Self-assembly of strings and languages

Erzsébet Csuhaj-Varjú\textsuperscript{a} Ion Petre\textsuperscript{b} György Vaszil\textsuperscript{c}

\textsuperscript{a}Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13-17, 1111 Budapest, Hungary
\textsuperscript{b}Academy of Finland and Department of Information Technologies, Åbo Akademi University, Turku Centre for Computer Science, Turku 20520, Finland
\textsuperscript{c}Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13-17, 1111 Budapest, Hungary

Abstract

Self-assembly is the process in which simple objects autonomously aggregate into large structures and it has become one of the major tools for nano-scale engineering. We propose in this paper a string-based framework inspired by the principle of self-assembly: two strings with a common overlap, say $uv$ and $vw$, yield a string $uvw$: we say that string $uvw$ has been assembled from strings $uv$ and $vw$. The operation may be extended in a natural way also to sets of strings. We answer several questions: what is the assembly power of a given set of strings, can a given set of strings be generated through assembly and if so, what is a minimal generator for it?

1 Introduction

Self-assembly, the process by which simple objects aggregate into complex structures, has been widely investigated in recent years as a solution for nano-scale engineering. The idea here is to design a set of simple objects in such a way that they will self-aggregate (with a high probability) exactly into the desired structures. A particular case is that of linear self-assembly in which one-dimensional objects such as DNA double strands interact with each other to form longer strands. Linear self-assembly has been for more than a decade now the core of most experiments in DNA computing, starting with the celebrated experiment of Adleman, see [1], and also [3], [7].

Email addresses: csuhaj@sztaki.hu (Erzsébet Csuhaj-Varjú), ipetre@abo.fi (Ion Petre), vaszil@sztaki.hu (György Vaszil).
We propose in this paper a string-based operation inspired by linear self-assembly: applied to two strings of the form $xy, yz$, the result of the operation is word $xyz$, in the style of DNA recombination. Clearly, two given strings may share more than one common overlap and so, the result of the operation is in general a set of strings. The operation may be easily extended to sets of words, by applying it to all possible pairs of strings from the two sets. It is then natural to consider the iterated version of this operation. Several questions are immediate: what are the closure properties of the classical language families under iterated assembly, can we decide if a given language can be obtained by iterated assembly and if so, can we effectively construct a minimal set of initial strings?

Several other frameworks inspired by (linear) self-assembly exist, see [4], [5], [6], [10], [11], [12]. Perhaps the best known model for self-assembly was proposed by Winfree, see [10]. His model concentrates on two-dimensional assembly of DNA complexes; linear self-assembly comes as a particular case of his model. Our operation motivated by linear self-assembly is a generalization of Winfree’s approach in the following sense: while in Winfree’s model the overlap between two “tiles” is of fixed length, we allow variable length – two words may overlap on anything from one letter to the length of the shorter of them.

In [6] the authors consider an operation on strings and languages similar to the one introduced in this paper and study its closure properties and some iterated variants. Our approach is different (see also Section 5): we are interested in questions related to nano-scale fabrication: given a finite set of objects (strings), what pattern (language) will they generate by self-assembly? Reversely, which patterns can be obtained through self-assembly and how can one find a finite set of objects that self-aggregate into the desired pattern? We answer these questions here in terms of languages.

2 Self-assembly of strings and languages

Throughout this paper, we assume that the reader is familiar with the basic notions of formal language theory; we list here only some notations and notions we use in the sequel. For further reading we refer to [8] and [2].

Throughout this paper, $\Sigma$ denotes a finite alphabet. Let $\Sigma^*$ be the set of all finite words (equivalently, strings) over $\Sigma$, and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ where $\varepsilon$ denotes the empty word. The class of regular, context-free, context-sensitive, and recursively enumerable languages are denoted by $\mathcal{L}(\text{REG})$, $\mathcal{L}(\text{CF})$, $\mathcal{L}(\text{CS})$, and $\mathcal{L}(\text{RE})$, respectively.

We say that a word $u$ is a prefix (suffix, subword, resp.) of $v$ if $v = uw$ ($v = wu$,
$v = w_1uw_2$, resp.), for some word $w$ (words $w_1, w_2$). We denote the set of prefixes (suffixes, subwords, resp.) of a word $v$ by $\text{Pref}(v)$ ($\text{Suff}(v)$, $\text{Sub}(v)$, resp.). For a language $L$ and $\alpha \in \{\text{Pref}, \text{Suff}, \text{Sub}\}$, we denote $\alpha(L) = \{\alpha(x) \mid x \in L\}$.

**Definition 1** Let $x, y \in \Sigma^+$. The (self-)assembly operation $S(x, y)$ is defined as follows:

$$S(x, y) = \{x'zy' \in \Sigma^+ \mid x = x'z, y = zy', z \neq \epsilon\}.$$  

We say that a word $x'zy'$, $z \neq \epsilon$, is a non-trivial assembly of $x'z$ and $zy'$ if $x' \neq \epsilon$ and $y' \neq \epsilon$, and is trivial otherwise.

For two languages $L_1, L_2 \subseteq \Sigma^+$, we define the assembly operation $S(L_1, L_2)$ as follows:

$$S(L_1, L_2) = \bigcup_{x_1 \in L_1, x_2 \in L_2} S(x_1, x_2).$$

Without risk of ambiguity, we also define the unary operations $S^n(L)$ for languages $L \subseteq \Sigma^+$ and for any $n \geq 1$ as follows:

$$S^n(L) = S(L, S^{n-1}(L)),$$

where $S^0(L) = L$. We also write $S$ instead of $S^1$. We then define the iterated assembly of $L$ as follows:

$$S^+(L) = \bigcup_{n \geq 1} S^n(L).$$

We say that a language $L$ is closed under iterated assembly if $L = S^+(L)$.

**Example 1**

(a) It is easy to see that $x \in S(x, x)$ holds for all $x \in \Sigma^+$. Consequently, $L \subseteq S(L)$, $L \subseteq S^+(L)$, or in general, $S^i(L) \subseteq S^j(L)$, $0 \leq i \leq j$, for any language $L$. Thus, $S^+(L) = \bigcup_{n \geq 0} S^n(L)$.

(b) $S(abb, bbc) = \{abb, bbc, abbc, abbbc\}$.

(c) Let $F_1 = \{abb, bbc\}$. Then $S^+(F_1) = \{abb, bbc, abbc, abbbc\}$.

(d) Let $F_2 = \{ab, bb, bc\}$. Then $S^+(F_2) = \{ab', bb', b'c, ab'^c \mid i \geq 1\}$.

(e) Note that $S(aba, aba) = \{aba, ababa\}$. The word $aba$ is a trivial assembly of $aba$ with itself, while $ababa$ is a non-trivial one.

(f) The word $ab$ is a trivial assembly of $a$ and $ab$ and also a trivial assembly of $ab$ and $b$.

Note that the set of those words $z$ as in Definition 1 is similar to the set of constants of a language $L$, see [9]. Indeed, recall that for a language $L$ over alphabet $\Sigma$, a word $\alpha \in \Sigma^+$ is called a constant of $L$ if $\alpha \in \text{Sub}(L)$ and for any $x, y, x', y' \in \Sigma^+$ such that $x\alpha y, x'\alpha y' \in L$, we have that $x\alpha y', x'\alpha y \in L$. On the other hand, observe that for a language $L$ with $L = S^+(L)$, the set
of the overlap words $z$ as in Definition 1 may be defined as the set of words $\beta \in \text{Pref}(L) \cap \text{Suff}(L) \cap \Sigma^+$ such that for any $x, y \in \Sigma^*$, if $x\beta, \beta y \in L$, then $x\beta y \in L$.

The following restricted form of the $S$ operation is useful in our considerations.

**Definition 2** The restricted assembly of two strings $x, y \in \Sigma^+$ is defined as follows:

$$\mathcal{R}(x, y) = \{x'zy' \in \Sigma^+ \mid x = x'z, y = zy', x', y' \neq \varepsilon\}.$$  

For two languages $L_1, L_2 \subseteq \Sigma^+$, we define the restricted assembly operation $\mathcal{R}(L_1, L_2)$ as follows:

$$\mathcal{R}(L_1, L_2) = \bigcup_{x_1 \in L_1, x_2 \in L_2} \mathcal{R}(x_1, x_2).$$

Without risk of ambiguity, we also define the unary operations $\mathcal{R}^n(L)$ for languages $L \subseteq \Sigma^+$ and for any $n \geq 1$ as follows:

$$\mathcal{R}^n(L) = \mathcal{R}(L, \mathcal{R}^{n-1}(L)),$$

where $\mathcal{R}^0(L) = L$. We also write $\mathcal{R}$ instead of $\mathcal{R}^1$. We then define the iterated restricted assembly of $L$ as follows:

$$\mathcal{R}^+(L) = \bigcup_{n \geq 1} \mathcal{R}^n(L).$$

Note that for any $x, y \in \Sigma^+$, if $z \in S(x, y) \setminus \mathcal{R}(x, y)$, then either $z = x$ and $y \in \text{Suff}(x)$, or $z = y$ and $x \in \text{Pref}(y)$.

**Example 2** (i) $S(ab, ab) = \{ab\}$ and $\mathcal{R}(ab, ab) = \emptyset$.

(ii) $S(a, ab) = \{ab\}$, $\mathcal{R}(a, ab) = \emptyset$.

The following lemma gives a canonical form for the words in $S^+(L)$ and in $\mathcal{R}^+(L)$, for any $L \subseteq \Sigma^+$.

**Lemma 1** For any language $L \subseteq \Sigma^+$ and any word $x \in S^+(L)$, there are words $u_i \in L$ and $\alpha_j, \beta_k \in \Sigma^*$, $0 \leq i \leq n$, $1 \leq j, k \leq n$, such that

(i) $x = \beta_1\alpha_1\beta_2\alpha_2 \ldots \beta_{n-1}\alpha_{n-1}\beta_n$;

(ii) $u_i = \alpha_{i-1}\beta_i \alpha_i \in L$, for all $1 \leq i \leq n$;

(iii) $\alpha_0 = \alpha_n = \varepsilon$, $\alpha_i \neq \varepsilon$, for all $1 \leq i \leq n - 1$, and $\beta_1, \beta_n \neq \varepsilon$.

Moreover, $x \in \mathcal{R}^+(L)$ if and only if $n \geq 2$. 


PROOF. The claims follow by noting that the words of \( L \) are of this form (trivially with \( n = 1 \)) and given two words of this form, the words produced by the assembly operation are also of this form, see Figure 1.

Fig. 1. The structure of \( x \in S^+(L) \), where \( u_i \in L, \beta_1, \beta_n \neq \varepsilon \), and \( \alpha_i \neq \varepsilon \), for all \( 1 \leq i \leq n-1 \), see Lemma 1.

With the notation of Lemma 1, we call \((u_1, u_2, \ldots, u_n)\) an \( L \)-decomposition for \( x \) and we write \( x \in S(u_1, u_2, \ldots, u_n) \). The decomposition is non-trivial if \( n \geq 2 \). Note that a word \( x \in S^+(L) \) may have several distinct \( L \)-decompositions.

Example 3 (i) If \( L = \{a^2, a^3\} \), then \( a^4 \) has two distinct \( L \)-decompositions: \( a^4 \in S(a^2, a^3) \cap S(a^3, a^2) \).
(ii) If \( L = \{aa, ab, aba, baa\} \), then \( abaa \) has three distinct \( L \)-decompositions: \( abaa \in S(ab, baa) \cap S(aba, aa) \cap S(aba, baa) \).

The following result is a simple consequence of Lemma 1.

Lemma 2 For any language \( L \subseteq \Sigma^+ \),

(i) \( S^+(L) \) is the set of all words with at least one \( L \)-decomposition and 
(ii) \( R^+(L) \) is the set of all words with non-trivial \( L \)-decompositions.

The following result gives an insight on non-trivial assemblies.

Lemma 3 Let \( L \subseteq \Sigma^+ \) and \( x \in L \).

(a) If \( y \in R^+(L) \), then \( S(x, y) \subseteq R^+(L) \cup \{x\} \).
(b) If \( y \in S^+(L) \), \( x \neq y \), and \( x = uv, y = vw, u, v, w \neq \varepsilon \), then \( uvw \in R^+(L) \).

PROOF. Consider a word \( \alpha \in S(x, y) \). If \( \alpha = x \) or \( \alpha = y \), then \( \alpha \in R^+(L) \cup \{x\} \), otherwise \( \alpha \) must have a non-trivial \( L \)-decomposition, thus also in this case \( \alpha \in R^+(L) \) which means that \( S(x, y) \subseteq R^+(L) \cup \{x\} \).

A similar argument can be used to prove part (b) of the claim.

3 Closure properties

The following result describes the closure of some well-known language classes under our assembly operation.
Theorem 4  (i) For any given regular (context-sensitive, recursive, recursively-enumerable, resp.) language \( L \), \( S^+(L) \) is regular (context-sensitive, recursive, recursively-enumerable, resp.) and effectively constructible.
(ii) \( L(CF) \) is not closed under iterated assembly.

The same results hold also when we replace the assembly operation \( S \) with its restricted variant \( R \).

PROOF. To prove (i) note that based on Lemma 1, we can construct an automaton starting from a language \( L \) which accepts the language \( S^+(L) \).

The idea is to construct an automaton which simulates the simultaneous work of two instances of \( M \), the machine accepting \( L \). Any word \( w \in S^+(L) \) can be written in the form above as \( w = u_1v_1\ldots u_tv_tu_{t+1} \), thus a machine \( M' \) accepting \( S^+(L) \) can be constructed in the following way.

Automaton \( M' \) should start by simulating the work of \( M \) until at some non-deterministically chosen point it should start the simulation of also a second instance of \( M \) and then simulate the simultaneous work of the two instances until the first one enters one of its accepting states. After this happens, \( M' \) should continue by only simulating the work of the second instance of \( M \) until the reading of the input is finished in an accepting state, or before that happens, it should be able to start also the simulation of the first instance of \( M \) again, etc. If \( M' \) is constructed this way, it reads each segment \( v_iu_{i+1}v_{i+1} \), \( 1 \leq i \leq t-1 \), or the beginning and ending parts \( u_1v_1, v_tu_{t+1} \) completely by one of the simulated instances of \( M \), while the overlapping parts, \( v_i \), \( 1 \leq i \leq t \) are read simultaneously by both of the simulated instances of \( M \).

We give the explicit construction for finite automata and regular languages.

Let \( L \subseteq \Sigma^* \) be a regular language, and let \( M = (\Sigma, Q, q_0, \delta, F) \) be a finite automaton with \( L = L(M) \). Let us assume, without loss of generality, that the transition mapping \( \delta \) maps \( Q \times \Sigma \) into subsets of \( Q \), thus, \( M \) never executes a so-called "\( \varepsilon \)-transition", a transition when the state is changed without reading any input symbol.

We construct a finite automaton \( M' = (\Sigma, Q', [q_0], \delta', F') \) such that \( S^+(L) = L(M') \). Let

\[
Q' = \{[q], [q_1, q_2] \mid q, q_1, q_2 \in Q\},
\]

and let

\[
\delta'([q], a) = \{[q'], [q', q_0] \mid q' \in \delta(q, a)\}, \quad \delta'([q_1, q_2], a) = \{[q'_1, q'_2] \mid q'_1 \in \delta(q_1, a), q'_2 \in \delta(q_2, a)\} \cup \{[q'_2] \mid q'_2 \in \delta(q_2, a), q_1 \in F\}
\]
for each \( a \in \Sigma \) and \([q], [q_1, q_2] \in Q'\). Let also
\[
F' = \{ [q] \mid q \in F \}.
\]
This automaton simulates the simultaneous work of two instances of \( M \) as described above. The context-sensitive case is very similar, thus it is left to the reader.

For a recursive or recursively-enumerable language \( L \) accepted by a Turing machine \( M \), one can build a two-tape Turing machine \( M' \) accepting \( S^+(M) \). Moreover, if \( M \) is always terminating, then so is \( M' \). The construction is based on Lemma 6: \( M' \) keeps the input on the first tape, guesses non-deterministically the decomposition in Lemma 6, writes each \( \alpha_{i-1}\beta_i\alpha_i \) on the second tape and simulates \( M \) on this input.

To prove (ii), consider the context-free language \( L = \{ a^i$b^i$\#, $b^i$#c^i \mid i \geq 1 \} \) over the alphabet \( \Sigma = \{a, b, c, $, \#\} \), and notice that
\[
S^+(L) = \{ a^i$b^i$#c^i, a^i$b^i$#, $b^i$#c^i \mid i \geq 1 \},
\]
which is not a context-free language.

Claims (i) and (ii) can be proved similarly when we replace \( S \) with \( R \), based on Lemma 1.

4 The base of a language

We study in this section how to construct languages through iterated assembly, starting from a given set of initial strings. Our main question here is the following: given a language, can it be generated though iterated assembly and if so, from which “starting point”? We define the notion of the base of a given language as a set of strings which can produce the words of the language by applying the iterated assembly operation. We prove that any language has a unique minimal base with respect to assembly and give its description. We also prove that the minimal base of a regular language is regular. The similar problem for context-free languages is difficult: it is undecidable if a context-free language has a base at all.

**Definition 3** Let \( L, B \subseteq \Sigma^+ \) be two languages. We say that \( B \) is a base of \( L \) if \( L = S^+(B) \). If \( B \) is finite, then we say that \( B \) is a finite base for \( L \) and also that \( L \) is finitely generated by \( B \). We say that \( B \) is a minimal base for \( L \) if for any other base \( B' \) of \( L \), if \( B' \subseteq B \), then \( B' = B \).

**Example 4** The set \( S^+(F_2) = \{ ab^i, bb^i, b^i c, ab^i c \mid i \geq 1 \} \) from Example 1 is finitely generated, \( F_2 = \{ ab, bb, bc \} \) is a minimal finite base.
Definition 4 Let $L \subseteq \Sigma^+$ be a language. A word $x \in \Sigma^+$ is irreducible over $L$ (with respect to the assembly operation) if $x \notin S^+(L \setminus \{x\})$. Otherwise, the word $x$ is called composite over $L$.

Lemma 5 Let $L \subseteq \Sigma^+$ be a language.

(i) If $x \in S^+(L)$ is irreducible over $L$, then $x \in L$ and it has only a trivial $L$-decomposition.
(ii) If a word $x \in \Sigma^+$ has only a trivial $L$-decomposition, then it is irreducible over $L$.

PROOF. The claims follow directly from Definition 4 and Lemma 1.

Lemma 6 For any language $L \subseteq \Sigma^+$, all words in a minimal base of $L$ must be irreducible in $L$.

PROOF. The claim follows observing that if $B$ is a base for $L$ and $x \in B$ is composite, then $B \setminus \{x\}$ is also a base for $L$.

These results have interesting implications concerning the uniqueness of the minimal base.

Theorem 7 If a language $L \subseteq \Sigma^+$ has a base, then it has a unique minimal base.

PROOF. It is easy to conclude from Lemma 6 that a minimal base coincides with the set of irreducible words over $L$. Indeed, the irreducible words over $L$ are in all bases of $L$.

For a language $L \subseteq \Sigma^+$, we denote by $\mathcal{B}(L)$ its unique minimal base, if it exists.

Clearly, a language $L$ has a base if and only if it is closed under iterated assembly, i.e., $L = S^+(L)$. For any such language, $L$ is its own base. As such, we are interested here in two problems.

Problem 1 Decide whether a language is closed under assembly.

Problem 2 For a language closed under assembly, find its minimal base.
Note that these two problems are inspired by questions in nano-scale engineering. Indeed, one must answer there whether a desired pattern can be obtained by assembly and if so, one must build a (minimal) set of objects that generate the pattern of interest.

**Example 5**
(i) The regular language \( L_1 = \{ ab^i c \mid i \geq 1 \} \) is closed under assembly. Indeed, for any \( x, y \in L \), if \( x \neq y \), then \( S(x, y) = \emptyset \).

(ii) The language \( L_2 = \{ a^i b^j c^k \mid i, j, k \geq 0 \} \) is also closed under assembly. Indeed, if \( z \in S(x, y) \setminus \{ x, y \} \) for \( x, y \in L \), then \( \text{Suff}(x) \cap \text{Pref}(y) \neq \emptyset \) and so, \( x = a^i b^j c^k \), for some \( i, j, k \geq 0 \) and either \( y = a^{i_1} b^{j_1} c^{k_1} \), or \( y = b^{j_2} c^{k_1} \), for some \( 0 \leq i_1 \leq i \), \( k_1 \geq k \), \( 0 \leq j_2 \leq j \). Then \( z = a^i b^j c^k \) and so, \( z \in L \).

In the following two theorems we give a solution to Problem 1. Based on the construction in the proof of Theorem 4, we can decide if \( S^+(L) = L \) holds for a regular language \( L \) by constructing the finite automaton accepting \( S^+(L) \) and then checking if \( S^+(L) = L \) holds.

**Theorem 8** The problem whether a base exists for any \( L \in \mathcal{L}(\text{REG}) \) is decidable.

In contrast to the above result, the same question is undecidable for the other language classes of the Chomsky hierarchy.

**Theorem 9** For \( L \in \mathcal{L}(X), \ X \in \{ \text{CF, CS, RE} \} \), it is undecidable if \( L = S^+(L) \) holds and thus, it is undecidable if a base for \( L \) exists.

**PROOF.** Let \( \Sigma \) be a finite alphabet and consider the language \( L = \$\Sigma^* \cup \Sigma^* \# \cup \$L'\# \) where \( L' \) is an arbitrary language over \( \Sigma \). Note that \( \$\Sigma^* \# \subseteq S^+(L) \). It is easy to prove that \( L = S^+(L) \) if and only if \( L' = \Sigma^* \). But the question whether \( L' = \Sigma^* \) is undecidable for context-free languages \( L' \) and so, the problem whether \( S^+(L) = L \) is also undecidable for context-free languages.

We prove now several lemmas leading to the solution of Problem 2, that is, to the description of the minimal base of any language.

**Lemma 10** For any language \( L \subseteq \Sigma^+ \), if \( x \notin R^+(L) \), then \( x \) is irreducible over \( L \).

**PROOF.** The claim follows from Lemma 2 and Lemma 5.

**Lemma 11** For any language \( L \subseteq \Sigma^+ \), \( S^+(L) = L \cup R^+(L) \).

**PROOF.**
By Lemma 2, if \( x \in S^+(L) \setminus R^+(L) \), then \( x \) has only trivial \( L \)-decompositions. The claim follows by Lemma 5.

**Lemma 12** For any language \( L \subseteq \Sigma^+ \), \( R^+(L) \) is the set of all composite words over \( L \).

**PROOF.** One inclusion was proved in Lemma 10. The other inclusion follows from Lemma 2 using a simple argument on the length of words.

We can describe now the minimal base of an arbitrary language \( L \).

**Theorem 13** For any language \( L \subseteq \Sigma^+ \) closed under assembly, \( \mathcal{B}(L) = S^+(L) \setminus R^+(L) \).

**PROOF.** The claim follows from Lemmas 5, 11, and 12.

Note that for any language \( L \) with the property that \( S(x, y) = \emptyset \) for all \( x, y \in L \), \( x \neq y \), we have that \( S^+(L) = L \). Moreover, in this case, for any \( L' \subseteq L \), \( S^+(L') = L' \). Consequently, \( L \) is a minimal base for itself. Clearly, if \( L \) is infinite then in this case, no finite base exists for \( L \).

The regular language \( L_1 \) in Example 5 shows that there are even regular languages that are closed under assembly but are not finitely generated. However, we prove in the following results that in the case of regular languages, we can either construct a finite base for a given language \( L \), or \( L \) is not finitely generated.

**Theorem 14** The minimal base of any regular language \( L \) closed under assembly is regular and can be effectively constructed.

**PROOF.** This is immediate from Theorem 13 since both \( S^+(L) \) and \( R^+(L) \) are regular and effectively constructible, according to Theorem 4.

**Corollary 15** It is decidable whether a given regular language \( L \)

(i) is closed under assembly (equivalently, it has a base);
(ii) has a non-trivial base (different than itself); in this case a minimal base for \( L \) can be effectively constructed;
(iii) is finitely generated; in this case a (minimal) finite base for \( L \) can be effectively constructed.
PROOF. The result follows from Theorem 14 since the equality and finiteness are decidable for regular languages.

5 Discussion

We proposed in this paper a simple operation for strings and languages motivated by self-assembly. Based on this, we were able to answer several questions inspired by the use of self-assembly in nano-scale fabrication.

In [6] an operation similar to our operations $S$ and $R$ was investigated. Their notion is only different in that they consider a dual Watson-Crick-like alphabet $\Sigma \cup \overline{\Sigma}$ and the inversion mapping $u \rightarrow \overline{u}$ defined in the usual way. With these, they consider the operation $\diamondsuit\text{WK}$, where $z \in x \diamondsuit\text{WK} y$ for words $x, y, z$ if and only if (i) $x = uv, y = \overline{vw}, z = \overline{uvw}$, or (ii) $x = z = u\overline{y}v$, or (iii) $x = uv, y = wu, z = \overline{wuv}$, or (iv) $y = z = uv\overline{v}$. Note however that Cases (iii) and (iv) are the symmetrics of Cases (i) and (ii), while Case (ii) is trivial in the sense that no new words are being produced. Our operation is similar to Case (i) above but brings one important advantage: it eliminates the inconvenience of dealing with a dual alphabet thus making all considerations more straightforward. Note also that a Watson-Crick alphabet and an operation defined as in (i)-(iv) above implicitly assume an intermediary step of DNA melting that will generate the result $z$ of the operation but also the complementary strand – this issue however, is not discussed in [6]. More importantly, the fundamental difference between our paper and [6] is in the approach. While [6] mainly considers closure properties in the framework of formal languages, we also approach the reverse direction: given a language, decide if it can be generated through iterated assembly and if so, find a minimal base for it.

Several directions may be further investigated. E.g., one may naturally consider the variant where recombination is only allowed for words $x = uv$ and $y = vv$ if $v$ is long enough, and allow dynamical changes of this parameter – this is a natural way to include reasoning about thermodynamical variations into the model. Several possibilities exist here: we may require that the length of $v$ is exactly $n$, or at most $n$, or at least $n$, for some nonnegative integer $n$. Thus, e.g., we may define the assembly of order $n$ as

$$S_n(x, y) = \{x'zy' \in \Sigma^+ \mid x = x'z, y = zy', z \in \Sigma^+, |z| = n\}.$$  

Obviously, we can consider assemblies of order at most $n$, say $S_{\leq n}$, or assemblies of order at least $n$, say $S_{\geq n}$. These operations demonstrate analogous properties to the non-restricted assembly, for example, the reader can easily verify that $\mathcal{L}(\text{REG})$ and $\mathcal{L}(\text{CS})$ are closed under the iterated version of these operations. We notice that bound $n$ serves as a size complexity measure as
well, thus, for example, the comparison of the language classes \( S^*_k(\text{REG}) \) and \( S^*_j(\text{REG}) \) for different \( k \) and \( j \) is certainly of interest, where \( S^*_n(\text{REG}) \) denotes the class of languages that are obtained by assembly of order \( n \) from regular languages.

As we mentioned also before, an important practical question in practice is whether a desired pattern can be obtained by self-assembly or not. Continuing this line of considerations, we can ask how much time is necessary to obtain this object starting from a certain starting point. In our model, the question translate into how many iterative steps are necessary to obtain a given language, starting from its minimal base. Of special interest are those languages that can be obtained in a finite number of iterative steps starting from their minimal bases. This “derivational complexity” measure gives information on the efficiency of the self-assembly process. Moreover, we can also ask how many steps are necessary to obtain a certain word starting from a given base, or if it can be obtained at all. Here one may also ask what is a minimal assembly of a given word, where minimality may be seen in at least two ways: minimality of the base or minimality of the number of iteration assembly steps. The uniqueness of this type of base is of course also an issue here.

A comparison between the descriptonal complexity of a regular language \( L \) and that of \( S^+(L) \) may also be interesting, e.g., in terms of size of their minimal automata. Connections between their minimal automata may give a valuable insight into the iterated assembly operation.

Acknowledgments

This work was supported in part by the Hungarian Scientific Research Fund “OTKA” grant no. T 042529, and by Academy of Finland, grant 203667. The work of Ion Petre is also supported by Academy of Finland, grant 108421. The authors are grateful to an anonymous reviewer for his/her careful observations.

References


