Disproving Termination of Non-Erasing Sole Combinatory Calculus with Tree Automata

Keisuke Nakano[1][0000−0003−1955−4225]
Munehiro Iwami[2][0000−0001−9925−450X]

[1] Tohoku University, Sendai, Miyagi, Japan
k.nakano@acm.org
[2] Shimane University, Matsue, Shimane, Japan
munehiro@cis.shimane-u.ac.jp

Abstract. We study the termination of sole combinatory calculus, which consists of only one combinator. Specifically, the termination for non-erasing combinators is disproven by finding a desirable tree automaton with a SAT solver as done for term rewriting systems by Endrullis and Zantema. We improved their technique to apply to non-erasing sole combinatory calculus, in which it suffices to search for tree automata with a final sink state. Our method succeeds in disproving the termination of 8 combinators, whose termination has been an open problem.

Keywords: Combinatory calculus · Non-termination · Tree automata.

1 Introduction

Combinatory logic [15, 3] has been used in computer science as a theoretical model of computation and also as a basis for the design of functional programming languages [18, 13]. It can be viewed as a variant of lambda calculus, in which a limited set of combinators, primitive functions without free variables, is used instead of lambda abstractions.

Combinators in combinatory logic are defined as $Zx_1x_2 \dots x_n \to e$ where Z is a combinator, $x_1, \dots, x_n$ are variables, and e is built from the variables with a function application. It is known that a small set of combinators can define a combinatorial calculus that is sufficient to cover all computable functions. Well-known sets of such combinators are $\{\mathbf{S}, \mathbf{K}\}$ and $\{\mathbf{B}, \mathbf{C}, \mathbf{K}, \mathbf{W}\}$ with $\mathbf{S}xyz \to xz(yz)$, $\mathbf{K}xy \to x$, $\mathbf{B}xyz \to x(yz)$, $\mathbf{C}xyz \to xzy$, and $\mathbf{W}xy \to xyy$ [1].

The subject of this paper is *sole combinatory calculus*, which consists of only one combinator. There have been several studies on sole combinatory calculi. Waldmann [19] investigated the $\mathbf{S}$ combinator to provide a procedure that decides whether an $\mathbf{S}$-term, built from $\mathbf{S}$ alone, has a normal form and further showed that the set of normalizing $\mathbf{S}$-terms is recognizable. Probst and Studer [14] proved that the sole combinatory calculus with the $\mathbf{J}$ combinator, defined by $\mathbf{J}xyzw \to xy(xwz)$ is strongly normalizing; that is, no $\mathbf{J}$-term

$$\mathbf{P}xyz \to z(xyz) \quad \mathbf{P}_3xyz \to y(xzy) \quad \boldsymbol{\mathcal{D}}_1xyzw \to xz(yw)(xz) \quad \boldsymbol{\mathcal{D}}_2xyzw \to xw(yz)(xw)$$
$$\mathbf{\Phi}xyzw \to x(yw)(zw) \quad \mathbf{\Phi}_2xyzw_1w_2 \to x(yw_1w_2)(zw_1w_2) \quad \mathbf{S}_1xyzw \to xyw(zw)$$
$$\mathbf{S}_2xyzw \to xzw(yzw) \qquad \mathbf{S}_3xyzwv \to xy(zv)(wv) \qquad \mathbf{S}_4xyzwv \to z(xwv)(ywv)$$

Fig. 1. Combinators and their reduction rules

has an infinite reduction sequence. Ikebuchi and Nakano [6] showed that the sole combinatory calculus with the $\mathbf{B}$ combinator is strongly normalizing and characterized by equational axiomatization for proving the looping and non-looping properties of repetitive right applications.

This paper concerns the non-termination of sole combinatory calculi, where termination means that no term has an infinite reduction. Let us say that a combinator is *terminating* when the corresponding sole combinatory calculus is terminating. Iwami [8] has investigated the termination of 37 combinators introduced in Smulyan's book [16] and has reported that 10 of them shown in Fig. 1 are of unknown termination.

We disprove the termination of 8 of the 10 combinators. The main idea of disproving the termination is to give a non-empty recognizable set of terms closed under the reduction as Endrullis and Zantema have done [4]. They showed that a SAT solver can find the corresponding tree automaton. We improve their method by showing that it suffices to search for tree automata with a final sink state in our setting and by reducing the number of variables in the SAT problem. Our implemetation disproves the termination of 8 combinators.

2 Preliminaries

A *signature* (or *alphabet*) Σ is a non-empty finite set of *function symbols*, each with a fixed natural number called *arity* (or *rank*)[3]. The set of all function symbols of arity n in Σ is written as $\Sigma^{(n)}$. We may write $f^{(n)}$ for $f \in \Sigma^{(n)}$. A function symbol of arity 0 is called a *constant* symbol. A set of *variables* is a countably infinite set disjoint from Σ. For a set $\mathcal{V}$ of variables, a set of *terms* over Σ, denoted by $\mathcal{T}_\Sigma(\mathcal{V})$, is inductively defined as the smallest set S such that $\mathcal{V} \subseteq S$ and $t_1, \ldots, t_n \in S$ implies $f(t_1, \ldots, t_n) \in S$ for every $f \in \Sigma^{(n)}$. The set of variables occurring in $t \in \mathcal{T}_\Sigma(\mathcal{V})$ is denoted by $\mathsf{FV}(t)$. In the rest of the paper, the set $\mathcal{V}$ of variables is fixed and contains $x, x_1, x_2, \ldots$ as its elements. A *substitution* is a finite map from variables to terms. We write $\mathsf{dom}(\alpha)$ for the domain of a substitution α. For a term t and a substitution α, we denote by $t\alpha$ an *instance* of t, a term obtained by replacing every variable x in t with $\alpha(x)$. Substitutions may be represented in the set notation as usual: we write $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ for substitution α when $\mathsf{dom}(\alpha) = \{x_1, \ldots, x_n\}$

[3] We use the terminology of term rewriting systems, whereas definitions are given alongside that of formal language theory for the readers, e.g., a *tree* and a *rank* in formal language are called a *(ground) term* and an *arity* in term rewriting, respectively.

and $\alpha(x_i) = t_i$ holds for each i and, in particular, $\emptyset$ for substitution α when $\mathsf{dom}(\alpha) = \emptyset$. A term containing no variables is called a *ground term*, and the set of ground terms is written as $\mathcal{T}_\Sigma$, i.e., $\mathcal{T}_\Sigma = \mathcal{T}_\Sigma(\emptyset)$. A *context* C is a term over $\Sigma \cup \{\square\}$ with a constant symbol $\square$, called a *hole*, such that $\square$ occurs exactly once in C. For a context C and a term t, we denote by $C[t]$ a term obtained by replacing the hole in C with t. We write $\mathcal{C}_\Sigma(\mathcal{V})$ for the set of contexts; in particular, $\mathcal{V}$ is omitted from the notation if it is empty, i.e., $\mathcal{C}_\Sigma = \mathcal{C}_\Sigma(\emptyset)$. A term $s \in \mathcal{T}_\Sigma$ is a *subterm* of $t \in \mathcal{T}_\Sigma$ if $t = C[s]$ holds with some $C \in \mathcal{C}_\Sigma$. The set of all subterms of t is written as $\mathsf{Sub}(t)$.

A *term rewriting system* (TRS) R over Σ is a set of rewriting rules of the form $l \to r$ with $l, r \in \mathcal{T}_\Sigma(\mathcal{V})$ where $\mathsf{FV}(r) \subseteq \mathsf{FV}(l)$. A TRS R is said to be *non-erasing* if $\mathsf{FV}(r) = \mathsf{FV}(l)$ holds for every rule $l \to r \in R$. A TRS R is said to be *left-linear* if each variable in $\mathsf{FV}(l)$ occurs exactly once in l for every rule $l \to r \in R$. A left-linear TRS R is said to be *orthogonal* if every pair of two (possibly the same) rules in R has no overlapping, which means that the left-hand side of one rule is not unifiable with any non-variable subterm of the left-hand side of the other rule. A *rewrite relation* $\to_R$ over $\mathcal{T}_\Sigma(\mathcal{V})$ induced by R is defined by $\{(C[l\alpha], C[r\alpha]) \mid C \in \mathcal{C}_\Sigma(\mathcal{V}),\ l \to r \in R,\ \alpha : \mathsf{FV}(l) \to \mathcal{T}_\Sigma(\mathcal{V})\}$. We write $\to_R^*$ for the reflexive and transitive closure of $\to_R$. A term t is called a *redex* of R if $t = l\alpha$ holds for some $l \to r \in R$ and substitution α; that is, a redex means a reducible part of a term. A term t is said to be in *normal form* with respect to R if there is no term u such that $t \to_R u$; in other words, no subterm of t is a redex of R. A set of normal forms with respect to R is denoted by $\mathsf{NF}(R)$. A TRS R is *terminating* or *strongly normalizing* if no infinite rewrite sequence $t_0 \to_R t_1 \to_R t_2 \to_R \cdots$ with $t_i \in \mathcal{T}_\Sigma(\mathcal{V})$ and $i \in \mathbb{N}$ exists. A TRS R is *weakly normalizing* if for every term t there exists a term $u \in \mathsf{NF}(R)$ such that $t \to_R^* u$ holds. A rewrite step $s \to_R t$ is *innermost*, denoted by $s \xrightarrow{\text{I}}_R t$, if no proper subterm of the contracted redex is itself a redex [12], that is, the relation $\xrightarrow{\text{I}}_R$ is defined by a subset of $\to_R$ as $\{(C[l\alpha], C[r\alpha]) \mid C \in \mathcal{C}_\Sigma(\mathcal{V}),\ l \to r \in R,\ \alpha : \mathsf{FV}(l) \to \mathcal{T}_\Sigma(\mathcal{V}),\ \mathsf{Sub}(l\alpha) \subseteq \mathsf{NF}(R) \cup \{l\alpha\}\}$. A TRS R is *weakly innermost normalizing* if for every term t there exists a term $u \in \mathsf{NF}(R)$ such that $t \xrightarrow{\text{I}}_R^* u$ holds. A set S of terms is *closed* under R if for every $t \in S$, $t \to_R u$ implies $u \in S$.

A (non-deterministic bottom-up) *tree automaton* is a quadruple $A = \langle Q, \Sigma, F, \Delta \rangle$ where Q is a finite set of states, $F \subseteq Q$ is a set of final states, and Δ is a set of state transition rules of the form $f(q_1, \ldots, q_n) \rightsquigarrow q$ with $f \in \Sigma^{(n)}$, $q_1, \ldots, q_n, q \in Q$. The set Δ can be viewed as a TRS where $\rightsquigarrow$ is used instead of $\to$. The arrow $\rightsquigarrow_\Delta$ is used for the rewrite relation over $\mathcal{T}_{\Sigma \cup Q}$ induced by the rules in Δ where the subscript Δ may often be omitted if no confusion arises. We write $\rightsquigarrow^*$ for the reflexive and transitive closure of $\rightsquigarrow$. The set $\mathcal{L}(A, q)$ for $q \in Q$ is defined by $\mathcal{L}(A, q) \equiv \{t \in \mathcal{T}_\Sigma \mid t \rightsquigarrow_\Delta^* q\}$. The set of terms *accepted* by A is defined by $\mathcal{L}(A) \equiv \bigcup_{q \in F} \mathcal{L}(A, q)$. A state q is said to be *reachable* if $\mathcal{L}(A, q)$ is not empty. A state q is called a *sink state* when, for every $f \in \Sigma^{(n)}$ $(n > 0)$ and $q', q_1, \ldots, q_n \in Q$ with $q_i = q$ for some i, $f(q_1, \ldots, q_n) \rightsquigarrow q \in \Delta$ holds but $f(q_1, \ldots, q_n) \rightsquigarrow q' \in \Delta$ does not hold with $q' \neq q$. When q is a sink state, it is easy to show that $t \in \mathcal{L}(A, q)$ implies $C[t] \in \mathcal{L}(A, q)$ for every context $C \in \mathcal{C}_\Sigma$

and every ground term $t \in \mathcal{T}_\Sigma$. Two tree automata A_1 and A_2 are said to be *equivalent* if $\mathcal{L}(A_1) = \mathcal{L}(A_2)$ holds.

3 Termination of sole combinatory calculus

A combinatory calculus is specified by certain kinds of combinators and reduction rules for each combinator. A reduction rule for a combinator Z has the form $Z\ x_1\ \ldots\ x_n \to e$ where e is built by combining $x_1, \ldots, x_n$ with function application. One of the most familiar combinatory calculi is given by the $\mathbf{S}$ and $\mathbf{K}$ combinators defined by $\mathbf{S}\ x_1\ x_2\ x_3 \to x_1\ x_3\ (x_2\ x_3)$ and $\mathbf{K}\ x_1\ x_2 \to x_1$. The calculus is well known to be Turing-complete in the sense that these two combinators are sufficient to represent all computable functions. Reductions in combinatory calculus are easily simulated by a TRS using the set of constant symbols for combinators and a binary function symbol @ for function application. For example, reduction rules for the $\mathbf{S}$ and $\mathbf{K}$ combinatory calculus can be represented by a TRS consisting of $@(@(@(\mathbf{S}, x_1), x_2), x_3) \to @(@(x_1, x_3), @(x_2, x_3))$ and $@(@(\mathbf{K}, x_1), x_2) \to x_1$, which is obviously non-terminating because of the Turing-completeness of the corresponding reduction system.

In this paper, we are interested in the question for sole combinatory calculus, which is built only by one combinator. We start with formal definitions of several notions on the sole combinatory calculus in terms of term rewriting. For Z is a combinator, we denote by Σ_Z a signature consisting of a constant Z and binary function symbol @. A *Z-term*, which is built only from Z in combinatory calculus, is represented by a term in $\mathcal{T}_\Sigma$. A *sole combinatory calculus* R_Z induced by a combinator Z is a TRS over $\mathcal{T}_{\Sigma_Z}$ where R_Z is a singleton set of a left-linear rule of the form $@(\ldots @(@(Z, x_1), x_2) \ldots, x_n) \to e$ with a term e built only from @ and variables $x_1, \ldots, x_n$. A combinator Z is said to be *terminating* if R_Z is terminating. It is known that $\mathbf{B}$ [6] and $\mathbf{J}$ [14] are terminating while $\mathbf{O}$ [9, 7, 4] and $\mathbf{S}$ [19] are not.

The following lemma allows us to consider only the case of weakly innermost normalizing instead of terminating since the rule of sole combinatory calculus is orthogonal.

Lemma 1 ([11, Theorem 11])**.** An orthogonal TRS R is terminating if and only if R is weakly innermost normalizing.

The readers might recall a similar result shown by Church [2] that an orthogonal non-erasing TRS R is terminating if and only if R is weakly normalizing. Since we are concerned with non-erasing combinatory calculus, this result may seem more convenient. However, in the context of the present work, we intend to employ the above lemma because the 'innermost' condition plays a crucial role in our method which will be detailed later.

4 Disproving termination of combinators by tree automata

Endrullis and Zantema have proposed a procedure for disproving weakly and strongly normalizing by finding tree automata that disprove termination of arbitrary forms of left-linear TRSs. This section first explains how to disprove termination using the search for tree automata, and shows that it is sufficient to find a restricted form of tree automata to disprove the termination of non-erasing combinators. Then, we present how to find such tree automata with a SAT solver. Finally, we show the non-termination of 8 combinators with our implementation of the method.

4.1 Disproving termination with tree automata

The idea of disproving termination of a TRS R by Endrullis and Zantema is to find a non-empty set of reducible ground terms (i.e., not in normal form), which is closed under the rules in R. It is easy to see that the existence of such a set implies that R is not weakly normalizing because any reduction from a term in the set is always infinite. Endrullis and Zantema considered the case where the set is recognizable by a tree automaton as defined below, though they did not give a name to it.

Definition 2. A *termination-disproving automaton (TDA)* for a left-linear TRS R is a tree automaton $A = \langle Q, \Sigma, F, \Delta \rangle$ such that (2-1) there exists a state $q \in F$ that is reachable, (2-2) $\mathcal{L}(A) \cap \mathsf{NF}(R) = \emptyset$, and (2-3) $t \in \mathcal{L}(A, q)$ implies $u \in \mathcal{L}(A, q)$ for all $q \in Q$ and $t, u \in \mathcal{T}_\Sigma$ with $t \to_R u$.

It is easy to see that the set $\mathcal{L}(A)$ for a TDA A for R can disprove weak normalizability of R: the conditions (2-1) and (2-2) allow us to choose a term not in normal form, and the conditions (2-2) and (2-3) force any reduction from the term to be infinite. The condition (2-3) is a bit strong in the sense that it requires every set $\mathcal{L}(A, q)$ with $q \in Q$ to be closed under reductions in R. The last condition might be relaxed to require the closure property only for final states in Q_F. However, we require it for all states in Q to make the SAT solver–based disproof search easier as done by Endrullis and Zantema.

As we will see later, it is sufficient to find a TDA with a sink state as the final state. The restricted form of tree automata is defined as follows.

Definition 3. A *termination-disproving automaton with a final sink (TDA-S)* for an orthogonal TRS R is a tree automaton $A = \langle Q, \Sigma, \{q_F\}, \Delta \rangle$ with $q_F \in Q$ such that (3-1) q_F is sink and reachable, (3-2) $\mathcal{L}(A) \cap \mathsf{NF}(R) = \emptyset$, and (3-3) $t \in \mathcal{L}(A, q)$ implies $u \in \mathcal{L}(A, q) \cup \mathcal{L}(A)$ for all $q \in Q$ and $t, u \in \mathcal{T}_\Sigma$ with $t \xrightarrow{\text{i}}_R u$.

We assume here that R is orthogonal, which is stronger than left-linear, because every sole combinatory calculus is orthogonal. A major difference from TDAs is that a TDA-S has a sink state q_F as the unique final state. The sink state is reachable so that $\mathcal{L}(A)$ is non-empty. In addition, it requires the closure property only under the innermost reduction and allows the reduction onto $\mathcal{L}(A)$. The latter relaxation corresponds to a minor improvement done by Endrullis and Zantema where the closure property allows the reduction onto terms accepted by the 'larger' states in the total order over states, which will be used for reachability checking with Lemma 6. The following theorem states that the existence of a TDA-S for an orthogonal TRS R implies that R is not terminating.

Theorem 4. An orthogonal TRS R is not terminating if a TDA-S for R exists.

Proof. Let $A = \langle Q, \Sigma, \{q_F\}, \Delta \rangle$ be a TDA-S for an orthogonal TRS R. By Lemma 1, it suffices to show that R is not weakly innermost normalizing. We prove by contradiction that for all $t \in \mathcal{L}(A)$, no reduction sequence from t reaches a normal form, which disproves the weak innermost normalizability of R due to the non-emptiness of $\mathcal{L}(A)$ implied by the (3-1) condition of A. Suppose that there exists a term such that a reduction sequence starting from the term reaches a normal form. Let $t \in \mathcal{L}(A)$ be such a term which has the shortest reduction sequence $t = t_0 \to^{\mathrm{I}}_R t_1 \to^{\mathrm{I}}_R \cdots \to^{\mathrm{I}}_R t_n$ with $t_n \in \mathsf{NF}(R)$. From the (3-2) condition, t is not in normal form, i.e., $n > 0$. From the (3-3) condition with $q = q_F$, $t_1 \in \mathcal{L}(A)$ holds. Since we have a reduction sequence from $t_1 \in \mathcal{L}(A)$ which reaches a normal form, it contradicts the assumption that the reduction sequence from t is the shortest one. $\square$

We will try to find a TDA-S (which has exactly one final state) to disprove the termination of a sole combinatory calculus instead of a TDA. Since the disproof search will be done by fixing the number of states and increasing it iteratively, we have to show that the number of states of a TDA-S is not required to be as large as that of a TDA. Note that we do not need to find a TDA-S equivalent to a TDA if it exists. It is well-known that every non-deterministic tree automaton can be converted into an equivalent one that has exactly one final state (by introducing a fresh final state) but it may have one more state than the original one. The following lemma guarantees that the number of states of a TDA-S is not required to be larger than that of a TDA to disprove the termination of an orthogonal TRS. The proof idea is to construct a TDA-S A from a given TDA A_0 by forcing a final state of A_0 to be the final sink state of A and removing states that accept only terms whose subterm is accepted by the sink state.

Lemma 5. Let R be an orthogonal TRS. If a TDA $A_0 = \langle Q_0, \Sigma, F_0, \Delta_0 \rangle$ for R exists, then a TDA-S $A = \langle Q, \Sigma, \{q_F\}, \Delta \rangle$ for R also exists with $|Q| \leq |Q_0|$.

Proof. Let $A_0 = \langle Q_0, \Sigma, F_0, \Delta_0 \rangle$ be a TDA for an orthogonal TRS R, where A_0 satisfies the conditions (2-1), (2-2), and (2-3). Without loss of generality, all states in Q_0 are reachable; otherwise, we could remove unreachable states from Q_0. From (2-1) of A_0, the set F_0 is not empty, hence we can choose a final state $q_F \in F_0$. Let $L_F \subseteq \mathcal{T}_\Sigma$ and $Q_F \subseteq Q_0$ be defined by $L_F = \{C[t] \mid C \in \mathcal{C}_\Sigma,\ t \in \mathcal{L}(A_0, q_F)\}$ and $Q_F = \{q \in Q_0 \mid \mathcal{L}(A_0, q) \subseteq L_F\}$, respectively. Note that $q_F \in Q_F$ holds in particular. Then we define a tree automaton $A = \langle Q, \Sigma, \{q_F\}, \Delta \rangle$

with $Q = (Q_0 \setminus Q_F) \cup \{q_F\}$ and $\Delta = \Delta_1 \cup \Delta_2$ where

$$\Delta_1 = \{f(q_1, \dots, q_n) \rightsquigarrow q \in \Delta_0 \mid \{q_1, \dots, q_n\} \subseteq Q_0 \setminus Q_F,\ q \in Q\} \text{ and}$$
$$\Delta_2 = \{f(q_1, \dots, q_n) \rightsquigarrow q_F \mid f \in \Sigma^{(n)},\ q_F \in \{q_1, \dots, q_n\} \subseteq Q\}$$

so that q_F is to be a sink final state in A. Since $|Q| \leq |Q_0|$ obviously holds, it suffices to show that A is a TDA-S for R.

Before proving that A is a TDA-S for R, we show

(5-1) $t \in L_F$ if and only if $t \in \mathcal{L}(A, q_F)$,
(5-2) $t \in \mathcal{L}(A, q)$ implies $t \in \mathcal{L}(A_0, q)$ for all $q \in Q_0 \setminus Q_F$, and
(5-3) $t \in \mathcal{L}(A_0, q)$ implies $t \in \mathcal{L}(A, q) \cup \mathcal{L}(A, q_F)$ for all $q \in Q_0 \setminus Q_F$,

for all $t \in \mathcal{T}_\Sigma$. These statements can be shown by simultaneous induction on the structure of t. Suppose that $t = f(t_1, \dots, t_n)$ with $f \in \Sigma^{(n)}$ and $t_1, \dots, t_n \in \mathcal{T}_\Sigma$. On the (5-1) statement, we examine two cases according to whether $t_i \in L_F$ or not for each i. In the case where $t_i \in L_F$ holds for some $1 \leq i \leq n$, the 'if'-statement of (5-1) is obvious from the definition of L_F, hence we show the 'only if'-statement. Since $t_i \in \mathcal{L}(A, q_F)$ holds from the induction hypothesis, we have $t \in \mathcal{L}(A, q_F)$ using a transition rule in Δ_2. Thus, the 'only if'-statement of (5-1) also holds. In the case where $t_i \notin L_F$ holds for all $1 \leq i \leq n$, we first show the 'if'-statement of (5-1). Assume $t \in \mathcal{L}(A, q_F)$ holds. Then, there exists $q_1, \dots, q_n \in Q$ such that $f(q_1, \dots, q_n) \rightsquigarrow q_F \in \Delta$ and $t_i \in \mathcal{L}(A, q_i)$ for every $1 \leq i \leq n$. If $q_i = q_F$ holds for some $1 \leq i \leq n$, then we have $t_i \in L_F$ from the induction hypothesis, hence $t \in L_F$. If $q_i \in Q_0 \setminus Q_F$ holds for all $1 \leq i \leq n$, then the transition rule is in Δ_1 and we have $t_i \in \mathcal{L}(A_0, q_i)$ from the induction hypothesis of (5-2) for all i. Using the same transition rule, we have $t \in \mathcal{L}(A_0, q_F)$, hence $t \in L_F$. Therefore, the 'if'-statement of (5-1) holds. For the 'only if'-statement of (5-1), assume $t \in L_F$. Since $t_i \notin L_F$ holds for all $1 \leq i \leq n$, we have $t \in \mathcal{L}(A_0, q_F)$ owing to the definition of L_F. Then, there exists $q_1, \dots, q_n \in Q_0$ such that $f(q_1, \dots, q_n) \rightsquigarrow q_F \in \Delta_0$ and $t_i \in \mathcal{L}(A_0, q_i)$ for every $1 \leq i \leq n$. Note that $t_i \notin L_F$ implies $q_i \in Q_0 \setminus Q_F$ by the definition of Q_F. Thus, the transition rule is in Δ_1 and we have $t_i \in \mathcal{L}(A, q_i) \cup \mathcal{L}(A, q_F)$ from the induction hypothesis of (5-3). When $t_i \in \mathcal{L}(A, q_i)$ holds for all i, we have $t \in \mathcal{L}(A, q_F)$ using the same transition rule. When $t_i \in \mathcal{L}(A, q_F)$ holds for some $1 \leq i \leq n$, we have $t \in \mathcal{L}(A, q_F)$ using the transition rule in Δ_2. Therefore, the 'only if'-statement of (5-1) holds.

On the (5-2) statement, assume $t \in \mathcal{L}(A, q)$ with $q \in Q_0 \setminus Q_F$, that is, $q \neq q_F$. Then, there exist $q_1, \dots, q_n \in Q$ such that $f(q_1, \dots, q_n) \rightsquigarrow q \in \Delta$ and $t_i \in \mathcal{L}(A, q_i)$ for every $1 \leq i \leq n$. Note that $q_i \neq q_F$ holds for all $1 \leq i \leq n$ because of the construction of Δ. Since we have $q_i \in Q_0 \setminus Q_F$ for every $1 \leq i \leq n$, the transition rule is in Δ_1 and $t_i \in \mathcal{L}(A_0, q_i)$ holds from the induction hypothesis for every $1 \leq i \leq n$. Using the same transition rule, we have $t \in \mathcal{L}(A_0, q)$. Therefore, (5-2) holds.

On the (5-3) statement, assume $t \in \mathcal{L}(A_0, q)$ with $q \in Q_0 \setminus Q_F$. Then, there exist $q_1, \dots, q_n \in Q_0$ such that $f(q_1, \dots, q_n) \rightsquigarrow q \in \Delta_0$ and $t_i \in \mathcal{L}(A_0, q_i)$ for every $1 \leq i \leq n$. When $q_i \in Q_F$ holds for some $1 \leq i \leq n$, we have $t_i \in L_F$ by the definition of Q_F, hence $t_i \in \mathcal{L}(A, q_F)$ holds from the induction hypothesis of (5-1). Using a transition rule in Δ_2, we have $t \in \mathcal{L}(A, q_F)$. When $q_i \in Q_0 \setminus Q_F$ holds for all $1 \leq i \leq n$, the transition rule is in Δ_0 and we have $t_i \in \mathcal{L}(A, q_i) \cup \mathcal{L}(A, q_F)$ from the induction hypothesis for all $1 \leq i \leq n$. If $t_i \in \mathcal{L}(A, q_i)$ holds for all i, then we have $t \in \mathcal{L}(A, q)$ using the same transition rule. If $t_i \in \mathcal{L}(A, q_F)$ holds for some i, then we have $t \in \mathcal{L}(A, q_F)$ using the transition rule in Δ_2. Therefore, (5-3) holds.

Now we are ready to show that A is a TDA-S for R. Concerning the (3-1) condition, the set L_F is not empty since q_F is reachable in A_0 due to the (2-1) condition of A_0. Then we have q_F is reachable also in A owing to (5-1). In addition, q_F is a sink state in A, hence the (3-1) condition holds for A.

The (3-2) condition is shown by contradiction. Suppose that there exists a term $t \in \mathcal{L}(A) \cap \mathsf{NF}(R)$. Then we have $t \in L_F$ from (5-1), hence $t = C[t_0]$ holds for some $C \in \mathcal{C}_\Sigma$ and $t_0 \in \mathcal{L}(A_0, q_F)$. From $t_0 \in \mathcal{L}(A_0, q_F) \subseteq \mathcal{L}(A_0)$ and the (2-2) condition of A_0, we have $t_0 \notin \mathsf{NF}(R)$, and thus $t = C[t_0] \notin \mathsf{NF}(R)$ holds. This contradicts the assumption, hence the (3-2) condition holds for A.

Concerning the (3-3) condition, assume that we have $t \in \mathcal{L}(A, q)$ with $q \in Q$ and $t \rightarrow^{\mathrm{I}}_R u$ for some $u \in \mathcal{T}_\Sigma$. In the case of $q = q_F$, we have $t \in L_F$ by (5-1), hence $t = C[t_0]$ holds for some $C \in \mathcal{C}_\Sigma$ and $t_0 \in \mathcal{L}(A_0, q_F)$. Since the t_0 is not in normal form by the (2-2) condition of A_0 and $\rightarrow^{\mathrm{I}}_R$ is an innermost relation, we have either $u = C[u_0]$ with $t_0 \rightarrow^{\mathrm{I}}_R u_0 \in \mathcal{T}_\Sigma$ or $u = D[t_0]$ with some $D \in \mathcal{C}_\Sigma$. In the former case, we have $u_0 \in \mathcal{L}(A_0, q_F)$ by $t_0 \in \mathcal{L}(A_0, q_F)$ and the (2-3) condition of A_0. Then $u_0 \in \mathcal{L}(A, q_F)$ holds due to $u_0 \in \mathcal{L}(A_0, q_F) \subseteq L_F$ and (5-1). Since q_F is a sink state in A, $u \in \mathcal{L}(A, q_F)$ holds. In the latter case, $u \in \mathcal{L}(A, q_F)$ holds because $t_0 \in \mathcal{L}(A_0, q_F) \subseteq L_F$ implies $t_0 \in \mathcal{L}(A, q_F)$ by (5-1) and q_F is a sink state in A. Therefore, the (3-3) condition holds for A in the case of $q = q_F$. In the case of $q \in Q_0 \setminus Q_F$, we have $t \in \mathcal{L}(A_0, q)$ by (5-2), hence $u \in \mathcal{L}(A_0, q)$ holds by the (2-3) condition of A_0. Since we have $u \in \mathcal{L}(A, q) \cup \mathcal{L}(A, q_F)$ by (5-3), the (3-3) condition holds for A in the case of $q \in Q_0 \setminus Q_F$. □

Since we try to find a TDA-S in the ascending order of the number of states as Endrullis and Zantema have done for a TDA, one of the TDA-Ss for a given TRS with the smallest number of states can be found if it exists. It might be possible to find a TDA-S even with a smaller number of states than a TDA because of the relaxed closure property (3-3). Our experiment results in Section 4.3 do not show such a case, though.

4.2 SAT encoding of termination-disproving tree automata

Endrullis and Zantema showed that the problem of finding TDAs can be reduced to the boolean satisfiability problem (SAT, for short) by fixing the number of states of tree automata. Although we essentially follow their method, we will present a method which can find a TDA-S more efficiently because of the restriction of its form. The efficiency of the disproof search is improved not only by finding a TDA-S instead of a TDA but also by specializing their method for non-erasing sole combinatory calculus.

We present our SAT encoding method to be self-contained, where the differences from the method by Endrullis and Zantema (*EZ method*, for short) are explicitly explained for each step. We first explain problem settings and definitions used in our method, introduce propositional variables and their meaning, and then show propositional formulas over them that must hold.

Problem setting and definitions Let Z be a non-erasing combinator whose termination is to be disproved. Recall that the reduction rule of Z is represented by a singleton TRS R_Z over $\Sigma_Z = \{Z^{(0)}, @^{(2)}\}$. Let $l_Z \to r_Z$ be the unique rule of R_Z where $l_Z = @(\ldots @(@(Z, x_1), x_2) \ldots, x_{M_Z})$ with some $M_Z \geq 1$ and r_Z is built from the binary function symbol $@$ and variables $x_1, x_2, \ldots, x_{M_Z}$ so that $\mathsf{FV}(l_Z) = \mathsf{FV}(r_Z)$ holds for R_Z to be non-erasing. We write U_Z for the set $\mathsf{Sub}(l_Z) \cup \mathsf{Sub}(r_Z)$. The *left depth* $\mathsf{ld}(t)$ of a Z-term $t \in \mathcal{T}_{\Sigma_Z}$ is defined by $\mathsf{ld}(Z) = 0$ and $\mathsf{ld}(@(t_1, t_2)) = 1 + \mathsf{ld}(t_1)$. The left depth of a term is useful in determining whether the term is redex or not. A Z-term t has the left depth M_Z if and only if t is a redex of R_Z.

Let $A = \langle Q, \Sigma, \{q_F\}, \Delta \rangle$ be a TDA-S with a final sink state $q_F \in Q$, which is to be found if it exists. We fix the number of states as $|Q| = N$ in our encoding. We iteratively ask the SAT solver to find a TDA-S increasing N one by one, starting with $N = M_Z + 1$ because no automaton with less than $M_Z + 1$ states can recognize the existence of a redex of R_Z. We use a function $\mathsf{mld}_A : Q \to \mathbb{N}$ defined by $\mathsf{mld}_A(q) = \min_{t \in \mathcal{L}(A,q)} \mathsf{ld}(t)$. The function is total because the TDA-S to be found has only reachable states.

Propositional variables Our SAT encoding involves three classes of propositional variables. The first one has the form of either $v_{@(q_1,q_2) \rightsquigarrow q}$ or $v_{Z \rightsquigarrow q}$ with $q_1, q_2, q \in Q$, which identify Δ. These variables are expected to satisfy

$v_{@(q_1,q_2) \rightsquigarrow q}$ is true iff $@(q_1, q_2) \rightsquigarrow q \in \Delta$, for all $q_1, q_2, q \in Q$, and

$v_{Z \rightsquigarrow q}$ is true iff $Z \rightsquigarrow q \in \Delta$, for all $q \in Q$.

This class of variables has been employed in the EZ method.

The second class of propositional variables has the form of either $v_{q,m}$ with $q \in Q \setminus \{q_F\}$ and $m < M_Z$ or $v_{q,\mathsf{rdx}}$ with $q \in Q$. They are expected to satisfy

$v_{q,m}$ is true iff $m = \mathsf{mld}_A(q)$, for all $q \in Q \setminus \{q_F\}$ and $m < M_Z$, and

$v_{q,\mathsf{rdx}}$ is true only if $\mathcal{L}(A, q) \cap \mathsf{NF}(R_Z) = \emptyset$, for all $q \in Q$.

This class of variables is newly introduced in our method in order to check the existence of redexes. Instead, the EZ method employs propositional variables that represent reachability of states of a tree automaton obtained by product construction of two tree automata, A and B, where B accepts all terms in normal form of R_Z.

The third class of propositional variables has the form of $v_{t,\alpha,q}$ with a term $t \in U_Z$, a substitution $\alpha : \mathcal{V}(t) \to Q \setminus \{q_F\}$, and a state $q \in Q$. Note that the number of possible substitutions α is finite because U_Z and Q are finite. Therefore the number of this class of variables is also finite. These variables are expected to satisfy

$v_{t,\alpha,q}$ is true iff $t\alpha \rightsquigarrow^* q$

for all $t \in U_Z$, $\alpha : \mathcal{V}(t) \to Q \setminus \{q_F\}$, and $q \in Q$. This class of variables has been employed in the EZ method, with the difference of the domain of substitutions. In their method, each substitution is defined over the set of all states including final states, while our encoding excludes the final state from the domain of substitutions. The difference makes the number of this class of variables much smaller, which will reduce the number of clauses passed to the SAT solver. We will explain later the reason why the final state can be left out in our encoding method.

Besides the three classes of propositional variables, our implementation of the proposed encoding method employs extra variables which is equivalent to conjunction of the other variables. They are introduced in order for the number of clauses to be smaller using a standard technique of Tseitin transformation [17], where sub-formulas are replaced by new propositional variables to avoid exponential brow-up of the number of clauses. The original method by Endrullis and Zantema may also have used the technique thought it is not explicitly mentioned in their article. Their implementation is currently no longer available, so we cannot be sure how they actually do it.

Propositional formulas Recall that a TDA-S $A = \langle Q, \Sigma_Z, \{q_F\}, \Delta\rangle$ with $|Q| = N$ is to be found by the SAT solver. The Boolean values of the propositional variables introduced so far exactly identify Δ. It must also be ensured, however, that there is no inconsistency in the valuation of propositional variables, and that A satisfies the TDA-S conditions. The propositional formulas to be satisfied will be shown in sequence condition by condition. It is assumed that they will eventually be combined together by conjunction and passed to the SAT solver to find an appropriate valuation.

Firstly, the final state q_F must be sink according to the (3-1) condition. Hence the following propositional formulas must hold:

$$\bigwedge_{q_F \in \{q_1,q_2\} \subseteq Q} \left(v_{@(q_1,q_2) \rightsquigarrow q_F} \wedge \bigwedge_{q \in Q \setminus \{q_F\}} \neg v_{@(q_1,q_2) \rightsquigarrow q} \right) \tag{1}$$

All of these propositional variables are immediately forced to be assigned to either true or false in the phase of unit propagation of the SAT solver. In addition, the (3-1) condition requires that q_F is reachable. Since we will find a TDA-S with the smallest number of states, we assume that all states in Q are reachable. The reachability of states can be paraphrased as the existence of a total order on Q with appropriate properties as considered in the EZ method.

We will employ the following lemma to specify the total order. The proof is similar to that of the EZ method except that the final state is fixed as a sink state in our setting. The details of the proof are found in the full version of this paper [10].

Lemma 6. Let $A = \langle Q, \Sigma_Z, \{q_F\}, \Delta\rangle$ be a tree automaton with a sink state q_F. Then, all states in Q are reachable if and only if there exists a total order $<$ on Q with maximal element q_F such that for every $q \in Q$ there exists either $Z \rightsquigarrow q \in \Delta$ or $@(q_1, q_2) \rightsquigarrow q \in \Delta$ with $q_1 < q$ and $q_2 < q$.

Lemma 6 makes it easy to find a TDA-S in which all states are reachable. Without loss of generality, we can fix an ordered sequence of all states in Q as $p_1 < p_2 < \cdots < p_N = q_F$ (recall $|Q| = N$) and force the states to satisfy the property in Lemma 6. The restriction is directly encoded by

$$\bigwedge_{q\in Q} \left(v_{Z\rightsquigarrow q} \vee \bigvee_{q_1,q_2<q} v_{@(q_1,q_2)\rightsquigarrow q} \right), \tag{2}$$

which has been employed also in the EZ method, although our encoding differs in that the ordered sequence of states should end with the unique final state.

Secondly, A must satisfy the (3-2) condition, which requires that A accepts no terms in normal form. Since $\mathcal{L}(A) = \mathcal{L}(A, q_F)$, a propositional formula

$$v_{q_F,\mathsf{rdx}} \tag{3}$$

(which is just a unit clause) should hold. To ensure that the Boolean value of this variable is valid, all of the propositional variables of the form $v_{q,m}$ or $v_{q,\mathsf{rdx}}$ should be properly assigned as follows. The propositional variables $v_{q,m}$ for $q \in Q \setminus \{q_F\}$ and $m < M_Z$ are expected to satisfy $m = \mathsf{mld}_A(q)$. It is easy to see that $\mathsf{mld}_A(q)$ can be effectively computed for each $q \in Q$ from the transition rules in Δ. The next lemma claims this fact as a logical statement so as to be used for giving appropriate propositional formulas. The proof is found in the full paper [10].

Lemma 7. Let $A = \langle Q, \Sigma_Z, \{q_F\}, \Delta\rangle$ be a tree automaton with a combinator Z where all states in Q are reachable. Then for every $q \in Q$ and $m \in \mathbb{N}$, (7-1) $\mathsf{mld}_A(q) = 0$ if and only if $Z \rightsquigarrow q \in \Delta$, and (7-2) $\mathsf{mld}_A(q) = m > 0$ if and only if there is neither $Z \rightsquigarrow q \in \Delta$ nor $@(q_1, q_2) \rightsquigarrow q \in \Delta$ with $\mathsf{mld}_A(q_1) < m-1$ and there exists $@(q_1, q_2) \rightsquigarrow q \in \Delta$ with $\mathsf{mld}_A(q_1) = m - 1$.

We only need the lemma for non-final states though it holds even for the final state q_F because the propositional variables $v_{q,m}$ are given only for $q \in Q \setminus \{q_F\}$. The statement (7-1) indicates that $v_{q,0}$ for each $q \in Q \setminus \{q_F\}$ has an appropriate Boolean value by the following propositional formula:

$$\bigwedge_{q\in Q\setminus\{q_F\}} \left(v_{q,0} \Leftrightarrow v_{Z\rightsquigarrow q} \right). \tag{4}$$

We could use the same propositional variable for $v_{q,0}$ and $v_{Z\rightsquigarrow q}$ in an efficient implementation, though. Additionally, the statement (7-2) indicates that the propositional variable $v_{q,m}$ with $m > 0$ has an appropriate Boolean value by the following propositional formula:

$$\bigwedge_{q\in Q\setminus\{q_F\}} \bigwedge_{m<M_Z} (v_{q,m} \Leftrightarrow (\neg v_{Z\rightsquigarrow q} \wedge P_1(q,m) \wedge P_2(q,m))) \tag{5}$$

where

$$P_1(q,m) = \bigwedge_{0\le k<m-1} \bigwedge_{q_1,q_2\in Q\setminus\{q_F\}} \left(v_{@(q_1,q_2)\rightsquigarrow q} \Rightarrow \neg v_{q_1,k}\right) \text{ and}$$

$$P_2(q,m) = \bigvee_{q_1,q_2\in Q\setminus\{q_F\}} \left(v_{@(q_1,q_2)\rightsquigarrow q} \wedge v_{q_1,m-1}\right).$$

The propositional variable $v_{q,\mathsf{rdx}}$ for each $q \in Q$ is expected to be true only if $\mathcal{L}(A,q)\cap\mathsf{NF}(R_Z) = \emptyset$. The next lemma is used for giving propositional formulas in order for the variables $v_{q,\mathsf{rdx}}$ to have an appropriate Boolean value.

Lemma 8. Let $A = \langle Q, \Sigma_Z, \{q_F\}, \Delta\rangle$ be a tree automaton with a sink state q_F, let R_Z be a TRS with a combinator Z, and let $\mathscr{Q} \subseteq Q$ be a set of states such that for all $q \in \mathscr{Q}$, (8-1) $Z \rightsquigarrow q \notin \Delta$ holds, and (8-2) $\mathsf{mld}_A(q_1) = M_Z - 1$ holds if there is $@(q_1,q_2) \rightsquigarrow q \in \Delta$ with $q_1, q_2 \in Q\setminus\mathscr{Q}$. Then, $\mathcal{L}(A,q)\cap\mathsf{NF}(R_Z) = \emptyset$ holds for every $q \in \mathscr{Q}$.

The proof is found in the full paper [10]. Let $\mathscr{Q} \subseteq Q$ be the set of states q such that $v_{q,\mathsf{rdx}}$ is true. Lemma 8 guarantees that $v_{q,\mathsf{rdx}}$ is true only if $\mathcal{L}(A,q) \cap \mathsf{NF}(R_Z) = \emptyset$ whenever $\mathscr{Q}$ satisfies the conditions (8-1) and (8-2), that is, the following propositional formulas should be true: for (8-1),

$$\bigwedge_{q\in Q} (v_{q,\mathsf{rdx}} \Rightarrow \neg v_{Z\rightsquigarrow q}); \tag{6}$$

for (8-2),

$$\bigwedge_{q\in Q} \bigwedge_{q_1,q_2\in Q\setminus\{q_F\}} \left(v_{q,\mathsf{rdx}} \wedge v_{@(q_1,q_2)\rightsquigarrow q} \wedge \neg v_{q_1,\mathsf{rdx}} \wedge \neg v_{q_2,\mathsf{rdx}} \Rightarrow v_{q_1,M_Z-1}\right). \tag{7}$$

Finally, we need to ensure that A is almost closed under R_Z in the sense of the (2-3) condition. As the EZ method does, we use the following lemma which gives a procedure to have the condition. While the EZ method employs the existing results [5, Proposition 12], we give a proof of this lemma in the full paper [10] because the (3-3) condition is different from the closure property.

Lemma 9. Let $A = \langle Q, \Sigma, \{q_F\}, \Delta\rangle$ be a tree automaton with a sink state q_F, and let R be a left-linear TRS over Σ. Then, $s \in \mathcal{L}(A,q)$ implies $t \in \mathcal{L}(A,q)\cup\mathcal{L}(A)$ for all $q \in Q$ and $s,t \in \mathcal{T}_\Sigma$ with $s \to^{\mathrm{I}}_R t$ if $l\alpha \rightsquigarrow^*_\Delta q$ implies $r\alpha \rightsquigarrow^*_\Delta q$ or $r\alpha \rightsquigarrow^*_\Delta q_F$ for all $q \in Q$, $l \to r \in R$ and $\alpha : \mathsf{FV}(l) \to Q$.

Recall that the propositional variable $v_{t,\alpha,q}$ is true when a term t is accepted with $q \in Q$ if every variable x in t is substituted with a term accepted with a state $\alpha(x)$. Following the EZ method, the procedure of computing the state of each subterm is simulated by imposing relations among the propositional variables. Unlike their method, we only need to consider the case where the range of α is $Q \setminus \{q_F\}$ since the closure property always holds if $\alpha(x) = q_F$ for some x because q_F is sink and R_Z is non-erasing. This arrangement would substantially reduce the number of propositional variables. For each subterm $t = @(t_1, t_2) \in U_Z$, we should have

$$\bigwedge_{q\in Q} \bigwedge_{\alpha:\mathsf{FV}(t)\to Q\setminus\{q_F\}} \left(v_{t,\alpha,q} \Leftrightarrow \bigvee_{q_1,q_2\in Q} \left(v_{t_1,\alpha[t_1],q_1} \wedge v_{t_2,\alpha[t_2],q_2} \wedge v_{@(q_1,q_2)\rightsquigarrow q} \right) \right) \qquad (8)$$

where $\alpha[t]$ stands for the substition α whose domain is restricted to $\mathsf{FV}(t)$. For each leaf in the subterms U_Z, we should have

$$\bigwedge_{q\in Q} \left(\left(v_{Z,\emptyset,q} \Leftrightarrow v_{Z\rightsquigarrow q} \right) \wedge \bigwedge_{x\in\mathsf{FV}(l_Z)} \left(v_{x,\{x\mapsto q\},q} \wedge \bigwedge_{q'\in Q\setminus\{q\}} \neg v_{x,\{x\mapsto q'\},q} \right) \right) \qquad (9)$$

where the former part indicates that we could use the same propositional variable for $v_{Z,\emptyset,q}$ and $v_{Z\rightsquigarrow q}$. And then, the (3-3) condition requires the propositional formula

$$\bigwedge_{\alpha:\mathsf{FV}(l_Z)\to Q} \left(v_{l_Z,\alpha,q} \Rightarrow v_{r_Z,\alpha,q} \vee v_{r_Z,\alpha,q_F} \right). \qquad (10)$$

4.3 Applying to non-termination of specific combinators

We have shown how to construct propositional logic formulas for the existence of TDA-S with fixed number of states for a sole combinatory calculus. We have implemented the construction and input the obtained formulas to a SAT solver to examine termination for combinators shown in Fig. 1, for which termination is unknown. We confirmed that our method can efficiently find a TDA-S compared to the method developed by Endrullis and Zantema for a TDA. Note that their implementation is not currently publicly available, so we have re-implemented it. For fairness, we have also applied the same optimization (e.g., Tseitin transformation) as in the implementation of our construction. Our implementation is written in OCaml and uses the Kissat SAT solver. We also implement the ability to output the smallest term accepted by the found TDA-S, thus allowing the output of a non-normalizable term.

The results of the examination for each combinator are shown in Table 1. Due to page limitation, the details of the TDA-S obtained are found in the full paper [10] with a found counterexample for termination. Here we only show the number of states of the TDA-S, the size of the propositional logic formula used (number of propositional variables and clauses), and the computation time (the sum of encoding and solving time) including the attempts to fewer

Table 1. Results of disproving termination of combinators given in Fig. 1.

	EZ method				Our method			
	#Q	#vars	#clauses	time (s)	#Q	#vars	#clauses	time (s)
$\mathbf{P}$	4	15,648	75,237	0.71	4	7,116	33,543	0.22
$\mathbf{P}_3$	6	161,544	793,351	11.5	6	96,058	469,233	5.6
$\mathcal{D}_1$	6	936,810	4,594,129	97.0	6	462,528	2,264,943	32.6
$\mathcal{D}_2$	6	936,810	4,594,129	96.6	6	462,528	2,264,943	32.9
$\mathbf{\Phi}$	7	1,975,302	9,741,173	260.6	7	1,099,527	5,416,581	108.3
$\mathbf{\Phi}_2$	–	—	—	—	9	55,695,683	276,052,931	39,268.4
$\mathbf{S}_1$	7	1,975,302	9,741,173	262.1	7	1,099,527	5,416,581	111.6
$\mathbf{S}_2$	6	745,002	3,655,825	81.6	6	379,278	1,857,693	30.3

states. The 'EZ method' column shows the results of (our re-implementation of) the original method by Endrullis and Zantema; the 'Our method' column shows the results of our construction method introduce in the present paper. The EZ method failed to find a TDA for $\mathbf{\Phi}_2$ within 24-hour computation. Note that the number of propositional variables and clauses is the same for different combinators. This is because they depend only on the number of states, the number of variables in the left-hand side of the combinator's rule and the number of subterms in the right-hand side. A major improvement in our method is a reduction of the number of variables of the form $v_{t,\alpha,q}$ by restricting the range of α, which is the most significant part of the SAT encoding. The number of possible substitutions is reduced from $|Q|^N$ to $(|Q|-1)^N$, where N is the number of variables in the left-hand side of the rule. Our method can indeed generate less propositional logic formulas than the original one, and also succeed in disproving the termination of more combinators. Both method failed to find the disproof of termination for the remaining combinators $\mathbf{S}_3$ and $\mathbf{S}_4$ in Fig. 1. We have confirmed that their termination cannot be disproven by a TDA-S with at most 9 states.

5 Concluding remark

We have proposed a method to disprove the termination of sole combinatory calculi with tree automata by extending the method proposed by Endrullis and Zantema. Specifically, we have shown that a tree automaton with a final sink state is sufficient to disprove termination of non-erasing combinators. We have succeeded in disproving the termination of 8 combinators which is unknown of their termination found in Smullyan's book. The remaining two combinators $\mathbf{S}_3$ and $\mathbf{S}_4$ still have unknown termination. We may need more improvement to disprove their termination. However, our method in which termination is disproved by a tree automaton with a final sink state may be applicable to other non-erasing term rewriting systems. It would be interesting to investigate how effectual our method is in more general settings.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments. This work was partially supported by JSPS KAKENHI Grant Numbers, JP21K11744, JP22H00520, and JP22K11904.

References

1. Barendregt, H.P.: The lambda calculus – its syntax and semantics, Studies in logic and the foundations of mathematics, vol. 103. North-Holland (1985)
2. Church, A.: The Calculi of Lambda Conversion. (AM-6) (Annals of Mathematics Studies). Princeton University Press, USA (1985)
3. Curry, H.B.: Grundlagen der kombinatorischen Logik. American Journal of Mathematics **52**(3), 509–536 (1930)
4. Endrullis, J., Zantema, H.: Proving non-termination by finite automata. In: 26th International Conference on Rewriting Techniques and Applications (RTA 2015). LIPIcs, vol. 36, pp. 160–176 (2015). https://doi.org/10.4230/LIPICS.RTA.2015.160
5. Genet, T.: Decidable approximations of sets of descendants and sets of normal forms. In: 9th International Conference on Rewriting Techniques and Applications (RTA 1998). LNCS, vol. 1379, pp. 151–165. Springer (1998). https://doi.org/10.1007/BFB0052368
6. Ikebuchi, M., Nakano, K.: On properties of B-terms. Log. Methods Comput. Sci. **16**(2) (2020). https://doi.org/10.23638/LMCS-16(2:8)2020
7. Iwami, M.: On the acyclic and related properties of combinators (in Japanese). IPSJ Trans. Prog. (PRO) **2**(2), 97–104 (Mar 2009)
8. Iwami, M.: Non-ω-strong head normalization, non-ground loop and acyclic of several combinators (in Japanese). IPSJ Trans. Prog. (PRO) **16**(3), 14–27 (Aug 2023)
9. Klop, J.W.: New fixed point combinators from old. In: Reflections on type theory, lambda-calculus, and the mind : Essays dedicated to Henk Barendregt on the occasion of his 60th birthday (2007), https://www.cs.ru.nl/barendregt60/essays/klop/
10. Nakano, K., Iwami, M.: Disproving Termination of Non-Erasing Sole Combinatory Calculus with Tree Automata (Full Version). CoRR **abs/2406.14305** (2024), https://arxiv.org/abs/2406.14305
11. O'Donnell, M.J.: Computing in Systems Described by Equations, LNCS, vol. 58. Springer (1977). https://doi.org/10.1007/3-540-08531-9
12. Ohlebusch, E.: Advanced topics in term rewriting. Springer New York, NY (2002), http://www.springer.com/computer/swe/book/978-0-387-95250-5
13. Peyton Jones, S.L.: The Implementation of Functional Programming Languages. Prentice-Hall (1987)
14. Probst, D., Studer, T.: How to normalize the Jay. Theor. Comput. Sci. **254**(1-2), 677–681 (2001). https://doi.org/10.1016/S0304-3975(00)00379-0
15. Schönfinkel, M.: Über die Bausteine der mathematischen Logik. Mathematische Annalen **92**(3), 305–316 (Sep 1924). https://doi.org/10.1007/BF01448013
16. Smullyan, R.: Diagonalization and Self-reference. Oxford logic guides, Clarendon Press (1994)
17. Tseitin, G.S.: Automation of Reasoning 2, chap. On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer Berlin Heidelberg (1983). https://doi.org/10.1007/978-3-642-81955-1_28
18. Turner, D.A.: A new implementation technique for applicative languages. Softw. Pract. Exp. **9**(1), 31–49 (1979). https://doi.org/10.1002/SPE.4380090105
19. Waldmann, J.: The Combinator S. Information and Computation **159**(1-2), 2–21 (2000), https://doi.org/10.1006/inco.2000.2874