The Termination Problem of Equality Saturation is Undecidable

Yihong Zhang

Aug 2, 2023

In this note, we study the decidability of the termination of equality saturation and related problems.

Background

Term rewriting

A term rewriting system (TRS) R consists of a set of rewrite rules. R defines a *rewrite relation* $\to_R$. We omit the subscript R when it's clear from the context. Let $\to^*$ be the transitive closure of binary relation $\to$. We define $(\leftarrow_R) = (\to_R)^{-1}$, $(\leftrightarrow) = \to_R \cup \leftarrow_R$, and $(\approx) = \leftrightarrow_R^*$. $\approx$ is an equivalence relation.

A normal form is a term that cannot be rewritten any further. We say n is a normal form of t if t can be reduced to n. A TRS R is *terminating* if there is no infinite rewriting chain $t_1 \to_R t_2 \ldots$. A TRS R is *confluent* if for all t, t_1, t_2, $t_1 \leftarrow_R^* t \to_R^* t_2$ implies there exists a t' such that $t_1 \to_R^* t' \leftarrow_R^* t_2$. We call a confluent and terminating TRS *convergent*. Every term in a terminating TRS has at least one normal form, every term in a confluent TRS has at most one normal form, and every term in a convergent TRS has exactly one normal form.

We call a term rewriting system left-linear (resp. right-linear) if variables in the left-hand side (resp. right-hand side) of each rewrite rule occur only once. For example, $R_1 = \{f(x, y) \to g(x)\}$ is left-linear, while $R_2 = \{f(x, x) \to g(x)\}$ is not left-linear. A TRS is linear if it's left-linear and right linear.

Finite tree automata

A finite tree automaton (FTA) is a tuple $\mathcal{A} = (Q, F, Q_f, \Delta)$, where Q is a set of states, F is a set of function symbols, $Q_f \subseteq Q$ is a set of final states, and Δ is a set of transitions of the form $f(q_1, \ldots, q_n) \to q$ where $q, q_1, \ldots, q_n \in Q$. A term t is accepted by a state by $\mathcal{A}$ if it can be rewritten to a final state $q_f \in Q_f$ of $\mathcal{A}$, i.e., $t \to^* q_f \in Q_f$. Since Q and F can be determined by Δ, we omit them and use (Q_f, Δ) to denote a FTA for brevity. Let $\mathcal{L}(\mathcal{A})$ be the set of terms accepted by FTA $\mathcal{A}$. A language L is called regular if it is accepted by some FTA ($\exists \mathcal{A}, L = \mathcal{L}(\mathcal{A})$).

Regular languages and FTAs are closed under union, intersection, and complement. Moreover, it is possible to define a tree automaton that accepts any term: define

$$\mathcal{A}_* = (\{q_*\}, \{f(q_*|_{i=1\ldots n}) \to q_* \mid n\text{-ary symbol } f \in F\})$$

for a fresh state q_*.

Given a *left-linear* term-rewriting system R, the set of normal forms of R is regular. The set of normal forms of R is the complement of the set of rewritable terms, i.e., terms whose subterm match some left-hand side patterns of R. We give such a construction below. The construction here requires left linearity to ensure that each "hole" in the left-hand side patterns can pick terms independently. For example, the set of rewritable terms of rule $f(x, x) \to x$ is not regular.

Procedure *termsMatchingPattern*(p)

Input: A linear pattern p.

Output: An FTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A})$ contains all terms matching the given pattern.

begin

1. $q_f \leftarrow$ *mkFreshState*();

case p **of**

  $f(p_1, \ldots, p_k) \Rightarrow$

    $(q_i, \Delta_i) \leftarrow$ *termsMatchingPattern*(p_i) **for** $i = 1, \ldots, k$;

    $q \leftarrow$ *mkFreshState*();

    $\Delta \leftarrow \{f(q_1, \ldots, q_k) \rightarrow q\} \cup \bigcup_{i=1,\ldots,k} \Delta_i$;

    return (q, Δ);

  $x \Rightarrow$ **return** A_*;

end

Procedure *subtermsMatchingPattern*(p)

Input: A linear pattern p.

Output: An FTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A})$ contains all terms containing the given pattern.

begin

$q_f \leftarrow$ *mkFreshState*();

$(q_p, \Delta) \leftarrow$ *termsMatchingPattern*(p);

$\Delta \leftarrow \Delta \cup \{q_p \rightarrow q_f\}$;

for each n-ary symbol f **where** $n > 0$ **do**

  for $i = 1, \ldots, n$ **do**

    $\Delta \leftarrow \Delta \cup \{f(q_*|_{j=1,\ldots,i-1}, q_f, q_*|_{j=i+1,\ldots,n}) \rightarrow q_f\}$;

return (q_f, Δ);

end

Procedure *normalForms*(R)

Input: A left-linear TRS R.

Output: An FTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A})$ is the set of normal forms of R.

begin

return $\overline{\bigcup_{lhs \rightarrow rhs \,\in\, R} \mathit{subtermsMatchingPattern}(lhs)}$;

end

E-graphs and equality saturation

We call an FTA deterministic if for every term t,

$$t \rightarrow^* q_1 \wedge t \rightarrow^* q_2 \rightarrow q_1 = q_2.$$

We call an FTA reachable its every state accepts some term. An e-graph G is a deterministic and reachable FTA (Q_f, Δ) with $|Q_f| = 1$. Moreover, G induces a relation $\approx_G$ defined as follows: if for two terms t_1 and t_2 there exists a state q in G such that $t_1 \rightarrow^* q \leftarrow^* t_2$, $t_1 \approx t_2$. $\approx_G$ is symmetric and reflexive. Moreover, if $t_1 \rightarrow^* q \leftarrow^* t_2$ and $t_2 \rightarrow^* q' \leftarrow^* t_3$, since an E-graph is deterministic, $t_1 \rightarrow^* q = q' \leftarrow^* t_3$, so $\approx_G$ is also transitive.

Turing machines

A Turing machine $\mathcal{M} = (Q, \Sigma, \Pi, \Delta, q_0, \beta)$ consists of a set of states Q, the input and the tape alphabet Σ and Π (with $\Sigma \subseteq \Pi$), a set of transitions Δ, an initial state $q_0 \in Q$, and a special blanket symbol $\beta \in \Pi$. Each transition in Δ is a quintuple in $Q \times \Pi \times \Pi \times \{L, R\} \times Q$. For example, transition $q_i abRq_j$ means if the current state is q_i and the symbol being scanned is a, then replace a with b, move the head to the right, and transit to state q_j. We assume the Turing machine is two-way infinite, so that the head can move in both directions indefinitely. Each configuration of $\mathcal{M}$ can be represented as $\triangleright u q_i a v \triangleleft$, where $\triangleright, \triangleleft$ are left and right end markers, u is the string to the left of the read/write head, q_i is the current state, a is the symbol being scanned, and v is the string to the right. We say $w_1 \vdash_{\mathcal{M}} w_2$ if configuration w_1 can transit to configuration w_2 in a Turing machine $\mathcal{M}$, and we omit $\mathcal{M}$ when it's clear from the context.

Termination of Equality Saturation

Theorem 1. The following problem is R.E.-complete:

> Instance: a set of rewrite rules R, a term t.
>
> Question: does EqSat terminate with R and t?

Proof. First, this problem is in R.E. since we can simply run EqSat with R and t to test whether it terminates. To show this problem is R.E.-hard, we reduce the termination problem of Turing machines to the termination of EqSat. We use the technique by [1]. In particular, for each Turing machine $\mathcal{M}$, we produce a string rewriting system R such that the equivalence closure of R, $(\approx_R) = (R \cup R^{-1})^*$, satisfies that each equivalence class of $\approx_R$ corresponds to a trace of the Turing machine. As a result, the Turing machine halts iff its trace is finite iff the corresponding equivalence class in R is finite iff EqSat terminates.

In this proof, we consider a degenerate form of EqSat that works with *string* rewriting systems instead of term rewriting systems. Every string corresponds to a term, and every string rewrite rule corresponds to a rewrite rule. For example, the string uvw corresponds to a term $u(v(w(\epsilon)))$, where $u(\cdot), v(\cdot), w(\cdot)$ are unary functions and ϵ is a constant, and a string rewrite rule $uvw \to vuw$ corresponds to a (linear) term rewriting rule $u(v(w(x))) \to v(u(w(x)))$ where x is a variable.

It is useful to define several sets of symbols for our construction. For each Turing machine $\mathcal{M}$, we define $\overline{Q} = \{\overline{q} \mid q \in Q\}$. We also define $\overline{\Sigma}$, $\overline{\Pi}$ in a similar way. In our encoding, we use $\overline{Q}$ to denote states where the symbol being scanned is to the left of the state, and we use $\overline{\Sigma}$ and $\overline{\Pi}$ to denote alphabets that are to the left of the states. Moreover, we introduce two sets of "dummy" symbols L_z and R_z for z ranges over $Q \times (\{\triangleleft\} \cup \Pi)$ and $(\{\triangleright\} \cup \overline{\Pi}) \times \overline{Q}$. Let D_L and D_R be the set of all L_z and R_z respectively. We use these dummy symbols to make the string rewriting system that we will later define Church-Rosser.

The rewriting system we are going to define works over the set of strings $\mathit{CONFIG} = \triangleright(\overline{\Pi} \cup D_L)^*(Q \cup \overline{Q})(\Pi \cup D_R)^*\triangleleft$. Strings in CONFIG is in a many-to-one mapping, denoted as π, to configurations of a Turing machine. $\pi(w)$ converts each $\overline{a}\overline{q}_i$ to $q_i a$, removes dummy symbols L_z and R_z, and replace $\overline{a}$ with a. For example $\pi(\triangleright L_{q_0,a}\overline{b}L_{q_1,b}\overline{c}\overline{q_3}dR_{q_i,\triangleleft}\triangleleft) = \triangleright b q_3 c d \triangleleft$

Now, the transitions in $\mathcal{M}$, we define our string rewriting system R as follows.

transitions in $\mathcal{M}$	rewrites in R
$q_i a b R q_j$	$q_i a \to_R L_{q_i a}\overline{b}q_j$
	$\overline{a}\overline{q}_i \to_R L_{\overline{a}q_i}\overline{b}q_j$
$q_i \beta b R q_j$	$q_i\triangleleft \to_R L_{q_i\triangleleft}\overline{b}q_j\triangleleft$
	$\triangleright\overline{q}_i \to_R \triangleright L_{\triangleright\overline{q}_i}\overline{b}q_j$
$q_i a b L q_j$	$q_i a \to_R \overline{q}_j b R_{q_i a}$
	$\overline{a}\overline{q}_i \to_R \overline{q}_j b R_{\overline{a}\overline{q}_i}$
$q_i \beta b L q_j$	$q_i\triangleleft \to_R \overline{q}_j b R_{q_i\triangleleft}\triangleleft$
	$\triangleright\overline{q}_i \to_R \triangleright\overline{q}_j b R_{\triangleright\overline{q}_i}$

Moreover, for each z, we have the following two additional (sets of) auxiliary rewrite rules

$$q_i R_z \to_R L_z L_z q_i$$
$$L_z \overline{q}_i \to_R \overline{q}_i R_z R_z$$

for any z.

To explain what these two rules do, let us define two types of strings. Type-A strings are strings where the symbol being scanned is to the immediate right of q_i or to the immediate left of $\overline{q_i}$. In other words, we call a string s

a type-A string if s contains $q_i a$ or $\overline{aq_i}$. Type-B strings are strings that are not type-A: they are strings where there are dummy symbols in between the state and the symbol being scanned. The rewrite rules above convert any type-B strings into type-A in a finite number of steps.

Now, we observe that R has several properties:

1. Reverse convergence: the critical pair lemma implies that if a rewriting system is terminating and all its critical pairs are convergent, it is convergent. Define $\leftarrow_R = (\rightarrow_R)^{-1}$. $\leftarrow_R$ is terminating since rewrite rules in $\leftarrow_R$ decreases the sizes of terms (that is, rewrite rules in $\rightarrow_R$ increases the sizes of terms), and $\leftarrow_R$ has no critical pairs. Therefore, $\leftarrow_R$ is convergent.
2. For each type-A string w, then either
   - there exists no w' with $w \rightarrow_R w'$ and $\pi(w)$ is a halting configuration;
   - there exists a unique w' such that $w \rightarrow_R w'$ and $\pi(w) \vdash \pi(w')$.
3. For each type-B string w, there exists a unique w' such that $w \rightarrow_R w'$, and $\pi(w) = \pi(w')$. Moreover, if $w_0 \rightarrow_R w_1 \rightarrow_R \ldots$ is a sequence of type-B strings, the sequence must be bounded in length, since the state symbols q_i and $\overline{q}_i$ move towards one end according to the auxillary rules above.
4. By 2 and 3, $w \rightarrow_R w_1$ and $w \rightarrow_R w_2$ implies $w_1 = w_2$. In other words, $\rightarrow_R$ is a function.

These observations allows us to prove the following lemma

Lemma 2. Let $w_0 = \triangleright q_0 s \triangleleft$ be an initial configuration. w_0 is obviously in *CONFIG*. Moreover, given a Turing machine $\mathcal{M}$, construct a string rewriting system R as above. $\mathcal{M}$ halts on w_0 if and only if $[w_0]_R$, the equivalence class of w_0 in R, is finite.

Proof. Consider $S : w_0 \rightarrow_R w_1 \rightarrow_R \ldots$, a sequence of *CONFIG* starting with w_0. By the above observations, S must have a subsequence of type-A strings $w_0 \rightarrow_R^* w_{a_1} \rightarrow_R^* w_{a_2} \rightarrow_R^* \ldots$ with

$$\pi(w_0) = \ldots = \pi(w_{a_1-1}) \vdash \pi(w_{a_1}) = \ldots = \pi(w_{a_2-1}) \vdash \pi(w_{a_2}) = \ldots .$$

An overview of the trace $w_0, w_{a_1}, w_{a_2}, \ldots$ and its properties is shown below:

Rw	w_0	$\rightarrow_R$	$\underbrace{w_1 \rightarrow_R \ldots \rightarrow_R w_{a_1-1}}_{\text{finite}}$	$\rightarrow_R$	w_{a_1}	$\underbrace{w_{a_1+1} \rightarrow_R \ldots \rightarrow_R w_{a_2-1}}_{\text{finite}}$	$\rightarrow_R$	w_{a_2}	...
Type	A		B ... B		A	B ... B		A	
Config	$\pi(w_0)$	$=$	$\pi(w_1) = \ldots = \pi(w_{a_1-1})$	$\vdash_{\mathcal{M}}$	$\pi(w_{a_1})$	$\pi(w_{a_1+1}) = \ldots = \pi(w_{a_2-1})$	$\vdash_{\mathcal{M}}$	$\pi(w_{a_2})$	...

Now we prove the claim:

- $\Leftarrow$: Suppose $[w_0]_R$ is finite. We show that there exists a *finite* sequence S of $w_0 \rightarrow_R w_1 \rightarrow_R \ldots \rightarrow_R w_n$ such that there is no w' such that $w_n \rightarrow_R w'$. If this is not the case, then an infinite rewriting sequence $w_0 \rightarrow_R w_1 \rightarrow \ldots$ must exist. Because $[w_0]_R$ is finite, for the sequence to be infinite, there must exist distinct i, j such that $w_i = w_j$ in the sequence. However, this is impossible, because $\rightarrow_R$ always increases the sizes of terms.

  By our observation above, if there is no such w' that $w_n \rightarrow_R w'$ in sequence S, it has to be the case that w_n is type-A and $\pi(w)$ is a halting configuration.

  Now, take the subsequence of S that contains every type-A string:

  $$w_0 \rightarrow_R^* w_{a_1} \rightarrow_R^* \ldots \rightarrow_R^* w_{a_k} = w_n.$$

  We have $\pi(w_{a_i}) \vdash \pi(w_{a_{i+1}})$ for all i and $\pi(w_{a_k})$ is a halting configuration. This implies a finite trace of the Turing machine:

  $$w_0 \vdash \pi(w_{a_1}) \vdash \ldots \vdash \pi(w_{a_n}).$$

  Since we only consider deterministic Turing machines, the Turing machine halts on w_0.

- $\Rightarrow$: Suppose otherwise $\mathcal{M}$ halts on w_0 and $[w_0]_R$ is infinite.

  By definition, w_0 is a normal form with respect to $\leftarrow_R$, and because $\leftarrow_R$ is convergent, if there exists a w such that $w \approx_R w_0$, then $w_0 \rightarrow_R^* w$. The fact that $[w_0]_R$ is infinite implies w_0 can be rewritten to infinitely many strings w. Because $\to_R$ satisfies the functional dependency, it has to be the case that there exists an infinite rewriting sequence: $S : w_0 \to_R w_1 \to_R \ldots$. Taking the subsequence of S consisting of every type-A strings:

$$w_0 \to_R^* w_{a_1} \to_R^* \ldots .$$

This implies an infinite trace of the Turing machine:

$$w_0 \vdash \pi(w_{a_1}) \vdash \ldots ,$$

which is a contradiction.

We are ready to prove the undecidability of the termination problem of EqSat:

Given a Turing machine $\mathcal{M}$. We construct the following two-tape Turing machine $\mathcal{M}'$:

$\mathcal{M}'$ alternates between the following two steps:

1. Simulate one transition of $\mathcal{M}$ on its first tape.
2. Read the string on its second tape as a number, compute the next prime number, and write it to the second tape.

$\mathcal{M}'$ halts when the simulation of $\mathcal{M}$ reaches an accepting state.

It is known that a two-tape Turing machine can be simulated using a standard Turing machine, so we assume $\mathcal{M}'$ is a standard Turing machine and takes input string (s_1, s_2), where s_1 is the input on its first tape and s_2 is the input on its second tape. Let R' be the string rewriting system derived from $\mathcal{M}'$ using the encoding we introduced in the lemma.

Given a string s, let w be the initial configuration $\rhd q_0(s, 2)\lhd$. The following conditions are equivalent to each other:

1. $\mathcal{M}$ halts on input s.
2. $\mathcal{M}'$ halts on input $(s, 2)$.
3. $[w]_{R'}$ is finite.
4. $[w]_{R'}$ is regular.

Note that (3) implies (4) trivially, and (4) implies (3) because if $[w]_{R'}$ is infinite, it must not be regular since the trace of $\mathcal{M}'$ computes every prime number.

Now run EqSat with initial string w and rewriting system $\leftrightarrow_{R'}$. EqSat terminates if and only if $\mathcal{M}$ halts on s:

- $\Rightarrow$: Suppose EqSat terminates with output E-graph G. Strings equivalent to w in G is exactly the equivalence class of w, i.e., $[w]_G = [w]_{R'}$. Moreover, every e-class in an E-graph represents a regular language, so $[w]_G$ is regular. Therefore, $[w]_G$ is finite.
- $\Leftarrow$: Suppose $\mathcal{M}$ halts on s. This implies $[w]_{R'}$ is finite. Because EqSat monotonically enlarges the set of represented terms, it has to stop in a finite number of iterations.

Because the halting problem of a Turing machine is undecidable, the termination problem of EqSat is undecidable as well. ■

Theorem 3. The following problem is undecidable.

Instance: a set of rewrite rules R, a term w.

Problem: Is $[w]_R$ regular?

Proof.

To show the undecidability, we reduce the halting problem of Turing machines to this problem. As shown in Theorem 1, given a Turing machine $\mathcal{M}$, $\mathcal{M}$ halts on an input s if and only if $[w]_{R'}$ is regular for $w = \rhd q_0(s, 2)\lhd$. ■

For a particular kind of rewrite systems, we show this regularity problem is R.E.-complete.

Theorem 4. The following problem is R.E.-complete.

Instance: a set of linear, convergent rewrite rules R, a term w.

Problem: Is $[w]_R$ regular?

Proof.

As we show in Theorem 1, the regularity of $\leftarrow_R$ is undecidable. Note that $\leftarrow_R$, the inverse of $\rightarrow_R$ defined in Theorem 1, is convergent. Moreover, because every string rewriting system is a linear term rewriting system and therefore a left-linear term rewriting system, $\leftarrow_R$ is left-linear. Therefore, the regularity of left-linear, convergent term rewriting systems is undecidable. Additionally, we show the regularity problem is in R.E. by showing a semi-decision procedure for it.

Procedure *equivClassOf*(R, w)

Input: a left-linear, convergent term rewriting system R, a term w.

Output: an E-graph that represents $[w]_R$ if exists.

begin

1. **for each** E-graph G such that $w \in \mathcal{L}(G)$ **do**
2. **if** $G = \mathit{runEqSatOneIter}(G, \leftrightarrow_R)$ **then**
3. **if** $\mathcal{L}(G) \cap \mathit{normalForms}(R) = \{w\}$ **then**
4. **return** G;

end

We show the correctness of our algorithm in two steps.

- We show that if an e-graph G is returned, $\mathcal{L}(G) = [w]_R$: First, if $G = \mathit{runEqSatOneIter}(G, \leftrightarrow_R)$, we have, for any term t,

$$t \in \mathcal{L}(G) \Rightarrow [t]_R \subseteq \mathcal{L}(G). \tag{1}$$

  Suppose this is not the case. There must exist term u, v where $u \leftrightarrow_R v$, $u \in \mathcal{L}(G)$, and $v \in \mathcal{L}(G)$, and running one iteration of equality saturation will further enlarge the e-graph, which is a contradiction. Therefore, since $w \in \mathcal{L}(G)$, $[w]_R \subseteq \mathcal{L}(G)$.

  Second, we show $\mathcal{L}(G) \subseteq [w]_R$. Suppose this is not the case. There exists a term $u \in \mathcal{L}(G)$ that is in a different equivalence class than $[w]_R$. By (1), $[u]_R \subseteq \mathcal{L}(G)$. Because R is convergent, $[u]_R$ has a normal form n_u that is contained in $\mathcal{L}(G)$, but line 3 ensures that $\mathcal{L}(G)$ has one normal form which is w, a contradiction.

- On the other hand, if there exists an e-graph G such that $\mathcal{L}(G) = [w]_R$. This case is straightforward: if $\mathcal{L}(G) = [w]_R$, G is "saturated" with regard to $\leftrightarrow_R$. Moreover, since R is convergent, $[w]_R$ has only one normal form which is w. ■

References

[1] Narendran, P., Ó'Dúnlaing, C. and Rolletschek, H. 1985. Complexity of certain decision problems about congruential languages. *Journal of Computer and System Sciences.* 30, 3 (1985), 343–358. DOI:https://doi.org/https://doi.org/10.1016/0022-0000(85)90051-0.