RAIRO-Theor. Inf. Appl. 46 (2012) 593–613
DOI: 10.1051/ita/2012020

Available online at:
www.rairo-ita.org

STRING ASSEMBLING SYSTEMS

Martin Kutrib[1] and Matthias Wendlandt[1]

Abstract. We introduce and investigate string assembling systems which form a computational model that generates strings from copies out of a finite set of assembly units. The underlying mechanism is based on piecewise assembly of a double-stranded sequence of symbols, where the upper and lower strand have to match. The generation is additionally controlled by the requirement that the first symbol of a unit has to be the same as the last symbol of the strand generated so far, as well as by the distinction of assembly units that may appear at the beginning, during, and at the end of the assembling process. We start to explore the generative capacity of string assembling systems. In particular, we prove that any such system can be simulated by some nondeterministic one-way two-head finite automaton, while the stateless version of the two-head finite automaton marks to some extent a lower bound for the generative capacity. Moreover, we obtain several incomparability and undecidability results as well as (non-)closure properties, and present questions for further investigations.

Mathematics Subject Classification. 68Q05, 68Q42.

1. Introduction

The vast majority of computational models in connection with language theory processes or generates words, that is, strings of symbols out of a finite set. The possibilities to control the computation naturally depend on the devices in question. Over the years lots of interesting systems have been investigated. With the advent of investigations of devices and operations that are inspired by the study of biological processes, and the growing interest in nature-based problems

Keywords and phrases. String assembling, double-stranded sequences, stateless, two-head finite automata, decidability, closure properties.

[1] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany.
{kutrib,matthias.wendlandt}@informatik.uni-giessen.de

modeled in formal systems, a very old control mechanism has been rekindled. If the raw material that is processed or generated by computational models is double stranded in such a way that corresponding symbols are uniquely related (have to be identical, for example), then the correctness of the complementation of a strand is naturally given. The famous Post's Correspondence Problem can be seen as a first study showing the power of double-stranded string generation. That is, a list of pairs of substrings $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ is used to generate synchronously a double-stranded string, where the upper and lower string have to match. More precisely, a string is said to be generated if and only if there is a nonempty finite sequence of indices $i_1, i_2, \ldots, i_k$ such that $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$. It is well-known that it is undecidable whether a PCP generates a string at all [11]. A more recent approach are sticker systems [1,6,10], where basically the pairs of substrings may be connected to form pieces that have to fit to the already generated part of the double strand. In addition, for variants the pieces may be added from left as well as from right. So, the generation process is subject to control mechanisms and restrictions given, for example, by the shape of the pieces.

Here we consider string assembling systems that are also double-stranded string generation systems. As for Post's Correspondence Problem the basic assembly units are pairs of substrings that have to be connected to the upper and lower string generated so far synchronously. The substrings are not connected as may be for sticker systems. However, we have two further control mechanisms. First, we require that the first symbol of a substring has to be the same as the last symbol of the strand to which it is connected. One can imagine that both symbols are glued together one at the top of the other and, thus, just one appears in the final string. Second, as for the notion of strictly locally testable languages [7,15] we distinguish between assembly units that may appear at the beginning, during, and at the end of the assembling process.

The paper is organized as follows: the next section contains preliminaries and the definition of string assembling systems as well as some meaningful examples that are a starting point to explore the generative capacity of the systems. Then Section 3 deals with an upper and lower bound for the generative capacity. It is shown that any string assembling system can be simulated by some nondeterministic one-way two-head finite automaton, while the stateless version of the two-head finite automaton marks to some extent a lower bound for the generative capacity. More precisely, up to at most four additional symbols in the strings generated, any stateless nondeterministic one-way two-head finite automaton can be simulated by some string assembling system. Several incomparability results are derived. In particular, we obtain incomparability with (deterministic) (linear) context-free languages, regular languages, and sticker system languages. Though not all regular languages belong to the family of languages generated by string assembling systems, any regular language can be represented by such a system and a weak homomorphism. The main subject of Section 4 are closure properties of the family of languages generated by string assembling systems. In particular, the non-closure under five of the six AFL operations is shown, where the remaining

one, the iteration, is an open problem. Furthermore we obtain non-closure under complementation. The only positive closure property is for reversal. In Section 5 we investigate several decidability problems. It turns out that emptiness, finiteness, inclusion, regularity, and context-freeness are all undecidable. In Section 6 we finally present untouched or unanswered questions which may be interesting and fruitful for further research.

2. Preliminaries and definitions

We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. For the number of occurrences of a symbol a in w we use the notation $|w|_a$. Generally, for a singleton set $\{a\}$ we simply write a. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, that is, $L - \{\lambda\} = L' - \{\lambda\}$.

As mentioned before, a string assembling system generates a double-stranded string by assembling units. Each unit consists of two substrings, the first one is connected to the upper and the second one to the lower strand. The corresponding symbols of the upper and lower strand have to be equal. Moreover, a unit can only be assembled when the first symbols of its substrings match the last symbols of their strands. In this case the matching symbols are glued together on at the top of the other. The generation has to begin with a unit from the set of initial units. Then it may continue with units from a different set. When a unit from a third set of ending units is applied the process necessarily stops. The generation is said to be valid if and only if both strands are identical when the process stops. More precisely:

Definition 2.1. A *string assembling system* (SAS) is a quadruple $\langle \Sigma, A, T, E \rangle$, where

1. Σ is the finite, nonempty set of *symbols* or *letters*;
2. $A \subset \Sigma^+ \times \Sigma^+$ is the finite set of *axioms* of the forms (uv, u) or (u, uv), where $u \in \Sigma^+$ and $v \in \Sigma^*$;
3. $T \subset \Sigma^+ \times \Sigma^+$ is the finite set of *assembly units*; and
4. $E \subset \Sigma^+ \times \Sigma^+$ is the finite set of *ending assembly units* of the forms (vu, u) or (u, vu), where $u \in \Sigma^+$ and $v \in \Sigma^*$.

The next definition formally says how the units are assembled.

Definition 2.2. Let $S = \langle \Sigma, A, T, E \rangle$ be an SAS. The *derivation relation* $\Rightarrow$ is defined on specific subsets of $\Sigma^+ \times \Sigma^+$ by

1. $(uv, u) \Rightarrow (uvx, uy)$ if
   (i) $uv = ta$, $u = sb$, and $(ax, by) \in T \cup E$, for $a, b \in \Sigma$, $x, y, s, t \in \Sigma^*$; and
   (ii) $vx = yz$ or $vxz = y$, for $z \in \Sigma^*$.

2. $(u, uv) \Rightarrow (uy, uvx)$ if
   (i) $uv = ta$, $u = sb$, and $(by, ax) \in T \cup E$, for $a, b \in \Sigma$, $x, y, s, t \in \Sigma^*$; and
   (ii) $vx = yz$ or $vxz = y$, for $z \in \Sigma^*$.

A derivation is said to be *successful* if it initially starts with an axiom from A, continues with assembling units from T, and ends with assembling an ending unit from E. The process necessarily stops when an ending assembly unit is added. The sets A, T, and E are not necessarily disjoint.

The *language* $L(S)$ *generated* by S is defined to be the set

$$L(S) = \{\, w \in \Sigma^+ \mid (p, q) \Rightarrow^* (w, w) \text{ is a successful derivation} \,\},$$

where $\Rightarrow^*$ refers to the reflexive, transitive closure of the derivation relation $\Rightarrow$.

In order to clarify our notation we give three meaningful examples.

Example 2.3. The following SAS $S = \langle \{a, b, c\}, A, T, E \rangle$ generates the non-context-free language $\{\, a^n b^n c^n \mid n \geq 1 \,\}$.

$$\begin{aligned} &A = \{(a, a)\}, \quad T = T_a \cup T_b \cup T_c, \quad E = \{(c, c)\}, \text{ where} \\ &T_a = \{(aa, a), (ab, a)\}, \quad T_b = \{(bb, aa), (bc, ab)\}, \\ &T_c = \{(cc, bb), (c, bc), (c, cc)\}. \end{aligned}$$

The units in T_a are used to generate the prefixes $a^n b$. Initially, only the unit (aa, a) is applicable repeatedly. Then only (ab, a) can be used to generate the upper string $a^n b$ and the lower string a. After that the unit (bb, aa) from T_b has to be used exactly as many times as the unit (aa, a) has been applied before. Then an application of unit (bc, ab) is the sole possibility. This generates the upper string $a^n b^n c$ and the lower string $a^n b$. For the last part the units from T_c are used. Similarly as before, repeated applications of (cc, bb) yield to the upper string $a^n b^n c^n$ and the lower string $a^n b^n$. So, it remains to complement the c's in the lower string. This is done by the units (c, bc), which can be applied only once, and (c, cc) which can be applied arbitrarily often. However, the derivation is successful only if the number of c's in the upper and lower string match when the unit from E is applied.

The construction of Example 2.3 can be extended to an arbitrary number of symbols.

Corollary 2.4. *Let $k \geq 1$ be a constant and $\Sigma = \{a_1, a_2, \ldots, a_k\}$ be an alphabet. Then the language $\{\, a_1^n a_2^n \ldots a_k^n \mid n \geq 1 \,\}$ is generated by an SAS.*

The next example uses the same mechanism as the previous one, but extends the substrings generated. The mechanism is to copy parts of the string generated so far by using markers and the matching of the upper and lower strings.

Example 2.5. Let Σ be an alphabet not containing the symbols $\{\$_1, \$_2, \$_3\}$. The following SAS $\langle \Sigma \cup \{\$_1, \$_2, \$_3\}, A, T, E\rangle$ generates the non-context-free language $\{\, \$_1 w \$_2 w \$_3 \mid w \in \Sigma^+ \,\}$.

$$A = \{(\$_1, \$_1)\}, \quad T = T_1 \cup T_2 \cup T_3, \quad E = \{(\$_3, \$_3)\}, \text{ where for } x, y \in \Sigma,$$
$$T_1 = \{(\$_1 x, \$_1), (xy, \$_1), (x\$_2, \$_1)\}, \quad T_2 = \{(\$_2 x, \$_1 x), (xy, xy), (x\$_3, x\$_2)\},$$
$$T_3 = \{(\$_3, \$_2 x), (\$_3, xy), (\$_3, x\$_3)\}.$$

Similar as in Example 2.3, the units from T_1 are used to generate the upper string $\$_1 w \$_2$ and the lower string $\$_1$. Then units from T_2 are assembled to form the upper string $\$_1 w \$_2 w \$_3$ and the lower string $\$_1 w \$_2$. Finally, one obtains $\$_1 w \$_2 w \$_3$ as upper and lower string by the units in T_3.

The construction of Example 2.5 can be extended to an arbitrary number of copies of w.

Corollary 2.6. *Let $k \geq 1$ be a constant and Σ be an alphabet not containing the symbols $\{\$_1, \$_2, \ldots, \$_k\}$. Then the language $\{\, \$_1 w \$_2 w \ldots \$_{k-1} w \$_k \mid w \in \Sigma^+ \,\}$ is generated by an SAS.*

Now we turn to a basic example where the underlying technique is to utilize the lengths difference between the upper and lower string.

Example 2.7. Let $k \geq 2$ be a constant. The following SAS $\langle \{a, b\}, A, T, E\rangle$ generates the language $\{\, bw \mid w \in \{a, b\}^*, |w|_a \bmod k = 0 \,\}$.

$$A = \{(b, b)\}, \quad E = \{\, (x, x) \mid x \in \{a, b\} \,\},$$
$$T = \{(xa, x), (xb, yz), (a, u), (b, u) \mid x, y, z \in \{a, b\}, u \in \{a, b\}^k \,\}.$$

The idea of the construction is to start with a b and then to add arbitrary symbols to the upper string. Whenever an a is added, the difference between the lengths of the upper and lower string is increased by one, while it remains as it is when a b is added. The only possibility to complement the lower string is to add substrings of length k. Therefore, in any situation, the difference between the lengths modulo k is equal to the number of a's in the upper string modulo k. So, if both strings are identical, they do belong to the language as stated.

3. Generative capacity

In order to explore the generative capacity of SAS we start with an upper bound. In particular, we show that any SAS can be simulated by some nondeterministic one-way two-head finite automaton. The family of languages accepted by such devices is known to be a proper subfamily of languages accepted by nondeterministic *two-way* two-head finite automata (see [3], for example). From the complexity point of view, the two-way case is well explored. There is a strong relation to the

computational complexity class $\mathsf{NL} = \mathsf{NSPACE}(\log n)$. In [2] it has been shown that NL is characterized by the class of nondeterministic two-way multi-head finite automata. So, together with the next theorem, we obtain that the family of languages generated by SAS is properly included in NL.

A one-way two-head finite automaton is a finite automaton having a single read-only input tape whose inscription is the input word in between two endmarkers. The two heads of the automaton can move to the right or stay on the current tape square. Denote such a device as $\langle S, \Sigma, \delta, \rhd, \lhd, s_0, F\rangle$, where S is the finite set of *internal states*, Σ is the set of *input symbols*, $\rhd \notin \Sigma$ and $\lhd \notin \Sigma$ are the *left and right endmarkers*, $s_0 \in S$ is the *initial state*, $F \subseteq S$ is the set of *accepting states*, and δ is the partial transition function mapping $S \times (\Sigma \cup \{\rhd, \lhd\})^2$ into the subsets of $S \times \{0,1\}^2$, where 1 means to move the head one square to the right and 0 means to keep the head on the current square. Whenever $(s', d_1, d_2) \in \delta(s, a_1, a_2)$ is defined, then $d_i = 0$ if $a_i = \lhd$, for $i \in \{1, 2\}$.

A one-way two-head finite automaton starts with both heads on the left endmarker. It halts when the transition function is not defined for the current situation. The input is accepted if and only if the computation halts in an accepting state.

Theorem 3.1. *Let $S = \langle \Sigma, A, T, E\rangle$ be an SAS. There exists a nondeterministic one-way two-head finite automaton M that accepts $L(S)$.*

Proof. Basically, the idea of the construction of M is to guess dependent on the currently scanned input symbols (the current overlappings) which assembly unit comes next. Then the guess is verified by reading the upper strand with the first and the lower strand with the second head. The last symbol is the new overlapping and, thus, the heads stay on it for the new guess. After each verification, M determines whether the assembling process is completed and guesses another unit to be assembled otherwise. At the beginning one of the axioms is guessed and verified as the ordinary units. In detail, for any $(x_1^{(i)} x_2^{(i)} \dots x_k^{(i)}, y_1^{(i)} y_2^{(i)} \dots y_\ell^{(i)}) \in A$, where $x_j^{(i)} \in \Sigma$ for $1 \le j \le k$, $y_j^{(i)} \in \Sigma$ for $1 \le j \le \ell$, we set

$$\left(p_1^{(i)}, 1, 1\right) \in \delta(s_0, \rhd, \rhd)$$

where $p_1^{(i)}$ is a new state. This implements the guessing of the axiom. Further, for any $(x_1^{(i)} x_2^{(i)} \dots x_k^{(i)}, y_1^{(i)} y_2^{(i)} \dots y_\ell^{(i)}) \in A \cup T \cup E$ we provide a new state $p_1^{(i)}$, and

$$\left(p_{n+1}^{(i)}, 1, 0\right) \in \delta\left(p_n^{(i)}, x_n^{(i)}, y_1^{(i)}\right)$$
$$\left(q_1^{(i)}, 0, 0\right) \in \delta\left(p_k^{(i)}, x_k^{(i)}, y_1^{(i)}\right)$$

for $1 \le n < k$ and new states $p_{n+1}^{(i)}$ and $q_1^{(i)}$, implements the verification of the upper strand. Similarly,

$$\left(q_{n+1}^{(i)}, 0, 1\right) \in \delta\left(q_n^{(i)}, x_k^{(i)}, y_n^{(i)}\right)$$

for $1 \leq n < \ell$ and new states $q_{n+1}^{(i)}$, implements the verification of the lower strand. Finally, the automaton M guesses whether another unit, say unit j, is to be assembled by

$$\left(p_1^{(j)}, 0, 0\right) \in \delta\left(q_\ell^{(i)}, x_k^{(i)}, y_\ell^{(i)}\right),$$

where $x_k^{(i)} = x_1^{(j)}$ and $y_\ell^{(i)} = y_1^{(j)}$ for some $(x_1^{(j)} x_2^{(j)} \dots x_{k'}^{(j)}, y_1^{(j)} y_2^{(j)} \dots y_{\ell'}^{(j)}) \in T \cup E$, or whether the assembling process is completed by

$$(s_t, 1, 1) \in \delta\left(q_\ell^{(i)}, x_k^{(i)}, y_\ell^{(i)}\right)$$
$$\{(s_a, 0, 0)\} = \delta(s_t, \lhd, \lhd)$$

for all $(x_1^{(i)} x_2^{(i)} \dots x_k^{(i)}, y_1^{(i)} y_2^{(i)} \dots y_\ell^{(i)}) \in E$, where s_a and s_t are new states, and s_a is the only accepting state. □

The previous theorem and its preceding discussion together with the proper inclusion of NL in NSPACE(n) (see, for example, [9]), which in turn is equal to the family of context-sensitive languages, reveals the following corollary.

Corollary 3.2. *The family of languages generated by SAS is properly included in* NL *and, thus, in the family of context-sensitive languages.*

Combining Theorem 3.1 and Example 2.3 we obtain the following relations to context-free languages.

Lemma 3.3. *The family of languages generated by SAS is incomparable with the family of (deterministic) (linear) context-free languages.*

Proof. By Example 2.3 the non-context-free language $\{ a^n b^n c^n \mid n \geq 1 \}$ does belong to the family of languages generated by SAS. Conversely, by Theorem 3.1 any SAS can be simulated by a nondeterministic one-way two-head finite automaton. It is well known that the latter cannot accept the deterministic and linear context-free language $\{ wcw^R \mid w \in \{a, b\}^* \}$. □

Although the basic mechanisms of sticker systems and string assembling systems seem to be closely related, their generative capacities differ essentially. While the copy language $\{ \$_1 w \$_2 w \$_3 \mid w \in \Sigma^+ \}$ of Example 2.5 is generated by an SAS, it is not generated by any sticker system. So, one could say, SAS can copy while sticker systems cannot.

Conversely, some variant of the mirror language $\{ w \mid w \in \{a, b\}^* \text{ and } w = w^R \}$ is generated by many variants of sticker systems (that can generate all linear context-free languages), but cannot be generated by any SAS, since it cannot be accepted by any nondeterministic one-way two-head finite automaton. So, sticker systems can handle mirrored inputs while SAS cannot.

Following the discussion preceding Theorem 3.1, the simulation of SAS by nondeterministic one-way two-head finite automata gives only a rough upper bound for the generative capacity of SAS. Interestingly, the stateless version of

the two-head finite automaton marks to some extent a lower bound for the generative capacity. More precisely, up to at most four additional symbols in the words generated, any stateless nondeterministic one-way two-head finite automaton can be simulated by some SAS. Actually, such a stateless automaton is a one-state device so that the transition function maps the two input symbols currently scanned to the head movements. Therefore, the automaton cannot accept by final state. It rejects if any of the heads falls off the tape or if the transition function is not defined for the current situation. If the transition function instructs both heads to stay, the automaton halts and accepts [5,13].

Theorem 3.4. *Let $M = \langle \{s\}, \Sigma, \delta, \rhd, \lhd, s, \emptyset \rangle$ be a stateless nondeterministic one-way two-head finite automaton and $\$, \#, ?, ! \notin \Sigma$. There exists a string assembling system S such that any word generated by S contains each of the symbols $\$, \#, ?, !$ at most once, and $h(L(S)) = L(M)$, for the homomorphism $h(\$) = h(\#) = h(?) = h(!) = \lambda$ and $h(a) = a$, for $a \in \Sigma$.*

Proof. The underlying idea of the construction of S is to guess and assemble the next symbol to the corresponding strand whenever a head moves to the right. The guesses are subsequently verified by simulating transitions of M. Special care has to be taken for the situation where M accepts. Moreover, by inspecting the transition function we can check whether $(0,0) \in \delta(\rhd, \rhd)$. If so, language $L(M)$ is equal to Σ^*. In this case we obtain an SAS immediately. So, in the following we assume $(0,0) \notin \delta(\rhd, \rhd)$.

We start the construction with the units from A that implement the guessing of the first symbol.

$$A = \{ (x, x) \mid x \in \Sigma \}.$$

The transitions of M are written in its short form, that is, we omit the unique state. Next, we turn to the construction of T. Let $a, b \in \Sigma$ be some symbols. Then we define all following units for all $x, y \in \Sigma$.

(1) $(x, ay) \in T$ if $(0,1) \in \delta(\rhd, a)$
(2) $(x, a) \in T$ if $(1,0) \in \delta(\rhd, a)$
(3) $(x, ay) \in T$ if $(1,1) \in \delta(\rhd, a)$
(4) $(a, y) \in T$ if $(0,1) \in \delta(a, \rhd)$
(5) $(ax, y) \in T$ if $(1,0) \in \delta(a, \rhd)$
(6) $(ax, y) \in T$ if $(1,1) \in \delta(a, \rhd)$.

The units from (1) to (6) simulate the behavior of M as long as one of its heads stays on the left endmarker. After having guessed the first symbol by a unit from A, now a new symbol x or y is guessed and assembled whenever a head moves, while the previously guessed symbol a is verified by the simulation of a step of M. As long as one head stays on the left endmarker, it has not verified the first symbol of its strand and, so, for any possible first symbol a unit has to be provided (see x in units (1)–(3) and y in units (4)–(6)).

Once both heads have been moved off the left endmarker, symbols from both strands are guessed and verified by simulations of steps of M as follows.

(7) $(a, by) \in T$ if $(0,1) \in \delta(a,b)$
(8) $(ax, b) \in T$ if $(1,0) \in \delta(a,b)$
(9) $(ax, by) \in T$ if $(1,1) \in \delta(a,b)$.

So far, the SAS generates its strands exactly as M would read them. Next we turn to the accepting transitions of M. Here it may happen that M accepts without reading the whole input. In this case, an arbitrary string can still follow. We concatenate it with the help of the symbols `$` and `#`.

(10) $(x, a\$) \in T$ if $(0,0) \in \delta(\triangleright, a)$
(11) $(a\$, y) \in T$ if $(0,0) \in \delta(a, \triangleright)$
(12) $(a\$, b) \in T$ if $(0,0) \in \delta(a,b)$
(13) $(a, b\$) \in T$ if $(0,0) \in \delta(a,b)$.

The `$` marks the rightmost position of a head when M accepts. Note that at most one of the units (12) or (13) is applicable, dependent on which head is in front.

Next, the other strand is complemented with the help of the `$`. Then a symbol `#` is assembled with whose help an arbitrary string is concatenated to the lower strand.

(14) $(\$, xy) \in T$
(15) $(xy, \$) \in T$
(16) $(\$\#, x\$\#) \in T$
(17) $(x\$\#, \$\#) \in T$
(18) $(\#, \#y) \in T$
(19) $(\#, xy) \in T$.

When the process of concatenating symbols is completed, a symbol `!` is assembled that is used to complement the upper strand.

(20) $(\#, \#!) \in T$
(21) $(\#, x!) \in T$
(22) $(\#x, !) \in T$
(23) $(\#!, !) \in T$
(24) $(xy, !) \in T$
(25) $(!, xy) \in T$
(26) $(x!, !) \in T$
(27) $(!, y!) \in T$.

Units (20) and (21) finish the process of concatenating symbols. The units (22)–(24) and (26) are used to complement the upper strand. Units (25) and (27) can be used to complement the lower strand. They are needed in the following.

Before we turn to the construction of the set E, we consider situations where at least one head of M reaches the right endmarker. In this case the process of concatenating symbols is also finished but the simulations must be continued until M finally accepts or rejects. We continue the simulation with the help of the symbol `?`.

(28) $(ax, y?) \in T$ if $(1,0) \in \delta(a, \triangleleft)$
(29) $(a?, y?) \in T$ if $(1,0) \in \delta(a, \triangleleft)$
(30) $(ax, ?) \in T$ if $(1,0) \in \delta(a, \triangleleft)$
(31) $(a?, ?) \in T$ if $(1,0) \in \delta(a, \triangleleft)$
(32) $(x?, ay) \in T$ if $(0,1) \in \delta(\triangleleft, a)$
(33) $(x?, a?) \in T$ if $(0,1) \in \delta(\triangleleft, a)$
(34) $(?, ay) \in T$ if $(0,1) \in \delta(\triangleleft, a)$
(35) $(?, a?) \in T$ if $(0,1) \in \delta(\triangleleft, a)$.

Units (28) and (30) as well as (32) and (34) are used to add the symbol ? by guessing, and to continue the simulation. Units (29) and (31) as well as (33) and (35) are used to guess the completion of the complementation.

We consider the remaining accepting situations of M.

(36) $(ax, y!) \in T$ if $(0,0) \in \delta(a, \lhd)$
(37) $(a!, y!) \in T$ if $(0,0) \in \delta(a, \lhd)$
(38) $(ax, ?!) \in T$ if $(0,0) \in \delta(a, \lhd)$
(39) $(a?!, ?!) \in T$ if $(0,0) \in \delta(a, \lhd)$
(40) $(x, y!) \in T$ if $(0,0) \in \delta(\rhd, \lhd)$
(41) $(x!, ay) \in T$ if $(0,0) \in \delta(\lhd, a)$
(42) $(x!, a!) \in T$ if $(0,0) \in \delta(\lhd, a)$
(43) $(?!, ax) \in T$ if $(0,0) \in \delta(\lhd, a)$
(44) $(?!, a?!) \in T$ if $(0,0) \in \delta(\lhd, a)$
(45) $(x!, y) \in T$ if $(0,0) \in \delta(\lhd, \rhd)$
(46) $(?!, ?!) \in T$ if $(0,0) \in \delta(\lhd, \lhd)$.

When M accepts with exactly one head on the right endmarker a symbol ! is used to complement the remaining strand. When this symbol is added, M has to distinguish whether the other head reaches the right endmarker in the next step ((37), (39), (42), (44)), whether the other head is still on the left endmarker ((40), (45)), or whether the other head is not at an endmarker ((36), (38), (41), (43)). Unit (46) is used when both heads arrive at the right endmarker at the same time.

Finally, we define the set E by $E = \{(!, !)\}$. Only by using this unit the generation of a word by S is successful. If and only if in the preceding process a symbol ! is assembled, M has accepted. □

Stateless multi-head finite automata are studied in detail in [5, 13]. Though it is an open problem whether the additional symbols used in the simulation of the previous proof are necessary, there is a language generated by SAS which is not accepted by any stateless nondeterministic one-way two-head finite automaton.

Lemma 3.5. *The language* $\{ a^{2n} \mid n \geq 1 \}$ *is generated by an SAS but not accepted by any stateless nondeterministic one-way two-head finite automaton.*

Since the family of languages generated by SAS is properly included in the context-sensitive languages and incomparable with (deterministic) (linear) context-free languages, it is natural to compare it with regular languages, too.

Theorem 3.6. *The family of languages generated by SAS is incomparable with the family of (unary) regular languages.*

Proof. By the previous results it remains to be shown that there is a unary regular language not generated by any SAS. To this end, we consider the language $L = \{a\} \cup \{ a^{2n} \mid n \geq 2 \}$, and assume that it is generated by some SAS $\langle \{a\}, A, T, E \rangle$.

Since $a \in L$, the unit (a, a) must belong to A as well as to E. Now we define the three sets $T_e = \{ (a^i, a^i) \in T \mid i \geq 2 \}$, $T_u = \{ (a^i, a^j) \in T \mid i > j \geq 1 \}$, and $T_l = \{ (a^i, a^j) \in T \mid 1 \leq i < j \}$.

If T_e is not empty, all its units are of the form (a^{4+2k}, a^{4+2k}), $k \geq 0$. Otherwise, starting with unit $(a, a) \in A$, assembling a unit from T_e not of this form, and

ending with unit $(a, a) \in E$ results in a string of length greater than one but not of length a^{4+2k}. But this string does not belong to L. Now, starting with unit $(a, a) \in A$, assembling a fixed unit (a^{4+2k_0}, a^{4+2k_0}) from T_e twice, and ending with unit $(a, a) \in E$ results in the string $a^{4+2k_0+4+2k_0-1}$ of odd length. But the only string in L of odd length is a. So, we conclude that T_e must be empty.

If T_u or T_l is empty in addition, either the lower or the upper string gets longer and longer compared with the other string. In this case only a finite language is generated, which is a contradiction. Therefore, T_u and T_l both are not empty. Fix a unit (a^{i_1}, a^{j_1}) from T_u and a unit (a^{i_2}, a^{j_2}) from T_l. Now, assembling $j_2 - i_2$ units (a^{i_1}, a^{j_1}) and $i_1 - j_1$ units (a^{i_2}, a^{j_2}) extends the upper as well as the lower string by $i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1$ symbols. So, the effect is as assembling a unit $(a^{i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1 + 1}, a^{i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1 + 1})$ from T_e. We conclude that T_u and T_l both have to be empty, and obtain the same contradiction as above. □

The previous theorem shows that SAS cannot even generate all unary regular languages. However, all finite regular languages L can be generated.

Proposition 3.7. *Every finite regular language is generated by an SAS.*

Proof. Given a finite regular language, an SAS generating L is constructed as follows. For all $w \in L$ a unit (w, w) is added to A, T is set to be empty, and $E = \{ (x, x) \mid x \in \Sigma \}$. □

Though not all regular languages belong to the family of languages generated by SAS, all regular languages can be represented by an SAS and a weak homomorphism.

Theorem 3.8. *Let L be a regular language. There is an SAS S and a λ-free letter-to-letter homomorphism h such that $L = h(L(S))$.*

Proof. Let a regular language $L \subseteq \Sigma^*$ be given by a complete deterministic finite automaton with state set $Q = \{q_1, q_2, \ldots, q_n\}$, initial state q_1, set of accepting states F, and transition function δ. Let $\Sigma' = \{ a' \mid a \in \Sigma \}$ be a copy of the input alphabet. A λ-free letter-to-letter homomorphism $h : (\Sigma \cup \Sigma')^* \to \Sigma$ is defined as $h(a) = h(a') = a$, for all $a \in \Sigma$.

Next we construct an SAS $S = \langle \Sigma \cup \Sigma', A, T, E \rangle$ such that $L = h(L(S))$. Basically, the idea of the construction is to extend the upper strings by blocks of length n whose last symbol is primed. The current state of the simulated DFA is encoded by the length difference between the upper and the shorter lower string.

We start by constructing the units of set A for all strings having at least length n. Let $a_n a_{n-1} \ldots a_2 a_1 \in \Sigma^n$ and $\delta(q_1, a_n a_{n-1} \ldots a_2 a_1) = q_i$. Then the unit

$$(a_n a_{n-1} \ldots a_2 a_1', a_n a_{n-1} \ldots a_{i+1})$$

is included in A. Note that only the last symbol of the overlapping of the upper string is primed and that the length of the overlapping is $1 \leq i \leq n$, that is, the index of the state the DFA is in after initially processing the input

string $a_n a_{n-1} \dots a_2 a_1$. For all strings having a length of at most n that do belong to L, we proceed as follows. Let $a_k a_{k-1} \dots a_2 a_1 \in \Sigma^k$, $1 \leq k \leq n$ and $\delta(q_1, a_k a_{k-1} \dots a_2 a_1) \in F$. Then the unit

$$(a_k a_{k-1} \dots a_2 a'_1, a_k a_{k-1} \dots a_2 a'_1)$$

is included in A. Note that in this case there is no overlapping.

Next, the units of set T are constructed. Let $a_n a_{n-1} \dots a_2 a_1 \in \Sigma^n$, $a_{n+1} \in \Sigma$, $q_i \in Q$, $b_i b_{i-1} \dots b_2 \in \Sigma^{i-1}$, and $b_{i+1} \in \Sigma$. Then the unit

$$(a'_{n+1} a_n a_{n-1} \dots a_2 a'_1, b_{i+1} b_i b_{i-1} \dots b_3 b_2 a'_{n+1} a_n a_{n-1} \dots a_{j+1})$$

is included in T if $i < n$ and $\delta(q_i, a_n a_{n-1} \dots a_2 a_1) = q_j$, and

$$(a'_{n+1} a_n a_{n-1} \dots a_2 a'_1, b'_{i+1} b_i b_{i-1} \dots b_3 b_2 a'_{n+1} a_n a_{n-1} \dots a_{j+1})$$

if $i = n$ and $\delta(q_i, a_n a_{n-1} \dots a_2 a_1) = q_j$. Since in any situation the overlapping includes exactly one primed symbol, by aligning it is ensured that only such units can be assembled that match the currently encoded state of the DFA.

In addition, units are provided that remove the overlapping when an accepting state can be reached. Let $a_k a_{k-1} \dots a_2 a_1 \in \Sigma^k$, $1 \leq k \leq n$, $a_{k+1} \in \Sigma$, $q_i \in Q$, $b_i b_{i-1} \dots b_2 \in \Sigma^{i-1}$, and $b_{i+1} \in \Sigma$. Then the unit

$$(a'_{k+1} a_k a_{k-1} \dots a_2 a'_1, b_{i+1} b_i b_{i-1} \dots b_3 b_2 a'_{k+1} a_k a_{k-1} \dots a'_1)$$

is included in T if $i < n$ and $\delta(q_i, a_k a_{k-1} \dots a_2 a_1) \in F$, and

$$(a'_{k+1} a_k a_{k-1} \dots a_2 a'_1, b'_{i+1} b_i b_{i-1} \dots b_3 b_2 a'_{k+1} a_k a_{k-1} \dots a'_1)$$

if $i = n$ and $\delta(q_i, a_k a_{k-1} \dots a_2 a_1) \in F$. Note that no further unit from T can be assembled when there is no overlapping.

Finally, the set E terminates the generation successfully when there is no overlapping: $E = \{ (a', a') \mid a \in \Sigma \}$.

Altogether, computations of the given DFA are simulated whereby primed versions of input symbols are used to ensure that only correct units are assembled, where correct means with respect to the current state encoded. The homomorphism just removes the primes and, hence, $h(L(S)) = L$. □

4. Closure properties

Finally we consider closure properties of the family of languages generated by SAS. Closure under certain operations indicates a certain robustness of the language families, while non-closure properties may serve, for example, as a valuable basis for extensions. As it turns out the language family considered here is not closed under five of the six AFL operations, where the remaining one, the iteration, is an open problem. Furthermore we obtain non-closure under complementation. The only positive closure property is for reversal.

Lemma 4.1. *Let $L \subseteq \Sigma^*$ be a language generated by an SAS. If $|a^+ \cap L| = \infty$, for some symbol $a \in \Sigma$, then there exist constants $p, q \geq 1$ such that $a^p v \in L$, $v \in \Sigma^*$, implies $a^{p+q} v \in L$.*

Proof. Let $S = \langle \Sigma, A, T, E \rangle$ be an SAS generating L, $a \in \Sigma$, $|a^+ \cap L| = \infty$, and three sets defined by $T_e = \{ (a^i, a^i) \in T \mid i \geq 2 \}$, $T_u = \{ (a^i, a^j) \in T \mid i > j \geq 1 \}$, and $T_l = \{ (a^i, a^j) \in T \mid 1 \leq i < j \}$.

Since L includes infinitely many words from a^+, set T_e, or T_u and T_l both are non-empty. Otherwise only finitely many words from a^+ are generated. If T_e is not empty, choose a unit $(a^i, a^i) \in T_e$ and set $q = i - 1$. If, otherwise, T_u and T_l both are non-empty, choose a unit (a^{i_1}, a^{j_1}) from T_u and a unit (a^{i_2}, a^{j_2}) from T_l. Assembling $j_2 - i_2$ units (a^{i_1}, a^{j_1}) and $i_1 - j_1$ units (a^{i_2}, a^{j_2}) extends the upper as well as the lower string by $i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1$ symbols. In this case set $q = i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1$.

Set p to be the length of the longest string appearing in a unit from A. Then any derivation of a word $a^p v \in L$ has to start with a unit of the form $(a^+, a^+) \in A$. So, assembling the unit(s) that extend(s) the upper as well as the lower string by q symbols a immediately after the start, and letting the remaining derivation unchanged generates the string $a^{p+q} v$. □

The next example applies the lemma to show that a language does not belong to the family of languages generated by SAS.

Example 4.2. Let L be the language $\{ a^n b^n \mid n \geq 1 \} \cup a^+$, and assume L is generated by an SAS. Then Lemma 4.1 can be applied with constants $p, q \geq 1$. Since $a^p b^p \in L$ we derive $a^{p+q} b^p \in L$, a contradiction. So, L does not belong to the family of languages generated by SAS.

Next, we turn to the closure properties under Boolean operations. The non-closure under union can be seen by the previous example. The stronger result that the family of languages generated by SAS is not even closed under union with a symbol has already been shown in the Proof of Theorem 3.6, where it was proven that the language $\{a\} \cup \{ a^{2^n} \mid n \geq 2 \}$ is not generated by any SAS. Clearly, $\{a\}$ as well as $\{ a^{2^n} \mid n \geq 2 \}$ are generated by SAS.

Theorem 4.3. *The family of languages generated by SAS is not closed under union (with a symbol).*

In order to disprove the closure under complementation we can again apply Lemma 4.1 to an appropriate witness language.

Theorem 4.4. *The family of languages generated by SAS is not closed under complementation.*

Proof. By Corollary 2.4 the language $L = \{ a^n b^n \mid n \geq 1 \}$ is generated by an SAS. In contrast to the assertion assume its complement $\overline{L}$ also belongs to the family of languages generated by SAS. Since all words a^i, for $i \geq 1$, belong to $\overline{L}$, Lemma 4.1 can be applied with constants $p, q \geq 1$. So, $a^p b^{p+q} \in \overline{L}$ implies $a^{p+q} b^{p+q} \in \overline{L}$, a contradiction. □

The remaining Boolean operation is the intersection. Since the family of languages generated by SAS is incomparable with the family of regular languages, but includes the languages Σ^*, for any alphabet Σ, its non-closure under intersection with regular languages follows immediately:

Corollary 4.5. *The family of languages generated by SAS is not closed under intersection with regular languages.*

Furthermore, the family of languages in question is not closed under intersection.

Theorem 4.6. *The family of languages generated by SAS is not closed under intersection.*

Proof. The following two SAS are generalizations of example 2.5.

$S_1 = \langle \{a, b, \$_1, \$_2, \$_3, \$_4, \$_5\}, A, T_1, E\rangle$ generates the language

$$\{\, \$_1 w \$_2 u \$_3 v \$_4 w \$_5 \mid u, v, w \in \{a, b\}^+ \,\}.$$

$A = \{(\$_1, \$_1)\}, \quad E = \{(\$_5, \$_5)\},$
$T_1 = T_{1,1} \cup T_{1,2} \cup T_{1,3} \cup T_{1,4} \cup T_{1,5},$ where for $x, y \in \Sigma$,
$T_{1,1} = \{(\$_1 x, \$_1), (xy, \$_1), (x\$_2, \$_1)\},\ T_{1,2} = \{(\$_2 x, \$_1), (xy, \$_1), (x\$_3, \$_1)\},$
$T_{1,3} = \{(\$_3 x, \$_1), (xy, \$_1), (x\$_4, \$_1)\},\ T_{1,4} = \{(\$_4 x, \$_1 x), (xy, xy), (x\$_5, x\$_2)\},$
$T_{1,5} = \{(\$_5, \$_2 x), (\$_5, xy), (\$_5, x\$_3), (\$_5, \$_3 x), (\$_5, x\$_4), (\$_5, \$_4 x), (\$_5, x\$_5)\}.$

Duplicate units are included for readability. The units from $T_{1,1}$ are used to generate the upper string $\$_1 w \$_2$ and the lower string $\$_1$. Then the upper string is extended to $\$_1 w \$_2 u \$_3$ by the units from $T_{1,2}$, and further to $\$_1 w \$_2 u \$_3 v \$_4$ by the units from $T_{1,3}$. Then units from $T_{1,4}$ are assembled to form the upper string $\$_1 w \$_2 u \$_3 v \$_4 w \$_5$ and the lower string $\$_1 w \$_2$. Finally, one obtains $\$_1 w \$_2 u \$_3 v \$_4 w \$_5$ as upper and lower string by the units in $T_{1,5}$.

Following a similar principle, $S_2 = \langle \{a, b, \$_1, \$_2, \$_3, \$_4, \$_5\}, A, T_2, E\rangle$ generates the language

$$\{\, \$_1 u \$_2 w \$_3 w \$_4 v \$_5 \mid u, v, w \in \{a, b\}^+ \,\}.$$

$A = \{(\$_1, \$_1)\}, \quad E = \{(\$_5, \$_5)\},$
$T_2 = T_{2,1} \cup T_{2,2} \cup T_{2,3} \cup T_{2,4} \cup T_{2,5},$ where for $x, y \in \Sigma$,
$T_{2,1} = \{(\$_1 x, \$_1), (xy, \$_1), (x\$_2, \$_1), (\$_2, \$_1 x), (\$_2, xy), (\$_2, x\$_2)\},$
$T_{2,2} = \{(\$_2 x, \$_2), (xy, \$_2), (x\$_3, \$_2)\}, T_{2,3} = \{(\$_3 x, \$_2 x), (xy, xy), (x\$_4, x\$_3)\},$
$T_{2,4} = \{(\$_4, \$_3 x), (\$_4, xy), (\$_4, x\$_4)\}, T_{2,5} = \{(\$_4 x, \$_4 x), (xy, xy), (x\$_5, x\$_5)\}.$

Duplicate units are included for readability.

The intersection $L(S_1) \cap L(S_2) = \{\, \$_1 w_1 \$_2 w_2 \$_3 w_2 \$_4 w_1 \$_5 \mid w_1, w_2 \in \{a, b\}^+ \,\}$ is not accepted by any nondeterministic one-way two-head finite automaton [14] and, thus, by Theorem 3.1 not generated by any SAS. □

Concerning the catenation operation it is an open question whether the family of languages generated by SAS is closed under iteration. However, it is not closed under concatenation.

Theorem 4.7. *The family of languages generated by SAS is not closed under concatenation.*

Proof. By Corollary 2.4 and Theorem 3.6 the languages $L_1 = \{ a^n b^n \mid n \geq 1 \}$ and $L_2 = a^+$ are generated by SAS. We consider their concatenation. It remains to be shown that $L_1 L_2$ does not belong to the family of languages generated by SAS. Contrarily, we assume that it is generated by an SAS $\langle \{a, b\}, A, T, E \rangle$ and analyze the derivation of a word $a^n b^n a^{dn^2}$, where n is large enough and $d+1$ is the length of the longest string appearing in a unit from A, T, or E. To this end, subsets of T are defined as follows. $T_e(a) = \{ (a^i, a^i) \in T \mid i \geq 2 \}$, $T_u(a) = \{ (a^i, a^j) \in T \mid i > j \geq 1 \}$, and $T_l(a) = \{ (a^i, a^j) \in T \mid 1 \leq i < j \}$.

If $T_u(a)$ contains a unit (a^{i_1}, a^{j_1}) and $T_l(a)$ a unit (a^{i_2}, a^{j_2}), then the assembling of $j_2 - i_2$ units (a^{i_1}, a^{j_1}) and $i_1 - j_1$ units (a^{i_2}, a^{j_2}) extends the upper as well as the lower string by $k = i_1 j_2 - i_2 j_1 - i_1 - j_2 + i_2 + j_1$ symbols a. If $T_e(a)$ contains a unit (a^i, a^i), its assembling extends the upper as well as the lower string by $k = i - 1$ symbols a. Let (a^{i_0}, a^{j_0}) be the unit from A which starts the generation of the string $a^n b^n a^{dn^2}$. Then the upper and lower string can be extended by k symbols yielding (a^{i_0+k}, a^{j_0+k}), while the remaining derivation is unchanged. Therefore, string $a^{n+k} b^n a^{dn^2}$ is generated as well, which is a contradiction. We conclude that $T_e(a)$ and one of $T_u(a)$ and $T_l(a)$ must be empty. Without loss of generality, we assume $T_l(a)$ is empty.

So, the initial part of the derivation of $a^n b^n a^{dn^2}$ is of the form $(a^{i_0}, a^{j_0}) \Rightarrow^* (a^n b^{i_1}, a^{j_1})$, $1 \leq j_1 \leq n$ and $1 \leq i_1 \leq d$. We distinguish two cases for the continuation of the derivation.

Case 1. All symbols b of the upper string are generated before the first b of the lower string appears: $(a^n b^{i_1}, a^{j_1}) \Rightarrow^* (a^n b^n a^{i_2}, a^{j_2})$, $1 \leq j_2 \leq n$ and $0 \leq i_2 < d$. The next part of the derivation is until the first symbol a appears in the suffix of the lower string: $(a^n b^n a^{i_2}, a^{j_2}) \Rightarrow^* (a^n b^n a^{i_3}, a^n b^n a^{j_3})$, $0 \leq i_3 \leq 3dn$ and $1 \leq j_3 \leq d$. Since $T_e(a)$ and $T_l(a)$ are empty, the upper and lower string cannot be matched when $i_3 > j_3 + d$. If $i_3 \leq j_3 + d$ then we obtain $(a^n b^n a^{i_4}, a^n b^n a^{j_4})$, $i_4 \geq j_4 + d$ and $j_4 \leq j_3 + (j_3 + d)(d-1) \leq (j_3 + d)d \leq 2d^2$, after assembling at most $j_3 + d$ further units from $T_u(a)$. Again, since $T_e(a)$ and $T_l(a)$ are empty, the upper and lower string cannot be matched when $i_4 > j_4 + d$. Thus, dn^2 is bounded by $2d^2 + d$, a contradiction.

Case 2. The first symbol b of the lower string appears before the first suffix a of the upper string: $(a^n b^{i_1}, a^{j_1}) \Rightarrow^* (a^n b^{i_2}, a^n b^{j_2})$, $1 \leq i_2 \leq n$, $1 \leq j_2 \leq d$. If the derivation continues such that the first symbol a of the suffix appears first in the upper string, we obtain the same contradiction as in Case 1: $(a^n b^{i_2}, a^n b^{j_2}) \Rightarrow^* (a^n b^n a^{i_3}, a^n b^{j_3})$, $1 \leq j_3 \leq n$, $1 \leq i_3 \leq d$, or $(a^n b^{i_2}, a^n b^{j_2}) \Rightarrow^* (a^n b^n a^{i_3}, a^n b^n a^{j_3})$, $1 \leq i_3, j_3 \leq d$.

Therefore, we assume that the derivation continues such that the first symbol a of the suffix appears in the lower string only: $(a^n b^{i_2}, a^n b^{j_2}) \Rightarrow^* (a^n b^{i_3}, a^n b^n a^{j_3})$, $1 \leq i_3 \leq n$ and $1 \leq j_3 \leq d$. Consider the situation when the first suffix a also appears in the upper string: $(a^n b^n a^{i_4}, a^n b^n a^{j_4})$, $1 \leq i_4 \leq d$ and $1 \leq j_4$. Since $T_e(a)$ and $T_l(a)$ are empty, at most $j_4 + d - i_4$ further units from $T_u(a)$ can be assembled until the upper and lower string cannot be matched anymore. After that the length of the lower suffix is at most $j_4 + (j_4 + d - i_4)(d-1) < j_4 d + d^2 + i_4 \leq j_4 d + d^2 + d$. Therefore, the only way to generate a suffix of length dn^2 is to derive a situation where $j_4 d + d^2 + d > dn^2$, that is, $j_4 + d + 1 > n^2$ when the first suffix a appears in the upper string. This implies that there must be a unit $(b, a^p) \in T$ where $p \geq 2$. Now the idea is to assemble this unit additionally in the initial part of the derivation $(a^{i_0}, a^{j_0}) \Rightarrow^* (a^n b^{i_1}, a^{j_1})$. For every two numbers n and n' the pairs $(i_1, n - j_1)$ and $(i_1, n' - j_1)$ must be different. Otherwise some string $a^n b^{n'} a^{dn'^2}$, $n \neq n'$, could be generated. Since $i_1 \leq d$ and $T_u(a)$ is non-empty there are two numbers $n_1 < n_2$ such that their corresponding initial derivations yield to $(a^{n_1} b^{x_1}, a^{y_1})$ and $(a^{n_2} b^{x_2}, a^{y_2})$, where $x_1 = x_2$ and $|y_1 - y_2|$ is a multiple of $p - 1$. Let $y_1 < y_2$ and $y_2 - y_1 = c(p-1)$, $c \geq 1$. Then, by assembling the unit (b, a^p) additionally c times during the initial derivation of n_1, we obtain the result $(a^{n_1} b^{x_1}, a^{y_1 + c(p-1)}) = (a^{n_1} b^{x_2}, a^{y_2})$. So, the string $a^{n_1} b^{n_2} a^{dn_2^2}$ is generated, a contradiction, which completes the proof. □

The next (non-)closure properties are for homomorphisms. It turns out that the family in question is not closed even under weak, that is, non-erasing letter-to-letter homomorphisms, and non-erasing inverse homomorphisms.

Theorem 4.8. *The family of languages generated by SAS is not closed under λ-free letter-to-letter homomorphisms.*

Proof. We start to construct an SAS that generates the language

$$\{\, \$a^1\$a^3\$ \ldots \$a^{2i+1}\# \mid i \geq 1 \,\}.$$

$S = \langle \{a, b, \$, \#\}, A, T, E \rangle$, where

$$A = \{(\$a\$, \$)\}, \quad E = \{(\#, \#)\},$$
$$T = \{(\$a, \$a), (aa, aa), (aaa\$, a\$), (aaa\#, a\$), (\#, \$a), (\#, aa), (\#, a\#)\}.$$

Initially, the upper string $\$a\$$ and the lower string $\$$ is generated. Now assume that the upper string is $\$a^1\$a^3\$ \ldots \$a^{2i+1}$ and the lower string is $\$a^1\$a^3\$ \ldots \$a^{2i-1}$. Then by assembling the unit $(\$a, \$a)$, $2i$ times the unit (aa, aa), and the unit $(aaa\$, a\$)$ the upper string is extended to $\$a^1\$a^3\$ \ldots \$a^{2(i+1)+1}$ and the lower string to $\$a^1\$a^3\$ \ldots \$a^{2i+1}$. After repeating this process, the derivation is terminated by assembling the units containing a #.

The length of a word $w_i = \$a^1\$a^3\$ \ldots \$a^{2i+1}\# \in L(S)$ is $(i+1)^2 + i + 1 + 1$. Let the λ-free letter-to-letter homomorphism $h : \{a, \$, \#\}^* \to \{a\}^*$ be defined by $h(a) = h(\$) = h(\#) = a$. So, we obtain $h(L(S)) = \{\, a^{n^2+n+1} \mid n \geq 2 \,\}$ which is

a non-semilinear language. In particular $h(L(S))$ is non-regular. The main result of [4] can be used to conclude that any unary nondeterministic one-way multi-head finite automata language is regular and, therefore, by Theorem 3.1 that any unary language generated by an SAS is regular as well. So, $h(L(S))$ does not belong to the family of languages generated by SAS. □

Theorem 4.9. *The family of languages generated by SAS is not closed under inverse λ-free homomorphisms.*

Proof. The following SAS generates the language $\{ (ac)^n b^n \mid n \geq 1 \} \cup (ac)^+$.
$S = \langle \{a, b, c\}, A, T, E \rangle$, where

$$A = \{(ac, a), (ac, ac)\}, \quad E = \{(c, c), (b, b)\},$$
$$T = \{(cac, cac), (cac, a), (cb, ac), (bb, cac), (b, cb), (b, bb)\}.$$

Initially, it is guessed whether a string from $\{ (ac)^n b^n \mid n \geq 1 \}$ or from $(ac)^+$ is generated. In the former case the derivation starts with the unit (ac, a) and in the latter case with (ac, ac). In this way, during the generation of the $(ca)^+$ part of the word, the last symbol of the lower string indicates which case applies. If it is a c, a word from $(ca)^+$ is generated.

Now the λ-free homomorphism $h : \{a, b\}^* \to \{a, b, c\}^*$ is defined as $h(a) = ac$ and $h(b) = b$. So, the language $h^{-1}(L(S))$ is $\{ a^n b^n \mid n \geq 1 \} \cup a^+$, which is not generated by any SAS by example 4.2. □

Finally, we turn to a sole positive closure property

Theorem 4.10. *The family of languages generated by SAS* is *closed under reversal.*

Proof. Given an SAS $S = \langle \Sigma, A, T, E \rangle$ we construct an SAS $S_R = \langle \Sigma, A_R, T_R, E_R \rangle$ such that $L(S) = (L(S_R))^R$ by defining

$$A_R = \{ (u^R, v^R) \mid (u, v) \in E \},$$
$$E_R = \{ (u^R, v^R) \mid (u, v) \in A \}, \text{ and}$$
$$T_R = \{ (u^R, v^R) \mid (u, v) \in T \}.$$

So, the strings in the units are reversed and the sets of axiom and ending units are interchanged. Let the generation of a string w consist in assembling the units (u_1, v_1), $(u_2, v_2), \ldots, (u_n, v_n)$, where $(u_1, v_1) \in A$ and $(u_n, v_n) \in E$. Then it is evident that the units $(u_n^R, v_n^R), (u_{n-1}^R, v_{n-1}^R), \ldots, (u_2^R, v_2^R)$, (u_1^R, v_1^R) can be assembled to generate the reversal w^R of the string w. Since $(u_n^R, v_n^R) \in A_R$ and $(u_1^R, v_1^R) \in E_R$, string w^R is generated by S_R. For symmetric reasons it follows $L(S) = (L(S_R))^R$. □

Finally, in Table 1 we summarize our results on closure and non-closure properties.

TABLE 1. Closure properties of the family of languages generated by SAS.

	$\sim$	$\cup$	$\cap$	$\cap_{REG}$	$\cdot$	$+$	h_λ	h^{-1}	R
SAS	no	no	no	no	no	?	no	no	yes

5. Decidability problems

It seems to be an obvious choice to proof the undecidability of several problems for SAS by reduction of Post's Correspondence Problem (PCP) (see, for example, [12]).

Let Σ be an alphabet and an instance of the PCP be given by two lists $\alpha = u_1, u_2, \ldots, u_n$ and $\beta = v_1, v_2, \ldots, v_n$ of words from Σ^+. It is well-known that it is undecidable whether a PCP has a solution [11], that is, whether there is a nonempty finite sequence of indices $i_1, i_2, \ldots, i_k$ such that $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$. In the sequel we call $i_1, i_2, \ldots, i_k$ as well as $u_{i_1} u_{i_2} \ldots u_{i_k}$ a solution of the PCP. We start to show that emptiness is undecidable from which further undecidability results follow.

Theorem 5.1. *Emptiness is undecidable for SAS.*

Proof. In order to prove the assertion, for a given PCP we construct an SAS that generates exactly the strings which are solutions of the PCP. Since it is undecidable whether such a solution exists it is undecidable whether the SAS generates a string at all, that is, non-emptiness and emptiness.

Let an instance of the PCP be given by the two lists $\alpha = u_1, u_2, \ldots, u_n$ and $\beta = v_1, v_2, \ldots, v_n$ of nonempty words over some alphabet Σ. The SAS $S = \langle \Sigma, A, T, E \rangle$ is defined by

$$\begin{aligned} A &= \{ (u_i, v_i) \mid 1 \leq i \leq n \}, \\ T &= \{ (xu_i, yv_i) \mid 1 \leq i \leq n \text{ and } x, y \in \Sigma \}, \\ E &= \{ (x, x) \mid 1 \leq i \leq n \text{ and } x \in \Sigma \}. \end{aligned}$$

If $i_1, i_2, \ldots, i_k$ is a solution of the PCP, then the string $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$ is generated by S as follows. The initial unit from A is (u_{i_1}, v_{i_1}). Subsequently, the units $(x_{i_2} u_{i_2}, y_{i_2} v_{i_2})$ to $(x_{i_k} u_{i_k}, y_{i_k} v_{i_k})$ from T are assembled, where x_{i_j} matches the last symbol of $u_{i_{j-1}}$ and y_{i_j} matches the last symbol of $v_{i_{j-1}}$. Finally, the unit (x, x) from E is used to finish the generation, where x matches the last symbol of u_{i_k} which is equal to the last symbol of v_{i_k}.

Conversely, any successful generation of S must begin with a unit from A, say (u_{i_1}, v_{i_1}). Then there may follow units (xu_i, yv_i) from T, say $(x_{i_2} u_{i_2}, y_{i_2} v_{i_2})$ to $(x_{i_k} u_{i_k}, y_{i_k} v_{i_k})$, such that x_{i_j} matches the last symbol of $u_{i_{j-1}}$ and y_{i_j} matches the last symbol of $v_{i_{j-1}}$. In order to complete the generation successfully, a unit from E has finally to be assembled. Since this unit does not extend the string

generated any more, we derive that $u_{i_1}u_{i_2}\dots u_{i_k} = v_{i_1}v_{i_2}\dots v_{i_k}$ and, thus, have generated a solution of the PCP. This completes the proof. □

From the construction in the Proof of Theorem 5.1 and the undecidability of emptiness we can derive several further undecidability results immediately.

Theorem 5.2. *Finiteness, infiniteness, equivalence, and inclusion are undecidable for SAS.*

Proof. An SAS which generates the empty language can effectively be constructed. Therefore, the decidability of equivalence would imply the decidability of emptiness.

Similarly, the decidability of inclusion would imply the decidability of equivalence.

Any PCP either has no solution or infinitely many solutions. So, the SAS constructed in the Proof of Theorem 5.1 generates finitely many strings if and only if the PCP has no solution. This implies the undecidability of finiteness and infiniteness of SAS. □

Since SAS have been seen to generate even non-context-free languages, the questions whether regularity or context-freeness are decidable arise immediately.

Theorem 5.3. *Regularity and context-freeness are undecidable for SAS.*

Proof. First, we modify the construction given in the Proof of Theorem 5.1 such that the SAS generates exactly the strings which are solutions to the given PCP followed by the special symbol #. So, let an instance of the PCP be given by the two lists $\alpha = u_1, u_2, \dots, u_n$ and $\beta = v_1, v_2, \dots, v_n$ of nonempty words over some alphabet Σ not containing the symbols $\#, a, b, c$. The SAS $S = \langle \Sigma \cup \{\#\}, A, T, E\rangle$ is defined by

$$\begin{aligned} A &= \{\, (u_i, v_i) \mid 1 \le i \le n \,\}, \\ T &= \{\, (xu_i, yv_i) \mid 1 \le i \le n \text{ and } x, y \in \Sigma \,\}, \\ E &= \{\, (x\#, x\#) \mid 1 \le i \le n \text{ and } x \in \Sigma \,\}. \end{aligned}$$

The only modification is that the final unit assembled from E now extends the string by the special symbol.

Next, we modify S to the SAS $S' = \langle \Sigma \cup \{\#, a, b, c\}, A', T', E'\rangle$ such that

$$L(S') = \{\, w\#a^n b^n c^n \mid n \ge 1 \text{ and } w \text{ is a solution of the PCP} \,\}.$$

To this end, we consider the SAS $S'' = \langle \{a, b, c\}, A'', T'', E''\rangle$ from example 2.3 generating the language $\{\, a^n b^n c^n \mid n \ge 1 \,\}$.

Let $\tilde{T} = \{\, (x\#\tilde{u}, x\#\tilde{v}) \mid (\tilde{u}, \tilde{v}) \in A'' \,\}$, and define $A' = A$, $T' = T \cup \tilde{T} \cup T''$, and $E' = E''$. Since the alphabets of S and S'' are disjoint and both do not contain the special symbol #, it is evident that

$$L(S') = \{\, w\#a^n b^n c^n \mid n \ge 1 \text{ and } w \text{ is a solution of the PCP} \,\}.$$

If the PCP has no solution, the units from $\tilde{T}$ can never be assembled and, thus, language $L(S')$ is empty, that is, regular and context free. Conversely, a straightforward application of Ogden's lemma [8] shows that $L(S')$ is not context free if the PCP has a solution. This shows the undecidability of context-freeness as well as regularity. □

6. Conclusions

We have studied string assembling systems, which is a computational model that generates strings from copies out of a finite set of assembly units. In particular, it turned out that any string assembling system can be simulated by some nondeterministic one-way two-head finite automaton, while the stateless version of the two-head finite automaton marks to some extent a lower bound for the generative capacity. The family of languages generated by SAS is properly included in NL and, thus, in the family of context-sensitive languages. It is incomparable with the family of (deterministic) (linear) context-free languages as well as with the family of regular languages. It includes the language $\{ a^{2n} \mid n \geq 1 \}$ which is not accepted by any stateless nondeterministic one-way two-head finite automaton. So, SAS can copy substrings while sticker systems cannot. Conversely, some variant of the mirror language $\{ w \mid w \in \{a, b\}^* \text{ and } w = w^R \}$ is generated by many variants of sticker systems (that can generate all linear context-free languages), but cannot be generated by any SAS. So, one could say, sticker systems can handle mirrored inputs while SAS cannot. It turned out that the language family considered is not closed under five of the six AFL operations, where the remaining one, the iteration, is an open problem. Furthermore we obtained non-closure under complementation. The only positive closure property is for reversal. Moreover, emptiness, finiteness, inclusion, regularity, and context-freeness are all undecidable for SAS.

Nevertheless, several fundamental questions for string assembling systems remain untouched or unanswered and may be interesting and fruitful for further investigations. We mention three of them:

Is it possible to find a more precise lower bound for the computational capacity? Are the additional symbols used in the simulation of a stateless two-head automaton in the Proof of Theorem 3.4 necessary?

Is the family of languages generated by SAS closed under iteration? If not, it forms an anti-AFL.

Is universality ($L(S) = \Sigma^*$) decidable?

References

[1] R. Freund, G. Păun, G. Rozenberg and A. Salomaa, Bidirectional sticker systems, in *Pacific Symposium on Biocomputing (PSB)*. World Scientific, Singapore (1998) 535–546.

[2] J. Hartmanis, On non-determinancy in simple computing devices. *Acta Inf.* **1** (1972) 336–344.

[3] M. Holzer, M. Kutrib and A. Malcher, Multi-head finite automata: origins and directions. *Theoret. Comput. Sci.* **412** (2011) 83–96.
[4] O.H. Ibarra, A note on semilinear sets and bounded-reversal multihead pushdown automata. *Inf. Process. Lett.* **3** (1974) 25–28.
[5] O.H. Ibarra, J. Karhumäki and A. Okhotin, On stateless multihead automata: hierarchies and the emptiness problem. *Theoret. Comput. Sci.* **411** (2009) 581–593.
[6] L. Kari, G. Păun, G. Rozenberg, A. Salomaa and S. Yu, DNA computing, sticker systems, and universality. *Acta Inf.* **35** (1998) 401–420.
[7] R. McNaughton, Algebraic decision procedures for local testability. *Math. Syst. Theory* **8** (1974) 60–76.
[8] W.F. Ogden, A helpful result for proving inherent ambiguity. *Math. Syst. Theory* **2** (1968) 191–194.
[9] C.H. Papadimitriou, *Computational Complexity*. Addison-Wesley (1994)
[10] G. Păun and G. Rozenberg, Sticker systems. *Theoret. Comput. Sci.* **204** (1998) 183–203.
[11] E.L. Post, A variant of a recursively unsolvable problem. *Bull. AMS* **52** (1946) 264–268.
[12] A. Salomaa, *Formal Languages*. Academic Press, New York (1973)
[13] L. Yang, Z. Dang and O.H. Ibarra, On stateless automata and P systems, in *Workshop on Automata for Cellular and Molecular Computing.* MTA SZTAKI (2007) 144–157.
[14] A.C. Yao and R.L. Rivest, $k + 1$ heads are better than k. *J. ACM* **25** (1978) 337–340.
[15] Y. Zalcstein, Locally testable languages. *J. Comput. Syst. Sci.* **6** (1972) 151–167.

Communicated by C. Mereghetti.
Received November 8, 2011. Accepted June 13, 2012.