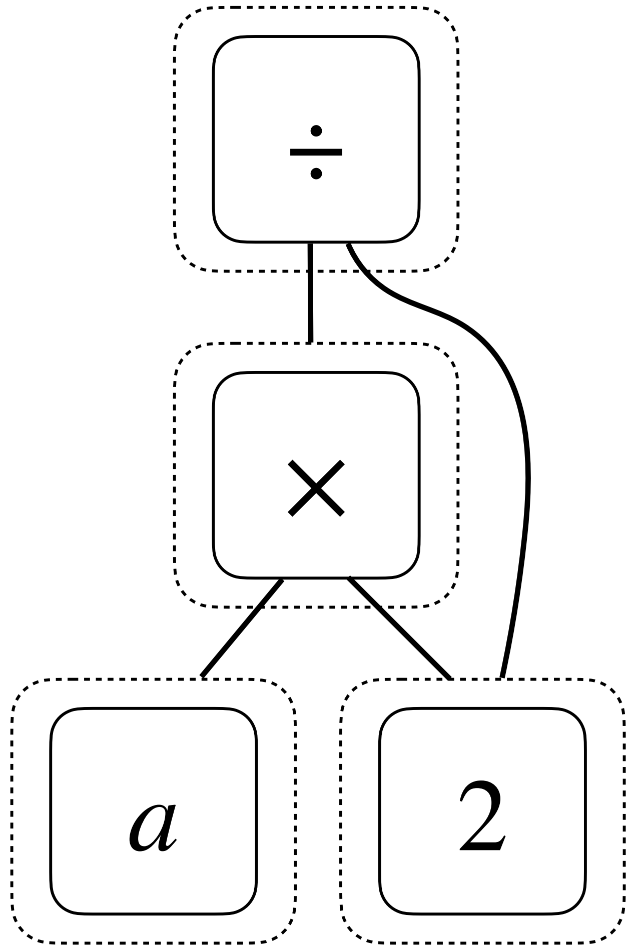
E-graphs and Equality Saturation



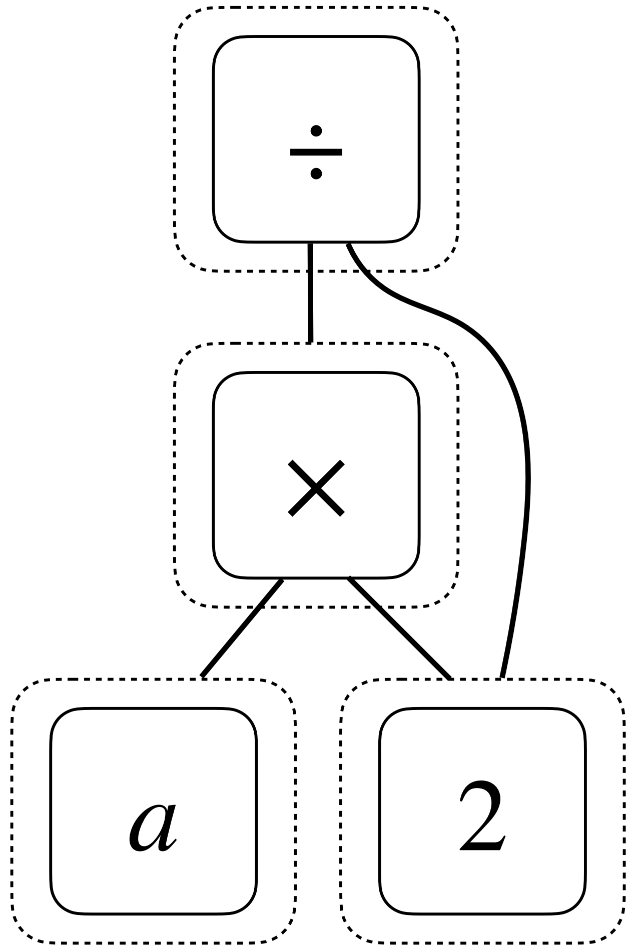
E-graphs and Equality Saturation

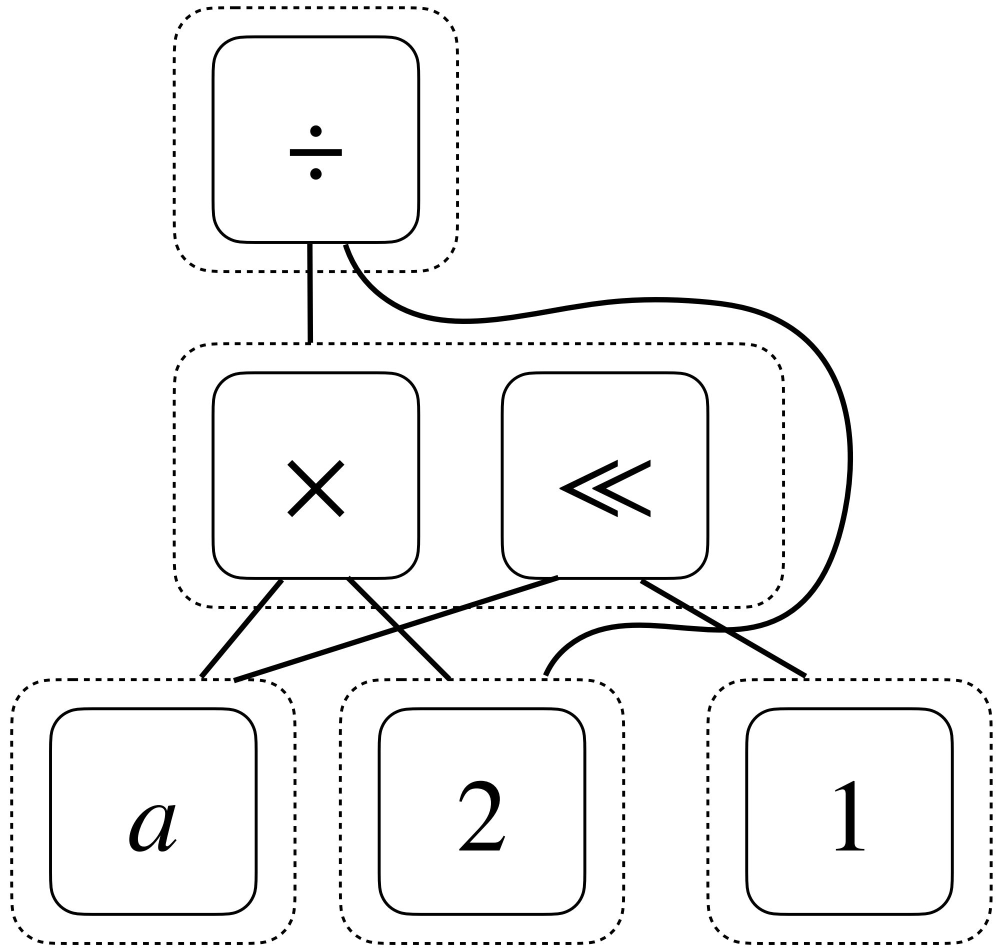


E-graphs and Equality Saturation



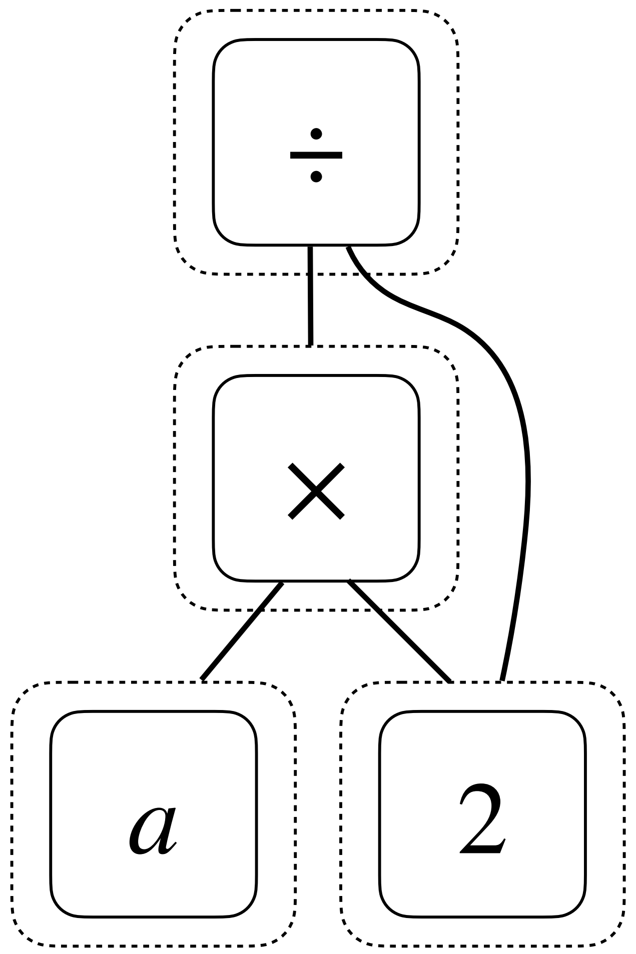
E-graphs and Equality Saturation

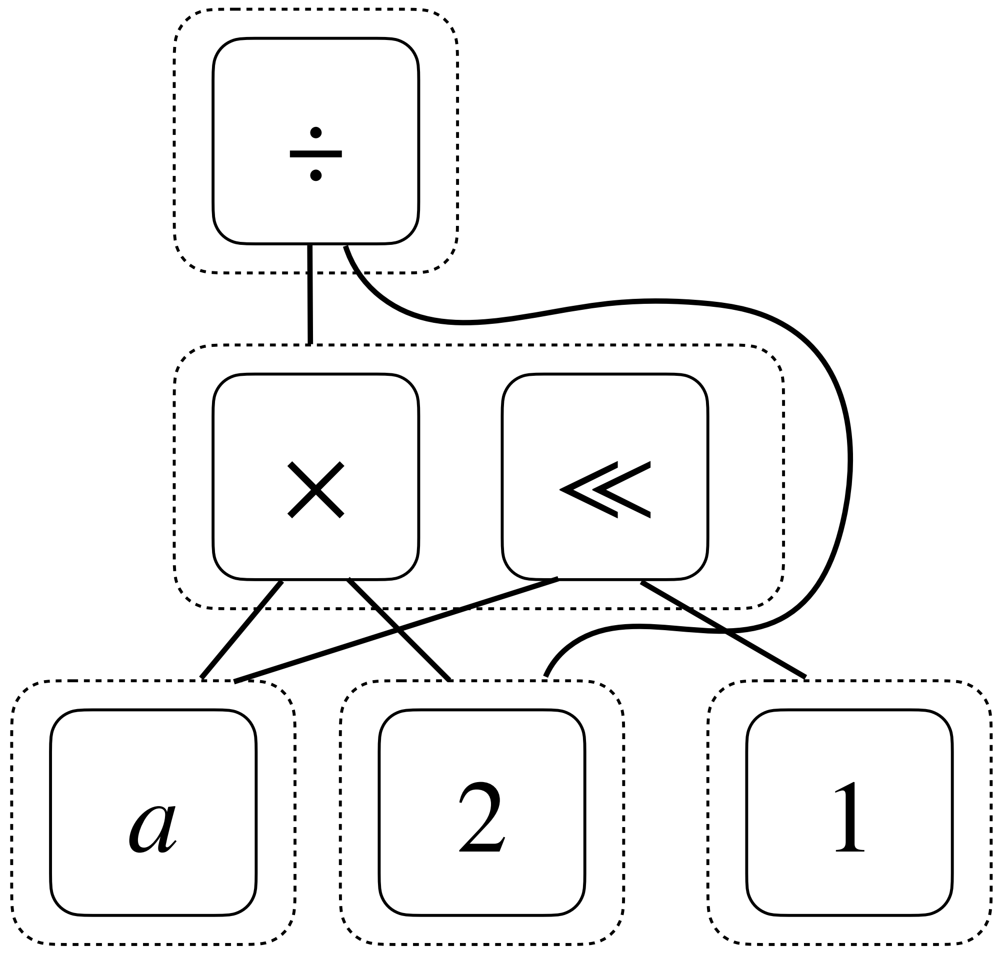


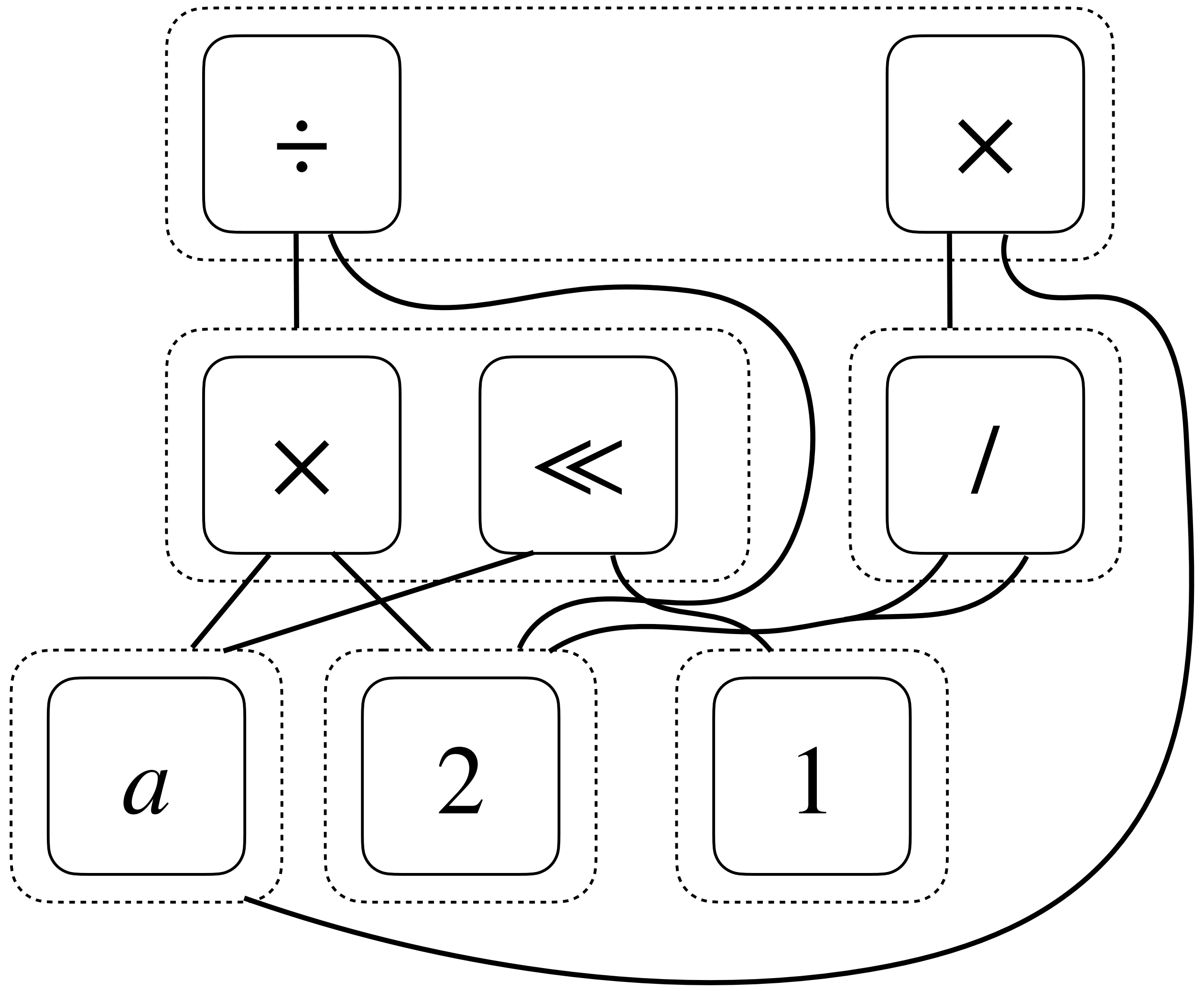



```
x * 2 => x << 1
```

E-graphs and Equality Saturation



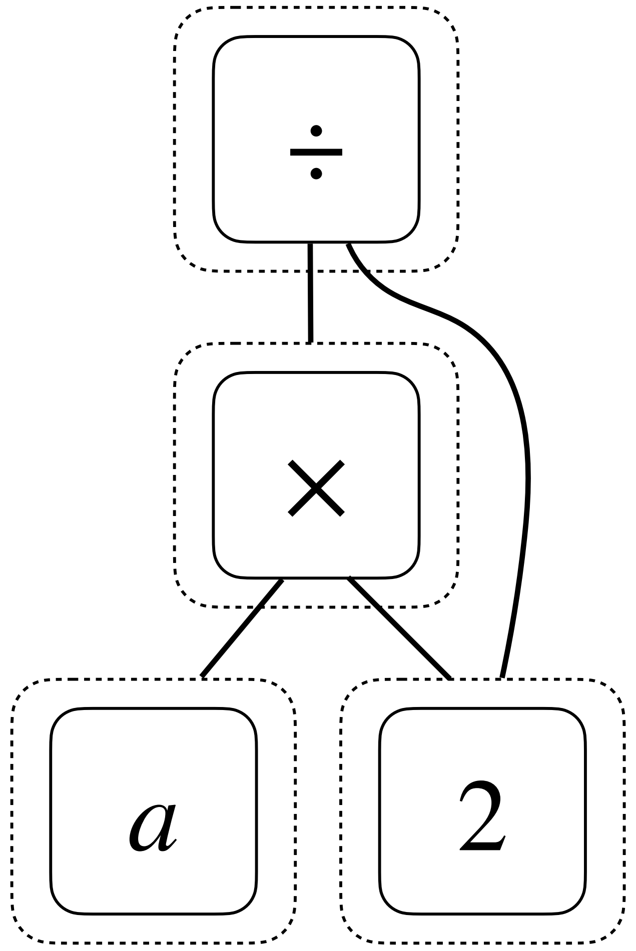


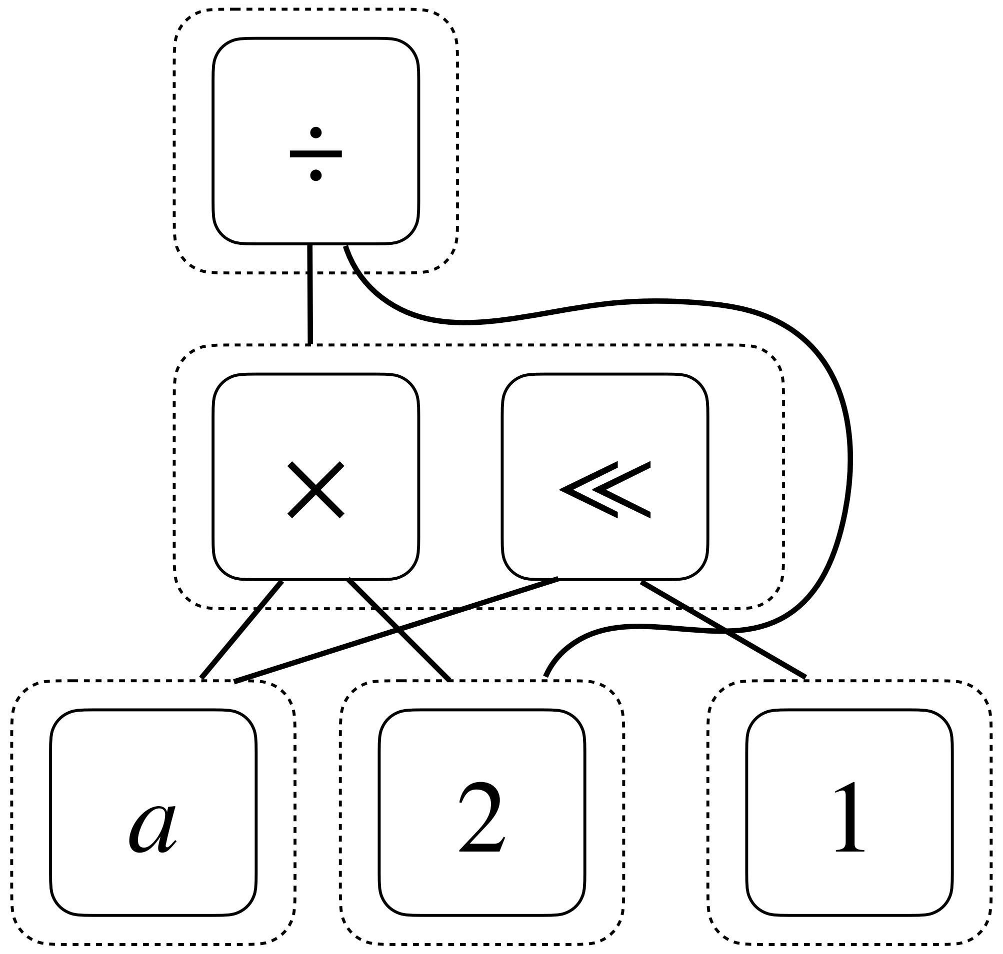


x * 2 => x << 1

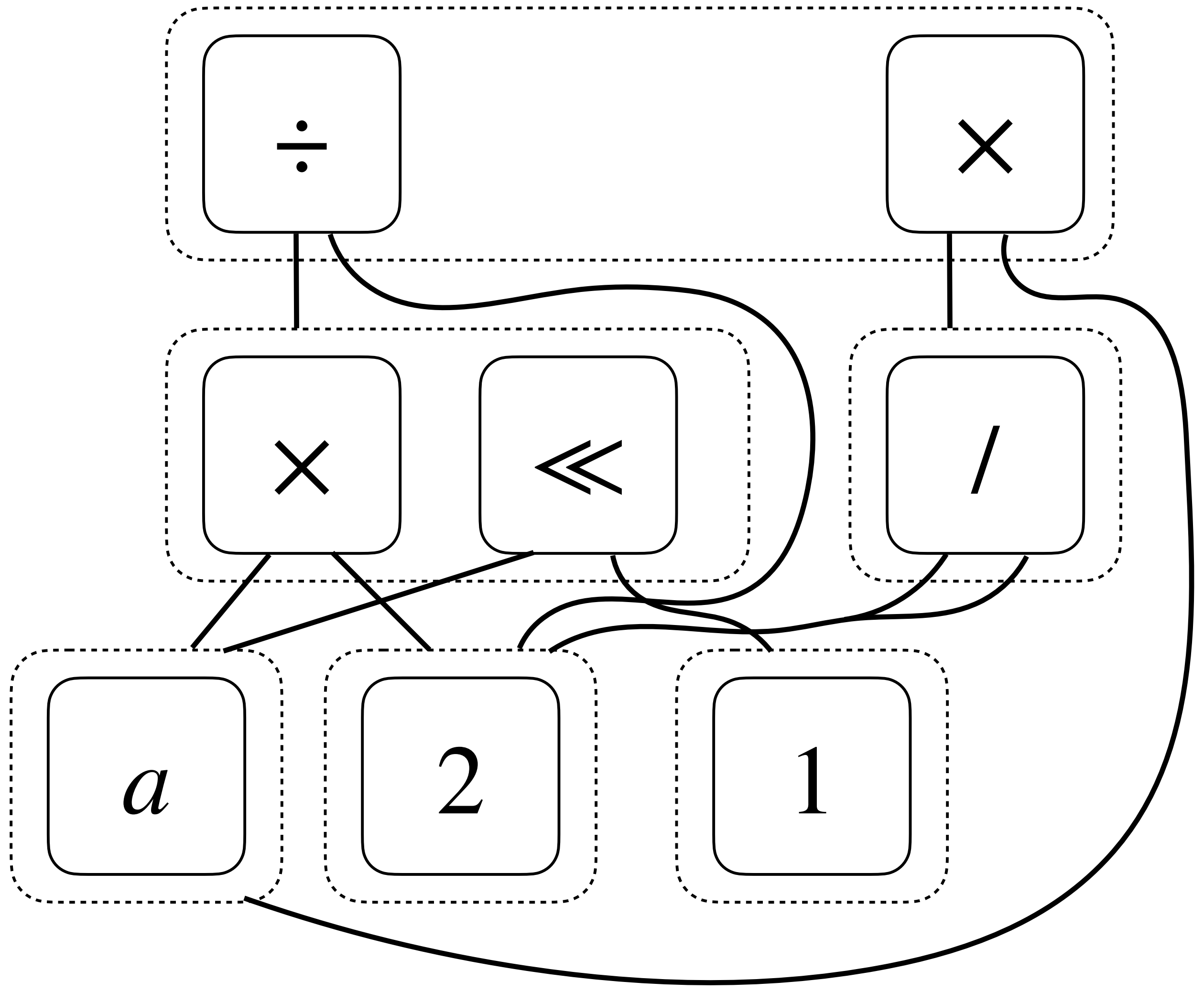
(x * y) / z => x * (y / z)

E-graphs and Equality Saturation

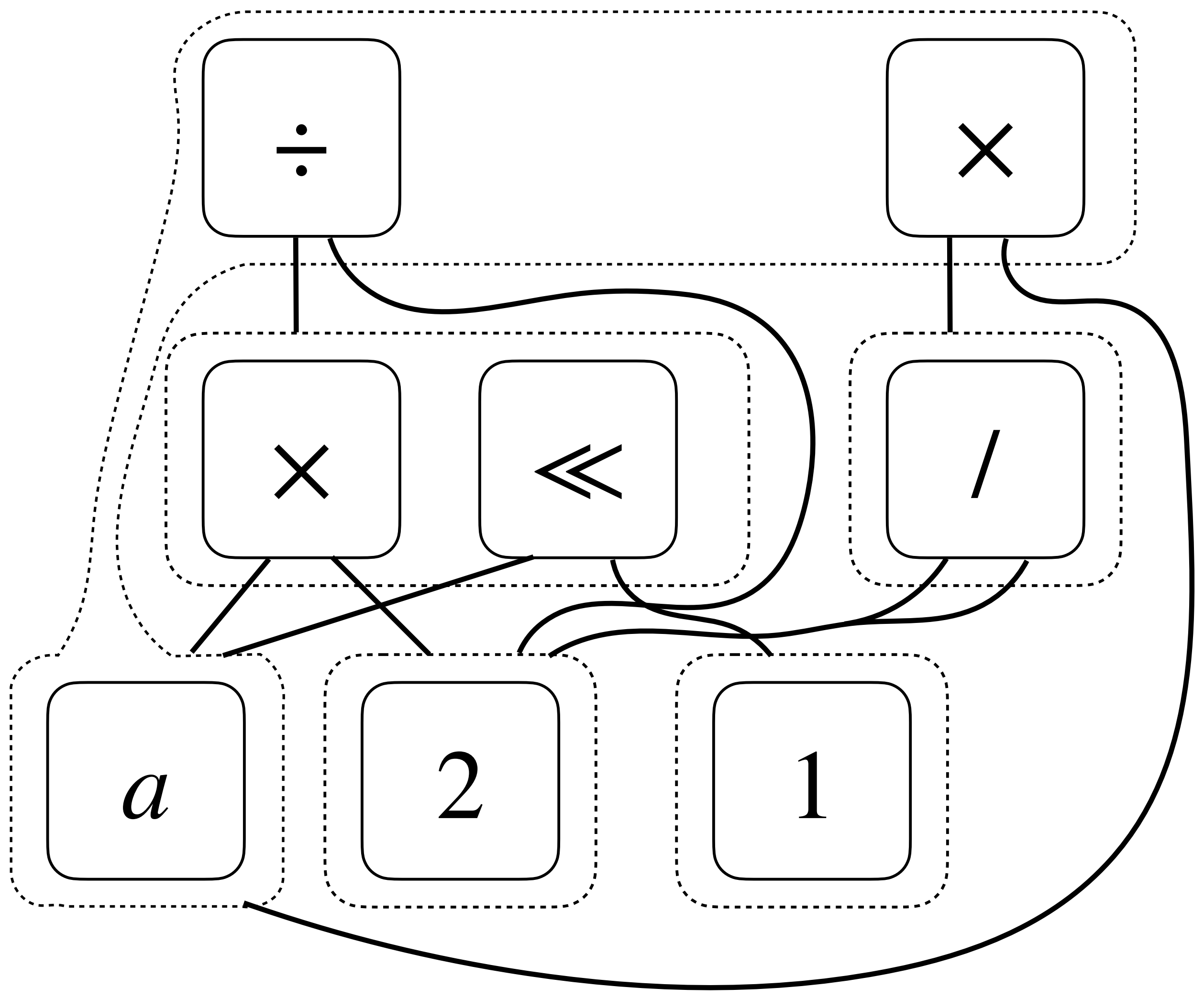




x * 2 => x << 1



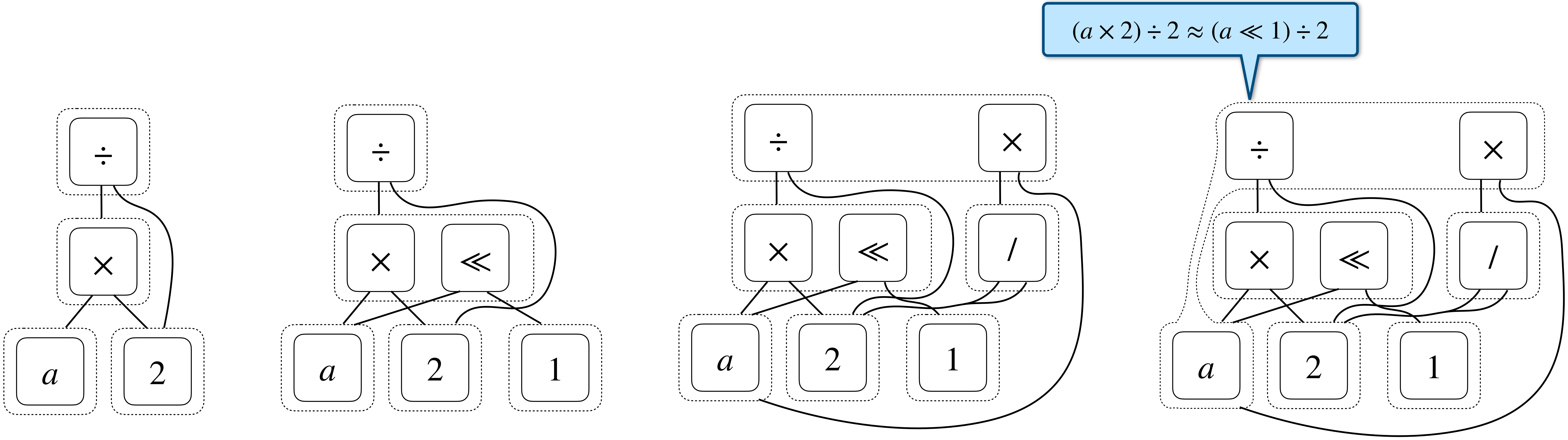
(x * y) / z => x * (y / z)



x / x => 1

x * 1 => x

E-graphs and Equality Saturation



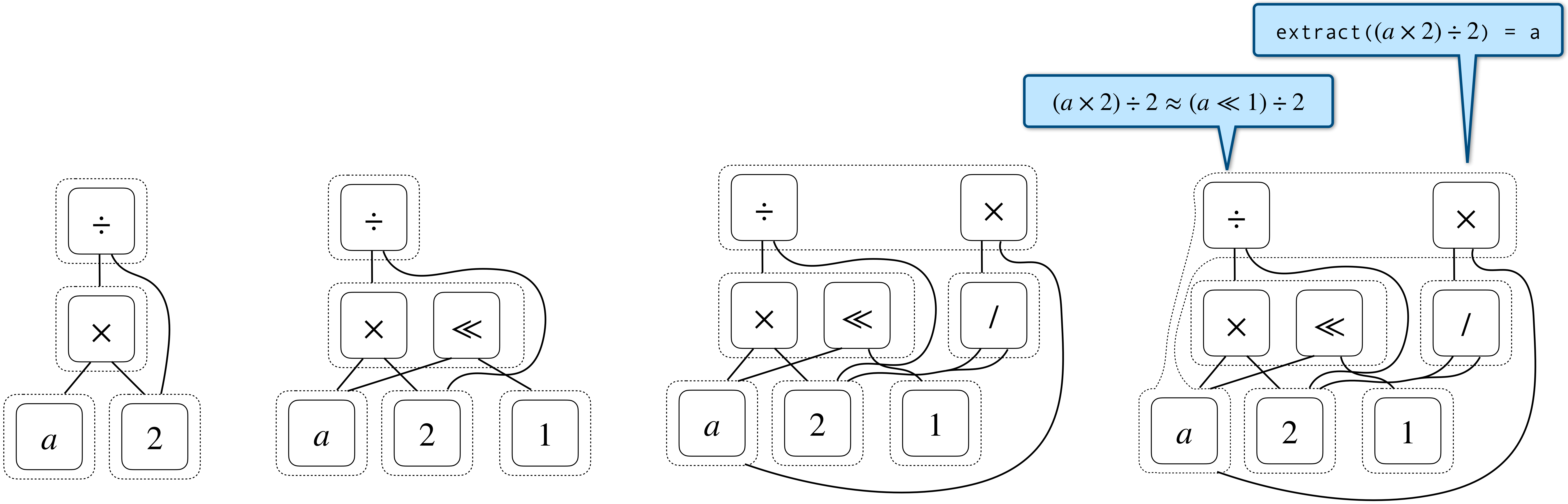
x * 2 => x << 1

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

x / x => 1

x * 1 => x

E-graphs and Equality Saturation



x * 2 => x << 1

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

x / x => 1
x * 1 => x

E-graphs as Tree Automata

E-graphs as Tree Automata

- Deterministic tree automaton

$$\mathscr{A} = (Q, \Sigma, \Delta).$$

E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton

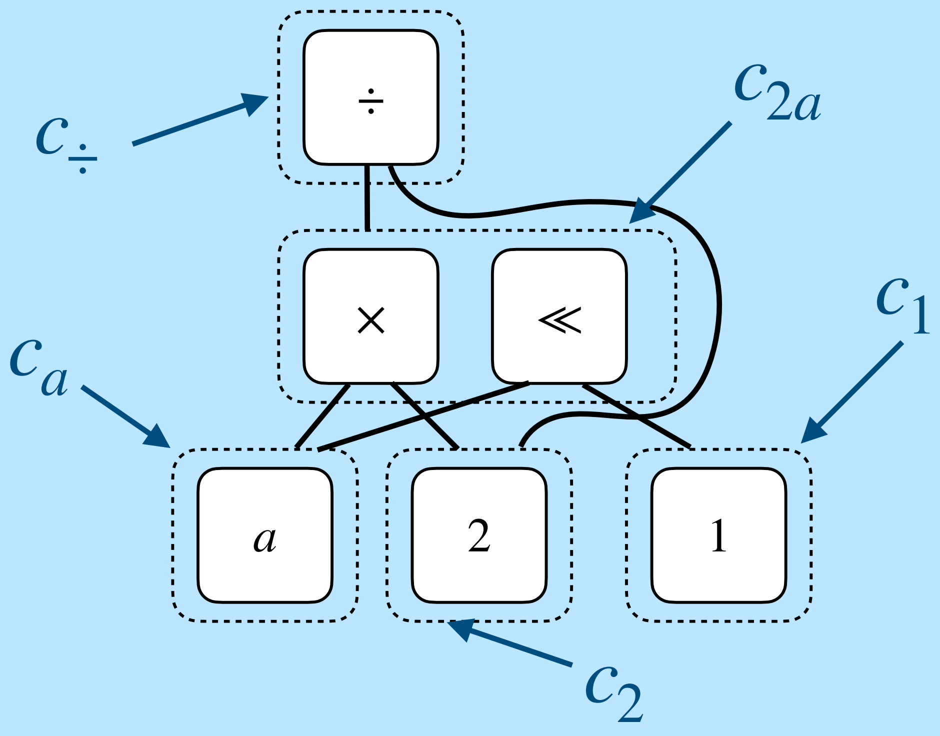
$$\mathscr{A} = (Q, \Sigma, \Delta).$$

E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton

$$\mathscr{A} = (Q, \Sigma, \Delta).$$



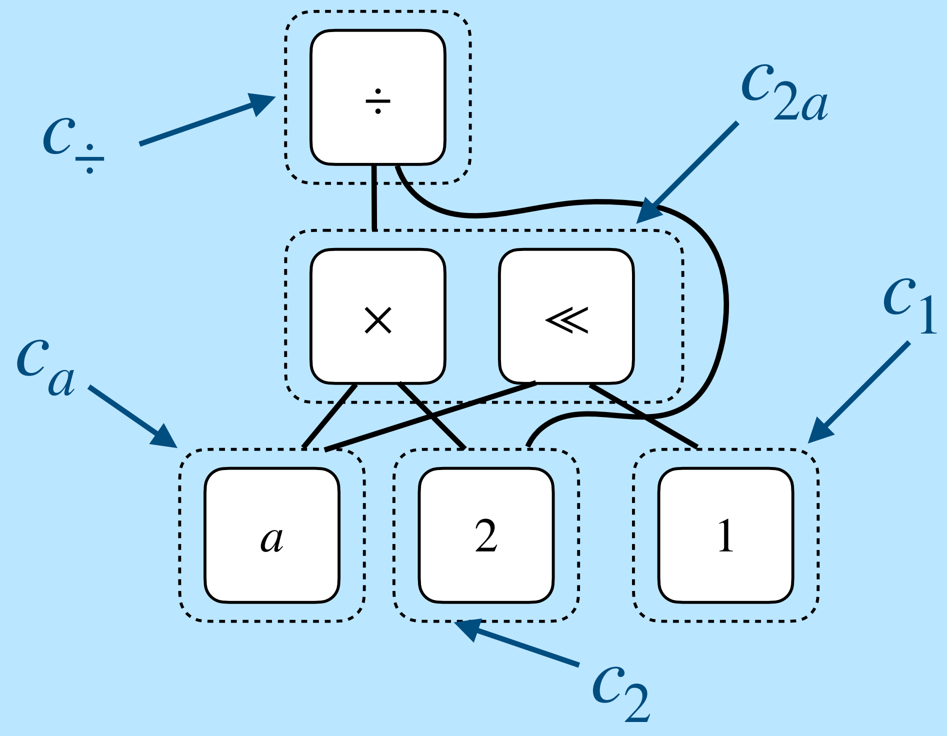
E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton

$$\mathscr{A} = (Q, \Sigma, \Delta).$$

$Q = \{c_1, c_2, c_a, c_{2a}, c_{\div}\},$



E-graphs as Tree Automata

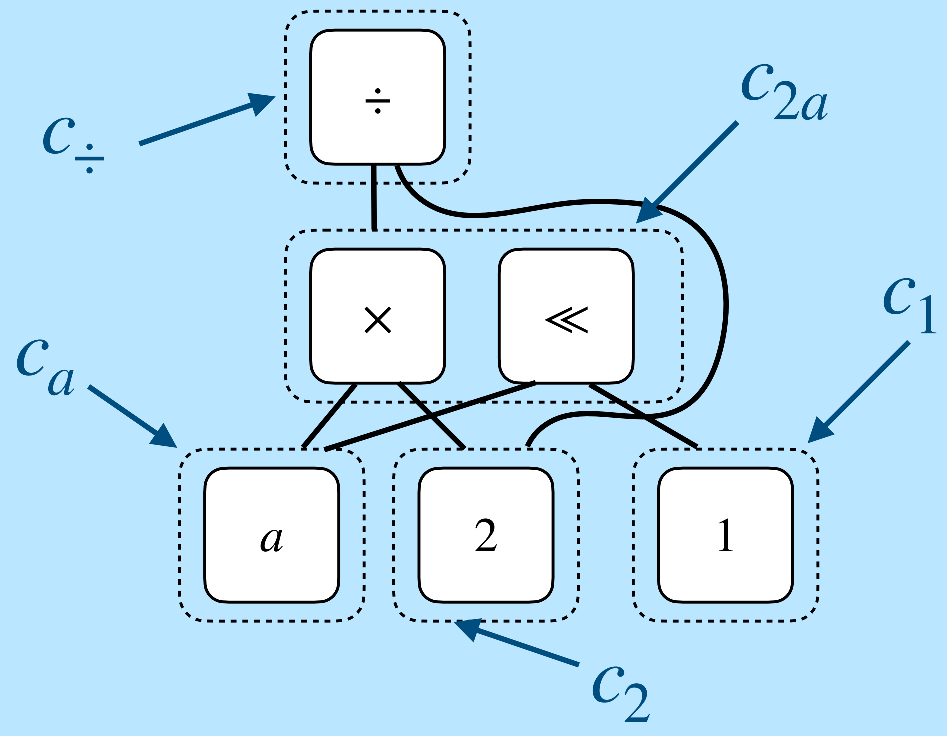
We allow a tree automaton to be infinite.

- Deterministic tree automaton

$$\mathscr{A} = (Q, \Sigma, \Delta).$$

$$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$$

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$

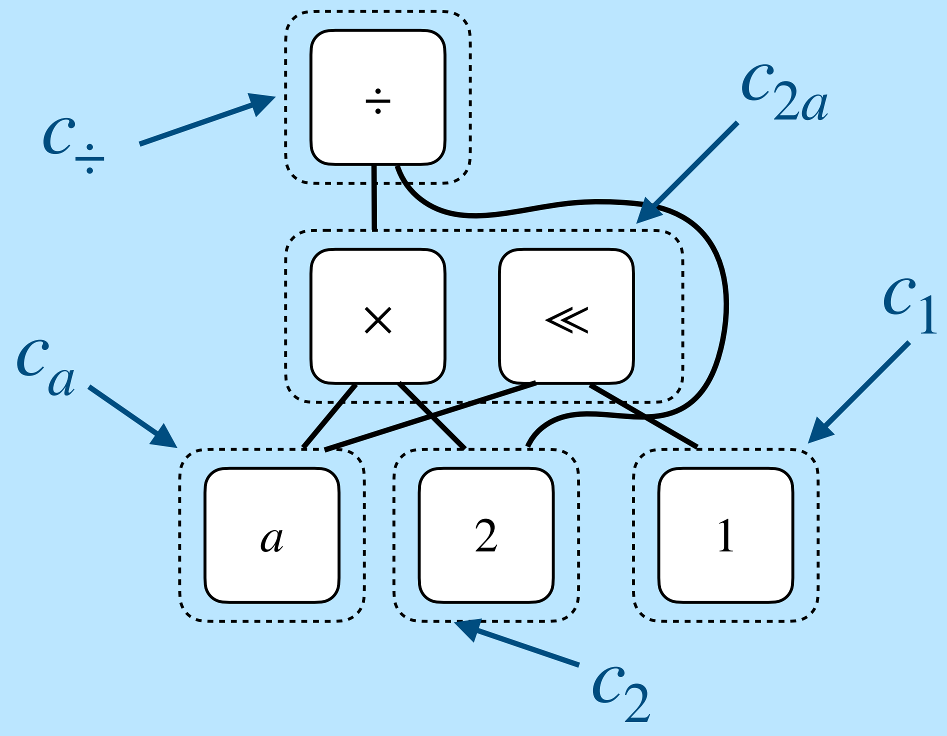


E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$
- A term t is _represented_ at $q \in Q$ if $t \to^*_{\mathscr{A}} q$.

$$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$$
$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



E-graphs as Tree Automata

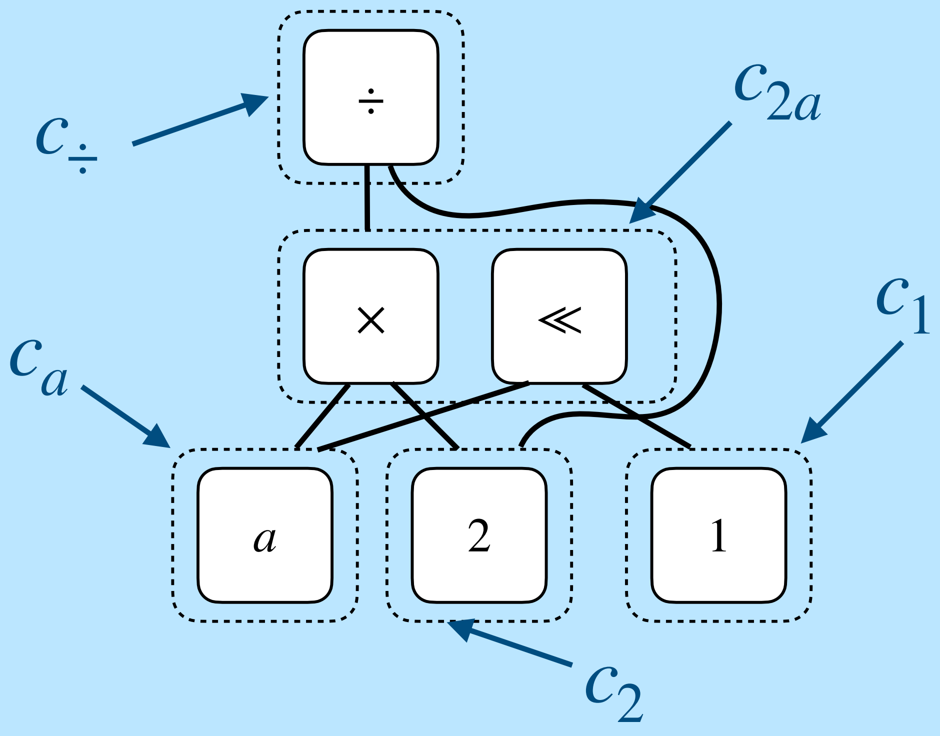
We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$

- A term t is <u>*represented*</u> at $q \in Q$ if $t \to^*_{\mathscr{A}} q$.

$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



$$(a \times 2) \div 2 \ \to^* \ (c_a \times c_2) \div c_2 \ \to \ c_{2a} \div c_2 \ \to \ c_\div$$

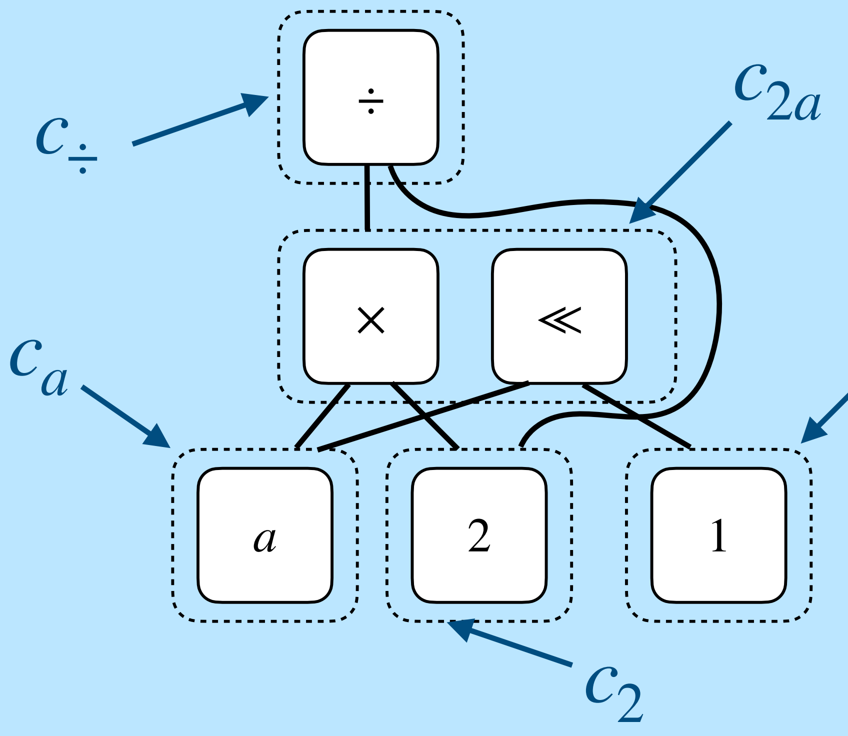
E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$
- A term t is <u>*represented*</u> at $q \in Q$ if $t \to^*_{\mathscr{A}} q$.
- An <u>*E-graph*</u> G is a reachable deterministic tree automaton.
  - E-classes = states, E-nodes = transitions

$$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$$

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



$$(a \times 2) \div 2 \ \to^* \ (c_a \times c_2) \div c_2 \ \to \ c_{2a} \div c_2 \ \to \ c_\div$$

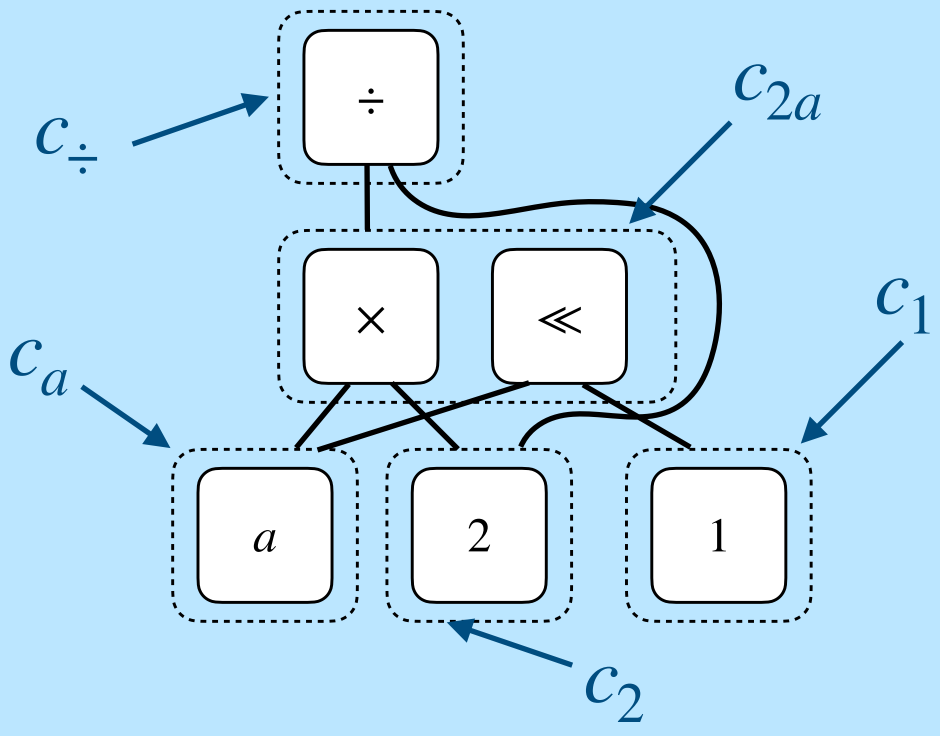
E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$
- A term t is *represented* at $q \in Q$ if $t \to^*_{\mathscr{A}} q$.
- An *E-graph* G is a reachable deterministic tree automaton.
  - E-classes = states, E-nodes = transitions
- [Kozen '93] The *semantics* of an E-graph G is $\approx_G$:
$$t_1 \approx_G t_2 \Leftrightarrow \exists q \,.\, t_1 \to^*_G q \leftarrow^*_G t_2$$

$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\}$,

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



$$(a \times 2) \div 2 \quad \to^* \quad (c_a \times c_2) \div c_2 \quad \to \quad c_{2a} \div c_2 \quad \to \quad c_\div$$

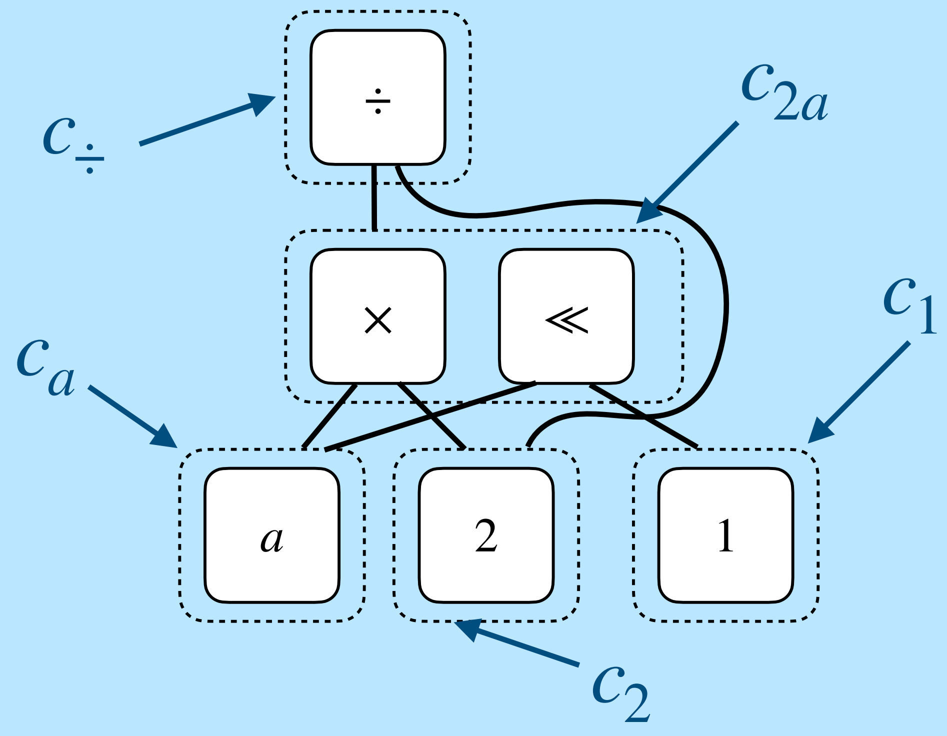
E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$
- A term t is *represented* at $q \in Q$ if $t \rightarrow^*_{\mathscr{A}} q$.
- An *E-graph* G is a reachable deterministic tree automaton.
  - E-classes = states, E-nodes = transitions
- [Kozen '93] The *semantics* of an E-graph G is $\approx_G$:
$$t_1 \approx_G t_2 \Leftrightarrow \exists q \,.\, t_1 \rightarrow^*_G q \leftarrow^*_G t_2$$

$$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$$

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



$$(a \times 2) \div 2 \ \rightarrow^* \ (c_a \times c_2) \div c_2 \ \rightarrow \ c_{2a} \div c_2 \ \rightarrow \ c_\div$$

$\approx_G$ is defined as $a \approx_G a$, $2 \approx_G 2$, $\ldots$ and two non-trivial identities

$$a \times 2 \approx_G a \ll 1$$
$$(a \times 2) \div 2 \approx_G (a \ll 1) \div 2$$

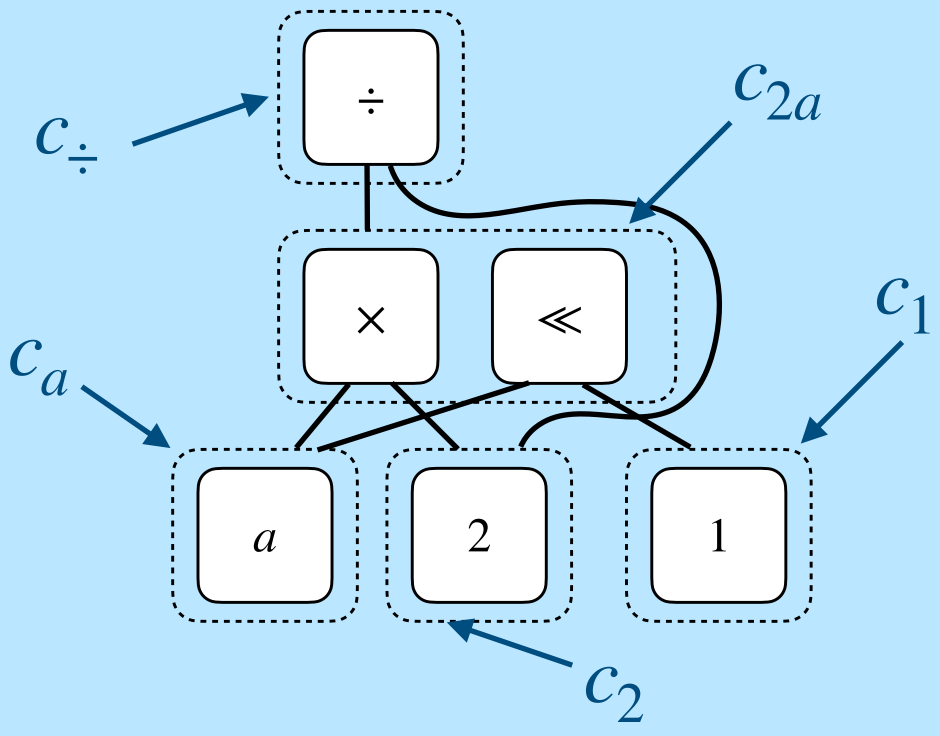
E-graphs as Tree Automata

We allow a tree automaton to be infinite.

- Deterministic tree automaton
$$\mathscr{A} = (Q, \Sigma, \Delta).$$
- A term t is *represented* at $q \in Q$ if $t \to^*_{\mathscr{A}} q$.
- An *E-graph* G is a reachable deterministic tree automaton.
  - E-classes = states, E-nodes = transitions
- [Kozen '93] The *semantics* of an E-graph G is $\approx_G$:
$$t_1 \approx_G t_2 \Leftrightarrow \exists q \,.\, t_1 \to^*_G q \leftarrow^*_G t_2$$
- $\approx_G$ is a partial congruence relation, or a PCR (= symmetric, transitive, and congruent).

$$Q = \{c_1, c_2, c_a, c_{2a}, c_\div\},$$

$$\Delta = \begin{cases} a() \to c_a, \\ 1() \to c_1,\ 2() \to c_2, \\ \times(c_a, c_2) \to c_{2a}, \\ \ll(c_a, c_1) \to c_{2a} \\ \div(c_{2a}, c_2) \to c_\div \end{cases}$$



$$(a \times 2) \div 2 \ \to^* \ (c_a \times c_2) \div c_2 \ \to \ c_{2a} \div c_2 \ \to \ c_\div$$

$\approx_G$ is defined as $a \approx_G a,\ 2 \approx_G 2,\ \ldots$ and two non-trivial identities

$$a \times 2 \approx_G a \ll 1$$
$$(a \times 2) \div 2 \approx_G (a \ll 1) \div 2$$

Congruence closure over E-graph

Congruence closure over E-graph

- Tree automata <u>*homomorphism*</u> $h : \mathscr{A} \to \mathscr{B}$.

Congruence closure over E-graph

- Tree automata <u>*homomorphism*</u> $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.

Congruence closure over E-graph

- Tree automata <u>*homomorphism*</u> $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:

Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.

Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
    - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
    - $\sqsubseteq$ forms a partial order.

Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.

Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.
- *Congruence closure*: For arbitrary $\mathscr{A}$, there exists a unique minimal E-graph G such that $\mathscr{A} \sqsubseteq G$.

Congruence closure over E-graph

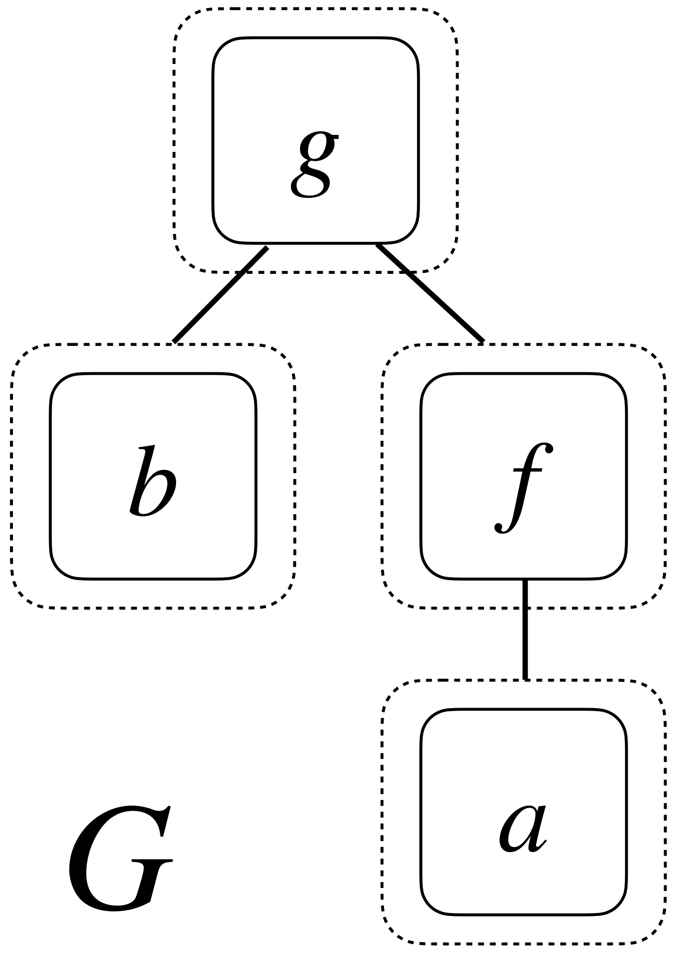
- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.
- *Congruence closure*: For arbitrary $\mathscr{A}$, there exists a unique minimal E-graph G such that $\mathscr{A} \sqsubseteq G$.

CC($\mathscr{A}$) can be computed in $O(n \log n)$ for finite $\mathscr{A}$

Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.
- *Congruence closure*: For arbitrary $\mathscr{A}$, there exists a unique minimal E-graph G such that $\mathscr{A} \sqsubseteq G$.

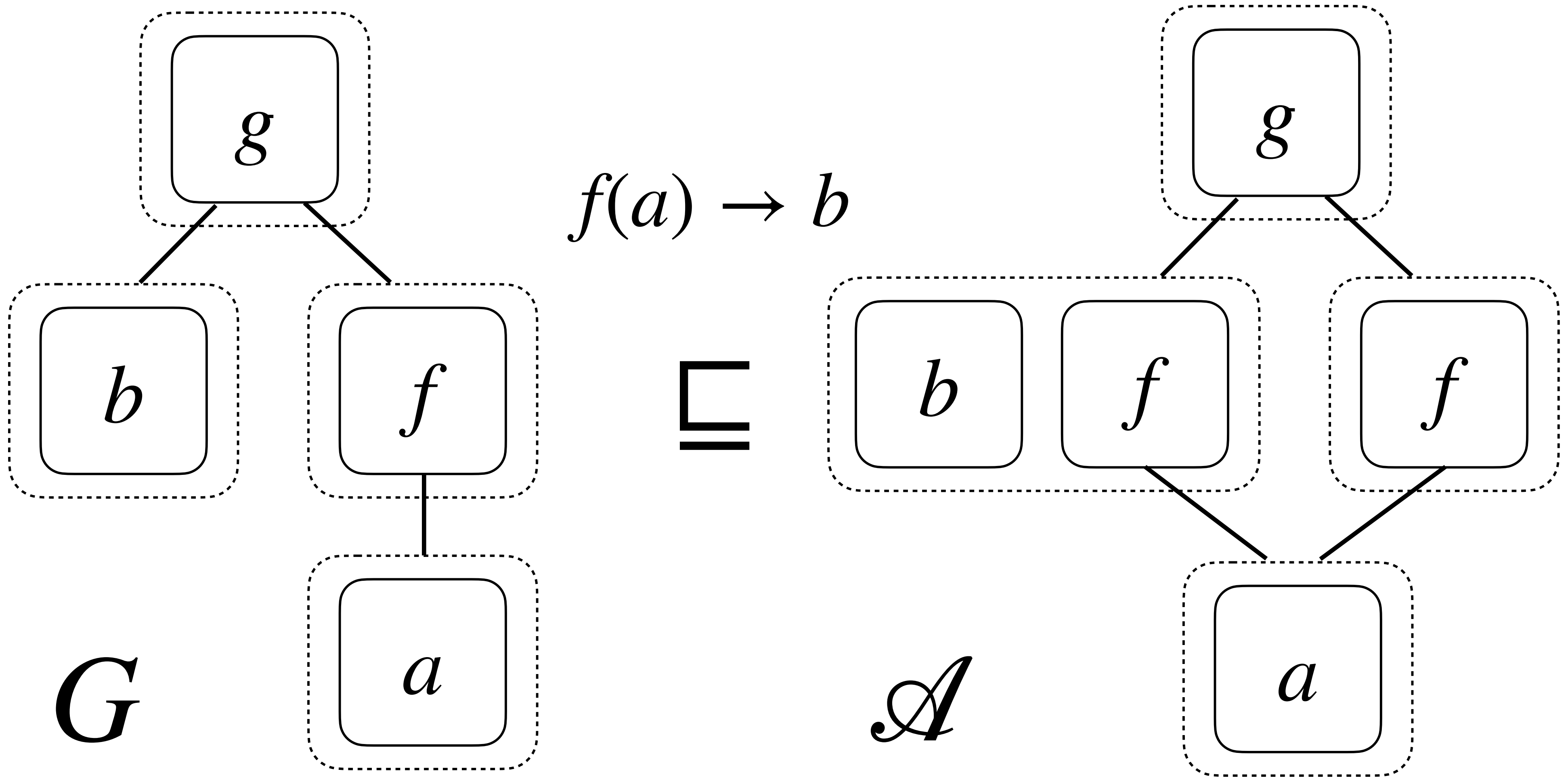
CC($\mathscr{A}$) can be computed in $O(n \log n)$ for finite $\mathscr{A}$



Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.
- *Congruence closure*: For arbitrary $\mathscr{A}$, there exists a unique minimal E-graph G such that $\mathscr{A} \sqsubseteq G$.

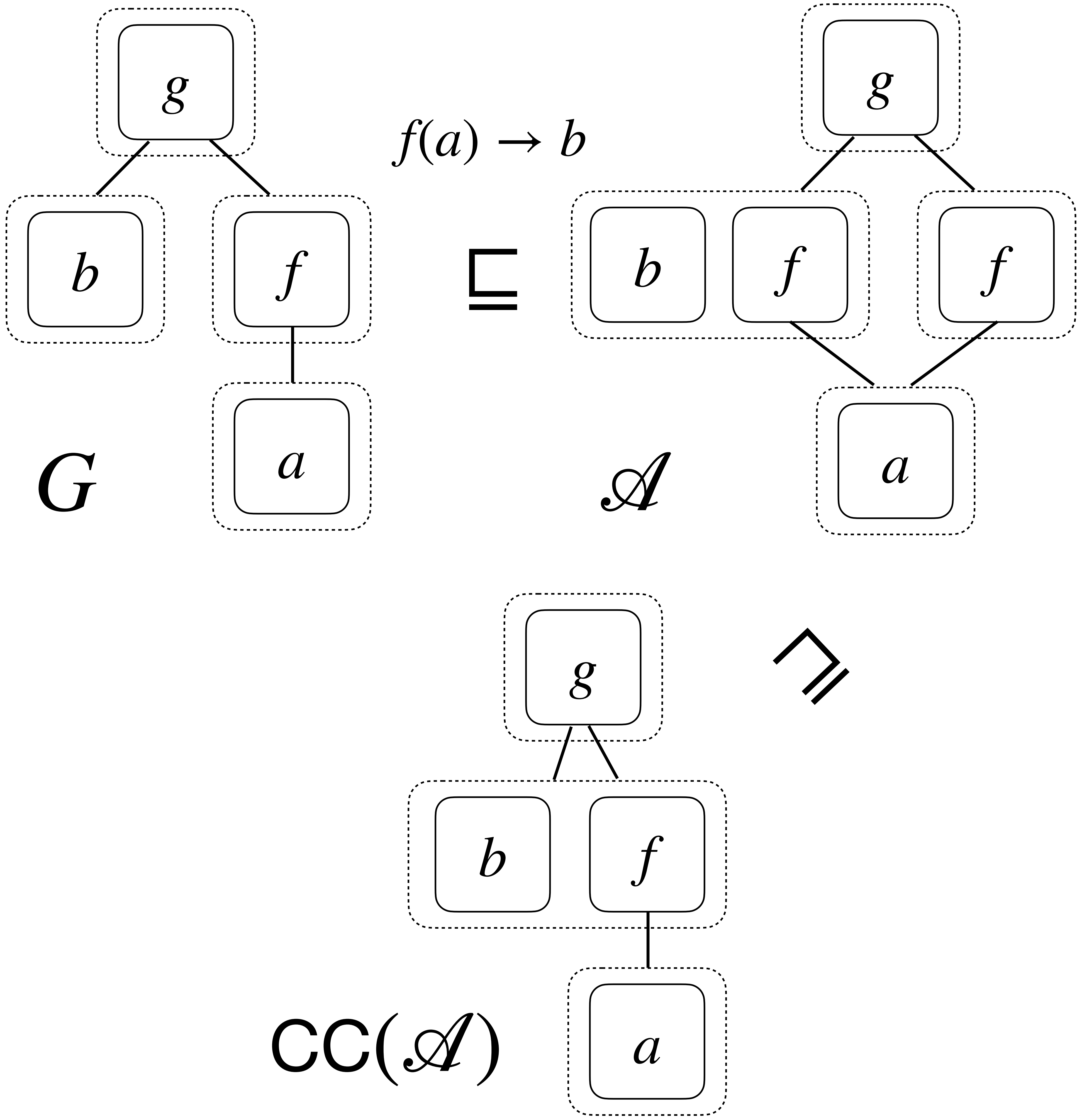
CC($\mathscr{A}$) can be computed in $O(n \log n)$ for finite $\mathscr{A}$



Congruence closure over E-graph

- Tree automata *homomorphism* $h : \mathscr{A} \to \mathscr{B}$.
- Define $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\exists h : \mathscr{A} \to \mathscr{B}$.
- When restricted to E-graphs:
  - $h_1, h2 : G_1 \to G_2$ implies $h_1 = h_2$.
  - $\sqsubseteq$ forms a partial order.
  - $G_1 \sqsubseteq G_2$ implies $\approx_{G_1} \subseteq \approx_{G_2}$.
- *Congruence closure*: For arbitrary $\mathscr{A}$, there exists a unique minimal E-graph G such that $\mathscr{A} \sqsubseteq G$.

CC($\mathscr{A}$) can be computed in $O(n \log n)$ for finite $\mathscr{A}$



Equality Saturation

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\text{ICO}_R = \text{CC} \circ T_R$$

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\mathrm{ICO}_R = \mathrm{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\text{ICO}_R = \text{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\text{EqSat}(R, G) = \bigsqcup_{i \geq 0} \text{ICO}_R^{(i)}(G)$$

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\mathsf{ICO}_R = \mathsf{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\mathsf{EqSat}(R, G) = \bigsqcup_{i \geq 0} \mathsf{ICO}_R^{(i)}(G)$$

- The *model semantics* of EqSat: EqSat(R, G) is a universal model of R and G
  (see the paper for defn. of universal models).

Equality Saturation

- The *immediate consequence operator*:
(rule application T_R defined in the paper)

$$\mathsf{ICO}_R = \mathsf{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\mathsf{EqSat}(R, G) = \bigsqcup_{i \geq 0} \mathsf{ICO}_R^{(i)}(G)$$

- The *model semantics* of EqSat: EqSat(R, G) is a universal model of R and G
(see the paper for defn. of universal models).

- The two semantics coincide.

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\mathsf{ICO}_R = \mathsf{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\mathsf{EqSat}(R, G) = \bigsqcup_{i \geq 0} \mathsf{ICO}_R^{(i)}(G)$$

- The *model semantics* of EqSat: $\mathsf{EqSat}(R, G)$ is a universal model of R and G
  (see the paper for defn. of universal models).

- The two semantics coincide.

Properties of EqSat

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\mathsf{ICO}_R = \mathsf{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\mathsf{EqSat}(R, G) = \bigsqcup_{i \geq 0} \mathsf{ICO}_R^{(i)}(G)$$

- The *model semantics* of EqSat: $\mathsf{EqSat}(R, G)$ is a universal model of R and G
  (see the paper for defn. of universal models).
- The two semantics coincide.

Properties of EqSat

- (Inflationary) $G \sqsubseteq \mathsf{EqSat}(R, G)$.

Equality Saturation

- The *immediate consequence operator*:
  (rule application T_R defined in the paper)

$$\text{ICO}_R = \text{CC} \circ T_R$$

- The *fixpoint semantics* of EqSat

$$\text{EqSat}(R, G) = \bigsqcup_{i \geq 0} \text{ICO}_R^{(i)}(G)$$

- The *model semantics* of EqSat: $\text{EqSat}(R, G)$ is a universal model of R and G
  (see the paper for defn. of universal models).
- The two semantics coincide.

Properties of EqSat

- (Inflationary) $G \sqsubseteq \text{EqSat}(R, G)$.
- (Finite convergence)
  If $\text{EqSat}(R, G)$ is finite, Equality Saturation converges in a finite number of steps.

EqSat and Term Rewriting

EqSat and Term Rewriting

- Let R be a TRS, $s \in T_\Sigma$, $G = \text{EqSat}(R, s)$
    - If $s \to_R^* t$, then $s \approx_G t$.
    - If $s \approx_G t$, then $s \leftrightarrow_R^* t$.

EqSat and Term Rewriting

- Let R be a TRS, $s \in T_\Sigma$, $G =$ EqSat(R, s)
  - If $s \to_R^* t$, then $s \approx_G t$.
  - If $s \approx_G t$, then $s \leftrightarrow_R^* t$.
- If R is bidirectional, $\to_R^*$, $\approx_G$, $\leftrightarrow_R^*$ coincide.
  - In this case, EqSat semi-decides the word problem.

Equality Saturation and the Chase

Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.

Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.
- Skolem Chase $\leq$ EqSat.

Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.
- Skolem Chase $\leq$ EqSat.
- EqSat $\leq$ Standard Chase.
  - EqSat terminates $\Leftrightarrow$
  - $\exists$ a finite run of the encoding chase $\Leftrightarrow$
  - All *EGD-fair* runs of the encoding chase are finite.

Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.
- Skolem Chase $\leq$ EqSat.
- EqSat $\leq$ Standard Chase.
  - EqSat terminates $\Leftrightarrow$
  - $\exists$ a finite run of the encoding chase $\Leftrightarrow$
  - All *EGD-fair* runs of the encoding chase are finite.

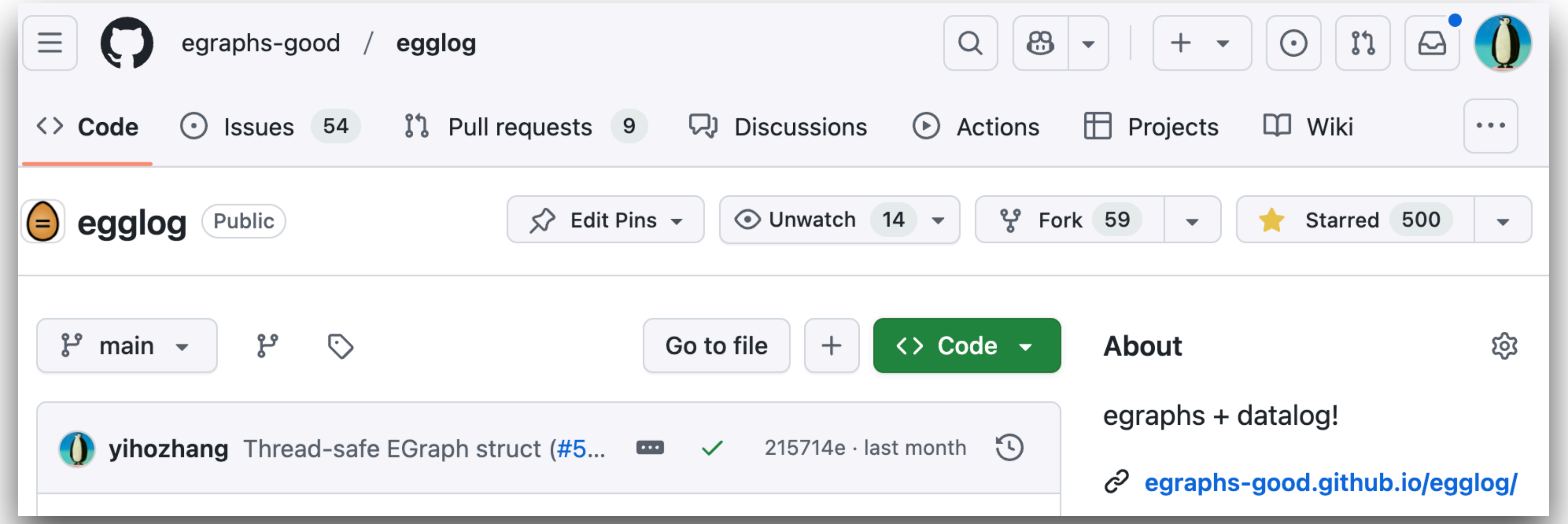
The chase sequence needs to apply EGD often enough

Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.
- Skolem Chase $\leq$ EqSat.
- EqSat $\leq$ Standard Chase.
  - EqSat terminates $\Leftrightarrow$
  - $\exists$ a finite run of the encoding chase $\Leftrightarrow$
  - All *EGD-fair* runs of the encoding chase are finite.

The chase sequence needs to apply EGD often enough

We build an entire system out of this idea!
(Zhang et. al. 2023)

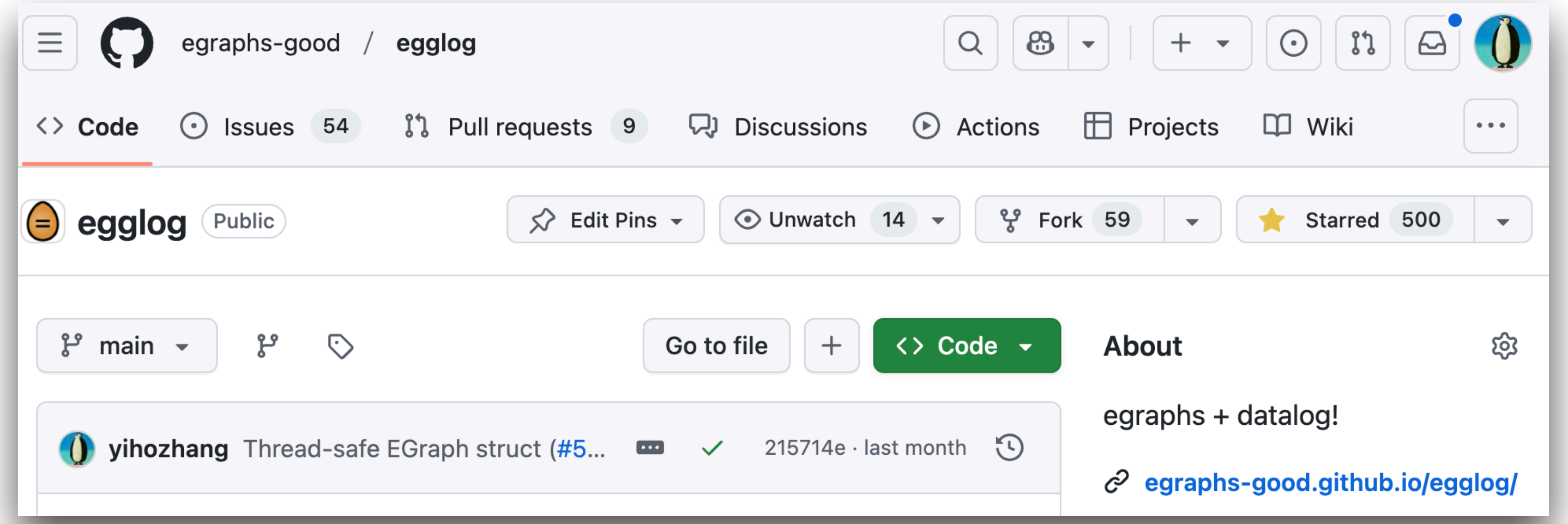


Equality Saturation and the Chase

- The Chase
  - A generalization of Datalog.
  - Used in data integration and query optimization.
  - Two variants: Skolem Chase, Standard Chase.
- Skolem Chase $\leq$ EqSat.
- EqSat $\leq$ Standard Chase.
  - EqSat terminates $\Leftrightarrow$
  - $\exists$ a finite run of the encoding chase $\Leftrightarrow$
  - All *EGD-fair* runs of the encoding chase are finite.

The chase sequence needs to apply EGD often enough

We build an entire system out of this idea!
(Zhang et. al. 2023)



- Great speedup: Query optimization, semi-naive evaluation, worst-case optimal join, ...
- New expressive power: Multi-pattern rules, monotonic aggregation, ...

Termination Theorem

- (Single-instance) Does EqSat terminate with for a single term t?
  - Recursive enumerable (R.E.)—complete.
- (All-E-graph) Does EqSat terminate for all E-graph G?
  - R.E.-hard.
- (All-term) Does EqSat terminate for all $t \in T_\Sigma$?
  - Π_2-complete.

Termination Theorem

- (Single-instance) Does EqSat terminate with for a single term t?
  - Recursive enumerable (R.E.)—complete.
- (All-E-graph) Does EqSat terminate for all E-graph G?
  - R.E.-hard.
- (All-term) Does EqSat terminate for all $t \in T_\Sigma$?
  - Π_2-complete.

Strictly harder than R.E.

Termination Theorem

- (Single-instance) Does EqSat terminate with for a single term t?
  - Recursive enumerable (R.E.)—complete.
- (All-E-graph) Does EqSat terminate for all E-graph G?
  - R.E.-hard.
- (All-term) Does EqSat terminate for all $t \in T_\Sigma$?
  - Π_2-complete.

Strictly harder than R.E.

More details in the paper!

Summary

- The fixpoint and model semantics of Equality Saturation
- Connections to Term Rewriting and the Chase
- Undecidability of Termination
- Open problems
  - Extraction
  - Provenance

Also check out the egglog system!

GitHub: @egraphs-good/egglog