International Journal of Computer Mathematics
Vol. 00, No. 00, Month 200x, 1–19

RESEARCH ARTICLE

Finding Shuffle Words that Represent Optimal Scheduling of Shared Memory Access

Daniel Reidenbach[a] and Markus L. Schmid[a]*

[a]*Department of Computer Science, Loughborough University, Loughborough, Leicestershire, LE11 3TU, United Kingdom*
(*Received 00 Month 200x; in final form 00 Month 200x*)

In the present paper, we introduce and study the problem of computing, for any given finite set of words, a shuffle word with a minimum so-called scope coincidence degree. The scope coincidence degree is the maximum number of different symbols that parenthesise any position in the shuffle word. This problem is motivated by an application of a new automaton model and can be regarded as the problem of scheduling shared memory accesses of some parallel processes in a way that minimises the number of memory cells required. We investigate the complexity of this problem and show that it can be solved in polynomial time.

Keywords: string algorithms; shuffle; memory access scheduling

AMS Subject Classification: 68Q25; 68Q42; 90B35; 90C27.

1. Introduction

In this work, we introduce and investigate a problem on shuffling words. A word is a sequence of symbols, e. g., $u := \mathbf{abacbc}$ and $v := \mathbf{abc}$, and two words are shuffled by interleaving them in such a way that the relative order of their symbols is preserved. More precisely, a *shuffle word* of u and v is any word that can be obtained by inserting the symbols of u into v without changing their order. Hence, **abacbcabc**, **aabbaccbc**, **abaabcbcc** or **abcabacbc** are possible shuffle words of u and v whereas **caabbacbc** is not. Shuffle words of more than two words are defined iteratively, e. g., w is a shuffle word of the words u_1, u_2 and u_3 if and only if it is a shuffle word of u_1 and v, where v is a shuffle word of u_2 and u_3.

We study the problem of constructing, for a given set of words, a shuffle word such that a certain parameter is minimised. Intuitively, this parameter is the maximum number of different symbols that parenthesise any position in the shuffle word. Motivation for this computational problem can be derived from the following problem on scheduling memory accesses.

We assume that some process is executed, which requires access to different values stored in the memory. These memory accesses can be modeled as a sequence of symbols, e. g., the process $u := \mathbf{abacbc}$ needs to access the values **a**, **b** and **c** in the order $\mathbf{a}, \mathbf{b}, \mathbf{a}, \mathbf{c}, \mathbf{b}, \mathbf{c}$. Obviously, u can be executed by reserving a single memory cell for each of the values **a**, **b** and **c**. On the other hand, u can also be divided into two parts, $u_1 := \mathbf{aba}$ and $u_2 := \mathbf{cbc}$, and for the execution of u_1, we use two memory cells in order to store values **a** and **b** and for the execution of u_2, we use

*Corresponding author. Email: M.Schmid@lboro.ac.uk

ISSN: 0020-7160 print/ISSN 1029-0265 online

DOI: 10.1080/0020716YYxxxxxxxx
http://www.informaworld.com

the *same* two memory cells for values **c** and **b**. Consequently, u can be executed with only two instead of three memory cells. The situation is more involved if an additional process $v := \mathbf{abc}$ is supposed to be executed in parallel to u. If we first execute process u and then process v, then our sequence of memory accesses reads **abacbcabc** and it can be easily verified that now we necessarily need an individual memory cell for each of the values **a**, **b** and **c**. However, we can again get along with only two memory cells if we interleave, or shuffle, the processes u and v. More precisely, we first carry out only the initial part **aba** of process u and then interrupt its execution. After that, process $v = \mathbf{abc}$ is executed and then u is completed by executing **cbc**. The memory accesses are now **abaabccbc** and again we can use two memory cells to first store values **a** and **b** and then reuse the same cells in order to store values **b** and **c**.

We observe that this scheduling problem is actually a problem on shuffling words. More precisely, the problem of finding the best way of organising the memory accesses of all processes directly translates into computing a shuffle word of all the processes that minimises the parameter determining the number of memory cells required. The naive way of solving this problem, i. e., investigating all possible shuffle words, seems inappropriate, since there is an exponential number of them. However, we can present an algorithm solving this problem for arbitrary input words and a fixed alphabet size in polynomial time.

The problem described above shows similarities to the problem of *register allocation* (see, e. g., [4, 6]), a problem that plays an important role for the optimisation of compilers. However, a closer look reveals substantial differences between these two tasks of allocating values to memory cells. In register allocation we deal with the problem that there are not enough registers to store all necessary values and, thus, some values need to be stored in the main memory. Consequently, the optimisation objective is to find an allocation such that the number of accesses to the main memory is minimised, since these constitute a much more expensive CPU operation compared to accessing a register. On the other hand, in our scheduling problem the number of registers is not fixed and our objective is to minimise the number of these registers. Furthermore, in register allocation the periods during which the values must be accessible in registers can be arbitrarily changed by storing them in the main memory, and there is usually not the problem of sequentialising several processes.

Another motivation for the introduced problem on shuffling words results from an application of a new automaton model with two input heads [7]. Within the scope of [7], these two input heads are moved over whole factors of the input word, thus, each input head trajectory can be interpreted as a process that needs to access lengths of factors in a certain order. By interleaving these trajectories in a specific way, the number of factor lengths that need to be stored simultaneously can be decreased, which does not only affect the memory usage of the automaton, but it also has a significant impact on the runtime of its computations. Consequently, the main result of the present paper can be used to improve the approach in [7], but we shall not discuss this aspect explicitly here. We believe that this nontrivial problem might occur in other practical situations as well, but, to the knowledge of the authors, it is not covered by any literature on scheduling (see, e. g., [1, 3]) and the same holds for the research on the related *common supersequence problem* (see, e. g., [5]).

2. Basic Definitions

In the following, let Σ be a finite alphabet. A *word* (*over* Σ) is a finite sequence of symbols from Σ, and ε stands for the *empty word*. The symbol Σ^+ denotes the set of all nonempty words over Σ, and $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in \Sigma^*$ is a *factor* of a string $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If $u_1 = \varepsilon$ (or $u_2 = \varepsilon$), then v is a *prefix* of w (or a *suffix*, respectively). The notation $|K|$ stands for the size of a set K or the length of a string K. The term $\mathrm{alph}(w)$ denotes the set of all symbols occurring in w and, for each $a \in \mathrm{alph}(w)$, $|w|_a$ refers to the number of occurrences of a in w. A word w' is a *permutation* of a word w if and only if $\mathrm{alph}(w) = \mathrm{alph}(w')$ and, for every $a \in \mathrm{alph}(w)$, $|w|_a = |w'|_a$. If we wish to refer to the symbol at a certain position j, $1 \leq j \leq n$, in a word $w = a_1 \cdot a_2 \cdots a_n$, $a_i \in \Sigma$, $1 \leq i \leq n$, we use $w[j] := a_j$. Furthermore, for each j, j', $1 \leq j < j' \leq |w|$, let $w[j, j'] := a_j \cdot a_{j+1} \cdots a_{j'}$ and $w[j, -] := w[j, |w|]$. In case that $j > |w|$, we define $w[j, -] = \varepsilon$.

Next, we formally define the notion of shuffle words, which have already been informally explained in the introduction. Let u and v be words over the alphabet Σ. The *shuffle operation*, denoted by $\sqcup\!\sqcup$, is a binary operation on words, defined by

$$u \sqcup\!\sqcup v := \{x_1 y_1 x_2 y_2 \cdots x_n y_n \mid n \in \mathbb{N}, x_i, y_i \in (\Sigma \cup \{\varepsilon\}), 1 \leq i \leq n, \\ u = x_1 x_2 \cdots x_n, v = y_1 y_2 \cdots y_n\}.$$

The shuffle operation is extended to the case of more than two words in an inductive way. Let $w_1, w_2, \ldots, w_k \in \Sigma^*$ be arbitrary words. Then $w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k := \{w_1 \sqcup\!\sqcup u \mid u \in w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k\}$. Furthermore, $\Gamma := w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$ is called the *shuffle* of $w_1, \ldots, w_k$ and each word $w \in \Gamma$ is a *shuffle word* of $w_1, \ldots, w_k$. For example, $\mathbf{cbaaabcbcb} \in \mathbf{bbc} \sqcup\!\sqcup \mathbf{caab} \sqcup\!\sqcup \mathbf{bac}$.

Finally, we introduce a special property of words that is important for our main problem. For an arbitrary $w \in \Sigma^*$ and any $b \in \mathrm{alph}(w)$ let l, r, $1 \leq l, r \leq |w|$, be chosen such that $w[l] = w[r] = b$ and there exists no k, $k < l$, with $w[k] = b$ and no k', $r < k'$, with $w[k'] = b$. Then the *scope of* b *in* w ($\mathrm{sc}_w(b)$ for short) is defined by $\mathrm{sc}_w(b) := (l, r)$. Note that in the case that for some word w we have $w[j] = b$ and $|w|_b = 1$, the scope of b in w is (j, j). Now we are ready to define the so called *scope coincidence degree*: Let $w \in \Sigma^*$ be an arbitrary word and, for each i, $1 \leq i \leq |w|$, let

$$\mathrm{scd}_i(w) := |\{b \in \Sigma \mid b \neq w[i], \mathrm{sc}_w(b) = (l, r) \text{ and } l < i < r\}| \ .$$

We call $\mathrm{scd}_i(w)$ the scope coincidence degree of position i in w. Furthermore, the scope coincidence degree of the word w is defined by

$$\mathrm{scd}(w) := \max\{\mathrm{scd}_i(w) \mid 1 \leq i \leq |w|\} \ .$$

As an example, we now consider the word $w := \mathbf{acacbbdeabcedefdeff}$. It can easily be verified that $\mathrm{scd}_8(w) = \mathrm{scd}_9(w) = 4$ and $\mathrm{scd}_i(w) < 4$ if $i \notin \{8, 9\}$. Hence, $\mathrm{scd}(w) = 4$.

3. The Problem of Computing Shuffle Words with Minimum Scope Coincidence Degree

As described in the practical motivation given in the introduction, it is our goal to solve the following problem: for given input words which model sequences of memory accesses, construct a shuffle word, such that, for any memory access in the shuffle word, the maximum number of different values that already have been accessed and shall again be accessed later on is minimal. For example, $w := \mathbf{abaabccbc}$ is a shuffle word of $\mathbf{abacbc}$ and $\mathbf{abc}$ and we can observe that, for each position i in w, there exists at most one other symbol different from $w[i]$ that has an occurrence to both sides of position i. On the other hand, for the shuffle word $\mathbf{abacbcabc}$, there is an occurrence of $\mathbf{a}$ and $\mathbf{b}$ to both sides of the occurrence of $\mathbf{c}$ at position 4. The maximum over all these numbers of symbols occurring to both sides of an occurrence of another symbol is precisely the scope coincidence degree. Consequently, our main problem can be described as the problem of computing, for given input words, a shuffle word with a minimum scope coincidence degree.

Problem 3.1 For an arbitrary alphabet Σ, let the problem $\mathrm{SWminSCD}_\Sigma$ be the problem of finding, for given $w_i \in \Sigma^*$, $1 \leq i \leq k$, a shuffle word $w \in w_1 \sqcup \ldots \sqcup w_k$ with minimum scope coincidence degree.

We wish to point out that the alphabet Σ in the definition of $\mathrm{SWminSCD}_\Sigma$ is a constant and not part of the input. More precisely, the input words for the problem $\mathrm{SWminSCD}_\Sigma$ are required to exclusively contain symbols from the alphabet Σ, which shall be important for complexity considerations.

Let $w_1, w_2, \ldots, w_k$ be some words over the alphabet Σ. We can note that the cardinality of the shuffle $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ is, in the worst case, given by the multinomial coefficient [2]:

$$|w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k| \leq \binom{n}{|w_1|, |w_2|, \ldots, |w_k|} = \frac{n!}{|w_1|! \times |w_2|! \times \ldots \times |w_k|!} .$$

This directly implies that the naive approach to solving $\mathrm{SWminSCD}_\Sigma$, i. e., to enumerate all elements in the shuffle $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ in order to find one with minimum scope coincidence degree, is not suitable, since the search space of this procedure can be exponentially large. Hence, a polynomial time algorithm cannot simply search the whole shuffle $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$, which implies that a more sophisticated strategy is required. Before we present a successful approach to $\mathrm{SWminSCD}_\Sigma$ in Section 5, we wish to discuss some simple observations. First, we note that solving $\mathrm{SWminSCD}_\Sigma$ on input $w_1, w_2, \ldots, w_k$ by first computing a minimal shuffle word w of w_1 and w_2 (ignoring $w_3, \ldots, w_n$) and then solving $\mathrm{SWminSCD}_\Sigma$ on the smaller input $w, w_3 \ldots, w_n$ and so on is not possible. This can be easily comprehended by considering the words $w_1 := \mathbf{ab}$ and $w_2 := \mathbf{bc}$ and observing that $w := \mathbf{abbc}$ is a shuffle word of w_1 and w_2 that is optimal, since $\mathrm{scd}(w) = 0$. Now, it is not possible to shuffle w with $w_3 := \mathbf{cba}$ in such a way that the resulting shuffle word has a scope coincidence degree of less than 2; however, $w' := w_2 \cdot w_3 \cdot w_1 = \mathbf{bccbaab} \in w_1 \sqcup w_2 \sqcup w_3$ and $\mathrm{scd}(w') = 1$. We can further note that w is in fact the only optimal shuffle word of w_1 and w_2, thus, in terms of the above described approach, we necessarily have to start with a shuffle word of w_1 and w_2 that is *not* optimal in order to obtain an optimal shuffle word of all three words w_1, w_2 and w_3.

Intuitively, it seems obvious that the scope coincidence degree only depends on the leftmost and rightmost occurrences of the symbols. In other words, removing a symbol from a word that does not constitute a leftmost or rightmost occurrence

should not change the scope coincidence degree of that word. For instance, if we consider a word $w := \alpha \cdot c \cdot \beta$, where c is a symbol occurring in both α and β, then all symbols in the word w that are in the scope of c are still in the scope of c with respect to the word $\alpha \cdot \beta$. Consequently, we can first remove all occurrences of symbols that are neither leftmost nor rightmost occurrences, then solve $\mathrm{SWminSCD}_\Sigma$ on these reduced words and finally insert the removed occurrences into the shuffle word in such a way that the scope coincidence degree does not increase. This reduction of the input words results in a smaller, but still exponentially large search space. Hence, this approach does not seem to help us solving $\mathrm{SWminSCD}_\Sigma$ in polynomial time. For completeness, we discuss this matter in a bit more detail in the following subsection.

3.1 *Scope Reduced Words*

As mentioned above, all symbols in the word $w := \alpha \cdot c \cdot \beta$ that are in the scope of c, where c is a symbol occurring in both α and β, are still in the scope of c with respect to the word $\alpha \cdot \beta$. However, in order to conclude $\mathrm{scd}(w) = \mathrm{scd}(\alpha \cdot \beta)$, we also have to consider the following situation. In case that $\mathrm{scd}_{|\alpha|+1}(w) = \mathrm{scd}(w)$ (i. e., the position of the symbol c under consideration has maximum scope coincidence degree in w) it is no longer as obvious that this particular occurrence of c can be removed without changing the scope coincidence degree of w.

In this case, we can show that there must exist a position i in w, different from position $|\alpha| + 1$, that also has a maximum scope coincidence degree, i. e., $\mathrm{scd}_i(w) = \mathrm{scd}_{|\alpha|+1}(w)$:

LEMMA 3.2 *Let $w := \alpha \cdot c \cdot \beta \in \Sigma^*$, where $c \in \Sigma$, $1 \leq |\alpha|$, $1 \leq |\beta|$. If $c \in (\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta))$, then $\mathrm{scd}(w) = \mathrm{scd}(\alpha \cdot \beta)$.*

Proof Let $w' := \alpha \cdot \beta$. Since the occurrence of c at position $|\alpha| + 1$ is neither a leftmost nor a rightmost occurrence, it is obvious that $\mathrm{scd}_i(w) = \mathrm{scd}_i(w')$, $1 \leq i \leq |\alpha|$, and $\mathrm{scd}_{i+1}(w) = \mathrm{scd}_i(w')$, $|\alpha| + 1 \leq i \leq |\alpha \cdot \beta|$. First, we observe that if $\mathrm{scd}_{|\alpha|+1}(w) \leq \mathrm{scd}_{|\alpha|}(w)$, then we can conclude

$$\begin{aligned}\mathrm{scd}(w) &= \max\{\mathrm{scd}_i(w) \mid 1 \leq i \leq |w|, i \neq |\alpha| + 1\} \\ &= \max\{\mathrm{scd}_i(w') \mid 1 \leq i \leq |w'|\} \\ &= \mathrm{scd}(w')\,.\end{aligned}$$

So in order to prove the statement of the lemma, it is sufficient to show that $\mathrm{scd}_{|\alpha|+1}(w) \leq \mathrm{scd}_{|\alpha|}(w)$. Now, as $1 \leq |\alpha|$, there exists a $b \in \Sigma$ such that $\alpha = \alpha' \cdot b$. In case that $b = c$, $\mathrm{scd}_{|\alpha|+1}(w) = \mathrm{scd}_{|\alpha|}(w)$. Therefore, in the following, we assume that $b \neq c$ and define the set $\Gamma := \{a \mid a \in (\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta)) \setminus \{b, c\}\}$. There are two cases depending on whether or not $b \in \mathrm{alph}(\beta)$.

If $b \in \mathrm{alph}(\beta)$, then $\mathrm{scd}_{|\alpha|+1}(w) = |\Gamma| + |\{b\}|$ and $\mathrm{scd}_{|\alpha|}(w) = |\Gamma| + |\{c\}|$. Hence, $\mathrm{scd}_{|\alpha|+1}(w) = \mathrm{scd}_{|\alpha|}(w)$. If, on the other hand, $b \notin \mathrm{alph}(\beta)$, then $\mathrm{scd}_{|\alpha|+1}(w) = |\Gamma|$ and $\mathrm{scd}_{|\alpha|}(w) = |\Gamma| + |\{c\}|$, which implies $\mathrm{scd}_{|\alpha|+1}(w) < \mathrm{scd}_{|\alpha|}(w)$. ■

By iteratively applying Lemma 3.2, it can easily be seen that all occurrences of symbols from a word that are neither leftmost nor rightmost occurrences can be removed without changing its scope coincidence degree. The next definition shall formalise that procedure.

DEFINITION 3.3 *Let $w = b_1 \cdot b_2 \cdots b_n$, $b_i \in \Sigma$, $1 \leq i \leq n$, be arbitrarily chosen and, for each i, $1 \leq i \leq n$, let $c_i := \varepsilon$ if $b_i \in (\mathrm{alph}(w[1, i-1]) \cap \mathrm{alph}(w[i+1, -]))$*

and $c_i := b_i$ *otherwise. The word* $c_1 \cdot c_2 \cdots c_n$ *(denoted by* $\mathrm{sr}(w)$*) is called the* scope reduced version of w. *An arbitrary word* $v \in \Sigma^*$*, such that, for each* $b \in \mathrm{alph}(w)$*,* $|w|_b \leq 2$*, is said to be* scope reduced.

We can now use the previous result and Definition 3.3 in order to show that, regarding the problem $\mathrm{SWminSCD}_\Sigma$, we can restrict our considerations to input words that are scope reduced:

LEMMA 3.4 *Let* $w_1, w_2, \ldots, w_k \in \Sigma^*$. *There is a word* $u \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ *with* $\mathrm{scd}(u) = m$ *if and only if there is a word* $v \in \mathrm{sr}(w_1) \sqcup \mathrm{sr}(w_2) \sqcup \ldots \sqcup \mathrm{sr}(w_k)$ *with* $\mathrm{scd}(v) = m$.

Proof We prove the *only if* direction by showing that any $u \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ can be transformed into a $v \in \mathrm{sr}(w_1) \sqcup \mathrm{sr}(w_2) \sqcup \ldots \sqcup \mathrm{sr}(w_k)$ with $\mathrm{scd}(u) = \mathrm{scd}(v)$. The *if* direction shall be shown in a similar way.

The basic idea is that all the symbols from the words w_i, $1 \leq i \leq k$, that are neither leftmost nor rightmost occurrences, can simply be removed from a shuffle word of $w_1, \ldots, w_k$ in order to obtain a shuffle word of $\mathrm{sr}(w_1), \ldots, \mathrm{sr}(w_k)$, and, analogously, inserting these symbols anywhere, but always between two occurrences of the same symbol, into a shuffle word of $\mathrm{sr}(w_1), \ldots, \mathrm{sr}(w_k)$ results in a shuffle word of $w_1, \ldots, w_k$. The equivalence of the scope coincidence degree can then be established by Lemma 3.2.

Let $u \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ be arbitrarily chosen. By definition of a shuffle, we can assume that all symbols in u are marked with one of the numbers in $\{1, 2, \ldots, k\}$ in such a way that, for each i, $1 \leq i \leq k$, by deleting all symbols from u that are not marked with i, we obtain exactly w_i. Hence, all symbols in u are of form $b^{(i)}$, where $b \in \Sigma$ and $1 \leq i \leq k$. Next, we obtain a word v from u in the following way. For each $b \in \Sigma$ and each i, $1 \leq i \leq k$, we delete all occurrences of symbols $b^{(i)}$ that are neither leftmost nor rightmost occurrences of the symbol $b^{(i)}$. After that, we unmark all symbols. Since, for each i, $1 \leq i \leq k$, we removed all symbols in u originating from w_i except the left- and rightmost occurrences in w_i, we conclude that v is a shuffle word of $\mathrm{sr}(w_1), \mathrm{sr}(w_2), \ldots, \mathrm{sr}(w_k)$ and, by Lemma 3.2, we can conclude that $\mathrm{scd}(u) = \mathrm{scd}(v)$.

In order to prove the *if* direction, we arbitrarily choose a word $v \in \mathrm{sr}(w_1) \sqcup \mathrm{sr}(w_2) \sqcup \ldots \sqcup \mathrm{sr}(w_k)$. Again, we may assume that all symbols in v are marked in the same way as before. Now, for every $b \in \Sigma$ and every i, $1 \leq i \leq k$, if $|w_i|_b > 2$, we define $n_{b,i} := |w_i|_b - 2$. Next, we construct a word u from v by applying the following algorithm:

1: Set $u \leftarrow v$
2: **for all** $b \in \Sigma$ **do**
3: **for all** i, $1 \leq i \leq k$, **do**
4: **if** $|w_i|_b > 2$ **then**
5: Let α, β, γ such that $u = \alpha \cdot b^{(i)} \cdot \beta \cdot b^{(i)} \cdot \gamma$
6: Set $u \leftarrow \alpha \cdot b \cdot b^{n_{b,i}} \cdot \beta \cdot b \cdot \gamma$
7: **end if**
8: **end for**
9: **end for**

For every $b \in \Sigma$ and every i, $1 \leq i \leq k$, with $|w_i|_b > 2$, we can conclude that there are exactly 2 occurrences of $b^{(i)}$ in u, thus, the factorisation in line 5 is unique. In line 6, we simply insert $n_{b,i} = |w_i|_b - 2$ occurrences of symbol b in between the two occurrences of $b^{(i)}$, which we unmark. Consequently, the word u constructed by the above given algorithm is a shuffle word of $w_1, w_2, \ldots, w_k$ and, by Lemma 3.2, we can conclude that $\mathrm{scd}(u) = \mathrm{scd}(v)$. ■

The previous result also shows how to obtain a solution for $\mathrm{SWminSCD}_\Sigma$ on input words $w_1, w_2, \ldots, w_k$ from a solution for $\mathrm{SWminSCD}_\Sigma$ on the scope reduced input words $\mathrm{sr}(w_1), \mathrm{sr}(w_2), \ldots, \mathrm{sr}(w_k)$. Although the above made observations are more or less irrelevant for our main results, we shall use them at the very end of this work in order to obtain a better complexity bound.

In the following section, we shall establish basic results on the scope coincidence degree of words. These results shall then be applied later on in order to analyse the scope coincidence degree of shuffle words.

4. Further Properties of the Scope Coincidence Degree

In the present section, we investigate the scope coincidence degree in more detail. Our central question is how we can transform a word without increasing its scope coincidence degree. We can first note that, for some word w and some position i, $1 \leq i \leq |w|$, if we permute the prefix $w[1, i-1]$ or the suffix $w[i+1, -]$, then the scope coincidence degree of position i in w, i. e., $\mathrm{scd}_i(w)$, does not change. This is due to the fact that $\mathrm{scd}_i(w)$ is the number of distinct symbols that occur to both sides of position i, which is not affected by a permutation of $w[1, i-1]$ or $w[i+1, -]$.

PROPOSITION 4.1 *Let $u, v \in \Sigma^*$ with $|u| = |v|$. If, for some i, $1 \leq i \leq |u|$, $u[i] = v[i]$ and $u[1, i-1]$ is a permutation of $v[1, i-1]$ and $u[i+1, -]$ is a permutation of $v[i+1, -]$, then $\mathrm{scd}_i(u) = \mathrm{scd}_i(v)$.*

This implies that we can permute the part to the right and to the left of some position in a word without changing the scope coincidence degree of this specific position. On the other hand, the scope coincidence degree of the positions in the permuted parts do not necessarily remain unchanged, which means that the scope coincidence degree of the whole word may change. It can be shown, however, that if a factor of a word w contains no leftmost occurrence of a symbol with respect to w (it may contain rightmost occurrences of symbols), then this factor can be replaced by a permutation of itself without changing the scope coincidence degree of the whole word:

LEMMA 4.2 *Let α, β, π, $\pi' \in \Sigma^*$, where π is a permutation of π' and $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$. Then $\mathrm{scd}(\alpha \cdot \pi \cdot \beta) = \mathrm{scd}(\alpha \cdot \pi' \cdot \beta)$.*

Proof We prove $\mathrm{scd}(v) = \mathrm{scd}(v')$, where $v := \alpha \cdot \pi \cdot \beta$ and $v' := \alpha \cdot \pi' \cdot \beta$. By Proposition 4.1, we can conclude that, for each i, with $1 \leq i \leq |\alpha|$ or $|\alpha \cdot \pi| + 1 \leq i \leq |v|$, $\mathrm{scd}_i(v) = \mathrm{scd}_i(v')$. So it remains to examine the numbers $\mathrm{scd}_i(v)$, $\mathrm{scd}_i(v')$, where $|\alpha| + 1 \leq i \leq |\alpha \cdot \pi|$. In particular, we take a closer look at $\mathrm{scd}_{|\alpha|+1}(v)$ and $\mathrm{scd}_{|\alpha|+1}(v')$, which are determined by the number of symbols different from $\pi[1]$ ($\pi'[1]$, respectively) that occur in both factors α and $\pi[2, -] \cdot \beta$ ($\pi'[2, -] \cdot \beta$, respectively). These symbols can be divided into two sets, the set of symbols occurring in $\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta)$ but not in $\mathrm{alph}(\pi)$ ($\mathrm{alph}(\pi')$, respectively) on the one hand, and the set $\mathrm{alph}(\pi) \setminus \{\pi[1]\}$ ($\mathrm{alph}(\pi') \setminus \{\pi'[1]\}$, respectively) on the other hand. This is due to the fact that $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$; thus, all symbols in $\mathrm{alph}(\pi) \setminus \{\pi[1]\}$ ($\mathrm{alph}(\pi') \setminus \{\pi'[1]\}$, respectively) have an occurrence to the left and to the right of position $|\alpha| + 1$ in v (v', respectively). Therefore, $\mathrm{scd}_{|\alpha|+1}(v) = \mathrm{scd}_{|\alpha|+1}(v') = (m-1) + r$, where $m := |\mathrm{alph}(\pi)|$ and $r := |(\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta)) \setminus \mathrm{alph}(\pi)|$. If we consider the numbers $\mathrm{scd}_i(v)$, $\mathrm{scd}_i(v')$, $|\alpha| + 2 \leq i \leq |\alpha \cdot \pi|$ we encounter the same situation with the only difference that not necessarily all $m-1$ symbols in $\mathrm{alph}(\pi) \setminus \{v[i]\}$ ($\mathrm{alph}(\pi') \setminus \{v'[i]\}$, respectively) have to occur to the right of position i. Hence, $\mathrm{scd}_{|\alpha|+i}(v) = r + m'$ and $\mathrm{scd}_{|\alpha|+i}(v) = r + m''$, where $m' \leq (m-1)$ and $m'' \leq (m-1)$. In conclusion,

- $\max\{\mathrm{scd}_i(v) \mid |\alpha| + 1 \leq i \leq |\alpha \cdot \pi|\} = \mathrm{scd}_{|\alpha|+1}(v)$,
- $\max\{\mathrm{scd}_i(v') \mid |\alpha| + 1 \leq i \leq |\alpha \cdot \pi'|\} = \mathrm{scd}_{|\alpha|+1}(v')$, and
- $\mathrm{scd}_{|\alpha|+1}(v) = \mathrm{scd}_{|\alpha|+1}(v')$.

Thus, $\mathrm{scd}(v) = \mathrm{scd}(v')$. ■

In the next two lemmas, we show how words can be manipulated, such that their scope coincidence degree does not increase. The statement of the first lemma can be paraphrased in the following way. Let w be a word and let i and j, $1 \leq i < j \leq |w|$, be two positions of w with $w[i] = w[j] = b$, $b \in \Sigma$. Furthermore, we assume that $w[1, i-1]_b = 0$, i. e., the occurrence of b at position i is the leftmost occurrence of b. We can now move the occurrence of b at position j to the left and, as long as this symbol is not moved to the left of position i, such an operation does not increase the scope coincidence degree of w. At first glance, this observation seems evident, since moving an occurrence of b in this way shortens the scope of symbol b or leaves it unchanged. However, it can happen that, after moving the occurrence of b, the scope coincidence degree of the new position of b has increased compared to its old position. Intuitively, this is the case if we move this certain b into a region of the word where many scopes coincide. This possible increase of the scope coincidence degree of that certain position, as shown in the next lemma, does not affect the scope coincidence degree of the whole word.

LEMMA 4.3 *For all $\alpha, \beta, \gamma \in \Sigma^*$ and for each $b \in \Sigma$ with $b \in \mathrm{alph}(\alpha)$,*

$$\mathrm{scd}(\alpha \cdot b \cdot \beta \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta \cdot b \cdot \gamma)\,.$$

Proof Let $w := \alpha \cdot b \cdot \beta \cdot \gamma$ and $w' := \alpha \cdot \beta \cdot b \cdot \gamma$. Furthermore, let $j := |\alpha \cdot b|$. We prove the statement of the lemma by showing that $\mathrm{scd}_i(w) \leq \mathrm{scd}(w')$, for each i, $1 \leq i \leq |w|$.

By applying Proposition 4.1, we can conclude that for each i with $1 \leq i \leq j-1$ or $|\alpha \cdot b \cdot \beta| + 1 \leq i \leq |w|$, $\mathrm{scd}_i(w) = \mathrm{scd}_i(w')$. For the positions in w that are in factor β, i. e. the positions i with $j+1 \leq i \leq j+|\beta|$, we observe the following. For each i, $j+1 \leq i \leq j+|\beta|$, there is a certain number of symbols different from symbol $w[i]$ that occur to the left and to the right of position i in w. Regarding w', as in w' the factor β is simply shifted one position to the left, the very same symbols occur to the left and to the right of position $i-1$ in w'. In addition to that, we know that symbol b has an occurrence to the left and to the right of position $i-1$ in w', whereas in w and with respect to position i, this is not necessarily the case. Therefore, we can conclude that $\mathrm{scd}_i(w) \leq \mathrm{scd}_{i-1}(w')$, $j+1 \leq i \leq j+|\beta|$.

So far, we showed that $\mathrm{scd}_i(w) \leq \mathrm{scd}(w')$ for each i with $1 \leq i \leq |w|$ and $i \neq j$. Thus, it only remains to take a closer look at position j in w and, in particular, at the number $\mathrm{scd}_j(w)$. In general, it is possible that $\mathrm{scd}_j(w) > \mathrm{scd}_{|\alpha\cdot\beta|+1}(w')$, but we shall see that always $\mathrm{scd}_j(w) \leq \mathrm{scd}(w')$ holds. We consider the symbol y at position $j-1$ in w, i. e. the last symbol of the factor α (recall that $|\alpha| \geq 1$). Now we can write w as $w := \alpha' \cdot y \cdot b \cdot \beta \cdot \gamma$, where $\alpha = \alpha' \cdot y$. If $y = b$, then obviously $\mathrm{scd}_j(w) = \mathrm{scd}_{j-1}(w)$ and we already know that $\mathrm{scd}_{j-1}(w) = \mathrm{scd}_{j-1}(w')$, hence, $\mathrm{scd}_j(w) \leq \mathrm{scd}(w')$. We assume that, on the other hand, $y \neq b$. Furthermore, we assume to the contrary that $\mathrm{scd}_j(w) = m > \mathrm{scd}(w')$. This implies that $|\Gamma| = m$, where $\Gamma := |(\mathrm{alph}(\alpha) \cap \mathrm{alph}(\beta \cdot \gamma)) \setminus \{b\}|$. Next we consider the set $\Gamma' = (\mathrm{alph}(\alpha') \cap \mathrm{alph}(\beta \cdot b \cdot \gamma)) \setminus \{y\}$ and note that, since $b \in \mathrm{alph}(\alpha')$, $b \in \Gamma'$. We observe now that we have $|\Gamma| = |\Gamma'|$ if $y \in \Gamma$, and $|\Gamma| < |\Gamma'|$ if $y \notin \Gamma$, hence, $|\Gamma| \leq |\Gamma'|$ and, as $|\Gamma'| = \mathrm{scd}_{j-1}(w')$, $m \leq \mathrm{scd}_{j-1}(w')$ is implied, which is a contradiction. ■

By a repeated application of Lemma 4.3, it can be easily shown that it is also possible to move several symbols in a word to the left without increasing the scope

coincidence degree. For the sake of convenience, we now state this observation in a form that is particularly tailored to our later applications for shuffle words.

LEMMA 4.4 *Let* α, γ, $\beta_i \in \Sigma^*$, $0 \leq i \leq n$, $n \in \mathbb{N}$, *and let* $d_i \in \Sigma$, $1 \leq i \leq n$, *such that* $d_i \in \mathrm{alph}(\alpha)$, $1 \leq i \leq n$. *Then*

$$\mathrm{scd}(\alpha \cdot d_1 \cdot d_2 \cdots d_n \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma)\,.$$

Proof We prove $\mathrm{scd}(w) \leq \mathrm{scd}(w')$, where

- $w := \alpha \cdot d_1 \cdot d_2 \cdots d_n \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$,
- $w' := \alpha \cdot \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma$.

We can obtain a word u_1 from w' by moving d_1 to the left until it is positioned directly to the left of factor β_1. Furthermore, for each i, $2 \leq i \leq n$, we can obtain a word u_i from u_{i-1} by moving the symbol d_i to the left in the same way (i. e. d_i is then positioned between d_{i-1} and β_1). Obviously, $u_n = w$ and, by Lemma 4.3, $\mathrm{scd}(u_1) \leq \mathrm{scd}(w')$ and $\mathrm{scd}(u_i) \leq \mathrm{scd}(u_{i-1})$, $2 \leq i \leq n$, hence, $\mathrm{scd}(w) \leq \mathrm{scd}(w')$. ■

We can observe that, in the previous lemma, the d_i, $1 \leq i \leq n$, can be any symbols and the only condition that needs to be satisfied is that all these symbols have at least one occurrence in the prefix α. Furthermore, from Lemma 4.2, it directly follows that the order of the symbols d_i, $1 \leq i \leq n$, between α and β_1 can be arbitrarily changed without increasing the scope coincidence degree of the word. However, since in the following we are concerned with shuffle words, it is convenient to state Lemma 4.4 as given above, i. e., in such a way that the relative order of the symbols that are moved is preserved.

5. Solving the Problem SWminSCD$_\Sigma$

In this section, we show how the problem SWminSCD$_\Sigma$ can be solved efficiently. To this end, we first define a general way of constructing shuffle words. After that, we identify a simpler and standardised way of producing well-formed shuffle words and then, by applying the lemmas given in the previous section, we show that among all those well-formed shuffle words that can be constructed in the simple way, there must be at least one with minimum scope coincidence degree. Since the set of well-formed shuffle words is considerably smaller than the set of all shuffle words, it follows that this method can be carried out in polynomial time.

In the following, we consider words as stack-like data structures, i. e., we interpret words $w_1, w_2, \ldots, w_k \in \Sigma^*$ as stacks where the leftmost symbols $w_i[1]$, $1 \leq i \leq k$, are the topmost stack elements of these stacks. Now, by successively applying the pop operation, we can empty these stacks and every time we pop a symbol from the stack, it is appended to the end of an initially empty word w. Obviously, the word w that is obtained as soon as all the stacks are empty is a shuffle word of $w_1, w_2, \ldots, w_k$.

It seems useful to reason about different ways of constructing a shuffle word rather than about actual shuffle words, as this allows us to ignore the fact that in general a shuffle word can be constructed in several completely different ways. In particular the following unpleasant situation seems to complicate the analysis of shuffle words. If we consider a shuffle word w of the words $w_1, w_2, \ldots, w_k$, it might be desirable to know, for a symbol b on a certain position j, which w_i, $1 \leq i \leq k$, is the origin of that symbol. Obviously, this depends on how the shuffle word has been constructed from the words w_i, $1 \leq i \leq k$, and for different ways of constructing w, the symbol b on position j may originate from different words w_i, $1 \leq i \leq k$.

In particular, if we want to alter shuffle words by moving certain symbols, it is essential to know the origin words w_i, $1 \leq i \leq k$, of the symbols, as this determines how they can be moved without destroying the shuffle properties.

We now formalise the way of constructing a shuffle word by utilising the stack analogy introduced above. An arbitrary configuration (of the content) of the stacks corresponding to words w_i, $1 \leq i \leq k$, can be given as a tuple $(v_1, \ldots, v_k)$ of suffixes, i. e. $w_i = u_i \cdot v_i$, $1 \leq i \leq k$. Such a configuration $(v_1, \ldots, v_k)$ is then changed into another configuration $(v_1, \ldots, v_{i-1}, v_i', v_{i+1}, \ldots, v_k)$, by a pop operation, where $v_i = b \cdot v_i'$ for some i, $1 \leq i \leq k$, and for some $b \in \Sigma$. Initially, we start with the stack content configuration $(w_1, \ldots, w_k)$ and as soon as all the stacks are empty, which can be represented by $(\varepsilon, \ldots, \varepsilon)$, our shuffle word is complete. Hence, we can represent a way to construct a shuffle word by a sequence of these tuples of stack contents:

DEFINITION 5.1 *A* construction sequence *for words $w_1, w_2, \ldots, w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, is a sequence $s := (s_0, s_1, \ldots, s_m)$, $m := |w_1 \cdots w_k|$ such that*

- *$s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$, where, for each i, $0 \leq i \leq m$, and for each j, $1 \leq j \leq k$, $v_{i,j}$ is a suffix of w_j ,*
- *$s_0 = (w_1, \ldots, w_k)$ and $s_m = (\varepsilon, \varepsilon, \ldots, \varepsilon)$,*
- *for each i, $0 \leq i \leq m-1$, there exists a j_i, $1 \leq j_i \leq k$, and a $b_i \in \Sigma$ such that $v_{i,j_i} = b_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$.*

The shuffle word $w = b_0 \cdot b_1 \cdots b_{m-1}$ is said to correspond *to s. In a step from s_i to s_{i+1}, $0 \leq i \leq m-1$, of s, we say that the symbol b_{i+1} is* consumed.

To illustrate the definition of construction sequences, we consider an example construction sequence $s := (s_0, s_1, \ldots, s_9)$ corresponding to a shuffle word of the words $w_1 := \mathbf{abacbc}$ and $w_2 := \mathbf{abc}$:

$$s := ((\mathbf{abacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{bc}), (\mathbf{bacbc}, \mathbf{c}),$$
$$(\mathbf{acbc}, \mathbf{c}), (\mathbf{acbc}, \varepsilon), (\mathbf{cbc}, \varepsilon), (\mathbf{bc}, \varepsilon), (\mathbf{c}, \varepsilon), (\varepsilon, \varepsilon)) .$$

The shuffle word corresponding to s is $w := \mathbf{aabbcacbc}$, and it is easy to see that $\mathrm{scd}(w) = 2$.

In the next definition, we introduce a certain property of construction sequences that can be easily described in an informal way. Recall that in an arbitrary step from s_i to s_{i+1} of a construction sequence s, exactly one symbol b is consumed. Hence, at each position $s_i = (v_1, \ldots, v_k)$ of a construction sequence, we have a part u of already consumed symbols, which is actually a prefix of the shuffle word we are about to construct and some suffixes $v_1, \ldots, v_k$ that remain to be consumed. A symbol b that is consumed can be an *old* symbol that already occurs in the part u or it can be a *new* symbol that is consumed for the first time. Now the special property to be introduced next is that this consumption of symbols is greedy with respect to old symbols: Whenever a new symbol b is consumed in a step from s_i to $s_{i+1} = (v_1, \ldots, v_k)$, we require the construction sequence to first consume as many old symbols as possible from the remaining $v_1, \ldots, v_k$ before another new symbol is consumed. For the sake of uniqueness, this greedy consumption of old symbols shall be defined in a canonical order, i. e. we first consume all the old symbols from v_1, then all the old symbols from v_2 and so on. However, this consumption is canonical only with respect to old symbols. Thus, there are still several possible greedy construction sequences for some input words w_i, $1 \leq i \leq k$, since whenever a new symbol is consumed, we have a choice of k possible suffixes to take this symbol from. We formally define this greedy property of construction sequences.

DEFINITION 5.2 *Let $w \in w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, and let $s := (s_0, s_1, \ldots, s_{|w|})$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq |w|$, be an arbitrary construction sequence for w. An element s_i, $1 \leq i \leq |w| - 1$, of s satisfies the* greedy property *if and only if $w[i] \notin \mathrm{alph}(w[1, i-1])$ implies that for each j, $1 \leq j \leq k$, $s_{i+|u_1 \cdots u_j|} = (\overline{v}_{i,1}, \ldots, \overline{v}_{i,j}, v_{i,j+1}, \ldots, v_{i,k})$, where $v_{i,j} = u_j \cdot \overline{v}_{i,j}$ and u_j is the longest prefix of $v_{i,j}$ such that $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1, i])$.*

A construction sequence $s := (s_0, s_1, \ldots, s_{|w|})$ for some $w \in \Sigma^*$ is a greedy construction sequence *if and only if, for each i, $1 \leq i \leq |w| - 1$, s_i satisfies the greedy property. A shuffle word w that corresponds to a greedy construction sequence is a* greedy shuffle word.

As an example, we again consider the words $w_1 = \mathbf{abacbc}$ and $w_2 = \mathbf{abc}$. This time, we present a greedy construction sequence $s := (s_0, s_1, \ldots, s_9)$ for w_1 and w_2:

$$s := ((\mathbf{abacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{bc}), (\mathbf{bacbc}, \mathbf{c}),$$
$$(\mathbf{acbc}, \mathbf{c}), (\mathbf{cbc}, \mathbf{c}), (\mathbf{cbc}, \varepsilon), (\mathbf{bc}, \varepsilon), (\mathbf{c}, \varepsilon), (\varepsilon, \varepsilon)) .$$

Obviously, the shuffle word $w := \mathbf{aabbaccbc}$ corresponds to the construction sequence s and $\mathrm{scd}(w) = 1$. To show that s is a greedy construction sequence, it is sufficient to observe that s_1, s_3 and s_6 (the elements where a new symbol is consumed) satisfy the greedy property. We only show that s_3 satisfies the greedy property as s_1 and s_6 can be handled analogously. First, we recall that $s_3 = (\mathbf{bacbc}, \mathbf{c})$ and note that, in terms of Definition 5.2, we have $u_1 := \mathbf{ba}$, $\overline{v}_{3,1} := \mathbf{cbc}$, $u_2 := \varepsilon$ and $\overline{v}_{3,2} := \mathbf{c}$. By definition, s_3 only satisfies the greedy property if $s_{3+|u_1|} = (\overline{v}_{3,1}, v_{3,2})$ and $s_{3+|u_1 \cdot u_2|} = (\overline{v}_{3,1}, \overline{v}_{3,2})$. Since $|u_1| = |u_1 \cdot u_2| = 2$, $\overline{v}_{3,1} = \mathbf{cbc}$, $v_{3,2} = \overline{v}_{3,2} = \mathbf{c}$ and $s_5 = (\mathbf{cbc}, \mathbf{c})$, this clearly holds.

In the next definition, an algorithm shall be presented that converts an arbitrary construction sequence into a greedy one. In order to illustrate how this can be done, we define $s := (s_0, s_1, \ldots, s_m)$ to be an arbitrary construction sequence that is not a greedy construction sequence. Thus, there exists an element s_i that does not satisfy the greedy property and, without loss of generality, we assume that s_i is the leftmost such element. Now our algorithm simply redefines all the elements s_j, $i+1 \leq j \leq m$, in such a way that s_i satisfies the greedy property (and the resulting sequence is still a construction sequence). By applying this algorithm iteratively, we can obtain a greedy construction sequence.

DEFINITION 5.3 *We define an algorithm G that transforms a construction sequence. Let $s := (s_0, s_1, \ldots, s_m)$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$, be an arbitrary construction sequence that corresponds to a shuffle word w. In the case that s is a greedy construction sequence, we define $G(s) := s$. If s is not a greedy construction sequence, then let p, $1 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. Furthermore, for each j, $1 \leq j \leq k$, let u_j be the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1, p])$ and let $v_{p,j} = u_j \cdot \overline{v}_{p,j}$. For each j, $1 \leq j \leq k$, let $\sigma_j : \Sigma^* \to \Sigma^*$ be a mapping defined by $\sigma_j(x) := \overline{v}_{p,j}$ if $|x| > |\overline{v}_{p,j}|$ and $\sigma_j(x) := x$ otherwise, for each $x \in \Sigma^*$. Furthermore, let the mapping $\sigma : (\Sigma^*)^k \to (\Sigma^*)^k$ be defined by $\sigma((v_1, \ldots, v_k)) := (\sigma_1(v_1), \ldots, \sigma_k(v_k))$, $v_j \in \Sigma^*$, $1 \leq j \leq k$. Finally, we define $G(s) := (s'_0, s'_1, \ldots, s'_{m'})$, where the elements s'_i, $0 \leq i \leq m'$, are defined by the following procedure.*

1: $s'_i := s_i$, $0 \leq i \leq p$
2: ***for all*** j, $1 \leq j \leq k$, ***do***
3: $\quad s'_{p+|u_1 \cdots u_j|} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
4: $\quad$ ***for all*** l_j, $2 \leq l_j \leq |u_j|$, ***do***
5: $\quad\quad s_{p+|u_1 \cdots u_{j-1}|+l_j-1} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j-1}, u_j[l_j, -] \cdot \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$

6: **end for**
7: **end for**
8: $q' \leftarrow p+1$
9: $q'' \leftarrow p+|u_1 \cdots u_k|+1$
10: **while** $q' \leq m$ **do**
11: **if** $\sigma(s_{q'-1}) \neq \sigma(s_{q'})$ **then**
12: $s'_{q''} := \sigma(s_{q'})$
13: $q'' \leftarrow q''+1$
14: **end if**
15: $q' \leftarrow q'+1$
16: **end while**

As mentioned above, we explain the previous definition in an informal way and shall later consider an example. Let $s := (s_0, s_1, \ldots, s_m)$ be an arbitrary construction sequence and let p and the u_j, $1 \leq j \leq k$, be defined as in Definition 5.3. The sequence $s' := (s'_0, s'_1, \ldots, s'_{m'}) := G(s)$ is obtained from s in the following way. We keep the first p elements and then redefine the next $|u_1 \cdots u_k|$ elements in such a way that s'_p satisfies the greedy property as described by Definition 5.2. This is done in lines 1 to 9 of the algorithm. Then, in order to build the rest of s', we modify the elements s_i, $p+1 \leq i \leq m$. First, for each component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, if $|\overline{v}_{p,j}| < |v_{i,j}|$ we know that $v_{i,j} = \overline{u}_j \cdot \overline{v}_{p,j}$, where $\overline{u}_j$ is a suffix of u_j. In s', this part $\overline{u}_j$ has already been consumed by the new elements s'_i, $p+1 \leq i \leq p+|u_1 \cdots u_k|$, and is, thus, simply cut off and discarded by the mapping σ in Definition 5.3. More precisely, if a component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, of an element s_i is longer than $\overline{v}_{p,j}$, then $\sigma_j(v_{i,j}) = \overline{v}_{i,j}$. If on the other hand $|v_{i,j}| \leq |\overline{v}_{p,j}|$, then $\sigma(v_{i,j}) = v_{i,j}$. This is done in lines 10 to 16 of the algorithm.

The following proposition shows that $G(s)$ actually satisfies the conditions to be a proper construction sequence:

PROPOSITION 5.4 *For each construction sequence s of some words $w_1, \ldots, w_k$, $G(s)$ is also a construction sequence of the words $w_1, \ldots, w_k$.*

Proof Let $s := (s_0, s_1, \ldots, s_m)$ and $s' := (s'_0, s'_1, \ldots, s'_{m'}) := G(s)$, where $s_i := (v_{i,1}, \ldots, v_{i,k})$, $0 \leq i \leq m$, $s'_{i'} := (v'_{i',1}, \ldots, v'_{i',k})$, $0 \leq i' \leq m'$. We assume that s is not greedy, as otherwise $G(s) = s$ and the statement of the proposition trivially holds. Hence, let p, $1 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. In order to show that s' is a construction sequence, we need to show that the following conditions hold:

(1) $s'_0 = (w_1, w_2, \ldots, w_k)$.
(2) $s'_{m'} = (\varepsilon, \varepsilon, \ldots, \varepsilon)$.
(3) For each i, $0 \leq i \leq m'-1$, there exists a j_i, $1 \leq j_i \leq k$, and a $b_i \in \Sigma$, such that $v'_{i,j_i} = b_i \cdot v'_{i+1,j_i}$ and $v'_{i,j'} = v'_{i+1,j'}$, $j' \neq j_i$.

Condition 1 is clearly satisfied as $s'_0 = s_0 = (w_1, \ldots, w_k)$. We note that it is sufficient to prove that condition 3 is satisfied, as this implies condition 2 and, furthermore, $m = m'$. For each i, $0 \leq i \leq p+|u_1 \cdots u_k|-1$, condition 3 is clearly satisfied. To show the same for each i, $p+|u_1 \cdots u_k| \leq i \leq m'$, we consider the mapping σ from Definition 5.3. This mapping is defined in a way that, for an arbitrary $s_i = (v_{i,1}, \ldots, v_{i,k})$, $\sigma(s_i) = (\widetilde{v}_{i,1}, \ldots, \widetilde{v}_{i,k})$, with, for each j, $1 \leq j \leq k$, either $\widetilde{v}_{i,j} = v_{i,j}$, if $|v_{i,j}| \leq |\overline{v}_{p,j}|$ or $\widetilde{v}_{i,j} = \overline{v}_{p,j}$, if $|v_{i,j}| > |\overline{v}_{p,j}|$, where $\overline{v}_{p,j}$ is defined as in Definition 5.3. Consequently, for each i, $p+1 \leq i \leq m$, we have either $\sigma(s_{i-1}) = \sigma(s_i)$ or $\sigma(s_{i-1}) = (\widetilde{v}_{i,1}, \ldots, \widetilde{v}_{i,j-1}, b \cdot \widetilde{v}_{i,j}, \widetilde{v}_{i,j+1}, \ldots, \widetilde{v}_{i,k})$ and $\sigma(s_i) = (\widetilde{v}_{i,1}, \ldots, \widetilde{v}_{i,k})$, for some j, $1 \leq j \leq k$. In lines 10 to 16 of the algorithm, we

ignore the $\sigma(s_i)$ with $\sigma(s_i) = \sigma(s_{i-1})$ and only keep $\sigma(s_i)$ in the sequence s' if $\sigma(s_i) \neq \sigma(s_{i-1})$. Hence, condition 1 holds for all i, $0 \leq i \leq m-1$, and, moreover, this implies $m' = m$. ■

Now, as an example for Definition 5.3, we consider the construction sequence

$$s := ((\mathbf{abacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{bc}), (\mathbf{bacbc}, \mathbf{c}),$$
$$(\mathbf{acbc}, \mathbf{c}), (\mathbf{acbc}, \varepsilon), (\mathbf{cbc}, \varepsilon), (\mathbf{bc}, \varepsilon), (\mathbf{c}, \varepsilon), (\varepsilon, \varepsilon))$$

of the words $w_1 = \mathbf{abacbc}$ and $w_2 = \mathbf{abc}$, as given below Definition 5.1. The shuffle word that corresponds to this construction sequence is $w := \mathbf{aabbcacbc}$. We now illustrate how the construction sequence $s' := (s'_0, s'_1, \ldots, s'_m) := G(s)$ is constructed by the algorithm G. First, we note that $s_3 = (\mathbf{bacbc}, \mathbf{c})$ is the first element that does not satisfy the greedy property, since in the step from s_4 to s_5, the symbol $\mathbf{c}$ is consumed before the leftmost (and old) symbol $\mathbf{a}$ from $v_{4,1}$ is consumed. Thus, $s'_i = s_i$, $1 \leq i \leq 3$. As $w[1,3] = \mathbf{aab}$, we conclude that $u_1 := \mathbf{ba}$ and $u_2 := \varepsilon$. So the next two elements s'_4 and s'_5 consume the factor u_1 from $\mathbf{bacbc}$, hence, $s'_4 = (\mathbf{acbc}, \mathbf{c})$ and $s'_5 = (\mathbf{cbc}, \mathbf{c})$. Now let σ be defined as in Definition 5.3, thus,

$$\sigma(s_3) = (\mathbf{cbc}, \mathbf{c}), \sigma(s_4) = (\mathbf{cbc}, \mathbf{c}), \sigma(s_5) = (\mathbf{cbc}, \varepsilon),$$
$$\sigma(s_6) = (\mathbf{cbc}, \varepsilon), \sigma(s_7) = (\mathbf{bc}, \varepsilon), \sigma(s_8) = (\mathbf{c}, \varepsilon), \sigma(s_9) = (\varepsilon, \varepsilon)\,.$$

Since $\sigma(s_3) = \sigma(s_4)$ and $\sigma(s_5) = \sigma(s_6)$, we ignore $\sigma(s_4)$ and $\sigma(s_6)$; hence,

$$s'_6 = \sigma(s_5) = (\mathbf{cbc}, \varepsilon), s'_7 = \sigma(s_7) = (\mathbf{bc}, \varepsilon),$$
$$s'_8 = \sigma(s_8) = (\mathbf{c}, \varepsilon), s'_9 = \sigma(s_9) = (\varepsilon, \varepsilon)\,.$$

In conclusion,

$$s' = ((\mathbf{abacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{abc}), (\mathbf{bacbc}, \mathbf{bc}), (\mathbf{bacbc}, \mathbf{c}),$$
$$(\mathbf{acbc}, \mathbf{c}), (\mathbf{cbc}, \mathbf{c}), (\mathbf{cbc}, \varepsilon), (\mathbf{bc}, \varepsilon), (\mathbf{c}, \varepsilon), (\varepsilon, \varepsilon))\,.$$

As claimed above, the algorithm presented in Definition 5.3 can be used in order to transform a construction sequence into a greedy construction sequence. In the following proposition, we formally prove this by showing that if in a construction sequence $s := (s_0, s_1, \ldots, s_m)$ an element s_p is the first element that does not satisfy the greedy property, then in $G(s) := (s'_0, s'_1, \ldots, s'_m)$ the element s'_p satisfies the greedy property.

PROPOSITION 5.5 *Let $s := (s_0, s_1, \ldots, s_m)$ be any construction sequence that is not greedy, and let p, $0 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. Let $s' := (s'_0, s'_1, \ldots, s'_m) := G(s)$ and, if s' is not greedy, let q, $0 \leq q \leq m$, be the smallest number such that s'_q does not satisfy the greedy property. Then $p < q$.*

Proof Let $s_i := (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$. We assume that s' is not greedy and note that, by Definition 5.3, $s'_i = s_i$, $1 \leq i \leq p$. Hence, all the elements s'_i, $1 \leq i \leq p-1$, satisfy the greedy property. To prove $p < q$ it is sufficient to show that s'_p satisfies the greedy property.

Since $w[1,p] = w'[1,p]$, we can conclude that $w'[p] \notin \mathrm{alph}(w'[1,p-1])$. Furthermore, line 5 of the algorithm given in Definition 5.3 makes sure that, for every j,

$1 \leq j \leq k$,

$$s'_{p+|u_1 \cdots u_j|} = (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k}) \,,$$

where the $\overline{v}_{p,j}$ are defined as in Definition 5.3. Consequently, s'_p satisfies the greedy property, and therefore $p < q$. ■

This directly implies that, by iteratively applying the algorithm presented in Definition 5.3, any construction sequence can be turned into a greedy one. More importantly, these repeated applications of the algorithm on the construction sequence do not increase the scope coincide degree of the corresponding shuffle words:

LEMMA 5.6 *Let s be an arbitrary construction sequence that corresponds to the shuffle word w and let w' be the shuffle word corresponding to $G(s)$. Then* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

Proof Let $s := (s_0, s_1, \ldots, s_m)$, $s' := G(s) := (s'_0, s'_1, \ldots, s'_m)$ and, for each i, $0 \leq i \leq m$, $s_i := (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $s'_i := (v'_{i,1}, v'_{i,2}, \ldots, v'_{i,k})$. In this proof we shall use a special terminology: If for some i, i' with $1 \leq i < i' \leq m$, $v_{i,j} := u_{i,j} \cdot v_{i',j}$, $1 \leq j \leq k$, then we say that the $u_{i,j}$, $1 \leq j \leq k$, are consumed from the $v_{i,j}$, $1 \leq j \leq k$, by the part $s_i, s_{i+1}, \ldots, s_{i'}$.

If s is a greedy construction sequence, then $G(s) = s$ and we are done. Therefore, we assume that s is not a greedy construction sequence and let p, $0 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. For each $v_{p,j}$, $1 \leq j \leq k$, we define $v_{p,j} = u_j \cdot \overline{v}_{p,j}$, where u_j is the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1,p])$.

To prove $\mathrm{scd}(w') \leq \mathrm{scd}(w)$, we have to consider two possible cases. The first case is that $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$, i. e. $w[p]$ is the last new symbol that is consumed in s; thus $\overline{v}_{p,j} = \varepsilon$, $1 \leq j \leq k$. The second case is that this property is not satisfied, so there exists a $c \in \Sigma$, such that $w[p+1,-] = \alpha \cdot c \cdot \beta$ with $c \notin \mathrm{alph}(w[1, p+|\alpha|])$. In other words, c is the next new symbol that is consumed in s after b is consumed in the step from s_{p-1} to s_p.

We start with the latter case and note that we can write w as follows:

$$w = \alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta \,,$$

where $|\alpha_1| = p - 1$, $c \notin \mathrm{alph}(\alpha_1 \cdot b \cdot \alpha_2)$ and $\mathrm{alph}(\alpha_2) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$. Before we continue, we explain the main idea of the proof. By definition of the transformation G, we know that the shuffle word w' begins with the same prefix as w, i. e. $w' = \alpha_1 \cdot b \cdot \delta$, but the suffix δ may differ from $\alpha_2 \cdot c \cdot \beta$. In the following we show that the suffix $\alpha_2 \cdot c \cdot \beta$ from w can be gradually transformed into δ without increasing the scope coincidence degree of w.

Next, we take a closer look at w and notice that α_2 exclusively consists of symbols from the prefixes u_j, $1 \leq j \leq k$. That is due to the fact that $\mathrm{alph}(\alpha_2) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$ and, for each j, $1 \leq j \leq k$, u_j is the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$. Consequently, we can consider the prefixes u_j, $1 \leq j \leq k$, as being factorised into $u_j = \widetilde{u}_j \cdot \widehat{u}_j$ such that

$$s_{p+|\alpha_2|} = (\widehat{u}_1 \cdot \overline{v}_{p,1}, \widehat{u}_2 \cdot \overline{v}_{p,2}, \ldots, \widehat{u}_k \cdot \overline{v}_{p,k}) \,.$$

In other words, as $s_p = (\widetilde{u}_1 \cdot \widehat{u}_1 \cdot \overline{v}_{p,1}, \ldots, \widetilde{u}_k \cdot \widehat{u}_k \cdot \overline{v}_{p,k})$, exactly the prefixes $\widetilde{u}_j$ are consumed by the part $s_p, s_{p+1}, \ldots, s_{p+|\alpha_2|}$ of s, and, hence, $\alpha_2 \in \widetilde{u}_1 \sqcup\!\sqcup \widetilde{u}_2 \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \widetilde{u}_k$. Moreover, the suffix $c \cdot \beta$ exclusively consists of symbols consumed in steps from s_i to s_{i+1}, $p + |\alpha_2| \leq i \leq m - 1$. Thus, $c \cdot \beta \in \widehat{u}_1 \cdot \overline{v}_{p,1} \sqcup\!\sqcup \widehat{u}_2 \cdot \overline{v}_{p,2} \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \widehat{u}_k \cdot \overline{v}_{p,k}$.

Now let $n := |\widehat{u}_1 \cdot \widehat{u}_2 \cdots \widehat{u}_k|$. We can conclude that the n symbols from the factors $\widehat{u}_j$, $1 \leq j \leq k$, occur somewhere in $c \cdot \beta$, and, furthermore, since $c \notin \text{alph}(\widehat{u}_j)$, $1 \leq j \leq k$, we also know that all these n symbols occur in β. Thus we can write

$$\beta = \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma \,,$$

where the symbols d_j, $1 \leq j \leq n$, are exactly the symbols consumed from the $\widehat{u}_j$, $1 \leq j \leq k$, i. e., for each $i \in \{|\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_{i'}| \mid 1 \leq i' \leq n\}$ there exists a j_i, $1 \leq j_i \leq k$, such that $v_{i,j_i} = d_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$, and, furthermore, $|v_{i+1,j_i}| \geq |\overline{v}_{p,j_i}|$. This means, in particular, that $d_1 \cdot d_2 \cdots d_n \in \widehat{u}_1 \sqcup\!\sqcup \widehat{u}_2 \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \widehat{u}_k$ and $c \cdot \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \overline{v}_{p,k}$, and therefore,

$$\alpha_2 \cdot d_1 \cdot d_2 \cdots d_n \in \widetilde{u}_1 \cdot \widehat{u}_1 \sqcup\!\sqcup \widetilde{u}_2 \cdot \widehat{u}_2 \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \widetilde{u}_k \cdot \widehat{u}_k = u_1 \sqcup\!\sqcup u_2 \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup u_k \,.$$

On the other hand, by Definition 5.3, we know that s' is constructed such that the prefixes u_j, $1 \leq j \leq k$, are consumed by $s'_p, s'_{p+1}, \ldots, s'_{|u_1 \cdot u_2 \cdots u_k|}$ in a canonical way, i. e. we can write w' as

$$w' = \alpha_1 \cdot b \cdot u_1 \cdot u_2 \cdots u_k \cdot c' \cdot \beta' \,.$$

Since, for each j, $1 \leq j \leq k$, u_j is the longest prefix of $v_{p,j}$ with $\text{alph}(u_j) \subseteq \text{alph}(\alpha_1 \cdot b)$, we know that $c' \notin \text{alph}(\alpha_1 \cdot b \cdot u_1 \cdots u_k)$. In the following, we show that $c' = c$. To this end, we recall that $s_{p+|\alpha_2|} = (\widehat{u}_1 \cdot \overline{v}_{p,1}, \ldots, \widehat{u}_k \cdot \overline{v}_{p,k})$ and the symbol c is consumed in the step from $s_{p+|\alpha_2|}$ to $s_{p+|\alpha_2|+1}$. More precisely, for some j', $1 \leq j' \leq k$, $\widehat{u}_{j'} = \varepsilon$, $\overline{v}_{p,j'}[1] = c$ and $v_{p+|\alpha_2|+1,j'} = \overline{v}_{p,j'}[2,-]$. Since $|\widehat{u}_j \cdot \overline{v}_{p,j}| \geq |\overline{v}_{p,j}|$, $1 \leq j \leq k$, we can conclude that $\sigma(s_{p+|\alpha_2|}) = (\overline{v}_{p,1}, \ldots, \overline{v}_{p,k})$ and, for the same reason, $\sigma(s_i) = (\overline{v}_{p,1}, \ldots, \overline{v}_{p,k})$, for each i, $p \leq i \leq p + |\alpha_2|$. Hence, $\sigma(s_{i-1}) = \sigma(s_i)$, $p + 1 \leq i \leq p + |\alpha_2|$, and $\sigma(s_{p+|\alpha_2|}) \neq \sigma(s_{p+|\alpha|+1})$. By recalling lines 10 to 16 of Definition 5.3, we can observe that this implies $s'_{p+|u_1 \cdots u_k|} = (v_{p,1}, \ldots, v_{p,k})$ and, furthermore, $s'_{p+|u_1 \cdots u_k|+1} = \sigma(s_{p+|\alpha_2|+1}) = (v_{p,1}, \ldots, v_{p,j'-1}, v_{p,j'}[2,-], v_{p,j'+1}, \ldots, v_{p,k})$, where $v_{p,j'}[1] = c$. This directly implies $w'[p + |u_1 \cdots u_k| + 1] = c$ and thus, $c = c'$.

Next, we show that $\beta' = \beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$. We already know that $\beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \overline{v}_{p,k}$ and clearly $\beta' \in \overline{v}_{p,1} \sqcup\!\sqcup \overline{v}_{p,2} \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup \overline{v}_{p,k}$, too. Now we recall that $\beta = \beta_1 \cdot d_1 \cdot \beta_2 \cdot d_2 \cdots \beta_n \cdot d_n \cdot \gamma$ is constructed by the part $t := (s_{p+|\alpha_2|+1}, s_{p+|\alpha_2|+2}, \ldots, s_m)$ of the construction sequence s and β' is constructed by $t' := (s'_{p+|u_1 \cdots u_k|+1}, s'_{p+|u_1 \cdots u_k|+2}, \ldots, s'_m)$. By Definition 5.3, t' is the same as $(\sigma(s_{p+|\alpha_2|+1}), \sigma(s_{p+|\alpha_2|+2}), \ldots, \sigma(s_m))$ with the only difference, that duplicate elements have been removed. These duplicate elements are exactly the elements that consume the symbols d_i, $1 \leq i \leq n$, and therefore, we can conclude that t and t' construct the same shuffle word.

We consider now the scope coincidence degree of w. Obviously,

$$\text{scd}(w) = \text{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta) = \text{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_n \cdot d_n \cdot \gamma) \,.$$

Next, we recall that $d_i \in \text{alph}(\alpha_1 \cdot b \cdot \alpha_2)$, $1 \leq i \leq n$, and therefore, by applying Lemma 4.4, we can move all the symbols d_i, $1 \leq i \leq n$, to the left, directly next to symbol c, without increasing the scope coincidence degree, i. e.

$$\text{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot c \cdot \beta_1 \cdot d_1 \cdots \beta_n \cdot d_n \cdot \gamma) \geq \text{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot d_1 \cdots d_n \cdot c \cdot \beta_1 \cdots \beta_n \cdot \gamma) \,.$$

Now we recall that $\alpha_2 \cdot d_1 \cdots d_n \in u_1 \sqcup\!\sqcup \ \ldots \ \sqcup\!\sqcup u_k$, and, thus, is actually a permutation of $u_1 \cdots u_k$. Moreover, by definition, for each j, $1 \leq j \leq k$, $\text{alph}(u_j) \subseteq$

$\mathrm{alph}(\alpha_1 \cdot b)$. Consequently, by Lemma 4.2, we can substitute $u_1 \cdots u_k$ for $\alpha_2 \cdot d_1 \cdots d_n$ without changing the scope coincidence degree and, furthermore, we can substitute β' for $\beta_1 \cdot \beta_2 \cdots \beta_n \cdot \gamma$:

$$\mathrm{scd}(\alpha_1 \cdot b \cdot \alpha_2 \cdot d_1 \cdots d_n \cdot c \cdot \beta_1 \cdots \beta_n \cdot \gamma) = \mathrm{scd}(\alpha_1 \cdot b \cdot u_1 \cdots u_k \cdot c \cdot \beta') = \mathrm{scd}(w') \,.$$

Hence, $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

It remains to prove $\mathrm{scd}(w') \leq \mathrm{scd}(w)$ for the case that $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$. In this case, the situation is not as difficult as before. We can write w' as

$$w' = \alpha_1 \cdot b \cdot u_1 \cdots u_k \,.$$

Furthermore, $\mathrm{alph}(w[p+1,-]) \subseteq \mathrm{alph}(w[1,p])$ implies $s_p = (u_1, u_2, \ldots, u_k)$; thus,

$$w = \alpha_1 \cdot b \cdot \alpha_2 \,,$$

where α_2 is a permutation of $u_1 \cdots u_k$. As $\mathrm{alph}(\alpha_2) = \mathrm{alph}(u_1 \cdots u_k) \subseteq \mathrm{alph}(\alpha_1 \cdot b)$, we can apply Lemma 4.2 and conclude $\mathrm{scd}(w') = \mathrm{scd}(w)$. ■

This lemma now enables us to prove our main result, which can be stated as follows. Every construction sequence can be transformed into a greedy construction sequence in such a way that the scope coincidence degree of the corresponding shuffle words does not increase. Since this also applies to the construction sequences corresponding to shuffle words with minimum scope coincidence degree, we can conclude that there necessarily exists a greedy shuffle word with minimum scope coincidence degree. Consequently, $\mathrm{SWminSCD}_\Sigma$ can be solved by investigating only the greedy shuffle words.

THEOREM 5.7 *Let $w \in w_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, be an arbitrary shuffle word. There exists a greedy shuffle word w' such that* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

Proof Let s be an arbitrary construction sequence of w. We define $s' := G^{|\Sigma|}(s)$, where $G^k(s)$ is the k-fold application of the mapping G on s, i. e. $G^k(s) = G(G(\ldots G(s))\ldots)$. Obviously, in w there exist $|\Sigma|$ positions i, $1 \leq i \leq |w|$, such that $w[i] \notin \mathrm{alph}(w[1, i-1])$. Thus, in s there exist at most $|\Sigma|$ elements s_i, $1 \leq i \leq |w|$, that do not satisfy the greedy property. Therefore, by Proposition 5.5, we conclude that s' is a greedy construction sequence and Lemma 5.6 implies that $\mathrm{scd}(w') \leq \mathrm{scd}(w)$, where w' is the shuffle word corresponding to s'. ■

We now present an algorithm – referred to as SolveSWminSCD – that applies the above established way to construct greedy shuffle words and enumerates all possible greedy shuffle words in order to solve $\mathrm{SWminSCD}_\Sigma$.

As a central data structure in our algorithm, we use a stack that is able to store tuples of the form $(w, (v_1, v_2, \ldots, v_k))$, where $w, v_i \in \Sigma^*$, $1 \leq i \leq k$. In the following, all push or pop operations refer to this stack. Initially, the stack stores $(\varepsilon, (w_1, w_2, \ldots, w_k))$ (line 1), where $(w_1, w_2, \ldots, w_k)$ is the input of the algorithm. We shall see that throughout the whole execution of the algorithm, the stack exclusively stores elements $(w, (v_1, v_2, \ldots, v_k))$, where, for each i, $1 \leq i \leq k$, either $v_i[1] \notin \mathrm{alph}(w)$ or $v_i = \varepsilon$. For the initial element $(\varepsilon, (w_1, w_2, \ldots, w_k))$, this property is clearly satisfied. In the main part of the algorithm, we first pop an element $(w, (v_1, v_2, \ldots, v_k))$ (line 3) and then, for each i, $1 \leq i \leq k$, with $v_i \neq \varepsilon$, we carry out the following steps (lines 7 to 12). First we append $b := v_i[1]$ to the end of w, i. e. $w := w \cdot b$ and $v_i := v_i[2,-]$ (lines 8 and 9), then, for each j, $1 \leq j \leq k$, we compute the longest prefix u_j of v_j, such that $\mathrm{alph}(u_j) \in \mathrm{alph}(w \cdot v_i[1])$ (line 11). After

Algorithm *1* SolveSWminSCD

```
1: optShuffle := ε, minscd := |Σ|, push (ε, (w_1, ..., w_k))
2: while the stack is not empty do
3:   Pop element (w, (v_1, ..., v_k))
4:   if |v_1 · v_2 ··· v_k| = 0 and scd(w) < minscd then
5:     optShuffle := w
6:     minscd := scd(w)
7:   else
8:     for all i, 1 ≤ i ≤ k, with v_i ≠ ε do
9:       b := v_i[1]
10:      v_i := v_i[2, −]
11:      Let u_j, 1 ≤ j ≤ k, be the longest prefix of v_j with alph(u_j) ⊆ alph(w · b)
12:      Push (w·b·u_1·u_2 ··· u_k, (v_1[|u_1|+1, −], v_2[|u_2|+1, −], ..., v_k[|u_k|+1, −]))
13:    end for
14:  end if
15: end while
16: Output optShuffle
```

that, we append all these factors u_j, $1 \leq j \leq k$, to w, i. e. $w := w \cdot u_1 \cdot u_2 \cdots u_k$ and $v_j := v_j[|u_j| + 1, -]$. Finally, $(w, (v_1, v_2, \ldots, v_k))$ is pushed on the stack (line 12). When this is done for each i, $1 \leq i \leq k$, with $v_i \neq \varepsilon$, we pop another element and repeat these steps. Sooner or later, we necessarily pop a tuple $(w, (\varepsilon, \varepsilon, \ldots, \varepsilon))$ and according to how the algorithm constructs the new elements that are pushed on the stack, we can conclude that w is a greedy shuffle word of the words $w_1, w_2, \ldots, w_k$. Thus, we compute $\mathrm{scd}(w)$ and save both w and $\mathrm{scd}(w)$ in case that $\mathrm{scd}(w)$ is smaller than our current minimum (lines 5 and 6). The algorithm terminates as soon as the stack is completely empty.

The next proposition states that the number $\mathrm{scd}(w)$, which is required in lines 4 and 6 of the algorithm SolveSWminSCD, can be computed efficiently:

PROPOSITION 5.8 *Let $w \in \Sigma$ be arbitrarily chosen. Then the number* $\mathrm{scd}(w)$ *can be computed in time* $\mathrm{O}(|w| \times |\Sigma|)$.

Proof We illustrate a procedure that computes $\mathrm{scd}(w)$. First of all, we move over the word w from left to right, determining the scopes of the symbols in $\mathrm{alph}(w) := \{b_1, b_2, \ldots, b_m\}$, i. e. for each b_i, $1 \leq i \leq m$, we obtain $(l_i, r_i) := \mathrm{sc}_w(b_i)$. Then we initialise $|w|$ counters $c_1 := 0, c_2 := 0, \ldots, c_{|w|} := 0$, and, for each i, $1 \leq i \leq m$, j, $l_i < j < r_i$, we increment c_j if $w[j] \neq b_i$. Finally, $\mathrm{scd}(w) = \max\{c_i \mid 1 \leq i \leq n\}$. ■

It remains to establish the time complexity of algorithm SolveSWminSCD and to prove its correctness.

THEOREM 5.9 *On an arbitrary input $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$, the algorithm* SolveSWminSCD *computes its output $w \in w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$ in time $\mathrm{O}(|w_1 \cdots w_k| \times |\Sigma| \times k^{|\Sigma|})$, and there exists no word $w' \in w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$ with $\mathrm{scd}(w') < \mathrm{scd}(w)$.*

Proof We shall first prove the correctness of the algorithm SolveSWminSCD, i. e., SolveSWminSCD computes a shuffle word with minimum scope coincidence degree, and then we take a closer look at its runtime.

By definition of the algorithm SolveSWminSCD, it is obvious that the output is a greedy shuffle word of the input words $w_1, w_2, \ldots, w_k$. From Theorem 5.7, we can derive that, in order to prove that w is a shuffle word with minimum scope coincidence degree, it is sufficient to show that the algorithm SolveSWminSCD considers all possible greedy shuffle words and therefore outputs a greedy shuffle

word with minimum scope coincidence degree. To this end, let $s := (s_0, s_1, \ldots, s_m)$ be an arbitrary greedy construction sequence that corresponds to the shuffle word $w \in w_1 \sqcup\!\sqcup w_2 \sqcup\!\sqcup \ldots \sqcup\!\sqcup w_k$. We can factorise w into $w = b_1 \cdot \alpha_1 \cdot b_2 \cdot \alpha_2 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}$, where, for each i, $2 \leq i \leq |\Sigma|$, $b_i \notin \mathrm{alph}(b_1 \cdot \alpha_1 \cdot b_2 \cdot \alpha_2 \cdots b_{i-1} \cdot \alpha_{i-1})$. Let, for each i, $1 \leq i \leq |\Sigma|$, $p_i := |b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}|$. We observe, furthermore, that in the construction sequence s, for each i, $1 \leq i \leq |\Sigma|$, we can associate the element s_{p_i} with the symbol b_i at position $|b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}| + 1$ in w, as the symbol b_i is consumed in the step from s_{p_i} to s_{p_i+1}. More precisely, for each i, $1 \leq i \leq |\Sigma|$, there exists a q_i, $1 \leq q_i \leq k$, such that $s_{p_i} = (v_{p_i,1}, \ldots, v_{p_i,k})$, where $v_{p_i,q_i} = b_i \cdot v_{p_i+1,q_i}$. Moreover, since s is a greedy construction sequence, we know that for each j, $1 \leq j \leq k$, either $v_{p_i,j}[1] \notin \mathrm{alph}(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1})$ or $v_{p_i,j} = \varepsilon$. Consequently, by definition of the algorithm and since s is a greedy construction sequence, we can conclude that, for each i, $1 \leq i \leq |\Sigma| - 1$, if we pop the tuple $(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}, (v_{p_i,1}, \ldots, v_{p_i,k}))$ from the stack in line 3, then in iteration q_i of the loop in lines 7 to 12, we push the element $(b_1 \cdot \alpha_1 \cdots b_i \cdot \alpha_i, (v_{p_{i+1},1}, \ldots, v_{p_{i+1},k}))$ on the stack. Moreover, if we pop the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|-1} \cdot \alpha_{|\Sigma|-1}, (v_{p_{|\Sigma|-1},1}, \ldots, v_{p_{|\Sigma|-1},k}))$ in line 3, then the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}, (\varepsilon, \varepsilon, \ldots, \varepsilon))$ is pushed on the stack in iteration $q_{|\Sigma|}$ of the loop in lines 7 to 12. As $(\varepsilon, (v_{p_1,1}, \ldots, v_{p_1,k})) = (\varepsilon, (w_1, \ldots, w_k))$ and $(\varepsilon, (w_1, \ldots, w_k))$ is pushed on the stack in line 1, we can conclude that all the tuples $(b_1 \cdot \alpha_1 \cdots b_{i-1} \cdot \alpha_{i-1}, (v_{p_i,1}, \ldots, v_{p_i,k}))$, $1 \leq i \leq |\Sigma|$, are pushed on the stack and thus, also popped from it, at some point of the execution of the algorithm. As shown above, this implies that in particular the tuple $(b_1 \cdot \alpha_1 \cdots b_{|\Sigma|} \cdot \alpha_{|\Sigma|}, (\varepsilon, \varepsilon, \ldots, \varepsilon)) = (w, (\varepsilon, \varepsilon, \ldots, \varepsilon))$ is popped from the stack.

Since w has been arbitrarily chosen, we can conclude that each possible greedy shuffle word of the words $w_1, w_2, \ldots, w_k$ is considered by the algorithm SolveSWminSCD. Thus, SolveSWminSCD computes a shuffle word with minimum scope coincidence degree.

Next, we consider the runtime of SolveSWminSCD. First, we determine the total number of elements that are pushed on the stack during the execution of algorithm SolveSWminSCD. To this end, we note that if we pop an element $(w, (v_1, v_2, \ldots, v_k))$ from the stack in line 3, then in lines 7 to 12 we push at most k elements $(w', (v'_1, v'_2, \ldots, v'_k))$ on the stack and, furthermore, $|\,\mathrm{alph}(w')| = |\,\mathrm{alph}(w)| + 1$. Hence, we cannot push more than $k^{|\Sigma|}$ elements on the stack. We conclude the proof by estimating the time complexity caused by a single stack element $(w, (v_1, v_2, \ldots, v_k))$. The lines 8 to 13 as well as line 3 can each be executed in time $\mathrm{O}(|w \cdot v_1 \cdots v_k|)$. In lines 4 and 6, we have to know the number $\mathrm{scd}(w)$, which, by Proposition 5.8, can be computed in time $\mathrm{O}(|w| \times |\Sigma|)$. Hence, for each element that is pushed on the stack at some point of the algorithm, we require time $\mathrm{O}(|w \cdot v_1 \cdots v_k| \times |\Sigma|) = \mathrm{O}(|w_1 \cdot w_2 \cdots w_k| \times |\Sigma|)$. Since, as explained initially, at most $k^{|\Sigma|}$ elements are pushed on the stack, we can conclude that the total runtime of the algorithm SolveSWminSCD is $\mathrm{O}(|w_1 \cdots w_k| \times |\Sigma| \times k^{|\Sigma|})$. ■

In Section 3.1, it is shown how $\mathrm{SWminSCD}_\Sigma$ can be solved on scope reduced words, i. e., we first delete all the occurrences of symbols in the input words that are neither leftmost nor rightmost occurrences, we solve $\mathrm{SWminSCD}_\Sigma$ for these reduced input words and then, as described in the proof of Lemma 3.4, we reinsert the deleted symbols in an appropriate way in order to obtain a shuffle word of the original input words. Taking these considerations into account, we can prove the following result about the time complexity of $\mathrm{SWminSCD}_\Sigma$:

THEOREM 5.10 *The problem* $\mathrm{SWminSCD}_\Sigma$ *on an arbitrary input* $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$ *can be solved in time* $\mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$.

Proof We observe that we can solve the problem $\mathrm{SWminSCD}_\Sigma$ on an arbi-

trary input $w_1, w_2, \ldots, w_k$ in the following way. First, we use the algorithm SolveSWminSCD to compute a $w' \in \mathrm{sr}(w_1) \sqcup \mathrm{sr}(w_2) \sqcup \ldots \sqcup \mathrm{sr}(w_k)$ with minimum scope coincidence degree. After that, from w', we obtain a $w \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ with $\mathrm{scd}(w) = \mathrm{scd}(w')$ by inserting the symbols into w' that have been removed in order to scope reduce the words $w_1, w_2, \ldots, w_k$. By the proof of Lemma 3.4, it is obvious that both, scope reducing the input words and obtaining w from w' by inserting the removed symbols, can be done in time $\mathrm{O}(|w_1 \cdot w_2 \cdots w_k|)$. Since $|\mathrm{sr}(w_1) \cdot \mathrm{sr}(w_2) \cdots \mathrm{sr}(w_k)| = \mathrm{O}(2|\Sigma|k)$, we can conclude that, in case that the input words are scope reduced, the runtime of SolveSWminSCD is $\mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$. Hence, with the assumption that $|w_1 \cdot w_2 \cdots w_k| = \mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$, we conclude that $\mathrm{SWminSCD}_\Sigma$ can be solved in time $\mathrm{O}(|\Sigma|^2 \times k^{|\Sigma|+1})$. ■

6. Conclusion

In this paper, we have introduced and investigated the problem $\mathrm{SWminSCD}_\Sigma$, i. e., the problem of computing a shuffle word for given input words over the alphabet Σ that is optimal with respect to the scope coincidence degree. We have presented an algorithm solving $\mathrm{SWminSCD}_\Sigma$, which makes use of the fact that there necessarily exists a shuffle word with a minimum scope coincidence degree that can be constructed in a canonical way. Consequently, we obtain an upper bound for the time complexity of this problem, which is dominated by the number of input words and the alphabet size; the length of the input words, on the other hand, is not a crucial factor. Since we have assumed the alphabet to be a constant, the problem is solvable in polynomial time, but the complexity of the problem remains open for the general case, i. e., if the alphabet is considered part of the input (we denote this problem by SWminSCD). We further note that in case that SWminSCD is NP-complete, then our algorithm is of special interest as it demonstrates the fixed-parameter tractability of this problem, with respect to the parameters of the number of input words and the alphabet size.

References

[1] R.W. Conway, W.L. Maxwell, and L.W. Miller, *Theory of Scheduling*, Addison-Wesley Publishing Company, Reading, Mass. (1967).

[2] P. Flajolet, D. Gardy, and L. Thimonier, *Birthday paradox, coupon collectors, caching algorithms and self-organizing search*, Discrete Applied Mathematics 39 (1992), pp. 207–229.

[3] R. Graham, E. Lawler, J. Lenstra, and A. Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics 5 (1979), pp. 287–326.

[4] L.P. Horwitz, R.M. Karp, R.E. Miller, and S. Winograd, *Index register allocation*, Journal of the ACM 13 (1966), pp. 43–61.

[5] D. Maier, *The complexity of some problems on subsequences and supersequences*, Journal of the ACM 25 (1978), pp. 322–336.

[6] F.M.Q. Pereira, *A survey on register allocation* (2008), `http://compilers.cs.ucla.edu/fernando/publications/drafts/survey.pdf`.

[7] D. Reidenbach and M.L. Schmid, *A Polynomial Time Match Test for Large Classes of Extended Regular Expressions*, in *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, *Lecture Notes in Computer Science*, vol. 6482, 2011, pp. 241–250.