EqSat is not better than term rewriting*

Yihong Zhang, Oliver Flatt

EGRAPHS 2023

* but tree automata completion is!

Agenda

We study the termination problem of EqSat ($\text{TERM}_{\text{EqSat}}$):

- $\text{TERM}_{\text{EqSat}}$ and $\text{TERM}_{\text{TmRw}}$ do not imply each other.
    - This refutes the misconception that EqSat is always a better replacement of term rewriting.
- We show **tree automata completion (TAC)**, a technique similar to EqSat with the property that $\text{TERM}_{\text{TmRw}}$ implies TERM_{TAC}.
    - We show an application of Tree Automata completion to rewrite rule synthesis.
- We introduce two tricks for ensuring EqSat termination in practice and their corresponding guarantees.

Agenda

We study the termination problem of EqSat ($\mathbf{TERM}_{\mathbf{EqSat}}$):

Termination of Equality Saturation

Tree Automata Completion

Termination Tricks with Guarantees

The termination problem of EqSat

Definition

- Instance: a rewriting system R, a term t.
- Question: does running EqSat with R on initial term t terminate in a finite number of iterations?

If EqSat terminates, the equalities are saturated.

- Optimality in program optimization.
- Decidability in theory solving.

The termination problem of EqSat

Folklore: “EqSat is term rewriting but more powerful”.

- E-graphs can represent infinitely many terms; so
- EqSat should terminate for more term rewriting systems.

This is not true.

Associativity does not terminate

Let R be

$$(x \cdot y) \cdot z \to x \cdot (y \cdot z)$$
$$0 \cdot a \to 0$$

R is terminating in TmRw.

Associativity does not terminate

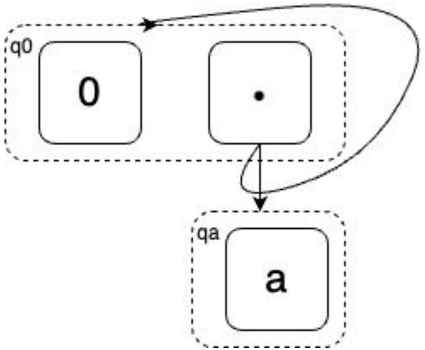
Let R be

$$(x \cdot y) \cdot z \to x \cdot (y \cdot z)$$
$$0 \cdot a \to 0$$

R is terminating in TmRw.

However, with initial term $0 \cdot a$, EqSat will apply rule $0 \cdot a \to 0$ and create a cyclic E-graph that represents

$$0,\ 0 \cdot a,\ (0 \cdot a) \cdot a,\ ((0 \cdot a) \cdot a) \cdot a,\ \ldots$$



Associativity does not terminate

Let R be

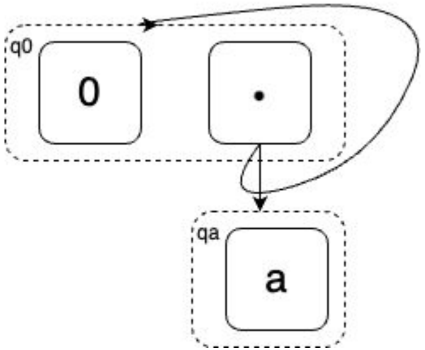
$$(x \cdot y) \cdot z \to x \cdot (y \cdot z)$$
$$0 \cdot a \to 0$$

R is terminating in TmRw.

However, with initial term 0 · a, EqSat will apply rule 0 · a → 0 and create a cyclic E-graph that represents

0, 0 · a, (0 · a) · a, ((0 · a) · a) · a, ...

Associativity will reassociate these terms and produce a, a · a, a · a · a, ..., which are pairwise inequivalent.



Associativity does not terminate

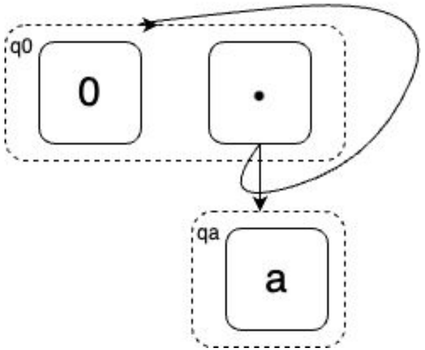
Let R be

$$(x \cdot y) \cdot z \to x \cdot (y \cdot z)$$
$$0 \cdot a \to 0$$

R is terminating in TmRw.

However, with initial term 0 · a, EqSat will apply rule 0 · a → 0 and create a cyclic E-graph that represents

0, 0 · a, (0 · a) · a, ((0 · a) · a) · a, ...



Associativity will reassociate these terms and produce a, a · a, a · a · a, ..., which are pairwise inequivalent.

This requires an infinite number of E-classes, which is impossible.

Convergent TRSes do not terminate

There are also convergent term rewriting systems that do not terminate in EqSat.

Convergence (termination + confluence) is one of the strongest properties in term rewriting.

Why?

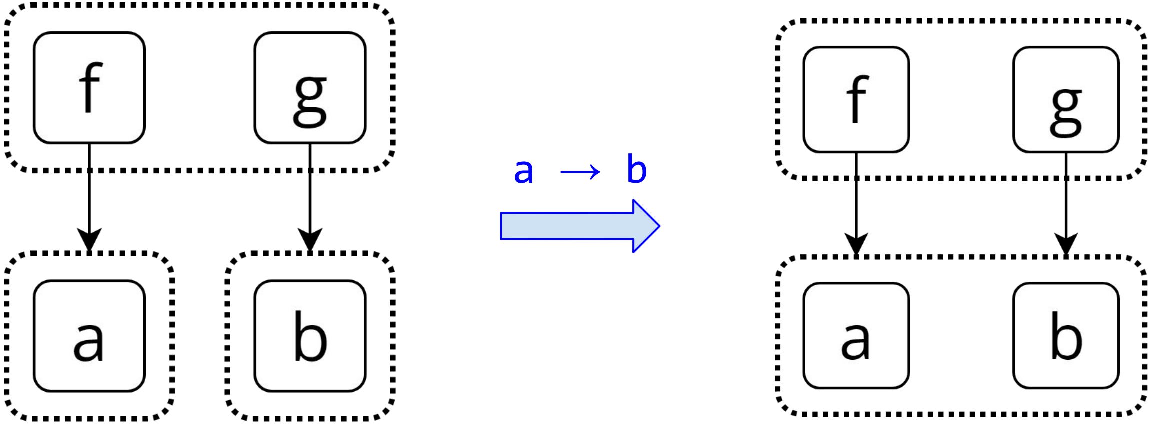
If a TRS is terminating, then the set of derivable terms should always be finite.

- It is natural to think EqSat should terminate as well.

The issue: **EqSat is not exactly term rewriting!**

- EqSat tracks equivalences, not rewritability!
- EqSat can derive terms not derivable in term rewriting.

Why?



Represents
f(a)   g(b)

Represents
f(a)   g(b)
f(b)   g(a)

Term rewriting will never derive g(a)!

Agenda

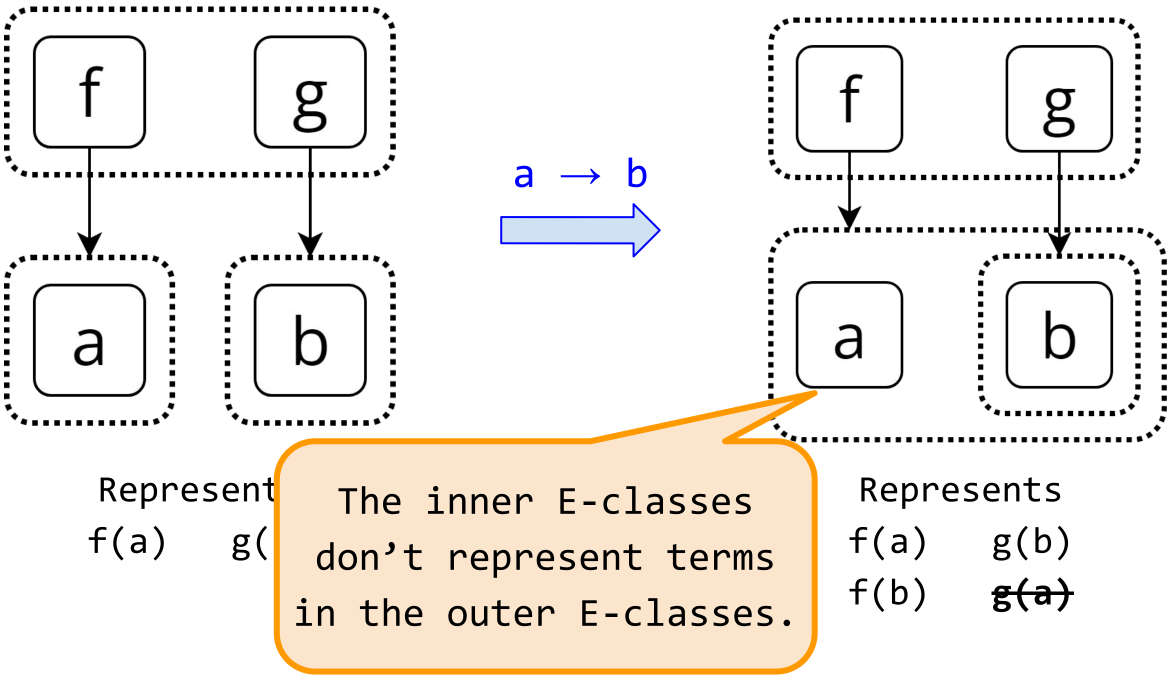
We study the termination problem of EqSat ($\mathbf{TERM}_{\mathbf{EqSat}}$):

Termination of Equality Saturation

⇨ **Tree Automata Completion** ⇦

Termination Tricks with Guarantees

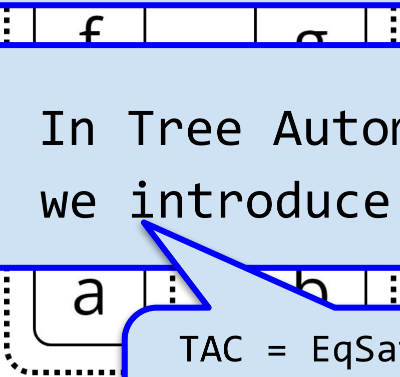
Tree automata completion



Tree

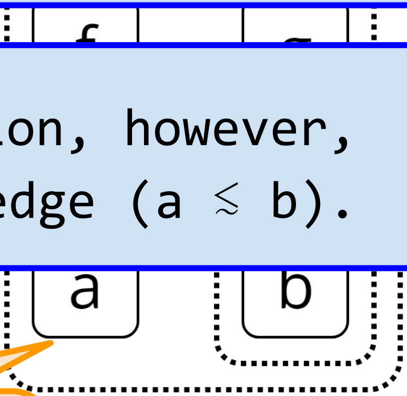
In EqSat, when we merge two terms, we introduce an equivalence edge between them (a ≈ b).

In Tree Automata Completion, however, we introduce an *ordered* edge (a ≲ b).



TAC = EqSat
- equivalence
+ preorder

...lasses
don't represent terms
in the outer E-classes.



Represents
f(a) g(b)
f(b) ~~g(a)~~

Tree

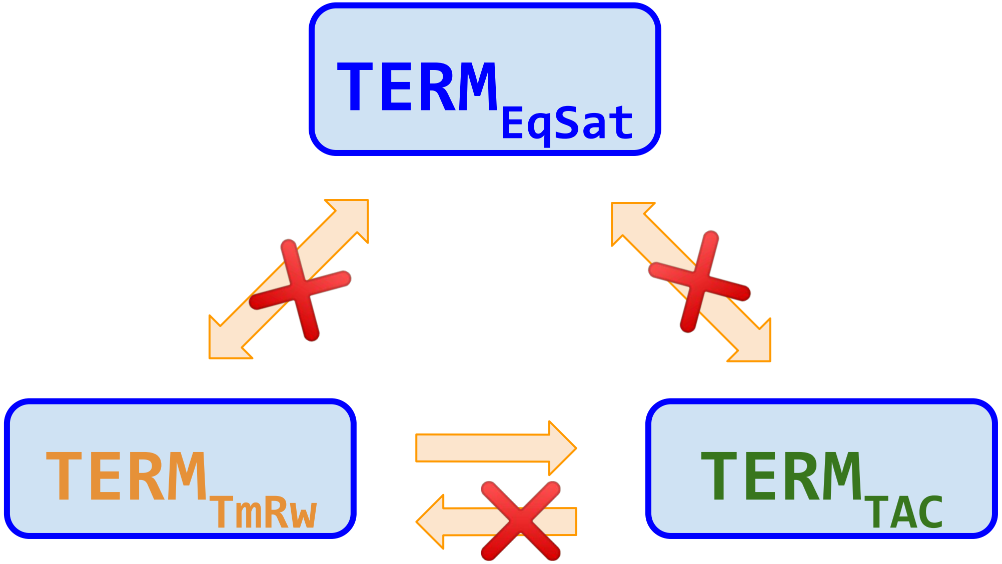
In EqSat, when we merge two terms, we introduce an equivalence edge between them (a ≈ b).

In Tree Automata Completion, however, we introduce an *ordered* edge (a ≲ b).

Guarantee: Tree Automata Completion will only derive terms derivable in term rewriting.

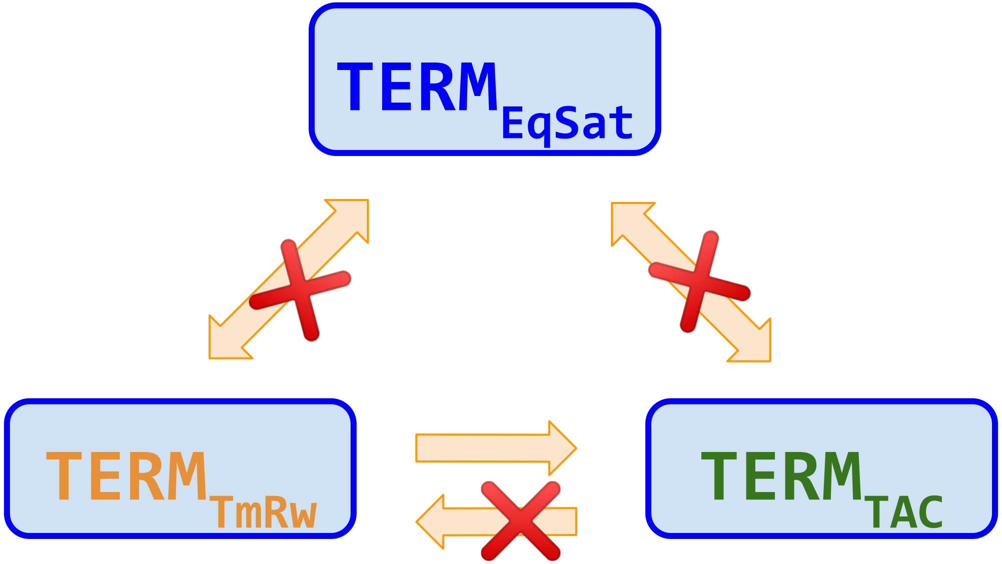
Corollary: $\text{TERM}_{\text{TmRw}}$ implies TERM_{TAC}.

Comparison



$TERM_{TmRw}$ and $TERM_{TAC}$ are known to be undecidable.

Comparison



$TERM_{TmRw}$ and $TERM_{TAC}$ are known to be undecidable.

We additionally showed $TERM_{EqSat}$ is undecidable.

Implementation of Tree Automata Completion

- We have not implemented it.
- Tree Automata Completion is computationally harder than EqSat.
    - Need to maintain and query a DAG than an equivalence relation.

Potential Applications of Tree Automata Completion

- Applications that require strong **termination guarantees.**
- Applications where **rewritability/refinement** is desired.
- Ruler: rewrite rule synthesis.

This talk

Tree automata completion for rewrite rule synthesis

Ruler's rule validation problem

Instance: a list of expression pairs (lhs_i, rhs_i)

Question: For each i, can EqSat use lhs_i to derive rhs_i?

Ruler currently inserts all lhs_i, runs EqSat once, and checks if each rhs_i exists and is equivalent to lhs_i.

- \+ Very fast thanks to batching.
- \- This can be unsound.

Tree automata completion for rewrite rule synthesis

x + 0 → x

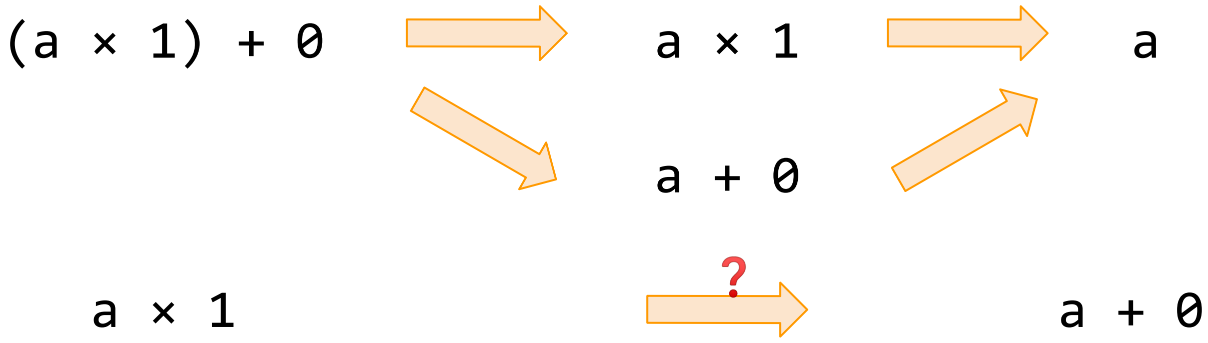
x × 1 → x

(a × 1) + 0 ⟹? a

a × 1 ⟹? a + 0

Tree automata completion for rewrite rule synthesis

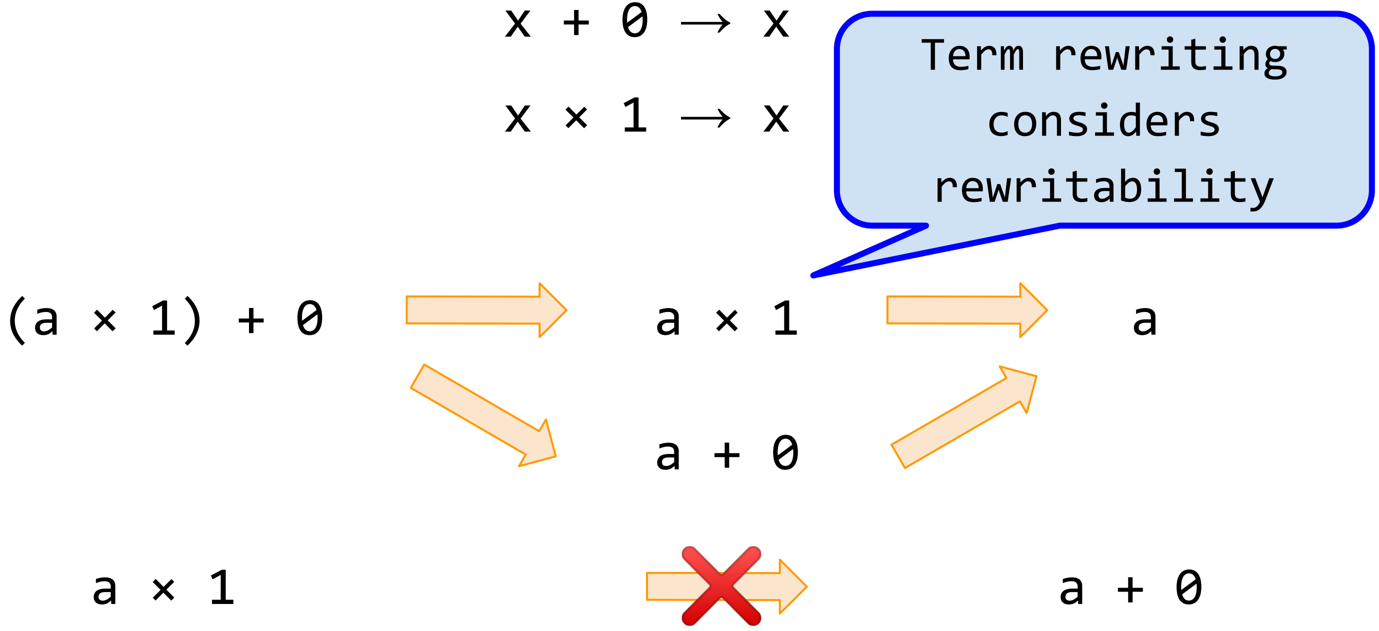
```
x + 0 → x
x × 1 → x
```



Tree automata completion for rewrite rule synthesis

$$x + 0 \rightarrow x$$

$$x \times 1 \rightarrow x$$



Tree automata completion for rewrite rule synthesis

x + 0 → x

x × 1 → x

EqSat considers term equivalences

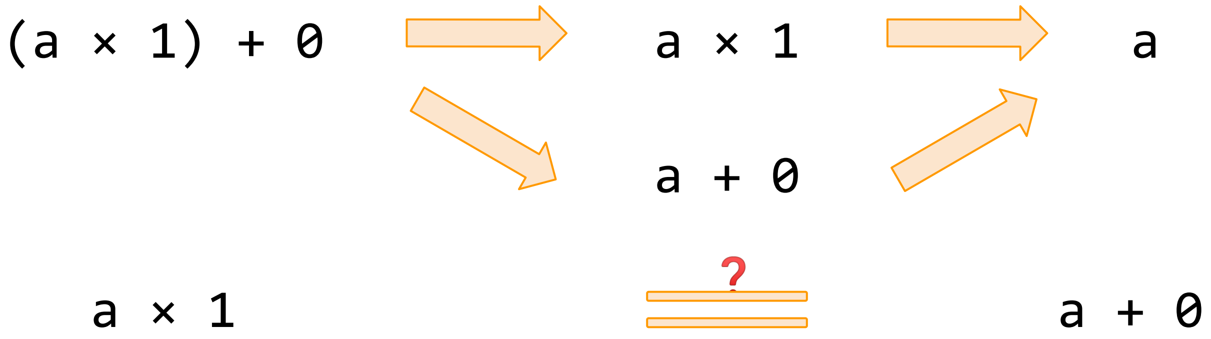
(a × 1) + 0 $\stackrel{?}{=}$ a

a × 1 $\stackrel{?}{=}$ a + 0

Tree automata completion for rewrite rule synthesis

$$x + 0 \rightarrow x$$

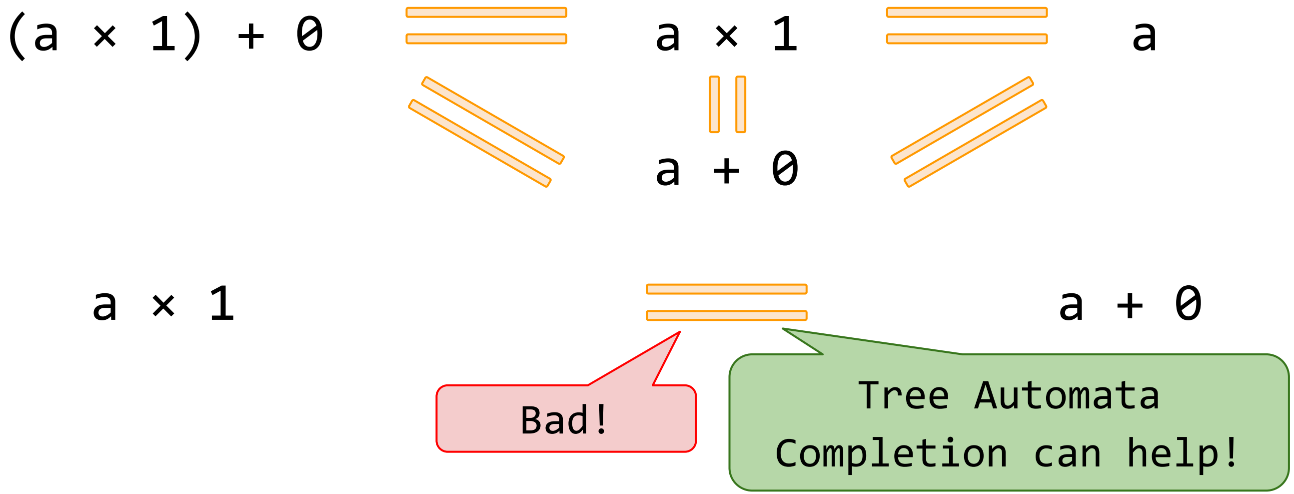
$$x \times 1 \rightarrow x$$



Tree automata completion for rewrite rule synthesis

$$x + 0 \rightarrow x$$

$$x \times 1 \rightarrow x$$



Agenda

We study the termination problem of EqSat ($\mathbf{TERM}_{\mathbf{EqSat}}$):

Termination of Equality Saturation

Tree Automata Completion

Termination Tricks with Guarantees

Practical approaches to termination

In practice, people ensure termination by running EqSat for a finite number of iterations.

Guarantee: If there exists a proof to u = v of the form

$$\exists\ w.\ u \longrightarrow^{\leq N} w \longleftarrow^{\leq N} v,$$

running EqSat for N iterations can prove u = v.

Observation: Different termination strategy gives us different guarantees.

Merge-only Equality Saturation

Given an E-graph G, merge-only EqSat applies a rule only when both the left-hand side and the right-hand side are already in the E-graph.

Merge-only Equality Saturation

Given an E-graph G, merge-only EqSat applies a rule only when both the left-hand side and the right-hand side are already in the E-graph.

Termination: Notice merge-only EqSat shrinks the number of E-classes in each iteration.

Merge-only Equality Saturation

Given an E-graph G, merge-only EqSat applies a rule only when both the left-hand side and the right-hand side are already in the E-graph.

Termination: Notice merge-only EqSat shrinks the number of E-classes in each iteration.

Guarantee: If u = v can be proved using only terms in G, this proof can be obtained with merge-only EqSat.

(Depth-)bound … ation

Min depth of all terms represented.

Still admits infinite terms!

Depth-bounded EqSat tracks ca … class … s:

$$\texttt{depth(c) = min}_{\texttt{c represents t}}\texttt{depth(t)}$$

(Depth-)bounded Equality Saturation

Depth-bounded EqSat tracks each E-class with a depth analysis:

$$\text{depth}(c) = \min_{c \text{ represents } t} \text{depth}(t)$$

During rule application, we apply a rule only when the right-hand side has depth ≤ N.

(Depth-)bounded Equality Saturation

Depth-bounded EqSat tracks each E-class with a depth analysis:

$$\text{depth}(c) = \min_{c \text{ represents } t} \text{depth}(t)$$

During rule application, we apply a rule only when the right-hand side has depth ≤ N.

Termination: The depth constraint bounds the # of possible E-classes, which bounds the # of possible E-graphs.

(Depth-)bounded Equality Saturation

Guarantee: if u = v can be proved using terms with depth ≤ N,
this proof can be obtained with depth-bounded EqSat.

+ Useful for program optimization.
- Hardly terminate for realistic N.

Can be generalized to *F*-bounded EqSat, where *F* is some constraints that bounds the number of possible E-classes (e.g., size).

Takeaways

- The termination problem of EqSat is not trivial.
- Tree Automata Completion = EqSat - ($\approx$) + ($\lesssim$).

Takeaways

- The termination problem of EqSat is not trivial.
- Tree Automata Completion = EqSat - ($\approx$) + ($\lesssim$).
- Two termination strategies.
  - Merge-only EqSat.
  - Depth-bounded EqSat.