The Termination Problem of Equality Saturation is Undecidable

Yihong Zhang

Aug 11, 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* $\to_R$ as follows: $u \to_R v$ if and only if there exists a rule $(l \to r) \in R$ and a substitution σ such that a subterm of u at path p is $l\sigma$, and v is u whose subterm at path p is substituted with $r\sigma$. We omit the subscript R when it's clear from the context. Let $\to^*$ be the transitive closure of binary relation $\to$. We define $(\leftarrow_R) = (\to_R)^{-1}$, $(\leftrightarrow_R) = (\to_R) \cup (\leftarrow_R)$, and $(\approx_R) = \leftrightarrow_R^*$. $\approx_R$ 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 reduces to n and n is a normal form. A TRS R is *terminating* if there is no infinite rewriting chain $t_1 \to_R t_2 \to \dots$. 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*. Terms in a terminating TRS have at least one normal form, terms in a confluent TRS have at most one normal form, and terms in a convergent TRS have 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) pattern 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 $\mathcal{A} = (Q, F, Q_{final}, \Delta)$, where Q is a set of states, F is a set of function symbols, $Q_{final} \subseteq Q$ is a set of final states, and Δ is a set of transitions of the form $f(q_1, \dots, q_n) \to q$ where $q, q_1, \dots, q_n \in Q$. A term t is accepted by $\mathcal{A}$ if it can be rewritten to a final state $q_{final} \in Q_{final}$ of $\mathcal{A}$, i.e., $t \to^* q_{final} \in Q_{final}$. Since Q and F can be determined by Δ, we omit them and use (Q_{final}, 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 ($\exists \mathcal{A}, L = \mathcal{L}(\mathcal{A})$). When the set of transitions Δ is clear from the context, we further omit it and use $\mathcal{L}(c)$ to denote the language accepted the tree automata $\mathcal{A} = (c, \Delta)$.

Regular languages and FTAs are closed under union, intersection, and complementation. Moreover, it is possible to define a tree automaton that accepts any term: it is the complement of an empty FTA. To give an explicit construction, define

$$\mathcal{A}_* = (\{q_*\}, \{f(\underbrace{q_*, \dots, q_*}_{n}) \to q_* \mid n\text{-ary symbol } f \in F\})$$

for a fresh state q_*.

Left-regularity of a term rewriting system implies the regularity of its normal forms: Given a term-rewriting system R, 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. Suppose R is left-linear. The set of rewritable terms is regular. Since regularity is preserved under complementation, the set of normal forms of R is regular. Below is an algorithm that computes a tree automata whose language is the set of normal forms. 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.   **case** p **of**

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

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

    4. $q \leftarrow$ *mkFreshState*();

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

    6.   **return** (q, Δ);

    7. $x \Rightarrow$ **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

    1. $q_{final} \leftarrow$ *mkFreshState*();

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

    3. $\Delta \leftarrow \Delta \cup \{q_p \rightarrow q_{final}\}$;

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

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

    6. $\Delta \leftarrow \Delta \cup \{f(\underbrace{q_*, \ldots, q_*}_{i-1}, q_{final}, \underbrace{q_*, \ldots, q_*}_{n-i}) \rightarrow q_{final}\}$;

    7.   **return** (q_{final}, 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

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

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 for every state q there exists a ground term t such that $t \rightarrow^* q$. An e-graph G is a deterministic and reachable FTA $(\{q_{final}\}, \Delta)$ with a single final state. Moreover, G induces a relation $\approx_G$ defined

as follows: for any terms t_1 and t_2, if there exists a state q in G such that $t_1 \to^* q \leftarrow^* t_2$, then $t_1 \approx t_2$[1]. $\approx_G$ is an equivalence relation: $\approx_G$ is symmetric and reflexive by definition. Moreover, if $t_1 \to^* q \leftarrow^* t_2$ and $t_2 \to^* q' \leftarrow^* t_3$, since an E-graph is deterministic, $t_1 \to^* q = q' \leftarrow^* t_3$, so $\approx_G$ is also transitive.

Equality saturation is defined as the inductive fixed point of rule applications and rebuilding:

$$EqSat(R, w) = (CC \circ T_p)^\infty (mkEGraph(w)).$$

We omit the definition of CC and T_p here, which can be found in the egg and egglog paper. But we remind the readers several properties of equality saturation here: First, $T_p(G)$ is not necessarily deterministic, even when G is an E-graph (i.e., deterministic and reachable tree automaton). The congruence closure operator CC recovers the determinicity. Second,

$$\{v \mid u \in \mathcal{L}(G), u = v \lor u \to_R v\} \subseteq \mathcal{L}(T_p(G)) \subseteq \mathcal{L}((CC \circ T_p)(G));$$

that is, the e-graph after rule application at least represents terms derived from the initial e-graph by rewriting zero or one time, and the congruence closure further grows the E-graph. In many cases, the set containment is strict, see my earlier blog post on tree automata completion. As a corollary,

$$\{t \mid w \to_R^* t\} \subseteq EqSat(w).$$

Finally, in terms of the equivalence relation derived from E-graphs, we have

$$(\approx_G) \subseteq \left(\approx_{T_p(G)}\right) \subseteq \left(\approx_{(CC \circ T_p)(G)}\right).$$

Given a term rewriting system R where every rule preserves the set of variables (i.e., $vars(lhs) = vars(rhs)$), let R^{-1} be the term rewriting system obtained by swapping left-hand sides and right-hand sides of patterns in R. We have $[w]_R = \mathcal{L}(EqSat(R \cup R^{-1}, w))$

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 blank 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 $q_i a b R q_j$ 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 $\rhd u q_i a v \lhd$, where $\rhd, \lhd$ 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 term rewriting system 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 [3]. 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 following statements are equivalent:

- the Turing machine halts;

---

[1]Note that this definition of $\approx_G$ is different from the congruence relation defined in the Myhill-Nerode theorem for trees [2]. For example, consider an E-graph with transitions $\{a() \to c_1, b() \to c_2, f(c_1) \to c_f, f(c_2) \to c_f\}$. In the Myhill-Nerode theorem, fa and fb would be equivalent while in our definition they are not because they are accepted by different states.

- the trace of the Turing machine is finite;
- the equivalence class of the initial configuration in R is finite;
- 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 special constant. A string rewrite rule $uvw \rightarrow vuw$ corresponds to a (linear) term rewriting rule $u(v(w(x))) \rightarrow v(u(w(x)))$ 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 $Q \times (\{\lhd\} \cup \Pi)$ and $(\{\rhd\} \cup \overline{\Pi}) \times \overline{Q}$. 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 $CONFIG = \rhd(\overline{\Pi} \cup D_L)^*(Q \cup \overline{Q})(\Pi \cup D_R)^*\lhd$. Strings in $CONFIG$ is in a many-to-one mapping to configurations of a Turing machine. We denote this mapping as π: $\pi(w)$ 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 $\pi(\rhd L_{q_0,a}\overline{b}L_{q_1,b}\overline{cq_3}dR_{q_i,\lhd}\lhd) = \rhd bq_3cd\lhd$

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

transitions in $\mathcal{M}$	rewrites in R
$q_i abRq_j$	$q_i a \rightarrow_R L_{q_i a}\overline{b}q_j$
	$\overline{a}\overline{q}_i \rightarrow_R L_{\overline{a}q_i}\overline{b}q_j$
$q_i \beta bRq_j$	$q_i \lhd \rightarrow_R L_{q_i \lhd}\overline{b}q_j \lhd$
	$\rhd\overline{q}_i \rightarrow_R \rhd L_{\rhd\overline{q}_i}\overline{b}q_j$
$q_i abLq_j$	$q_i a \rightarrow_R \overline{q}_j bR_{q_i a}$
	$\overline{a}\overline{q}_i \rightarrow_R \overline{q}_j bR_{\overline{a}q_i}$
$q_i \beta bLq_j$	$q_i \lhd \rightarrow_R \overline{q}_j bR_{q_i \lhd}\lhd$
	$\rhd\overline{q}_i \rightarrow_R \rhd\overline{q}_j bR_{\rhd\overline{q}_i}$

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

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

for any z.

To explain what the 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 $\overline{a}\overline{q}_i$. 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 R^{-1} to be a TRS derived from R by swapping left and right hand side. R^{-1} is terminating since rewrite rules in R^{-1} decreases the sizes of terms (that is, rewrite rules in R^{-1} increases the sizes of terms), and R^{-1} has no critical pairs. Therefore, R^{-1} 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 $w \rightarrow_R w'$. Moreover, $\pi(w) \vdash \pi(w')$.
3. For each type-B string w, there exists a unique w' such that $w \rightarrow_R w'$. It holds that $\pi(w) = \pi(w')$. Moreover, if $w_0 \rightarrow_R w_1 \rightarrow_R ...$ 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

4. By 2 and 3, $w \to_R w_1$ and $w \to_R w_2$ implies $w_1 = w_2$. In other words, $\to_R$ 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 $S : w_0 \to_R w_1 \to_R ...$, a sequence of *CONFIG* starting with w_0. By the above observations, S must have a subsequence of type-A strings $w_0 \to_R^* w_{a_1} \to_R^* w_{a_2} \to_R^* ...$ with

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

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

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

Now we prove the claim:

- $\Leftarrow$: Suppose $[w_0]_R$ is finite. We show that there exists a *finite* sequence S of $w_0 \to_R w_1 \to_R ... \to_R w_n$ such that there is no w' such that $w_n \to_R w'$. If this is not the case, then an infinite rewriting sequence $w_0 \to_R w_1 \to ...$ 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 $\to_R$ always increases the sizes of terms.

  By our observation above, if there is no such w' that $w_n \to_R w'$ 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:

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

  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:

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

  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 $w_0 \to_R^* w$. 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: $S : w_0 \to_R w_1 \to_R ...$. Taking the subsequence of S consisting of every type-A strings:

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

  This implies an infinite trace of the Turing machine:

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

  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.

5

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 $\triangleright q_0(s, 2) \triangleleft$. The following conditions are equivalent:

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. ∎

The regularity problem of term rewriting systems

In the last section, we have shown that the termination of equality saturation is undecidable. Here, we consider a more general question: Given a term rewriting system and a term, is the equivalence class of the term defined by the term rewriting system regular? In general, for term rewriting systems consisting of variable-preserving rewrite rules, if equality saturation terminates on $R \cup R^{-1}$, the equivalence class is regular. However, there are cases where the equivalence class of a term is regular, equality saturation does not terminate: an example is term rewriting system $\{f(x) \rightarrow f(g(x))\}$ and term $f(a)$.

This regularity problem is undecidable, as we show here.

Theorem 3. The following problem is undecidable.

Instance: a term rewriting system 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 $[w]_{R'}$ (as constructed in Theorem 1) is regular for $w = \triangleright q_0(s, 2) \triangleleft$. ∎

Different from the termination of equality saturation, it is open whether this problem is recursive enumerable. However, 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 left-linear and convergent term rewriting system R, a term w.

Problem: Is $[w]_R$ regular?

Proof.

As we show in Theorem 1, the regularity of R (and therefore R^{-1}) is undecidable. Because every string rewriting system is a linear term rewriting system R^{-1} is left-linear. Moreover, R^{-1} is convergent. 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** *isFixPoint*$(G, R \cup R^{-1})$ **then**

    3.   **if** $\mathcal{L}(G) \cap normalForms(R) = \{w\}$ **then**

    4.   **return** G;

end

In the above algorithm, *isFixPoint* checks if the E-graph G is "saturated" with respect to R and R^{-1}. It does this by checking that, for each matched left-hand side $l\sigma$ of the pattern, the right-hand side $r\sigma$ exists in the E-graph and is equivalent to $l\sigma$. Some care needs to be taken here: consider a rewrite rule $f(x, y) \to g(x)$. The reverse form of this rule is $g(x) \to f(x, y)$. The right-hand side pattern of this rewrite rule has a larger variable set than the left-hand side. To handle this rewrite rule, *isFixPoint* matches the left-hand side pattern $g(x)$, and each match produces a substitution $\{x \mapsto c_x\}$ at root E-class c. *isFixPoint* checks regular set containment $\{f(t_x, t_y) \mid t_x \in \mathcal{L}(c_x), t_b \in *\} \subseteq \mathcal{L}(c)$, where $*$ is the universe of all terms.

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, we show $[w]_R \subseteq \mathcal{L}(G)$: if *isFixPoint*$(G, R \cup R^{-1})$ holds, we have, for any term t, if t is accepted by G, the entire equivalence class of t is also accepted by G; that is

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

     Since $w \in \mathcal{L}(G)$, $[w]_R \subseteq \mathcal{L}(G)$. To prove (1), suppose this is not the case. There must exist term u, v where $u \leftrightarrow_R v$, $u \in \mathcal{L}(G)$, and $v \notin \mathcal{L}(G)$, which contradicts *isFixPoint*$(G, R \cup R^{-1})$.
  - 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 $\mathcal{L}(G) = [w]_R$, it will be returned. This case is straightforward: if $\mathcal{L}(G) = [w]_R$, G is "saturated" with regard to $\leftrightarrow_R$, so the check at line 3 passes. Moreover, since R is convergent, $[w]_R$ has only one normal form which is w, so the check at line 4 also passes. Therefore, G will be returned. ∎

Relaxing left-linearity

In the last theorem, the set of normal forms is intersected with the language represented by the e-graph, and checked for equivalence with the singleton set $\{w\}$. This is why we require the term rewriting system to be left-linear: the set of normal forms of a left-linear term rewriting system is regular, and regular languages are closed under intersection and has decidable equivalence checking. Therefore, if there is a representation of normal forms for general term rewriting systems that is closed under intersection and has decidable equivalence checking, then we can lift the left linearity condition in the last theorem.

Indeed, such a representation exists. It is called *tree automata with global constraints* (TAGCs) [1]. Essentially, TAGCs extend tree automata with *global* equality and inequality constraints. TAGCs are closed under union and intersection, but not under complementation. Moreover, the emptiness and membership problem of TAGC is decidable. Therefore, if we can represent the set of normal forms of a term rewriting system as a TAGC, then

line 3 of the algorithm used in the last theorem can be checked decidably for a general term rewriting system by considering its equivalent form:

$$w \in L \land L \cap \overline{\{w\}} = \emptyset$$

where $L = \mathcal{L}(G) \cap normalForms(R)$. I claim $normalForms(R)$ can be represented as an TAGC for an arbitrary term rewriting system, although I will omit the details of the construction here since it is a bit technical[2]. Putting everything together, we have the following generalization of Theorem 4:

Theorem 5. The following problem is R.E.-complete.

Instance: a convergent term rewriting system R, a term w.

Problem: Is $[w]_R$ regular?

References

[1]   Barguñó, L., Creus, C., Godoy, G., Jacquemard, F. and Vacher, C. 2010. The emptiness problem for tree automata with global constraints. *2010 25th annual IEEE symposium on logic in computer science* (2010), 263–272.

[2]   Kozen, D. 1992. On the myhill-nerode theorem for trees. *Bulletin of the EATCS*. 47, (1992), 170–173.

[3]   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.

---

[2]In particular, we cannot use our strategy for left-linear term rewriting systems by first representing the set of rewritable terms and taking its complement, since TAGC is not closed under complementation. Luckily, for each rewrite rule, we can explicitly represent terms that does not match the left-hand side with inequality constraints, and then take the intersection of them.