International Journal of Foundations of Computer Science

On the Prefix–Suffix Duplication Reduction*

Szilárd Zsolt Fazekas†
Akita University, Department of Mathematical Science and Electrical-Electronic-Computer Engineering, Japan
szilard.fazekas@gmail.com

Robert Mercaş‡
Loughborough University, Department of Computer Science, UK
R.G.Mercas@Lboro.ac.uk

Daniel Reidenbach§
Loughborough University, Department of Computer Science, UK
D.Reidenbach@Lboro.ac.uk

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

This work answers some questions proposed by Bottoni, Labella, and Mitrana (*Theoretical Computer Science* 682, 2017) regarding the prefix–suffix reduction on words. The operation is defined as a reduction by one half of every square that is present as either a prefix or a suffix of a word, leading thus to a finite set of words associated to the starting one. The iterated case considers consecutive applications of the operations, on all the resulting words. We show that the classes of linear and context-free language are closed under iterated bounded prefix–suffix square reduction, and that for a given word we can determine in $\mathcal{O}(n^2 \log n)$ time all of its primitive prefix–suffix square roots.

1. Introduction

Repetitions constitute one of the building blocks of combinatorics on words and stringology. They are represented as consecutive occurrences of the same factor, and their presence is widely recognized in different aspects of life, ranging from biology to mathematics, music to engineering. Furthermore, their study is also the stepping stone in the area of combinatorics on words, being firstly investigated by Thue at the beginning of the previous century [17, 18].

*We kindly thank our anonymous referees that helped us improve and fix our initial version of this work.
†Akita University
‡Loughborough University
§Loughborough University

The duplication, or copy, operation on words and languages was introduced and investigated independently in a series of papers [9, 2, 5, 14, 19]. The operation considers duplicating any of the factors of a word, and is easily extendable to languages, by considering all words in the given language. To this end, closure properties were studied for languages equipped with such an operation in the case of one application of the operation, the iterative case, and in certain bounded settings (see also [13, 11]).

An operation complementary to the above, namely duplication reduction was also introduced and studied already more than ten years ago [12].

Recently, in [10], the authors introduced a new similar operation that extends words to the left and to the right by doubling either a prefix or a suffix of them. As one can easily see, this operation is a restriction of the above, where the operation is allowed only at the ends of words. The bounded case of this operation constitutes a topic of investigation of [6]. A similar operation is that defined in [7] where the authors investigate the algorithmic and combinatorial aspects of the prefix–suffix duplication operation, when a part of the extended second half already exists (that is $xyxw$ is extended to $yxyxw$, while $wxyx$ is extended to $wxyxy$).

Following the previous pattern, an investigation of the operation dual to the prefix–suffix duplication is initiated in [1]. The investigation in [1] is strongly related to the algorithmic investigation presented in [8], where the authors consider all square-free factors of a word. Our present work has in mind this latter operation of prefix–suffix square reduction and tries to answer some of the questions that were proposed in [1].

Our work is structured as follows. After some preliminary notation and definitions presented in the following section, Section 3 looks at the closure of linear and context-free languages under the bounded iterated reduction of prefixes and suffixes of words. Section 4 provides a $\mathcal{O}(n^2 \log n)$ time algorithm that identifies all primitive prefix-suffix square roots (irreducible factors under this operation) of a given word. We end with a concluding section where we propose a new conjecture regarding the prefix–suffix reduction roots.

2. Preliminaries

We assume the reader to be familiar with general aspects of formal languages and complexity theory (for more details see, e.g., [15, 16]). Next we present some of the most important notion that we will use throughout this work. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a set V is denoted by $|V|$. A *word* w over an alphabet V represents any finite sequence of symbols from V. The *empty word* associated to any alphabet V is denoted by ε. The set of all finite words over V is denoted by V^*, while by V^+ we denote the set of all non-empty finite words. For a positive integer ℓ, we define by $V^{\leq \ell}$ the set of all words of length at most ℓ, and by V^ℓ the set of all words of length precisely ℓ. A language L is a set of words over V, i.e., $L \subseteq V^*$.

For a word w we denote by $|w|$ its length, that is the number of all (not necessarily distinct) symbols in w. By $alph(w)$ we denote the set of symbols from w, i.e., the minimal alphabet necessary in order to obtain w. By extension, $alph(L) = \bigcup_{x \in L} alph(x)$. The concatenation between two words w and v is denoted by wv and represents the extension of the word w by the word v. For all words w, x, v, y, with $w = xvy$, we call v a factor of w. Whenever x is empty v is a prefix ($v \leq_{\text{pref}} w$) of w, and when y is empty v is a suffix ($y \leq_{\text{suff}} w$) of w. For any integers i, j with $0 < i \leq j \leq |w|$, by $w[i..j]$ we denote the factor of w starting at position i and ending at position j. As a convention, whenever $i > j$, $w[i..j] = \varepsilon$, and, for $i = j$, $w[i..j] = w[i]$ is a letter. A word w is called a repetition if there exist a word v and an integer i such that $w = v^i$, that is w is represented by i concatenations of v. If $w = v^i$ is true only for $i = 1$, then w is called primitive.

For a word w over an alphabet V, in [1], the following operations are defined:

- prefix square reduction: $\sqsubset(w) = \{uv \mid w = uuv, \text{ for } u \in V^+ \text{ and } v \in V^*\}$
- suffix square reduction: $\sqsupset(w) = \{vu \mid w = vuu, \text{ for } u \in V^+ \text{ and } v \in V^*\}$
- prefix–suffix square reduction: $\sqsubset\!\!\sqsupset(w) = \sqsubset(w) \cup \sqsupset(w)$

We can easily extend the above operation to languages $\sqsubset\!\!\sqsupset(L) = \bigcup_{w \in L} \sqsubset\!\!\sqsupset(w)$, as well as define an iterated version of it:

$$\begin{aligned} \sqsubset\!\!\sqsupset^0(w) &= \{w\}, \\ \sqsubset\!\!\sqsupset^{k+1}(w) &= \sqsubset\!\!\sqsupset(\sqsubset\!\!\sqsupset^k(w)), \text{ for any } k \geq 0 \\ \sqsubset\!\!\sqsupset^*(w) &= \bigcup_{k \geq 0} \sqsubset\!\!\sqsupset^k(w). \end{aligned}$$

Furthermore, for a fixed positive integer p, one can define the bounded versions of the operation:

- p-prefix square reduction: ${}_p\sqsubset(w) = \{uv \mid w{=}uuv, \text{ for } u \in V^{\leq p} \text{ and } v \in V^*\}$
- p-suffix square reduction: $\sqsupset_p(w) = \{vu \mid w{=}vuu, \text{ for } u \in V^{\leq p} \text{ and } v \in V^*\}$
- p-prefix–suffix square reduction: $\sqsubset\!\!\sqsupset_p(w) = {}_p\sqsubset(w) \cup \sqsupset_p(w)$

These operations are naturally extended to languages similarly to the unbounded case. Furthermore, the iterated version of the bounded prefix–suffix square reduction is defined similarly to the unbounded case. As in the case of the unbounded square reductions defined above, each form of bounded square reduction is actually the inverse of the corresponding form of bounded duplication introduced in [6].

To this end, for a word w, we denote by $\sqrt[\sqsubset]{w} = \{u \mid u \in \sqsubset^*(w) \text{ and } \sqsubset(u) = \emptyset\}$ the primitive prefix square roots for the word w. In other words, this represents all words that are obtained from w by consecutive prefix reductions, such that they themselves are irreducible, i.e., they cannot be further reduced. Moreover, we denote by $\sqrt[\sqsupset]{w}$ and $\sqrt[\sqsubset\!\!\sqsupset]{w}$ the corresponding primitive suffix square roots and the primitive prefix–suffix square roots of w, respectively. The same notion, can easily

be extended to languages, $\sqrt[\square\!\square]{L} = \bigcup_{w \in L} \sqrt[\square\!\square]{w}$ (or should be necessary to the bounded cases, but these do not make the object of this investigation).

3. Roots and Bounded Iteration

One of the first questions proposed by the authors in [1] regards the regularity of the set of primitive unbounded prefix–suffix square roots of a regular language. In particular, the authors conjecture that the set of primitive unbounded prefix–suffix square roots of a regular language is still regular if and only if it is finite.

We show that the primitive unbounded prefix–suffix square roots of regular languages can determine finite, infinite regular or non-regular languages.

Let $Q_{\square\!\square}$ be the language of all prefix–suffix square-free words over a given alphabet V.

Now take, for instance, the infinite regular language $L = ab^+a$. It is clear that $\sqrt[\square\!\square]{L} = L$, because all words in L are primitive w.r.t. prefix–suffix square reduction ($L \cap Q_{\square\!\square} = L$); thus the statement "the set of primitive unbounded prefix–suffix square roots of a regular language is still regular if and only if it is finite" does not hold.

Next consider the language LL. In this language, the only words which are not prefix–suffix square-free are the ones of the form $(ab^na)^2$, for $n \geq 1$. This means that $\sqrt[\square\!\square]{LL} = \{ab^maab^na \mid m, n \geq 1 \text{ and } m \neq n\} \cup ab^+a$, which can be shown not to be regular by a simple application of the pumping lemma to

$$ab^+aab^+a \setminus \sqrt[\square\!\square]{LL} = \{ab^naab^na \mid n \geq 1\}.$$

The above can be summarized by the following statement.

Proposition 1. *There exist regular languages whose set of primitive prefix–suffix square roots is not regular. Furthermore, there exist infinite regular languages whose set of primitive prefix–suffix square roots is regular and infinite.*

The regularity of the prefix–suffix square reduction of a regular language was shown in [1] by explicitly constructing an NFA for the reduced language. For a slightly more bird's-eye view, however, we can explain it through means of language operations without going into the details of the automata construction. To simplify the argument, we only consider prefix square reduction here.

Let us assume that the starting regular language is given by the DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where δ is the transition function extended to words in the usual way, i.e., $\delta(p, \varepsilon) = p$ and $\delta(p, wa) = \delta(\delta(p, w), a)$, for all $p \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$. Now consider the languages $L_{p,q}$ which contain the words taking $\mathcal{A}$ from state p to state q, i.e., $L_{p,q} = \{w \mid \delta(p, w) = q\}$. Then, the language

$$\mathrm{SqPref}_q = \bigcup_{p \in Q} (L_{q_0,p} \cap L_{p,q})$$

consists of all square prefixes which take $\mathcal{A}$ to state q; so we get that the square prefix reduction of $L(\mathcal{A})$ is

$$\square\!\!\!\!\square\,(L(\mathcal{A})) = \bigcup_{q\in Q, f\in F} \left(\mathrm{SqPref}_q \cdot L_{q,f}\right),$$

which is a finite union of concatenations of regular languages, hence regular.

This observation might help in characterizing the iterated unbounded prefix–suffix square reduction of regular languages, but, unfortunately, in this paper we are unable to establish any stronger results on the unbounded case. Instead, we shall continue with some insights into the less complex iterated bounded reductions.

For a word of the form $uuv = xyy$ we say that the square prefix uu *overlaps* with the square suffix yy, if $|u| > |x|$ or $|y| > |v|$. In the former case reducing the prefix uu results in the word uv, which is shorter than yy, so the subsequent reduction of the suffix yy is not possible, whereas in the latter case, reducing the suffix yy first makes the subsequent reduction of the prefix uu impossible. In the case of bounded reduction, long enough words avoid the problems of overlapping squares. Let $w' \in \square\!\!\!\!\square_p^2(w)$ and $|w'| \geq 2p$, such that $w = uw'y$, for $u, y \in \Sigma^{\leq p} \setminus \{\lambda\}$, that is we obtained w' from w by reducing the p-bounded square prefix uu and p-bounded square suffix yy. Because of the length lower bound on w', we have $w' = uzy$, for some possibly empty word z, hence $w = uuzyy$, meaning that the square prefix uu does not overlap with the square suffix yy in w.

Theorem 2. *The classes of context-free languages and linear languages are closed under iterated bounded prefix–suffix square reduction.*

Proof. As each context-free language can be accepted by a PDA which reads a letter from the tape in each step, suppose we are given such a PDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$. The proof goes by constructing a PDA $\mathcal{B} = (Q', \Sigma, \Gamma, \delta', q_0, F')$ which guesses and simulates $\mathcal{A}$ on the square prefixes that have been reduced, until it decides that no more prefixes should be reduced; then it reads w and finally guesses and reads the reduced suffixes, one by one.

The simulation is very similar to the regular case, but for the following technical detail: the simulation of prefix–suffix reduction in finite automata was done by reading the whole guessed prefix or suffix at once and transitioning to the state obtained, whereas here we read the letters of the guessed prefixes and suffixes one at a time.

In what follows, p is the length bound for reduced prefixes and suffixes, as in the definition of $\square\!\!\!\!\square_p$. To avoid treating the complicated cases, we restrict ourselves to recognizing the words of $\square\!\!\!\!\square_p^*(L(\mathcal{A}))$ which can be obtained from a word in $L(\mathcal{A})$ without reducing overlapping prefix and suffix squares. This includes all words of length at least $2p$ of the language $\square\!\!\!\!\square_p^*(L(\mathcal{A}))$ per the explanation before the theorem, so

$$\square\!\!\!\!\square_p^*(L(\mathcal{A})) \setminus \Sigma^{<2p} \subseteq L(\mathcal{B}) \subseteq \square\!\!\!\!\square_p^*(L(\mathcal{A})).$$

Further, $\square\!\!\square_p^*(L(\mathcal{A}))$ is (linear) context-free if and only if $L(\mathcal{B})$ is (linear) context-free if and only if $\square\!\!\square_p^*(L(\mathcal{A}))\setminus\Sigma^{<2p}$ is (linear) context-free, since the differences between any two of the three languages are finite and it is well known that subtracting a finite language from a (linear) context-free one, maintains the (linear) context-freeness of the result.

In order to simplify the argument for linear context-free languages we make the following claim. The newly constructed PDA pushes and pops symbols from the stack exactly when the original would, therefore if $\mathcal{A}$ is a one-turn PDA, then the new one will be, too. This means that if the starting language is linear, the iterated reduction will be so, as well.

The state set of $\mathcal{B}$ is $Q' = Q \cup \big(Q \times (\Sigma^{\leq p} \cdot \{\square, \$, \boxminus\} \cdot \Sigma^{\leq p})\big)$, where $\square, \$, \boxminus \notin \Sigma$. To improve legibility we dispense here with the $(state, push) \in \delta(state, read, pop)$ notation and write instead the compact form

$$state, read, pop \longrightarrow state, push.$$

The transitions δ' are of the following types (for all $q \in Q$):

(1) Guess the first prefix x to reduce.
$\forall x \in \Sigma^{\leq p}$:

$$q_0, \lambda, \lambda \longrightarrow \binom{q_0}{x\,\square}, \lambda$$

(2) Guess the factor y following x, such that x is reducible (prefix of y).
$\forall x, y \in \Sigma^{\leq p}$ with $x \leq_{pref} y$:

$$\binom{q}{x\,\square}, \lambda, \lambda \longrightarrow \binom{q}{x\,\square\,y}, \lambda)$$

(3) Read the letters of the prefix being reduced and simulate the transitions of $\mathcal{A}$ on the letters read.
$\forall q' \in Q$, $a \in \Sigma$, $b \in \Gamma$, $c \in \Gamma^*$, $x \in \Sigma^{\leq p-1}$ and $y \in \Sigma^{\leq p}$ such that $(q', c) \in \delta(q, a, b)$:

$$\binom{q}{ax\,\square\,y}, \lambda, b \longrightarrow \binom{q'}{x\,\square\,y}, c$$

(4) Guess the next prefix to reduce.
$\forall x \in \Sigma^+$, $y, z \in \Sigma^*$ such that $x \leq_{pref} yz$ and $|xy| \leq p$, $1 \leq |yz| \leq p$:

$$\binom{q}{\square\,xy}, \lambda, \lambda \longrightarrow \binom{q}{x\,\square\,yz}, \lambda$$

(5) Finish guessing prefixes.
$\forall x \in \Sigma^{\leq p}$:

$$\binom{q}{\square\,x}, \lambda, \lambda \longrightarrow \binom{q}{\$x}, \lambda$$

(6) Read the letters of the last reduced prefix and match them to the starting portion of our input as to check if the reduction can happen.
$\forall q' \in Q$, $a \in \Sigma$, $b \in \Gamma$, $c \in \Gamma^*$ and $y \in \Sigma^*$ such that $ay \in \Sigma^{\leq p}$ and $(q', c) \in \delta(q, a, b)$:

$$\binom{q}{\$ay}, a, b \longrightarrow \binom{q'}{\$y}, c$$

(7) Read the letters of the input which are not part of the reduced square prefixes or suffixes.
$\forall a \in \Sigma$, $b \in \Gamma$ and $c \in \Gamma^*$ such that $(q', c) \in \delta(q, a, b)$:

$$\binom{q}{\$}, a, b \longrightarrow \binom{q'}{\$}, c$$

We proceed similarly for guessing and reading the suffixes, so we add the following transitions (for all $q \in Q$):

(8) Read and remember the suffix x of length p of the input word, by adding more letters as long as the factor read is shorter than p.
$\forall x \in \Sigma^{\leq p-1}$, $a \in \Sigma$:

$$\binom{q}{x\$}, a, \lambda \longrightarrow \binom{q}{xa\$}, \lambda$$

(9) Transition to guessing new suffix mode.
$\forall x, y \in \Sigma^{\leq p}$ with $y \leq_{suff} x$:

$$\binom{q}{x\$}, \lambda, \lambda \longrightarrow \binom{q}{x \boxminus y}, \lambda$$

(10) Read the letters of the guessed suffixes and simulate $\mathcal{A}$ on them, while always remembering the last p letters read.
$\forall q' \in Q$, $a \in \Sigma$, $b \in \Gamma$, $c \in \Gamma^*$, $x \in \Sigma^{\leq p-1}$ and $y \in \Sigma^{\leq p}$ such that $(q', c) \in \delta(q, a, b)$:

$$\binom{q}{ax \boxminus y}, \lambda, b \longrightarrow \binom{q'}{x \boxminus y}, c$$

(11) Guess another reduced suffix.
$\forall x, y \in \Sigma^{\leq p}$ with $|xy| = p$ and $z \leq_{suff} xy$:

$$\binom{q}{x \boxminus y}, \lambda, \lambda \longrightarrow \binom{q}{xy \boxminus z}, \lambda$$

(12) Stop guessing suffixes.
$\forall x, y \in \Sigma^{\leq p}$:

$$\binom{q}{x \boxminus y}, \lambda, \lambda \longrightarrow \binom{q}{xy}, \lambda$$

(13) Finish simulating $\mathcal{A}$ on the remaining letters of the last guessed suffix. $\forall q' \in Q, a \in \Sigma, x \in \Sigma^{\leq p-1}, b \in \Gamma, c \in \Gamma^*$ such that $(q', c) \in \delta(q, a, b)$:

$$\binom{q}{ax}, \lambda, b \longrightarrow \binom{q'}{x}, c$$

The set of final states is $F' = \{\binom{q}{\lambda} \mid q \in F\}$. If we reached the final state and we read the whole input w, it means that the word w' that we guessed was in $L(\mathcal{A})$, and w can be obtained from w' by p-bounded prefix-suffix reductions, so $w \in \sqsubset\sqsupset_p^*(L(\mathcal{A}))$.

As mentioned earlier, if $\mathcal{A}$ is one-turn, then $\mathcal{B}$ is one-turn, too: the only pushdown operations are in transitions 3, 6, 7, 10 and 13 and in all cases they mimic a pushdown operation of $\mathcal{A}$.

As the simulation is rather technical, we illustrate the working of $\mathcal{B}$ on an input $u_m wz$ which was obtained through iterated bounded prefix–suffix square reduction from a word $u_1 \cdots u_m u_m w z v_1 \cdots v_n \in L(\mathcal{A})$, where $|z| = p$ and all $u_i, v_j \in \Sigma^{\leq p}$, and $u_i \leq_{pref} u_{i+1} \cdots w$ and $v_j \leq_{suff} z v_1 \cdots v_{j-1}$:

- by transitions type 1, we enter prefix reduction mode ($\sqsubset$); we guess u_1 and put it on the "tape" simulated in the second component of the states: that is $u_1 \sqsubset$;
- by type 2, we guess a prefix y of $u_2 \cdots$ such that $u_1 \leq_{pref} y$ and put it on the simulated tape after the left-reduction symbol: that is $u_1 \sqsubset y$;
- whenever we have $ax \sqsubset u_i \cdots$ on the simulated tape, by type 3 we simulate the state transition and pushdown operations of $\mathcal{A}$ on reading a, and erase a: that is $x \sqsubset u_i \cdots$;
- when all letters of u_i have been read ($\sqsubset u_{i+1} \cdots$), by type 4 we guess u_{i+1}, put it on the left side of $\sqsubset$ and guess a factor which follows it, so that u_{i+1} is reducible: that is $u_{i+1} \sqsubset u_{i+2} \cdots$;
- after reading the letters of the first copy of u_m, we have $\sqsubset u_m$ on the simulated tape. By type 5, we enter a new operating mode $\sqsubset \to \$$ and we match the letters on the simulated tape against the letters of u_m on the actual tape, by type 6, while simulating the state transitions and pushdown operations of $\mathcal{A}$;
- after we finished reading u_m, we read w from the tape and simulate $\mathcal{A}$ on it by type 7;
- by type 8, we read and put z on the simulated tape: that is $z\$$;
- by type 9, we change to suffix guessing mode ($\$ \to \sqsupset$) and we guess v_1: that is $z \sqsupset v_1$;
- we keep guessing v_i and simulating $\mathcal{A}$ on their letters by types $10 - 13$, similarly to the prefix case. □

4. Algorithms

In this section we study an algorithmic problem concerning the primitive prefix–suffix square roots of a word. In particular, we investigate the problem when we are given a word w and we need to find all of its primitive prefix–suffix square roots. Easier versions of this problem, such as investigating the complexity of deciding whether a word has a unique primitive prefix–suffix square root, as well as finding its shortest and/or longest primitive prefix–suffix square root, otherwise, were suggested in [1] (these results can be derived in a straightforward fashion from our solution). The authors suggested, upon raising these problems, that employing the results and techniques from [8] should lead to a solution, aiming this way for $\mathcal{O}(n^2 \log n)$ time solutions.

To this end, we mention that a remark from Example 1 in [8] shows that the number of square-free factors in a word can be quadratic with respect to the length of a word. In spite of the fact that this result does not mean that the number of primitive prefix–suffix square roots of a given word is quadratic with respect to the word length, it gives us a good starting point for the investigation. In this section we confirm the intuition of the authors from [1] by providing a simple $\mathcal{O}(n^2 \log n)$ dynamic programming solution to the problem of finding all primitive prefix–suffix square roots of a word (these results can also be deduced in a not too difficult of a manner following the approach from [10]).

The following combinatorial results are well known (see, e. g., [4]). We note that, the notion of primitivity of words mostly used in this section is the classical one of words that are not expressible as several concatenations of the same factor.

Lemma 3 ([3]) *Let u_1, u_2, and u_3 be primitive words, with $|u_1| < |u_2| < |u_3|$ and u_i^2 prefixes (suffixes) of a word w, for every i with $1 \leq i \leq 3$. Then $2|u_1| < |u_3|$ and, as a consequence, $|\{u | u \text{ primitive, } u^2 \text{ prefix (respectively, suffix) of } w\}| \leq 2 \log |w|$.*

Assume that $w \in \Sigma^*$ is of length n. For each i with $1 \leq i \leq n$ we define the set

$$P_i = \{u \mid u \text{ is a primitive word such that } u^2 \text{ is a prefix of } w[i..n]\}.$$

Lemma 3 shows that $|P_i| \leq 2 \log n$ for every i with $1 \leq i \leq n$. Generally, we can represent the elements of P_i in various efficient manners. For our purpose, for each $u \in P_i$ it is enough to store its length. For valid i, k we shall use $P_i[k]$ to denote both the kth primitive word whose square starts at position i, as well as its length.

Lemma 4 ([3]) *Let $w \in \Sigma^*$ be a word of length n. We can compute in $\mathcal{O}(n \log n)$ time all the sets P_i associated to w, where $i \in \{1, 2, \ldots, n\}$.*

Note that in [3] there are examples of words of length n for which $\sum_{i \leq n} |P_i| \in \Theta(n \log n)$. On the same page, we denote by S_i the set corresponding to suffixes.

Before stating the main result of this section, we make the observation that, for any primitive prefix–suffix square root w' of w, there exists another (non-primitive) prefix–suffix square root w'' of w, such that $w'' \in \square\!\!+\!\!\square^*(w)$ and $w' \in \square\!\!+\!\!\square(w'')$ (w' is

obtained in a single step reduction from w'').

Theorem 5. *Given a word w of length n, we can compute in $\mathcal{O}(n^2 \log n)$ time all the primitive prefix–suffix square roots of w.*

Proof. Following Lemma 4, in $\mathcal{O}(n \log n)$ time we can find all sets P_i and S_i corresponding to every position i in w, where $1 \leq i \leq n$. We recall that by an element k in any of the sets P_i and S_i we refer to both the length of a primitive word that is a square starting at position i, respectively ending at position j, as well as the primitive word itself.

Next, we construct an $n \times n$ matrix R, where for every position $[i, j]$ in the matrix, where $0 \leq i, j < n$, we mark if $w[i..j]$ is a primitive root, a (non-primitive) root, or it is not reachable, i.e., $w[i..j] \notin \square\!\!\square^*(w)$. To put this formally, we have:

$$\mathrm{R}[i,j] = \begin{cases} 1, w[i..j] \in \square\!\!\square^*(w), P_i[1] > \frac{j-i}{2} \text{ or } P_i = \emptyset, \text{ and } S_j[1] > \frac{j-i}{2} \text{ or } S_j = \emptyset; \\ 0, w[i..j] \in \square\!\!\square^*(w) \text{ and } P_i[1] \text{ or } S_j[1] \leq \frac{j-i}{2}; \\ -1, w[i..j] \notin \square\!\!\square^*(w). \end{cases}$$

In particular, for every pair of positions we check (dynamically) if there exists a prefix–suffix square reduction from the initial word that renders the corresponding factor. If, furthermore, no prefix–suffix square reduction can be performed on this factor, we conclude that the factor represents a primitive prefix–suffix square root of the word, hence marking it by 1. For any pair $[i, j]$ we can obviously disregard the factor corresponding to it when $\mathrm{R}[i, j] = -1$, as no prefix–suffix reduction can derive it from the original word. Next, we present our algorithm:

Algorithm 6. ***Find prefix-suffix roots***

```
initialize all position of R with −1;
R[0, n − 1] = 0;
for i := 0 to n − 1 do
  for j := n − 1 downto i do
    if R[i, j] == 0 then
      if (P_i == ∅ or P_i[1] > (j−i)/2) and (S_j == ∅ or S_j[1] > (j−i)/2) then
        R[i, j] = 1
      else
        for k ∈ P_i do
          if i + 2k ≤ j + 1 then
            R[i + k, j] = max(R[i + k, j], 0)
        for k ∈ S_j do
          if j − 2k ≥ i − 1 then
            R[i, j − k] = max(R[i, j − k], 0)
return R
```

Since the sizes of the sets P_i and S_j are in $\mathcal{O}(\log n)$, the complexity of the

algorithm is straightforward. In particular, for none of the $\mathcal{O}(n^2)$ pairs $[i, j]$ are we updating more than $\mathcal{O}(\log n)$ positions in R.

When looking at its correctness, we see that by considering all possible reductions in lines 9–14 of our Algorithm, we ensure also that the cases when one has to interlap prefix and suffix reductions, all possibilities are considered. Our algorithm correctly assigns value 1 to every pair of position $[i, j]$ whenever the factor $w[i, j]$ can be obtained from w by iterated prefix–suffix reductions, and, moreover, the factor does not have any square prefixes or suffixes, of a length shorter than its, that allow further reductions, i.e., making it therefore a primitive prefix–suffix square root of the word (lines 6–7 of the Algorithm). □

As a direct consequence of Theorem 5 the following result is straightforward.

Corollary 7. *We can decide in $\mathcal{O}(n^2 \log n)$ time whether a given word of length n has a unique primitive prefix–suffix square root, a unique shortest or a unique longest primitive prefix–suffix square root. Moreover, in the same time complexity we can compute the set of all shortest or longest primitive prefix–suffix square roots of a word.*

5. Concluding Remarks

We conclude this paper with a few more words about the set of prefix–suffix square roots $\sqrt[\square\square]{L}$ of a given regular language L. As in the remark after Proposition 1, we restrict the argument to prefix square reduction, as to keep things simple.

Let the set of all words irreducible by p-bounded prefix square reduction over an alphabet Σ be Q_p. This language is regular for any p, as one can easily hard-code checking prefixes of constant bounded length into a finite automaton; hence so is its complement $\overline{Q_p} = \Sigma^* \setminus Q_p$. Furthermore, $\sqrt[\square p]{L}$, the set of p-bounded prefix square roots of L is regular, as well, because it is equal to $\square_p^*(L) \cap Q_p$. Hence, if $\sqrt[\square\square]{L} = \sqrt[\square p]{L}$, for some p, then $\sqrt[\square\square]{L}$ is regular.

Consider now the inverse implication. Suppose $\sqrt[\square\square]{L}$ is regular. Then, $L' = \sqrt[\square p]{L} \setminus \sqrt[\square\square]{L}$ is regular, too, and it consists only of prefix reducible words, as for any irreducible word $w \in \sqrt[\square p]{L}$, naturally $w \in \sqrt[\square\square]{L}$. All words in L' have a square prefix but do not have a p-bounded square prefix. If L' is finite, then $\sqrt[\square\square]{L} = \sqrt[\square q]{L}$ for some $q > p$. If L' is infinite (and we know it is regular), then perhaps one can use a pumping argument to arrive at a contradiction. Therefore, we conjecture that $\sqrt[\square\square]{L}$ is regular if and only if there exists some p such that $\sqrt[\square\square]{L} = \sqrt[\square p]{L}$. For a given p and and a given L this condition is decidable, because one can construct a DFA accepting $\sqrt[\square p]{L}$ and check if it accepts any word with a square prefix (using the argument given after Proposition 1). If the conjecture is true, then finding an upper bound on p depending on L would mean the question of regularity of the set of primitive prefix square roots of a regular language is decidable.

References

[1] P. Bottoni, A. Labella and V. Mitrana, Prefix-suffix square reduction, Theoretical Computer Science **682** (2017) 49–56.
[2] D. P. Bovet and S. Varricchio, On the regularity of languages on a binary alphabet generated by copying systems, Information Processing Letters **44**(3) (1992) 119–123.
[3] M. Crochemore, An optimal algorithm for computing the repetitions in a word, Information Processing Letters **12**(5) (1981) 244–250.
[4] M. Crochemore and W. Rytter, Squares, cubes, and time-space efficient string searching, Algorithmica **13**(5) (1995) 405–425.
[5] J. Dassow, V. Mitrana and G. Păun, On the regularity of duplication closure, Bulletin of EATCS **69** (1999) 133–136.
[6] M. Dumitran, J. Gil, F. Manea and V. Mitrana, Bounded prefix-suffix duplication: Language theoretic and algorithmic results, International Journal Foundations of Computer Science **26**(7) (2015) 933–952.
[7] M. Dumitran and F. Manea, Prefix-suffix square completion, Proceedings of 10th International Conference on Combinatorics on Words, WORDS, LNCS **9304** (2015), pp. 147–159.
[8] M. Dumitran, F. Manea and D. Nowotka, On prefix/suffix-square free words, Proceedings of 22nd International Symposium on String Processing and Information Retrieval, SPIRE, LNCS **9309**, (Springer, 2015), pp. 54–66.
[9] A. Ehrenfeucht and G. Rozenberg, On regularity of languages generated by copying systems, Discrete Applied Mathematics **8**(3) (1984) 313–317.
[10] J. García-López, F. Manea and V. Mitrana, Prefix-suffix duplication, Journal of Computer and System Sciences **80**(7) (2014) 1254–1265.
[11] M. Ito, P. Leupold and K. Shikishima-Tsuji, Closure of language classes under bounded duplication, Proceedings of 10th International Conference on Developments in Language Theory, DLT, LNCS **4036** (2006), pp. 238–247.
[12] P. Leupold, Duplication roots, Proceedings of 11th International Conference on Developments in Language Theory, DLT, LNCS **4588** (2007), pp. 290–299.
[13] P. Leupold, V. Mitrana and J. M. Sempere, Formal languages arising from gene repeated duplication, Aspects of Molecular Computing. Essays in Honour of Tom Head on his 70th Birthday, eds. N. Jonoska, G. Păun and G. Rozenberg, LNCS **2950** (Elsevier, 2004), pp. 297–308.
[14] P. Leupold, C. M. Vide and V. Mitrana, Uniformly bounded duplication languages, Discrete Applied Mathematics **146**(3) (2005) 301–310.
[15] C. H. Papadimitriou, Computational Complexity (Addison-Wesley, Reading, MA, 1995).
[16] G. Rozenberg and A. Salomaa, Handbook of Formal Languages (Springer, Berlin, 1997).
[17] A. Thue, Über unendliche Zeichenreihen, Norske vid. Selsk. Skr. I. Mat. Nat. Kl. Christiana **7** (1906) 1–22.
[18] A. Thue, Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen, Norske vid. Selsk. Skr. I. Mat. Nat. Kl. Christiana **1** (1912).
[19] M.-W. Wang, On the irregularity of the duplication closure, Bulletin of EATCS **70** (2000) 162–163.