Prefix-Suffix Square Reduction

Paolo Bottoni

Department of Computer Science, "Sapienza" University of Rome, Via Salaria 113, 00198 Rome, Italy

Anna Labella

Department of Computer Science, "Sapienza" University of Rome, Via Salaria 113, 00198 Rome, Italy

Victor Mitrana[1]

Faculty of Mathematics and Computer Science,University of Bucharest, Str. Academiei 14, 010014, Bucharest, Romania

Abstract

In this work we introduce the operations of unbounded and bounded prefix-suffix square reduction. We show that, in general, the time complexity of the unbounded prefix-suffix square reduction of a language increases by an n factor in comparison to the complexity of the given language. This factor is just the bound in the case of bounded prefix-suffix square reduction and is not necessary for regular languages. As a consequence, the class of regular languages is closed under unbounded and bounded prefix-suffix square reduction. We show that the class of regular languages is still closed under iterated bounded prefix-suffix square reduction, but the unbounded case remains open. We also show that there are linear languages such that their iterated unbounded prefix-suffix square reduction is not context-free and one cannot algorithmically decide when this is the case. Afterwards, we define the primitive prefix-suffix square root of a word w as a word x that can be obtained from w by iterated prefix-suffix square reductions and is irreducible, i.e., no further prefix or suffix square reduction can be applied. We prove that the language of primitive prefix-suffix square roots of all words over an alphabet is never regular for alphabets with at least two symbols in the unbounded case, and always regular in the bounded case. The papers ends with a brief discussion on some open problems and some algorithmic aspects.

Keywords: duplication, prefix-suffix square reduction, primitive square reduction root, formal languages

Email addresses: bottoni@di.uniroma1.it (Paolo Bottoni), labella@di.uniroma1.it (Anna Labella), mitrana@fmi.unibuc.ro (Victor Mitrana)

[1]Research supported by the Visiting Professor Program - "Sapienza" University of Rome.

1. Introduction

One of the most frequent and vividly investigated phenomena in different contexts is that of repetitive structures. In genetics, for instance, one of the less understood mutations among the genome rearrangements is the duplication of a segment of a chromosome [17]. In the process of duplication, a stretch of DNA is duplicated, yielding two or more adjacent copies, also called tandem repeats. It is commonly asserted that approximately 5% of the genome is involved in duplications and the distribution of these tandem repeats varies widely along the chromosomes [22]. An interesting property of tandem repeats is to make a so-called "phylogenetic analysis" possible, which might be useful in the investigation of the evolution of species by determining the most likely duplication history [24]. The detection of these tandem repeats, as well as algorithms for tandem repeats reconstructing history, have received a great deal of attention in bioinformatics [1, 2, 21]. Thus, duplicating factors and reducing squares to one of their halves is an interesting algorithmic problem with some motivation from bioinformatics. Reductions of squares seems to be of importance in data compression, where the compressed words have to contain some information that allows the reconstruction of the original word [9, 10].

Treating chromosomes and genomes as languages raises the possibility that the structural information contained in biological sequences can be generalized and investigated by formal language theory methods [20]. Thus, the interpretation of duplication as a formal operation on words has inspired several works in the area of Formal Languages opened by [4, 23] and continued in a series of papers, see, e.g., [11, 12] and the references therein. A special type of duplications inspired by the tandem repeats, known as telomeres, which appear only at the end of chromosomes, has been considered in [8]. They are considered to be protective DNA-protein complexes found at the end of eukaryotic chromosomes which stabilize the linear chromosomal DNA molecule [3, 18]. The length of telomeric DNA is important for the chromosome stability: the loss of telomeric repeat sequences may result in chromosome fusion and lead to chromosome instability [15]. Thus, in [8], one considers duplications that may only appear at the ends of the words only, called prefix-suffix duplications. In this context, the aforementioned work investigates the class of languages that can be defined by the iteratively application of the prefix-suffix duplication to a word and try to compare it to other well studied classes of languages. Starting from the biochemical reality that inspired the definition of this operation in [8], namely that the telomeres cannot be arbitrarily long, a restricted variant of duplication, called bounded duplication, was introduced in [6]. In this variant, the length of the prefix or suffix that is duplicated is bounded by a predefined constant. In both papers algorithms for computing different measures on these operations for words and languages are presented.

On the other hand, the inverse operation, namely reducing repetitions, is very natural. Returning to our source of inspiration, for computing a phy-

logenetic network (a set of evolutionary relationships between different genes, chromosomes, genomes, or even species) it is necessary to detect all squares and compute all possible direct predecessors. But this is just one step, if we want to compute other possible predecessors, not necessarily direct ones, hence this process must be iterated. In this paper, we follow the line opened in [13, 14] for the operations considered in [8] and [6]. We define the unbounded prefix-suffix square reduction, which is the inverse of the duplication defined in [8] and the bounded prefix-suffix square reduction, which is the inverse of the duplication defined in [5, 6]. Our operation can be informally defined as the process of reducing a square (tandem repeat) to one of its halves, provided that the square is either a prefix or a suffix of the current word. We consider both variants: bounded and unbounded as well as, for each of them, the non-iterated and the iterated cases.

The paper is organized as follows. In Section 2, we recall all concepts and notations we need and give the formal definitions for the unbounded and bounded prefix-suffix square reductions. Then, in Section 2, we investigate the non-iterated version of these operations. We show that, in general, the time complexity of the unbounded prefix-suffix square reduction of a language increases by an n factor in comparison to the complexity of the given language. This factor is just the bound in the case of bounded prefix-suffix square reduction. This factor is not necessary for regular languages. As a consequence, the class of regular languages is closed under unbounded and bounded prefix-suffix square reduction. On the other hand, there are linear languages such that their unbounded prefix-suffix square reduction is not even context-free. Then we show that the space complexity remain unchanged. The iterated version is then considered in Section 4. We show that the class of regular languages is still closed under iterated bounded prefix-suffix square reduction, but the unbounded case remains open. We also show that there are linear languages such that their iterated unbounded prefix-suffix square reduction is not context-free and one cannot algorithmically decide when this holds. Afterwards, we define the primitive prefix-suffix square root of a word w as a word x that can be obtained from w by iterated prefix-suffix square reductions and is irreducible, i.e., no further prefix or suffix square reduction can be applied. The primitive prefix-suffix square root of a language contains all the primitive prefix-suffix square roots of its words. The language of primitive prefix-suffix square roots over a given alphabet V is the primitive prefix-suffix square root of V^*. We prove that this language is never regular for alphabets with at least two symbols in the unbounded case, and always regular in the bounded case. The papers ends with Section 5, containing a brief discussion on some open problems and some algorithmic aspects.

We close this introduction by stressing that the investigation we pursue here is not aimed to tackle real biological solutions. The biological phenomenon is just a source of inspiration for our approach; our approach uses the biological concepts at a simplified level and only from a theoretical computer science point of view. Its aim is actually to provide a better understanding of the structural properties of strings obtained by prefix-suffix square reductions. On the long

run, such tools might provide the foundations on which applications working with real data are built.

2. Preliminaries

We assume the reader to be familiar with fundamental concepts of formal language theory and complexity theory (see, e.g., [19] and [16], respectively). We start by summarizing the notions used throughout this work. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any finite sequence of symbols from an alphabet V is called a *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε; we further let $V^+ = V^* \setminus \{\varepsilon\}$. Given a word w over an alphabet V, we denote by $|w|$ its length, while $|w|_a$ denotes the number of occurrences of the symbol $a \in V$ in w. Furthermore, $alph(w)$ denotes the minimal alphabet $W \subseteq V$ such that $w \in W^*$, i.e. $alph(w) = \{a \in V \mid |w|_a \neq 0\}$. Obviously, $alph(L) = \bigcup_{x \in L} alph(x)$. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w. For a word w, $w[i..j]$ denotes the subword of w starting at position i and ending at position j, $1 \leq i \leq j \leq |w|$; by convention, $w[i..j] = \varepsilon$ if $i > j$. If $i = j$, then $w[i..j]$ is the i-th symbol of w, which is simply denoted by $w[i]$.

Given a word x over an alphabet V, we consider the following operations:

- *Prefix square reduction*, namely $\sqsubset\!| (x) = \{uy \mid x = uuy \text{ for some } u \in V^+\}$. The *suffix square reduction* is defined analogously, i.e., $|\!\sqsupset (x) = \{yu \mid x = yuu \text{ for some } u \in V^+\}$.
- *Prefix-suffix square reduction*, namely $\sqsubset\!\sqsupset (x) = \sqsubset\!| (x) \cup |\!\sqsupset (x)$.

The prefix-suffix square reduction is naturally extended to languages L by $\sqsubset\!\sqsupset (L) = \bigcup_{x \in L} \sqsubset\!\sqsupset (x)$. We further define:

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

Note that the operations $\sqsubset\!|$, $|\!\sqsupset$, and $\sqsubset\!\sqsupset$ are actually the inverse of the operations *PD* (prefix duplication), *SD* (suffix duplication), and *PSD* (prefix-suffix duplication), respectively, introduced in [8]. However, we have preferred to use $\sqsubset\!\sqsupset$ instead of PSD^{-1} for two reasons: (i) the iterated version might lead to misinterpretations; (ii) the symbol $\sqsubset\!\sqsupset$ nicely visualizes the operation going from a square to one of its halves.

We also define a restricted variant of the prefix-suffix square reduction called *bounded prefix-suffix square reduction*. Formally, given a word $x \in V^*$ and a positive integer p, we define the p-prefix-suffix square reductions by:

– *p-prefix square reduction*: ${}_p\sqsubset\!|(x) = \{uy \mid x = uuy \text{ for some } u \in V^+, |u| \leq p\}$.

– *p-suffix square reduction*: $|\!\sqsupset_p(x) = \{yu \mid x = yuu \text{ for some } u \in V^+, |u| \leq p\}$.

– *p-prefix-suffix square reduction*: ${}_p\sqsubset\!|\!\sqsupset_p(x) = {}_p\sqsubset\!|(x) \cup |\!\sqsupset_p(x)$.

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 [5].

3. Non-iterated prefix-suffix square reduction

We write $L \in \mathcal{O}(f(n))$, if L can be decided in $\mathcal{O}(f(n))$ time.

Theorem 1. *Let $f(n)$ be a monotone function such that $f(2n) \in \mathcal{O}(f(n))$.*
1. If $L \in \mathcal{O}(f(n))$, then $\sqsubset\!|\!\sqsupset(L) \in \mathcal{O}(nf(n))$.
2. If $L \in \mathcal{O}(f(n))$, then ${}_p\sqsubset\!|\!\sqsupset_p(L) \in \mathcal{O}(pf(n))$, for any $p \geq 1$.

Proof. 1. It suffices to show that $\sqsubset\!|(L) \in \mathcal{O}(nf(n))$ holds as $|\!\sqsupset(L) \in \mathcal{O}(nf(n))$ can be shown analogously. Let w be a word of length n over $alph(L)$. The following algorithm returns the truth value of $w \in \sqsubset\!|(L)$.

Algorithm 1 .

Input: $w \in V^+$, $|w| = n$.
Output: **True**, if $w \in \sqsubset\!|(L)$, and **False**, if $w \notin \sqsubset\!|(L)$.

```
1: for i = 1 to n do
2:   if w[1..i]w ∈ L then
3:     return True; HALT;
4:   end if
5: end for
6: return False
```

As the condition in line 2 can be checked in $\mathcal{O}(f(i+n))$ time and (by hypothesis) $f(i+n) \leq f(2n) \in \mathcal{O}(f(n))$, the proof of the first statement is over.

2. The second statement follows similarly: it is sufficient to modify the range of the loop index i in the first line of Algorithm 1 so that the algorithm iterates on the interval $1..p$. □

There are many functions with this property: polynomials, roots, logarithms, etc. It follows that the class **P** of polynomially recognizable languages by deterministic Turing machines is closed under unbounded and bounded prefix-suffix square reductions.

A natural problem is to identify classes of languages for which the n or p factor is not needed. To this aim we can prove:

Theorem 2. *Let L be a regular language and let A be a deterministic finite automaton with k states accepting L.*
1. The membership problem for $\square\!\!+\!\!\square\ (L)$ lies in $\mathcal{O}(kn)$.
2. The membership problem for ${}_p\square\!\!+\!\!\square_p(L)$ lies in $\mathcal{O}(kn)$, for any $p \geq 1$.

Proof. 1. As above, it suffices to show that $\square|\ (L) \in \mathcal{O}(kn)$. Let $A = (Q, V, \delta, q_1, F)$ with $Q = \{q_1, q_2, \ldots, q_k\}$ and $w \in V^*, |w| = n$. We define the following arrays:

$$\begin{aligned} N[i,j] &= \delta(q_i, w[1..j]), 1 \leq i \leq k, 1 \leq j \leq n, \\ R[i,j] &= \delta(q_i, w[j..n]), 1 \leq i \leq k, 1 \leq j \leq n. \end{aligned}$$

These arrays can be computed in $\mathcal{O}(kn)$ time. Indeed, for each $1 \leq i \leq k$ the values of $N[i,j]$ can be computed in the increasing order of j, while the values of $R[i,j]$ can be computed in the decreasing order of j. We now claim that $w \in \square|\ (L)$ iff one of the following conditions is satisfied:

$$\begin{aligned} &(i) \quad R[N[N[1,i],i],i+1] \in F, \text{ for some } 1 \leq i \leq n-1, \\ &(ii) \quad N[N[1,n],n] = \delta(q_1, ww) \in F. \end{aligned}$$

Indeed, the first item can be rewritten as:

$$\begin{aligned} R[N[N[1,i],i],i+1] &= \delta(N[N[1,i],i], w[i+1..n]) = \delta(\delta(N[1,i], w[1..i]), \\ &\quad w[i+1..n]) = \delta(\delta(\delta(q_1, w[1..i]), w[1..i]), w[i+1..n]) \\ &= \delta(q_1, w[1..i]w) \in F. \end{aligned}$$

As one can see, the overall time of the algorithm is $\mathcal{O}(kn)$.

2. The second assertion can be proved by an easy modification of the above construction. □

The last proof leads to the following statement:

Corollary 1. *If L is a regular language, then all languages $\square\!\!+\!\!\square\ (L)$ and ${}_p\square\!\!+\!\!\square_p(L)$, $p \geq 1$, are also regular.*

Note the opposite situations regarding the closure of the class of regular languages under the two complementary operations *PSD*, defined in [8], and $\square\!\!+\!\!\square$. It is obvious that $PSD(ab^+a) \cap ab^+aab^+a = \{ab^naab^na \mid n \geq 1\}$, which is not regular. In some sense, this result may be explained by the fact that prefix-suffix duplication "adds" some dependencies, while the prefix-suffix square reduction "eliminates" such dependencies. However, this intuitive "explanation" does not hold for other classes of languages. For instance, there are linear languages L such that $\square\!\!+\!\!\square\ (L)$ is not context-free. Indeed, it suffices to consider the linear language

$$L = \{a^nba^mba^mba^nb \mid n, m \geq 1\}.$$

It is obvious that

$$\square\!\!+\!\!\square\ (L) \cap a^+ba^+ba^+b = \{a^nba^nba^nb \mid n \geq 1\},$$

which is not context-free.

However, the situation is different for the bounded prefix-suffix square reduction:

Theorem 3. *The classes of linear and context-free languages are closed under bounded prefix-suffix square reduction.*

Proof. Let $p \geq 1$ and L be a linear or context-free language over an alphabet V. For each word $x \in V^+$ of length at most p, we write

$$_xL = L \cap \{xx\}V^*, \quad \text{and } L_x = L \cap V^*\{xx\}).$$

Clearly, both

$$_p\square\!\!+\!\!\square_p(L) = \bigcup_{x \in V^+, |x| \leq p} ((x \setminus_x L) \cup (L_x/x)),$$

where the operators $\setminus$ and $/$ are for the left and right quotient, respectively. The statement follows now by the closure properties of the two families of languages. □

Theorem 4. *Let $f(n) \geq \log n$ be a space constructible function.*
*1. If $L \in$ **DSPACE**$(f(n))$, then $\square\!\!+\!\!\square\ (L) \in$ **DSPACE**$(f(n))$ as well.*
*2. If $L \in$ **DSPACE**$(f(n))$, then $_p\square\!\!+\!\!\square_p(L) \in$ **DSPACE**$(f(n))$, for all $p \geq 1$.*

Proof. We will prove the statement for the unbounded variant only. Let L be a language accepted by an off-line deterministic Turing machine M in space $f(n)$. Again, we shall prove that $\square\!\!|\ (L) \in$ **DSPACE**$(f(n))$, the proof for $|\!\!\square\ (L) \in$ **DSPACE**$(f(n))$ being analogous. Assume that M has just one working tape. Our idea is to implement Algorithm 1 on an off-line Turing machine M' with three working tapes. The first tape of M' will simulate the working tape of M, i.e., the first tape of M' will always have the same contents as the working tape of M, and in any simulation step both machines will read the same symbol on these tapes. The other two tapes will be used for storing integers in binary representation. Thus, the second tape of M' contains an integer i between 1 and n, while the third tape contains an integer j between 0 and $i+n+1$, where n is the length of the input word. The set of states of M' contains two copies of each state of M; let us say that one copy is red and the other one is green. As a general rule, green states are used to encode the information that $j \leq i$ and red states to encode the information that $i < j$. The Turing machine M' works on an input w of length n as follows:

- M' will store 1 on both its second and third tapes and enters the green copy of the initial state of M. Being in the green copy of the initial state of M with i on its second tape and 1 on its third tape, M' is going to simulate the work of M on the input word $w[1..i]w$ inductively as follows.

- In a green copy of the state q of M with the integers i, j ($j \leq i$) on its second and third tapes, M' reads the j-th symbol on its input tape (which is $w[j]$, provided that $j \neq 0$, or the left end marker, otherwise) and simulates the move of M in the state q by reading the j-th symbol on its input tape. Remember that the word on its input tape is $w[1..i]w$, therefore the symbol read by M is also $w[j]$. Let us assume that M, being in state q, reads the symbol a, enters q', and moves its input head to the left or right. If M moved its input head to the left, then M' changes the content of the third tape from j to $j-1$ and enters a green copy of q'. If M moved its input head to the right, then M' changes the content of the third tape from j to $j+1$ and enters either a green copy of q', if $j \leq i$, or a red copy of q', otherwise.

- In a red copy of the state q of M with the integers i, j ($j > i$) on its second and third tapes, M' reads the $(j-i)$-th symbol on its input tape (which is $w[j-i]$, provided that $j - i \leq n$, or the right end marker, otherwise) and simulates the move of M in the state q by reading the j-th symbol on its input tape. Remember that the current contents of its input tape is $w[1..i]w$, therefore the symbol read by M would be also $w[j-i]$. Let us assume that M, being in state q, reads the symbol a, enters q', and moves its input head to the left or right. If M moved its input head to the left, then M' changes the content of the third tape from j to $j-1$ and enters either a green copy of q', if $j \leq i$, or a red copy of q', otherwise. If M moved its input head to the right, then M' changes the content of the third tape from j to $j+1$ and enters a red copy of q'.

- If M reaches a final state during the above simulations, then M' enters a final state and accepts its input. If M does not accept the input $w[1..i]w$, then M' increases the value of i, provided that $i < n$, makes again $j = 1$, enters the green copy of the initial state of M and continues with the simulation of M on the new input $w[1..i]w$. If $i = n$, then M' halts and rejects w.

It is clear that M' is deterministic and accepts exactly $\square\!|\,(L)$. Moreover, the space used on the two extra tapes is at most $\log n$, therefore M' decides $\square\!|\,(L)$ is $\mathcal{O}(f(n))$ space. By the well-known fact that $\mathbf{DSPACE}(f(n))$ is closed under union, the proof is complete. $\square$

Let $\mathbf{L}$ be as usual the complexity class $\mathbf{DSPACE}(\log n)$; as a direct consequence of this theorem, since the second and third tapes only contain representations of numbers $1 \leq i \leq n$, $0 \leq j \leq 2n+1$, thus employing at most $\mathcal{O}(\log n)$ space, we state:

Corollary 2. $\mathbf{L}$ *is closed under bounded and unbounded prefix-suffix square reduction.*

It is worth mentioning that the proof of Theorem 4 works for the nondeterministic classes as well.

4. Iterated prefix-suffix square reduction

We do not know whether $\square\!\!+\!\!\square^*(L)$ is still regular, provided that L is regular. However, we can give an answer for the iterated bounded prefix-suffix square reduction:

Theorem 5. *The class of regular languages is closed under iterated bounded prefix-suffix square reduction.*

Proof. Let $A = (Q, V, \delta, q_0, F)$ be a DFA accepting the language L. We set ${}_p\square\!\!+\!\!\square_p{}^*(L) = L_1 \cup L_2$, where L_1 contains all words of ${}_p\square\!\!+\!\!\square_p{}^*(L)$ of length at least $2p$, while $L_2 = {}_p\square\!\!+\!\!\square_p{}^*(L) \setminus L_1$. As L_2 is finite and can be effectively computed, we only have to prove that L_1 is regular. To this aim, we construct a finite automaton with ε-moves $A' = (Q', V, \delta', q_0, \{q_f\})$, where $q_f \notin Q$, as follows. The set of states Q' is

$$\begin{aligned} Q' \;=\; & Q \cup \{q_f\} \cup \{(q,x) \mid q \in Q, x \in V^+, |x| = p\} \cup \\ & \{(x,q) \mid q \in Q, x \in V^+, |x| = p\}, \end{aligned}$$

while the transition mapping δ' is the extension of δ which enables the following computations in A'. For the sake of simplicity we prefer to explain the computations of A' instead of defining the mapping δ'. From these explanations, the reader may define completely the transition mapping δ'.

1. From its initial state, q_0, A' may reach by an ε-move any of the states (q_0, x) with $x \in V^+, |x| = p$.
2. From a state (q, x), with $q \in Q$, $x \in V^+, |x| = p$, the automaton A' may reach by an ε-move any of the states (s, y) such that there exists $z \in V^+, |z| \leq p$, with $\delta(q, z) = s$, and $x = (zy)[1..p], y = zy'$, for some $y' \in V^*$.
3. From a state (q, x), with $q \in Q$, $x \in V^+, |x| = p$, the automaton A' may also reach the state $\delta(q, x)$ by reading x, analogously to the computation of A by reading x.
4. From any state of Q, by reading a symbol from V, A' makes the same move as A does.
5. From any state q of Q, by reading a word x of length p, A' may reach the state $(x, \delta(q, x))$ analogously to the computation of A by reading x.
6. From a state (x, q), with $q \in Q$, $x \in V^+, |x| = p$, the automaton A' may reach by an ε-move any of the states (y, s) such that there exists $z \in V^+, |z| \leq p$, with $\delta(q, z) = s$, and $y = (xz)[|xz| - p + 1..|xz|], x = x'z$, for some $x' \in V^*$.
7. From any state (x, q), with $q \in F$, $x \in V^+, |x| = p$, by an ε-move, A' may reach the final state q_f.

Now, the computation of A' on an input word w of length at least $2p$ is done as follows. It starts from its initial state q_0 and nondeterministically enters one of the states (q_0, α) with $|\alpha| = p$. By a series of ε-moves (this series may

be empty), A' nondeterministically reaches a state (q, x) such that there exists $\beta \in V^*$, having the prefix α, with $\delta(q_0, \beta) = q$ and $x \in {}_p\square^*(\beta x)$. Then A' reads x and enters a state s. This means that the following conditions are satisfied:

(i) $\delta(q_0, \beta x) = s$,

(ii) x is a prefix of w of length p,

(iii) $x \in {}_p\square^*(\beta x)$.

Then A' may read another part, say y, from its input word and enters the state r. Note that y may be the empty word. Nondeterministically, A' reads a segment of length p from its input word, say z, and enters the state $(z, \delta(r, z))$. Denote $t = \delta(r, z)$. By a series of ε-moves (this series may be empty), A' reaches a state (v, μ) such that there exists $\gamma \in V^*$, having the suffix μ, with $\delta(t, \gamma) = v$ and $z \in \sqsupset_p{}^*(z\gamma)$. If v is a final state of A, then A' reads ε and enters its final state q_f. This means that the following conditions are satisfied:

(i) $\delta(t, z\gamma) = v$,

(ii) $z \in \sqsupset_p{}^*(z\gamma)$.

We can see now that if z is not a suffix of w, then the computation gets stuck before reading the whole input word. Therefore, z must be a suffix of w of length p. It follows that the input word w can be written as $w = xyz$, with $|x| = |z| = p$, $\beta xyz\gamma \in L$, and $w \in {}_p\square\!\!\sqsupset_p(\beta xyz\gamma)$, hence $w \in L_1$. □

A closer look to the previous proof gives hints to prove the following result:

Theorem 6. *The class of linear languages is closed under iterated bounded prefix-suffix square reduction.*

Proof. Let Let $G = (N, V, S, P)$ be a linear grammar; without loss of generality we may assume that the rules of P are of the forms:

(i) $A \to aB, A, B \in N, a \in V$,

(ii) $A \to Ba, A, B \in N, a \in V$,

(iii) $A \to \varepsilon, A \in N$.

As in the previous proof, we set ${}_p\square\!\!\sqsupset_p{}^*(L) = L_1 \cup L_2$, where L_1 contains all words of ${}_p\square\!\!\sqsupset_p{}^*(L)$ of length at least $2p$, while $L_2 = {}_p\square\!\!\sqsupset_p{}^*(L) \setminus L_1$. As L_2 is finite and can be effectively computed, we only have to prove that L_1 is linear. We construct the linear grammar $G' = (N', V, (\varepsilon, S, \varepsilon), P')$, where $N' = N \cup \{(x, A, y) \mid A \in N, x, y \in V^*, |x| \leq 2p, |y| \leq 2p\}$, and P' contains, besides all the rules of P, the following rules:

1. $(x, A, y) \to (xa, B, y)$, if $A \to aB \in P, |xa| \leq 2p$,
2. $(x, A, y) \to (x, B, ay)$, if $A \to Ba \in P, |ay| \leq 2p$,
3. $(x, A, y) \to (u, B, y)$, if $A \to aB \in P$, and $u \in {}_p\square(xa)$,
4. $(x, A, y) \to (x, B, v)$, if $A \to Ba \in P$, and $v \in \sqsupset_p(ay)$,

for all $A \in N$, $x, y \in V^*$, $|x| \leq 2p$, $|y| \leq 2p$, and

5. $(x, A, y) \to xAy, A \in N, x, y \in V^*, p \leq |x|, |y| \leq 2p$.

It is easy to see that any derivation in G' starts with a sequence of derivations in which only rules of the forms 1-4 are applied, hence the sentential form consists in just one nonterminal of the form (x, A, y) from N'. After this sequence, the derivation continues with a rule of the form 5. Now the sentential form is xAy such that there exist $u, v, z \in V^*$ with $x \in {}_p\square^*(u)$, $y \in \square_p{}^*(v)$, $A \longrightarrow^* z$ and $uzv \in L(G)$. Therefore, $L(G') = L_1$. □

Theorem 7.
1. There are linear languages L such that $\square\!\!+\!\!\square^*(L)$ is not context-free.
2. Given a linear language L one cannot algorithmically decide whether or not $\square\!\!+\!\!\square^*(L)$ is context-free.

Proof. 1. Let

$$x = (x_1, x_2, \ldots, x_n), \qquad y = (y_1, y_2, \ldots, y_n),$$

be two Post lists over the alphabet $V = \{a, b\}$ for some $n \geq 2$. We construct the linear language

$$\begin{aligned} L(x,y) \quad = \quad & \{ca^{i_1}ba^{i_2}b\ldots a^{i_k}bca^{j_1}ba^{j_2}b\ldots a^{j_m}bc^r y_{j_m}\ldots y_{j_2}y_{j_1}cx_{i_k}\ldots x_{i_2}x_{i_1} \mid \\ & k, m \geq 1, 1 \leq i_1, i_2, \ldots, i_k \leq n, 1 \leq j_1, j_2, \ldots, j_m \leq n, r \geq 3.\} \end{aligned}$$

We consider the language $\square\!\!+\!\!\square^*(L) \cap \{c\}V^+\{c^r \mid r \geq 3\}V^+\{c\}$. This language is either empty or infinite. We prove that if this language is not empty, then it cannot be context-free. The language $\square\!\!+\!\!\square^*(L) \cap \{c\}V^+\{c^r \mid r \geq 3\}V^+\{c\}$ is actually the set

$$\begin{aligned} S \quad = \quad & \{ca^{j_1}ba^{j_2}b\ldots a^{j_m}bc^r y_{j_m}\ldots y_{j_2}y_{j_1}c \mid r \geq 3, \text{ and } j_1, j_2, \ldots, j_m \\ & \text{is a solution of the PCP for } x \text{ and } y\}. \end{aligned}$$

It is an easy exercise to prove that this language, if not empty, does not satisfy the pumping lemma.

2. We take the same linear language $L(x, y)$ as above. Clearly, $\square\!\!+\!\!\square^*(L)$ is context-free if and only if $\square\!\!+\!\!\square^*(L) \cap \{c\}V^+\{c^r \mid r \geq 3\}V^+\{c\}$ is context-free, which is equivalent to $\square\!\!+\!\!\square^*(L) \cap \{c\}V^+\{c^r \mid r \geq 3\}V^+\{c\} = \emptyset$. But this holds if and only if the Post Correspondence Problem for the two lists has no solution, which is not decidable. □

We say that a word z is a primitive prefix-suffix square root of a word x if $z \in \square\!\!+\!\!\square^*(x)$ and z is both prefix and suffix square free (it neither starts nor ends with a square). Denote by $\sqrt[\square\!\!+\!\!\square]{x}$ the set of all primitive prefix-suffix square roots of x. For a language L we write $\sqrt[\square\!\!+\!\!\square]{L} = \bigcup_{x \in L} \sqrt[\square\!\!+\!\!\square]{x}$. The primitive bounded prefix-suffix square root of a word, the set of all primitive bounded prefix-suffix square roots of a word and of a language are defined analogously. A natural question concerns the language of primitive prefix-suffix square roots of all words over a language, that is $\sqrt[\square\!\!+\!\!\square]{V^*}$.

Theorem 8.
1. The language $\sqrt[\square\square]{V^*}$ *is regular if and only if* V *is a singleton.*
2. The language $\sqrt[p\square\square p]{V^*}$ *is regular for any alphabet* V*.*

Proof. 1. Clearly, $\sqrt[\square\square]{\{a\}^*} = \{a\}$, hence it is regular. Assume that $V = \{a, b\}$. We start from the easy fact that each word aba^nbaba^mb has a prefix or a suffix square if and only if at least one of the followings is true:
(i) $m = 0, 1$,
(ii) $n = 1, 2$,
(iii) $n = m$.
It is known that a regular language $L \subseteq V^*$ induces on V^* the Myhill-Nerode equivalence relation: $x \equiv_L y$ if, for all $w \in V^*$, we have that $xw \in L$ if and only if $yw \in L$. It follows that any two words aba^nb and aba^mb, with $n, m \geq 3, n \neq m$ can be distinguished under the Myhill-Nerode equivalence induced by $\sqrt[\square\square]{V^*}$. Indeed, for $x = aba^nb$, $y = aba^mb$, and $w = aba^mb$, we have $xw \in \sqrt[\square\square]{V^*}$, while $yw \notin \sqrt[\square\square]{V^*}$. Therefore, by the Myhill-Nerode theorem, $\sqrt[\square\square]{V^*}$ cannot be regular.
2. The complement of the language $\sqrt[p\square\square p]{V^*}$ is the regular language

$$\{xx \mid 1 \leq |x| \leq p\}V^* \cup V^*\{xx \mid 1 \leq |x| \leq p\},$$

hence we are done. □

5. Discussion and final remarks

As one can see, in the case of non-iterated prefix-suffix square reduction, we provided a rather complete picture. The situation is quite different in the case of iterated unbounded prefix-suffix square reduction in the sense that the status of a few problems remained open:

1. Is the iterated unbounded prefix-suffix square reduction of a regular language still regular? Actually, we cannot say whether this language is recursive.
2. Is the primitive unbounded prefix-suffix square root of a regular language still regular? We believe that the primitive unbounded prefix-suffix square root of a regular language is still regular if and only if it is finite.

We have not considered here algorithmic problems like in [8] and [6]. However, as the operations considered here are the inverses of those in the aforementioned papers, some natural algorithmic problems that could be considered have already been solved in [8] and [6]. For instance, these problems have already been solved for prefix-suffix duplication and their solutions are identical to those for prefix-suffix square reductions:

1. Given two words x and w, does x belong to $\square\!\!+\!\!\square^*(w)$?
2. Given two words x and w, what is the prefix-suffix square reduction distance between x and w? The prefix-suffix square reduction distance between two words is defined as the minimal number of prefix-suffix square reductions applied to one of the words in order to obtain the other one.

It is worth mentioning that Section 4 from [7] deals with several algorithmic problems concerning the primitive prefix-suffix square roots of a word. In that work, the primitive prefix-suffix square roots are called *PSD*-irreducible roots. However, there are still some algorithmic problems regarding the primitive prefix-suffix square roots of a word which seem interesting to us:

1. Given a word w, compute the number of all its primitive prefix-suffix square roots of w. Note that the number of all prefix-suffix square free factors of a word of length n can be computed in $\mathcal{O}(n \log n)$ time (see [7]). It is very likely that the ideas from [7] may be used for solving this problem.
2. What is the complexity of deciding whether a given word has a unique primitive prefix-suffix square root, a unique shortest or longest primitive prefix-suffix square root?
3. Compute the set of all shortest or longest primitive prefix-suffix square roots of a word. Compute only the cardinal of these sets.

6. References

[1] G. Benson, L. Dong, Reconstructing the duplication history of a tandem repeat, In: T. Lengauer, R. Schneider, P. Bork, D. L. Brutlag, J. I. Glasgow, H.-W. Mewes, R. Zimmer (Eds.), *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, Heidelberg, Germany, AAAI, 1999, pp. 44–53.

[2] D.A. Benson, Tandem repeat finder: A program to analyze DNA sequences, *Nucleic Acids Research* 27–2 (1999) 573–580.

[3] S.R. Chan, E.H. Blackburn, Telomeres and telomerase, *Philos. Trans. R. Soc. Lond. B. Biol. Sci.* 359 (2004) 109–121.

[4] J. Dassow, V. Mitrana, Gh. Păun, On the regularity of duplication closure, *Bull. European Assoc. Theor. Comput. Sci.* 68 (1999) 133–136.

[5] M. Dumitran, J. Gil, F. Manea, V. Mitrana, Bounded prefix-suffix duplication, In: M. Holzer and M. Kutrib (Eds.), *Conference on Implementation of Automata and Applications CIAA 2014*, LNCS 8587, Springer International Publishing Switzerland, 2014, pp. 176-187.

[6] M. Dumitran, J. Gil, F. Manea, V. Mitrana, Bounded Prefix-Suffix Duplication: Language Theoretic and Algorithmic Results, *Int. J. Found. Comput. Sci.* 26 (2015) 933-952.

[7] M. Dumitran, F. Manea, D. Nowotka, On prefix/suffix-square free words, In: C. Iliopoulos et al. (Eds.), *Proceedings of SPIRE 2015*, Springer International Publishing Switzerland, 2015, pp. 54-66.

[8] J. Garcia Lopez, F. Manea, V. Mitrana, Prefix-suffix duplication, *J. Comput. Syst. Sci.*, 80–7 (2014) 1254–1265.

[9] L. Ilie, S. Yu, K. Zhang, Repetition complexity of words, In: O. H. Ibarra, L. Zhang (Eds.), *COCOON 2002*, LNCS 2387, Springer, 2002, pp. 320-329.

[10] L. Ilie, S. Yu, K. Zhang, Word complexity and repetitions in words, *Int. J. Found. Comput. Sci.* 15 (2004) 4155.

[11] M. Ito, P. Leupold, K. Shikishima-Tsuji, Closure of language classes under bounded duplication, In: O. Ibarra, Z. Dan (Eds.) *Developments in Language Theory*, LNCS 4036, Springer, 2006, pp. 238–247.

[12] P. Leupold, V. Mitrana, J.M. Sempere, Formal languages arising from gene repeated duplications, In: N. Jonoska, G. Paun, G. Rozenberg (Eds.), *Aspect of Molecular Computing* LNCS 2950, Springer-Verlag, Berlin, 2004, pp. 301-310.

[13] P. Leupold, Duplication roots, In: T. Harju, J. Karhumäki, A. Lepistö (Eds.) *Developments in Language Theory*, LNCS 4588, Springer, 2007, pp. 290–299.

[14] P. Leupold, Reducing repetitions, In: J. Holub, J. Zdarek (Eds.), *Prague Stringology Conf.*, 2009, pp. 225–236.

[15] J.P. Murnane, Telomere dysfunction and chromosome instability, *Mutat. Res.* 730 (2012) 28–36.

[16] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[17] E. Pennisi, Genome duplications: The stuff of evolution?, *Science* 294 (2001) 2458–2460.

[18] R.J. Preston, Telomeres, telomerase and chromosome stability, *Radiat. Res.* 147 (1997) 529–534.

[19] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. I–III, Springer-Verlag, Berlin, 1997.

[20] D.B. Searls, The computational linguistics of biological sequences, In: L. Hunter (Ed.), *Artificial Intelligence and Molecular Biology*, MIT Press, Cambridge, MA, 1993, pp. 47–120.

[21] D. Sokol, G. Benson, J. Tojeira, Tandem repeats over the edit distance, *Bioinformatics* 23–2 (2007) 30–35.

[22] L.D. Stein, End of the beginning, *Nature* 431 (2004) 915–916.

[23] M.-w. Wang, On the irregularity of the duplication closure, *Bull. European Assoc. Theor. Comput. Sci.* 70 (2000) 162–163.

[24] I. Wapinski, A. Pfeffer, N. Friedman, A. Regev, Natural history and evolutionary principles of duplication in fungi, *Nature* 449 (2007) 54–61.