Finding Shuffle Words that Represent Optimal Scheduling of Shared Memory Access

Daniel Reidenbach and Markus L. Schmid *

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, United Kingdom
{D.Reidenbach,M.Schmid}@lboro.ac.uk

Abstract. 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

1 Introduction

A *shuffle word* of two words u and v is any word w that can be produced by inserting all symbols of u somewhere into v, in such a way that their relative order, given by u, is preserved. Thus, w comprises both u and v as a (scattered) subword, and each of its letters corresponds to exactly one letter of either u or v. Shuffle words of more than two words are constructed iteratively.

In the present paper, we wish to propose and study a question on shuffle words that is mainly motivated by the following problem on scheduling of memory accesses: Let us assume we have k processes and m values stored in memory cells, and all these processes need to access the stored values at some points during their execution. A process does not necessarily need all the m values at the same time, so a process might get along with less than m memory cells by, for example, first using a memory cell for a value x and then, as soon as x is not needed anymore, using the same cell for another, and previously unneeded, value y. As an example, we assume that process w_1 uses the values $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$ in the order $\mathtt{abacbc}$. This process only needs two memory cells: In the first cell, $\mathtt{b}$ is permanently stored, and the second cell first stores $\mathtt{a}$ until it is not required anymore and then stores value $\mathtt{c}$. This is possible, since the part of w_1 where $\mathtt{a}$ occurs and the part where $\mathtt{c}$ occurs can be completely separated

* Corresponding author.

from each other. If we now assume that the k processes cannot access the shared memory simultaneously, then the question arises how we can sequentially arrange all memory accesses such that a minimum overall number of memory cells is required. For example, if we assume that, in addition to process $w_1 = \texttt{abacbc}$, there is another process $w_2 := \texttt{abc}$, then we can of course first execute w_1 and afterwards w_2, which results in the memory access sequence `abacbcabc`. It is easy to see that this requires a memory cell for each value $\texttt{a}, \texttt{b}$ and `c`. On the other hand, we can first execute `aba` of process w_1, then process $w_2 = \texttt{abc}$, and finally the remaining part `cbc` of w_1. This results in `abaabccbc`, which allows us to use a single memory cell for both values `a` and `c` as before.

This scheduling problem can directly be formalised as a question on shuffle words. To this end, we merely have to interpret each of the k processes as a word over an alphabet of cardinality m, where m is the number of different values to be stored. Hence, our problem of finding the best way to organise the memory accesses of all processes directly translates into computing a shuffle word of the k processes that minimises the parameter determining the number of memory cells required. Unfortunately, even for $k = 2$, there is an exponential number of possible ways to schedule the memory accesses. However, we can present an algorithm solving this problem for arbitrary input words and a fixed alphabet size in polynomial time.

The above described problem is similar to the task of *register allocation* (see, e. g., [4,6]), which plays an important role in compiler optimisation. However, in register allocation, the problem is to allocate a number of m values accessed by a process to a fixed number of k registers, where $k < m$, with the possibility to temporarily move values from a register into the main memory. Since accessing the main memory is a much more expensive CPU operation, the optimisation objective is to find an allocation such that the number of memory accesses is minimised. The main differences to the problem investigated in this work are that the number of registers is fixed, 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.

Our practical motivation and the definition of the above introduced problem result from an application of a new automaton model with two input heads [7]. In our application, these two input heads need to travel over factors of the input word; to this end, they need to know the lengths of these factors. Thus, each input head movement can be interpreted as a process that needs to access lengths of factors in a certain order. Within the scope of [7], the overall number of values that need to be stored simultaneously does not only affect the memory usage of the automaton; it also has a significant impact on the runtime of its computations. Thus, our problem on shuffle words is crucial in this context. Although we consider this nontrivial problem fundamental and believe that it might occur in other practical situations as well, it is not covered by any literature on scheduling (see, e. g., [1,3]) we are aware of, and the same holds for the research on the related *common supersequence problems* (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 $\mathtt{a} \in \mathrm{alph}(w)$, $|w|_{\mathtt{a}}$ refers to the number of occurrences of $\mathtt{a}$ in w. If we wish to refer to the symbol at a certain position j, $1 \leq j \leq n$, in a word $w = \mathtt{a}_1 \cdot \mathtt{a}_2 \cdot \cdots \cdot \mathtt{a}_n$, $\mathtt{a}_i \in \Sigma$, $1 \leq i \leq n$, we use $w[j] := \mathtt{a}_j$. Furthermore, for each j, j', $1 \leq j < j' \leq |w|$, let $w[j, j'] := \mathtt{a}_j \cdot \mathtt{a}_{j+1} \cdot \cdots \cdot \mathtt{a}_{j'}$ and $w[j, -] := w[j, |w|]$. In case that $j > |w|$, we define $w[j, -] = \varepsilon$.

We now formally introduce the notion of a shuffle word. The *shuffle operation*, denoted by $\sqcup$, is a binary operation on words, defined inductively by

- $u \sqcup \varepsilon = \varepsilon \sqcup u = \{u\}$, for each $u \in \Sigma^*$,
- $\mathtt{a} \cdot u \sqcup \mathtt{b} \cdot v = \mathtt{a} \cdot (u \sqcup \mathtt{b} \cdot v) \cup \mathtt{b} \cdot (\mathtt{a} \cdot u \sqcup v)$, for all $u, v \in \Sigma^*$ and $\mathtt{a}, \mathtt{b} \in \Sigma$.

We extend the definition of the shuffle operation to the case of more than two words in the obvious way. Furthermore, for arbitrary words $w_1, w_2, \ldots, w_k \in \Sigma^*$, we call $\Gamma := w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ 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, $\mathtt{bcaabac} \in \mathtt{abc} \sqcup \mathtt{ba} \sqcup \mathtt{ca}$.

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

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

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 $w := \mathtt{acacbbdeabcedefdeff}$. 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

In our practical motivation given in the introduction, we state that we wish to sequentially arrange parallel sequences of memory accesses. These sequences shall be modelled by words and the procedure of sequentially arranging them is described by the shuffle operation. Furthermore, our goal is to construct a shuffle word such that, for any memory access in the shuffle word, the maximum number of values that already have been accessed and shall again be accessed later on is minimal. For instance, in the shuffle word `abaabccbc` of `abacbc` and `abc`, for each position i, $1 \leq i \leq 9$, there exists at most one other symbol that has an occurrence to either side of position i. On the other hand, with respect to the shuffle word `abacbcabc` we observe that at position 4 symbol `c` occurs while both symbols `a` and `b` have an occurrence to either side of position 4. This number of symbols occurring to both sides of an occurrence of another symbol is precisely the scope coincidence degree. Hence, our central problem is the problem of finding, for any given set of words, a shuffle word with a minimum scope coincidence degree.

Problem 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.

Note that in the definition of $\mathrm{SWminSCD}_\Sigma$, the alphabet Σ is constant and not part of the input; hence, for each alphabet Σ, inputs for the problem $\mathrm{SWminSCD}_\Sigma$ have to consist of words over the alphabet Σ exclusively. This shall be important for complexity considerations.

A naive approach to solving $\mathrm{SWminSCD}_\Sigma$ on input $(w_1, w_2, \ldots, w_k)$ would be to enumerate all elements in $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ in order to find one with minimum scope coincidence degree. However, the size of this search space is too large, as 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]. More precisely,

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

where $n := \sum_{i=1}^{k} |w_i|$, and $x!$ denotes the factorial of an integer x. This demonstrates that the search space of a naive algorithm can be exponentially large. Therefore, a polynomial time algorithm cannot simply search the whole shuffle $u_1 \sqcup u_2 \sqcup \ldots \sqcup u_k$, which implies that a more sophisticated strategy is required.

Before we present a successful approach to $\mathrm{SWminSCD}_\Sigma$ in the next section, we 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 $w_1 := \mathtt{aa}$ and $w_2 := \mathtt{bb}$ and observe that $w := \mathtt{aabb}$ 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 $w_3 := \mathtt{ba}$ in such a way that the resulting shuffle word has a scope coincidence degree of 0; however, $w' := \mathtt{bbbaaa} \in w_1 \sqcup w_2 \sqcup w_3$ and $\mathrm{scd}(w') = 0$.

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 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. A formalisation and proof of correctness of this approach is omitted. 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.

In the following section, we shall establish basic results about 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 this section, we take a closer look at the scope coincidence degree. We are particularly interested in how words can be transformed without increasing their scope coincidence degree. First, we consider a proposition which describes a very basic property of the scope coincidence degree that directly follows from its definition. It can roughly be stated by saying that the scope coincidence degree of a certain position i does not depend on the order of the symbols occurring to the left and to the right of i.

Proposition 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 $\mathrm{scd}_i(u) = \mathrm{scd}_i(v)$.*

Hence, for every position in a word we can permute the part to the left or to the right of this position without changing its scope coincidence degree. The scope coincidence degree of the positions in the parts that are permuted is not necessarily stable, and thus the scope coincidence degree of the whole word may change. However, if a factor of a word w satisfies a certain property, i. e., it contains no leftmost occurrence of a symbol with respect to w (it may, however, contain rightmost occurrences of symbols), then we can arbitrarily permute this factor without changing the scope coincidence degree of the whole word:

Lemma 1. *Let α, β, π, $\pi' \in \Sigma^*$, where π is a permutation of π' and $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$. Then $\mathrm{scd}(\alpha \cdot \pi \cdot \beta) = \mathrm{scd}(\alpha \cdot \pi' \cdot \beta)$.*

The next two lemmas show that if certain conditions hold, then we can move one or several symbols in a word to the left without increasing the scope coincidence degree. The first result of that kind is related to the situation where only one symbol is moved, and the second lemma describes the case where several symbols are moved and therefore makes use of the first lemma.

We can informally summarise the first lemma in the following way. We assume that at position i in a word w a certain symbol $\mathtt{b}$ occurs and, furthermore, this is not the leftmost occurrence of $\mathtt{b}$. Then we can move this symbol to the left without increasing the scope coincidence degree of w as long as it is not moved to the left of the leftmost occurrence of a $\mathtt{b}$ in w. This seems plausible, as such an operation shortens the scope of symbol $\mathtt{b}$ or leaves it unchanged. However, we might move this certain $\mathtt{b}$ into a region of the word where many scopes coincide; thus, it is possible that the scope coincidence degree of the new position of $\mathtt{b}$ increases compared to its old position. We can show that this increase of the scope coincidence degree of that certain position does not affect the scope coincidence degree of the whole word:

Lemma 2. *For all $\alpha, \beta, \gamma \in \Sigma^*$ and for each $\mathtt{b} \in \Sigma$ with $\mathtt{b} \in \mathrm{alph}(\alpha)$,*

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

Obviously, if for some word w the condition of Lemma 2 is satisfied not only for one symbol $\mathtt{b}$ but for several symbols $\mathtt{d}_1, \mathtt{d}_2, \ldots, \mathtt{d}_n$, then we can separately move each of these $\mathtt{d}_i$, $1 \leq i \leq n$, to the left and conclude that the scope coincidence degree of the resulting word does not increase compared to w. This observation is described by the following lemma.

Lemma 3. *Let α, γ, $\beta_i \in \Sigma^*$, $0 \leq i \leq n$, $n \in \mathbb{N}$, and let $\mathtt{d}_i \in \Sigma$, $1 \leq i \leq n$, such that $\mathtt{d}_i \in \mathrm{alph}(\alpha)$, $1 \leq i \leq n$. Then*

$$\mathrm{scd}(\alpha \cdot \mathtt{d}_1 \cdot \mathtt{d}_2 \cdot \cdots \cdot \mathtt{d}_n \cdot \beta_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta_1 \cdot \mathtt{d}_1 \cdot \beta_2 \cdot \mathtt{d}_2 \cdot \cdots \cdot \beta_n \cdot \mathtt{d}_n \cdot \gamma)\,.$$

Concerning the previous lemma, we observe that we can as well position the symbols $\mathtt{d}_i$, $1 \leq i \leq n$, in any other order than $d_1 \cdot d_2 \cdot \cdots \cdot d_n$ and would still obtain a word with a scope coincidence degree that has not increased. Furthermore, with Lemma 1, we can conclude that the scope coincidence degree is exactly the same, no matter in which order the symbols $\mathtt{d}_i$, $1 \leq i \leq n$, occur between α and β_1.

5 Solving the Problem SWminSCD$_\Sigma$

In this section, we present an efficient way to solve SWminSCD$_\Sigma$. Our approach is established by identifying a certain set of well-formed shuffle words which contains at least one shuffle word with minimum scope coincidence degree and, moreover, is considerably smaller than the set of all shuffle words. To this end, we shall first introduce a general concept for constructing shuffle words, and then a simpler and standardised way of constructing shuffle words is defined.

By applying the lemmas given in the previous section, we are able to show that there exists a shuffle word with minimum scope coincidence degree that can be constructed in this simple way.

Let $w_1, w_2, \ldots, w_k \in \Sigma^*$ be arbitrary words. We consider these words as stack-like data structures where the leftmost symbol is the topmost stack element. Now we can empty these stacks by successively applying the pop operation and every time we pop a symbol from a stack, we append this symbol to the end of an initially empty word w. Thus, as soon as all stacks are empty, we obtain a word built up of symbols from the stacks, and this word is certainly a shuffle word of $w_1, w_2, \ldots, w_k$.

It seems to be 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 $\mathtt{b}$ 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 $\mathtt{b}$ 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 to construct 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 $v_i = \mathtt{b} \cdot v_i'$ for some i, $1 \leq i \leq k$, and for some $\mathtt{b} \in \Sigma$. 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 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 $s := (s_0, s_1, \ldots, s_m)$, $m := |w_1 \cdot \cdots \cdot w_k|$ 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 $\mathtt{b}_i \in \Sigma$ such that $v_{i,j_i} = \mathtt{b}_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$.*

The shuffle word $w = \mathtt{b}_0 \cdot \mathtt{b}_1 \cdot \cdots \cdot \mathtt{b}_{m-1}$ 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 $\mathtt{b}_{i+1}$ 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 $w_1 := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$ and $w_2 := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}$:

$$\begin{aligned} s := ((&\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}), (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}), (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{b} \cdot \mathtt{c}),\\ &(\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{c}), (\mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{c}), (\mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \varepsilon), (\mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \varepsilon),\\ &(\mathtt{b} \cdot \mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon)) \,. \end{aligned}$$

The shuffle word corresponding to s is $w := \mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$, 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 $\mathtt{b}$ 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 $\mathtt{b}$ 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 $\mathtt{b}$ 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. Obviously, there are still several possible greedy construction sequences for some input words w_i, $1 \leq i \leq k$, as whenever a new symbol is consumed, we have a choice of k possible suffixes to consume this symbol from. We formally define this greedy property of construction sequences.

Definition 2. *Let $w \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, 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, $1 \leq j \leq k$, $s_{i+|u_1 \cdot \ldots \cdot u_j|} = (\overline{v}_{i,1}, \ldots, \overline{v}_{i,j}, v_{i,j+1}, \ldots, v_{i,k})$, 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 $w_1 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$ and $w_2 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}$. This time, we present a greedy construction sequence $s := (s_0, s_1, \ldots, s_9)$ for w_1 and w_2:

$$\begin{aligned} s := ((&\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}), (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}),\\ &(\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{b} \cdot \mathtt{c}), (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{c}), (\mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{c}),\\ &(\mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \mathtt{c}), (\mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c},\, \varepsilon), (\mathtt{b} \cdot \mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon)) \,. \end{aligned}$$

Obviously, the shuffle word $w := \mathtt{a} \cdot \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$ 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 $s_3 = (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{c})$ and note that, in terms of Definition 2, we have $u_1 := \mathtt{b} \cdot \mathtt{a}$, $\overline{v}_{3,1} := \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$, $u_2 := \varepsilon$ and $\overline{v}_{3,2} := \mathtt{c}$. 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 $|u_1| = |u_1 \cdot u_2| = 2$, $\overline{v}_{3,1} = \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$, $v_{3,2} = \overline{v}_{3,2} = \mathtt{c}$ and $s_5 = (\mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{c})$, this clearly holds.

In the following, we show how we can transform an arbitrary construction sequence $s := (s_0, s_1, \ldots, s_m)$ into a greedy one. Informally speaking, this is done by determining the first element s_i that does not satisfy the greedy property and then we simply redefine all the elements s_j, $i+1 \leq j \leq m$, in a way such that s_i satisfies the greedy property. If we apply this method iteratively, we can obtain a greedy construction sequence. Next, we introduce the formal definition of that transformation and explain it in more detail later on.

Definition 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: $\quad s'_{p+|u_1 \cdot \cdots \cdot u_j|} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
4: $\quad$ ***for all*** l_j, $2 \leq l_j \leq |u_j|$, ***do***
5: $\quad\quad s_{p+|u_1 \cdot \cdots \cdot u_{j-1}|+l_j-1} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j-1}, u_j[l_j, -] \cdot \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
6: $\quad$ ***end for***
7: ***end for***
8: $q' \leftarrow p+1$
9: $q'' \leftarrow p + |u_1 \cdot \cdots \cdot u_k| + 1$
10: ***while*** $q' \leq m$ ***do***
11: $\quad$ ***if*** $\sigma(s_{q'-1}) \neq \sigma(s_{q'})$ ***then***
12: $\quad\quad s'_{q''} := \sigma(s_{q'})$
13: $\quad\quad q'' \leftarrow q''+1$
14: $\quad$ ***end if***
15: $\quad q' \leftarrow q'+1$
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 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 $|u_1 \cdot \cdots \cdot u_k|$ elements in such a way that s'_p satisfies the greedy property as described by Definition 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, $p+1 \leq i \leq p+|u_1 \cdot \cdots \cdot u_k|$, and is, thus, simply cut off and discarded by the mapping σ in Definition 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 18 of the algorithm.

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

Proposition 2. *For each construction sequence s of some words $w_1, \ldots, w_k$, $G(s)$ is also a construction sequence of the words $w_1, \ldots, w_k$.*

Now, as an example for Definition 3, we consider the construction sequence

$$\begin{aligned} s := ((&\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),\\ &(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}),\\ &(\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon)) \end{aligned}$$

of the words $w_1 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$ and $w_2 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}$, as given below Definition 1. The shuffle word that corresponds to this construction sequence is $w := \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$. 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 $s_3 = (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c})$ is the first element that does not satisfy the greedy property, since in the step from s_4 to s_5, the symbol $\mathtt{c}$ is consumed before the leftmost (and old) symbol $\mathtt{a}$ from $v_{4,1}$ is consumed. Thus, $s'_i = s_i$, $1 \leq i \leq 3$. As $w[1,3] = \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}$, we conclude that $u_1 := \mathtt{b}\cdot\mathtt{a}$ and $u_2 := \varepsilon$. So the next to elements s'_4 and s'_5 consume the factor u_1 from $\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$, hence, $s'_4 = (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c})$ and $s'_5 = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c})$. Now let σ be defined as in Definition 3, thus,

$$\begin{aligned} &\sigma(s_3) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), \sigma(s_4) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), \sigma(s_5) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon),\\ &\sigma(s_6) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), \sigma(s_7) = (\mathtt{b}\cdot\mathtt{c},\, \varepsilon), \sigma(s_8) = (\mathtt{c},\, \varepsilon), \sigma(s_9) = (\varepsilon,\, \varepsilon)\,. \end{aligned}$$

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

$$\begin{aligned} &s'_6 = \sigma(s_5) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), s'_7 = \sigma(s_7) = (\mathtt{b}\cdot\mathtt{c},\, \varepsilon),\\ &s'_8 = \sigma(s_8) = (\mathtt{c},\, \varepsilon), s'_9 = \sigma(s_9) = (\varepsilon,\, \varepsilon)\,. \end{aligned}$$

In conclusion

$$\begin{aligned} s' = ((\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),\\ (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}),\\ (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon))\,. \end{aligned}$$

Next, we show that if in a construction sequence $s := (s_0, s_1, \ldots, s_m)$ the 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. This follows from Definition 3 and has already been informally explained.

Proposition 3. *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 $p < q$.*

More importantly, we can also state that the scope coincidence degree of the shuffle word corresponding to $G(s)$ does not increase compared to the shuffle word that corresponds to s. To this end, we shall employ the lemmas introduced in Section 4.

Lemma 4. *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* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

The previous lemma is very important, as it implies our next result, which can be stated as follows. By iteratively applying the algorithm G, we can transform each construction sequence, including the ones corresponding to shuffle words with minimum scope coincidence degree, into a greedy construction sequence that corresponds to a shuffle word with a scope coincidence degree that is the same or even lower:

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

This particularly implies that there exists a greedy shuffle word with minimum scope coincidence degree. Hence, $\mathrm{SWminSCD}_\Sigma$ reduces to the problem of finding a greedy shuffle word with minimum scope coincidence degree.

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

Next, we state that this algorithm works correctly and establish its time complexity.

Theorem 2. *On an arbitrary input $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$, the algorithm* SolveSWminSCD *computes its output $w \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ in time* $\mathrm{O}(|w_1 \cdot \cdots \cdot w_k| \times k^{|\Sigma|})$ *and there exists no $w' \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ with* $\mathrm{scd}(w') < \mathrm{scd}(w)$.

Algorithm 1 SolveSWminSCD

optShuffle := ε, minscd := $|\Sigma|$, push $(\varepsilon, (w_1, \ldots, w_k))$
while the stack is not empty **do**
  Pop element $(w, (v_1, \ldots, v_k))$
  if $|v_1 \cdot v_2 \cdot \cdots \cdot v_k| = 0$ and $\mathrm{scd}(w) <$ minscd **then**
    optShuffle := w
    minscd := $\mathrm{scd}(w)$
  else
    for all i, $1 \leq i \leq k$, with $v_i \neq \varepsilon$ **do**
      $\mathtt{b} := v_i[1]$
      $v_i := v_i[2, -]$
      Let u_j, $1 \leq j \leq k$, be the longest prefix of v_j with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w \cdot \mathtt{b})$
      Push $(w \cdot \mathtt{b} \cdot u_1 \cdot u_2 \cdot \cdots \cdot u_k, (v_1[|u_1|+1, -], v_2[|u_2|+1, -], \ldots, v_k[|u_k|+1, -]))$
    end for
  end if
end while
Output optShuffle

By applying the observation from Section 3 – i. e., we can solve SWminSCD by first deleting all the occurrences of symbols in the input words that are neither leftmost nor rightmost occurrences and then solving SWminSCD for the reduced input words – we can prove the following result about the time complexity of SWminSCD:

Theorem 3. *The problem* SWminSCD *on an arbitrary input* $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$ *can be solved in time* $\mathrm{O}(|\Sigma| \times k^{|\Sigma|+1})$.

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*, 39:207–229, 1992.
3. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
4. L. P. Horwitz, R. M. Karp, R. E. Miller, and S. Winograd. Index register allocation. *Journal of the ACM*, 13:43–61, 1966.
5. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.
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*, volume 6482 of *Lecture Notes in Computer Science*, pages 241–250. Springer, 2011.