Scattered Context Grammars with One Non-Context-Free Production and Six Nonterminals are Computationally Complete

Martin Havel[1 [0009−0008−3000−6514]], Alexander Meduna[1 [0000−0002−2341−0606]], and Zbyněk Křivka[1 [0000−0001−8309−0280]]

Brno University of Technology, Faculty of Information Technology, Božetěchova 1/2, 612 00 Brno, Czech republic {ihavelm, meduna, krivka}@fit.vut.cz

Abstract. The present paper explains how to reduce the size of scattered context grammars with respect to the number of both non-context-free productions and nonterminals. It proves that every recursively enumerable language is generated by a six-nonterminal scattered context grammar with a single non-context-free production. Open problems are proposed.

Keywords: Scattered context grammars · Size reduction · The number of non-context-free productions · The number of nonterminal symbols · Computational completeness · Descriptional complexity.

1 Introduction

Over its history, formal language theory has always struggled to reduce its grammars with respect to various numbers of components (see Sections 1.2 and 1.3 in Chapter 4 in [13]). Perhaps most intensively, it has studied how to reduce the number of nonterminals and productions without affecting the generative power of the grammars in question. The present paper continues with this study in terms of scattered context grammars (see [11]).

Recall that restricting the amount of nonterminals was always of great interest without disrupting its computational power or how computational power is changed. Restricting the number of nonterminals was already researched for phase-structured grammar (see [3]), automata, (see [4], context-free grammars (see [2], [7]), EOL (see [8], [14]) and countless others. For scattered context grammars, it was already demonstrated that scattered context grammars with a single non-context-free production are as powerful as their unlimited versions because they characterize the family of recursively enumerable languages (see [6]). This paper improves this result by reducing the number of non-context-free productions and, simultaneously, the number of nonterminals. Specifically, it demonstrates that six-nonterminal scattered context grammars with one non-context-free production characterize this family.

Of course, concerning the number of non-context-free productions, this statement represents the best possible result because scattered context grammars

without any non-context-free productions are as powerful as context-free grammars, so they are less powerful than their unlimited versions. With respect to the number of nonterminals, however, an improvement of this statement represents a challenging open problem. Indeed, as one-nonterminal scattered context grammars are weaker than their unlimited versions (see Theorem 6.1 in [11]), we ask whether there exists i in {2, 3, 4, 5, 6, ...} such that i-nonterminal scattered context grammars with one non-context-free production characterize the family of recursively enumerable languages.

2 Definitions

This paper assumes that the reader is familiar with the language theory (see [9]), including scattered context grammars (see [11], [12]).

For a set, Q, card(Q) denotes the cardinality of Q. For an alphabet, V, V^* represents the free monoid generated by V under the operation of concatenation. The unit of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation. For $w \in V^*$, $|w|$ and rev(w) denote the length of w and the reversal of w, respectively. Furthermore, suffix(w) denotes the set of all suffixes of w, and prefix(w) denotes the set of all prefixes of w. For $w \in V^*$ and $T \subseteq V$, occur(w, T) denotes the number of occurrences of symbols from T in w, and erase(w, T) denotes the string obtained by removing all occurrences of symbols from T in w. For instance, occur($abdabc, \{a, d\}$) = 3 and erase($abdabc, \{a, d\}$) = bbc. If $T = \{a\}$, where $a \in V$, we simplify occur($w, \{a\}$) and erase($w, \{a\}$) to occur(w, a) and erase(w, a), respectively.

A *scattered context grammar* is a quadruple, $G = (N, T, P, S)$, where N and T are alphabets such that $N \cap T = \emptyset$. Symbols in N are referred to as nonterminals while symbols in T are terminals. N contains S—the start symbol of G. P is a finite non-empty set of productions or rules such that every $p \in P$ has the form

$$(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n),$$

where $n \geq 1$, and for all $i = 1, 2, \ldots, n$, $A_i \in N$ and $x_i \in (N \cup T)^*$. If each x_i satisfies $|x_i| \leq 1$, $i = 1, 2, \ldots, n$, then $(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n)$ is said to be simple. If $n = 1$, then $(A_1) \to (x_1)$ is referred to as a context-free production; for brevity, we hereafter write $A_1 \to x_1$ instead of $(A_1) \to (x_1)$. If for some $n \geq 1$, $(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n) \in P, v = u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1}$, and $w = u_1 x_1 u_2 x_2 \ldots u_n x_n u_{n+1}$ with $u_i \in (N \cup T)^*$ for all $i = 1, 2, \ldots, n$, then v directly derives w in G, symbolically written as $v \Rightarrow w$ [$(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n)$] or, simply, $v \Rightarrow w$ in G. In the standard manner, extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, define $\Rightarrow^+$ and $\Rightarrow^*$. The language of G, $L(G)$, is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$. A derivation of the form $S \Rightarrow^* w$ with $w \in T^*$ is called a successful derivation.

A *queue grammar* (see [5]) is a sextuple, $Q = (V, T, W, F, s, R)$, where V and W are alphabets satisfying $V \cap W = \emptyset$, $T \subseteq V$, $F \subseteq W$, $s \in (V - T)(W - F)$, and $R \subseteq (V \times (W - F)) \times (V^* \times W)$ is a finite relation such that for every

$a \in V$, there exists an element $(a, b, z, c) \in R$. If $u, v \in V^*W$ such that $u = arb$; $v = rzc$; $a \in V$; $r, z \in V^*$; $b, c \in W$; and $(a, b, z, c) \in R$, then $u \Rightarrow v$ $[(a, b, z, c)]$ in Q or, simply, $u \Rightarrow v$. In the standard manner, extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, define $\Rightarrow^+$ and $\Rightarrow^*$. The language of Q, $L(Q)$, is defined as $L(Q) = \{w \in T^* : s \Rightarrow^* wf,\ f \in F\}$.

As a slight modification of the notion of a queue grammar, there is the notion of a left-extended queue grammar such that it is a queue grammar that during every derivation step shifts the rewritten symbol in front of the beginning of its sentential form (see # below); in this way, it records the derivation history, which plays a crucial role in the proof of correctness of Algorithm 1 in the next section. Formally, a *left-extended queue grammar* is a sextuple, $Q = (V, T, W, F, s, P)$, where V, T, W, F, and s are defined as in a queue grammar. $P \subseteq (V \times (W - F)) \times (V^* \times W)$ is a finite relation (as opposed to an ordinary queue grammar, this definition does not require that for every $a \in V$, there exists an element $(a, b, z, c) \in R$). Furthermore, assume that $\# \notin V \cup W$. If $u, v \in V^*\{\#\}V^*W$ so that $u = w\#arb; v = wa\#rzc; a \in V; r, z, w \in V^*; b, c \in W;$ and $(a, b, z, c) \in R$, then $u \Rightarrow v$ $[(a, b, z, c)]$ in Q or, simply, $u \Rightarrow v$. In the standard manner, extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, define $\Rightarrow^+$ and $\Rightarrow^*$. The language of $Q, L(Q)$, is defined as $L(Q) = \{v \in T^* : \#s \Rightarrow^* w\#vf$ for some $w \in V^*$ and $f \in F\}$.

Next, we recall two already known properties of left-extended queue grammars (introduced in [10] and [6]), subsequently used in the proof of Algorithm 1.

Lemma 1. *For every recursively enumerable language, L, there exists a left-extended queue grammar, Q, satisfying $L(Q) = L$.*

Definition 1. *Let $Q = (V, T, W, F, s, R)$ be a left-extended queue grammar. Q is said to be in* normal form 2 *if every $(a, b, x, c) \in R$ satisfies $a \in V - T, b \in W - F$, and $x \in ((V - T)^* \cup T^*)$ and in addition, for every $q \in W - F$, there is no more than one $a \in V - T$ such that $(a, q, x, p) \in R$, where $x \in (V - T)^* \cup T^*$ and $p \in W$.*

Lemma 2. *For every left-extended queue grammar H, there exists an equivalent left-extended queue grammar Q in normal form 2.*

In general, two grammars are *equivalent* if both generate the same language.

3 Results

This section demonstrates that every recursively enumerable language, L, is generated by a scattered context grammar, $G = (N, T, P, S)$, such that $L = L(G)$, (i) P contains a single non-context-free production of the form $(1, 2, 0, 3, 0, 2, 1) \rightarrow (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$, and (ii) N consists of six nonterminals. This demonstration is based on left-extended queue grammars, which are computationally complete.

First, we define certain *codes* and their counterparts, referred to as *anti-codes*. We establish several general results about these codes and anti-codes, which are

used later in this section. Then, we give the major algorithm that turns any left-extended queue grammar to an equivalent scattered context grammar having firstly a single non-context-free production and secondly seven nonterminals. We establish several results about this algorithm, based on which, we verify that the algorithm is correct. Finally, we improve the result sketched about so we demonstrate that every recursively enumerable language L is generated by a scattered context grammar having no more than (i) one non-context-free production and (ii) six nonterminals.

Next, we explain how to turn any left-extended queue grammar Q in second normal form to an equivalent scattered context grammar G with a single non-context-free production and only seven nonterminals. The basic idea consists of the simulation of a derivation in Q by context-free productions in an utterly arbitrary way, after which the only scattered context production is repeatedly used to verify that the derivation is correct. More precisely, G generates every $x \in L(G)$ by performing (I) and (II).

First, we introduce a coding generated by context-free productions of G that handles any context dependency. It derives uxv from its start symbol, where u and v are codes over $\{0,1,2,3\}$, where 0 through 3 are special encoding nonterminals. The first phase ends when precisely two 2s are generated in the current sentential form. Then, in (II), by using its only non-context-free production π of the form

$$(1,2,0,3,0,2,1) \to (2,\varepsilon,\varepsilon,\varepsilon,\varepsilon,\varepsilon,2),$$

it removes u and v. G completes its removal successfully and, thereby, generates x iff x belongs to $L(Q)$. Thus, $L(G) = L(Q)$.

Definition 2. *For $k \geq 1$, let $X_k, Y_k \subseteq \{0,1,3\}^*$ such that*

$$X_k = \{103\}^+\{1\}\{10\}^+ \cap \{x\colon x \in \{0,1,3\}^*, \mathrm{occur}(x,1) = k\}$$

and

$$Y_k = \{301\}^+\{0301\}\{01\}^+ \cap \{x\colon x \in \{0,1,3\}^*, \mathrm{occur}(x,1) = k\}$$

Let A, B be a finite (non-empty) sets. Let us define an injection $\iota\colon A \times B \to X_n$ where n is a positive integer great enough to allow us to introduce ι as injection (a proof that such a constant necessarily exists is simple and left to the reader). Extend the domain of ι to $(A \times B)^*$ in the standard way.

Let $\beta\colon \{0,1\}^* \to \{0,1,3\}^*$ be a homomorphism such that $\beta(0) = 30$ and $\beta(1) = 1$.

Let $\kappa\colon A \times B \to Y_n$ be an injection with $\kappa(a,b) = z10301w$ where $z = \beta(\mathrm{rev}(v))$ and $w = \mathrm{rev}(\mathrm{erase}(u,\{3\}))$ where $u11v = \iota(a,b)$. Extend the domain of κ to $(A \times B)^*$ in the standard way too.

We refer to $x, y \in \{0,1,3\}^*$ as a code string *and its* anticode string *iff $x = \iota(a,b)$ and $y = \kappa(a,b)$ for some $a \in A$ and $b \in B$.*

Example 1. For instance, for $A = \{C, D\}$ and $B = \{s\}$ ($|A| = 2$, $|B| = 1$) and $n = 4$, we can define ι as $\iota(C,s) = 10311010$ and $\iota(D,s) = 103103110$. Then, $\kappa(C,s) = 301301030101$ and $\kappa(D,s) = 30103010101$.

The following lemma demonstrates that in scattered context grammar with two special productions, we can safely annihilate some code string and its corresponding anticode string.

Definition 3. *We say that a code string and its corresponding anticode string are* annihilated *if* $xuwvz \Rightarrow^* \omega(x)w\omega'(z)$ *where* $u = \iota(a,b)$ *and* $v = \kappa(a,b)$*, for some* $a \in A$ *and* $b \in B$*,* $w \in (N \cup T)^*$*, and* ω *and* ω' *replaces the last 1 by 2 and the first 1 by 2, respectively.*

Lemma 3. *Let* $G = (N, T, P, S)$ *be a scattered context grammar with* $N = N' \cup D$*,* $D = \{0, 1, 2, 3\}$*, and the only two productions in* P *that rewrite nonterminals from* D *are*

$$(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$$

$$2 \to \varepsilon$$

For any sentential form $xuwvz$ *with* $u = \iota(a,b)$ *and* $v = \kappa(a,b)$ *for some* $a \in A$ *and* $b \in B$ *(will be specified later), where* $x, u, v, z \in D^*$*,* u *and* v *can be annihilated.*

Claim. Any substrings from V^* in the sentential form do not matter for the annihilation (i.e. cannot block the annihilation of some code and its anticode).

More formally, if $u \in D^*$ can be annihilated by productions from P in G, then u scattered with x, where $x \in (V - D)^*$, can be rewritten to x in G as the rest of u.

Definition 4. *Based on* $\iota\colon A \times B \to X_k$*, define the substitution* ν *from* A *to* 2^{X_k} *by*

$$\nu(a) = \{\iota(a,b)\colon b \in B\}$$

for every $a \in A$*; then, extend its domain to* A^* *in the standard way.*

Next, using $\kappa\colon A \times B \to Y_k$*, define the substitution* μ *from* B *to* 2^{Y_n} *by*

$$\mu(b) = \{\kappa(a,b)\colon a \in A\}$$

for every $b \in B$*; then, extend its domain to* B^* *in the standard way too.*

Example 2. Notice that every $x \in \nu(a)$ is a string of the form

$$103103 \cdots 10311010 \cdots 10$$

in which there are n occurrences of 1 and precisely one occurrence of substring 11. Notice that every $y \in \mu(q)$ is a string of the form

$$301301 \cdots 30103010101 \cdots 01$$

in which there are n occurrences of 1 and precisely one occurrence of 030.

Before giving a proof of the correctness, we describe Algorithm 1 informally. Recall that the input Q is left-extended, so it records the prefix of all symbols rewritten at the beginning of the queue during the generation of $x \in L(Q)$.

In addition, apart from recording this prefix, Q also records all the states through which Q passes through. That is, it always records two identical states to the left. Furthermore, based on Q generate code two times denoted as δ defined later in this section and generates mirroring anticode later in the derivation to ensure that all terminals are generated.

When simulating this generation, G takes an advantage of this recorded rewriting history to verify that the entire simulation has been performed properly. To give a more precise insight into this simulation and subsequent verification, assume that Q generates $x \in L(Q)$ by using a sequence of productions $r_1, r_2, \ldots, r_m$. G simulates this generation as follows.

(I) During the first phase, G makes $S \Rightarrow^* q_0u_1q_1q_1u_2q_2q_2 \cdots u_{m-1}q_{m-1}q_{m-1}u_mq_mq_mxq_mq_mv_mq_{m-1}q_{m-1}v_{m-1} \cdots q_2q_2v_2q_1q_1v_1q_0$ where each u_j and v_j encodes a production applied during the jth derivation step in Q, and all of them have the same number of 1s. More specifically, each $u_i = {}_pu_i\,{}_su_i$ where ${}_pu_i$ is a prefix of u_i over $\{0,1,3\}$ and ${}_su_i$ is a suffix of u_i over $\{0,1\}$. Let us call the beginning of ${}_su_i$ as u_i-*break*. Similarly, each $v_i = {}_pv_i\,{}_sv_i$ where ${}_pv_i$ is a prefix of v_i over $\{0,1\}$ and ${}_sv_i$ is a suffix of v_i over $\{0,1,3\}$. Let us call the end of ${}_pv_i$ as v_i-*break*. The position of u_i-break and v_i-break encodes a production in Q so this encoding satisfies the following property:

$$\text{if } (a,q,x,p) \text{ and } (b,o,y,r) \text{ are two productions}$$
$$\text{encoded by the same break position, then } p = r \text{ and } x = y.$$

The states are encoded analogously.

(II) During the second phase, G has to make sure that the simulation of the generation of x in Q is performed correctly. To do so, G has to verify that for $j = m, \ldots, 2, 1$, both u_j and v_j encode the same production r_j in Q. G makes this verification solely by using π so it eliminates all u_m through u_1 and, simultaneously, v_m through v_1 in the insight-out way.
To explain the elimination process more precisely, consider this portion

$$u_1u_2 \cdots {}_pu_i \cdot {}_su_i \;\; x \;\; {}_pv_i \cdot {}_sv_i \cdots v_2v_1$$

where $\cdot$ points out the position of u_i-break and v_i-break, respectively. G can eliminate the codes u_i and v_i if and only if u_i-break and v_i-break are simultaneously rewritten by π; thereby, it guaranties that ${}_su_i$ and ${}_pv_i$ have the same number of 1s and 0s, therefore, u_i and v_i encode the same production r_i. Analogously, G verifies that during any two consecutive derivation steps, in the first step, Q enters the same state from which it performs the other step.

Definition 5. *Define function δ over $\{y\colon y \in \nu(a), a \in A\}^*$ as follows*

1. $\delta(\varepsilon) = \varepsilon$

2. *let* $z = x_1x_2x_3\cdots x_n$ *for some* $n \geq 1$, $x_i \in \nu(a_i)$ *for some* $a_i \in A$; *then,* $\delta(z) = x_1^2x_2^2x_3^2\cdots x_n^2$.

Let $D = \{0, 1, 2, 3\}$.

Algorithm 1 *Input: A left-extended queue grammar* $Q = (V, T, W, F, s, R)$ *in normal form 2.*
Output: A scattered context grammar $G = (N, T, P, S)$ *such that* $L(G) = L(Q)$.
Method:

1. *Set* $N = \{S, 4, 5\} \cup D$.
2. *Consider the definition of* ι *(see Definition 2) with* $A = V - T$ *and* $B = W - F$. *Based on* A, B, *and* ι, *consider specific definitions of* κ, μ, ν, *and* δ *(see Definitions 2, 4, and 5).*
3. *Initialize* M *with* $\emptyset$. *Perform (3i) through (3v), given next, to construct* M.
   (i) *if* $s = a_0q_0$, *where* $a_0 \in V-T$ *and* $q_0 \in W-F$, *then add* $S \to 1\delta(u)4w1$ *to* M, *for all* $u \in \nu(a_0)$ *and* $w \in \mu(q_0)$;
   (ii) *if* $(a, q, y, p) \in R$, *where* $a \in V - T$, $p, q \in W - F$, *and* $y \in (V - T)^*$, *then add* $4 \to \delta(u)4w$ *to* M, *for all* $u \in \nu(y)$ *and* $w \in \mu(p)\mu(q)$;
   (iii) *add* $4 \to 205$ *to* M;
   (iv) *if* $(a, q, y, p) \in R$, *where* $a \in V - T$, $p, q \in W - F$, $y \in T^*$, *then add* $5 \to y5w$ *to* M, *for all* $w \in \mu(p)\mu(q)$;
   (v) *if* $(a, q, y, p) \in R$, *where* $a \in V - T$, $q \in W - F$, $y \in T^*$, *and* $p \in F$, *then add* $5 \to y302w$ *to* M, *for all* $w \in \mu(q)$.
4. *Set* $O = \{(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2),\ 2 \to \varepsilon\}$.
5. *Set* $P = M \cup O$.

Hereafter, for brevity, we refer to $(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$ as π.

Basic Idea. Algorithm 1 extends the Algorithm proposed in [6], but for clarity, we explained it completely. Next, we sketch the reason why $L(G) = L(Q)$.

What we need to demonstrate is that for any $y \in T^*$,

$$S \Rightarrow^* y \text{ in } G \text{ if and only if } \#s \Rightarrow^* w\#yf \text{ in } Q$$

with $s = a_0q_0$, $w \in (V - T)^*$, and $f \in F$.

To rephrase this equivalence more precisely, we need to show that $S \Rightarrow^* y$ in G if and only if for some $m \geq 1$, Q makes $\#a_0q_0 \Rightarrow^* a_0 \cdots a_m\#yf$ according to (a_0, q_0, z_0, q_1) through (a_m, q_m, z_m, q_{m+1}), where $q_{m+1} = f$. To see why this equivalence holds true, take any $S \Rightarrow^* y$ with $y \in L(G)$. Examine the construction of P to see that $S \Rightarrow^* y$ in G has, in a greater detail, the form

$$S \Rightarrow^* 1\nu_1 20y302\nu_2 1 \Rightarrow^* y$$

with $\nu_1, \nu_2 \in D^*$, $\nu_1 \in \nu(a_0 \cdots a_\ell), \nu_2 \in \mu(q_k \ldots q_0)$, where $\ell, k \geq 1$. Consequently, we see that proving the equivalence requires a demonstration that in the derivation of the above form in G,

(I) $m = \ell = k$;

(II) $\nu_1 \in \nu(a_0 \cdots a_\ell)$ with $a_0, \ldots, a_\ell \in V - T$;
(III) $\nu_2 \in \mu(q_k \ldots q_0)$ with $q_0, \ldots, q_k \in W - F$.

Consider (II) above. Observe that ν_1 encodes the prefix of all the front queue symbols (including the erased symbols) rewritten during the generation of y. This is the reason why we assume that Q is a left-extended queue grammar, which records this prefix as opposed to any ordinary queue grammar, which throws it away.

During the sketch of this basic idea, we refer to all symbols that occur somewhere to the left of y as *left nonterminals*, and we refer to all symbols that occur somewhere to the right of y as *right nonterminals*. From the definition of ν, it follows that

$$1\nu_1 20 = 1103103 \cdots 103\underline{11}0 \cdots 1010 \cdots 103103 \cdots 103\underline{11}0 \cdots 101010 \cdots \\ \cdots 103103 \cdots 103\underline{11}0 \cdots 1020$$

Counting from the right to the left, we refer to the ith underlined occurrence of 11 as the ith *left turn*, $1 \leq i \leq n$. From the definition of μ, it follows that

$$302\nu_2 1 = 302301 \cdots 301\underline{030}101 \cdots 0101 \cdots 301301301 \cdots 301\underline{030}101 \cdots \\ \cdots 0101 \cdots 301301 \cdots 301\underline{030}101 \cdots 01011$$

Counting from the left to the right, we refer to the ith underlined occurrence of 030 as the ith *right turn*.

Let us examine $1\nu_1 20y302\nu_2 1 \Rightarrow^* y$ in a greater detail. The first 1 and the last 1 are produced by a production from step (3i) in the construction. Furthermore, in front of y, 20 is made by a production from (3iii), and behind y, 302 is produced by a production from (3v). Observe that all the left and right nonterminals can be removed only by π and $2 \to \varepsilon$. In $1\nu_1 20y302\nu_2 1$, there exist two occurrences of 2. Production π is applicable as soon as two occurrences of 2 appear in the rewritten string, and its application does not change the number of these occurrences. Consequently, during $1\nu_1 20y302\nu_2 1 \Rightarrow^* y$, $2 \to \varepsilon$ is applied only during the last two steps while all the preceding steps are made by using π. The first 2 is always a left nonterminal and the other occurs always as a right nonterminal. Considering these observations and π, we see that if a string contains 1 somewhere in between the left 2 and the right 2, then G cannot derive a terminal string from it. Consequently, during $1\nu_1 20y302\nu_2 1 \Rightarrow^* y$, G eliminates 1s in an inside-out way so that it always rewrites the rightmost occurrence within the left 1s and, simultaneously, the leftmost occurrence among the right 1s. Unless a string contains 0, 3, 0 in this order scattered somewhere in between the left 2 and the right 2, then G cannot apply π and derive a terminal string. More specifically, every successful derivation in G is of the form

$$S \Rightarrow^* 1\nu_1 20y302\nu_2 1 \Rightarrow^* 1203y021 \Rightarrow 2y2 \Rightarrow^2 y$$

Let $1v_3yv_41 \Rightarrow 1v_3'yv_4'1$ be a direct derivation step in $1\nu_1 20y302\nu_2 1 \Rightarrow^* 1203y021$. Then, in a greater detail, this step has one of these five forms

a) $1v_5\underline{10}\underline{20}y\underline{302}30\underline{1}v_61 \Rightarrow 1v_520y302v_61$ with $v_51020 = v_3$ and $302301v_6 = v_4$;
b) $1v_5\underline{120}y\underline{302}030\underline{1}v_61 \Rightarrow 1v_52y0302v_61$ with $v_5120 = v_3$ and $3020301v_6 = v_4$;
c) $1v_5\underline{103}\underline{2}y\underline{030}20\underline{1}v_61 \Rightarrow 1v_5203y02v_61$ with $v_51032 = v_3$ and $030201v_6 = v_4$;

d) $1v_5\underline{1}\underline{03}\underline{203}y\underline{02}\underline{0}\underline{1}v_61 \Rightarrow 1v_5203y02v_61$ with $v_5103203 = v_3$ and $0201v_6 = v_4$;
e) $1v_5\underline{1}\underline{0}\underline{203}y\underline{02}\underline{3}\underline{0}\underline{1}v_61 \Rightarrow 1v_520y302v_61$ with $v_510203 = v_3$ and $02301v_6 = v_4$.

Hereafter, to point out that a derivation step $u \Rightarrow v\ [\pi]$ satisfies one specific form of the five previous forms ($X \in \{a, b, c, d, e\}$), we write $u \Rightarrow_{X)} v$.

Suppose that the leftmost right turn occurs closer to y than the rightmost left turn does. For instance,

$$1103103 \cdots 1031031101010101020y3020301010101 \cdots 01011$$

From this string, G performs these steps

$$\begin{array}{ll} & 1103103 \cdots 1031031101010101\underline{0}2\underline{0}\ y\ \underline{302}0\underline{3}0\underline{1}010101 \cdots 01011 \\ \Rightarrow & 1103103 \cdots 1031031101010\underline{1}0\underline{2}0\ y\ \underline{0302}0\underline{1}0101 \cdots 01011 \\ \Rightarrow & 1103103 \cdots 1031031101010200\ y\ 020101 \cdots 01011 \end{array}$$

Observe that in between the two 2s, no 3 occurs, so G cannot derive a terminal string from it.

Suppose that the leftmost right turn occurs farther from y than the rightmost left turn does. For instance,

$$1103103 \cdots 10310311020y3023013013013010301010101 \cdots 01011$$

From this string, G performs these steps

$$\begin{array}{ll} & 1103103 \cdots 1031031\underline{1}0\underline{2}0\ y\ \underline{302}3\underline{0}1\underline{3}013013010301010101 \cdots 01011 \\ \Rightarrow & 1103103 \cdots 103103\underline{12}0\ y\ \underline{302}3\underline{0}1\underline{3}013010301010101 \cdots 01011 \\ \Rightarrow & 1103103 \cdots 1031032\ y\ 3023013010301010101 \cdots 01011 \end{array}$$

Observe that in between the two 2s, only one 0 occurs, so G cannot derive a terminal string from it.

Next, we give an example for some $a \in V - T$ and $q \in W - F$, where the right and left turns match.

$$1103103 \cdots 103110101010 \cdots 101020y302301301 \cdots 30103010101 \cdots 01011$$

where

$$\iota(aq) = 103103 \cdots 103110101010 \cdots 1010 \in \nu(a)$$

and

$$301301 \cdots 30103010101 \cdots 0101 \in \mu(q)$$

Consequently, G always simultaneously eliminates the ith left turn and the ith right turn in the way sketched next

$$\begin{array}{ll} & 1103103 \cdots 103110101010 \cdots 10\underline{1}0\underline{20}\ y\ \underline{302}3\underline{0}1\underline{3}01 \cdots 30103010101 \cdots 01011 \\ \Rightarrow_{a)} & 1103103 \cdots 103110101010 \cdots \underline{1}0\underline{20}\ y\ \underline{302}3\underline{0}1\underline{} \cdots 30103010101 \cdots 01011 \\ \Rightarrow_{a)} & 1103103 \cdots 103110101010 \cdots 20\ y\ 302 \cdots 30103010101 \cdots 01011 \\ \Rightarrow & \\ \Rightarrow & 1103103 \cdots 1031031\underline{1}0\underline{20}\ y\ \underline{302}3\underline{0}10301010101 \cdots 01011 \\ \Rightarrow_{a)} & 1103103 \cdots 103103\underline{12}0\ y\ \underline{302}0\underline{3}0\underline{1}010101 \cdots 01011 \\ \Rightarrow_{b)} & 1103103 \cdots 103\underline{1}03\underline{2}\ y\ \underline{0302}0\underline{1}0101 \cdots 01011 \\ \Rightarrow_{c)} & 1103103 \cdots 103203\ y\ 020101 \cdots 01011 \end{array}$$

In this way, G simulates the successful derivation of y performed by Q so

$$S \Rightarrow^* 1\nu_1 20y302\nu_2 1 \Rightarrow^* y$$

with $\nu_1, \nu_2 \in D^*$, $\nu_1 \in \nu(a_0 \cdots a_m)$ with $a_0, \ldots, a_m \in V - T$, and $\nu_2 \in \mu(q_m \ldots q_0)$ with $q_0, \ldots, q_m \in W - F$. Thus, $L(Q) = L(G)$.

Next, we illustrate the construction of a simulating scattered context grammar for a very simple left-extended queue grammar.

Example 3. To make this example as readable as possible, we introduce some notation. First, $\lceil \underline{a}, q \rceil$ denotes the code of $\iota(a, q)$ from $\nu(a)$ for any $a \in V - T$. Then, $\lfloor a, \underline{q} \rfloor$ denotes an element (code) based on $\iota(a, q)$ from $\mu(q)$ for any $q \in W - F$.

Consider a left-extended queue grammar $Q = (V, T, W, \{f\}, Ss, R)$ in normal form 2 with $V = \{S, X, a, b\}, T = \{a, b\}, W = \{s, p, q, f\}$. Let R consist of (S, s, XS, p), (X, p, aa, q) and (S, q, bb, f).

Considering the construction from Algorithm 1, we present the introduction of some productions in the resulting scattered context grammar, $G = (N, T, P, S)$, in the corresponding steps with rules not deriving words replaced with '...' for brevity:

(i): Add $\ldots$, $S \to 1\lceil \underline{S}, s \rceil \lceil \underline{S}, s \rceil 4 \lfloor S, \underline{s} \rfloor 1$, $\ldots$ into P;
(ii): For $(S, s, XS, p) \in R$, add $\ldots$, $4 \to \lceil \underline{X}, p \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \lceil \underline{S}, q \rceil 4 \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor$, $\ldots$ into P;
(iii): Add $4 \to 205$ into P;
(iv): For $(X, p, aa, q) \in R$, add $\ldots$, $5 \to aa5 \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor$, $\ldots$ into P;
(v): For $(S, q, bb, f) \in R$, add $\ldots$, $5 \to bb302 \lfloor S, \underline{q} \rfloor$, $\ldots$ into P.

Now, we explore how a derivation

$$\#Ss \Rightarrow S\#XSp \Rightarrow SX\#Saaq \Rightarrow SXS\#aabbf$$

in Q is simulated in G.

We explore the first phase of a derivation of $aabb$ in G:

$$\begin{aligned}
& S \\
\Rightarrow\ & 1\lceil \underline{S}, s \rceil \lceil \underline{S}, s \rceil 4 \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow\ & 1\delta(\lceil \underline{S}, s \rceil) \lceil \underline{X}, p \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \lceil \underline{S}, q \rceil 4 \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow\ & 1\delta(\lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil) 205 \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow\ & 1\delta(\lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil) 20aa5 \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow\ & 1\delta(\lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil) 20aabb302 \lfloor S, \underline{q} \rfloor \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1
\end{aligned}$$

According to the construction, the injection ι should handle $\mathrm{card}((V - T) \times (W - F))$ elements. Thus, for simple coding, k should be at least $\mathrm{card}((V - T) \times$

$(W-F))+2$. In the second phase of this example, we need to code six pairs—(S,s), (S,p), (S,q), (X,s), (X,p), and (X,q). Take $k=8$. Next, we introduce ι for these pairs—that is,

$$\begin{aligned}
\iota(Ss) &= 1031(10)^6 = \lceil \underline{S}, s \rceil \quad & \iota(Xs) &= (103)^4 1(10)^3 = \lceil \underline{X}, s \rceil \\
\iota(Sp) &= (103)^2 1(10)^5 = \lceil \underline{S}, p \rceil \quad & \iota(Xp) &= (103)^5 1(10)^2 = \lceil \underline{X}, p \rceil \\
\iota(Sq) &= (103)^3 1(10)^4 = \lceil \underline{S}, q \rceil \quad & \iota(Xq) &= (103)^6 110 = \lceil \underline{X}, q \rceil \\
\kappa(Ss) &= (301)^6 030101 = \lfloor S, \underline{s} \rfloor \quad & \kappa(Sq) &= (301)^4 0301(01)^3 q = \lfloor S, \underline{q} \rfloor \\
\kappa(Xp) &= (301)^2 0301(01)^5 = \lfloor X, \underline{p} \rfloor
\end{aligned}$$

Now we illustrate an erasure in the second phase of the derivation with $y = aabb$ in G, which is similar to erasure in [6]:

$$\begin{aligned}
& 1\delta(\lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil) \lceil \underline{S}, q \rceil (103)^3 1(10)^4\ 20y302\ (301)^4 0301(01)^3 \lfloor S, \underline{q} \rfloor \delta(\lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor) 1 \\
\Rightarrow^8\ & 1\delta(\lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil)(103)^3 1(10)^4\ 203y02\ (301)^4 0301(01)^3 \delta(\lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor) 1 \\
\Rightarrow^8\ & 1\lceil \underline{S}, s \rceil \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil (103)^5 1(10)^2\ 203y02\ (301)^2 0301(01)^5 \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow^8\ & 1\lceil \underline{S}, s \rceil \lceil \underline{S}, s \rceil (103)^5 1(10)^2\ 203y02\ (301)^2 0301(01)^5 \lfloor S, \underline{s} \rfloor \lfloor S, \underline{s} \rfloor 1 \\
\Rightarrow^8\ & 1\lceil \underline{S}, s \rceil 1031(10)^6\ 203y02\ (301)^6 030101 \lfloor S, \underline{s} \rfloor 1 \\
& 1\ 1031(10)^6\ 203y02\ (301)^6 030101\ 1 \\
\Rightarrow^8\ & 1\ 203y02\ 1 \\
\Rightarrow\ & 2\ y\ 2 \\
\Rightarrow^2\ & y
\end{aligned}$$

Lemma 4. *Algorithm 1 is correct.*

The rigorous proof is due to the lengthiness included in the appendix.

Theorem 1. *Let L be a recursively enumerable language; then, there is a scattered context grammar $G=(N,T,P,S)$ such that (a) $L=L(G)$, (b) N has no more than six nonterminals, and (c) P has no more than one non-context-free production.*

Proof. Let us consider the algorithm based on the Algorithm 1 such as:

Algorithm 2 *Input: A left-extended queue grammar $Q=(V,T,W,F,s,R)$ in normal form 2.*
Output: A scattered context grammar $G=(N,T,P,4)$ such that $L(G)=L(Q)$.
Method:

1. *Set $N=\{4,5\}\cup D$.*

2. *Consider the definition of ι (see Definition 2) with $A = V - T$ and $B = W - F$. Based on A, B, and ι, consider specific definitions of κ, μ, ν, and δ (see Definitions 2, 4, and 5).*
3. *Initialize M with $\emptyset$. Perform (3i) through (3v), given next, to construct M.*
   (i) *if $s = a_0q_0$, where $a_0 \in V - T$ and $q_0 \in W - F$, then add $4 \to 1\delta(u)4w1$ to M, for all $u \in \nu(a_0)$ and $w \in \mu(q_0)$;*
   (ii) *if $(a, q, y, p) \in R$, where $a \in V - T$, $p, q \in W - F$, and $y \in (V - T)^*$, then add $4 \to \delta(u)4w$ to M, for all $u \in \nu(y)$ and $w \in \mu(p)\mu(q)$;*
   (iii) *add $4 \to 205$ to M;*
   (iv) *if $(a, q, y, p) \in R$, where $a \in V - T$, $p, q \in W - F$, $y \in T^*$, then add $5 \to y5w$ to M, for all $w \in \mu(p)\mu(q)$;*
   (v) *if $(a, q, y, p) \in R$, where $a \in V - T$, $q \in W - F$, $y \in T^*$, and $p \in F$, then add $5 \to y302w$ to M, for all $w \in \mu(q)$.*
4. *Set $O = \{(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2),\ 2 \to \varepsilon\}$.*
5. *Set $P = M \cup O$.*

Lemma 5. *Algorithm 2 is correct.*

It allows two new subsets of derivations compared to Algorithm 1. Therefore, to show that a grammar from Algorithm 2 generates identical language as a grammar from Algorithm 1, we need to show that all new sets of derivations do not change the language. Therefore, we split the two new derivations in two Claims to prove Algorithm 2 is correct.

Claim. If a derivation of grammar G begins with the rule $4 \to 1\delta(u)4w1$ and there are n applications of the rule $4 \to 1\delta(u)4w1$, where $n \neq 1$, then it does not generate any words.

Proof. If there is n applications of the rule $4 \to 1\delta(u)4w1$ after the first derivation step with the rule $4 \to 1\delta(u)4w1$, where $n \geq 1$ then a sentential form is $(1\delta(u))^n 1$ $\delta(u)41\delta(u)(w1)^n$. Therefore we would have to be able in the annihilation phase to eliminate more nonterminals 1 but we do not have enough nonterminals 0 to use the rule $(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$ which is only possible way to eliminate nonterminal 1. Therefore, this does not generate any words.

Claim. If a derivation of grammar G does not start with rule $4 \to 1\delta(u)4w1$, then it does not generate any words.

Proof. There is no rule to get rid of nonterminal 0 since the only possibility to get rid of 0 is rule $(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$, which is not possible because there cannot be a nonterminal 1 before a nonterminal 0. Or if nonterminals 1 are generated in different positions, then they interrupt the dependency between codes and anticodes and, in this case, do not generate any words. Therefore, the grammar does not generate any words.

□

4 Conclusion

In conclusion, we would like to propose open problems. Firstly, is it possible to reduce the grammar further to five or fewer nonterminals? Secondly, we already know that two-nonterminal scattered context grammars are computationally complete (see [1]). So are scattered context grammars with a single non-context-free production (see [6]). Consider two-nonterminal scattered context grammars with one non-context-free production. Are they computationally complete, too? If not, then study, from a more general viewpoint, whether there exist constants n and m such that n-nonterminal scattered context grammars with m non-context-free productions are computationally complete.

Acknowledgments. This work was supported by Brno University of Technology grant FIT-S-23-8209.

References

1. Csuhaj-Varjú, E., Vaszil, G.: Scattered context grammars generate any recursively enumerable languages with two nonterminals. Information Processing Letters **110**(20), 902–907 (2010)
2. Dassow, J., Stiebe, R.: Nonterminal complexity of some operations on context-free languages. Fundamenta Informaticae **83**(1-2), 35–49 (2008). https://doi.org/10.3233/FUN-2008-831-205
3. Geffert, V.: Context-free-like forms for the phrase-structure grammars. In: Chytil, M.P., Koubek, V., Janiga, L. (eds.) Mathematical Foundations of Computer Science 1988. pp. 309–317. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
4. Ginsburg, S.: Algebraic and automata-theoretic properties of formal languages. Journal of Symbolic Logic **41**(4), 788–789 (1976). https://doi.org/10.2307/2272400
5. Kleijn, H.C.M., Rozenberg, G.: On the generative power of regular pattern grammars. Acta Informatica **20**(4), 391–411 (Dec 1983). https://doi.org/10.1007/BF00264281
6. Křivka, Z., Meduna, A.: Scattered context grammars with one non-context-free production are computationally complete. Fundamenta Informaticae **179**(4), 361–384 (2021). https://doi.org/10.3233/FI-2021-2028
7. Maurer, H.A., Penttonen, M., Salomaa, A., Wood, D.: On non context-free grammar forms. Mathematical Systems Theory **12**, 297–324 (1978)
8. Maurer, H.A., Salomaa, A., Wood, D.: EOL forms. Acta Informatica **8**, 75–96 (1977)
9. Meduna, A.: Automata and Languages: Theory and Applications. Springer, London (2000)
10. Meduna, A.: Generative power of three-nonterminal scattered context grammars. Theoretical Computer Science **246**(1), 279–284 (2000). https://doi.org/10.1016/S0304-3975(00)00153-5
11. Meduna, A., Techet, J.: Scattered Context Grammars and their Applications. WIT Press, UK, WIT Press (2010), `https://www.fit.vut.cz/research/publication/c62000/`
12. Meduna, A., Zemek, P.: Regulated Grammars and Automata. Springer (2014), `https://www.fit.vut.cz/research/publication/c111518/`

13. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 1: Word, Language, Grammar. Springer, New York (1997)
14. Čulik, K., Maurer, H., Ottmann, T.: On two-symbol complete EOL forms. Theoretical Computer Science **6**(1), 69–92 (1978). https://doi.org/10.1016/0304-3975(78)90005-1