JFP **27**, e6, 31 pages, 2017.
doi:10.1017/S0956796816000332

*Normal-order reduction grammars**

MACIEJ BENDKOWSKI

Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, ul. Prof. Łojasiewicza 6, 30-348 Kraków, Poland
(*e-mail:* maciej.bendkowski@tcs.uj.edu.pl)

Abstract

We present an algorithm which, for given n, generates an unambiguous regular tree grammar defining the set of combinatory logic terms, over the set $\{S, K\}$ of primitive combinators, requiring exactly n normal-order reduction steps to normalize. As a consequence of Curry and Feys's standardization theorem, our reduction grammars form a complete syntactic characterization of normalizing combinatory logic terms. Using them, we provide a recursive method of constructing ordinary generating functions counting the number of SK-combinators reducing in n normal-order reduction steps. Finally, we investigate the size of generated grammars giving a primitive recursive upper bound.

1 Introduction

Since the time of the pioneering works of Schönfinkel (1924) and Curry (1930), combinatory logic is known as a powerful, yet extremely simple in structure, formalism expressing the notion of computability. With the dawn of functional programming languages in the early 1970s, combinatory logic, with its standard normal-order reduction scheme (Curry & Feys, 1958), is used as a practical implementation of lazy semantics in languages such as SASL (Turner, 1979) or its successor Miranda (Turner, 1986). Lack of bound variables in the language resolves the intrinsic problem of substitution in λ-calculus, making the reduction relation a simple computational step and so, in consequence, the leading workhorse in implementing call-by-need reduction schemes.

Surprisingly, little is known about the combinatorial properties of normal-order reduction and, in particular, its behaviour in the 'typical' case of large random combinators. In this paper, we provide a complete combinatorial characterization of normalizing SK-combinators through regular tree grammars (see, e.g., Comon *et al.*, 2007). We construct a recursive family $(R_n)_{n\in\mathbb{N}}$ of grammars defining combinators reducing in exactly n normal-order reductions. By Curry and Feys's standardization theorem (Curry & Feys, 1958), normal-order evaluation of normalizing combinators leads to their normal forms; hence our *normal-order reduction grammars* form a

* This work was partially supported within the Polish National Science Center grant 2013/11/B/ST6/00975.

complete partition of normalizing combinators. Our approach is algorithmic in nature and provides fully automated methods for constructing $(R_n)_{n\in\mathbb{N}}$ as well as their corresponding ordinary generating functions.

The benefits of such a characterization are twofold. First, the effective computation of ordinary generating functions corresponding to combinators reducing in any fixed number of normal-order reduction steps makes quantitative properties of normal-order reduction, and hence also normalization in general, amenable to the powerful methods of analytic combinatorics (Flajolet & Sedgewick, 2009). In consequence, we obtain a promising novel approach to quantitative investigations relating computations with their syntactic realisation in combinatory logic.

Second, the effective computation of normal-order reduction grammars and their ordinary generating functions enables uniform random generation of SK-combinators with a given normal-order reduction length via Boltzmann samplers (Duchon *et al.*, 2004). With the growing popularity of random software testing (see, e.g. , Pałka *et al.*, 2011) random generation of large λ-terms and combinators with prescribed properties, such as typeability or normalization, became of both practical and theoretical importance. In this approach to software verification, large random terms are generated and used to check the programmer-declared function invariants, making it crucial to understand and exploit the semantic properties of so generated terms.

State-of-the-art research in this field includes counting and generating λ-terms (see, e.g. , Grygiel & Lescanne, 2013, 2015; Gittenberger & Gołębiewski, 2016), their restricted classes (Bodini *et al.*, 2015), investigating their asymptotic properties (David *et al.*, 2013; Bendkowski *et al.*, 2016) as well as the asymptotic properties of combinatory logic (Bendkowski *et al.*, 2015). Main tools used in this line of research include formal power series and generating functions. Interested in a particular counting sequence $(a_n)_{n\in\mathbb{N}}$ corresponding to a set of terms A, we construct a suitable generating function, which treated as a complex function in one variable z yields a Taylor series expansion around $z = 0$ with coefficients forming our sequence $(a_n)_{n\in\mathbb{N}}$. Methods of analytic combinatorics (Flajolet & Sedgewick, 2009) allow us to derive, sometimes surprisingly accurate, asymptotic approximations of the growth rate of $(a_n)_{n\in\mathbb{N}}$ and, in consequence, use them to study the asymptotic behaviour of A.

Finding appropriate generating functions plays therefore an important role in the process of investigating properties of 'typical' terms. In Bendkowski *et al.* (2015), authors investigated the asymptotic density of weakly normalizing terms in the set of all combinators, showing that a 'typical' combinator cannot have a trivial 0–1 asymptotic probability of normalization. The result was obtained by constructing large classes of terms with and without the normalization property. Though sufficient for the purpose of showing the non-trivial behaviour of normalization, their classes reveal the combinatorial structure of just a small asymptotic portion of normalizing terms. In this context, our results provide an answer as to why it is difficult to describe the combinatorial structure of large asymptotic portions of normalizing combinators.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

1.1 Outline

The paper is organized as follows. In Sections 1.2 and 1.3, we give preliminary definitions and notational conventions. In Section 1.4, we explain our pseudo-code notation and related implementation. In Section 2, we present a high-level overview on the algorithm. In Section 3, we analyse the algorithm giving proofs of soundness 3.2, completeness 3.3, and unambiguity 3.4. In Section 3.5, we give a recursive construction of ordinary generating functions corresponding to $(R_n)_{n\in\mathbb{N}}$. In Section 3.6, we discuss some consequences and applications of normal-order reduction grammars. Finally, in Section 3.7, we investigate the size of the generated grammars.

1.2 Combinatory logic

We consider the set of terms over primitive combinators S and K. In other words, the set $\mathscr{C}$ of combinatory logic terms defined as $\mathscr{C} := S \mid K \mid \mathscr{C}\,\mathscr{C}$. We follow standard notational conventions (see, e.g., Barendregt, 1984)—we omit outermost parentheses and drop parentheses from left-associated terms, e.g., instead of $((SK)(KK))$, we write $SK(KK)$. We use $\to_w$ to denote the normal-order reduction relation (reduce the leftmost outermost redex) to which we usually refer briefly as the reduction relation. We use lower case letters $x, y, z, \ldots$ to denote combinatory logic terms. And so, in this notation, the reduction rules for S and K as are given as follows:

$$Sxyz \to_w xz(yz)$$
$$Kxy \to_w x$$

For an introduction to combinatory logic, we refer the reader to Barendregt (1984) and Curry & Feys (1958).

1.3 Regular tree grammars

In order to characterize terms normalizing in n steps, we use regular tree grammars (see, e.g., Comon *et al.*, 2007), a generalization of regular word grammars. A *regular tree grammar* $G = (S, N, \mathscr{F}, P)$ consists of an axiom S, a set N of non-terminal symbols such that $S \in N$, a set of terminal symbols $\mathscr{F}$ with corresponding arities and a finite set of production rules P of the form $\alpha \to \beta$ where $\alpha \in N$ is a non-terminal and $\beta \in T_{\mathscr{F}}(N)$ is a term in the corresponding term algebra $T_{\mathscr{F}}(N)$, i.e., the set of directed trees built upon terminals $\mathscr{F}$ according to their associated arities. To build terms of grammar G, we start with the axiom S and use the corresponding derivation relation, denoted by $\to$, as defined through the set of production rules P.

Example 1
Consider the following regular tree grammar defined as $B = (S, N, \mathscr{F}, P)$ where $S := \mathscr{B}$, $N := \{\mathscr{B}\}$, $\mathscr{F} := \{\bullet, \circ(\cdot,\cdot)\}$, and P consists of the two following rules:

$$\mathscr{B} \to \circ(\mathscr{B}, \mathscr{B})$$
$$\mathscr{B} \to \bullet$$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Note that B defines the set of terms isomorphic to plane binary trees where leafs correspond to the nullary constant $\bullet$ and inner nodes correspond to the binary terminal $\circ(\cdot, \cdot)$.

In our endeavour, we are going to recursively construct regular tree grammars generating sets of combinatory logic terms. We describe their axioms, terminal and non-terminal symbols here, leaving the algorithm, given in the following section, to define the remaining production rules. And so, the nth grammar R_n will have:

i. an axiom $S = R_n$;
ii. a set $\mathscr{F}$ of terminal symbols consisting of two nullary constants S, K and a single binary application operator;
iii. a set of non-terminal symbols $N = \{\mathscr{C}\} \cup \{R_0, \dots, R_n\}$ where $\mathscr{C}$ denotes the axiom of the set of all combinatory logic terms, as defined in the previous section.

In other words, the grammar R_n defining terms normalizing in n steps, will reference all previous grammars $R_0, \dots, R_{n-1}$ and the set of all combinatory logic terms $\mathscr{C}$.

Throughout the paper, we adopt the following common definitions and notational conventions regarding trees. We use lower case letters $\alpha, \beta, \gamma, \delta, \dots$ to denote trees, i.e., elements of the term algebra $T_{\mathscr{F}}(N)$ where $N = \{\mathscr{C}\} \cup \{R_0, \dots, R_n\}$ for some n. Capital letters $X, Y, \dots$ are used to designate one of S or K without specifying which one. We define the *size* of α as the number of applications in α. We say that α is *normal* if either α is of size 0, or $\alpha = X\alpha_1 \dots \alpha_m$, for some $m \geqslant 1$, where all $\alpha_1, \dots, \alpha_m$ are normal. In the latter case, we say moreover that α is *complex*. Since we are going to work exclusively with normal trees, we assume that all trees are henceforth normal. We say that a complex α is of *length* m if α is in form of $X\alpha_1 \dots \alpha_m$. Otherwise, if α is not complex, we say that it is of length 0. The *degree* of α, denoted as $\rho(\alpha)$, is the minimum natural number n such that α does not contain references to any R_i for $i \geqslant n$. In particular, if α does not reference any reduction grammar, its degree is equal to 0. We use $L_G(\alpha)$ to denote the language of α in grammar G. Since R_n does not reference grammars of greater index, we have $L_{R_{\rho(\alpha)-1}}(\alpha) = L_{R_n}(\alpha)$ for arbitrary $n \geqslant \rho(\alpha)$. And so, for convenience, we use $L(\alpha)$ to denote the language of α in grammar $R_{\rho(\alpha)-1}$ if $\rho(\alpha) > 0$. Otherwise, if $\rho(\alpha) = 0$, we assume that $L(\alpha)$ denotes the language of α in grammar $\mathscr{C}$. Finally, we say that two normal trees are *similar* if both start with the same combinator X and are of equal length.

Example 2
Consider the following trees:

i. $\alpha = S(KR_1)\mathscr{C}$;
ii. $\beta = K(\mathscr{C}S)R_0$.

Note that both α and β are of size 3 and of equal length 2, although they are not similar since both start with different combinators. Moreover, only α is normal as β has a subtree $\mathscr{C}S$, which is of positive size, but does not start with a combinator.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Since α contains a reference to R_1 and no other reduction grammar, its degree is equal to 2, whereas the degree of β is equal to 1.

A crucial observation, which we are going to exploit in our construction, is the fact that normal trees *preserve* length of generated terms. In other words, if α is of length $m \geqslant 1$, then any term $x \in L(\alpha)$ is of length m as well, i.e., $x = Xx_1 \dots x_m$.

1.4 Pseudo-codes and implementation

We state our algorithm using functional pseudo-codes formalising key design subroutines. The adopted syntax echoes the basic Haskell notation and built-in primitives, though we use certain abbreviations making the overall presentation more comprehensible. And so, we use the following data structure representing normal trees.

```
-- | Normal trees.
data Tree = S | K | C | R Int
          | App Tree Tree
```

In our subroutines, we use the following 'syntactic sugar' abbreviating the structure of normal trees.

```
-- | Syntactic sugar.
X a_1 ... a_m := App X (App a_1 (... App a_{m-1} a_m) ...)
```

Moreover, we allow the use of this abbreviated notation in pattern matching, meaning that by writing `(X a_1 ... a_m)`, we expect a complex tree of length m for some $m \in \mathbb{N}$. If multiple arguments are supposed to share the same length, we use the same natural number m, e.g. `(X a_1 ... a_m)` and `(X b_1 ... b_m)`.

Finally, suppose we have a function `f :: a -> [b]` and wish to obtain the set-theoretic union of the image of `[a]` through `f`. In such a case, we use the following subroutine.

```
unionMap :: Eq b => (a -> [b]) -> [a] -> [b]
unionMap f xs = nub (concatMap f xs)
```

A working Haskell implementation of our algorithm is available at Bendkowski (2016).

2 Algorithm

The key idea used in the construction of reduction grammars is to generate new productions in R_{n+1} based on the productions in R_n. Necessarily, any term normalizing in $n+1$ steps reduces directly to a term normalizing in n steps; hence, their syntactic structure should be closely related. As the base of our inductive construction, we use the set of normal forms R_0 given by

$$R_0 := S \mid K \mid SR_0 \mid KR_0 \mid SR_0R_0$$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Clearly, primitive combinators S and K are in normal form. If we take a normal form x, then both Sx and Kx are again normal since we did not create any new redex. For the same reason, any term Sx_1x_2 where x_1 and x_2 are normal forms is itself in normal form. And so, with the above grammar, we have captured exactly all redex-free terms.

Let us consider productions of R_0. Note that from both the cases of SR_0 and KR_0, we can abstract a more general rule—if x reduces in n steps, then Sx and Kx reduce in n steps as well, since after reducing x we have no additional redexes left to consider. It follows that any R_n should contain productions SR_n and KR_n. Similarly, from the case of SR_0R_0, we can abstract a more general rule—if Sx_1x_2 reduces in n steps, then both x_1 and x_2 must reduce in total of n steps. The normal-order reduction of Sx_1x_2 proceeds to normalize x_1 and then x_2 in sequence. As there is no head redex, after n steps, we obtain a term in normal form. And so, R_n should also contain productions SR_iR_{n-i} for $i \in \{0, \ldots, n\}$.

As we have noticed, all the above productions do not contain head redexes and hence do not increase the total amount of required reduction steps to normalize. Formalizing the above observations, we say that α is *short* if either $\alpha = X\alpha_1$ or $\alpha = S\alpha_1\alpha_2$. Otherwise, α is said to be *long*. Hence, we can set *a priori* the short productions of R_n for $n \geqslant 1$ and continue to construct the remaining long productions. Naturally, as we consider terms over two primitive combinators S and K, we distinguish two types of long productions, i.e. S- and K-Expansions.

2.1 K-Expansions

Let us consider a production $\alpha = X\alpha_1 \ldots \alpha_m$ where $m \geqslant 0$. The set K-Expansions(α) is defined as

$$\left\{K(X\alpha_1 \ldots \alpha_k)\mathscr{C}\alpha_{k+1} \ldots \alpha_m \mid k \in \{0, \ldots, m-1\}\right\}$$

Proposition 3

Let $x \in L(K(X\alpha_1 \ldots \alpha_k)\mathscr{C}\alpha_{k+1} \ldots \alpha_m)$. If $x \to_w y$, then $y \in L(X\alpha_1 \ldots \alpha_m)$.

Proof

Let $x = K(Xx_1 \ldots x_k)zx_{k+1} \ldots x_m$. Let us consider its direct reduct $y = Xx_1 \ldots x_kx_{k+1} \ldots x_m$. Clearly, $x_i \in L(\alpha_i)$ for $i \in \{1, \ldots, m\}$ which finishes the proof. □

In other words, the set K-Expansions(α) has the property that any K-Expansion of α generates terms that reduce in one step to terms generated by α. If we compute the sets K-Expansions(α) for all productions $\alpha \in R_n$, we have almost constructed all of the long K-productions in R_{n+1}. What remains is to include the production $KR_n\mathscr{C}$ as any term $x \in L(KR_n\mathscr{C})$ reduces directly to $y \in L(\alpha)$ for some production $\alpha \in R_n$.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

We use the following subroutine computing the set of K-Expansions of a given production.

```
-- | Returns K-Expansions of the given production.
kExpansions :: Tree -> [Tree]
kExpansions p = case p of
    (K a_1 ... a_m) -> kExpansions' K [a_1,...,a_m]
    (S a_1 ... a_m) -> kExpansions' S [a_1,...,a_m]
    where
        kExpansions' _ [] = []
        kExpansions' h [x_1,...,x_k] = K h C x_1 ... x_k
            : kExpansions' (App h x_1) [x_2,...,x_k]
```

2.2 S-Expansions

Let us consider a production $\alpha = X\alpha_1 \dots \alpha_m$ where $m \geqslant 0$. We would like to define the set S-Expansions(α) similarly to K-Expansions(α), i.e., in such a way that any term generated by an S-Expansion of α reduces in a single step to some $y \in L(\alpha)$. Unfortunately, defining and computing such a set is significantly more complex than the corresponding K-Expansions(α).

Let $q = Xx_1 \dots x_k z(yz)$. Suppose that $q \in L(\alpha)$ for some production $\alpha \in R_n$. Since $S(Xx_1 \dots x_k)yz \to_w q$, we would like to guarantee that $S(Xx_1 \dots x_k)yz \in L(\beta)$ for some $\beta \in$ S-Expansions(α). Assume that $\alpha = X\alpha_1 \dots \alpha_k \gamma\delta$ where $z \in L(\gamma)$ and $yz \in L(\delta)$. Note that although $z \in L(\gamma)$ and $yz \in L(\delta)$, it might happen that $\delta \neq \delta'\gamma$ for any δ'. Indeed, take $\delta = \mathscr{C}$ and $\gamma = KR_n$. Let $z = Kz_1$ for an arbitrary $z_1 \in L(R_n)$ whereas $y = S$. We have $z = Kz_1 \in L(KR_n)$ and $yz = y(Kz_1) \in L(\mathscr{C})$; however, γ is not a suffix of δ.

The above example highlights the main obstacle in finding S-Expansions(α)—given γ, δ such that $z \in L(\gamma)$ and $yz \in L(\delta)$ we cannot decompose δ into $\delta'\gamma$ and set $\beta = S(X\alpha_1 \dots \alpha_k)\delta'\gamma$ as the resulting S-Expansions(α). Such a set of S-Expansions might generate a language containing additional terms, not reducing in a single step to some $y \in L(\alpha)$. Instead, we have to be more subtle and find $\eta\zeta$ such that $L(\zeta) \subseteq L(\gamma)$ and $L(\eta\zeta) \subseteq L(\delta)$. Using them instead of $\gamma\delta$ we construct a, potentially huge, set $\mathscr{T}$ of trees in form of $S(X\alpha_1 \dots \alpha_k)\eta\zeta$ constituting S-Expansions(α). Fortunately, it is possible to find a finite set consisting of $\eta\zeta$ such that $L(\mathscr{T})$ captures the indented set of S-Expansions and nothing more.

In order to find such a set, we require an additional 'rewriting' operation on trees that allows us their specialisation, i.e., narrowing the languages they generate. Let us consider the following extension $\rhd$ of the standard derivation relation $\to$:

$$\alpha \rhd \beta \Leftrightarrow \alpha \to \beta \vee (\alpha = \mathscr{C} \wedge \exists_{n \in \mathbb{N}}\ \beta = R_n)$$

In other words, $\alpha \rhd \beta$ if β can be derived from α, e.g., β is a production of α, or α is a tree representing the set of all combinatory logic terms, whereas β represents the set of combinatory logic terms reducing in n steps.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Let $\trianglerighteq$ denote the transitive-reflexive closure of $\triangleright$. If $\alpha \trianglerighteq \beta$, we say that α *rewrites to* β. The important property of $\trianglerighteq$ is the fact that if $\alpha \trianglerighteq \beta$, then $L(\beta) \subseteq L(\alpha)$. Since $\trianglerighteq$ is not a total order on trees, there exist trees incomparable through $\trianglerighteq$, e.g., S and K. To denote the fact that α does not rewrite to β and vice versa, we use the symbol $\alpha \parallel \beta$. In such a case we say that α and β are *non-rewritable*. Otherwise, if one of them rewrites to the other ($\alpha \trianglerighteq \beta$ or $\beta \trianglerighteq \alpha$), meaning that α and β are *rewritable*, we use the symbol $\alpha \bowtie \beta$.

Equipped with the rewriting operation $\trianglerighteq$, we have obtained a tool to specialise trees. Now, in order to find appropriate $\eta\zeta$ such that $L(\zeta) \subseteq L(\gamma)$ and $L(\eta\zeta) \subseteq L(\delta)$, it suffices to focus on the *rewriting set* $\textsc{RewritingSet}(\gamma, \delta)$ of γ and δ, consisting of $\eta\zeta$ such that $\gamma \trianglerighteq \eta$ and $\delta \trianglerighteq \eta\zeta$. In our algorithm, we will guarantee that the constructed rewriting set allows us to find the intended unambiguous set of S-Expansions.

The task of computing $\textsc{RewritingSet}(\gamma, \delta)$ is a fairly straightforward case analysis of δ's structure, except for when $\delta = X\delta_1 \dots \delta_m$ and $\gamma \parallel \delta_m$. In order to continue our computations, we have to find a set of trees τ such that both $\gamma \trianglerighteq \tau$ and $\delta_m \trianglerighteq \tau$. In order words, the set $\textsc{MeshSet}(\gamma, \delta_m)$ of all trees (also called *meshes*) τ such that both γ and δ_m rewrite to. In our algorithm, we construct the $\textsc{MeshSet}(\gamma, \delta_m)$ in such a way, that it generates the whole joint sublanguage of both γ and δ_m. Once found $\textsc{MeshSet}(\gamma, \delta_m)$ allows us to finish the construction of $\textsc{RewritingSet}(\gamma, \delta)$ and hence also the desired set of S-Expansions.

And so, finding S-Expansions(α) for some $\alpha \in R_{n+1}$ depends on computing appropriate mesh sets that, in turn, might depend on the previously computed grammars $R_0, \dots, R_n$. For that reason, we start with defining the MeshSet operation. Then, we give the RewritingSet operation, based on the definition of MeshSet. Finally, using RewritingSet, we present S-Expansions, which together with K-Expansions constitutes the defining algorithm of reduction grammars $(R_n)_{n \in \mathbb{N}}$.

2.2.1 Mesh set

In the endeavour of finding appropriate S-Expansions rewritings, we need to find common *meshes* of given non-rewritable trees $\alpha \parallel \beta$. In other words, a complete partition of $L(\alpha) \cap L(\beta)$ using all possible trees γ such that $\alpha, \beta \trianglerighteq \gamma$. For this purpose, we use the following pseudo-code subroutines.

```
1  -- | Given X α1 ... αm and X β1 ... βm computes
2  -- the family {γ1, ..., γm} of tree meshes.
3  mesh :: [Tree] -> [Tree] -> [[Tree]]
4  mesh (x : xs) (y : ys)
5      | x `rew` y = [y] : mesh xs ys -- case when x ▷ y
6      | y `rew` x = [x] : mesh xs ys -- case when y ▷ x
7      | otherwise = meshSet x y : mesh xs ys -- case when x ∥ y
8  mesh [] [] = []
```

The function Mesh, when given two similar productions $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$, constructs a family $\{\gamma_i\}_{i=1}^m$ where each γ_i depends on the comparison

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

of corresponding arguments. In the case when x rewrites to y (denoted as `x `rew` y` in the pseudo-code), the singleton $\{y\}$ is constructed. Similarly, when $y \trianglerighteq x$, the singleton $\{x\}$ is constructed. Otherwise, when x and y are both non-rewritable, γ_i is computed using the MeshSet subroutine.

```
-- | Returns the mesh set of given trees.
meshSet :: Tree -> Tree -> [Tree]
meshSet (X a_1 ... a_m) (X b_1 ... b_m) =
        cartesian X [mesh a_i b_i | i <- [1..m]]
meshSet (R k) b @ (X b_1 ... b_m) =
        unionMap (\p -> meshSet p b) (productions $ R k)
meshSet b @ (X b_1 ... b_m) (R k) =
        unionMap (\p -> meshSet b p) (productions $ R k)
meshSet _ _ = []
```

When given two similar trees $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$, MeshSet computes meshes $\gamma_1, \dots, \gamma_m$ of corresponding arguments α_i and β_i using the subroutine Mesh. Next, argument meshes $\{\gamma_i\}_{i=1}^m$ are used to construct meshes for α and β, using the subroutine Cartesian that computes the Cartesian product $\{X\} \times \gamma_1 \times \cdots \times \gamma_m$ using term application. In the case when one of MeshSet's argument is a reduction grammar R_k and the other α is complex, MeshSet computes recursively mesh sets of α and each production $\delta \in R_k$, outputting their set-theoretic union. In any other case, MeshSet returns the empty set.

Example 4

Let $\alpha = K\mathscr{C}R_0S$ and $\beta = KS(SR_0\mathscr{C})S$. Consider $\textsc{MeshSet}(\alpha, \beta)$. Both α and β are similar and complex; hence, MeshSet proceeds directly to construct mesh sets of corresponding arguments of α and β. Since $\mathscr{C} \trianglerighteq S$, we get $\gamma_1 = \{S\}$. Then, as both R_0 and $SR_0\mathscr{C}$ are non-rewritable, $\gamma_2 = \textsc{MeshSet}(R_0, SR_0\mathscr{C})$. It follows that $\textsc{MeshSet}(R_0, SR_0\mathscr{C})$ is equal to $\bigcup_{\delta \in R_0} \textsc{MeshSet}(\delta, SR_0\mathscr{C})$. Further inspection reveals that $\textsc{MeshSet}(R_0, SR_0\mathscr{C}) = \{SR_0R_0\}$ and thus $\gamma_2 = \{SR_0R_0\}$. Finally, $\gamma_3 = \{S\}$ as S rewrites trivially to itself. Since each γ_i is a singleton, it follows that

$$\textsc{MeshSet}(\alpha, \beta) = \{KS(SR_0R_0)S\}$$

We leave the analysis of MeshSet until we fully define the construction of reduction grammars $(R_n)_{n \in \mathbb{N}}$.

2.2.2 Rewriting set

Consider again our previous example of $q = Xx_1 \dots x_k z(yz) \in L(\alpha)$ where $\alpha = X\alpha_1 \dots \alpha_k \gamma\delta$ such that both $z \in L(\gamma)$ and $yz \in L(\delta)$. In order to capture terms reducing to α via an S-redex, we need to find all pairs of trees η, ζ such that $\gamma \trianglerighteq \zeta$ and $\delta \trianglerighteq \eta\,\zeta$. Since such pairs of trees follow exactly the structure of $z(yz)$, we can

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

use them to define the set S-Expansions(α). And so, to find such rewriting pairs, we use the following RewritingSet pseudo-code subroutine.

```
1  -- | Given α and β computes their rewriting set.
2  rewritingSet :: Tree -> Tree -> [Tree]
3  rewritingSet a S = []
4  rewritingSet a K = []
5  rewritingSet a C = [C a]
6  rewritingSet a (R k) = unionMap
7          (\p -> rewritingSet a p) (productions $ R k)
8  rewritingSet a (X b_1 ... b_m)
9          | a `rew` b_m => [X b_1 ... b_m]
10         | b_m `rew` a => [X b_1 ... b_{m-1} a]
11         | otherwise =>
12                 cartesian (X b_1 ... b_{m-1}) [meshSet a b_m]
```

The outcome of RewritingSet(α, β) depends on β's structure. If β is a primitive combinator S or K, RewritingSet returns the empty set. If $\beta = \mathscr{C}$, a singleton $\{\mathscr{C}\alpha\}$ is returned. When $\beta = R_k$ for some $k \in \mathbb{N}$, RewritingSet computes recursively the rewriting sets of α and $\gamma \in R_k$, outputting their set-theoretic union. Otherwise when $\beta = X\beta_1 \dots \beta_m$, RewritingSet determines whether $\alpha \rhd\!\lhd \beta_m$. If $\alpha \trianglerighteq \beta_m$, a singleton $\{X\beta_1, \dots, \beta_m\}$ is returned. Conversely, in the case of $\beta_m \trianglerighteq \alpha$, RewritingSet returns $\{X\beta_1, \dots, \beta_{m-1}\alpha\}$. Finally, if α and β_m are non-rewritable, RewritingSet invokes the Cartesian subroutine computing the Cartesian product of $\{X\beta_1, \dots, \beta_{m-1}\} \times$ MeshSet(α, β_m) using term application, passing afterwards its result as the computed rewriting set.

Example 5

Let us consider the rewriting set RewritingSet(S, R_0). Since $\beta = R_0$, we know that RewritingSet(S, R_0) $= \bigcup_{\gamma \in R_0}$ RewritingSet(S, γ). It follows therefore that in order to compute RewritingSet(S, R_0), we have to consider rewriting sets involving productions of R_0. Note that both productions S and K do not contribute new trees. It remains to consider productions SR_0, KR_0 and SR_0R_0. Evidently, each of them is complex and has R_0 as its final argument. Hence, their corresponding rewriting sets are SS, KS and SR_0S, respectively. And so, we obtain that

$$\textsc{RewritingSet}(S, R_0) = \{SS, KS, SR_0S\}$$

Similarly to the case of MeshSet, we postpone the analysis until we define the construction of $(R_n)_{n \in \mathbb{N}}$.

Equipped with the notion of mesh and rewriting sets, we are ready to define the set of S-Expansions. And so, let $\alpha = X\alpha_1 \dots \alpha_m$ where $m \geqslant 0$. The set S-Expansions(α) is defined as

$$\Big\{ S(X\alpha_1 \dots \alpha_k)\varphi_l\varphi_r\alpha_{k+3} \dots \alpha_m \mid k \in \{0, \dots, m-2\} \Big\}$$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

where $(\varphi_l \varphi_r) \in \textsc{RewritingSet}(\alpha_{k+1}, \alpha_{k+2})$. We use the following subroutine computing the set of S-Expansions for a given α.

```
-- | Returns S-Expansions of the given production.
sExpansions :: Tree -> [Tree]
sExpansions p = case p of
    (K a_1 ... a_m) -> sExpansions' K [a_1,...,a_m]
    (S a_1 ... a_m) -> sExpansions' S [a_1,...,a_m]
    where
        sExpansions' _ [] = []
        sExpansions' _ [_] = []
        sExpansions' h [x_1,x_2,...,x_k] =
              map (\(App l r) -> S h l r x_3 ... x_m)
              (rewritingSet x_1 x_2) ++
                   sExpansions' (App h x_1) [x_2,...,x_m]
```

Proposition 6
Let $x \in L(S(X\alpha_1 \dots \alpha_k)\varphi_l\varphi_r\alpha_{k+3} \dots \alpha_m)$. If $x \to_w y$, then $y \in L(X\alpha_1 \dots \alpha_k \varphi_r(\varphi_l\, \varphi_r)\, \alpha_{k+3} \dots \alpha_m)$.

Proof
Let $x = S(Xx_1 \dots x_k)wzx_{k+3} \dots x_m$. Consider its direct reduct $y = Xx_1 \dots x_k z(w\, z)\, x_{k+3} \dots x_m$. Clearly, $x_i \in L(\alpha_i)$ for i in proper range. Moreover, both $w \in L(\varphi_l)$ and $z \in L(\varphi_r)$, which finishes the proof. □

2.3 Algorithm pseudo-code

With the complete and formal definitions of both S- and K-Expansions, we are ready to give the main algorithm Reduction Grammar, which for given $n \in \mathbb{N}$ constructs the grammar R_n.

```
-- | Given n∈ℕ constructs R_n.
reductionGrammar :: Integer -> [Tree]
reductionGrammar 0 = [S, K, S (R 0), K (R 0), S (R 0) (R 0)]
reductionGrammar n = [S (R n), K (R n)]
        ++ [S (R $ n-i) R_i | i <- [0..n]]
        ++ [K (R $ n-1) C]
        ++ concatMap kExpansions (reductionGrammar $ n-1)
        ++ concatMap sExpansions (reductionGrammar $ n-1)
```

The grammar R_0 is given explicitly. Each next grammar R_n consists of five groups of productions where the first two define all the short ones and the last three define the long ones:

1. $\{\ SR_n, KR_n\ \}$;
2. $\{\ SR_{n-i}R_i \mid i = 0 \dots n\}$;
3. $\{\ KR_{n-i}C \mid i = 0 \dots n\ \}$;

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

4. $\bigcup\{$ K-Expansions(α) $\mid \alpha \in R_{n-1} \}$;
5. $\bigcup\{$ S-Expansions(α) $\mid \alpha \in R_{n-1} \}$.

Example 7
Following the construction algorithm, the grammar R_1, defining the set of SK-combinators reducing in a single step to their normal forms, is defined as follows:

$$R_1 = SR_1 \mid KR_1 \mid SR_0R_1 \mid SR_1R_0 \mid KR_0C \mid KSCR_0 \mid KKCR_0 \mid$$
$$KSCR_0R_0 \mid K(SR_0)CR_0 \mid SSSR_0 \mid SSKR_0 \mid SS(SR_0)R_0$$

Let us consider $\alpha = SSSR_0$. Since $\alpha \in$ S-Expansions(SR_0R_0), we get $\alpha \in R_1$. Note that S-Expansions(α) contains $\beta_1 = S(SS)SS$ and $\beta_2 = S(SS)KS$. It follows that $\beta_1, \beta_2 \in R_2$.

3 Analysis

3.1 Tree potential

Most of our proofs in the following sections are using inductive reasoning on the underlying tree structure. Unfortunately, in certain cases most natural candidates for induction such as tree size fail due to *self-referencing productions*, i.e., productions of R_n that explicitly use the non-terminal symbol R_n. In order to remedy such problems, we introduce the notion of *tree potential* $\pi(\alpha)$, defined inductively as

$$\pi(S) = \pi(K) = \pi(\mathscr{C}) = 0$$

$$\pi(X\alpha_1 \dots \alpha_m) = m + \sum_{i=1}^{m} \pi(\alpha_i)$$

$$\pi(R_n) = 1 + \max_{\gamma \in \Phi(R_n)} \pi(\gamma)$$

where $\Phi(R_n)$ denotes the set of productions of R_n that do not use the non-terminal symbol R_n. Such a statement of $\pi(R_n)$ is well defined due to the fact that R_0 is *finite* and the algorithm constructing R_{n+1} out of $R_0, \dots, R_n$ does not use unbounded recursion (see Proposition 8).

Moreover, note that such a definition of potential is almost identical to the notion of tree size. The potential of α is the sum of α's size and the weighted sum of all non-terminal grammar symbols occurring in α. Immediately from the definition, we get $\pi(R_0) = 1$. Moreover, $\pi(R_{n+1}) > \pi(R_n)$ for any $n \in \mathbb{N}$. Indeed, let $\alpha \in R_n$ be the witness of R_n's potential. Now, $(K\alpha\mathscr{C}) \in \Phi(R_{n+1})$ and so R_{n+1} has necessarily greater potential. Moreover, $\pi(\alpha) > \pi(\beta)$ if β is a subtree of α. It follows that the notion of tree potential is a good candidate for the intuitive tree complexity measure.

3.2 Soundness

In this section, we are interested in the soundness of Reduction Grammar. In particular, we prove that it is computable, terminates on all legal inputs and, for

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

given n, constructs a reduction grammar R_n generating only terms that require exactly n steps to normalize.

Let us start with showing that the rewriting relation is decidable.

Proposition 8
It is decidable to check whether $\alpha \trianglerighteq \beta$.

Proof
Induction over $n = \pi(\alpha) + \pi(\beta)$. If $\alpha = X$, then the only tree α rewrites to is X. On the other hand, if $\alpha = \mathscr{C}$, then α rewrites to any β. And so, it is decidable to check whether $\alpha \trianglerighteq \beta$ in case $n = 0$. Now, let us assume that $n > 0$. We have two remaining cases to consider.

i. If $\alpha = X\alpha_1 \dots \alpha_m$, then $\alpha \trianglerighteq \beta$ if and only if $\beta = X\beta_1 \dots \beta_m$ and $\alpha_i \trianglerighteq \beta_i$ for all $i \in \{1, \dots, m\}$. Since the total potential of $\pi(\alpha_i) + \pi(\beta_i)$ is less than n, we can use the induction hypothesis to decide whether all arguments of α rewrite to the respective arguments of β. It follows that we can decide whether $\alpha \trianglerighteq \beta$.
ii. If $\alpha = R_k$, then clearly $\alpha \trianglerighteq \beta$ if and only if $\beta = R_k$ or there exists a production $\gamma \in R_k$ such that $\gamma \trianglerighteq \beta$. Let us assume that γ is a production of R_k. Note that if $\gamma \trianglerighteq \beta$, then γ and β are similar. And so, since similarity is decidable, we can rephrase our previous observation as $\alpha \trianglerighteq \beta$ if and only if $\beta = R_k$ or there exists a production $\gamma \in R_k$ such that γ is similar to β and $\gamma \trianglerighteq \beta$. Checking whether $\beta = R_k$ is trivial, so let us assume the other option and start with the case when γ is a short production referencing R_k.
If $\gamma = XR_k$ is similar to $\beta = X\beta_1$, we know that $\gamma \trianglerighteq \beta$ if and only if $R_k \trianglerighteq \beta_1$. Since $\pi(R_k) + \pi(\beta_1) < n$, we know that checking whether $R_k \trianglerighteq \beta_1$ is decidable; hence, so is $\gamma \trianglerighteq \beta$.
Let us assume w.l.o.g. that $\gamma = SR_kR_0$. Hence, $\beta = S\beta_1\beta_2$. And so, $\gamma \trianglerighteq \beta$ if and only if $R_k \trianglerighteq \beta_1$ and $R_0 \trianglerighteq \beta_2$. Notice that $\pi(R_k) + \pi(\beta_1) < n$ as well as $\pi(R_0) + \pi(\beta_2) < n$. Using the induction hypothesis to both, we get that checking $R_k \trianglerighteq \beta_1$ and $R_0 \trianglerighteq \beta_2$ is decidable; hence, so is $\alpha \trianglerighteq \beta$.
Finally, if γ is a long production, we can rewrite it as $\gamma = X\gamma_1 \dots \gamma_m$, and so reduce this case to the previous one when both trees are complex, as $\pi(\gamma)$ is necessarily smaller than n.

□

Proposition 9
Let α, β be two trees. Then, both $\alpha \trianglerighteq \gamma$ and $\beta \trianglerighteq \gamma$ for any $\gamma \in \textsc{MeshSet}(\alpha, \beta)$.

Proof
Induction over $n = \pi(\alpha) + \pi(\beta)$. Let $M = \textsc{MeshSet}(\alpha, \beta)$. Clearly, it suffices to consider such α, β that $M \neq \emptyset$.

Let us assume that both $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$. If $\alpha_i \bowtie \beta_i$ for all $i \in \{1, \dots, m\}$, then M consists of a single tree $\gamma = X\gamma_1 \dots \gamma_m$ for which $\alpha_i, \beta_i \trianglerighteq \gamma_i$. Evidently, our claim holds. Suppose that there exists $i \in \{1, \dots, m\}$ such that $\alpha_i \parallel \beta_i$. Since $\pi(\alpha_i) + \pi(\beta_i) < n$, we can apply the induction hypothesis to $\textsc{MeshSet}(\alpha_i, \beta_i)$. The set $M' = \textsc{MeshSet}(\alpha_i, \beta_i)$ cannot be empty, and so let δ_i be an arbitrary mesh in

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

M'. We know that $\alpha_i, \beta_i \trianglerighteq \delta_i$. And so, if we consider an arbitrary $\gamma = X\gamma_i \dots \gamma_m \in M$, we get $\alpha_i, \beta_i \trianglerighteq \gamma_i$ for all $i \in \{1, \dots, m\}$, which implies our claim.

What remains is to consider the case when either $\alpha = R_k$ and β is complex or, symmetrically, $\beta = R_k$ and α is complex. Let us assume w.l.o.g. the former case. From the definition, $\textsc{MeshSet}(R_k, \beta)$ depends on the union of $\textsc{MeshSet}(\gamma, \beta)$ for $\gamma \in R_k$. Clearly, R_k rewrites to any of its productions. Let $\gamma \in R_k$ be a production referencing R_k. We have to consider two cases based on the structure of γ.

i. Let $\gamma = XR_k$. Then, $\pi(\gamma) = \pi(R_k) + 1$ and so we cannot use the induction hypothesis to $\textsc{MeshSet}(\gamma, \beta)$ directly. Note, however, that we can assume that $\beta = X\beta_1$, since otherwise $\textsc{MeshSet}(\gamma, \beta)$ would be empty. Therefore, we know that $\textsc{MeshSet}(R_k, \beta_1) \neq \emptyset$ to which we can now use the induction hypothesis, as $\pi(R_k) + \pi(\beta_1) < n$. Immediately, we get that $R_k, \beta \trianglerighteq \gamma$.
ii. W.l.o.g. let $\gamma = SR_kR_0$. Then, $\pi(\gamma) = 3 + \pi(R_k)$. Again, we cannot directly use the induction hypothesis. Note, however, that we can assume that $\beta = S\beta_1\beta_2$. And so we get $\pi(R_k) + \pi(\beta_1) < n$ and $\pi(R_0) + \pi(\beta_2) < n$. Using the induction hypothesis to both parts, we conclude that $R_k, \beta \trianglerighteq \gamma$ in this case as well.

To finish the proof, we need to show that our claim holds for all $\gamma \in R_k$ that do not reference R_k. Indeed, any such production has necessarily smaller potential than R_k, and so, we can use the induction hypothesis directly to the resulting mesh set. Evidently, our claim holds. □

In other words, $\textsc{MeshSet}(\alpha, \beta)$ is in fact a set of *meshes*, i.e., trees generating a joint portion of $L(\alpha)$ and $L(\beta)$. Note that along the lines of proving the above proposition, we have also showed that indeed $\textsc{MeshSet}(\alpha, \beta)$ terminates on all legal inputs, as the number of recursive calls cannot exceed $2(\pi(\alpha) + \pi(\beta))$—in the worst case, every second recursive call decreases the total potential sum of its inputs.

Proposition 10
Let α, β be two trees. Then, $\alpha \trianglerighteq \varphi_r$ and $\beta \trianglerighteq \varphi_l\varphi_r$ for any $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha, \beta)$.

Proof
We can assume that $\textsc{RewritingSet}(\alpha, \beta) \neq \emptyset$, as otherwise our claim trivially holds. Let $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha, \beta)$. Based on the structure of β, we have to three cases to consider.

i. If $\beta = \mathscr{C}$, then $\varphi_l\varphi_r = \mathscr{C}\alpha$. Immediately, $\alpha \trianglerighteq \alpha$ and $\mathscr{C} \trianglerighteq \mathscr{C}\,\alpha$.
ii. If $\beta = X\beta_1 \dots \beta_m$, then we have again exactly three possibilities. Both cases when $\alpha \bowtie \beta_m$ are trivial, so let us assume that $\alpha \parallel \beta_m$. It follows that there exists such a $\gamma \in \textsc{MeshSet}(\alpha, \beta_m)$ that $\varphi_l\varphi_r = X\beta_1 \dots \beta_{m-1}\gamma$. Due to Proposition 9, we know that $\alpha, \beta_m \trianglerighteq \gamma$ and so directly that $\alpha \trianglerighteq \varphi_r$ and $\beta \trianglerighteq \varphi_l\varphi_r$.
iii. Finally, suppose that $\beta = R_n$. Then, there exists a production $\gamma \in R_n$ such that $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha, \gamma)$. Note, however, that in this case $\gamma = X\gamma_1 \dots \gamma_m$, and so we can reduce this case to the already considered case above.

□

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Now we are ready to give the anticipated soundness theorem.

Theorem 11 (*Soundness*)
If $x \in L(R_n)$, then x reduces in n steps.

Proof
Induction over pairs (n, m) where m denotes the length of a minimal, in terms of length, derivation Σ of $x \in L(R_n)$. Let $n = 0$ and so $x \in L(R_0)$. If $m = 1$, then $x \in \{S, K\}$; hence, x is already in normal form. Suppose that $m > 1$. Clearly, $x \notin \{S, K\}$. Let $R_0 \to \alpha$ be the first production rule used in derivation Σ. Using the induction hypothesis to the reminder of the derivation, we know that x does not contain any nested redexes. Moreover, α avoids any head redexes and so we get that x is in normal form.

Let $n > 0$. We have to consider several cases based on the choice of the first production rule $R_n \to \alpha$ used in the derivation Σ.

i. $\alpha = SR_n$ or $\alpha = KR_n$. Using the induction hypothesis, we know that $x = Xy$ where y reduces in n steps. Naturally, so does x.
ii. $\alpha = SR_{n-i}R_i$ for some $i \in \{0, \ldots, n\}$. Then, $x = Syz$ where $y \in L(R_{n-i})$ and $z \in L(R_i)$. Note that both their derivations are in fact shorter than the derivation of x and thus applying the induction hypothesis to both y and z we know that they reduce in $n - i$ and i steps, respectively. Following the normal-order reduction strategy, we note that y and z reduce sequentially in x. Since x does not contain a head redex itself, we reduce it in total of n reductions.
iii. $\alpha = KR_{n-1}\mathscr{C}$. Directly from the induction hypothesis, we know that $x = Kyz$ where y reduces in $n - 1$ steps. And so $x \to_w y$, implying that x reduces in n steps.
iv. $\alpha = K(X\alpha_1 \ldots \alpha_k)\mathscr{C}\alpha_{k+1} \ldots \alpha_m$. Let $x \in L(\alpha)$. Clearly, x has a head redex and so let $x \to_w y$. Using Proposition 3, we know that $y \in L(X\alpha_1 \ldots \alpha_m)$. Moreover, by the construction of R_n, we get $\alpha \in$ K-Expansions$(X\alpha_1 \ldots \alpha_m)$ and therefore $y \in L(R_{n-1})$. It follows that y reduces in $n - 1$ steps and so x in n steps.
v. $\alpha = S(X\alpha_1 \ldots \alpha_k)\varphi_l\varphi_r\alpha_{k+3} \ldots \alpha_m$. Let $x \in L(\alpha)$. Clearly, x has a head redex and so let $x \to_w y$. Due to Proposition 6, we get that $y \in L(X\alpha_1 \ldots \alpha_k\varphi_r(\varphi_l\,\varphi_r)\,\alpha_{k+3} \ldots \alpha_m)$. In order to show that x reduces in n steps, it suffices to show that $y \in L(R_{n-1})$. Let us consider β such that $\alpha \in$ S-Expansions(β). From the structure of α, we can rewrite it as $\beta = X\alpha_1 \ldots \alpha_k\alpha_{k+1}\alpha_{k+2} \ldots \alpha_m$. Moreover, from Proposition 10 we know that $\alpha_{k+1} \trianglerighteq \varphi_r$ and $\alpha_{k+2} \trianglerighteq \varphi_l\,\varphi_r$. Clearly, $y \in L(\beta)$, which finishes the proof.

□

Combining the above result with the fact that each normalizing combinatory logic term reduces in a determined number of normal-order reduction steps, gives us the following corollary.

Corollary 12
If $L(R_n) \cap L(R_m) \neq \emptyset$, then $n = m$.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

3.3 Completeness

In this section, we are interested in the completeness of Reduction Grammar. In other words, we show that every term normalizing in exactly n steps is generated by R_n.

W start with some auxiliary lemmas showing the completeness of MeshSet and, in consequence, RewritingSet.

Lemma 13
Let α, β be two non-rewritable trees. Let x be a term. Then, $x \in L(\alpha) \cap L(\beta)$ if and only if there exists a mesh $\gamma \in \textsc{MeshSet}(\alpha, \beta)$ such that $x \in L(\gamma)$.

Proof
It suffices to show the sufficiency part, the necessity is clear from Proposition 9. We show this result using induction over the size $|x|$ of x. Let $x \in L(\alpha) \cap L(\beta)$. Let us start with noticing that $|\alpha| + |\beta| > 0$. Moreover, there are only two cases where $x \in L(\alpha) \cap L(\beta)$, i.e., when either $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$ or when exactly one of them is equal to some R_n and the other is complex. And so, let us consider these cases separately.

i. Suppose that $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$. It follows that we can rewrite x as $Xx_1 \dots x_m$ such that $x_i \in L(\alpha_i) \cap L(\beta_i)$. Clearly, if all $\alpha_i \bowtie \beta_i$, then there exists a mesh γ such that $x \in L(\gamma)$. Let us assume that some α_i and β_i are non-rewritable. Then, using the induction hypothesis, we find a mesh $\gamma_i \in \textsc{MeshSet}(\alpha_i, \beta_i)$ such that $x_i \in L(\gamma_i)$. Immediately, we get that there exists a mesh in $\textsc{MeshSet}(\alpha, \beta)$ that generates x.
ii. Let us assume w.l.o.g. that $\alpha = R_n$ and $\beta = X\beta_1 \dots \beta_m$. Since $x \in L(R_n)$, there must be such a production $\gamma \in R_n$ that $x \in L(\gamma)$. Although the size of x does not decrease, note that we can reduce this case to the one considered above since both γ and β are complex. Clearly, it follows that we can find a suiting mesh $\delta \in \textsc{MeshSet}(\gamma, \beta)$ such that $x \in L(\delta)$. Immediately, we get $\delta \in \textsc{MeshSet}(\alpha, \beta)$ that finishes the proof.

□

Lemma 14
Let α, β be two trees. Let x, yx be two terms. Then, $x \in L(\alpha)$ and $yx \in L(\beta)$ if and only if there exists such a $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha, \beta)$ that $x \in L(\varphi_r)$ and $yx \in L(\varphi_l\varphi_r)$.

Proof
Due to Proposition 10 the necessity is clear. What remains is to show the sufficiency part. Let $x \in L(\alpha)$ and $yx \in L(\beta)$. Consider the structure of β. If $\beta = \mathscr{C}$, then $\mathscr{C}\alpha \in \textsc{RewritingSet}(\alpha, \beta)$ and so $\varphi_l = \mathscr{C}, \varphi_r = \alpha$. Clearly, our claim holds. Now, consider the case when $\beta = X\beta_1 \dots \beta_m$. Based on the rewritability of α and β_m, we distinguish three subcases.

i. If $\alpha \trianglerighteq \beta_m$, then $X\beta_1 \dots \beta_m \in \textsc{RewritingSet}(\alpha, \beta)$. Since $yx \in L(\beta)$, we get $x \in L(\beta_m)$ and in consequence $x \in L(\varphi_r)$.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

ii. If $\beta_m \trianglerighteq \alpha$, then $X\beta_1 \dots \beta_{m-1}\alpha \in \textsc{RewritingSet}(\alpha, \beta)$. Since $\beta_m \trianglerighteq \alpha$, we know that $L(\alpha) \subseteq L(\beta_m)$ and so $yx \in L(X\beta_1 \dots \beta_{m-1}\alpha)$.
iii. If $\alpha \parallel \beta_m$, then we know that $x \in L(\alpha) \cap L(\beta_m)$. If not, then yx could not be a term of $L(\beta)$. And so, using Lemma 13 we find a mesh $\gamma \in \textsc{MeshSet}(\alpha, \beta_m)$ such that $x \in L(\gamma)$. We know that $X\beta_1 \dots \beta_{m-1}\gamma \in \textsc{RewritingSet}(\alpha, \beta)$. Clearly, it is the tree we were looking for.

It remains to consider the case when $\beta = R_k$. Note however that it can be reduced to the case when $\beta = X\beta_1 \dots \beta_m$. Indeed, since $x \in L(R_k)$, then there exists a production $\gamma \in R_k$ such that $x \in L(\gamma)$. From the previous arguments, we know that we can find a tree satisfying our claim. □

Using the above completeness results for MeshSet and RewritingSet, we are ready to give the anticipated completeness result of $(R_n)_{n\in\mathbb{N}}$.

Theorem 15 (*Completeness*)
If x reduces in n steps, then $x \in L(R_n)$.

Proof
Induction over pairs (n, s) where s denotes the size of x. The base case $n = 0$ is clear due to the completeness of R_0. Let $n > 0$.

Let us start with considering short terms. Let $x = Xy$ be a term of size s. Since x has no head redex, y must reduce in n steps as well. Now, we can apply the induction hypothesis to y and deduce that $y \in L(R_n)$. It follows that $x \in L(XR_n)$. Clearly, XR_n is a production of R_n and so $x \in L(R_n)$. Now, assume that $x = Syz$. Since x reduces in n steps and does not contain a head redex, there exists such $i \in \{0, \dots, n\}$ that y reduces in i steps and z reduces in $n - i$ steps. Applying the induction hypothesis to both y and z, we get that $y \in L(R_i)$, whereas $z \in L(R_{n-i})$. Immediately, we get that $x \in L(R_n)$ as $SR_iR_{n-i} \in R_n$.

What remains is to consider long terms. Let $x = Kx_1x_2$. Note that x_1 must reduce in $n - 1$ steps, as $x \to_w x_1$. And so, from the induction hypothesis, we get that $x_1 \in L(R_{n-1})$. Now we have $x \in L(KR_{n-1}\mathscr{C})$ and hence $x \in L(R_n)$ as $KR_{n-1}\mathscr{C}$ is a production of R_n.

Now, let $x = Kx_1 \dots x_m$ for $m \geqslant 3$. Since x has a head redex, we know that $x \to_w y = x_1x_3 \dots x_m$, which itself reduces in $n - 1$ steps. Let us rewrite y as $Xy_1 \dots y_kx_3 \dots x_m$ where $x_1 = Xy_1 \dots y_k$. We know that there exists a production $\alpha \in R_{n-1}$ such that $y \in L(\alpha)$. Let $\alpha = X\overline{\alpha_1} \dots \overline{\alpha_k}\alpha_3 \dots \alpha_m$. Clearly, there exists $\beta = K(X\overline{\alpha_1} \dots \overline{\alpha_k})\mathscr{C}\alpha_3 \dots \alpha_m \in \textsc{K-Expansions}(\alpha)$. We claim that $x \in L(\beta)$. Indeed, $y \in L(\alpha)$ implies that $y_i \in L(\overline{\alpha_i})$ and $x_j \in L(\alpha_j)$ for any i and j in proper ranges. Since $x_2 \in L(\mathscr{C})$, we conclude that $x \in L(\beta)$ and hence $x \in L(R_n)$.

Let $x = Sx_1 \dots x_m$ for $m \geqslant 3$. Since x has a head redex $x \to_w y = x_1x_3(x_2x_3)x_4 \dots x_m$ that reduces in $n - 1$ steps. Again, let us rewrite y as $Xy_1 \dots y_kx_3(x_2x_3)x_4 \dots x_m$ where $x_1 = Xy_1 \dots y_k$. Now, since $y \in L(R_{n-1})$, there exists a production $\alpha = X\overline{\alpha_1} \dots \overline{\alpha_k}\alpha_3\gamma\alpha_4 \dots \alpha_m \in R_{n-1}$ such that $y \in L(\alpha)$. We claim that there must be a production $\beta \in \text{S-}\textsc{Expansions}(\alpha)$ such that $x \in L(R_n)$. If so, the proof would be complete. Notice that $x_3 \in L(\alpha_3)$ and $x_2x_3 \in L(\gamma)$. Using Lemma 14, we know that there exists a tree $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha_3, \gamma)$ such that $x_3 \in L(\varphi_r)$ and $(x_2x_3) \in$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

$L(\varphi_l\varphi_r)$. And so $y \in L(X\overline{\alpha_1}\dots\overline{\alpha_k}\varphi_r(\varphi_l\varphi_r)\alpha_4\dots\alpha_m)$. Moreover, due to the fact that $\varphi_l\varphi_r \in \textsc{RewritingSet}(\alpha_3, \gamma)$, we know that the tree $\beta = S(X\overline{\alpha_1}\dots\overline{\alpha_k})\varphi_l\varphi_r\alpha_4\dots\alpha_m \in \text{S-}\textsc{Expansions}(\alpha)$ and so also $\beta \in R_n$. Since $x_2 \in L(\varphi_l)$, we get that $x \in L(\beta)$. □

3.4 Unambiguity

In this section, we show that reduction grammars are in fact *unambiguous*, i.e., every term $x \in L(R_n)$ has exactly one derivation. Due to the mutual recursive nature of MeshSet, RewritingSet and ReductionGrammar, we split the proof into two separate parts. In the following lemma, we show that MeshSet returns unambiguous meshes under the assumption that $R_0, \dots, R_n$ up to some n are themselves unambiguous. In the corresponding theorem, we use inductive reasoning that supplies the aforementioned assumption and thus, as a consequence, allows us to prove the main result.

Lemma 16
Let α, β be two trees such that $\gamma, \overline{\gamma} \in \textsc{MeshSet}(\alpha, \beta)$ where in addition $\rho(\alpha), \rho(\beta) \leqslant r + 1$. If $R_0, \dots, R_r$ are unambiguous and $L(\gamma) \cap L(\overline{\gamma}) \neq \emptyset$, then $\gamma = \overline{\gamma}$.

Proof
Induction over $n = \pi(\alpha) + \pi(\beta)$. Let $x \in L(\gamma) \cap L(\overline{\gamma})$. We can assume that $|\textsc{MeshSet}(\alpha, \beta)|$ is greater than 1 as the case for $|\textsc{MeshSet}(\alpha, \beta)| = 1$ is trivial. In consequence, the base case $n = 0$ is clear as the resulting MeshSet for two trees of potential 0 has to be necessarily empty. Hence, we have to consider two cases based on the structure of α and β.

i. Let $\alpha = X\alpha_1\dots\alpha_m$ and $\beta = X\beta_1\dots\beta_m$. Clearly, x is in form of $x = Xx_1\dots x_m$. Let $\alpha_i \parallel \beta_i$ be an arbitrary non-rewritable pair of arguments in α, β. It follows that $x_i \in L(\alpha_i) \cap L(\beta_i)$ and so, due to Lemma 13, there exists a mesh $\delta \in \textsc{MeshSet}(\alpha_i, \beta_i)$ such that $x_i \in L(\delta)$. Let $M_i = \textsc{MeshSet}(\alpha_i, \beta_i)$. Since $\pi(\alpha_i) + \pi(\beta_i) < n$, we can use the induction hypothesis to M_i and immediately conclude that δ is the only mesh in M_i generating x_i. And so, we know that γ and $\overline{\gamma}$ are equal on the non-rewritable arguments of α, β. Note that if $\alpha_i \bowtie \beta_i$, then both contribute a single mesh at position i. Immediately, we get that both γ and $\overline{\gamma}$ are also equal on the rewritable arguments of α and β; hence, finally $\gamma = \overline{\gamma}$.
ii. W.l.o.g. let $\alpha = R_k$ and $\beta = X\beta_1\dots\beta_m$. Clearly, as $\rho(\alpha) \leqslant r + 1$, we know that R_k is unambiguous. From the definition of MeshSet, there exist productions $\delta, \overline{\delta} \in R_k$ such that $\gamma \in \textsc{MeshSet}(\delta, \beta)$ and $\overline{\gamma} \in \textsc{MeshSet}(\overline{\delta}, \beta)$. We claim that $\gamma = \overline{\gamma}$ as otherwise $\delta, \overline{\delta}$ would generate a common term. Suppose that $\gamma \neq \overline{\gamma}$. From Lemma 13, we know that $L(\gamma) \subseteq L(\delta)$ and $L(\overline{\gamma}) \subseteq L(\overline{\delta})$. Since $x \in L(\gamma) \cap L(\overline{\gamma})$, we get that $x \in L(\delta) \cap L(\overline{\delta})$ and therefore a contradiction with the fact that R_k is unambiguous. It follows that $\gamma = \overline{\gamma}$, which finishes the proof.

□

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Theorem 17 (*Unambiguity*)
Let $\alpha, \beta \in R_n$. If $L(\alpha) \cap L(\beta) \neq \emptyset$, then $\alpha = \beta$.

Proof
Induction over n. Let $x \in L(\alpha) \cap L(\beta)$. Note that if $x \in L(\alpha) \cap L(\beta)$, then both α, β must be similar. We can therefore focus on similar productions of R_n. For that reason, we immediately notice that R_0 satisfies our claim.

Let $n > 0$. Since R_n does not contain combinators as productions, we can rewrite both α as $X\alpha_1 \ldots \alpha_m$ and β as $X\beta_1 \ldots \beta_m$. Let us consider several cases based on their common structure.

i. Let $X = K$. If $m = 1$, then α and β are equal as there is exactly one short K-production in R_n. If $m = 2$, then again $\alpha = \beta$, since there is a unique K-production $KR_{n-1}\mathscr{C}$ of length two in R_n. If $m > 2$, then both are K-Expansions of some productions in R_{n-1}. And so

$$\alpha = K(X\overline{\alpha_1} \ldots \overline{\alpha_k})\mathscr{C}\alpha_3 \ldots \alpha_m \in \text{K-Expansions}(\gamma)$$

$$\beta = K(X\overline{\beta_1} \ldots \overline{\beta_k})\mathscr{C}\beta_3 \ldots \beta_m \in \text{K-Expansions}(\delta)$$

where

$$\gamma = X\overline{\alpha_1} \ldots \overline{\alpha_k}\alpha_3 \ldots \alpha_m$$

$$\delta = X\overline{\beta_1} \ldots \overline{\beta_k}\beta_3 \ldots \beta_m$$

Since $x \in L(\alpha) \cap L(\beta)$, we can assume that x is in form of $K(Xy_1 \ldots y_k)x_2x_3 \ldots x_m$ where $y_i \in L(\overline{\alpha_i}) \cap L(\overline{\beta_i})$ and $x_j \in L(\alpha_j) \cap L(\beta_j)$. It follows that we can use the induction hypothesis to $\gamma, \delta \in R_{n-1}$ obtaining $\overline{\alpha_i} = \overline{\beta_i}$ and $\alpha_j = \beta_j$. Immediately, we get $\alpha = \beta$.

ii. Let $X = S$. If $m = 1$, then α and β are equal due to the fact that there is exactly one S-production of length one in R_n. If $m = 2$, then α, β are in form of $\alpha = SR_iR_{n-i}$ and $\beta = SR_jR_{n-j}$. Hence, $x = Sx_1x_2$ for some terms x_1, x_2. Since $x_1 \in L(R_i) \cap L(R_j)$ and $x_2 \in L(R_{n-i}) \cap L(R_{n-j})$, we know that $i = j$ due to Corollary 12 and thus $\alpha = \beta$. It remains to consider long S-productions. Let

$$\alpha = S(X\overline{\alpha_1} \ldots \overline{\alpha_k})\varphi_l\varphi_r\alpha_4 \ldots \alpha_m \in \text{S-Expansions}(\gamma)$$

$$\beta = S(X\overline{\beta_1} \ldots \overline{\beta_k})\overline{\varphi_l\varphi_r}\beta_4 \ldots \beta_m \in \text{S-Expansions}(\delta)$$

where

$$\gamma = X\overline{\alpha_1} \ldots \overline{\alpha_k}\alpha_2\alpha_3\alpha_4 \ldots \alpha_m$$

$$\delta = X\overline{\beta_1} \ldots \overline{\beta_k}\beta_2\beta_3\beta_4 \ldots \beta_m$$

It follows that we can rewrite x as $S(Xy_1 \ldots y_k)wzx_4 \ldots x_m$. Let us focus on the reduct $x \to_w y = Xy_1 \ldots y_kz(wz)x_4 \ldots x_m$. Evidently, $y \in L(\gamma) \cap L(\delta)$ and so according to the induction hypothesis, we know that $\gamma = \delta$, in particular,

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

$\alpha_2 = \beta_2$ and $\alpha_3 = \beta_3$. Hence, both $\varphi_l\varphi_r$ and $\overline{\varphi_l\varphi_r}$ are elements of the same RewritingSet. If we could guarantee that $\varphi_l\varphi_r = \overline{\varphi_l\varphi_r}$, then immediately $\alpha = \beta$ and the proof is finished. From the construction of the RewritingSet, we have two cases left to consider.

i. If $\alpha_3 = X\gamma_1 \dots \gamma_m$, then both $\varphi_l\varphi_r$ and $\overline{\varphi_l\varphi_r}$ are either in form of $X\gamma_1 \dots \gamma_{m-1}\,\varphi_r$ or $X\gamma_1 \dots \gamma_{m-1}\,\overline{\varphi_r}$. It follows that $\varphi_l = \overline{\varphi_l}$. It remains to show that $\varphi_r = \overline{\varphi_r}$. Note that $\rho(\alpha_2), \rho(\alpha_3) \leqslant n$ since both $\gamma, \delta \in R_{n-1}$. Moreover, from the induction hypothesis, we know that $R_0, \dots, R_{n-1}$ are unambiguous. And so, since $z \in L(\varphi_r) \cap L(\overline{\varphi_r})$, we can use Lemma 16 to conclude that $\varphi_r = \overline{\varphi_r}$.

ii. If $\alpha_3 = R_k$, then necessarily there exist such productions $\eta, \overline{\eta} \in R_k$ that $\varphi_l\varphi_r \in \text{RewritingSet}(\alpha_2, \eta)$, whereas $\overline{\varphi_l\varphi_r} \in \text{RewritingSet}(\alpha_2, \overline{\eta})$. Due to Proposition 10, we know that $L(\varphi_l\varphi_r) \subseteq L(\eta)$ and $L(\overline{\varphi_l\varphi_r}) \subseteq L(\overline{\eta})$. It implies that $wz \in L(\eta) \cap L(\overline{\eta})$; however, since $k < n$, we know from the induction hypothesis that R_k is unambiguous. Hence $\eta = \overline{\eta}$. Finally, it means that we can reduce this case to one of the previous cases when α_3 is complex, concluding that $\varphi_l\varphi_r = \overline{\varphi_l\varphi_r}$.

□

3.5 Generating functions

Fix an arbitrary normal-order reduction grammar R_n. Let us consider the counting sequence $(r_{n,k})_{k\in\mathbb{N}}$ where $r_{n,k}$ denotes the number of SK-combinators of size k reducing in n normal-order reduction steps. Suppose we associate with it a formal power series $R_n(z)$ defined as

$$R_n(z) = \sum_{k=0}^{\infty} r_{n,k}\, z^k$$

In the following theorem, we present a recursive method of computing the closed-form solution (i.e., finite representation using elementary functions) of $R_n(z)$ using the regular tree grammars $R_0, \dots, R_n$ and the inductive use of the *Symbolic Method* developed by Flajolet and Sedgewick (2009).

Theorem 18
For each $n \geqslant 0$, the ordinary generating function $R_n(z)$ corresponding to the sequence $(r_{n,k})_{k\in\mathbb{N}}$ has a computable closed form solution.

Proof
Induction over n. Let us start with giving previously computed closed-form solutions for $C(z)$, i.e., the generating function corresponding to the set of all SK-combinators, and $R_0(z)$ (Bendkowski *et al.*, 2015):

$$C(z) = \frac{1 - \sqrt{1 - 8z}}{2z} \qquad R_0(z) = \frac{1 - 2z - \sqrt{1 - 4z - 4z^2}}{2z^2} \tag{1}$$

Clearly, both $C(z)$ and $R_0(z)$ are computable.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Now, suppose that $n \geqslant 1$. Recall that in its construction, R_n might depend on previous reduction grammars $R_0, \ldots, R_{n-1}$, the set $\mathscr{C}$ of all SK-combinators and itself, via self-referencing productions. Due to Theorem 17, R_n is unambiguous and so we can express its generating function $R_n(z)$ as the unique solution of

$$R_n(z) = \sum_{\alpha \in R_n} z^{k(\alpha)} C(z)^{c(\alpha)} \prod_{i=0}^{n} R_i(z)^{r_i(\alpha)} \tag{2}$$

where $k(\alpha)$, $c(\alpha)$ and $r_i(\alpha)$ denote, respectively, the number of applications, the number of non-terminal symbols C and the number of non-terminal symbols R_i in α.

Note that R_n has exactly four self-referencing productions, i.e., SR_n, KR_n, SR_0R_n and SR_nR_0. It means that by converting them into appropriate functional equations, we can further rewrite (2) as

$$R_n(z) = 2zR_n(z) + 2z^2R_0(z)R_n(z) + \sum_{\alpha \in \Phi(R_n)} z^{k(\alpha)} C(z)^{c(\alpha)} \prod_{i=0}^{n-1} R_i(z)^{r_i(\alpha)} \tag{3}$$

where $\Phi(R_n)$ denotes the set of productions $\alpha \in R_n$ that do not reference R_n. By the induction hypothesis, we can compute the closed-form solutions for $R_0(z), \ldots, R_{n-1}(z)$ turning (3) into a linear equation in $R_n(z)$. Simplifying (1) for $R_0(z)$, we derive the final closed-form solution

$$R_n(z) = \frac{1}{\sqrt{1-4z-4z^2}} \sum_{\alpha \in \Phi(R_n)} z^{k(\alpha)} C(z)^{c(\alpha)} \prod_{i=0}^{n-1} R_i(z)^{r_i(\alpha)}$$

□

3.6 Other applications

In this section, we highlight some interesting consequences of the existence of normal-order reduction grammars. In particular, we prove that terms reducing in n steps have necessarily bounded length. Moreover, we show that the problem of deciding whether a given term reduces in n steps can be done in memory independent of the size of the term.

Proposition 19
If $\alpha \in R_n$, then α has length at most $2n + 2$.

Proof
Induction over n. The base case $n = 0$ is clear from the shape of R_0. Fix $n > 0$. Let us consider long productions in R_n. If β is a K-Expansion of some $X\alpha_1 \ldots \alpha_m \in R_{n-1}$, then

$$\beta = K(X\alpha_1 \ldots \alpha_k)\mathscr{C}\alpha_{k+1} \ldots \alpha_m \qquad \text{for} \quad 0 \leqslant k \leqslant m-1$$

In particular, the longest K-Expansion of $X\alpha_1 \ldots \alpha_m$ is in form of

$$\overline{\beta} = KX\alpha_1 \ldots \alpha_k \mathscr{C}\alpha_{k+1} \ldots \alpha_m$$

Note that $\overline{\beta}$ is of length $m + 2$ and so by the induction hypothesis at most $2n + 2$.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Now, let us consider the case when β is a S-Expansion of some $X\alpha_1 \dots \alpha_m \in R_{n-1}$. Then,

$$\beta = S(X\alpha_1 \dots \alpha_k)\varphi_l\varphi_r\alpha_{k+3} \dots \alpha_m \qquad \text{for} \quad 0 \leqslant k \leqslant m-2$$

where in addition $(\varphi_l\, \varphi_r) \in \textsc{RewritingSet}(\alpha_{k+1}, \alpha_{k+2})$. Again, the longest S-Expansion of $X\alpha_1 \dots \alpha_m$ is in form of

$$\overline{\beta} = SX\alpha_1 \dots \alpha_k\varphi_l\varphi_r\alpha_{k+3} \dots \alpha_m$$

It follows that $\overline{\beta}$ is of length at most $m+1$ and so also at most $2n+1$. □

In other words, terms reducing in n steps cannot be too long as their length is tightly bounded by $2n+2$. Now, let us consider the following two problems.

Problem: n-step-reducible
Input: A combinatory logic term $x \in L(\mathscr{C})$.
Output: yes if and only if x reduces in n steps.

Problem: reduces-in-n-steps
Input: A combinatory logic term $x \in L(\mathscr{C})$ and a number $n \in \mathbb{N}$.
Output: yes if and only if x reduces in n steps.

Let us start with the n-step-reducible problem. Since n in not a part of the input, we can compute R_n in constant time and memory. Using R_n, we build a bottom-up tree automaton recognizing $L(R_n)$ (Comon *et al.*, 2007) and use it to check whether $x \in L(R_n)$ in time $O(|x|)$, without using additional memory. On the other hand, the Naive algorithm, performing up to n normal-order reduction steps, requires $O(|x|)$ time and additional memory. At each step, the considered term doubles at most in size, as $Sxyz \to_w xz(yz)$. In order to find the next redex, we spend up to linear time in the current size of x; therefore, both size and time are bounded by

$$\begin{aligned} |x| + 2|x| + 4|x| + \dots + 2^n|x| &= |x|\left(1 + 2 + 4 + \dots + 2^n\right) \\ &= |x|\left(2^{n+1} - 1\right) = O(|x|) \end{aligned}$$

As a natural extension of the above discussion, we get the following corollary.

Corollary 20
The reduces-in-n-steps problem is decidable in space depending exclusively on n, independently of $|x|$.

3.7 Upper bound

In this section, we focus on the upper bound on the number $|R_n|$ of productions in R_n. We show that there exists a primitive recursive function $f : \mathbb{N} \to \mathbb{N}$ such that $|R_n| \leqslant f(n)$.

Following the scheme of the soundness proofs in Section 3.2, we construct suitable upper bounds using the notions of tree potential and degree. In the end of this

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

section, we show that these values are in fact bounded in each R_n, thus giving the desired upper bound.

Lemma 21
Let α, β be two trees of degree at most n such that their total potential $\pi(\alpha) + \pi(\beta)$ is equal to p. Then, the number of distinct meshes in $\textsc{MeshSet}(\alpha, \beta)$ is bounded by $|R_n|^{e\,p!}$.

Proof
Induction over the total potential p. Consider the following primitive recursive function $f_n : \mathbb{N} \to \mathbb{N}$:

$$f_n(k) = \begin{cases} 1 & \text{if } \ k = 0 \\ \big(|R_n| \cdot f_n(k-1)\big)^k & \text{otherwise} \end{cases}$$

We claim that $|\textsc{MeshSet}(\alpha, \beta)| \leqslant f_n(p)$. Note that it suffices to consider such α, β that $|\textsc{MeshSet}(\alpha, \beta)| > 1$ since f_n is an increasing function attaining positive values for any given input. It follows that the base case $p = 0$ is clear, as if $\pi(\alpha) + \pi(\beta) = 0$, then $\textsc{MeshSet}(\alpha, \beta)$ is necessarily empty. Now, let us assume that $p > 0$. From the construction of the common mesh set M of α and β, we can distinguish two cases left to consider.

i Suppose that $\alpha = X\alpha_1 \ldots \alpha_m$ and $\beta = X\beta_1 \ldots \beta_m$. In order to maximize the size of M, we can furthermore assume that none of the pairs α_i, β_i are rewritable. And so, the total number of meshes in M is equal to the product of all meshes in corresponding mesh sets for α_i and β_i. The degree of α_i and β_i is still at most n; however, $\pi(\alpha_i) + \pi(\beta_i) \leqslant p - 2$. Hence, using the induction hypothesis, we get $|\textsc{MeshSet}(\alpha_i, \beta_i)| \leqslant f_n(p-2)$. Since both α, β are of length $m \leqslant p$, we can furthermore state that

$$\begin{aligned} |M| &\leqslant (f_n(p-2))^m \leqslant (f_n(p-2))^p \\ &\leqslant (f_n(p-1))^p \leqslant \big(|R_n| \cdot f_n(p-1)\big)^p \\ &= f_n(p)s \end{aligned}$$

ii Let us assume w.l.o.g. that $\alpha = R_i$ and β is complex. In order to maximize the total number of meshes in M, we can moreover assume that all productions $\gamma \in R_i$ are similar to β and generate disjoint sets of meshes. We claim that $\textsc{MeshSet}(\gamma, \beta) \leqslant f_n(p-1)$. And so, if γ does not reference R_i, then our claim is trivially true. Suppose that γ is a self-referencing production. If $\gamma = XR_i$, then β is in form of $X\beta_1$. From the construction of M, we get that

$$|\textsc{MeshSet}(\gamma, \beta)| = |\textsc{MeshSet}(R_i, \beta_1)|$$

As $\pi(R_i) + \pi(\beta_1) \leqslant p - 1$, we can apply the induction hypothesis to $\textsc{MeshSet}(R_i, \beta_1)$ and immediately obtain $|\textsc{MeshSet}(\gamma, \beta)| \leqslant f_n(p-1)$. Now, suppose w.l.o.g. that $\gamma = SR_iR_0$ and hence $\beta = S\beta_1\beta_2$. Again, from the construction of M we know that

$$|\textsc{MeshSet}(\gamma, \beta)| = |\textsc{MeshSet}(R_i, \beta_1)| \cdot |\textsc{MeshSet}(R_0, \beta_2)|$$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Due to the fact that both $\pi(R_i) + \pi(\beta_1) \leqslant p-2$ and $\pi(R_0) + \pi(\beta_2) \leqslant p-2$, we can use the induction hypothesis and immediately get that

$$\begin{aligned} |\textsc{MeshSet}(\gamma, \beta)| &= |\textsc{MeshSet}(R_i, \beta_1)| \cdot |\textsc{MeshSet}(R_0, \beta_2)| \\ &\leqslant f_n(p-2)\, f_n(p-2) \end{aligned}$$

Note that $(f_n(p-2))^2 \leqslant f_n(p-1)$ for $p \geqslant 2$ and, in consequence, $|\textsc{MeshSet}(\gamma, \beta)| \leqslant f_n(p-1)$. Indeed, if $p = 2$, then $(f_n(p-2))^2 = 1 \leqslant f_n(1) = |R_n|$. Otherwise if $p > 2$, then

$$\begin{aligned} f_n(p-1) &= \big(|R_n| \cdot f_n(p-2)\big)^{p-1} \\ &= \Big(|R_n|^{p-1} (f_n(p-3))^{p-2}\Big)^{p-1} \\ &\geqslant \Big(|R_n|^{p-2} (f_n(p-3))^{p-2}\Big)^{p-1} \\ &= \big(|R_n| \cdot f_n(p-3)\big)^{(p-1)(p-2)} \end{aligned}$$

As $2(p-2) \leqslant (p-1)(p-2)$ for $p > 2$, we finally obtain

$$\begin{aligned} \big(|R_n| \cdot f_n(p-3)\big)^{(p-1)(p-2)} &\geqslant \big(|R_n| \cdot f_n(p-3)\big)^{2(p-2)} \\ &= (f_n(p-2))^2 \end{aligned}$$

We know therefore that $\textsc{MeshSet}(\gamma, \beta) \leqslant f_n(p-1)$ for each $\gamma \in R_i$. Finally, using the fact that $|R_i| \leqslant |R_n|$, we get

$$\begin{aligned} |M| &\leqslant |R_n| \cdot f_n(p-1) \\ &\leqslant \big(|R_n| \cdot f_n(p-1)\big)^p \\ &= f_n(p) \end{aligned}$$

And so, we know that $|\textsc{MeshSet}(\alpha, \beta)| \leqslant f_n(p)$. Solving the recurrence for $f_n(p)$, using, e.g., Mathematica® (Wolfram Research, 2015), we obtain the following closed form expression:

$$f_n(p) = |R_n|^{e\, p\, \Gamma(p,1)}$$

where

$$\Gamma(s, x) = (s-1)!\, e^{-x} \sum_{k=0}^{s-1} \frac{x^k}{k!}$$

is the upper incomplete gamma function (see, e.g., Abramowitz & Stegun, 1972). Simplifying the above expression in the case $x = 1$ and using the observation that $\sum_{k=0}^{s-1} \frac{1}{k!} \leqslant e$ for arbitrary s, we finally obtain the anticipated upper bound

$$f_n(p) \leqslant |R_n|^{e\, p!}$$

□

Lemma 22
Let α, β be two trees of degree at most n such that their total potential $\pi(\alpha) + \pi(\beta)$ is equal to p. Then, the number of distinct trees in $\textsc{RewritingSet}(\alpha, \beta)$ is bounded by $|R_n|^{1+e\, p!}$.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Proof
If $|\textsc{RewritingSet}(\alpha, \beta)| \leqslant 1$, then our claim is trivially true. Let us focus therefore on the remaining cases when either $\beta = X\beta_1 \dots \beta_m$ and both β_m and α are non-rewritable, or $\beta = R_i$.

First, consider the former case. Note that the resulting rewriting set is of equal size as $\textsc{MeshSet}(\alpha, \beta_m)$. Since $\pi(\alpha) + \pi(\beta_m) \leqslant p - 1$, we can use Lemma 21 to deduce that

$$|\textsc{RewritingSet}(\alpha, \beta)| = |\textsc{MeshSet}(\alpha, \beta_m)| \leqslant |R_n|^{e(p-1)!} < |R_n|^{1+e\,p!}$$

Now, let us consider the latter case. In order to maximize the resulting rewriting set we assume that each production $\gamma \in R_i$ generates a disjoint set of trees. We claim that each production γ contributes at most $|R_n|^{e\,p!}$ new trees to the resulting rewriting set and therefore $|\textsc{RewritingSet}(\alpha, \beta)| \leqslant |R_n|^{1+e\,p!}$, as there are at most $|R_n|$ productions in R_i. Indeed, consider an arbitrary $\gamma \in R_i$. Evidently, if $|\textsc{RewritingSet}(\alpha, \gamma)| \leqslant 1$, then our claim is true. Hence, let us assume that $|\textsc{RewritingSet}(\alpha, \gamma)| > 1$. It follows that γ is complex. Let us rewrite it as $X\gamma_1 \dots \gamma_m$. Note that as in the previous case, the resulting rewriting set is of equal size as $\textsc{MeshSet}(\alpha, \gamma_m)$. Since $\pi(\alpha) + \pi(\gamma_m) \leqslant p - 1$, we use Lemma 21 and get

$$|\textsc{RewritingSet}(\alpha, \gamma)| = |\textsc{MeshSet}(\alpha, \gamma_m)| \leqslant |R_n|^{e(p-1)!} < |R_n|^{e\,p!}$$

□

Lemma 23
Let α, β be two trees of total potential $\pi(\alpha) + \pi(\beta)$ equal to p. Then, each mesh in $\textsc{MeshSet}(\alpha, \beta)$ has potential bounded by $p!(1 + e)$.

Proof
Induction over total potential p. Again, it suffices to consider such α, β that $\textsc{MeshSet}(\alpha, \beta)$ is not empty. Immediately, the base case $p = 0$ is clear. Let us assume that $p > 0$. Consider the following primitive recursive function $f : \mathbb{N} \to \mathbb{N}$:

$$f(k) = \begin{cases} 1 & \text{if } \ k = 0 \\ k \cdot (f(k-1) + 1) & \text{otherwise} \end{cases}$$

Let $\gamma \in \textsc{MeshSet}(\alpha, \beta)$. We claim that $\pi(\gamma) \leqslant f(p)$. Note that f is an increasing function attaining positive values for any input. We have two cases to consider.

i. Suppose that $\alpha = X\alpha_1 \dots \alpha_m$ and $\beta = X\beta_1 \dots \beta_m$. Note that $\pi(\alpha_i) + \pi(\beta_i) \leqslant p - 2$ for each pair of corresponding arguments α_i, β_i. Using the induction hypothesis to pairs α_i, β_i and the fact that $\gamma \in \textsc{MeshSet}(\alpha, \beta)$ is similar to both α and β, we bound γ's potential by

$$\pi(\gamma) \leqslant m \cdot f(p-2) + m \leqslant p \cdot (f(p-2) + 1) \leqslant f(p)$$

ii. Assume w.l.o.g. that $\alpha = R_i$ and β is complex. It follows that $\gamma \in \textsc{MeshSet}(\delta, \beta)$ for some $\delta \in R_i$. If δ does not reference R_i, then clearly $\pi(\delta) \leqslant \pi(R_i) - 1$ and therefore $\pi(\gamma) \leqslant f(p-1)$. Now, suppose that δ is a self-referencing production of R_i.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

If $\delta = XR_i$, then β is in form of $X\beta_1$ and similarly $\gamma = X\gamma_1$. It follows that $\pi(\delta) = \pi(R_i) + 1$ and therefore $\pi(\delta) + \pi(\beta) = p + 1$. Note however that $\pi(\gamma_1) \leqslant f(p-1)$ as $\pi(R_i)+\pi(\beta_1) \leqslant p-1$. Due to that $\pi(\gamma) = 1+f(p-1) \leqslant f(p)$. Let us assume w.l.o.g. that $\delta = SR_iR_0$. Immediately, β is in form of $S\beta_1\beta_2$, whereas $\gamma = S\gamma_1\gamma_2$. Moreover, $\pi(\delta) = \pi(R_i) + 3$. Note however that both $\pi(R_i) + \pi(\beta_1) \leqslant p - 2$ and $\pi(R_0) + \pi(\beta_2) \leqslant p - 2$. We can therefore use the induction hypothesis and conclude that

$$\pi(\gamma) = 2 + \pi(\gamma_1) + \pi(\gamma_2) \leqslant 2 + 2 \cdot f(p-2)$$

Since $\pi(\delta) \geqslant 4$, we know that $p \geqslant 3$ and so we can further bound $\pi(\gamma)$ by

$$\begin{aligned}\pi(\gamma) &= 2\,(1 + f(p-2)) \\ &\leqslant (p-1)\,(1 + f(p-2)) \\ &= f(p-1) \leqslant f(p)\end{aligned}$$

Finally, we know that $\pi(\gamma) \leqslant f(p)$. What remains is to solve the recursion, using, e.g., Mathematica® (Wolfram Research, 2015), for f and give its closed form solution. It follows that

$$\begin{aligned}f(p) &= \Gamma(1+p) + e\,p\,\Gamma(p,1) \\ &\leqslant p! + e\,p! \\ &= p!(1+e)\end{aligned}$$

where

$$\Gamma(n) = (n-1)!$$

□

Lemma 24
Let α, β be two trees of potential $\pi(\alpha) + \pi(\beta) = p$. Then, each tree in RewritingSet(α, β) has potential bounded by $p!(1+e) + p$.

Proof
Let γ be an arbitrary tree in RewritingSet(α, β). Based on the structure of β, we have several cases to consider. If $\beta = \mathscr{C}$, then $\gamma = \mathscr{C}\alpha$ and so $\pi(\gamma) = \pi(\alpha)+1 = p+1$. Note that $1 < p!(1+e)$ for any p and thus our bound holds.

If $\beta = X\beta_1 \dots \beta_m$, then $\pi(\alpha) + \pi(\beta_m) \leqslant p - 1$. In both cases when $\alpha \rhd\lhd \beta_m$, the resulting tree has potential bounded by p and so also by $p!(1+e)+p$. Let us assume that $\alpha \parallel \beta_m$. We can therefore rewrite γ as $X\gamma_1 \dots \gamma_m$. Using Lemma 23, we know that $\pi(\gamma_m) \leqslant (p-1)!(1+e)$. Moreover, both α and β are similar to γ. Let us rewrite them as $X\alpha_1 \dots \alpha_m$ and $X\beta_1, \dots, \beta_m$, respectively. Note that for each $i < m$, γ_i is equal to α_i or β_i. It follows that we can bound the potential of $X\gamma_1 \dots \gamma_{m-1}$ by $p - 1$ and hence γ's potential by $(p-1)!(1+e) + p$.

Now, if $\beta = R_i$, then $\gamma \in$ RewritingSet(α, δ) for some $\delta \in R_i$. If δ does not reference R_i, we know that $\pi(\delta) \leqslant \pi(R_i) - 1 \leqslant p - 1$. Moreover, δ is complex, as otherwise RewritingSet$(\alpha, \delta) = \emptyset$. Using our previous argumentation, we can therefore conclude that $\pi(\gamma) \leqslant (p-1)!(1+e)+p$. Suppose that δ is a self-referencing

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

production of R_i. If $\delta = XR_i$, then α is in form of $X\alpha_1$ and $\gamma = X\gamma_1$. Immediately, $\pi(\alpha) + \pi(\delta) = p + 1$. If $R_i \bowtie \alpha_1$, then γ has potential bounded by p. Therefore, let us assume that $R_i \parallel \alpha_1$. Since $\pi(R_i) + \pi(\alpha_1) = p - 1$, we know from Lemma 23 that $\pi(\gamma_1) \leqslant (p-1)!(1+e)$. It follows immediately that $\pi(\gamma) \leqslant (p-1)!(1+e)+1 \leqslant p!(1+e)+p$.

Finally, suppose that $\delta = S\delta_1\delta_2$ and so $\alpha = S\alpha_1\alpha_2$. Immediately, $\gamma = S\gamma_1\gamma_2$. Again, if $\delta_2 \bowtie \alpha_2$, we can bound γ's potential by p. Hence, let us assume that $\delta_2 \parallel \alpha_2$. Clearly, $\pi(\alpha) + \pi(\delta) = p + 3$. Note however that $\pi(\alpha_1) + \pi(\delta_1) \leqslant p - 2$ and $\pi(\alpha_2) + \pi(\delta_2) \leqslant p - 2$, as both δ_1 and δ_2 are non-terminal reduction grammar symbols of positive potential. Using Lemma 23 to MeshSet(α_2, δ_2), we conclude that $\pi(\gamma_2) \leqslant (p-2)!(1+e)$. It follows that $\pi(\gamma) \leqslant (p-2)!(1+e)+p \leqslant p!(1+e)+p$. □

Lemma 25
There exists a primitive recursive function $\psi : \mathbb{N} \to \mathbb{N}$ such that $\pi(R_n) \leqslant \psi(n)$.

Proof
Consider the following function $\psi : \mathbb{N} \to \mathbb{N}$:

$$\psi(k) = \begin{cases} 1 & \text{if } k = 0 \\ 4\,(\psi(k-1)+2)! + 2\psi(k-1) + 5 & \text{otherwise} \end{cases}$$

Evidently, ψ is an increasing primitive recursive function. We show that $\psi(n)$ bounds the potential of R_n using induction over n. Since $\pi(R_0) = \psi(0) = 1$, the base case is clear. Let $n > 0$. In order to prove our claim, we have to check that $\pi(\alpha) \leqslant \psi(n) - 1$ for all productions $\alpha \in R_n$ that do not reference R_n.

i. Suppose that $\alpha = SR_{n-i}R_i$. The potential of α is equal to $2 + \pi(R_{n-i}) + \pi(R_i)$. Using the induction hypothesis, we know moreover that

$$\begin{aligned} \pi(\alpha) &\leqslant 2 + \psi(n-i) + \psi(i) \\ &\leqslant 2 + 2\psi(n-1) \\ &\leqslant \psi(n) - 1 \end{aligned}$$

ii. Let $\alpha = KR_{n-1}\mathscr{C}$. Due to the fact that $\pi(\alpha) = 2 + \pi(R_{n-1})$, we use the induction hypothesis and immediately obtain

$$\pi(\alpha) \leqslant 2 + \psi(n-1) \leqslant \psi(n) - 1$$

iii. Suppose that $\alpha \in$ K-Expansions(β) for some $\beta \in R_{n-1}$. Note that $\pi(\beta) \leqslant \psi(n-1) + 3$ as the productions of greatest potential in R_{n-1} are exactly $SR_{n-1}R_0$ and SR_0R_{n-1}. Since $\pi(\alpha) = 2 + \pi(\beta)$, we get

$$\pi(\alpha) \leqslant 5 + \psi(n-1) \leqslant \psi(n) - 1$$

iv. Finally, let $\alpha \in$ S-Expansions(β) for some $\beta \in R_{n-1}$. Again, $\pi(\beta) \leqslant \pi(R_{n-1}) + 3$ and hence from the induction hypothesis $\pi(\beta) \leqslant \psi(n-1) + 3$. Let us rewrite α as $S(X\beta_1 \ldots \beta_k)\varphi_l\varphi_r\beta_{k+3} \ldots \beta_m$ where $\beta = X\beta_1 \ldots \beta_m$. Note that $\pi(\alpha) \leqslant \pi(\beta) + \pi(\varphi_l) + \pi(\varphi_r) + 1$. Moreover, as $\pi(\varphi_l\varphi_r) = 1 + \pi(\varphi_l) + \pi(\varphi_r)$, we get $\pi(\alpha) \leqslant \pi(\beta) + \pi(\varphi_l\varphi_r)$. Since $\pi(\beta_{k+1}\beta_{k+2}) \leqslant \pi(\beta) - 1$ and thus, $\pi(\beta_{k+1}\beta_{k+2}) \leqslant$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

$\psi(n-1)+2$, we can use Lemma 24 to obtain

$$\pi(\varphi_l\varphi_r) \leqslant (\psi(n-1)+2)!(1+e)+\psi(n-1)+2$$

It follows therefore that

$$\begin{aligned}\pi(\alpha) &\leqslant \pi(\beta)+\pi(\varphi_l\varphi_r)\\ &\leqslant (\psi(n-1)+2)!(1+e)+2\psi(n-1)+5\\ &\leqslant \psi(n)-1\end{aligned}$$

where the last inequality follows from the fact that

$$(3-e)\,(\psi(n-1)+2)! \geqslant \frac{1}{5}(\psi(n-1)+2)! \geqslant \frac{6}{5} \geqslant 0$$

□

Theorem 26
There exists a primitive recursive function $\chi : \mathbb{N} \to \mathbb{N}$ such that the number $|R_n|$ of productions in R_n is bounded by $\chi(n)$.

Proof
Consider R_n for some $n > 0$. Note that R_n consists of:

i. two productions SR_n and KR_n;
ii. $n+1$ short S-productions in form of $SR_{n-i}R_i$;
iii. an additional K-production $KR_{n-1}\mathscr{C}$;
iv. K-Expansions(α) for each $\alpha \in R_{n-1}$;
v. S-Expansions(α) for each $\alpha \in R_{n-1}$.

It suffices therefore to bound the number of K- and S-Expansions, as the number of other productions in R_n is clear. Let us start with K-Expansions. Suppose that α is of length m. Then, we have $|\text{K-Expansions}(\alpha)| = m$. Using Proposition 19, we know that that each production $\alpha \in R_{n-1}$ is of length at most $2n$. It follows that there are at most $2n \cdot |R_{n-1}|$ K-Expansions in R_n. Now, let us consider S-Expansions. In order to bound the number of S-Expansions in R_n, we assume that each production $\alpha \in R_{n-1}$ is of length $2n$ and moreover each RewritingSet of appropriate portions of α generates a worst-case set of trees. And so, assuming that α is of length $2n$, we can rewrite it as $X\alpha_1 \dots \alpha_{2n}$. Let ψ denote the upper bound function on the potential of R_{n-1} from Lemma 25. Evidently, $\pi(\alpha) \leqslant \psi(n-1)+3$. Now, using Lemma 22, we know that each RewritingSet(α_i, α_{i+1}) contributes at most

$$|R_{n-1}|^{1+e\left(\psi(n-1)+3\right)!}$$

new S-Expansions. As there are at most $2n-1$ pairs of indices $(i, i+1)$ yielding RewritingSets, we get that the number of S-Expansions in R_n is bounded by

$$(2n-1) \cdot |R_{n-1}| \cdot |R_{n-1}|^{1+e\left(\psi(n-1)+3\right)!} \leqslant (2n-1) \cdot |R_{n-1}|^{2+3\left(\psi(n-1)+3\right)!}$$

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

Finally, since $|R_0| = 5$, we combine the above observations and get the following primitive recursive upper bound on $|R_n|$:

$$\chi(k) = \begin{cases} 5 & \text{if } k = 0 \\ 4 + k + 2k \cdot \chi(k-1) & \\ \quad +(2k-1) \cdot \chi(k-1)^{2+3\left(\psi(k-1)+3\right)!} & \text{otherwise} \end{cases}$$

□

4 Conclusion

We gave a complete syntactic characterization of normal-order reduction for combinatory logic over the set of primitive combinators S and K. Our characterization uses regular tree grammars and therefore exhibits interesting algorithmic applications, including the computation of corresponding generating functions. We investigated the complexity of the generated reduction grammars, giving a primitive recursive upper bound on the number of their productions. We emphasize the fact that although the size of R_n is bounded by a primitive recursive function of n, it seems to be enormously overestimated. Our computer implementation of the Reduction Grammar algorithm (Bendkowski, 2016) suggests that the first few numbers in the sequence $(|R_n|)_{n\in\mathbb{N}}$ are in fact

$$5, 12, 75, 625, 5673, 53164, 508199, \ldots$$

The upper bound $\chi(1)$ on the size of R_1 is already of order $6 \cdot 10^{84549}$ whereas the actual size of R_1 is equal to 12. Naturally, we conjecture that $(|R_n|)_{n\in\mathbb{N}}$ grows much slower than $(\chi(n))_{n\in\mathbb{N}}$, although the problem of giving better approximations on the size of R_n for large n is still open.

The effective existence of reduction grammars as well as their ordinary generating functions enables large random combinator generation with prescribed reduction lengths using, e.g., the prominent theory of Boltzmann samplers (Duchon *et al*., 2004). In principle, it is therefore possible to sample uniformly random SK-combinators reducing in any fixed number of normal-order reductions. Unfortunately, due to the quickly intractable size of reduction grammars, such tools are able to operate on just the first few grammars. Nonetheless, under the standard translation to λ-calculus, combinators reducing under just a few normal-order reduction steps are a novel, non-trivial source of large random normalizing λ-terms.

Interestingly, there exist other type-theoretic approaches to counting normalizing λ-terms. In Bernadet & Lengrand (2013) authors provide a non-idempotent intersection type system capturing strongly normalizing terms in, inter alia, pure λ-calculus. In this system, a λ-term is typeable if and only if each of its reduction sequences is finite. Moreover, as a byproduct, it is possible to obtain the length of the longest reduction sequence from the assigned type. Unfortunately, such a typing system does not yield effective sampling methods and hence cannot be used as a source of random normalizing λ-terms.

https://doi.org/10.1017/S0956796816000332 Published online by Cambridge University Press

In consequence, it is natural to ask the following question. Do similar combinatorial results hold in the universe of λ-calculus? In particular, is it possible to provide a recursive method computing reduction grammars in either typed or untyped λ-calculus? The context-sensitive substitution operation in λ-calculus suggest that if so, then the resulting reduction grammars ought to be context-sensitive as well, in particular, more expressible than regular grammars used for SK-combinators. It seems that although similar in nature to our investigations, this open problem requires significantly more advanced tools to address.

Acknowledgments

We would like to thank the anonymous referees for their insightful comments and suggestions on the paper, especially suggesting the paper by Bernadet & Lengrand (2013). We would also like to thank Katarzyna Grygiel for many fruitful discussions and valuable comments.

References

Abramowitz, M. & Stegun, I. (1972) *Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables*. Dover Publications.

Barendregt, H. P. (1984) *The Lambda Calculus, Its Syntax and Semantics*. vol. 103. North Holland.

Bendkowski, M. (2016) *Normal-Order Reduction Grammars—Haskell Implementation*. Available at: https://github.com/maciej-bendkowski/normal-order-reduction-grammars, Last accessed 03.01.2017.

Bendkowski, M., Grygiel, K. & Zaionc, M. (2015) Asymptotic properties of combinatory logic. In *Theory and Applications of Models of Computation*, Jain, R., Jain, S. & Stephan, F. (eds), Lecture Notes in Computer Science, vol. 9076. Springer International Publishing, pp. 62–72.

Bendkowski, M., Grygiel, K., Lescanne, P. & Zaionc, M. (2016) A natural counting of lambda terms. In *SOFSEM 2016: Theory and Practice of Computer Science*, Freivalds, R. M., Engels, G. & Catania, B. (eds), Lecture Notes in Computer Science, vol. 9587. Springer International Publishing, pp. 183–194.

Bernadet, A. & Lengrand, S. J. (2013) Non-idempotent intersection types and strong normalisation. *Log. Methods Comput. Sci.* **9**(4), 1–46.

Bodini, O., Gardy, D., Gittenberger, B. & Gołębiewski, Z. (2015) *On the Number of Unary-Binary Tree-Like Structures with Restrictions on the Unary Height*. arXiv:1510.01167.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S. & Tommasi, M. (2007) *Tree Automata Techniques and Applications*. Available at: http://www.grappa.univ-lille3.fr/tata. release October, 12th 2007.

Curry, H. B. (1930) Grundlagen der kombinatorischen Logik. *Am. J. Math.* **52**(3), 509–536.

Curry, H. B. & Feys, R. (1958) *Combinatory Logic*, vol. 1. Amsterdam: North Holland.

David, R., Grygiel, K., Kozik, J., Raffalli, C., Theyssier, G. & Zaionc, M. (2013) Asymptotically almost all λ-terms are strongly normalizing. *Log. Methods Comput. Sci.* **9**(1), 1–30.

Duchon, P., Flajolet, P., Louchard, G., & Schaeffer, G. (2004) Boltzmann samplers for the random generation of combinatorial structures. *Comb. Probab. Comput.* **13**, 2004.

Flajolet, P. & Sedgewick, R. (2009) *Analytic Combinatorics*, 1 edn. New York, NY, USA: Cambridge University Press.

Gittenberger, B. & Gołebiewski, Z. (2016) On the number of lambda terms with prescribed size of their De Bruijn representation. In *Proceedings of 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17–20, 2016, Orléans, France*, Ollinger, N. & Vollmer, H. (eds), LIPIcs, vol. 47. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 40:1–40:13.

Grygiel, K. & Lescanne, P. (2013) Counting and generating lambda terms. *J. Funct. Program.* **23**(5), 594–628.

Grygiel, K. & Lescanne, P. (2015) Counting and generating terms in the binary lambda calculus. *J. Funct. Program.* **25**, e24 (25 pages).

Pałka, M., Claessen, K., Russo, A. & Hughes, J. (2011) Testing an optimising compiler by generating random lambda terms. In *Proceedings of the 6th International Workshop on Automation of Software Test, AST 2011*, Waikiki, Honolulu, HI, USA, May 23–24, 2011, Bertolino, A., Foster, H. & Li, J. (eds). ACM.

Schönfinkel, M. (1924) Über die Bausteine der mathematischen Logik. *Math. Ann.* **92**(3), 305–316.

Turner, D. (1979) A new implementation technique for applicative languages. *Softw.: Pract. Exp.* **9**(1), 31–49.

Turner, D. (1986) An overview of Miranda. *SIGPLAN Not.* **21**(12), 158–166.

Wolfram Research, Inc. (2015) *Mathematica Version 10.3*. Champaign, Illinois.