Noname manuscript No.
(will be inserted by the editor)

On the overlap assembly of strings and languages

Srujan Kumar Enaganti · Oscar H. Ibarra · Lila Kari · Steffen Kopecki

Received: date / Accepted: date

Abstract This paper investigates properties of the binary string and language operation *overlap assembly* which was defined by Csuhaj-Varju, Petre and Vaszil as a formal model of the linear self-assembly of DNA strands: The overlap assembly of two strings, xy and yz, which share an "overlap" y, results in the string xyz. The study of overlap assembly as a formal language operation is part of ongoing efforts to provide a formal framework and rigorous treatment of DNA-based information and DNA-based computation. Other studies along these lines include theoretical explorations of splicing systems, insertion/deletion systems, substitution, hairpin extension, hairpin reduction, superposition, overlapping catenation, conditional concatenation, contextual intra- and intermolecular recombinations, template-guided recombination, as well as directed extension by PCR. In this context, we investigate overlap assembly and its properties: closure properties of basic language families under this operation, decision problems, as well as the possible use of iterated overlap assembly to generate combinatorial DNA libraries.

This research was supported by a Natural Science and Engineering Council of Canada (NSERC) Discovery Grant R2824A01 and a University of Western Ontario Grant to L.K., and US NSF Grant CCF-1117708 to O.H.I.

S.K. Enaganti · L. Kari · S. Kopecki
Department of Computer Science
The University of Western Ontario
London, ON
N6A 5B7, Canada

O. H. Ibarra
Department of Computer Science
University of California Santa Barbara
Santa Barbara, CA
93106, USA

1 Introduction

In this paper we investigate properties of a formal language operation that models the linear self-assembly of DNA strands which partially "overlap". This binary operation which, given input the strands xy and yz (where the overlap y is non-empty), produces the output xyz, was introduced in [7] where it was called "(self)-assembly" of strings and languages. To distinguish it from other types of DNA self-assembly, this operation is herein called *overlap assembly*. Experimentally, (parallel) overlap assembly of DNA strands under the action of the DNA Polymerase enzyme was used for gene shuffling in, e.g., [47]. In the context of experimental DNA Computing, overlap assembly was used in, e.g., [8, 13, 25, 42] for the formation of combinatorial DNA or RNA libraries. This operation can also be viewed as modelling a special case of an experimental procedure called cross-pairing PCR, introduced in [15] and studied in, e.g., [14, 16, 17, 35].

Conceptually, the study of overlap assembly as a formal language operation is part of a larger effort of formalizing DNA processes as computations, which dates back to 1987 when Tom Head proposed *splicing* as a formal language operation that models the recombination of DNA strands under the cut-and-paste action of restriction enzymes and ligases. Various types of splicing systems have been defined and their properties were studied in, e.g., [18, 19, 27, 31, 43]. Other bio-operations include *insertions* and *deletions* of strands, which are basic processes in RNA editing in molecular biology: based on these, *insertion-deletion systems* were defined as formal models of computation and have been widely studied, see, e.g., [9, 29, 30, 45, 46, 48, 49]. Another example of a bio-inspired operation is a type of *substitution* operation that models errors occurring in DNA-encoded information, and that was proposed in [28]. *Hairpin formation* is a naturally occurring phenomenon whereby a DNA strand that is partially self-complementary attaches to itself. Based on this phenomenon, the formal language operation called hairpin completion as well as its inverse operation called hairpin reduction have been defined and extensively studied, see [5, 32, 36, 37]. In the context of studies of cellular computing, the operations of contextual intra- and inter-molecular recombinations were proposed in [26, 33], the operations of loop, direct-repeat excision (ld), hairpin, inverted-repeat excision/reinsertion (hi) and double loop, alternating direct-repeat excision-reinsertion (dlad) were proposed in [11, 44], and the template-guided recombination was introduced in [2], as models for gene assembly in ciliates. Lastly, in [12], a language operation called *directed extension* was proposed, that models the enzymatic activity of the DNA Polymerase enzyme. The activity of DNA Polymerase presupposes the existence of a DNA single strand called *template*, and of a second short DNA strand called *primer*, that is Watson-Crick complementary to the template and binds to it. Given a supply of individual nucleotides, DNA polymerase then extends the primer, at one of its ends only, by adding invididual nucleotides complementary to the template nucleotides, one by one, until the end of the template is reached. Experimentally, the iteration of this process is used to obtain an exponential replication of DNA strands, in a protocol called *Polymerase Chain Reaction (PCR)*.

Among operations related to overlap assembly we cite the *superposition* operation, which was studied in [3, 38]. Superposition extends DNA strands in both directions, assuming the existence of Okazaki fragments in the solution. Another related operation, called *overlapping concatenation* was introduced as part of a study of tissue P systems, [39], that was designed to solve the shortest common superstring problem efficiently [34]. The overlapping concatenation between two words returns the longer word if it contains the other word as an infix, and otherwise returns the shortest string which contains the first word as a prefix and the second word as a suffix. Lastly, an operation called *conditional concatenation* was introduced in [10]: the conditional concatenation of two words returns their concatenation only when among their substrings (scattered substrings, of various forms) one can find a pair in a given control set.

This paper, which is a theoretical analysis of overlap assembly as a formal language operation, is organized as follows. Section 2 contains definitions and notations, including the definition of overlap assembly. In Sections 3, 4 we prove closure properties of various language classes under overlap assembly and investigate related decision problems. In Section 5, we investigate the iterated overlap assembly and demonstrate that, in theory, it can be an effective tool to generate a DNA combinatorial library.

2 Basic definitions and notations

An alphabet Σ is a finite non-empty set of symbols. Σ^* denotes the set of all words over Σ, including the empty word λ. Σ^+ is the set of all non-empty words over Σ. For words w, x, y, z such that $w = xyz$ we call the subwords x, y, and z *prefix*, *infix*, and *suffix* of w, respectively. The sets $\mathit{Pref}(w)$, $\mathit{Inf}(w)$, and $\mathit{Suff}(w)$ contain, respectively, all proper prefixes, infixes, and suffixes of w. By *proper*, we mean that the sets do not include the word w itself. This notation is extended to languages as follows: $\mathit{Suff}(L) = \bigcup_{w \in L} \mathit{Suff}(w)$. The complement of a language $L \subseteq \Sigma^*$ is $L^c = \Sigma^* \setminus L$.

An *involution* is a function $\theta : \Sigma^* \to \Sigma^*$ with the property that θ^2 is the identity. θ is called an *antimorphism* if $\theta(uv) = \theta(v)\theta(u)$. Traditionally, the Watson-Crick complementarity of DNA strands has been modeled as an antimorphic involution over the DNA alphabet $\Delta = \{A, C, G, T\}$.

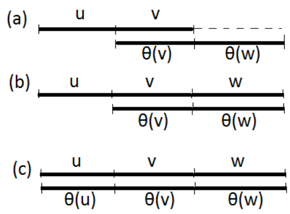

Fig. 1 (a) The two input DNA single-strands, uv and $\theta(w)\theta(v)$ bind to each other through their complementary segments v and $\theta(v)$, forming a partially double-stranded DNA complex. (b) DNA Polymerase extends the 3' end of the strand uv. (c) DNA polymerase extends the 3' end of the other strand. The resulting DNA double strand is considered to be the output of the *overlap assembly* of the two input single strands.

Using the convention that a word x over this alphabet represents the DNA single strand x in the 5' to 3' direction, the overlap assembly of a strand uv with a strand $\theta(w)\theta(v)$ first forms a partially double-stranded DNA molecule with v in uv and $\theta(v)$ in $\theta(w)\theta(v)$ attached to each other, see Figure 1(a). The DNA Polymerase enzyme will extend the 3' end of uv with the strand w, see Figure 1(b). Similarly, the 3' end of $\theta(w)\theta(v)$ will be extended, resulting in a full double strand whose upper strand is uvw, see Figure 1(c). Formally, the overlap assembly between uv and $\theta(w)\theta(v)$ is uvw. Assuming that all involved DNA strands are initially double-stranded, that is, whenever the strand x is available, its Watson-Crick complement $\theta(x)$ is also available, this model can be simplified as follows: Given two words x, y over an alphabet Σ, the *overlap assembly of x with y* is defined as, [7],

$$x \odot y = \{z \in \Sigma^+ \mid \exists u, w \in \Sigma^*, \exists v \in \Sigma^+ : x = uv, y = vw; z = uvw\}$$

The definition of overlap assembly can be extended to languages in the natural way. Note that, for a realistic model, we would need additional restrictions such as the fact that the "overlap" v should be of a sufficient length for the Watson-Crick pairing to happen, and should also not appear as a substring in other strings involved. In this paper, however, we do not invoke any of these restrictions.

A similar operation, the *superposition*, has been proposed by Bottoni et al. [3]. The result of the superposition operation between words $x, y \in \Sigma^+$, denoted by $x \diamond y$, consists of the set of all words $z \in \Sigma^+$ obtained by any of the four following cases ($\bar{\ }$ denotes the *morphic complement*, i.e., $\bar{\ }$ is a mapping such that $\overline{uv} = \bar{u}\bar{v}$ and $\bar{\bar{u}} = u$ for all words u, v):

1. If there exist $u, v \in \Sigma^*, w \in \Sigma^+$ such that $x = uw, y = \bar{w}v$, then $z = uw\bar{v} \in x \diamond_1 y$.
2. If there exist $u, v \in \Sigma^*$ such that $x = u\bar{y}v$, then $z = u\bar{y}v \in x \diamond_2 y$.
3. If there exist $u, v \in \Sigma^*, w \in \Sigma^+$ such that $x = wv, y = u\bar{w}$, then $z = \bar{u}wv \in x \diamond_3 y$.
4. If there exist $u, v \in \Sigma^*$ such that $y = u\bar{x}v$, then $z = \bar{u}x\bar{v} \in x \diamond_4 y$.

As before, the superposition is naturally extended to languages. The superposition operation and the overlap assembly are closely related. In particular, when we replace the complement $\bar{\ }$ by the identity, then case 1 is identical to the overlap assembly $x \overline{\odot} y = x \diamond_1 y$; case 3 is symmetrical to the overlap assembly $x \overline{\odot} y = y \diamond_3 x$; furthermore, cases 2 and 4 give $x \diamond_2 y = y \diamond_4 x = x$ if y is an infix of x. From this observation, it easily follows that when we consider the overlap assembly of one language L by itself, we have $L \overline{\odot} L = L \diamond L$. However, in the general case of two languages or when we consider a "real" complement function, the overlap assembly $L_x \overline{\odot} L_y$ does not give the same result as the superposition $L_x \diamond L_y$.

We will use the following notations: NPDA for nondeterministic pushdown automaton; DPDA for deterministic pushdown automaton; NCA for an NPDA that uses only one stack symbol in addition to the bottom of the stack symbol, which is never altered; DCA for deterministic NCA; NFA for nondeterministic finite automaton; DFA for deterministic finite automaton; NLBA for nondeterministic linear-bounded automaton; DLBA for deterministic linear-bounded automaton; NTM for nondeterministic Turing machine; DTM for deterministic Turing machine. As is well-known, NFAs, NPDAs, NLBAs, halting DTMs, and DTMs, accept exactly the regular languages, context-free languages (CFLs), context-sensitive languages (CSLs), recursive languages, and recursively enumerable languages. We refer the reader to [20] for the formal definitions of these devices.

A *counter* is an integer variable that can be incremented by 1, decremented by 1, left unchanged, and tested for zero. It starts at zero and cannot store negative values. Thus, a counter is a pushdown stack on unary alphabet, in addition to the bottom of the stack symbol which is never altered.

An automaton (NFA, NPDA, NCA, etc.) can be augmented with a finite number of counters, where the "move" of the machine also now depends on the status (zero or non-zero) of the counters, and the move can update the counters. It is well known that a DFA augmented with two counters is equivalent to a DTM [41].

In this paper, we will restrict the augmented counter(s) to be reversal-bounded in the sense that each counter can only reverse (i.e., change mode from nondecreasing to non-increasing and vice-versa) at most r times for some given r. In particular, when $r = 1$, the counter reverses only once, i.e., once it decrements, it can no longer increment. Note that a counter that makes r reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ 1-reversal counters. Closure and decidable properties of various machines augmented with reversal-bounded counters have been studied in the literature (see, e.g., [21, 22]). We will use the notation NFCM, NPCM, NCM, etc, to denote an NFA, NPDA, NCA, etc., augmented with reversal-bounded counters.

Example 1. $L = \{xx^r \mid x \in (a+b)^+, |x|_a = |x|_b\}$ can be accepted by an NPCM M with two 1-reversal counters. (The notation $|x|_a$ denotes the number of a's in the string x.) Note that L is not a CFL.

Briefly, M operates as follows: It scans the input and uses the pushdown stack to check that the input is a palindrome (this rquires M to "guess" the middle of the string) while using two counters C_1 and C_2 to store the numbers of a's and b's it encounters. Then, at the end of the input, on λ-transitions (i.e., without reading any input symbol), M decrements C_1 and C_2 simultaneusly and verifies that they become zero at the same time. Note that the counters are 1-reversal.

Example 2. $L_k = \{x_1\#\cdots\#x_k \mid x_i \in (a+b)^+, x_j \neq x_k \text{ for } j \neq k\}$ can be accepted by an NFCM M_k with $k(k+1)/2$ 1-reversal counters.

M_k operates as follows: It reads the input and verifies that for $1 \leq i < j \leq k$, x_i and x_j disagree in at least one position. To accomplish this, while scanning x_i, M_k stores in counter C_i a "guessed" position p_i of x_i and records in the state the symbol a_{p_i} in that location. Then later, when it is scanning x_j, M_k stores in counter C_j a guessed location p_j of x_j and records in the state the symbol a_{p_j} in that location. At the end of the input, on λ-transitions, M_k checks that $a_{p_i} \neq a_{p_j}$ and $p_i = p_j$ (by decrementing counters C_i and C_j simultaneously and confirming that they become zero at the sanme time).

3 Closure properties

In this section we study closure properties of various language classes under overlap assembly. We begin with the following general result.

Theorem 1 *Let $\mathscr{A}$ and $\mathscr{B}$ be two families of languages satisfying the following properties, where # is a symbol not in Σ:*

1. *If $L_x \subseteq \Sigma^*$ is in $\mathscr{A}$ and $L_y \subseteq \Sigma^*$ is in $\mathscr{B}$, then:*
   $L_x^\# = \{u\#v \mid |v| > 0, uv \in L_x\}$ is in $\mathscr{A}$, and
   $L_y^\# = \{v\#w \mid |v| > 0, vw \in L_y\}$ is in $\mathscr{B}$.
2. *If $L_1 \subseteq \Sigma^*$ is in $\mathscr{A}$, then $L_1\#\Sigma^*$ is in $\mathscr{A}$.*
   If $L_2 \subseteq \Sigma^*$ is in $\mathscr{B}$, then $\Sigma^*\#L_2$ is in $\mathscr{B}$.
3. *$\mathscr{A}$ is closed under intersection with languages in $\mathscr{B}$.*
4. *If $L \subseteq \Sigma^*\#\Sigma^+\#\Sigma^*$ is in $\mathscr{A}$ and h is a homomorphism that maps # to λ (the empty word) and leaves all other symbols unchanged, then $h(L)$ is in $\mathscr{A}$.*

Then $\mathscr{A}$ is closed under overlap assembly with $\mathscr{B}$, i.e., for any $L_x \in \mathscr{A}$ and $L_y \in \mathscr{B}$, $L_x \overline{\odot} L_y$ is in $\mathscr{A}$.

Proof Let $L_x, L_y \subseteq \Sigma^*$ be in $\mathscr{A}$ and $\mathscr{B}$, respectively. Let # be a symbol not in Σ. Then by (1), $L_x^\#$ is in $\mathscr{A}$ and $L_y^\#$ is in $\mathscr{B}$. Then by (2), $L_x^\#\#\Sigma^*$ is in $\mathscr{A}$ and $\Sigma^*\#L_y^\#$ in $\mathscr{B}$. Since $\mathscr{A}$ is closed under intersection with languages in $\mathscr{B}$ by (3), $L_x^\#\#\Sigma^* \cap \Sigma^*\#L_y^\#$ is in $\mathscr{A}$. Finally, from (4), $L_x \overline{\odot} L_y = h(L_x^\#\#\Sigma^* \cap \Sigma^*\#L_y^\#)$ is in $\mathscr{A}$. □

A symmetric theorem also holds when the roles of $\mathscr{A}$ and $\mathscr{B}$ in above theorem are switched.

Corollary 1 *The families of regular languages, context-sensitive languages, recursive languages, recursively enumerable languages, and NFCM languages are closed under overlap assembly.*

Proof Consider the case $\mathscr{A} = \mathscr{B}$. It is known or easily verified that the families above satisfy the properties in Theorem 1. In fact, for each family, one can effectively construct the machines satisfying the closure properties listed in the theorem. See, e.g., [20, 21]. □

Corollary 2

1. *If L_x is regular (resp., context-free, context-sensitive, recursive, recursively enumerable) and L_y is regular, then is $L_x \overline{\odot} L_y$ is regular (resp., context-free, context-sensitive, recursive, recursively enumerable).*
2. *If L_x is regular and L_y is regular (resp., context-free, context-sensitive, recursive, recursively enumerable), then $L_x \overline{\odot} L_y$ is regular (resp., context-free, context-sensitive, recursive, recursively enumerable).*

Proof Part 1 follows from Theorem 1. Part 2 follows from the symmetric version of Theorem 1 with the roles of $\mathscr{A}$ and $\mathscr{B}$ switched. □

Corollary 3 *If one of L_x and L_y is accepted by an NPCM and the other is accepted by an NFCM, then $L_x \overline{\odot} L_y$ is accepted by an NPCM.*

Proof This follows from Theorem 1 and its symmetric version by taking $\mathscr{A}$ to be the class of NPCM languages and $\mathscr{B}$ to be the class of NFCM languages. □

DSPACE($S(n)$) (resp., NSPACE($S(n)$)) denotes the family of languages accepted by $S(n)$ space-bounded DTMs (resp., NTMs). PTIME denotes the family of languages accepted by polynomial time-bounded DTMs.

Theorem 2 *Let L_x and L_y be CFLs (i.e., accepted by NPDAs). Then*

1. *$L_x \overline{\odot} L_y$ is in DSPACE$((\log n)^2)$.*
2. *$L_x \overline{\odot} L_y$ is in PTIME.*

Proof Let $L_x, L_y \subseteq \Sigma^*$ be languages. It is known that CFLs can be accepted by DTMs in $(\log n)^2$ space, i.e., they are in *DSPACE*$((\log n)^2)$. So let M_x and M_y be $(\log n)^2$ space-bounded DTMs that accept L_x and L_y, respectively. We construct a $(\log n)^2$ space-bounded DTM M accepting $L_x \overline{\odot} L_y$ as follows. Given input z of length n , M needs to determine if there is a partition $z = uvw$ for some $u, w \in \Sigma^*$ and $v \in \Sigma^+$ such that $|v| > 0$, $uv \in L_x$ and $vw \in L_y$. To do this, M needs two counters to record the positions i and j where v begins and ends. These counters need $\log n$ space to implement on the DTM. M can systematically examine all possible values of $1 \le i \le j \le n$ to see if for some $i \le j$, uv is accepted by M_x and vw is accepted by M_y. Clearly, M operates in $(\log n)^2$ space.

The construction for Part 2 follows from Part 1 by noting that CFLs are in PTIME. □

Corollary 4 *If L_x and L_y are CFLs, then $L_x \overline{\odot} L_y$ is a DCSL (deterministic CSL), but not necessarily a CFL.*

Proof That $L_x \overline{\odot} L_y$ is a DCSL follows from Theorem 2 and the observation that *DSPACE*$((\log n)^2)$ is properly contained in DSPACE(n)(the family of DCSLs). Now let

$$L_x = \{\#a^m b^m c^n \$ \mid m, n \ge 1\}$$
$$L_y = \{\#a^m b^n c^m \$ \mid m, n \ge 1\}$$

Clearly, L_x and L_y are LCFLs. In fact, they can be accepted by DCAs that make only one reversal on the counters. However, $L_x \overline{\odot} L_y = \{\#a^m b^m c^m \$ \mid m \ge 1\}$ is not CF. □

The ideas in the proof of Theorem 2 can be used to show the following:

Corollary 5 *The space classes* NSPACE(S) *and* DSPACE(S) *are each closed under overlap assembly for any space bound $S(n) \ge \log n$.*

As stated in Corollary 1, the family of NFCM languages is closed under overlap assembly. We give another proof below as the construction is needed later. For easy reference, since Corollary 1 includes other families, we restate the result for NFCM only, in the theorem below.

Theorem 3 *The family of languages accepted by NFCMs is closed under overlap assembly.*

Proof Let L_x and L_y be accepted by NFCMs M_x and M_y, respectively. We construct an NFCM M to accept $L_x \overline{\odot} L_y$ as in the proof of of Theorem 2. The only change is that when given input z, M guesses the beginning and end locations i and j of v in the partition $z = uvw$. M simulates M_x on the prefix of z that ends in position j (i.e., on uv) and starts simulating M_y starting in position i of the input z. M accepts if both M_x and M_y accepts. Note that if M_x and M_y have k_1 and k_2 reversal-bounded counters, respectively, then M will have $k_1 + k_2$ reversal-bounded counters. □

Let $\mathbb{N}$ be the set of non-negative integers and k be a positive integer. A subset Q of $\mathbb{N}^k$ is a *linear set* if there exist vectors $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{N}^k$ such that $Q = \{\mathbf{v}_0 + i_1\mathbf{v}_1 + \cdots + i_n\mathbf{v}_n \mid i_1, \ldots, i_n \in \mathbb{N}\}$. A finite union of linear sets is called a *semilinear set*.

A bounded language $L \subseteq w_1^* \cdots w_k^*$ (for some $k \geq 1$ and non-null words $w_1, \ldots, w_k$) is semilinear if there is a semilinear set $Q \subseteq \mathbb{N}^k$ such that $L = \{w_1^{i_1} \cdots w_k^{i_k} \mid (i_1, \ldots, i_k) \in Q\}$.

Corollary 6 *The family of semilinear languages is closed under overlap assembly.*

Proof It is known that a bounded language L (i.e., $\subseteq w_1^* \cdots w_k^*$ for some $k \geq 1$ and words $w_1, \ldots, w_k$) is semilinear if and only if it can be accepted by an NFCM [21]. The result follows from Theorem 3. □

Corollary 7 *The family of bounded languages accepted by DFCMs (i.e., DFAs augmented with reversal-bounded counters) is closed under overlap assembly.*

Proof This follows from Corollary 6 and the fact that every NFCM accepting a bounded language can be converted to an equivalent DFCM [23]. □

Finally, we consider the family of languages accepted by visibly pushdown automata. A visibly pushdown automaton (VPDA) [1], also known as input-driven pushdown automaton [40], is a restricted version of an NPDA. It is an NPDA where the input symbol determines the (push/stack) operation of the stack. It has a distinguished symbol $\perp$ at the bottom of the stack which is never altered or occur anywhere else. The input alphabet Σ is partitioned into three disjoint alphabets: Σ_c, Σ_r, Σ_ℓ. The machine pushes a specified symbol on the stack if it reads a *call symbol* in Σ_c on the input; it pops a specified symbol if the specified symbol is at top of the stack and it is not the bottom of the stack $\perp$ (otherwise it it does not pop $\perp$) if it reads a *return symbol* in Σ_r on the input; it does not use the (top symbol of) the stack and can only change state if it reads a *local symbol* in Σ_ℓ on the input. The partition into call, return, and local symbols is a property that is inherent to the alphabet Σ. Therefore, if two machines M_x and M_y operate on the same input alphabet Σ, then they have the same set of call, return, and local symbols, respectively.

A VPDA augmented with reversal-bounded counters is called VPCM. We allow the machine to have ε-moves, but in such moves, the stack is not used, only the state and counters are used and updated. Acceptance of an input string is when machine eventually falls off the right end of the input in an accepting state. See [22] for a formal definition.

Theorem 4 *The family of languages accepted by VPCMs is closed under overlap assembly.*

Proof The proof is similar to that of Theorem 3. In that proof, M_x and M_y are VPCMs. The VPCM M constructed from M_x and M_y needs only one pushdown stack, since the operations on the stack of these two machines (being input-driven) are synchronized, i.e., M_x pushes, pops, or leaves the stack unchanged if and only if M_y pushes, pops, or leaves the stack unchanged. □

Clearly, if both M_x and M_y are VPDAs (i.e., have no reversal-bounded counters), then so is M. Hence:

Corollary 8 *The family of languages accepted by VPDAs is closed under overlap assembly.*

We summarize this section's results regarding closure properties of language classes in the Chomsky hierarchy (plus finite languages) under overlap assembly in Table 1. For two language classes $\mathscr{X}$ and $\mathscr{Y}$, the intersection of row $\mathscr{X}$ with column $\mathscr{Y}$ shows the language class $\mathscr{Z}$ from the Chomsky hierarchy such that for all $L_x \in \mathscr{X}$ and $L_y \in \mathscr{Y}$ we have $L_x \overline{\odot} L_y \in \mathscr{Z}$. Noting that $FIN \subseteq REG \subseteq CF \subseteq CS \subseteq RE$ (modulo the condition that λ is not allowed in CS languages), all the entries in Table 1 (except for the case when L_x and L_y are finite) follow from Corollary 1. The case when $L_x \in FIN$ and $L_y \in FIN$, the result is in FIN is obvious.

Also note that each entry in the table is the smallest class from the Chomsky hierarchy which includes $L_x \overline{\odot} L_y$ for all $L_x \in \mathscr{X}$ and $L_y \in \mathscr{Y}$. This follows from Corollary 4 and the following observation: For a language $L \subseteq \Sigma^*$ and a symbol $\$ \notin \Sigma$, the languages $\$L$ and $L\$$ belong to the same classes in the Chomsky hierarchy as L. Furthermore, $L\$ \overline{\odot} \{\$\} = L\$$ and $\{\$\} \overline{\odot} \$L = \$L$.

$\mathscr{X} \setminus \mathscr{Y}$	FIN	REG	CF	CS	RE
FIN	FIN	REG	CF	CS	RE
REG	REG	REG	CF	CS	RE
CF	CF	CF	CS	CS	RE
CS	CS	CS	CS	CS	RE
RE	RE	RE	RE	RE	RE

Table 1 Closure properties of language classes in the Chomsky hierarchy under overlap assembly.

4 Decision problems

We have seen in Corollary 4 that the families of context-free languages (CFLs) and linear context-free languages (LCFLs) are not closed under overlap assembly. We will show that it is undecidable whether or not the overlap assembly of two CFLs (resp., LCFLs) is a CFL (resp., LCFL).

An NPDA (resp., DPDA) is 1-reversal if its stack makes only one reversal, i.e., once it pops, it can no longer push. It is well-known that 1-reversal NPDAs accept exactly the LCFLs. In the following theorems, "DCAs" always means a general DCA, i.e., there is no restriction on counter reversals.

Theorem 5 *It is undecidable, given 1-reversal DPDAs (resp., DCAs) M_x and M_y accepting languages L_x and L_y, respectively, whether $L_x \overline{\odot} L_y$ is a CFL or not.*

Proof Let $L_1, L_2 \subseteq \Sigma^*$ be accepted by 1-reversal DPDAs. Let $a, b, c, \#, \$$ be new symbols. Define the following languages:

$$L_x = \{\#a^m w b^m c^n \$ \mid m, n \geq 1, w \in L_1\}$$
$$L_y = \{\#a^m w b^n c^m \$ \mid m, n \geq 1, w \in L_2\}$$

It is easily verified that L_x and L_y can also be accepted by 1-reversal DPDAs. Then $L = L_x \overline{\odot} L_y = L_x \cap L_y$. Clearly, $L = \emptyset$ if and only if $L_1 \cap L_2 = \emptyset$. Now if $L = \emptyset$, then it is obviously a CFL. If $L \neq \emptyset$, we claim that it is not a CFL. For suppose L is a CFL. Apply a homomorphism that maps all symbols in Σ to λ (the empty word) and leaves all other symbols unchanged. Then the resulting language, L', must also be context-free, since CFLs are closed under homomorphism. We get a contradiction, since $L' = \{\#a^m b^m c^m \$ \mid m \geq 1\}$ is not context-free. The result now follows, since the emptiness of intersection of two languages accepted by 1-reversal DPDAs is undecidable [20].

If $L_1, L_2 \subseteq \Sigma^*$ are accepted by DCAs, define the languages:

$$L_x = \{\#w a^m b^m c^n \$ \mid m, n \geq 1, w \in L_1\}$$
$$L_y = \{\#w a^m b^n c^m \$ \mid m, n \geq 1, w \in L_2\}$$

Note that L_x and L_y can be accepted by DCAs as well. Using the same arguments as before, $L_x \overline{\odot} L_y$ is context-free if and only if $L_1 \cap L_2 = \emptyset$. However, the emptiness of intersection of two languages accepted by DCAs is undecidable [21]. $\square$

We need the notion of Parikh map of a language in the proof of the next result. Let $\Sigma = \{a_1, \ldots, a_k\}$. The Parikh map of a language $L \subseteq \Sigma^*$ is defined as

$$\{(|w|_{a_1}, \ldots, |w|_{a_k}) \mid w \in L\},$$

where $|w|_{a_i}$ is the number of a_i's in w.

Theorem 6 *It is undecidable, given 1-reversal DPDAs (resp., DCAs) M_x and M_y accepting languages L_x and L_y, respectively, whether $L_x \overline{\odot} L_y$ can be accepted by an NFCM.*

Proof Let $L_1, L_2 \subseteq \Sigma^*$ be accepted by 1-reversal DPDAs, and $a, b, c, \#, \$$ be new symbols. Define the following languages:

$$L_x = \{\#zwcz^R\$ \mid z \in (a+b)^+, w \in L_1\}$$
$$L_y = \{\#zwcz^R\$ \mid z \in (a+b)^+, w \in L_2\}$$

It is easily verified that L_x and L_y can be accepted by 1-reversal DPDAs. Then $L = L_x \overline{\odot} L_y = L_x \cap L_y$. If $L = \emptyset$, then it is obvious that it can be accepted by an NFCM. If $L \neq \emptyset$ and is accepted by an NFCM, then we can construct another NFCM that accepts the language, L', obtained by applying a homomorphism that maps all symbols in Σ to λ and leaves all other symbols unchanged. Clearly, $L' = \{\#zcz^R\$ \mid z \in (a+b)^+\}$. But it is known that L' cannot be accepted by an NFCM [6]. It follows that L cannot be accepted by an NFCM. Hence, it is undecidable whether $L_x \overline{\odot} L_y$ can be accepted by an NFCM.

For the second part, let $L_1, L_2 \subseteq \Sigma^*$ be accepted by DCAs, and $a, b, \#, \$$ be new symbols. Define the following languages:

$$L_x = \{\#a^{i_1} b a^{i_1+1} b a^{i_2} b a^{i_2+1} \cdots a^{i_k} b a^{i_k+1} w\$ \mid k \geq 1, i_1, \cdots, i_k \geq 1, i_1 = 1, w \in L_1\}$$
$$L_y = \{\#a^{j_1} b a^{j_2} b a^{j_2+1} \cdots a^{j_{k-1}} b a^{j_{k-1}+1} b a^{j_k} w \mid k \geq 1, j_1, \cdots, j_k \geq 1, j_1 = 1, w \in L_2\}$$

Then $L_x \overline{\odot} L_y = \{\#a^1 b a^2 b a^3 b a^4 \cdots a^{2k-1} b a^{2k} w\$ \mid k \geq 1, w \in L_1 \cap L_2\}$. Hence, $L_x \overline{\odot} L_y = \emptyset$ if and only if $L_1 \cap L_2 = \emptyset$. Suppose $L_x \overline{\odot} L_y \neq \emptyset$. One can verify that the Parikh map of if $L_x \overline{\odot} L_y \neq \emptyset$ is not a semilinear set. Since the Parikh map of any NFCM language is semilinear [21], it follows that if $L_x \overline{\odot} L_y \neq \emptyset$, it cannot be accepted by an NFCM. We conclude that $L_x \overline{\odot} L_y$ is accepted by an NFCM if and only if $L_1 \cap L_2 = \emptyset$, which is undecidable. $\square$

Another interesting decision question is to decide, whether $L_x \overline{\odot} L_y$ is empty, finite, or infinite.

Theorem 7

1. *It is decidable, given L_x and L_y, one of which is accepted by an NPCM and the other by an NFCM, whether $L_x \overline{\odot} L_y$ is empty, finite, or infinite.*
2. *It is decidable, given L_x and L_y, accepted by VPCMs, whether $L_x \overline{\odot} L_y$ is empty, finite, or infinite.*

Proof This follows from Corollary 3 and Theorem 4 and the fact that it is decidable, given an NPCM, whether the language it accepts is empty, finite, or infinite. $\square$

We end this section with a discussion of a special case of overlap assembly, when the languages L_x and L_y are the same. More precisely, if $L \subseteq \Sigma^*$, let

$$L_{\overline{\odot}} = L \,\overline{\odot}\, L = \{uvw \mid v \in \Sigma^+, u, w \in \Sigma^*, uv, vw \in L\}.$$

Obviously, the positive closure and decidable results in the previous section and this section when the class $\mathscr{A}$ = class $\mathscr{B}$ also hold for this special case of overlap assembly (by taking $L_y = L_x$). However the proofs for the non-closure and undecidable results need to be modified.

Theorem 8 *If L is accepted by a DCA, then $L_{\overline{\odot}}$ need not be a CFL.*

Proof Let $L = \{\%a^m\#b^m c^n \mid m,n \geq 1\} \cup \{\#b^m c^m\$ \mid m \geq 1\}$. Clearly, L can be accepted by a DCA that makes only one reversal on its counter.

Suppose $L_{\overline{\odot}}$ is a CFL. Define the regular language $L' = \%a^+\#b^+c^+\$$. Then the language $L'' = L_{\overline{\odot}} \cap L'$ must also be a CFL. We get a contradiction since $L'' = \{\%a^m\#b^m c^m\$ \mid m \geq 1\}$ is not a CFL. □

Theorem 9 *It is undecidable, given a language L accepted by a 1-reversal DPDA (resp., DCA) M, whether $L_{\overline{\odot}}$ is a CFL.*

Proof Let $L_1, L_2 \subseteq \Sigma^*$ be accepted by 1-reversal DPDAs. Define

$$L = \{\%a^m\#b^n w c^m\$ \mid m,n \geq 1, w \in L_1\} \cup \{\#b^m w c^m\$\$ \mid m \geq 1, w \in L_2\}.$$

It can be verified that L can be accepted by a 1-reversal DPDA. Then by an argument similar to that in the proof of Theorem 5, $L_{\overline{\odot}}$ is a CFL if and only if $L_1 \cap L_2 = \emptyset$, which is undecidable.

Now, let $L_1, L_2 \subseteq \Sigma^*$ be accepted by DCAs. Define

$$L = \{\%a^m\#b^n c^m w\$ \mid m,n \geq 1, w \in L_1\} \cup \{\#b^m c^m w\$\$ \mid m \geq 1, w \in L_2\}.$$

It can be verified that L can be accepted by a DCA. By an argument similar to that in the proof of Theorem 5, $L_{\overline{\odot}}$ is a CFL if and only if $L_1 \cap L_2 = \emptyset$, which is undecidable. □

5 Iterated overlap assembly

We define a combinatorial library of words as a set of the form

$$\{\alpha_1\alpha_2 \cdots \alpha_n \mid \alpha_i \in \{X_i, Y_i\} \text{ for } i = 1, \ldots, n\}$$

where $X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_n \in \Sigma^+$ are distinct sequences. It is often required that all X_i and Y_i are of the same length. However, some experiments use the fact that X_i and Y_i have different lengths: for example, in [42] all X_i have the same length which is shorter than the length of all Y_i, thus allowing to use gel electrophoresis to separate the strings from this library by how many X_i they contain.

Combinatorial libraries of DNA strands have applications in many areas, including DNA computing where, e.g., a *mix-and-split* procedure was used to generate the solution space (a combinatorial library of binary numbers) for a chess problem, [13]. A similar technique was used to generate the pool of solutions to a 20-variable solution of the 3-SAT problem, [4], the largest experiment to date that solved a computational problem with a DNA algorithm. Efficient generation of combinatorial libraries of this type, obtained by using XPCR, was initially proposed in [14], and further investigated in [16].

In this section we formally prove that the iterated overlap assembly can theoretically generate this library with some restrictions on the words X_i, Y_i. We consider the following library where an additional symbol \$ is inserted between every pair of X_i/Y_i and X_{i+1}/Y_{i+1}:

$$\{\alpha_1\$\alpha_2\$ \cdots \alpha_n\$ \mid \alpha_i \in \{X_i, Y_i\} \text{ for } i = 1 \ldots, n\}. \tag{1}$$

For simplicity, we view \$ as an additional letter that does not appear inside any of the words X_i or Y_i. The purpose of introducing the letters \$ is that each letter \$ has to match the position of another letter \$ during overlap assembly (i.e., no proper suffix of $\alpha_i\$$ is identical to a proper prefix of $\alpha_j\$$). If one prefers to avoid the introduction of this additional letter in the strings (e.g., for practical purposes), it is sufficient to design the set of strings

$$C = \{X_1, \ldots, X_n, Y_1, \ldots, Y_n\}$$

such that either C contains only equal-length words that are *overlap-free* or, less restrictive, C is a *solid code* (i.e., overlap- and infix-free), see e.g., [24]. In this case, the symbols \$ in the library (1) become markers (of width 0) which match during overlap assembly because of the design of the set C.

We start by generalizing the definition of $L_{\overline{\odot}} = L \,\overline{\odot}\, L$. The *iterated overlap assembly* of a language L, [7], is defined as follows:

$$\mu_0(L) = L \qquad \mu_{i+1}(L) = \mu_i(L) \,\overline{\odot}\, \mu_i(L)$$

$$\mu_*(L) = \bigcup_{i \geq 0} \mu_i(L)$$

In particular $\mu_1(L) = L \,\overline{\odot}\, L = L_{\overline{\odot}}$. Since $w \in w \,\overline{\odot}\, w$ for any nonempty word w, from the definition it easily follows that $\mu_i(L) \subseteq \mu_{i+1}(L)$ for $L \in \Sigma^+$. It can be shown (using intersections with appropriate regular languages) that Theorems 8 and 9 also hold for iterated overlap assembly.

We will now show that we can generate the combinatorial library (1) by *(i)* starting with a set of strands

$$\{\alpha_k\$\alpha_{k+1}\$ \mid 1 \leq k \leq n-1, \alpha_i \in \{X_i, Y_i\} \text{ for } i = 1, \ldots, n\},$$

(ii) iteratedly applying overlap assembly until no new strands are produced anymore (Theorem 11), and *(iii)* extracting the longest strands from the result. We will also show (Theorem 10) that the number of steps of this process is logarithmic in the size of the input.

Definition 1 A string $x \in L$ is said to be *terminal* with respect to language L if $x \overline{\odot} L = L \overline{\odot} x = \{x\}$.

Definition 2 A set of strings $T(L) \subseteq L$ is said to be *the maximal terminating set* of L if every $w \in T(L)$ is terminal with respect to L and for all $w \in L \backslash T(L)$, w is not terminal with respect to L, that is,

$$T(L) = \{w \in L \mid w \overline{\odot} L = L \overline{\odot} w = \{w\}\}$$

Lemma 1 *If $t \in T(L)$, then $t \in T(\mu_1(L))$. More generally, if $t \in T(L)$, then $t \in T(\mu_*(L))$*

Proof We prove the contrapositive: if $t \notin T(L \overline{\odot} L)$, then $t \notin T(L)$. There exists $w \in \mu_1(L)$ and $u \neq t$ such that either $u \in w \overline{\odot} t$ or $u \in t \overline{\odot} w$. If $w \in L$, then $t \notin T(L)$. Thus, $w \in \mu_1(L) \backslash L$. There are $w_1, w_3 \in \Sigma^*, w_2 \in \Sigma^+$ such that $w = w_1 w_2 w_3 \in w_1 w_2 \overline{\odot} w_2 w_3$ where $w_1 w_2, w_2 w_3 \in L$. If $u \in t \overline{\odot} w$, there are $u_1, u_3 \in \Sigma^*, u_2 \in \Sigma^+$ such that $u = u_1 u_2 u_3$, where $u = u_1 u_2$ and $w = u_2 u_3 = w_1 w_2 w_3$. There are two cases possible: (A) either u_2 is a proper prefix of $w_1 w_2$, or (B) $w_1 w_2$ is a prefix of u_2.

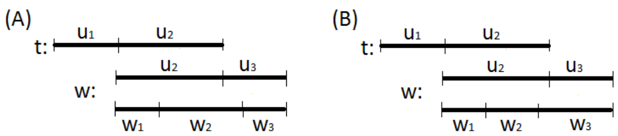

Fig. 2 Illustration of cases (A) and (B) from the proof of Lemma 1.

In case (A), there is $u_1 w_1 w_2 \neq t$ in $t \overline{\odot} w_1 w_2 \subseteq t \overline{\odot} L$ and therefore $t \notin T(L)$. In case (B), there is $u \in t \overline{\odot} w_2 w_3 \subseteq t \overline{\odot} L$ and therefore $t \notin T(L)$ because $w_2 \neq \varepsilon$. We can similarly prove that $t \notin T(L)$ when $w \in \mu_1(L)$ and $u \neq t$ exists such that $u \in w \overline{\odot} t$. Hence, we prove that $t \in T(L)$ implies $t \in T(\mu_1(L))$. By applying this result recursively, we can similarly prove that if $t \in T(L)$, then $t \in T(\mu_*(L))$. □

Definition 3 We define z_{k_1,k_2} for any $k_1 \leq k_2$ as follows.

$$z_{k_1,k_2} = \{\alpha_{k_1} \$ \alpha_{k_1+1} \$ \cdots \alpha_{k_2} \$ \mid \alpha_i \in \{X_i, Y_i\}, k_1 \leq i \leq k_2]\}$$

z_{k_1,k_2} is not defined for $k_1 > k_2$.

Informally, z_{k_1,k_2} is the set of words consisting of the catenation of $k_2 - k_1 + 1$ consecutive words α, separated by dollar signs. Note that, with this notation, $z_{1,n}$ represents the required combinatorial library.

Definition 4 We define $Z(m,n)$ for all $m \geq 2$ as equal to the union of all z_{k_1,k_2} such that $1 \leq k_2 - k_1 < m$ for $m \leq n$, and equal to $Z(n,n)$ for $m > n$:

$$Z(m,n) = \begin{cases} \bigcup_{p=1,\ldots,m-1} \bigcup_{k_1=1,\ldots,n-p} z_{k_1,k_1+p} & \text{if } 2 \leq m \leq n \\ Z(n,n) & \text{if } m > n. \end{cases}$$

Informally, $Z(m,n)$ is the set of all strands consisting of at most m consecutive words α (separated by dollar signs), where $2 \leq m \leq n$. With this notation, $Z(2,n)$ represents the initial starting set, and $Z(n,n)$ contains all strands consisting of catenations of consecutive words α, with the minimum number of consecutive words α in such a catenation being 2, and the maximum number being n. Note that $Z(n,n)$ contains the desired library $z_{1,n}$ as a subset.

Lemma 2 *Let $x = \alpha_{k_1} \$ \cdots \alpha_{k_2} \$$ and $y = \beta_{l_1} \$ \cdots \beta_{l_2} \$$ be words where $\alpha_i, \beta_i \in \{X_i, Y_i\}$ and $1 \leq k_1, k_2, l_1, l_2 \leq n$. If $k_1 \leq l_1 \leq k_2 \leq l_2$ and $\alpha_i = \beta_i$ for $i = l_1, l_1+1, \ldots, k_2$, then*

$$x \overline{\odot} y = \{\alpha_{k_1} \$ \alpha_{k_1+1} \$ \cdots \alpha_{k_2} \$ \beta_{k_2+1} \$ \beta_{k_2+2} \$ \cdots \beta_{l_2} \$\}.$$

Otherwise, $x \overline{\odot} y = \emptyset$.

Proof It is easy to see that

$$\alpha_{k_1} \$ \alpha_{k_1+1} \$ \cdots \alpha_{k_2} \$ \beta_{k_2+1} \$ \beta_{k_2+2} \$ \cdots \beta_{l_2} \$ \in x \overline{\odot} y$$

if $k_1 \leq l_1 \leq k_2 \leq l_2$ and $\alpha_i = \beta_i$ for $i = l_1, l_1+1, \ldots, k_2$, because $\alpha_{l_1} \$ \alpha_{l_1+1} \$ \cdots \alpha_{k_2} \$ = \beta_{l_1} \$ \beta_{l_1+1} \$ \cdots \beta_{k_2} \$$ can serve as overlap of x and y.

The words x and y cannot overlap in any other way since the symbols \$ have to match up in both words and all words $X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_n$ are distinct. In particular, when one of the conditions $k_1 \leq l_1 \leq k_2 \leq l_2$ and $\alpha_i = \beta_i$ for $i = l_1, l_1+1, \ldots, k_2$ is not satisfied, the two words x and y cannot form an overlap at all and, therefore, $x \overline{\odot} y = \emptyset$. □

Lemma 3 *If $2 \leq m_1, m_2 \leq n$, then $Z(m_1,n) \overline{\odot} Z(m_2,n) = Z(m_1+m_2-1, n)$.*

Proof Let $x \in Z(m_1,n)$, $y \in Z(m_2,n)$ and $w \in x \overline{\odot} y$. Clearly, we have $x = \alpha_{k_1} \$ \alpha_{k_1+1} \$ \cdots \alpha_{k_2} \$$ and $y = \beta_{l_1} \$ \beta_{l_1+1} \$ \cdots \beta_{l_2} \$$ where $1 \leq k_1, k_2, l_1, l_2 \leq n$, $k_2 - k_1 < m_1$, and $l_2 - l_1 < m_2$. From Lemma 2 we obtain that $w \in x \overline{\odot} y$ is only possible if $l_1 \leq k_2$ and $w = \alpha_{k_1} \$ \alpha_{k_1+1} \$ \ldots \alpha_{k_2} \$ \beta_{k_2+1} \$ \beta_{k_2+2} \$ \ldots \beta_{l_2} \$$. This implies that $l_2 - k_1 \leq l_2 - l_1 + k_2 - k_1 < m_1 + m_2 - 1$; note that we also have $l_2 - k_1 < n$. We conclude $w \in Z(m_1 + m_2 - 1, n)$ and, more general, $Z(m_1,n) \overline{\odot} Z(m_2,n) \subseteq Z(m_1 + m_2 - 1, n)$.

Conversely, consider a word $w = \alpha_k \$ \alpha_{k+1} \cdots \alpha_l \$ \in Z(m_1 + m_2 - 1, n)$ where $1 \leq k, l \leq n$, $1 \leq l - k < \min(m_1 + m_2 - 1, n)$, and $\alpha_i \in \{X_i, Y_i\}$. If $l - k < m_1$, then $w \in Z(m_1, n)$ and $y = \alpha_{l-1} \$ \alpha_l \$ \in Z(m_2, n)$; this implies that $w \in w \overline{\odot} y \subseteq Z(m_1,n) \overline{\odot} Z(m_2,n)$. Otherwise, we let $j = k + m_1 - 1$ and note that $x = \alpha_k\$\alpha_{k+1}\cdots\alpha_j\$ \in Z(m_1,n)$. Furthermore, because $l-k < m_1+m_2-1$, we have that $l-j = l-k-m_1+1 < m_2$ which implies that $y = \alpha_j\$\alpha_{j+1}\cdots\alpha_l\$ \in Z(m_2,n)$. By Lemma 2, we have $w \in x \overline{\odot} y \subseteq Z(m_1,n) \overline{\odot} Z(m_2,n)$. □

The following theorem shows that, starting from an initial set $Z(2,n)$ we will obtain, after $\lceil \log_2(n-1) \rceil$ or more overlap catenations, the set $Z(n,n)$ which is a superset of the combinatorial library $z_{1,n}$.

Theorem 10 *For $L = Z(2,n)$ and $k \geq 0$, we have $\mu_k(L) = Z(2^k+1,n)$. Moreover, $\mu_*(L) = Z(n,n)$.*

Proof We prove the statement by induction. Clearly, the statement holds for the base case where $k = 0$ as $\mu_0(L) = L = Z(2,n)$.

Using the induction hypothesis $\mu_k(L) = Z(2^k+1,n)$ and Lemma 3, we obtain that

$$\begin{aligned}\mu_{k+1}(L) &= \mu_k(L) \overline{\odot} \mu_k(L) = Z(2^k+1,n) \overline{\odot} Z(2^k+1,n) \\ &= Z(2\cdot(2^k+1)-1,n) = Z(2^{k+1}+1,n).\end{aligned}$$

Because $Z(m,n) \subseteq Z(n,n)$ for all $m \in \mathbb{N}$, we obtain the second statement $\mu_*(L) = Z(n,n)$. □

Next, we prove the main result of this section, namely that the maximal terminal set of $\mu_*(L) = \mu_k(L)$ is the desired combinatorial library $z_{1,n}$.

Theorem 11 *For $L = Z(2,n)$ we have $T(\mu_*(L)) = z_{1,n}$.*

Proof From Theorem 10, we know that $\mu_*(L) = Z(n,n)$. First, note that for every word $w = \alpha_1\$\alpha_2\$\cdots\alpha_n\$ \in z_{1,n} \subseteq Z(n,n)$ there does not exist any word $v \in Z(n,n)$ such that w is a proper prefix or proper suffix of v. Therefore, we must have $w \overline{\odot} Z(n,n) = Z(n,n) \overline{\odot} w = \{w\}$. Thus, $z_{1,n}$ only contains words which are terminal with respect to $\mu_*(L)$.

Next, consider a word

$$w = \alpha_{k_1}\$\alpha_{k_1+1}\$\ldots\alpha_{k_2}\$ \in Z(n,n)\setminus z_{1,n}$$

where $\alpha_i \in \{X_i, Y_i\}$, $1 \leq k_1 < k_2 \leq n$, and $k_1 > 1$ or $k_2 < n$. If $k_1 > 1$, then it is easy to see that

$$w \neq X_{k_1-1}\$\alpha_{k_1}\$\alpha_{k_1+1}\cdots\alpha_{k_2}\$ \in X_{k_1-1}\$\alpha_{k_1}\$ \overline{\odot} w \subseteq Z(n,n) \overline{\odot} w.$$

Otherwise, $k_2 < n$, and we have

$$w \neq \alpha_{k_1}\$\alpha_{k_1+1}\$\cdots\alpha_{k_2}\$X_{k_2+1}\$ \in w \overline{\odot} \alpha_{k_2}\$X_{k_2+1}\$ \subseteq w \overline{\odot} Z(n,n).$$

In either case, w is not terminal with respect to $\mu_*(L)$. We conclude that $T(\mu_*(L)) = z_{1,n}$. □

Observe that the result of iterated overlap assembly applied to the initial set $Z(2,n)$ produces the set $Z(n,n)$ that contains the required library $z_{1,n}$, but it contains also other intermediate strings. One can use various techniques to extract the library $z_{1,n}$ from this solution. For example, gel electrophoresis can be used to separate strands by length, and the longest strands, which are the desired combinatorial library strands, can then be extracted.

6 Conclusions

This paper studies properties of the operation of overlap assembly, a formal language operation that models the process of linear overlap assembly of DNA strands: Two DNA strands that partially "overlap", in the sense that the suffix of one is the Watson-Crick complement of a prefix of another, can be concatenated with the aid of the DNA Polymerase enzyme. We obtain closure properties of various language classes under this operation, and discuss various decision problems. We also investigate the iterated overlap assembly and demonstrate that, under some simplifying assumptions, it can be used to generate a DNA combinatorial library.

Acknowledgements We thank Giuditta Franco for useful suggestions and discussions on experimental aspects of XPCR, as well as Sepinoud Azimi and Florin Manea for pointing out important references.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. ACM Symposium on Theory of Computing, STOC, pp. 202–211. ACM-Press (2004)
2. Angeleska, A., Jonoska, N., Saito, M., Landweber, L.F.: RNA-guided DNA assembly. Journal of Theoretical Biology **248**(4), 706–720 (2007)
3. Bottoni, P., Labella, A., Manca, V., Mitrana, V.: Superposition based on Watson-Crick-like complementarity. Theory of Computing Systems **39**(4), 503–524 (2006)
4. Braich, R.S., Chelyapov, N., Johnson, C., Rothemund, P.W.K., Adleman, L.: Solution of a 20-Variable 3-SAT Problem on a DNA Computer. Science **296**(5567), 499–502 (2002)
5. Cheptea, D., Martín-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: hairpin completion. In: Proc. Transgressive Computing, TC, pp. 216–228 (2006)
6. Chiniforooshan, E., Daley, M., Ibarra, O.H., Kari, L., Seki, S.: One-reversal counter machines and multihead automata: Revisited. Theoretical Computer Science **454**, 81–87 (2012)
7. Csuhaj-Varjú, E., Petre, I., Vaszil, G.: Self-assembly of strings and languages. Theoretical Computer Science **374**(1-3), 74–81 (2007)
8. Cukras, A.R., Faulhammer, D., Lipton, R.J., Landweber, L.F.: Chess games: a model for RNA based computation. Biosystems **52**(1-3), 35–45 (1999)
9. Daley, M., Kari, L., Gloor, G., Siromoney, R.: Circular contextual insertions/deletions with applications to biomolecular computation. In: Proc. String Processing and Information Retrieval, SPIRE, pp. 47–54 (1999)
10. Dassow, J., Martín-Vide, C., Păun, G., Rodríguez-Patón, A.: Conditional Concatenation. Fundam. Inf. **44**(4), 353–372 (2000)
11. Ehrenfeucht, A., Petre, I., Prescott, D.M., Rozenberg, G.: Circularity and other invariants of gene assembly in ciliates, p. 8197. World Scientific (2001)
12. Enaganti, S.K., Kari, L., Kopecki, S.: A Formal Language Model of DNA Polymerase Activity. Fundamenta Informaticae **138**, 179–192 (2015)
13. Faulhammer, D., Cukras, A.R., Lipton, R.J., Landweber, L.F.: Molecular computation: RNA solutions to chess problems. Proceedings of the National Academy of Sciences **97**(4), 1385–1389 (2000)
14. Franco, G.: A Polymerase Based Algorithm for SAT. In: M. Coppo, E. Lodi, G. Pinna (eds.) Theoretical Computer Science, *Lecture Notes in Computer Science*, vol. 3701, pp. 237–250. Springer Berlin Heidelberg (2005)

15. Franco, G., Giagulli, C., Laudanna, C., Manca, V.: DNA Extraction by XPCR. In: C. Ferretti, G. Mauri, C. Zandron (eds.) Proc. DNA Computing, (DNA 11), *LNCS*, vol. 3384, pp. 104–112 (2005)
16. Franco, G., Manca, V.: Algorithmic Applications of XPCR. Natural Computing **10**(2), 805–819 (2011)
17. Franco, G., Manca, V., Giagulli, C., Laudanna, C.: DNA Recombination by XPCR. In: A. Carbone, N.A. Pierce (eds.) Proc. DNA Computing, (DNA 12), *LNCS*, vol. 3892, pp. 55–66 (2006)
18. Gatterdam, R.W.: Splicing systems and regularity. Int. J. of Computer Mathematics **31**(1-2), 63–67 (1989)
19. Head, T., Pixton, D., Goode, E.: Splicing systems: regularity and below. In: M. Hagiya, A. Ohuchi (eds.) DNA Based Computers: DNA Computing, DNA 8, *LNCS*, vol. 2568, pp. 262–268 (2003)
20. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Inc. (1978)
21. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25**(1), 116–133 (1978)
22. Ibarra, O.H.: Automata with reversal-bounded counters: A survey. In: Proc. Descriptional Complexity of Formal Systems, DCFS, pp. 5–22. Springer (2014)
23. Ibarra, O.H., Seki, S.: Characterizations of bounded semilinear languages by one-way and two-way deterministic machines. Int. J. Foundations of Computer Science **23**(06), 1291–1305 (2012)
24. Jürgensen, H., Konstantinidis, S.: Codes. In: Handbook of Formal Languages, pp. 511–607. Springer (1997)
25. Kaplan, P.D., Ouyang, Q., Thaler, D.S., Libchaber, A.: Parallel Overlap Assembly for the Construction of Computational DNA Libraries. Journal of Theoretical Biology **188**(3), 333–341 (1997)
26. Kari, L., Kari, J., Landweber, L.: Reversible Molecular Computation in Ciliates. In: J. Karhumäki, H. Maurer, G. Păun, G. Rozenberg (eds.) Jewels are Forever, pp. 353–363. Springer Berlin Heidelberg (1999)
27. Kari, L., Kopecki, S.: Deciding whether a regular language is generated by a splicing system. In: D. Stefanovic, A. Turberfield (eds.) DNA Computing and Molecular Programming (DNA 18), *LNCS*, vol. 7433, pp. 98–109 (2012)
28. Kari, L., Losseva, E.: Block substitutions and their properties. Fundamenta Informaticae **73**(1-2), 165–178 (2006)
29. Kari, L., Păun, G., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: characterizing recursively enumerable languages using insertion-deletion systems. In: DNA Based Computers III (DNA3), *DIMACS*, vol. 48, pp. 329–347 (1999)
30. Kari, L., Sosík, P.: On the weight of universal insertion grammars. Theoretical Computer Science **396**(1-3), 264–270 (2008)
31. Kim, S.M.: An algorithm for identifying spliced languages. In: T. Jiang, D. Lee (eds.) Proc. Computing and Combinatorics Conference, COCOON, *LNCS*, vol. 1276, pp. 403–411 (1997)
32. Kopecki, S.: On iterated hairpin completion. Theoretical Computer Science **412**(29), 3629–3638 (2011)
33. Landweber, L.F., Kari, L.: The evolution of cellular computing: natures solution to a computational problem. Biosystems **52**(13), 3–13 (1999)
34. Ledesma, L., Manrique, D., Rodríguez-Patón, A.: A tissue P system and a DNA microfluidic device for solving the shortest common superstring problem. Soft Computing **9**(9), 679–685 (2005)
35. Manca, V., Franco, G.: Computing by polymerase chain reaction. Mathematical Biosciences **211**(2), 282–298 (2008)
36. Manea, F., Martín-Vide, C., Mitrana, V.: On some algorithmic problems regarding the hairpin completion. Discrete Applied Mathematics **157**(9), 2143–2152 (2009)
37. Manea, F., Mitrana, V.: Hairpin completion versus hairpin reduction. In: S.B. Cooper, B. Löwe, A. Sorbi (eds.) Proc. Computability in Europe, CiE, *LNCS*, vol. 4497, pp. 532–541 (2007)
38. Manea, F., Mitrana, V., Sempere, J.: Some Remarks on Superposition Based on Watson-Crick-Like Complementarity. In: V. Diekert, D. Nowotka (eds.) Developments in Language Theory, *LNCS*, vol. 5583, pp. 372–383 (2009)
39. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science **296**(2), 295–326 (2003)
40. Mehlhorn, K.: Pebbling moutain ranges and its application of DCFL-Recognition. In: Proc. Automata, Languages and Programming, ICALP, pp. 422–435. Springer-Verlag (1980)
41. Minsky, M.L.: Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. The Annals of Mathematics **74**(3), 437–455 (1961)
42. Ouyang, Q., Kaplan, P.D., Liu, S., Libchaber, A.: DNA Solution of the Maximal Clique Problem. Science **278**(5337), 446–449 (1997)
43. Pixton, D.: Regularity of splicing languages. Discrete Applied Mathematics **69**(1-2), 101–124 (1996)
44. Prescott, D.M., Ehrenfeucht, A., Rozenberg, G.: Molecular operations for DNA processing in hypotrichous ciliates. European Journal of Protistology **37**(3), 241–260 (2001)
45. Păun, G., Pèrez-Jimènez, M.J., Yokomori, T.: Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. Int. J. of Foundations of Computer Science **19**(4), 859–871 (2008)
46. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer-Verlag New York, Inc. (2006)
47. Stemmer, W.P.: DNA shuffling by random fragmentation and reassembly: in vitro recombination for molecular evolution. Proceedings of the National Academy of Sciences **91**(22), 10,747–10,751 (1994)
48. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. Natural Computing **2**(4), 321–336 (2003)
49. Yong, M., Xiao-Gang, J., Xian-Chuang, S., Bo, P.: Minimizing of the only-insertion insdel systems. Journal of Zhejiang University Science A **6**(10), 1021–1025 (2005)