Prescribed Teams of Rules Working on Several Objects

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

Prescribed Teams of Rules Working on Several Objects

Artiom Alhazov[1], Rudolf Freund[2], Sergiu Ivanov[3], and Sergey Verlan[4]

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[3] IBISC, Univ. Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

[4] Univ. Paris Est Creteil, LACL, 94010, Creteil, France
verlan@u-pec.fr

Abstract. In this paper we consider prescribed sets of rules working on several objects either in parallel – in this case the rules have to take different objects – or else sequentially in any order – in this case several rules may take the same object to work on.
We show that prescribed teams of size two, i.e., containing exactly two rules, are sufficient to obtain computational completeness for strings with the simple rules being of the form $aI_R(b)$ – meaning that a symbol b can be inserted on the right-hand side of a string ending with a – and $D_R(b)$ meaning that a symbol b is erased on the right-hand side of a string. This result is established for systems starting with three initial strings. Using prescribed teams of size three, we may start with only two strings, ending up with the output string and the second string having been reduced to the empty string. We also establish similar results when using the generation of the anti-object b^- on the right-hand side of a string instead of deleting the object b, i.e. $bI_R(b^-)$ inserts the anti-object b^- and the annihilation rule $b\,b^-$ assumed to happen immediately whenever b and b^- *meet* deletes the b.

Keywords: computational completeness, insertion-deletion systems, prescribed teams, anti-objects

1 Introduction

Cooperation in its different forms is a feature which has attracted a lot of research in formal languages and theory of computation. Indeed, the family of context-free languages is quite well studied, and it is folklore that some easy to describe

languages are not context free, e.g. $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, the copy language ww, $w \in V^*$ for some alphabet V, etc. On the other hand, it is just as well-known that full cooperation – classically expressed by allowing multiple symbols in the left-hand side of the rewriting rules – yields computational completeness in many situations. We refer to [7] for a comprehensive overview of the multiple facets of cooperation.

This situation aroused interest in the expressive power of intermediate forms, in which some cooperation is allowed, but full cooperation is avoided. One of the possible implementations is by forcing some rules to only be applied together, a classic example being matrix grammars, in which the rules are grouped into sequences, and must be applied one after another, in order. Here, we focus on a less strict variant, in which the rules of a group must be applied together, but the order of their application is not imposed. This control mechanism is known as *prescribed teams* and was introduced in [1].

In this paper, we define prescribed teams in a general framework for rewriting as a control mechanism over abstract rules. In order to study the computational power of this device, we specialize the rules to string insertion and deletion operations. More concretely, we focus on insertions and deletions which are only allowed to occur at the right end of a string and which may depend on a finite context. We show that by allowing computation to happen on two strings at the same time, such insertion and deletion rules grouped in prescribed teams containing three rules can simulate any Turing machine (Theorem 1). We remark that even though insertion-deletion operations with matrix control have been quite extensively investigated (e.g., [3–6]), the power of insertion and deletion operations restricted to the right of the string and equipped with matrix control has never been studied to the best of our knowledge.

2 Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. Given a string $w = w_1 \dots w_n$ over V, with $w_i \in V$, $1 \leq i \leq n$, its mirror image is $w^R = w_n \dots w_1$. Moreover, instead of $w = w_1 \dots w_n$ we may also write $w = w(1) \dots w(n)$.

The cardinality of a set M is denoted by $|M|$. For further notions and results in formal language theory we refer to textbooks like [2] and [7].

2.1 Systems with Prescribed Teams of Rules

The main model we consider in this paper is a system of arbitrary objects which starts on a finite set of such objects and has prescribed teams of rules to work on these objects until no such team can be applied any more.

Definition 1. *A* system with prescribed teams of rules *is a construct*

$$G = (O, O_T, P, R, A) \quad \text{where}$$

- O *is a set of objects;*
- $O_T \subseteq O$ *is a set of* terminal objects*;*
- P *is a finite set of* rules*, i.e.,* $P = \{p_i \mid 1 \leq i \leq m\}$*, for some* $m \geq 0$*, and* $p_i \subseteq O \times O$*;*
- $R = \{T_1, \ldots, T_n\}$ *is a finite set of sets of rules from* P *called* prescribed teams*, i.e.,* $T_i \subseteq P,\ 1 \leq i \leq n$*;*
- A *is a finite set of initial objects in* O*.*

A rule $p \in P$ is called *applicable* to an object $x \in O$ if and only if there exists at least one object $y \in O$ such that $(x, y) \in p$; in this case we also write $x \Longrightarrow_p y$.

$|T_i|$ is called the *size* of the prescribed team T_i. If all prescribed teams have at most size s, then G is called a *system (with prescribed teams) of size* s. If all prescribed teams have the same size s, then G is called a *homogenous system (with prescribed teams) of size* s. The number of initial objects in A is called the *degree* of the system.

Computations in a system with prescribed teams of rules We may consider different variants of applications of the prescribed teams of rules as already indicated above when working on several objects, starting with the initial objects in A; in any case, at the beginning of a computation step we first have to choose a suitable team T_k:

parallel each rule in T_k is applied to a different object in the current set of objects; T_k can only be applied if every rule in T_k can be applied; we observe that in the parallel case the number of rules in any prescribed team must not exceed the number of initial objects;

sequential the rules in T_k are applied sequentially in any order – in this case several rules may take the same object to work on in several sequential derivation steps, yet again each rule is to be applied exactly once.

T_k can only be applied if every rule in T_k can be applied in the given derivation mode.

We mention that we do not consider the case where several rules from T_k may be applied to the same object at the same moment in parallel.

In case the mode of application is not clear from the context, we may specify it in the definition of the system, i.e., we then write

$$G = (O, O_T, P, T_1, \ldots, T_n, A, d)$$

where $d \in \{parallel, sequential\}$.

A *derivation step* in G using T_k in the mode d then can be written as

$$\{O_1, \ldots, O_m\} \Longrightarrow_{T_k,d} \{O'_1, \ldots, O'_m\}$$

where $\{O_1, \ldots, O_m\}$ is the current set of objects and $\{O'_1, \ldots, O'_m\}$ is the set of objects obtained after the application of T_k in mode d.

The derivation relation $\Longrightarrow_{G,d}$ of the system G in mode d then is the union of all derivation relations $\Longrightarrow_{T_k,d}$, $1 \leq k \leq n$. Given two objects $u, v \in O$, we write $u \Longrightarrow_d v$ ($u \Longrightarrow_{T_k,d} v$) to indicate that v can be obtained from u in one derivation/computation step from u (using the prescribed team T_k) in the derivation mode d. The reflexive and transitive closure of $\Longrightarrow_{G,d}$ is denoted by $\Longrightarrow_{G,d}*$. If the derivation mode d is obvious from the context, d is omitted in all these notations.

For arbitrary systems working in the sequential mode, we can prove the following complexity result:

Lemma 1. *Given a homogenous system*

$$G = (O, O_T, P, T_1, \ldots, T_n, A, sequential)$$

of size 1*, its computations are the union of the computations of the* m *systems*

$$G_k = (O, O_T, P, T_1, \ldots, T_n, A_k, sequential),$$

$1 \leq k \leq m$*, where* $A = \{A_k \mid 1 \leq k \leq m\}$.

Proof. In the sequential mode, a prescribed team can only work on one of the m subjects. Hence, the computations on the initial objects are independent of each other. Therefore also the terminal objects must be obtained from one of these initial objects A_k by the corresponding system G_k. □

2.2 Matrix Grammars Working on Several Objects

A model quite closely related to systems with prescribed teams of rules is the model of matrix grammars usually only considered to work on one object:

Definition 2. *A* matrix grammar working on several objects *is a construct*

$$G = (O, O_T, P, M, A) \quad \text{where}$$

- O *is a set of objects;*
- $O_T \subseteq O$ *is a set of* terminal objects*;*
- P *is a finite set of* rules*, i.e.,* $P = \{p_i \mid 1 \leq i \leq n\}$*, for some* $n \geq 0$*, and* $p_i \subseteq O \times O$*;*
- M *is a finite set of sequences of the form* $(p_1, \ldots, p_n)$*,* $n \geq 1$*, of rules in* P*; an element of* M *is called a* matrix*;*
- A *is a finite set of initial objects in* O*.*

A derivation step in the matrix grammar consists of choosing a matrix and applying the sequence of rules in the matrix in this order, yet allowing several rules to be applied to the same object.

Lemma 2. *Any system with prescribed teams of rules* $G = (O, O_T, P, R, A)$ *working in the sequential mode can be simulated by a matrix grammar* $G = (O, O_T, P, M, A)$.

Proof. From a prescribed team T in R we immediately get the corresponding set of matrixes for M by taking every possible sequence of the rules in T, i.e.,

$$M = \{[p_1 \dots, p_n] \mid \{p_1 \dots, p_n\} \in R, |\{p_1 \dots, p_n\}| = n\}.$$

Hence, matrix grammars are at least as powerful as systems with prescribed teams of rules working in the sequential mode. □

2.3 Turing Machines

The computational model we we will simulate for showing computational completeness for the systems with prescribed teams of rules defined above are Turing machines with one tape with left boundary marker Z_0:

Definition 3. *A Turing machine is a*

$$M = (Q, V, T_1, T_2, \delta, q_0, q_1, Z_0, B)$$

where

- Q *is a finite set of* states,
- V *is the* tape alphabet,
- $T_1 \subset V \setminus (\{Z_0, B\})$ *is the* input alphabet,
- $T_2 \subset V \setminus (\{Z_0, B\})$ *is the* output alphabet,
- $\delta \subset (Q \times V) \to (Q \times V \times \{L, R\})$ *is the* transition function,
- q_0 *is the* initial state,
- q_1 *is the* final state,
- $Z_0 \in V$ *is the* left boundary marker,
- $B \in V$ *is the* blank symbol.

A configuration of the Turing machine M can be written as Z_0uqvB^ω, where $u \in (V \setminus (\{Z_0\}))^*$, $v \in (V \setminus (\{Z_0\}))^+ \cup \{\lambda\}$, and B^ω indicates the remaining empty part of the tape, offering an unbounded number of tape cells initially carrying the blank symbol B; moreover, the current state $q \in Q$ is written to the right of the tape cell on which the read-write head of the Turing machine currently stands.

A transition between configurations is carried out according to the transition function δ in the following way for $(q, X; p, Y, D) \in \delta$:

- the state changes from q to p,
- the symbol X currently read is replaced by the symbol Y,
  and for
  $D = R$ the read-write head goes one step to the right; i.e.,
  $Z_0uXqUvB^\omega \Longrightarrow Z_0uYUpvB^\omega$;
  $D = L$ the read-write head goes one step to the left, i.e.,
  $Z_0uXqvB^\omega \Longrightarrow Z_0upYvB^\omega$;
  observe that the read-write head can never go to the left of Z_0.

For the derivation relation $\Longrightarrow$ as defined above, its reflexive and transitive closure is defined by $\Longrightarrow^*$.

Turing machines are well-known automata which can compute any partial recursive relation $f : {T_1}^* \to {T_2}^*$:

A successful computation of the Turing machine M starts with the input string w_{input} on its tape with the configuration

$$Z_0 w_{input} q_0 B^\omega$$

and ends up with the output string w_{output} on its tape with the configuration

$$Z_0 w_{output} q_1 B^\omega,$$

i.e.,

$$Z_0 w_{input} q_0 B^\omega \Longrightarrow^* Z_0 w_{output} q_1 B^\omega.$$

A successful computation *halts* in the final state q_1; without loss of generality, we may assume that from the final state q_1 no transition is possible.

The *generation* of a language $L \subseteq {T_2}^*$ can be seen as computing a partial recursive relation $g_L : \{\lambda\} \to {T_2}^*$, *acceptance* of a language $L \subseteq {T_1}^*$ as computing a partial recursive function $h_L : {T_2}^* \to \{\lambda\}$.

In order to show that another (string) computing device is computationally complete, an option is to simulate Turing machines, which is exactly what we will do in the next section.

3 Prescribed Teams of Rules on Strings

In this section we consider the objects to be strings. Moreover, we will restrict ourselves to special variants of insertion an deletion rules to be applied to strings.

3.1 Definitions for Prescribed Teams of Insertion and Deletion Rules on Strings

We are going to use the following notations:

Definition 4.
right insertions and deletions with contexts:

$uI_R(v)$ *to a string ending with u, v is appended;*
$uD_R(v)$ *from a string ending with uv, the end v is deleted.*

left insertions and deletions with contexts:

$uI_L(v)$ *to a string beginning with u, v is added as prefix;*
$uD_L(v)$ *from a string beginning with vu, the prefix v is deleted.*

right and left substitutions

$S_R(u,v)$ *in a string ending with* u, u *is replaced by* v;
$S_L(u,v)$ *in a string beginning with* u, u *is replaced by* v.

Both insertions and deletions can easily be replaced by substitutions:

Lemma 3. *Right and left insertions and deletions can be replaced by right and left substitutions, respectively.*

Proof. Right and left insertions and deletions are replaced by right and left substitutions, respectively, in the following way:

- $uI_R(v)$ by $S_R(u,uv)$;
- $uI_L(v)$ by $S_L(u,vu)$;
- $uD_R(v)$ by $S_R(uv,u)$;
- $uD_L(v)$ by $S_L(vu,u)$.

We remark that contexts need not be considered with substitutions as they can be deleted and re-inserted immediately. □

Example 1. Consider the system

$$G = (T'^*\{c'c\}T^*, T'^*\{c'c\}T^*, P, T_1, \ldots, T_n, \{c'c\})$$

working in the sequential derivation mode, where T is an arbitrary alphabet, $T' = \{a' \mid a \in T\}$, $c \notin T$, cc' is the only initial string, the set P consists of two disjoint sets of rules R and R',

$$\begin{aligned} R &= \{aI_R(b) \mid a \in T \cup \{c\}, b \in T\}, \\ R' &= \{a'I_L(b') \mid a \in T \cup \{c\}, b \in T\}, \end{aligned}$$

i.e., for each rule $p = aI_R(b)$ in R we have the corresponding rule $p' = a'I_L(b')$. The set of prescribed teams then is formed by all possible couples p, p', i.e.,

$$\bigcup_{1 \le i \le n} T_i = \bigcup_{p \in R} \{\, p, p'\}.$$

As both sets of rules work with disjoint alphabets, we get the derivations

$$\begin{aligned} \{c'c\} &\Longrightarrow_{\{cI_R(b_1), c'I_L(b'_1)\}} \{b'_1c'cb_1\} \ldots \\ &\Longrightarrow_{\{b_{m-1}I_R(b_m), b'_{m-1}I_L(b'_{m-1})\}} \{b'_m \ldots b'_1c'cb_1 \ldots b_m\} \end{aligned}$$

Hence, for the language generated by G we obtain (the context-free, but non-regular) language

$$\{(w')^Rc'cw \mid w \in T^*\}.$$

We observe that we have started with only one string and only used left and right insertion of one symbol in the context of another symbol.

We now are going to show that right insertion rules inserting just one symbol at the end of a string in the left context of just another single symbol together with right deletion rules eliminating the last symbol of a string, even without using any context, are already sufficient to obtain computational completeness. A similar result holds for the corresponding variants of left insertion and deletion rules.

3.2 Results for Prescribed Teams of Insertion and Deletion Rules on Strings

We first establish a result for systems of degree 2 and size 3.

Theorem 1. *The computations of a Turing machine M can be simulated by a homogenous string system with prescribed teams of size* 3 *and degree* 2 *using only rules of the form $aI_R(b)$ and $D_R(b)$.*

Proof. Let

$$M = (Q, V, T_1, T_2, \delta, q_0, q_1, Z_0, B)$$

be a Turing machine. In order to represent the configurations of M as finite strings, we use a right end marker Z_1 to mark the end of a finite representation $Z_0uqvB^mZ_1$ of the configuration Z_0uqvB^ω, where m may be any natural number ≥ 0; m depends on how far on the tape the read-write head has already proceeded during a computation.

We now construct a system G with prescribed teams using only rules of the form $aI_R(b)$ and $D_R(b)$ which can simulate the computations of the given Turing machine M. The basic idea is folklore – a configuration $Z_0uqvB^mZ_1$ is represented by two strings Z_0uq and $(vB^mZ_1)^R = Z_1B^mv^R$, which like stacks are only affected at the end of the strings. A special technical detail is that when we reach a situation where the second string is Z_1, no transition to the right is possible immediately, we first have to insert an additional blank B to then continue with the second string Z_1B. Moreover, in order to allow the rules to distinguish between the two strings, the second string is written in the primed alphabet $V' = \{X' \mid X \in V\}$.

The system with prescribed teams using only rules of the form $aI_R(b)$ and $D_R(b)$ is constructed with only two strings being processed, which represent these two parts of the configuration; the size of the teams can be restricted to be exactly three. Moreover, the system is constructed in such a way that the teams have to be applied in a sequential way, but the sequence of the application of the rules does not matter, yet the sequence in which the rules are given in the sets indicates in which sequence the rules are to be applied to obtain the desired result.

The main idea is that in every derivation step using a prescribed team of size 3 we only simulate one right insertion or deletion on one of the two strings in the second step, whereas the first step eliminates the symbol representing the current state and the third step inserts the symbol representing the next state. The state symbols always are placed at the end of the first string. Moreover, the three rules always must be applied exactly in this order, and each of the rules is applied exactly once.

$$G = ((V \cup Q) \cup (V \cup Q)' \cup Q'')^*, \{Z_0\}{T_2}^*, P, R, A, \mathit{sequential}),$$
$$A = \{Z_0 w_{\mathit{input}} q_0, {Z_1}'\}.$$

The set of rules P can be collected from the prescribed teams of rules described in the following for the transitions given by δ, and the intermediate states defined below are collected in Q'':

$(\mathbf{q}, \mathbf{X}; \mathbf{p}, \mathbf{Y}, \mathbf{L})$ With the first prescribed team the symbol X at the end of the first string is eliminated remembering the rule to be applied and the symbol W to the left of X in the intermediate state $[W; q, X; p, Y, L; U]$, where U is the first symbol at the end of the second string. Then the new symbol Y in its primed version is inserted to the second string using the second prescribed team.

Observe that the two teams must be applied exactly in this order, as the intermediate state $[W; q, X; p, Y, L; U]$ carrying all necessary information cannot be used otherwise:

1. $\{D_R(q), D_R(X), WI_R([W; q, X; p, Y, L; U])\}$, $W, U \in V$;
   the symbol W to the left of X and the symbol U at the end of the second string have to be guessed in a non-deterministic way.
   The rule $D_R(X)$ cannot be applied before the rule $D_R(q)$, as the deletions can only happen in the order the symbols appear at the end of the first string.
   $WI_R([W; q, X; p, Y, L; U])$ cannot be applied before the other two rules, as these then would not be applicable any more.
2. $\{D_R([W; q, X; p, Y, L; U]), U'I_R(Y'), WI_R(p)\}$.
   The rule $D_R([W; q, X; p, Y, L; U])$ must be applied before the rule $WI_R(p)$, because W is not a state symbol.
   The rule $U'I_R(Y')$ on the second string can be applied at any moment.

With these two teams, we obtain the following derivation:

$\{uWXq, {Z_1}'{B'}^m {v'}^R U'\} \Longrightarrow$
$\{uW[W; q, X; p, Y, L; U], {Z_1}'{B'}^m {v'}^R U'\} \Longrightarrow$
$\{uWp, {Z_1}'{B'}^m {v'}^R U'Y'\}$

$(\mathbf{q}, \mathbf{X}; \mathbf{p}, \mathbf{Y}, \mathbf{R})$ With the first prescribed team the symbol X at the end of the first string is eliminated, then the new symbol Y is inserted to the first string instead of X; with the third prescribed team the last symbol U (in its primed version) of the second string is deleted and remembered in the intermediate state $[q, X; p, Y, L; U']$; finally, using the fourth prescribed team, this symbol U' is inserted at the end of the first string.

Observe that the four teams must be applied exactly in this order, as the intermediate states

$$[W; q, X; p, Y, L; U],\ [W; q, X; p, Y, L; U'],\ \text{and}[\mathrm{W; q, X; p, Y, L; U''}]$$

carrying all necessary information cannot be used otherwise.

1. $\{D_R(q), D_R(X), WI_R([W; q, X; p, Y, L; U])\}$, $W, U \in V$;
   the symbol W to the left of X and the symbol U at the end of the second string have to be guessed in a non-deterministic way.
   The rule $D_R(X)$ cannot be applied before the rule $D_R(q)$, as the deletions can only happen in the order the symbols appear at the end of the first string.
   $WI_R([W; q, X; p, Y, L; U]$ cannot be applied before the other two rules, as these rules then would not be applicable any more.

2. $\{D_R([W;q,X;p,Y,L;U]), WI_R(Y), YI_R([q,X;p,Y,L;U'])\}$;
   The rule $D_R([W;q,X;p,Y,L;U])$ must be applied first, i.e., before the other two rules, as these rules require a left context not being a state. Using the same argument it follows that the rule $WI_R(Y)$ must be applied before the rule $YI_R([q,X;p,Y,L;U'])$.
3. $\{D_R([W;q,X;p,Y,L;U'])), D_R(U'), YI_R([q,X;p,Y,L;U''])\}$;
   The rule $D_R([W;q,X;p,Y,L;U'])$ must be applied before the rule $YI_R([q,X;p,Y,L;U''])$, because Y is not a state symbol. The rule $D_R(U')$ on the second string can be applied at any moment.
4. $\{D_R([W;q,X;p,Y,L;U'']), YI_R(U), UI_R(p)\}$.
   The rule $D_R([W;q,X;p,Y,L;U''])$ must be applied first, i.e., before the other two rules, also working on the first string, as these rules require a left context not being a state. Using the same argument it follows that the rule $UI_R(p)$ must be applied after the rule $YI_R(U)$.

With these four teams, we obtain the following derivation:

$\{uWXq, {Z_1}'{B'}^m {v'}^R U'\} \Longrightarrow$
$\{uW[W;q,X;p,Y,L;U], {Z_1}'{B'}^m {v'}^R U'\} \Longrightarrow$
$\{uWY[W;q,X;p,Y,L;U'], {Z_1}'{B'}^m {v'}^R U'\} \Longrightarrow$
$\{uWY[W;q,X;p,Y,L;U''], {Z_1}'{B'}^m {v'}^R\} \Longrightarrow$
$\{uWYUp, {Z_1}'{B'}^m {v'}^R\}$

insertion of B If one more blank is needed in front of the right end marker ${Z_1}'$, an intermediate step for any state q being part of a rule $(q,B;p,Y,R)$ to be applied must be carried out:
$\{D_R(q), {Z_1}' I_R(B'), WI_R(q)\}$, $W \in V$;
the symbol W at the end of the first string has to be guessed in a non-deterministic way.

Derivation:

$\{uWq, {Z_1}'\} \Longrightarrow \{uWq, {Z_1}'B'\}$

final cleaning First the remaining blanks are removed:
$\{D_R(q_1), D_R(B'), aI_R(q_1)\}$, $a \in T_2 \cup \{Z_0\}$.
The rule $D_R(q_1)$ must be applied before the rule $aI_R(q_1)$, because a is not a state symbol. The rule $D_R(B')$ on the second string can be applied at any moment.
Moreover, observe that with the final state q_1 of the Turing machine no transitions are defined any more, i.e., with q_1 the Turing machine has halted.

Applying this team of rules m times, we obtain the following derivation:
$\{Z_0 w_{output} q_1, {Z_1}'{B'}^m\} \Longrightarrow^m \{Z_0 w_{output} q_1, {Z_1}'\}$

When all the blank symbols on the second string have been erased, finally, the second string is completely eliminated, at the same time also the final state at the end of the terminal string is erased; in order to have exactly teams of size 3 we insert one blank again in an intermediate step using the intermediate state ${q_1}'$:

$\{D_R(q_1), {Z_1}'I_R(B'), aI_R({q_1}')\}$, $a \in T \cup \{Z_0\}$;
The rule $D_R(q_1)$ must be applied before the rule $aI_R({q_1}')$, because a is not a state symbol. The rule ${Z_1}'I_R(B')$ on the second string can be applied at any moment.

Finally, we use the following prescribed team of rules:
$\{D_R({q_1}'), D_R(B'), D_R({Z_1}')\}$;
When using this final team, the only restriction on the sequence how they are to be applied is that $D_R(B')$ must be applied before $D_R({Z_1}')$.

Derivation:
$\{Z_0 w_{output} q_1, {Z_1}'\} \Longrightarrow \{Z_0 w_{output} {q_1}', {Z_1}'B'\} \Longrightarrow^m \{Z_0 w_{output}, \lambda\}$

In sum, we observe that every computation of the Turing machine M

$$Z_0 w_{input} q_0 B^\omega \Longrightarrow Z_0 w_{output} q_1 B^\omega$$

can be simulated in G by a computation

$$\{Z_0 w_{input} q_0, {Z_1}'\} \Longrightarrow \{Z_0 w_{output}, \lambda\}.$$

A result $Z_0 w_{output}$ obtained in G represents the string w_{output}. Observe that Z_0 cannot be avoided as only non-empty strings can be handled on the first string of the system. □

We now show the somehow symmetric case using three initial strings, but only teams of size two:

Theorem 2. *The computations of a Turing machine M can be simulated by a homogenous string system with prescribed teams of size* 2 *and degree* 3 *using only rules of the form $aI_R(b)$ and $D_R(b)$, either working in the sequential or the parallel derivation mode.*

Proof. For $d \in \{sequential, parallel\}$, we construct the system

$$G' = ((V \cup Q) \cup (V \cup Q)' \cup Q'')^*, \{Z_0\}{T_2}^*, P, R, A', d),$$
$$A' = \{Z_0 w_{input} q_0, {Z_1}', p_0\}.$$

and follow the constructions given in the proofs of Theorem 1.

The initial set of strings now contains a third string which step by step will collect the (labels of) the rules applied in the computation steps. Except for the last cleaning step, the prescribed teams in the proof of Theorem 1 all are of the form $\{D_R(p), rule, WI_R(q)\}$, where W is an arbitrary symbol, but not a state symbol, and $rule$ is an insertion or deletion rule on the first or second string. With the help of the third string, the two rules $\{D_R(p), WI_R(q)\}$ now can be replaced by one single insertion rule $pI_R(q)$.

In contrast to the final cleaning established in the proof of Theorem 1, the final team now is the following:
$\{q_1 I_R({q_1}'), D_R({Z_1}')\}$.

At the end, the result of a successful computation is given by the first string and the second string has been reduced to the empty string as in the preceding proof, but the third string remains as a kind of garbage.

We finally remark that this construction, in contrast to the one given in the proof of Theorem 1, now not only works in the *sequential* derivation mode, but as well in the *parallel* derivation mode, as the rules in each prescribed team work on different strings. □

Remark 1. As outlined in the preceding proof, the prescribed teams of rules of size 2 only work on one the two strings representing the left and right part of the Turing tape by either deleting or inserting one symbol. The left context of the insertion rules is only needed to indicate to which string the new symbol has to be added.

In that sense, instead of simulating the computations of a Turing machine we could also have simulated the computations of a 2-stack automaton which also used the operations of deleting (*pop*) one symbol oder inserting (*push*) one symbol on one of its two stacks, together with changing state.

Whereas the preceding proof might have become even easier when just simulating *pop* and *push* actions on the two stacks, the intuition what these two stacks in fact represent would have got lost, especially why we need to insert a blank symbol when reaching the bottom of the second stack.

Remark 2. The third string remaining in the construction of the system G' constructed in the proof of Theorem 2 can be interpreted as the Szilard word of the computation in the system, hence, it is not only garbage, but carries useful information.

We now use the idea of anti-objects to replace the deletions of a symbol b by the insertion of the corresponding anti-symbol b^-, where in addition we assume that b and b^- immediately annihilate each other immediately before the next rules are applied. Therefore, any deletion rule $D_R(b)$ can be replaced by the corresponding insertion rule $bI_R(b^-)$. Hence, based on the preceding results, we immediately obtain the following ones:

Corollary 1. *The computations of a Turing machine M can be simulated by a homogenous string system with prescribed teams of size* 3 *and degree* 2 *using only rules of the form $aI(b)$ and $bI(b^-)$.*

Corollary 2. *The computations of a Turing machine M can be simulated by a homogenous string system with prescribed teams of size* 2 *and degree* 3 *using only rules of the form $aI(b)$ and $bI(b^-)$.*

3.3 Complexity Considerations for Prescribed Teams of Rules on Strings

In the preceding subsection we have already seen that there seems to be a trade-off between the size and the degree of string system with prescribed teams using

only rules of the form $aI_R(b)$ and $D_R(b)$, where one parameter has to be three and the other one can be restricted to two. We now especially consider the generating case.

- According to Lemma 1, having only systems of size 1, we only can get finite unions of languages generated by systems of size 1 and degree 1.
- Moreover, as matrix grammars according to Lemma 2 are at least as powerful as string systems with prescribed teams, an upper bound for string systems with prescribed teams of degree 1 are usual matrix grammars working on one string with the same rules.
- According to Lemma 3, insertions and deletions on the right can be replaced by substitutions on the right.

The proof of the following lemma is left to the interested reader.

Lemma 4. *The effect of a matrix using right substitution rules on one string can be simulated by just one right substitution rule, i.e., for any matrix grammar using right substitution rules on one string we can construct a standard sequential grammar using right substitution rules.*

In order to show that systems of either size 1 or degree 1 cannot generate more than regular languages, it therefore suffices to prove the following result:

Lemma 5. *Sequential grammars using only right substitution rules can only generate regular languages.*

Proof. Let us start with a sequential grammar using right substitution rules

$$G = (N, T, P, S) \text{ where}$$

- N is a set of *nonterminal symbols*;
- T is a set of *terminal symbols*;
- P is a finite set of *right substitution rules* over V, where $V = N \cup T$;
- $S \in V^+$ is the *axiom*.

Now let $n := max\{|uv| \mid S_R(u, v) \in P)\}$. Moreover, let A_0 be the set of all terminal strings of lengths k, $0 \leq k \leq 2n$ that can be derived in G, and A be the set of all strings of lengths k with $n \leq k \leq 2n$ which can be derived in G. The language generated by G then is the union of the (terminal) strings in A_0 and the languages generated by the sequential grammars $G(A') = (N, T, P(A'), A')$ for $A' \in A$. In $P(A')$ we will only allow the substitutions in P which do not decrease the lengths of sentential forms any more:

Those strings in the language generated by G of lengths at most $2n$ are already contained in A_0, therefore we only need to aim at terminal strings of lengths bigger than $2n$.

Starting from a string w with u at the end, $|u| = n$, there can only be a finite number of derivations from w, not decreasing the length, but increasing the length by at most n symbols, which can be captured by right substitution

rules $S_R(u,v)$ with $|u| \leq |v| \leq |u|+n$. For all possible u, we now collect all the possible right substitution rules fulfilling these conditions, which in sum yields $P(A')$.

We remark that this part of the proof is not constructive – we are only interested in the result itself.

For every such system $G(A') = (N, T, P(A'), A')$ we now can easily construct an extended regular grammar $G'(A') = (N', T, P'(A'), A')$ with extended regular rules of the forms $A \to wC$ and $A \to w$, $w \in T^*$, $A, C \in N' \cup T$.

The nonterminals in $N' \setminus \{S\}$ are of the form $[X]$ where $X \in (N \cup T)^n$. We start with the rule

$$S \to A'(1) \dots A'(|A'|-n)[A'(|A'|-n+1) \dots A'(|A'|)]$$

in $P'(A')$. Observe that $A'(1) \dots A'(|A'|-n)$ must be a terminal string, as otherwise nonterminals there will remain forever, hence, $L(G'(A')) = \emptyset$. If A' only consists of terminal symbols, we also take $S \to A'(1) \dots A'(|A'|)$ into $P'(A')$.

Now let $S_R(u,v)$ be a rule in $P(A')$ with $|u| = n$ and $|u| \leq |v| \leq |u|+n$:

- If $|u| = |v|$, then we take the rule $[u] \to [v]$ into $P'(A')$.
- If $|u| = |v|$ and v is a terminal string, then we also take the rule $[u] \to v$ into $P'(A')$.
- If $|u| < |v|$, we take the rule $[u] \to v(1) \dots v(|v|-n)[v(|v|-n+1) \dots v(|v|)]$ into $P'(A')$, but only if $v(1) \dots v(|v|-n)$ is a terminal string.
- If $|u| < |v|$ and v is a terminal string, then we also take the rule $[u] \to v$ into $P'(A')$.

The extended regular grammar $G'(A') = (N', T, P'(A'), S)$ now exactly simulates all possible terminal derivations in $G(A') = (N, T, P(A'), A')$, which observation completes the proof. □

As the family of regular languages are closed under union, putting together all the lemmas mentioned above, we obtain the following result:

Theorem 3. *String systems of either size* 1 *or degree* 1 *using only right insertion and deletion rules of the forms* $aI_R(b)$ *and* $D_R(b)$ *cannot generate more than regular languages.*

In sum, the main complexity question left open is the characterization of the languages which can be generated by string systems of size 2 and degree 2, homogenous or not. A thorough investigation of such systems will be given in an extended version of this paper.

4 Conclusion

In this paper we have considered the concept of applying prescribed teams of rules to a bounded number of initially given objects, with the rules to be applied

either in parallel to different objects or sequentially to these objects. Each rule in a team has to be applied exactly once with a successful application of a team.

When using prescribed teams for string objects either two initial strings and teams with three rules or else three initial strings and teams with two rules are sufficient to obtain computational completeness. As string operations we use very simple insertion and deletion rules, i.e., inserting one object in the left context of another symbol or the deletion of a symbol on the right-hand side of a string. It remains an open question for future research how computational completeness can be obtained with even less ingredients, i.e., with only two initial strings and teams with two rules.

Moreover, we have shown that systems with either only one initial string or else with teams of only one rule can generate only regular languages.

We have also considered the insertion of anti-symbols which annihilate the corresponding symbols instead of deleting a symbol.

Similar results can be obtained when using these operations of insertion and deletion on the left-hand sides of the strings.

Acknowledgements

The authors gratefully thank the three referees for their useful comments.

Artiom Alhazov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

References

1. Csuhaj-Varjú, E., Dassow, J., Kelemen, J.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Topics in computer mathematics, Gordon and Breach (1994)
2. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer (1989)
3. Fernau, H., Kuppusamy, L., Raman, I.: Investigations on the power of matrix insertion-deletion systems with small sizes. Natural Computing **17**(2), 249–269 (2018). https://doi.org/10.1007/s11047-017-9656-8
4. Fernau, H., Kuppusamy, L., Raman, I.: On the computational completeness of matrix simple semi-conditional grammars. Information and Computation **284**, 104688 (2022). https://doi.org/10.1016/j.ic.2021.104688
5. Fernau, H., Kuppusamy, L., Verlan, S.: Universal matrix insertion grammars with small size. In: Patitz, M.J., Stannett, M. (eds.) Unconventional Computation and Natural Computation – 16th International Conference, UCNC 2017, Fayetteville, AR, USA, June 5–9, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10240, pp. 182–193. Springer (2017). https://doi.org/10.1007/978-3-319-58187-3_14
6. Petre, I., Verlan, S.: Matrix insertion-deletion systems. Theoretical Computer Science **456**, 80–88 (2012). https://doi.org/10.1016/j.tcs.2012.07.002
7. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997). https://doi.org/10.1007/978-3-642-59136-5