

Compression via Unduplication

Peter Leupold*

Research Group in Mathematical Linguistics

Rovira i Virgili University, Tarragona, Spain

eMail: Peter.Leupold@web.de

Abstract

Data compression most commonly is based on the reduction of repetitions to a more compact representation. We investigate the reduction of direct repetitions, more exactly of squares. The main focus is on the question, how many different reduced normal forms one word can have under this reduction. We show that this number can be exponential in the length of the original word.

0 Repetitions and Duplication

We investigate the reduction of direct repetitions in strings. This reduction is not directly applicable for data compression, because it is not reversible, i.e. the original string cannot be reconstructed. However, reduction of squares directly models the reverse of a operation common in biochemistry.

A mutation, which occurs frequently in DNA strands, is the duplication of a factor inside a strand [19]. The result is called a tandem repeat, and the detection of these repeats has received a great deal of attention in bioinformatics [1, 20]. The reconstruction of possible duplication histories of a gene is used in the investigation of the evolution of a species [24]. Thus duplicating factors and deleting halves of squares is an interesting algorithmic problem with some motivation from bioinformatics, although squares do not need to be exact there. A very similar reduction was also introduced in the context of data compression by Ilie et al. [10, 11]. They, however conserve information about each reduction step in the resulting string such that the operation can also be undone again. In this way the original word can always be reconstructed, which is essential for data compression. We will present their approach in more detail in Section 2 and establish some relations between the two reductions.

So far, the interpretation of duplication as an operation on a string has mainly inspired work in Formal Languages, most prominently the duplication closure. Dassow et al. introduced the duplication closure of a word and showed that the languages generated are always regular over two letters [7]. Wang then proved that this is not the case over three or more letters [23]. These

*Peter Leupold's work was done while he was funded as a Beatriu de Pinós researcher by the by the Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya.

results had actually been discovered before in the context of copy systems [8], [3]. It remains an open problem, whether such duplication closures are always context-free or not. Later on, length bounds for the duplicated factor were introduced [17], [15], and also the closure of language classes under the duplication operations was investigated [12]. Finally, also a special type of codes robust against duplications was investigated [16].

Besides considering duplication as a generative operation elongating strings, also the effects of the inverse operation on words have been the object of investigations [15]. Here duplications are undone, i.e. one half of them is deleted leaving behind only the other half of the square. In this way words are reduced to square-free words, which are in some sense primitive under this notion; this is why we call the set of all square-free words reachable from a given word w the duplication root of w in analogy to concepts like the primitive root or the periodicity root of words. Duplication roots of languages were studied already in earlier work by the present author [14].

Here we will focus on duplication roots of single words. Mainly the following question is addressed: how many different duplication roots can a word have? We establish an exponential lower bound for this number as well as an upper bound. Besides any possible applications, this study of how repetitions in a sequence can be nested follows important lines of study in Combinatorics of Words, where repetitions have been in the center of attention from the very start in the work of Thue [22].

1 Definitions

We assume the reader to be familiar with fundamental concepts from Formal Language Theory such as alphabet, word, and language, which can be found in many standard textbooks like the one by Harrison [9]. The length of a finite word w is the number of not necessarily distinct symbols it consists of and is written $|w|$. The number of occurrences of a certain letter a in w is $|w|_a$. The i -th symbol we denote by $w[i]$. The notation $w[i \dots j]$ is used to refer to the part of a word starting at the i -th position and ending at the j -th position.

A word u is a *prefix* of w if there exists an $i \leq |w|$ such that $u = w[1 \dots i]$; if $i < |w|$, then the prefix is called *proper*. The set of all prefixes is $\text{pref}(w)$. Suffixes are the corresponding concept reading from the back of the word to the front and they are denoted by suff . We define the *letter sequence* $\text{seq}(u)$ of a word u as follows: any word u can be uniquely factorized as $u = x_1^{i_1} x_2^{i_2} \dots x_\ell^{i_\ell}$ for some integers $\ell \geq 0$ and $i_1, i_2, \dots, i_\ell \geq 1$ and for letters $x_1, x_2, \dots, x_\ell$ such that always $x_j \neq x_{j+1}$; then $\text{seq}(u) := x_1 x_2 \dots x_\ell$. Intuitively speaking, every block of several adjacent occurrences of the same letter is reduced to just one occurrence.

We call a word w *square-free* iff it does not contain any non-empty factor of the form u^2 , where exponents of words refer to iterated catenation, and thus u^i is the i -fold catenation of the word u with itself. A word w has a positive integer k as a *period*, if for all i, j such that $i \equiv j \pmod{k}$ we have $w[i] = w[j]$, if both $w[i]$ and $w[j]$ are defined.

For applying duplications to words we use string-rewriting systems. In our notation we mostly follow Book and Otto [2] and define such a *string-rewriting system* R on Σ to be a subset of $\Sigma^* \times \Sigma^*$. Its single-step reduction relation is defined as $u \rightarrow_R v$ iff there exists $(\ell, r) \in R$ such that for some u_1, u_2 we have $u = u_1 \ell u_2$ and $v = u_1 r u_2$. We also write simpler just $\rightarrow$, if it is clear which is the underlying rewriting system. By $\rightarrow^*$ we denote the relation's reflexive

and transitive closure, which is called the *reduction relation* or *rewrite relation*. The inverse of a single-step reduction relation $\rightarrow$ is $\rightarrow^{-1} := \{(\ell, r) : (r, \ell) \in R\}$. Further notation that will be used is $IRR(R)$ for the set of words irreducible for a string-rewriting system R . With this we come to the definition of duplications.

The string-rewriting system we use here is the *duplication relation* defined as $u \heartsuit v :\Leftrightarrow \exists z[z \in \Sigma^+ \wedge u = u_1 z u_2 \wedge v = u_1 z z u_2]$. Notice how the symbol $\heartsuit$ nicely visualizes the operation going from one origin to two equal halves. If we have length bounds $|z| \leq k$ or $|z| = k$ on the factors to be duplicated we write $\heartsuit^{\leq k}$ or $\heartsuit^k$ respectively; the relations are called *bounded* and *uniformly bounded duplication* respectively.

$\heartsuit^*$ is the reflexive and transitive closure of the relation $\heartsuit$. The *duplication closure* of a word w is then $w^\heartsuit := \{u : w \heartsuit^* u\}$. The languages $w^{\heartsuit^{\leq k}}$ and $w^{\heartsuit^k}$ are defined analogously. Because our main topic is the reduction of squares, we will mainly use the inverse of $\heartsuit$ and will denote it by $\succ := \heartsuit^{-1}$; the notations for length-bounded versions and iterated applications are used accordingly. Notice that for $\succ_{\leq k}$ the length bound does not refer to the length of the rules' left sides, but rather to half that length. This makes sense, because otherwise for all even k we would have $\succ_{\leq k} = \succ_{\leq k+1}$, and because this way the relations $\succ_{\leq k}$ and $\heartsuit^{\leq k}$ correspond. We will use a similar convention when talking about squares. Thus we will say that a square u^2 is of length $|u|$; in this case u will be called the *base* of this square.

With this we have all the prerequisites for defining the central notion of this work, the duplication root.

Definition 1. The *duplication root* of a non-empty word w is

$$\sqrt[\heartsuit]{w} := IRR(\succ) \cap \{u : w \succ^* u\}.$$

As usual, this notion is extended in the canonical way from words to languages such that

$$\sqrt[\heartsuit]{L} := \bigcup_{w \in L} \sqrt[\heartsuit]{w}.$$

The roots $\sqrt[\heartsuit^{\leq k}]{w}$ and $\sqrt[\heartsuit^k]{w}$ are defined in completely analogous ways, and also these are extended to entire languages in the canonical way. When we want to contrast the duplication (root) without length bound to the bounded variants we will at times call it *general duplication (root)*.

When talking about the elements of a word's duplication root, we will also call them simply roots; no confusion should arise. Similarly, where we say "the number of roots" we mean the root's cardinality. Though not completely correct, these formulations are more compact and in many cases easier to understand.

Finally, notice that all words in a duplication root are square-free, and over an alphabet of two letters only the seven square-free words $\{\lambda, a, b, ab, ba, aba, bab\}$ exist. They are uniquely determined by their first letter, the last letter, and the set of letters occurring in them. Thus most problems about duplication roots are trivial unless we have at least three letters. Therefore, unless otherwise stated, we will suppose an alphabet of size at least three in what follows. First off, we illustrate this definition with an example that also shows that duplication roots are in general not unique, i.e., the set $\sqrt[\heartsuit]{w}$ can contain more than one element as we will see further on.

Example 2. By undoing duplications, i.e., by applying rules from $\succ$, we obtain from the word $w = abcbabcbc$ the words in the set $\{abc, abcbc, abcbabc\}$; in a first step either the prefix $(abcb)^2$ or the suffix $(bc)^2$ can be reduced, only the former case results in a word with another square, which can be reduced to abc .

Thus we have the root $\sqrt[3]{abcbabcbc} = \{abc, abcbabc\}$. Exhaustive search of all shorter words shows that this is a shortest possible example of a word with more than one root over three letters.

As the examples have finite length, the bounded duplication root is in general not unique either. The uniformly bounded duplication root, however, is known to be unique over any alphabet [15].

2 The Relation to Repetition Complexity

In an effort to define a new measure for the complexity of words, Ilie et al. [10, 11] defined a reduction relation very similar to undoing duplications, which however remembers the steps it takes, and in this way the original word can be restored from the reduced one. For the definition let $D = \{0, 1, \dots, 9\}$ be the set of decimal digits, and Σ be an alphabet disjoint from D . The alphabet for the reduction relation is $T := \Sigma \cup D \cup \{\langle, \rangle, \wedge\}$. For a positive integer n let $\text{dec } n$ denote its decimal representation. Then the reduction relation $\Rightarrow$ is defined by $u \Rightarrow v$ iff $u = u_1 x^n u_2$, $v = u_1 \langle x \rangle \wedge \langle \text{dec } n \rangle u_2$ for some $u_1, u_2 \in T^*$, $x \in \Sigma^+$, $n > 2$. Finally, let h be the morphism erasing all symbols except the letters from Σ .

We illustrate in a simple example the different way of operation of the two relations.

Example 3. For the word $ababcbc$ there are two irreducible forms under $\Rightarrow$, namely $\langle ab \rangle^{(2)} cbc$ and $aba \langle bc \rangle^{(2)}$. Under $\succ$, however, the images of both words under h are further reducible to a common normal form: both $ababcbc \succ abcbc \succ abc$ and $ababcbc \succ ababc \succ abc$ are possible reductions leading to abc . Notice how the brackets block the further reduction of $abab$ in $aba \langle bc \rangle \wedge \langle 2 \rangle$ and of $bc bc$ in $\langle ab \rangle \wedge \langle 2 \rangle cbc$.

There are two main differences between the two relations.

- (i). A reduction $u^n \Rightarrow \langle u \rangle \wedge \langle n \rangle$ is done in a single step while the reduction $u^n \succ^* u$ will always take $n - 1$ steps.
- (ii). If $w \Rightarrow^* u$ then $w \succ^* h(u)$, but the reverse does not hold, see Example 3.

Despite these differences, the similarities are evident, and $\Rightarrow^*$ can be embedded in $\succ^*$. We state a further relation.

Proposition 4. For a word w , if $\sqrt[3]{w} \subseteq \{h(u) : w \Rightarrow^* u\}$ then $|\sqrt[3]{w}| = 1$.

Proof. Let p and q be two different words in $\sqrt[3]{w}$. Then there exist words u, p', q' such that $w \succ^* u$, $u \succ p' \succ^* p$, $u \succ q' \succ^* q$, but no reductions $p' \succ^* q$ or $q' \succ^* p$ exist. Intuitively this means that the paths to p and q divide in the point u , which thus is a greatest lower bound of $\{p, q\}$ in the set $w \succ^*$ with $\succ^*$ as partial order. The two unduplications in $u \succ p'$ and $u \succ q'$ must overlap, otherwise there would be a word v such that $p' \succ v$ and $q' \succ v$. Let the two

factors that are reduced be u_p^2 and u_q^2 , where $|u_p| > |u_q|$ without loss of generality; notice that $|u_p| = |u_q|$ would result in $p' = q'$.

The overlap of the unduplications must be greater than $|u_q|$. Otherwise there is a w' such that the unduplications are applied to a factor $u_p w' u_q$ or $u_q w' u_p$ and the effect can be seen as the deletion of u_p and u_q ; both would be possible consecutively. Further, the maximal repetition of u_p were it is reduced must be less than u_p^3 , otherwise the factor u_q would still be present after deletion of one u_p . This means that a reduction under $\Rightarrow$ can only result in $\langle u_p \rangle \sim \langle 2 \rangle$, no higher exponent, and no factor u_p can follow on either side.

There can be no factor u_q^2 directly preceding or following $\langle u_p \rangle$ on the side of the overlap. Otherwise, again derivations $p' \succ v$ and $q' \succ v$ would have been possible. This means that the square u_q^2 in $h(p')$ cannot be reduced, neither can an equivalent reduction leading to the same result be done. Analogous reasoning holds for the case that first u_q^2 is reduced to $\langle u_q \rangle \sim \langle 2 \rangle$, and thus $\{h(u) : w \Rightarrow^* u\}$ cannot contain any square-free word. $\square$

Intuitively this means that if $\Rightarrow$ can reduce a word to a square-free one, then the overlaps of its repetitive factors must be so minor that they do not lead to ambiguous duplication roots either. Already Example 3 shows that the converse of Proposition 4 does not hold.

From the proof of Proposition 4 we can extract an important property of the relation $\succ^*$ that characterizes the situation, when two strings derived from the same word can become incomparable.

Definition 5. Let w be a word. We will call two squares p^2 and q^2 a pair of *critical squares* in w , if w has a factor u such that

- (i). $p^2 \in \text{pref}(u)$,
- (ii). $q^2 \in \text{suff}(u)$,
- (iii). $|u| \leq 2(\max(|p|, |q|)) + \min(|p|, |q|) - 1$.

Without further proof we state the following.

Lemma 6. Let w , p , and q be words such that $w \succ p$ and $w \succ q$. If $\{v : p \succ^* v\} \cap \{v : q \succ^* v\} = \emptyset$, then w contains a pair of critical squares.

3 The Number of Duplication Roots

A decisive question for any algorithmic problem related to duplication is the one about the possible number of duplication roots of a word with respect to its length. To find an exact bound seems to be a very intricate problem, and so we try to find good upper and lower bounds on this number. More formally, we try to find bounds for the function defined as

$$\text{duproots}(n) := \max\{|\sqrt[n]{w}| : |w| = n\}.$$

The function duproots is monotonically growing. For any word w , duplicate one of its letters to obtain a word w' of length $|w| + 1$. Clearly $w' \succ w$ and thus $\sqrt[n]{w} \subseteq \sqrt[n]{w'}$. Consequently we

have $\text{duproots}(n) \leq \text{duproots}(n+1)$ for all $n > 0$. Therefore writing $|w| = n$ in the definition is equivalent to writing $|w| \leq n$.

Because it has often turned out to be very useful to consider problems about duplications with a length restriction, we also define the function

$$\text{bduproots}_{\leq k}(n) := \max\{|\sqrt[k]{w}| : |w| = n\}.$$

By definition we have $\text{bduproots}_{\leq k} \leq \text{duproots}$ and $\text{bduproots}_{\leq k} \leq \text{bduproots}_{\leq k+1}$ for all $k > 0$. We now try to characterize the growth of the function duproots more exactly.

3.1 Bounding from Above

Obviously, rules from $\succ$ can only be applied on square factors. Thus the number of squares is the number of possible distinct rule applications in a string. However, when we are interested in rule applications with distinct result and thus with potentially distinct roots, the number of runs captures this more exactly.

Recall that a *run* is a maximal repetition of exponent at least two in a string. It is known that the number of runs in a string of length n is linearly bounded by n [13]. A great deal of work has been done to determine the constant c such that $c \cdot n$ is the exact bound. The most recent results indicate that c lies between 1.6 [6] and 0.94 [18]. The following fact shows how this number plays a role for the number of possible reductions via $\succ$ and thus for the number of duplication roots.

Lemma 7. *Let w be a word with period k . Then all applications of rules from $\succ_k$ will result in the same word, i.e. $\{u : w \succ_k u\}$ is a singleton set.*

As a consequence of this, the number of distinct descendants of w with respect to $\succ$ is equal to the number of runs in w . In this way, the number of runs seems to play an important role for the computation of the maximal number of duplication roots.

To obtain a first approximation on this number, let us state the following: the number of runs in a string of length n is bounded linearly by the string's length. Reducing one square leaves the word's length in general in the order of n , thus also the number of runs is again in the order of n .

On the other hand, every reduction via $\succ$ removes at least one letter, thus there can be at most $n - 1$ steps in the reduction of a word of length n . More precisely, observe that deleting one half of a square cannot remove all copies of a letter from a given string. Thus all roots of a word over three letters have at least three letters themselves. Overall, there are up to $n - 3$ times up to n choices for reducing squares, and the number of different reduction paths lies in $\mathcal{O}(n^n)$. Using the upper bound on the number of runs we see that

$$\text{duproots}(n) \leq (1.6n)^{n-3}.$$

Of course, this gives a very rough upper bound. Most importantly, it disregards the fact that many reductions starting in different points will converge again at some point. Obviously, two rule applications in factors that do not overlap can be applied in either order with identical result. Further, not all of the strings reachable during a reduction will reach the maximum number of runs.

We recall a result from [14].

Lemma 8. *If for two words $u, v \in \Sigma^*$ we have $\text{seq}(u) = \text{seq}(v)$, then also $\sqrt[n]{u} = \sqrt[n]{v} = \sqrt[n]{\text{seq}(u)}$.*

This means that we can first do all the possible reductions of the form $x^2 \rightarrow x$ for single letters x . So for possible splits to different duplication roots we can assume that at least two letters are deleted in every step. Actually, also the fact that u from Example 3 is the shortest possible word with at least two distinct duplication roots shows that we need only consider applications of rules $u^2 \rightarrow u$ with $|u| \geq 2$. This lowers our upper bound to $(1.6n)^{\frac{n-3}{2}}$.

The improvement is not substantial, however. In the initial approach, in some sense all possible paths from w to words in $\sqrt[n]{w}$ in the Hasse diagram of the partial order $[w \succ, \succ^*]$ are counted. The improved version counts only the paths starting from $\text{seq}(w)$ as depicted in Figure 1. The optimal case, however would be to count only one path per element of $\sqrt[n]{w}$. We can take another step into this direction for the partial order $[w \succ^{\leq k}, \succ^{\leq k}]$. As exemplary value for k we choose 30, the reason for this will become evident in the next section.

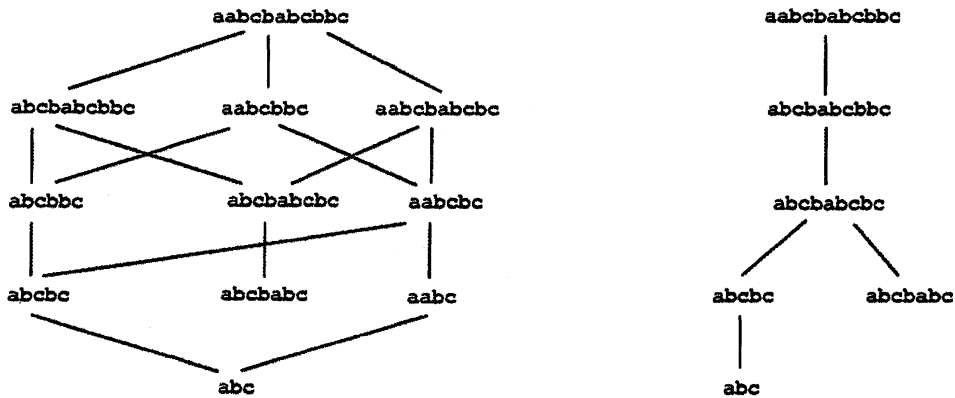


Figure 1: 10 versus 2 paths for the word $aabcbabcbbc$, by first reducing one-letter squares from left to right. The direction of reductions is top to bottom.

Lemma 6 characterizes the words, from which it may not be possible to rejoin outgoing paths. They need to have a critical overlap. The involved squares' bases cannot be longer than 30. Further, one must be shorter than the other, but of length at least two. So for a given square, there are at most 29 such candidates. They can overlap on either side, which gives 56 possible combinations. The shorter square must have more than one half of its length inside the other, and at least one letter must be outside the other. So for a square of length m we have $m - 1$ possible positions. The overall number of possible configurations is therefore $2 \sum_{2 \leq i \leq 29} i - 1 = 2 \sum_{1 \leq i \leq 28} i = 812$.

For calculating the number of possible roots of a word w , we now employ the following tactics. Again, we first compute $\text{seq}(w)$. Then we do not follow all possible paths from $\text{seq}(w)$, but rather select one random square. If it does not form part of a pair of critical squares, then we simply reduce it and proceed further with the next square. Otherwise, for all critical pairs we follow also the paths resulting from reducing the possible partners in these pairs. As we have seen, a square of length 30 can form part of at most 812 critical pairs. The length of the paths is subject to the same bound as for $\succ$, and thus we have to follow at most $812^{\frac{n-3}{2}}$ paths, which is the upper bound on $\text{bduproots}_{\leq 30}$.

Clearly, especially the first bound of $(1.6n)^{\frac{n-3}{2}}$ is very far off the real value. Indeed, all roots of a word w are shorter than w unless w is square-free. Let us label w 's letters from the start to the end. We can look at a rule $uu \rightarrow u$ like the deletion of one copy of u , so its labels disappear. Thus every word in $\sqrt[n]{w}$ corresponds to a subset of the set of $|w|$ labels. There are only $2^{|w|}$ such subsets, which gives us a much better upper bound, also independent of the alphabet size. We still have given the construction of our bound, because we feel that it bears potential for improvement even beyond $2^{|w|}$. Intuitively, the linear bound on the number of runs in a string means that they must be distributed rather evenly over the string's length. Further, results like the Theorem of Fine and Wilf suggest that one run can only form a very limited number of pairs of critical squares, so that even in the case of unbounded duplication we should be able to get an average constant bound like the one of 812 for $\succ_{\leq 30}$. By careful analysis of the possibilities, it should be possible to lower the bound even beyond $2^{|w|}$.

3.2 Bounding from Below

The upper bound on the number of duplication roots is very high and raises the question how far from the real number it is. By an example we now establish a lower bound for this number, which is also exponential. Thus it shows that the upper bound is not too bad.

Example 9. We construct an example of a sequence of words w_n , which are simply powers of a word w , namely $w_n := w^n$. The number of roots increases exponentially in n . This is a modification of a construction used earlier to present a simple language with infinite duplication root [14]. We start the construction of w from the word $u = abcbabcbc$; in Example 2 we have seen that the root of u consists of the two words $u_1 = abc$ and $u_2 = abcbabc$. The basic idea is to concatenate copies of u ; in every factor there is the choice of u_1 or u_2 and thus every additional copy of u doubles the number of roots. However, simple concatenation of u would allow further reductions. Therefore we need to modify and separate the different copies of u in ways that prevent the creation of further squares.

The first measure we take is permuting the letters. Let ρ be the morphism, which simply renames letters according to the scheme $a \rightarrow b \rightarrow c \rightarrow a$. Then $\rho(u)$ has the two roots $\rho(u_1)$ and $\rho(u_2)$; similarly, $\rho(\rho(u))$ has the two roots $\rho(\rho(u_1))$ and $\rho(\rho(u_2))$.

We will now use this ambiguity to construct the word w . This word over the four-letter alphabet $\{a, b, c, d\}$ is

$$w = u d \rho(u) d \rho(\rho(u)) d = abcbabcbc \cdot d \cdot bcacbcaca \cdot d \cdot cabacabab \cdot d.$$

Thus the duplication root of w contains among others the three words

$$\begin{aligned} w_a &= abc \cdot d \cdot bca \cdot d \cdot cabacab \cdot d \\ w_b &= abc \cdot d \cdot bcacbc \cdot d \cdot cab \cdot d \\ w_c &= abcbabc \cdot d \cdot bca \cdot d \cdot cab \cdot d, \end{aligned}$$

which are square-free. We now need to recall that a morphism h is called square-free, iff $h(v)$ is square-free for all square-free words v . Crochemore has shown that a uniform morphism h is square-free iff it is square-free for all square-free words of length 3 [5]. Here uniform means that all images of single letters have the same length, which is given in our case.

The morphism we define now is $\varphi(x) := w_x$ for all $x \in \{a, b, c\}$. Thus to establish the square-freeness of φ , we need to check this property for the images of all square-free words up to length 3. These are

$$\begin{aligned}
\varphi(aba) &= abcd bcad cabac abda bcdb cacb cad cabd abcd bcad cabac abd \\
\varphi(abc) &= abcd bcad cabac abda bcdb cacb cad cabd abcb abcd bcad cabd \\
\varphi(aca) &= abcd bcad cabac abda bcdb cacb cad cabd abcd bcad cabac abd \\
\varphi(acb) &= abcd bcad cabac abda bcdb cacb cad cabd abcd bcac bcad cabd \\
\varphi(bab) &= abcd bcac bcad cabd abcd bcad cabac abda bcdb cacb cad cabd \\
\varphi(bac) &= abcd bcac bcad cabd abcd bcad cabac abda bcdb cacb cad cabd \\
\varphi(bca) &= abcd bcac bcad cabd abcb abcd bcad cabd abcd bcad cabac abd \\
\varphi(bcb) &= abcd bcac bcad cabd abcb abcd bcad cabd abcd bcac bcad cabd \\
\varphi(cac) &= abcb abcd bcad cabd abcd bcad cabac abda bcdb cacb cad cabd \\
\varphi(cab) &= abcb abcd bcad cabd abcd bcad cabac abda bcdb cacb cad cabd \\
\varphi(cba) &= abcb abcd bcad cabd abcd bcac bcad cabd abcd bcad cabac abd \\
\varphi(cbc) &= abcb abcd bcad cabd abcd bcac bcad cabd abcb abcd bcad cabd,
\end{aligned}$$

where, of course, the images of all words shorter than three are contained in them. All the twelve words listed here are indeed square-free as an eager reader can check, and thus φ is square-free.

Now let t be an infinite square-free word over the letters a, b and c . Such a word exists [22]. Then all the words in $\varphi(\text{pref}(t))$ are square-free, too. From the construction of φ we know that for any word z of length i we can reach $\varphi(z)$ from w^i by undoing duplications. Therefore $\varphi(\text{pref}(t)) \subseteq \sqrt[i]{w^+}$. For two distinct square-free words t_1 and t_2 , also $\varphi(t_1) \neq \varphi(t_2)$. Finally, notice that for all positive $i \leq n$ we have $w^n \succ^* w^i$.

This means that all square-free words that are not longer than n lead to a different duplication root of w_n . Therefore $\text{bduproots}_{\leq 30} \leq s$, where $s(n)$ is the number of ternary square-free words of length up to n . This function's value is not known, however, it was first bounded to $6 \cdot 1.032^n \leq s(n) \leq 6 \cdot 1.379^n$ by Brandenburg [4]. A better lower bound was found by Sun $s(n) \geq 110^{\frac{n}{42}}$ [21]. w itself is of length $3|u| + 3 = 30$. So we see that $\text{bduproots}_{\leq 30}(n) \geq \frac{1}{30} 110^{\frac{n}{42}}$.

Example 9 leaves room for improvement in several respects.

- The word w is over a four-letter alphabet. The letter d is used to separate the different blocks that introduce the ambiguities and only use the alphabet $\{a, b, c\}$. The question is whether this function can also be fulfilled by an appropriate word over $\{a, b, c\}$; computer experiments with candidate words have always led to unwanted squares with some of the adjoining factors.
- The ambiguity of u that we use is only two-fold. Using the words w_3 and w_5 from Example 2, it might be possible to pack more choices into less room and thus improve the initial constant of $\frac{1}{5}$ with similar constructions. However, this would not change the magnitude. On the other hand, the resulting morphism would not be uniform, which would complicate the establishment of its square-freeness.

Summarizing this section up to this point, we have the following bounds for the function duproots .

Proposition 10. $\frac{1}{30} 110^{\frac{n}{2}} \leq \text{duproots}(n) \leq 2^n$ for all $n > 0$.

Because Example 9 uses only rules from $\succ_{\leq 30}$, its bound holds also for $\text{bduproots}_{\leq 30}$. So for this function we get a much sharper characterization of its growth.

Proposition 11. $\frac{1}{30} 110^{\frac{n}{2}} \leq \text{bduproots}_{\leq 30}(n) \leq \max\{812^{\frac{n-3}{2}}, 2^n\}$ for all $n > 0$.

While this upper bound is still enormous, we have at least achieved a bounding between two exponential functions. So for this case the bounds are much tighter, though still rather loose. For ternary alphabet, the upper bound $6 \cdot 1.379^n$ by Brandenburg can replace 2^n in both Propositions.

3.3 Computing the Number of Duplication Roots

Proposition 10 shows that the straight-forward approach to computing the function duproots will lead to exponential runtime. But it seems reasonable to assume that it is not necessary to actually compute the set $\sqrt[n]{w}$ to determine its size. Example 3 suggests that it suffices to identify the number of critical overlaps in the original word. In this case, even linear time might suffice. However, it remains to show that no new critical pairs can come up during a reduction, or at least that their number can be foreseen by looking at the original word.

References

- [1] BENSON, D. A. Tandem Repeat Finder: A Program to Analyze DNA Sequences. *Nucleic Acids Research* 27, 2 (1999), 573–580.
- [2] BOOK, R., AND OTTO, F. *String-Rewriting Systems*. Springer, Berlin, 1993.
- [3] BOVET, D. P., AND VARRICCHIO, S. On the Regularity of Languages on a Binary Alphabet Generated by Copying Systems. *Information Processing Letters* 44, 3 (1992), 119–123.
- [4] BRANDENBURG, F.-J. Uniformly Growing k -th Power-Free Homomorphisms. *Theor. Comput. Sci.* 23 (1983), 69–82.
- [5] CROCHEMORE, M. Sharp Characterizations of Squarefree Morphisms. *Theoretical Computer Science* 18 (1982), 221–226.
- [6] CROCHEMORE, M., AND ILIE, L. Maximal repetitions in strings. *Journal of Computer and System Sciences* 74, 5 (2008), 796–807.
- [7] DASSOW, J., MITRANA, V., AND PĂUN, G. On the Regularity of Duplication Closure. *Bulletin of the EATCS* 69 (1999), 133–136.
- [8] EHRENFEUCHT, A., AND ROZENBERG, G. On the Separating Power of EOL Systems. *RAIRO Informatique Théorique* 17, 1 (1983), 13–22.
- [9] HARRISON, M. A. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachusetts, 1978.

- [10] ILIE, L., YU, S., AND ZHANG, K. Repetition Complexity of Words. In: *COCOON* (2002), O. H. Ibarra and L. Zhang, Eds., vol. 2387 of *Lecture Notes in Computer Science*, Springer, pp. 320–329.
- [11] ILIE, L., YU, S., AND ZHANG, K. Word Complexity And Repetitions In Words. *Int. J. Found. Comput. Sci.* 15, 1 (2004), 41–55.
- [12] ITO, M., LEUPOLD, P., AND SHIKISHIMA-TSUJI, K. Closure of Language Classes Under Bounded Duplication. In: *Developments in Language Theory* (2006), O. H. Ibarra and Z. Dang, Eds., vol. 4036 of *Lecture Notes in Computer Science*, Springer, pp. 238–247.
- [13] KOLPAKOV, R. M., AND KUCHEROV, G. Finding Maximal Repetitions in a Word in Linear Time. In: *FOCS* (1999), pp. 596–604.
- [14] LEUPOLD, P. Duplication Roots. In: *Developments in Language Theory* (2007), T. Harju, J. Karhumäki, and A. Lepistö, Eds., vol. 4588 of *Lecture Notes in Computer Science*, Springer, pp. 290–299.
- [15] LEUPOLD, P., MARTÍN-VIDE, C., AND MITRANA, V. Uniformly bounded duplication languages. *Discrete Applied Mathematics* 146, 3 (2005), 301–310.
- [16] LEUPOLD, P., AND MITRANA, V. Uniformly bounded duplication codes. *Theor. Inform. Appl.* 41, 4 (2007), 411–424.
- [17] LEUPOLD, P., MITRANA, V., AND SEMPERE, J. M. Formal Languages Arising from Gene Repeated Duplication. In: *Aspects of Molecular Computing* (2004), N. Jonoska, G. Paun, and G. Rozenberg, Eds., vol. 2950 of *Lecture Notes in Computer Science*, Springer, pp. 297–308.
- [18] MATSUBARA, W., KUSANO, K., ISHINO, A., BANNAI, H., AND SHINOHARA, A. New Lower Bounds for the Maximum Number of Runs in a String. In: *Proceedings of the Prague Stringology Conference 2008* (Czech Technical University in Prague, Czech Republic, 2008), J. Holub and J. Žďárek, Eds., pp. 140–145.
- [19] PENNISI, E. MOLECULAR EVOLUTION: Genome Duplications: The Stuff of Evolution? *Science* 294, 5551 (2001), 2458–2460.
- [20] SOKOL, D., BENSON, G., AND TOJEIRA, J. Tandem repeats over the edit distance. *Bioinformatics* 23, 2 (2007), 30–35.
- [21] SUN, X. New Lower Bound On The Number of Ternary Square-Free Words. *Journal of Integer Sequences* 6, 3 (2003), 1–8.
- [22] THUE, A. Über die gegenseitige Lage gleicher Teile verschiedener Zeichenreihen. *Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (Kristiania)* 1 (1912), 1–67.
- [23] WANG, M.-W. On the Irregularity of the Duplication Closure. *Bull. EATCS* 70 (2000), 162–163.
- [24] WAPINSKI, I., PFEFFER, A., FRIEDMAN, N., AND REGEV, A. Natural History and Evolutionary Principles of Gene Duplication in Fungi. *Nature* 449 (2007), 54–61.