Superposition Based on Watson-Crick-like Complementarity

Paolo Bottoni **Anna Labella**
Department of Computer Science
University of Rome "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
{ bottoni,labella}@dsi.uniroma1.it

Vincenzo Manca
Department of Computer Science
University of Pisa
Corso Italia, 40 - 56125 Pisa, Italy
mancav@di.unipi.it

Victor Mitrana[1]
Faculty of Mathematics and Computer Science,
University of Bucharest,
Str. Academiei 14, 70109, Bucharest, Romania
and
Research Group in Mathematical Linguistics
Rovira i Virgili University,
Pça. Imperial Tarraco 1, 43005, Tarragona, Spain
vmi@correu.urv.ro

[1]Work supported by the departments of Computer Science from the universities of Roma and Pisa

Abstract.

In this paper we propose a new formal operation on words and languages, called *superposition*. By this operation, based on a Watson-Crick-like complementarity, we can generate a set of words, starting from a pair of words, in which the contribution of a word to the result need not be one subword only, as happens in classical bio-operations of DNA computing. Specifically, starting from two single stranded molecules x and y such that a suffix of x is complementary to a prefix of y, a prefix of x is complementary to a suffix of y, or x is complementary to a subword of y, a new word z, which is a prolongation of x to the right, to the left, or to both, respectively, is obtained by annealing. If y is complementary to a subword of x, then the result is x. This operation is considered here as an abstract operation on formal languages. We relate it to other operations in formal language theory and we settle the closure properties under this operation of classes in the Chomsky hierarchy. We obtain a useful result by showing that unrestricted iteration of the superposition operation, where the "parents" in a subsequent iteration can be any words produced during any preceding iteration step, is equivalent to restricted iteration, where at each step one parent must be a word from the initial language. This result is used for establishing the closure properties of classes in the Chomsky hierarchy under iterated superposition. Actually, since the results are formulated in terms of AFL theory, they are applicable to more classes of languages. Then we discuss "adult" languages, languages consisting of words that cannot be extended by further superposition, and show that this notion might bring us to the border of recursive languages. Finally, we consider some operations involved in classical DNA algorithms, such as Adleman's, which might be expressed through iterated superposition.

Keywords: Watson-Crick complementarity, superposition, restricted superposition closure, unrestricted superposition closure, maximal word.

1 Introduction

A DNA molecule consists of a double strand, each DNA single strand being composed by nucleotides which differ from each other by their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). The two strands which form the DNA molecule are kept together by the hydrogen bond between the bases: A always bonds with T, while C with G. This paradigm of *Watson-Crick complementarity* will be one of the main concepts used in defining the formal operation of superposition investigated in the present paper.

Two other biological principles used as sources of inspiration in this paper are that of *annealing* and that of *lengthening DNA by polymerases*. The first principle refers to fusing two single stranded molecules by complementary base pairing while the second one refers to adding nucleotides to one strand (in a more general setting to both strands) of a double stranded DNA molecule. The former operation requires a heated solution containing the two strands which is cooled down slowly. The latter one requires two single strands such that one (usually called *primer*) is bonded to a part of the other (usually called *template*) by Watson-Crick complementarity and a *polymerization buffer* with many copies of the four nucleotides that polymerases will concatenate to the primer by complementing the template.

We now informally explain the superposition operation and how it can be related to the aforementioned biological concepts. Let us consider the following hypothetical biological situation: two single stranded DNA molecules x and y are given such that a suffix of x is Watson-Crick complementary to a prefix of y or a prefix of x is Watson-Crick complementary to a suffix of y, or x is Watson-Crick complementary to a subword of y. Then x and y get annealed in a DNA molecule with a double stranded part by complementary base pairing and then a complete double stranded molecule is formed by DNA polymerases. The mathematical expression of this hypothetical situation defines the superposition operation. Assume that we have an alphabet and a complementary relation on its letters. For two words x and y over this alphabet, if a suffix of x is complementary to a prefix of y or a prefix of x is complementary to a suffix of y, or x is complementary to a subword of y, then x and y bond together by complementary letter pairing and then a complete double stranded word is formed by the prolongation of x and y. Now the upper word is considered to be the result of the superposition applied to x and y. Of course, all these phenomena are considered here in an idealized way. For instance, we allow polymerase to extend the shorter strand in either end (3' or 5') as well as in both, despite that in biology almost all polymerases extend in the direction from 5' to 3'.

This operation resembles some other operations on words: *sticking* investigated in [11, 6, 15] (a particular type of polyominoes with sticky ends are combined provided that the sticky ends are Watson-Crick complementary), *PA-matching* considered in [12] which is related to both the *splicing* and the *annealing* operations, as well as the *superposition* operation introduced in [2] (two words which may contain *transparent* positions are aligned one over the other and the resulting word is obtained by reading

the visible positions as well as aligned transparent positions).

The paper is organized as follows. In Section 2, we fix the terminology and give preliminary definitions. In Section 3 we formally introduce the superposition operation, relate it to other operations in formal language theory like PA-matching (Theorem 1), and settle the closure properties of classes in the Chomsky hierarchy under it. Actually, since the results are formulated in terms of AFL theory, they are applicable to more classes of languages than just those of the Chomsky hierarchy (Theorem 2, 3). In Section 4 we introduce the iterated superposition operation. A similar investigation concerning the closure properties of some AFLs under iterated superposition is done with direct consequences to the families in the Chomsky hierarchy (Theorem 4). Then we give a way of generating every regular language: we start from three finite languages to which we apply the iterated and non-iterated superposition and finally take the projection of the obtained language (Theorem 5). We obtain a useful result by showing that unrestricted iteration, where the "parents" in a subsequent iteration can be any words produced during any preceding iteration step, is equivalent to restricted iteration, where at each step one parent must be a word from the initial language (Theorem 6). Section 5 defines maximal ("adult") languages with respect to the iterated superposition closure of a language, i.e., informally, languages consisting of words which cannot be extended by further superposition, and shows that the membership problem is decidable for maximal languages w.r.t. the iterated superposition closure of finite languages (Theorem 9) but fails to be decidable for maximal languages w.r.t. the iterated superposition closure of context-sensitive languages (Theorem 10). Section 6 discusses how some DNA algorithms can be mathematically expressed in terms of superpositions. The paper ends by some final remarks.

2 Preliminaries

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly the notions of grammar and finite automaton [20].

An alphabet is always a finite set of letters. For a finite set A we denote by $card(A)$ the cardinality of A. The set of all words over an alphabet V is denoted by V^*. The empty word is written ε; moreover, $V^+ = V^* \setminus \{\varepsilon\}$. Given a word w over an alphabet V, we denote by $|w|$ its length. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w.

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the languages in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said to be here a *Watson-Crick involution*. Despite that this is nothing more than a fixed point-free involution, we prefer this terminology since the superposition defined later is inspired

by the DNA lengthening by polymerases, where the Watson-Crick complementarity plays an important role. Let $\bar{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from Ω^* to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ can intersect and they can be, but need not be, equal. Remember that the DNA alphabet consists of four letters, $V_{DNA} = \{A, C, G, T\}$, which are abbreviations for the four nucleotides and we may set $\overline{A} = T$, $\overline{C} = G$.

A morphism $h : V^* \longrightarrow U^*$ such that $h(a) \in U$ for all $a \in V$ is said to be a *coding*. If $U \subseteq V$ and $h(a) = a$, provided $a \in U$, and $h(a) = \varepsilon$ for all $a \in V \setminus U$, then h is said to be a *projection* (of V on U) and is denoted by $pr_{V,U}$. If $L \subseteq V^*$, $k \geq 1$, and $h : V^* \longrightarrow U^*$ is a morphism such that $h(x) \neq \varepsilon$ for all the subwords x of any word in L, $|x| = k$, then we say that h is *k-restricted erasing* on L. We stress that any morphism as above is actually the restriction of a morphism $\Omega^* \longrightarrow \Omega^*$.

A *finite transducer* is a 6-tuple $M = (Q, V_i, V_o, q_0, F, \delta)$ where Q, V_i, V_o are finite and nonempty sets (the set of states, the input alphabet, and the output alphabet, respectively), $q_0 \in Q$ (the initial state), $F \subseteq Q$ (the set of final states), and δ is the (transition-and-output) function from $Q \times (V_i \cup \{\varepsilon\})$ to finite subsets of $Q \times V_o^*$. This function is extended in a natural way to $Q \times V_i^*$. Every finite transducer M as above defines a *finite transduction*

$$M(\alpha) = \{\beta \in V_o^* \mid \text{ there exists } q \in F \text{ such that } (q, \beta) \in \delta(q_0, \alpha)\}, \ \alpha \in V_i^*.$$

If $M(\alpha) \neq \emptyset$, then we say that α is "accepted" by M. The language accepted by a finite transducer M is the set of all α such that $M(\alpha) \neq \emptyset$. The finite transduction M is extended to languages $L \subseteq V_i^*$ in the obvious way, namely $M(L) = \bigcup_{\alpha \in L} M(\alpha)$. If we ignore V_o and the output part of δ, then we obtain a *finite automaton* (with ε moves). A finite automaton is denoted (Q, V, q_0, F, δ). A language is *regular* iff it is accepted by a finite automaton.

3 Non-Iterated Superposition

Given two words $x \in V_1^+$ and $y \in V_2^+$ we denote by $\diamond$ the *superposition* operation defined as follows:

$$x \diamond y = \{z \in (V_1 \cup \overline{V}_2)^+ \mid \text{ one of the following conditions is satisfied:}$$

1. there exist $u \in V_1^*, w \in V_1^+, v \in V_2^*$ such that $x = uw, y = \overline{w}v, \text{ and } z = uw\overline{v}$.
2. there exist $u, v \in V_1^*$ such that $x = u\overline{y}v$ and $z = u\overline{y}v$.
3. there exist $u \in V_2^*, w \in V_1^+, v \in V_1^*$ such that $x = wv, y = u\overline{w}, \text{ and } z = \overline{u}wv$.
4. there exist $u, v \in V_2^*$ such that $y = u\overline{x}v$ and $z = \overline{u}x\overline{v}$.$\}$

This operation is schematically illustrated in Figure 1.

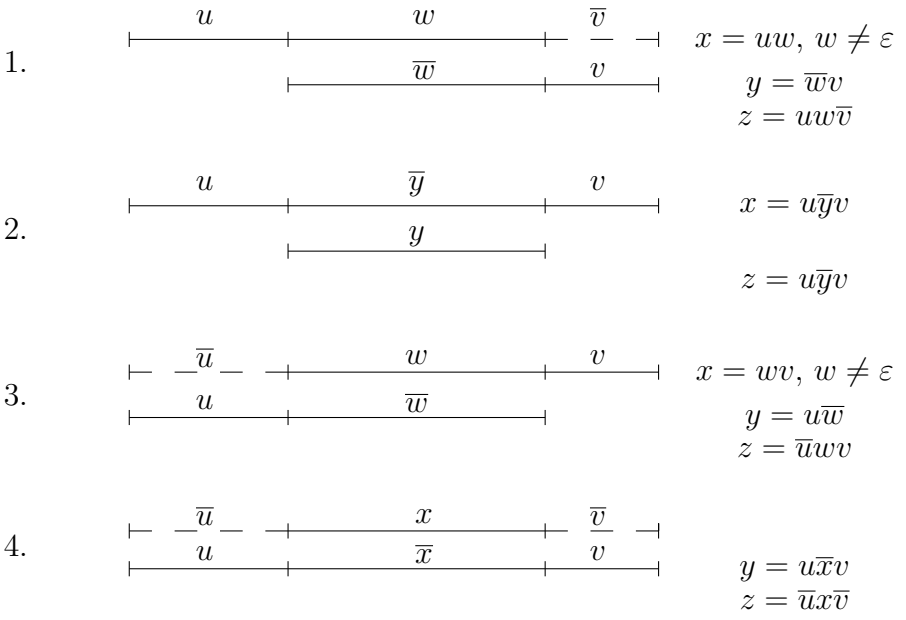

Figure 1: Superposition

More informally, starting from two single stranded molecules x and y such that a suffix of x is complementary to a prefix of y (case 1), a prefix of x is complementary to a suffix of y (case 3), or x is complementary to a subword of y (case 4), a new word z, which is a prolongation of x to the right, to the left, or to both, respectively, is obtained by annealing. If y is complementary to a subword of x, then the result is x (case 2). By this operation, based on the Watson-Crick complementarity, we can generate a finite set of words, starting from a pair of words, in which the contribution of a word to the result need not be one subword, as happens in classical bio-operations of DNA computing [14].

We stress from the very beginning the mathematical character of this definition: nature cannot distinguish which is the upper or the lower strand in the process of constructing a double stranded molecule from two single strands. Note that $y \diamond x = \overline{x \diamond y}$. Further, our model reflects polymerase reactions in both 5'$\longrightarrow$ 3' and 3'$\longrightarrow$ 5' directions. Due to the greater stability of 3' when attaching new nucleotides, DNA polymerase can act continuously only in the 5'$\longrightarrow$ 3' direction. However, polymerase can also act in the opposite direction, but in short "spurts" (Okazaki fragments).

We extend this operation to languages by

$$L_1 \diamond L_2 = \bigcup_{x \in L_1, y \in L_2} x \diamond y.$$

We write $\diamond(L)$ instead of $L \diamond L$.

Note that superposition is not associative. Indeed, take the alphabet $\{a, b, \bar{a}, \bar{b}\}$ and the words $x = ab$, $y = \bar{b}a$, $z = aa$. It is easy to see that $(x \diamond y) \diamond z = \{ab\bar{a}\bar{a}\}$ while $x \diamond (y \diamond z) = \emptyset$.

As observed in the Introduction, this operation is considered here as an abstract operation on formal languages. We relate it to other operations in formal language theory and we settle the closure properties of the families in the Chomsky hierarchy under it. Then, we consider the iterated version of this operation and define two types of languages obtained by iterated superposition.

We first recall a closely related operation on words, mentioned in the Introduction, inspired by the operation used in [18]. This operation, called the PA-matching, was defined in [12]. It belongs to "cut-and-paste" operations much investigated as basic operations for theoretical models of DNA computing (see details in [14]). Informally speaking, starting from two single stranded molecules x, y, such that a suffix w of x is a prefix of y, we can form the molecule with a double stranded part (both strands are identical) and the remaining sticky ends specified by x and y. The matching part is then ignored (removed), so that the resulting word consists of the prefix of x and the suffix of y which were not matched.

Formally, given two words $x \in V_1^+$ and $y \in V_2^+$, one defines

$$PAm(x, y) = \{uv \mid x = uw, y = wv, \text{ for some } w \in (V_1 \cap V_2)^+, \text{ and } u \in V_1^*, v \in V_2^*\}.$$

The operation is naturally extended to languages by

$$PAm(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} PAm(x, y).$$

We recall that a family of languages $\mathcal{F}$ is said to be a *trio* if it is closed under non-erasing morphisms, inverse morphisms and intersection with regular sets. A trio closed under arbitrary morphisms is called *full trio* known also as *cone*. We recall that all families of languages considered are restricted to languages in the letters of Ω and the morphisms do not map outside the universe Ω.

Theorem 1 *Every full trio closed under superposition is closed under PA-matching.*

Proof. Let $L_i \subseteq V_i^*$, $i = 1, 2$, be two languages in a given full trio. We will show that $PAm(L_1, L_2)$ is in the trio. We consider two new symbols (note that here, and in the later proofs, the new symbols together with their complements, which are also new, are taken from Ω) $\#, \$$, and two morphisms

$$h_1 : (V_1 \cup \{\#, \$\})^* \longrightarrow V_1 \text{ by } \quad h_1(a) = a, \ a \in V_1, \quad h_1(a) = \varepsilon, \text{ otherwise}$$

$$h_2 : (\overline{V}_2 \cup \{\overline{\#}, \overline{\$}\})^* \longrightarrow V_2 \text{ by } \quad h_2(\overline{a}) = a, \ a \in V_2, \quad h_2(a) = \varepsilon, \text{ otherwise .}$$

Note that h_1 is a projection. Words in the set

$$S = (h_1^{-1}(L_1) \cap R_1) \diamond (h_2^{-1}(L_2) \cap R_2)$$

with

$$R_1 = V_1^* \# V_1^+ \$ \qquad \text{and} \qquad R_2 = \overline{\# V_2^+ \$ V_2^*}$$

are either of the form

$$\# u \$ w \# v \$,\ w \in (V_1 \cup V_2)^+,\ u, v \in (V_1 \cup V_2)^*,$$

or of the form

$$u \# w \$ v,\ w \in (V_1 \cap V_2)^+,\ u, v \in (V_1 \cup V_2)^*.$$

In the former case, the words resulted from a superposition as shown in Figure 2, while in the latter case the words resulted from a superposition as shown in Figure 3.

Figure 2: Useless superposition

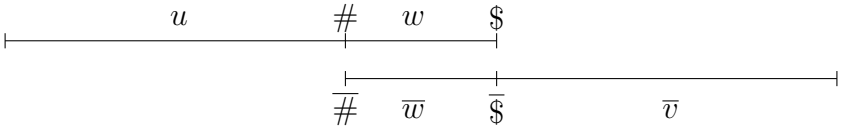

Figure 3: Useful superposition

By the hypothesis, S is in the same trio. There exists a finite transducer M which does not accept any input word of the first form while it deletes the segment $\#y\$$ from each input word of the second form, thus simulating the effect of PA-matching. It follows that $PAm(L_1, L_2) = M(S)$. Since any full trio is closed under finite transductions, the assertion follows. □

Theorem 2 *Every family of languages closed under coding, projection, concatenation with symbols, and superposition is closed under intersection and concatenation.*

Proof. The following equality is immediate

$$L_1 \cap L_2 = g((\# L_1 \$) \diamond (\overline{\# L_2 \$}))$$

where # and \$ are two new symbols, and g is a morphism which erases the two new symbols and leaves the others unchanged, i.e., it is a projection. Moreover, the equality

$$L_1 \cdot L_2 = g((\#L_1\$) \diamond (\overline{\$L_2\$}))$$

is also immediate. Since $\bar{\cdot}$ is a coding, the proof follows. □

Corollary 1 *The family of context-free languages fails to be closed under superposition.*

We observe that one can easily prove, using Post's Correspondence Problem, that the following problem is undecidable: Is $\diamond(L)$ context-free for a given context-free language L?

A family of languages $\mathcal{F}$ is closed under right superposition with regular languages if for any language $L \in \mathcal{F}$ and any regular language R, $L \diamond R \in \mathcal{F}$ holds. In a similar way the closure under left superposition with regular languages may be defined.

Theorem 3 *1. Every trio is closed under superposition iff it is closed under intersection.*

2. Every trio is closed under right and left superposition with regular languages.

Proof. 1. Let L_1, L_2 be two languages in the trio $\mathcal{F}$, $L_i \subseteq V_i^*$, $i = 1, 2$. We define the new alphabets $V_2' = \{a' \mid a \in V_2\}$ and $U = \{a'' \mid a \in V_1 \cap \overline{V}_2\}$. Then

$$L_1 \diamond L_2 = g(h_1^{-1}(L_1) \cap h_2^{-1}(L_2) \cap R),$$

where the morphisms h_1, h_2, g are defined by

$$\begin{aligned} h_1 &: (V_1 \cup V_2' \cup U)^* \longrightarrow V_1^*,\ h_1(a) = h_1(a'') = a,\ h_1(a') = \varepsilon, \\ h_2 &: (V_1 \cup V_2' \cup U)^* \longrightarrow V_2^*,\ h_2(a) = \varepsilon,\ h_2(a'') = \overline{a},\ h_2(a') = a, \\ g &: (V_1 \cup V_2' \cup U)^* \longrightarrow (V_1 \cup \overline{V}_2)^*,\ g(a) = g(a'') = a,\ g(a') = \overline{a}, \end{aligned}$$

and R is a regular language (the four parts of R correspond to cases 1-4 in Figure 1 in the same order) defined by

$$R = V_1^* U^+ (V_2')^* \cup V_1^* U^+ V_1^* \cup (V_2')^* U^+ V_1^* \cup (V_2')^* U^+ (V_2')^*.$$

Thus, if $\mathcal{F}$ is closed under intersection, then $L_1 \diamond L_2 \in \mathcal{F}$.

Conversely, by the proof of Theorem 2 it follows that every trio closed under superposition is closed under intersection, since the concatenation with symbols can be accomplished by an inverse morphism followed by intersection with a regular language, and the projection g from that proof is actually a 2-restricted erasing morphism on $(\#L_1\$) \diamond (\overline{\#L_2\$})$. Moreover, it is known (see Theorem IV 2.5 in [22]) that every trio $\mathcal{F}$ is closed under restricted erasing morphisms in the sense that $f(L)$ lies in $\mathcal{F}$ provided that $L \in \mathcal{F}$ and f is k-restricted erasing on L for some $k \geq 1$.

2. The second statement follows directly from the above proof. □

Corollary 2 *1. The families of regular, context-sensitive and recursively enumerable languages are closed under superposition.*

2. The family of context-free languages is closed under left and right superposition with regular languages.

4 Iterated Superposition

Given a language L we define the language obtained from L by unrestrictedly iterated application of superposition. This language, called the *unrestricted superposition closure of* L is denoted by $\diamond_u^*(L)$ and defined by

$$\begin{aligned} \mu_0(L) &= L, \\ \mu_{i+1}(L) &= \mu_i(L) \cup \diamond(\mu_i(L)), i \geq 0, \\ \diamond_u^*(L) &= \bigcup_{i \geq 0} \mu_i(L). \end{aligned}$$

Clearly, $\diamond_u^*(L)$ is the smallest language containing L and closed under superposition. More precisely, it is the smallest language K such that $L \subseteq K$ and $\diamond(K) \subseteq K$. In words, one starts with the words in L and applies iteratively superposition to any pair of words previously produced. Note the lack of any restriction in choosing the pair of words. All the obtained words are collected in the set $\diamond_u^*(L)$.

We say that a family $\mathcal{F}$ of languages is closed under unrestrictedly iterated superposition if $\diamond_u^*(L)$ is in $\mathcal{F}$ for any language $L \in \mathcal{F}$. A trio closed under union is called a *semi-AFL*.

Theorem 4 *Every semi-AFL closed under unrestrictedly iterated superposition is closed under superposition.*

Proof. We take two languages $L_i \subseteq V_i^*$, $i = 1, 2$, in such a family and consider three new symbols a, b, c. Then we construct the languages

$$\begin{aligned} E_1 &= aL_1 \cup L_2\overline{b} & E_2 &= cL_2c \cup \overline{a}(h_1^{-1}(L_1) \cap R_1)\overline{b} \\ E_3 &= L_1b \cup \overline{a}L_2 & E_4 &= cL_1c \cup \overline{a}(h_2^{-1}(L_2) \cap R_2)\overline{b}, \end{aligned}$$

where

- $h_1 = pr_{V_1 \cup \{\overline{c}\}, V_1}$, $h_2 = pr_{V_2 \cup \{\overline{c}\}, V_2}$
- R_1 is the regular language $R_1 = V_1^* \overline{c} V_1^+ \overline{c} V_1^*$
- R_2 is the regular language $R_2 = V_2^* \overline{c} V_2^+ \overline{c} V_2^*$.

As observed in the proof of Theorem 3, concatenation with symbols can be realized by an inverse morphism followed by the intersection with a regular language, hence E_1, E_2, E_3, and E_4 are still in the given family. We now show that

$$\begin{aligned} L_1 \diamond L_2 = g((\bigcup_{i \in \{1,3,4\}} \diamond_u^*(E_i)) \cap a(V_1 \cup \overline{V}_2 \cup \{c\})^* b) \cup \\ \overline{g(\diamond_u^*(E_2) \cap a(V_2 \cup \overline{V}_1 \cup \{c\})^* b)}, \end{aligned} \quad (1)$$

with g being a projection which erases all the new symbols. First, it is plain that $\diamond_u^*(E_i)$ is actually $\diamond(E_i) \cup E_i$, $1 \leq i \leq 4$. Second, the intersection of $\diamond(E_i) \cup E_i$, $i = 1, 3, 4$, with the regular language $a(V_1 \cup \overline{V}_2 \cup \{c\})^* b$ selects only those words t in $\diamond(E_i)$ such that:

(i) If $i = 1, 3$, then $t = azb$, $z \in (V_1 \cup \overline{V}_2)^+$, and z can be obtained in $L_1 \diamond L_2$ as shown in the cases 1 or 3 of Figure 1.

(ii) If $i = 4$, then $t = az_1cz_2cz_3b$, $z_1, z_3 \in \overline{V}_2^*$, $z_2 \in V_1^+$, and $z = z_1z_2z_3$ can be obtained in $L_1 \diamond L_2$ as shown in the case 4 of Figure 1.

Third, the intersection of $\diamond(E_2) \cup E_2$ with the regular language $a(V_2 \cup \overline{V}_1 \cup \{c\})^*b$ selects only those words t in $\diamond(E_2)$ such that $t = az_1cz_2cz_3b$, $z_1, z_3 \in \overline{V}_1^*$, $z_2 \in V_2^+$, and $z = z_1z_2z_3$ can be obtained in $L_2 \diamond L_1$ as shown in the case 4 of Figure 1. Consequently, $\overline{z}$ can be obtained in $L_1 \diamond L_2$ as shown in the case 2 of Figure 1. By these three facts the equality (1) follows. □

Corollary 3 *The family of context-free languages fails to be closed under unrestrictedly iterated superposition.*

We do not know whether the family of regular languages is closed under unrestrictedly iterated superposition. However, we show below how any regular language can be obtained starting from finite languages my means of superposition, iterated superposition and projection. Note that this is not a characterization of the class of regular languages since we do not know whether or not the unrestricted superposition closure of a finite language is regular.

Theorem 5 *Every regular language R can be written as*

$$R = h((L_1 \diamond (\diamond_u^*(L_2))) \diamond L_3),$$

where h is a projection and L_1, L_2, L_3 are finite languages.

Proof. Let us consider a finite automaton $A = (Q, V, q_0, F, \delta)$ that accepts R such that $q_0 \notin \delta(q, a)$ for any $q \in Q$ and $a \in V \cup \{\varepsilon\}$, $F = \{q_f\}$, $q_0 \neq q_f$, and $\delta(q_f, a) = \emptyset$ for all $a \in V \cup \{\varepsilon\}$. We define the finite languages:

$$\begin{aligned} L_1 &= \{q_0\}, \\ L_2 &= \{sas' \mid s' \in \delta(s, a), s, s' \in Q, a \in V \cup \{\varepsilon\}\} \cup \\ &\quad \{\overline{sas'} \mid s' \in \delta(s, a), s, s' \in Q, a \in V \cup \{\varepsilon\}\}, \\ L_3 &= \{\overline{q_f}\}. \end{aligned}$$

Words in the set $\diamond_u^*(L_2)$ are of the form

$$s_1a_1s_2a_2 \ldots s_na_ns_{n+1} \tag{2}$$

with $s_{i+1} \in \delta(s_i, a_i)$ for all $1 \leq i \leq n$, as well as their Watson-Crick complements. It follows that words in the set $L_1 \diamond (\diamond_u^*(L_2))$ are of the same form (2) but with $s_1 = q_0$. Furthermore, only those words which have $s_n = q_f$ are selected by applying the superposition to the previous language and L_3. Now h removes all symbols not in V and the proof is complete. □

We now introduce another superposition closure of a language which may be viewed as a "normal form" of iterated superposition. This is suggested by the proof of the previous theorem which holds for restricted superposition closure without any change.

The *restricted superposition closure of* L denoted by $\diamond_r^*(L)$ is defined in the following way:

$$\begin{aligned} \diamond_r^0(L) &= L, \\ \diamond_r^{k+1}(L) &= ((\diamond_r^k(L)) \diamond L) \cup (L \diamond (\diamond_r^k(L))) \cup \diamond_r^k(L) \\ \diamond_r^*(L) &= \bigcup_{k \geq 0} \diamond_r^k(L). \end{aligned}$$

Note the main difference between the unrestricted and restricted way of iterating superpositions. In the latter case, superposition takes place between a word produced so far and an initial word only. Note that $\diamond_r^*(L) \subseteq \diamond_u^*(L)$ for any language L. Surprisingly enough (remember that $\diamond$ is not associative), we have an equality between the two superposition closures of any language. In order to prove this "normal form" we need some preliminaries.

Given a word $x \in \diamond_r^*(L)$ we define

$$ord_r(x) = min\{i \mid x \in \diamond_r^i(L)\}.$$

Remark. Note that for any word $x \in \diamond_r^*(L)$ with $ord_r(x) \geq 1$, we have that $\overline{x} \in \diamond_r^*(L)$ and $ord_r(\overline{x}) \leq ord_r(x)$. Actually, only for some $x \in \diamond_r^*(L)$ with $ord_r(x) = 1$ it is possible to have $\overline{x} \in L$. This remark will be very useful in the proof of the next theorem.

Theorem 6 [Normal Form] $\diamond_r^*(L) = \diamond_u^*(L)$ *for any language* L.

Proof. Since $\diamond_u^*(L)$ is the smallest language containing L and closed under superposition, and since $\diamond_r^*(L)$ contains L, the proof is complete if we prove that $\diamond_r^*(L)$ is closed under $\diamond$. Assume that this is not true, hence there exist $\alpha, \beta \in \diamond_r^*(L)$ such that $\alpha \diamond \beta \not\subseteq \diamond_r^*(L)$. We take such a pair of words (α, β) with $ord_r(\alpha) + ord_r(\beta)$ being minimal among all these pairs and $ord_r(\beta)$ being minimal among all pairs (α, β) having $ord_r(\alpha) + ord_r(\beta)$ minimal. Clearly, $ord_r(\alpha) \cdot ord_r(\beta) \neq 0$ (i.e., $\alpha \notin L$ and $\beta \notin L$).

Let γ be an arbitrary element of $\alpha \diamond \beta$. The following two cases of producing γ from α and β have to be considered. They are schematically illustrated in Figure 4.

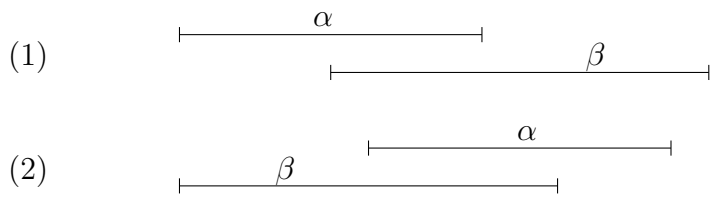

Figure 4: The two cases of producing γ

We shall consider the former case in detail; the latter follows from (1) by taking complements. We distinguish the following nine cases presented in Figure 5. In each case, the first segment represents α and the next two ones represent the two words to which a superposition was applied for producing β. Furthermore, one of these two words is in $\diamond_r^*(L)$ while the other is in L, both of them having an order with respect to the restricted superposition closure strictly smaller than β. For a better understanding, we have followed the order of the cases 1,4,3 of Figure 1 for the second and the third word. For a fixed case, we have ordered the possible situations depending on whether they overlap outside α, they overlap with α, or they overlap inside α.

We now examine each of the nine cases:

(a) If $vt \in L$ then $\overline{xyuv} \in \overline{yuv} \diamond xy \subseteq \diamond_r^*(L)$, since $ord_r(\overline{yuv}) + ord_r(xy) < ord_r(\alpha) + ord_r(\beta)$. Now $\gamma \in vt \diamond \overline{xyuv}$ which implies that $\gamma \in \diamond_r^*(L)$. Assume now that $vt \notin L$, hence $\overline{yuv} \in L$. Then $xyuv \in \diamond_r^*(L)$ and $ord_r(xyuv) \leq ord_r(\alpha) + 1$, hence $ord_r(xyuv) + ord_r(\overline{vt}) \leq ord_r(\alpha) + ord_r(\beta)$. But since $ord_r(\overline{vt}) \leq ord_r(vt) < ord_r(\beta)$, from the choice of α and β it follows that $\gamma \in xyuv \diamond \overline{vt} \subseteq \diamond_r^*(L)$.

(b) If $tuv \in L$, then $\overline{\gamma} \in tuv \diamond \overline{xy} \subseteq \diamond_r^*(L)$, otherwise $\gamma \in xy \diamond \overline{tuv} \subseteq \diamond_r^*(L)$ holds because $ord_r(xy) + ord_r(\overline{tuv}) < ord_r(\alpha) + ord_r(\beta)$.

(c,h,i) Clearly, $\gamma \in xy \diamond \overline{tuv} \subseteq \diamond_r^*(L)$.

(d) Clearly, $\gamma \in ytuv \diamond \overline{xy} \subseteq \diamond_r^*(L)$, since $\alpha \notin L$ so that $ord_r(\overline{\alpha}) \leq ord_r(\alpha)$.

(e,f) Clearly, $\gamma \in stuv \diamond \overline{xy} \subseteq \diamond_r^*(L)$, since $\alpha \notin L$ so that $ord_r(\overline{\alpha}) \leq ord_r(\alpha)$.

(g) This case is similar to (a). If $yuv \in L$, then $xyuv \in yuv \diamond \overline{xy} \subseteq \diamond_r^*(L)$ and $\gamma \in xyuv \diamond \overline{vt}$. Since $ord_r(xyuv) + ord_r(\overline{vt}) \leq ord_r(\alpha) + ord_r(\beta)$ and $ord_r(\overline{vt}) < ord_r(\beta)$, it follows that $\gamma \in \diamond_r^*(L)$. If $yuv \notin L$, then $xyuv \in xy \diamond \overline{yuv} \subseteq \diamond_r^*(L)$. We have that $\gamma \in xyuv \diamond \overline{vt}$ which is included in $\diamond_r^*(L)$ because $\overline{vt} \in L$.

As one can see, all cases lead to the fact that $\gamma \in \diamond_r^*(L)$, which is a contradiction. □

By this theorem we do not distinguish anymore the two languages defined by iterated superpositions applied to a language L and denote by $\diamond^*(L)$ the common language.

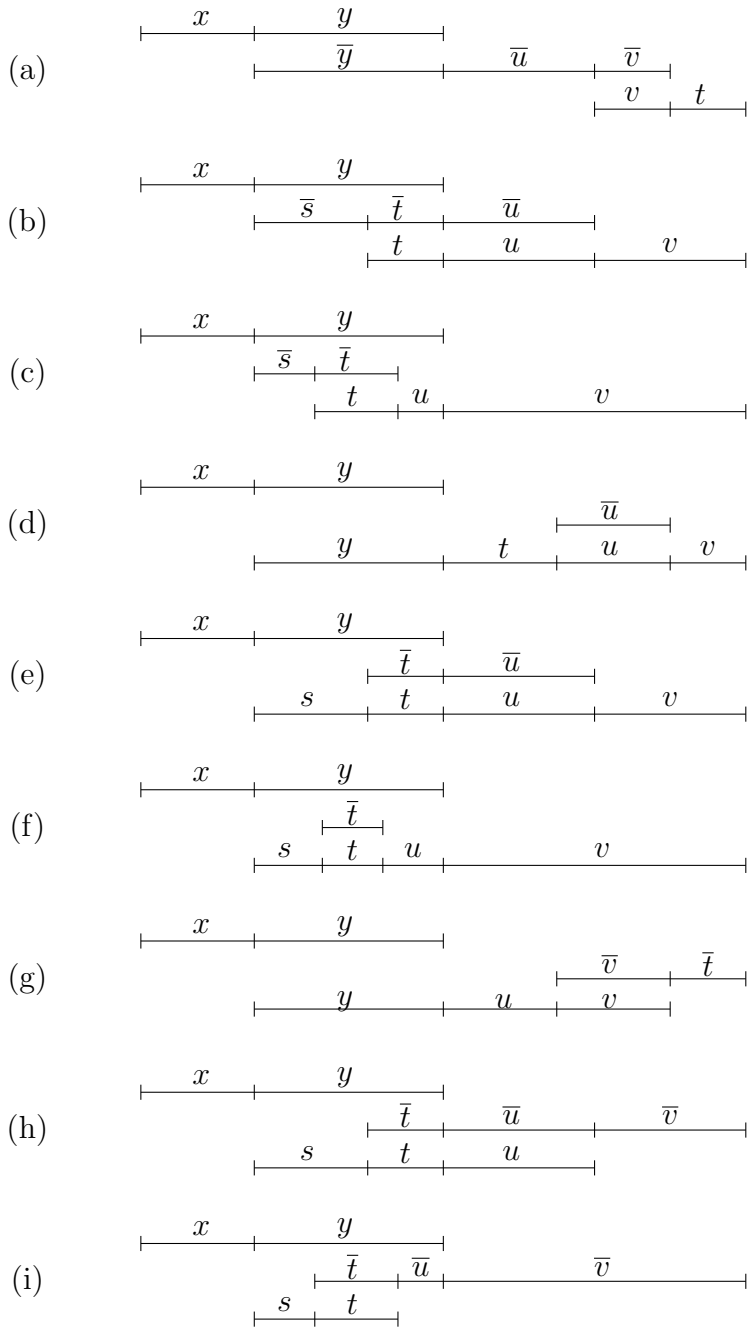

Figure 5: The nine cases

Since the class of context-sensitive languages is a space complexity class, we now show that such classes are closed under iterated superposition.

Theorem 7 *$NSPACE(f(n))$, where $f(n) \geq \log n$ is a space-constructible function, is closed under iterated superposition.*

Proof. Let us consider the following recursive boolean function which determines whether or not a given word x is in $\diamond^*(L)$:

Function *Membership*$(x, \diamond^*(L))$;
begin
Membership:=false;
if $x \in L$ then *Membership:*=true; endif; halt;
if *(x is a letter)* and $(x \notin L)$ then halt; endif;
choose nondeterministically a decomposition $x = uvw$, *with* $v \neq \varepsilon$;
proceed nondeterministically with

1. if $((Membership(uv, \diamond^*(L))$ and $(\overline{vw} \in L))$ or $((uv \in L)$ and $Membership(\overline{vw}, \diamond^*(L)))$ then *Membership:*=true; halt; endif;
2. if $((Membership(vw, \diamond^*(L))$ and $(\overline{uv} \in L))$ or $((vw \in L)$ and $Membership(\overline{uv}, \diamond^*(L)))$ then *Membership:*=true; halt; endif;
3. if $((Membership(v, \diamond^*(L))$ and $(\overline{uvw} \in L))$ or $((v \in L)$ and $Membership(\overline{uvw}, \diamond^*(L)))$ then *Membership:*=true; halt; endif;

end

As one can easily see, the algorithm is based on the Normal Form stated in Theorem 6 and actually closely follows the definition of $\diamond_r^*(L)$. This function can clearly be implemented on a nondeterministic (multi-tape) Turing machine in $f(n)$ space provided that L is accepted by a nondeterministic (multi-tape) Turing machine in $f(n)$ space.

Note that $\log n$ is needed in order to store the left- and right-hand border of the current subword within the input word. By finite state one can keep track of whether or not this subword is complemented. □

Corollary 4 *The families of context-sensitive and recursively enumerable languages are closed under iterated superposition.*

Proof. This statement is obvious for the family of recursively enumerable languages and follows from Theorem 7 for the family of context-sensitive languages because this family equals $NSPACE(n)$. □

We finish this section by pointing out an intriguing problem that remains without any answer. It is the version for superposition of a problem posed by T. Head in [10] and solved in [4] and [17] via rather complicated proofs. The problem is: Is $\diamond^*(L)$ regular, provided that L is regular? We strongly conjecture a positive answer as it was the case for the problem posed by Head. By Theorem 7 we infer that $NLOG$ is closed under iterated superposition, hence $\diamond^*(L) \in NLOG$ for any regular language L.

5 Maximal (Adult) Languages

As in the case of L systems, see, e.g., [19], we consider the *maximal (adult) words* with respect to the iterated superposition closure of some language L. A word x is a maximal word w.r.t. $\diamond^*(L)$ if $x \in \diamond^*(L)$ and $x \diamond (\diamond^*(L)) \subseteq \{x\}$.

We denote by $max \diamond^* (L)$ the set of all maximal words w.r.t. $\diamond^*(L)$. This language will be called the maximal language w.r.t. the iterated superposition closure of L. The result of Theorem 5 can now be written simpler as:

Theorem 8 *Every regular language is the projection of a maximal language w.r.t. the iterated superposition closure of a finite language.*

Proof. The equality $R = h(max \diamond^* (L_2))$, where R, h, and L_2 are defined in the proof of Theorem 5, is immediate. □

An interesting problem in our view is to determine, if possible, the maximal language w.r.t. the iterated superposition closure of a regular language. A first and natural question regards the decidability of the membership problem for this language. The next result gives an answer to this problem for a particular case: the initial language is finite.

Theorem 9 *If L is a finite language, then $max \diamond^* (L)$ is recursive.*

Proof. Let L be a finite language over an alphabet and y be an arbitrary word over the same alphabet. We put

$$m = max\{|w| \mid w \in L\} \qquad \text{and} \qquad t = max(m, |y|) + 1.$$

Denote by

– $Pref_{\leq k}(A)$ the set of all prefixes of length at most k of the words in A,
– $Suf_{\leq k}(A)$ the set of all suffixes of length at most k of the words in A,
– $Sub_k(A)$ the set of all subwords of length k of the words in A.

Let k_0 be the smallest number $k \geq t$ such that

$$\begin{aligned} Pref_{\leq t}(\diamond^{k-1}(L)) &= Pref_{\leq t}(\diamond^k(L)), \\ Suf_{\leq t}(\diamond^{k-1}(L)) &= Suf_{\leq t}(\diamond^k(L)), \\ Sub_t(\diamond^{k-1}(L)) &= Sub_t(\diamond^k(L)). \end{aligned}$$

It is obvious that k_0 exists and the above sets can be algorithmically computed by a standard iterative procedure.

Claim:

1. $Y = Pref_{\leq t}(\diamond^{k_0}(L)) = Pref_{\leq t}(\diamond^*(L))$.
2. $Y' = Suf_{\leq t}(\diamond^{k_0}(L)) = Suf_{\leq t}(\diamond^*(L))$.
3. $V = Sub_t(\diamond^{k_0}(L)) = Sub_t(\diamond^*(L))$.

Proof of the claim. We denote

$$Z = Pref_{\leq t}(\diamond_r^{k_0+1}(L)), \quad Z' = Suf_{\leq t}(\diamond_r^{k_0+1}(L)), \quad W = Sub_t(\diamond_r^{k_0+1}(L)).$$

It is sufficient to prove that $Y = Z$, $Y' = Z'$, and $V = W$. We shall prove the first equality (the second one can be proved analogously) and the third one.

We proceed to prove $Y = Z$, actually $Z \subseteq Y$ which suffices. We distinguish the following five cases, each of them considering a possible generation of a word in

$\diamond_r^{k_0+1}(L) \setminus \diamond_r^{k_0}(L)$ for which we prove that all its prefixes of length at most t are in Y. To improve the readability we indicated the prefix by a long vertical line.

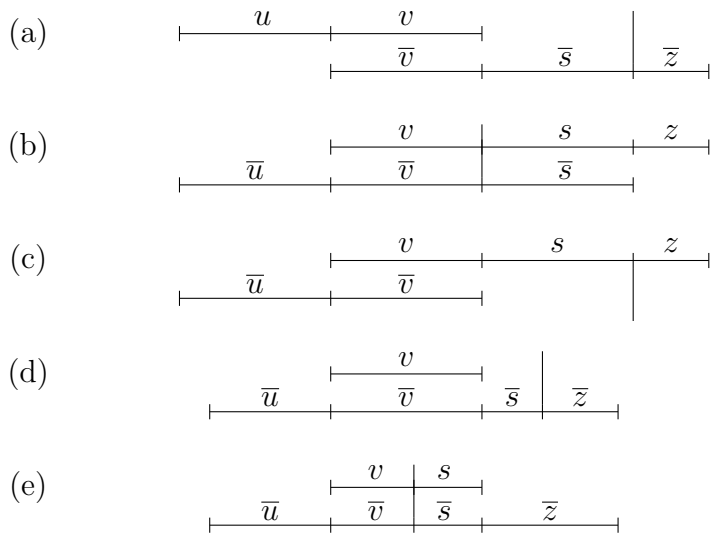

Figure 6

(i) $uvs \in Z$ obtained from a word in $uv \diamond \overline{vsz}$ as shown in Figure 6(a). Since $|x| \geq ord_r(x) + 1$ for all $x \in \diamond^*(L)$, it follows that $ord_r(uv) < k_0$, otherwise $|uv| \geq k_0 > t$. Consequently, $ord_r(\overline{vsz}) = k_0$ which implies that $\overline{vs} \in Y = X$, hence $uvs \in Y$.

(ii) $uv \in Z$ obtained from a word in $vsz \diamond \overline{uvs}$ as shown in Figure 6(b). If $ord_r(\overline{uvs}) = k_0$, then $ord_r(uvs) = k_0$, hence $uv \in Y$. If $ord_r(vsz) = k_0$, then $vs \in Y = X$ because $|\overline{uvs}| < t$ ($\overline{uvs} \in L$). But $vs \in X$ implies $uv \in Y$.

(iii) $uvs \in Z$ obtained from a word in $vsz \diamond \overline{uv}$ as shown in Figure 6(c). Clearly, $\overline{uv} \in L$ and $ord_r(vsz) = k_0$. Since $vs \in Y = X$, $uvs \in Y$ follows.

(iv) $uvs \in Z$ obtained from a word in $v \diamond \overline{uvsz}$ as shown in Figure 6(d). Clearly, $v \in L$ and $ord_r(\overline{uvsz}) = k_0$. Consequently, $ord_r(uvsz) = k_0$, hence $uvs \in Y$.

(v) $uv \in Z$ obtained from a word in $vs \diamond \overline{uvsz}$ as shown in Figure 6(e). If $ord_r(\overline{uvsz}) < k_0$, then $ord_r(vs) = k_0$ and $\overline{uvsz} \in L$. The former implies $|vs| \geq k_0 > t$ while the latter implies $|\overline{uvsz}| \leq t$. Therefore, $ord_r(\overline{uvsz}) = k_0$, hence $ord_r(uvsz) = k_0$, which leads to $uv \in Y$.

Since these cases are the only ones which might produce new prefixes, the first item of our claim is proved.

We now proceed to prove in a similar way the last item of the claim. An analysis of all the possibilities of getting new subwords of length t leads to the following six cases (to improve the readability we indicated the subword by two long vertical lines):

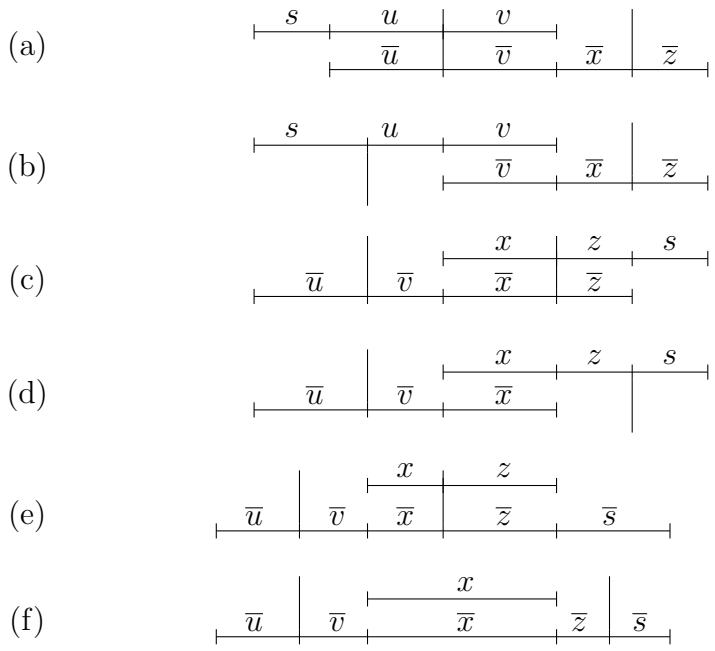

Figure 7

(i) $vx \in W$ obtained as shown in Figure 7(a). If $ord_r(\overline{uvxz}) = k_0$, then $ord_r(uvxz) = k_0$, hence $vx \in V$. If $ord_r(suv) = k_0$, then $uv \in Y' = X'$, hence $vx \in V$.

(ii) $uvx \in W$ obtained as shown in Figure 7(b). If $ord_r(\overline{vxz}) = k_0$, then $\overline{vx} \in Y = X$, hence $uvx \in V$. If $ord_r(suv) = k_0$, then $uv \in Y' = X'$, hence $uvx \in V$.

(iii) $vx \in W$ obtained as shown in Figure 7(c). If $ord_r(\overline{uvxz}) = k_0$, then $ord_r(uvxz) = k_0$, hence $vx \in V$. If $ord_r(xzs) = k_0$, then $xz \in Y = X$, hence $vx \in V$.

(iv) $vxz \in W$ obtained as shown in Figure 7(d). If $ord_r(\overline{uvx}) = k_0$, then $\overline{vx} \in Y' = X'$, hence $vxz \in V$. If $ord_r(xzs) = k_0$, then $xz \in Y = X$, hence $vxz \in V$.

(v) $vx \in W$ as shown in Figure 7(e). The unique possibility is $ord_r(\overline{uvxzs}) = k_0$, that is $ord_r(uvxzs) = k_0$ as well, hence $vx \in V$.

(vi) $vxz \in W$ as shown in Figure 7(f). This case leads to $vxz \in V$ in the same way as the previous case.

Now, it is easy to note that

$$y \in max\, \diamond^* (L) \text{ if and only if } (y \in \diamond^*(L)) \text{ and } (y \diamond (Y\$ \cup \$Y' \cup \$V\$) \subseteq \{y\}),$$

where \$ is a new symbol. The former condition is decidable since L is finite, $\diamond^*(L)$ is context-sensitive by Corollary 4, and hence it is recursive, and the latter one is decidable since Y, Y', V are finite. □

Theorem 10 *There exist context-sensitive languages such that the maximal language w.r.t. iterated superposition closure of such a language is not recursive.*

Proof. It is known that for any recursively enumerable language $L \subseteq V^*$ there exist a context-sensitive language E and three new symbols a, b, c not in V such that $E \subseteq Labc^*$, and for any w, $w \in L$ iff there exists $i \geq 0$ such that $wabc^i \in E$, see, e.g., Theorem III 9.9 in [22]. Let $L \subseteq V^*$ be a recursively enumerable language as above and $w \in V^+$ an arbitrary word. We consider the new alphabet

$$U = V \cup \overline{V} \cup \{a, b, c, d, \overline{a}, \overline{b}, \overline{c}, \overline{d}\},$$

where d is a further new symbol, and the context-sensitive language $F = dE \cup \{\overline{dxa} \mid x \in V^*\}$. We claim that

$$\overline{dwa} \in max \diamond^* (F) \text{ iff } w \notin L.$$

If $w \in L$, then there exists $i \geq 0$ such that $dwabc^i \in F$, hence $\overline{dwa}$ is obviously not maximal w.r.t. $\diamond^*(F)$. If $\overline{dwa}$ is not maximal w.r.t. $\diamond^*(F)$, then there exists $y \in E$ such that $\overline{dwa} \diamond dy$ contains a word other than $\overline{dwa}$. This is possible only if $y = wabc^i$ for some $i \geq 0$, hence $w \in L$.

Since there exist recursively enumerable languages that are not recursive the proof is complete. □

We give below another partial answer to the aforementioned problem; if our conjecture in the previous section turns out to be true, this partial answer will become a complete answer.

Theorem 11 *If the iterated superposition closure of a language L is a regular set, then the maximal language w.r.t. this closure is regular, too.*

Proof. We reduce the proof to a proof for the following statement: Given a regular language $K \subseteq V^*$ the language $\{x \in V^* \mid x \diamond K \subseteq \{x\}\}$ is regular. Obviously, we may assume that $\varepsilon \notin K$. Let $A = (Q, V, q_0, \{q_f\}, \delta)$ be a finite automaton without ε-moves which accepts K and satisfies the following conditions:

– $q_0 \neq q_f$,
– $\delta(q_f, a)$ is undefined for all $a \in V$,
– $q_0 \notin \delta(q, a)$ for any $q \in Q$ and $a \in V$.

For each pair of states q, s we define the set

$$E(q, s) = \{w \in V^+ \mid s \in \delta(q, w)\}.$$

We further assume that for every state $s \notin \{q_0, q_f\}$ both sets $E(q_0, s)$ and $E(s, q_f)$ are nonempty. Clearly, $E(q_0, q_f) = K$.

We define the following sets:

$Pref = \bigcup\{E(q_0, s) \mid s \in Q \setminus \{q_0, q_f\}\}$ (the set of proper prefixes of the words in K),

$Suf = \bigcup\{E(s, q_f) \mid s \in Q \setminus \{q_0, q_f\}\}$ (the set of proper suffixes of the words in K),
$Sub = \bigcup\{E(q, s) \mid q, s \in Q \setminus \{q_0, q_f\}\}$ (the set of subwords that are neither prefix nor suffix, of the words in K).

Then the required set $\{x \in V^* \mid x \diamond K \subseteq \{x\}\}$ equals $K \setminus (V^*\overline{Pref} \cup \overline{Suf}V^* \cup \overline{Sub})$. By the closure properties of the family of regular languages we are done. □

6 Solutions to Two NP-Complete Problems Based on Superpositions Suggested by DNA Manipulation

The first problem of this section is the *Hamiltonian Path Problem* (HPP). Let us consider a directed graph $G = (V, E)$, with $V = \{x_1, x_2, \ldots, x_n\}$ for which we are looking for a Hamiltonian path starting with x_1. *A Hamiltonian path in a directed graph is a path which contains all vertices exactly once.* It is known that the HPP is an NP-complete problem.

We consider the following idealized DNA-based algorithm which is quite similar in nature to that proposed in [1]: assume that we have put into a test tube the multiset of single strand DNA molecules (each molecule appears in a sufficiently large number of copies)

$$A = \{\#c_1\#c_j \mid (x_1, x_j) \in E\} \cup \{\overline{\#c_i\#c_j} \mid (x_i, x_j) \in E\},$$

where c_i is an oligonucleotide which encodes the vertex x_i, for each $1 \leq i \leq n$ and $\#$ is a distinct oligonucleotide which does not complementarily match any subsegment of any c_i. Clearly, this technique requires an amount of DNA exponential in the size of the problem instance which is not practical. This is a common difficulty with such techniques as was pointed out originally by Hartmanis, see [9]. By annealing, polymerase and melting the test tube will contain $u \diamond (A)$. We now add again A to the current content of the test tube and resume the process. Hence, by iterating this process for sufficiently long time, we may assume that the test tube contains all words in $u \diamond^* (A)$ long enough for encoding a possible solution to our problem. Let B the contents of the test tube at this stage.

In order to look for a molecule encoding a Hamiltonian path among the molecules in B we apply a procedure known as the *filter method* which consists in keeping all the strands where all the nodes are present, by using some *separation* procedure (e.g. biotyne-streptavidine affinity) and finally we check whether or not there is a molecule of length

$$n \cdot |\#| + |c_1c_2 \ldots c_n|$$

which might be realized by using the *gel electrophoresis*.

The following mathematical algorithm based on the superposition operation is inspired by the first two steps of the aforementioned DNA-based algorithm:

Procedure *Hamiltonian_Path;*
begin
$B := \mu_{n\cdot|\#|+|c_1c_2\ldots c_n|}(A)$;
for *each* $2 \leq i \leq n$
  $B := B \diamond \{\overline{\#c_i\#}\}$;
endfor
end

To our best knowledge, this is a purely mathematical algorithm, unlikely to be realized in a lab context for the time being, because it is based on a noncommutative operation, an intrinsic commutativity being assumed when one tries to implement it.

The second problem we discuss here is the *Bipartite Covering Problem* (BCP) which can be formulated in the following way [13]. *Given a finite set C and n pairs of mutually disjoint subsets of C, (A_i, B_i), $1 \leq i \leq n$, decide whether or not there exist a subset X_i in each pair $1 \leq i \leq n$ such that $C = X_1 \cup X_2 \cup \ldots \cup X_n$.* It can be easily shown that BCP is equivalent to the NP-complete problem, SAT problem, which can be stated as follows: *Given a propositional formula, decide whether or not it can be satisfied for some values of its propositional variables* (see also [13]).

Assume that $C = \{x_1, x_2, \ldots, x_m\}$ for some $m \geq 1$; each set A_i, $1 \leq i \leq n$, is encoded by an oligonucleotide $\$d_{(i,1)}\$ \ldots \$d(i, j_i)\$$, where $A_i = \{x_{(i,1)} \ldots d(i, j_i)\}$, $\$d_{(i,1)}\$ \ldots \$d_{(i,j_i)}\$$, where $A_i = \{x_{(i,1)} \ldots d_{(i,j_i)}\}$, and consider the distinct oligonucleotides $\#_i$, $1 \leq i \leq n$, and $\#$. Now put

$$X = \{\#_1\} \cup \{\overline{\#_s d_s \#_{s+1}} \mid 1 \leq i \leq n-1\} \cup \{\overline{\#_n d_n \#}\}$$

and proceed as above to generate $max\ \diamond^*(X)$. By the separation sequence in the above abstract algorithm, inspired from the filter method, one checks the existence of a maximal word which contains all segments d_i, $1 \leq i \leq m$. Such a maximal word exists if and only if the given instance of BCP has solutions.

7 Final Remarks

We have presented a new operation which provides a way to simulate (in well-defined cases) several formal operations commonly employed in DNA computing. In particular, we have shown how the iteration of this operation provides a theoretical model for some concrete operations performed in DNA-based procedures.

It is easy to notice that many results obtained here remain valid for a definition which makes use of commutativity, namely the result of superposition applied to x and y is the union $x \diamond y \cup y \diamond x$.

We recall here two open problems which appear attractive to us:

1. Is $\diamond^*(L)$ regular, provided that L is regular?

2. If the answer is no, is the maximal language w.r.t. the iterated superposition closure of a regular language recursive?

We now briefly discuss a few other possible formal operations on words inspired by the three biological phenomena which the superposition operation is based on. It is known that a single stranded DNA molecule might produce a hairpin structure. In many DNA-based algorithms, these DNA molecules cannot be used in the subsequent computations. In a series of papers (see, e.g., [5, 7, 8]) the problem of finding sets of DNA sequences which are unlikely to lead to "bad" hybridizations is considered. On the other hand, these molecules which may form a hairpin structure have been used as the basic feature of a new computational model reported in [21], where an instance of the 3-SAT problem has been solved by a DNA-algorithm in which the second phase is mainly based on the elimination of hairpin structured molecules. Different types of hairpin languages are defined in [16] and [3] where they are studied from a language theoretical point of view. One more superposition operation, which is now a unary operation, might be defined as follows: a word $uvw\overline{v}xy$ in which a hairpin structure determined by the complementarity of v and $\overline{v}$ appears produces the new word $\overline{y}uvw\overline{v}xy$. This *hairpin superposition* operation which seems to be mathematically attractive is schematically illustrated in Figure 8.

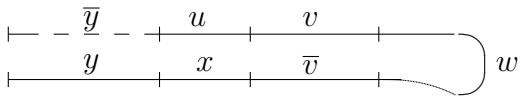

Figure 8: Hairpin superposition

Another superposition operation, more complicated, but based on a fact rather frequent in nature, is informally described for two single stranded DNA molecules x and y as follows: in the aim of being Watson-Crick complementary to a prefix of y, a suffix of x makes some loops. Then x and y get annealed in a DNA molecule with a double stranded part (the upper strand having some loops) by complementary base pairing and then a complete double stranded molecule is formed by DNA polymerases. Clearly, the same may happen with the prefix of y. Furthermore, all cases 1-4 of Figure 1 may be modified in this respect. We call it *superposition with compensation loops*. For a better understanding we illustrate this case in Figure 9, where the superposition with compensation loops applied to the pair of words (x, y) results in the word $x\overline{z}$.

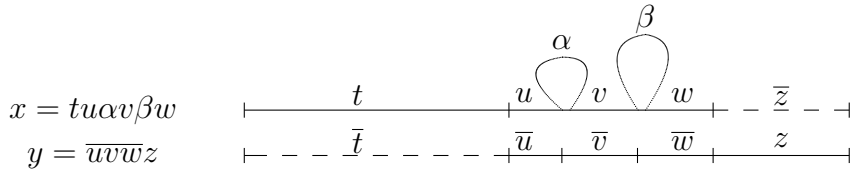

Figure 9: Superposition with compensation loops

Once again, it turns out that manipulation of DNA molecules is the source of inspiration for interesting operations from the formal language theory point of view. We hope to return to these topics in a forthcoming paper.

Acknowledgments. The authors are thankful to the editor as well as to the referees for their valuable comments and suggestions which improved the presentation.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226(1994), 1021–1024.

[2] P. Bottoni, G. Mauri, P. Mussio, Gh. Păun, Grammars working on layered strings, *Acta Cybernetica*, 13(1998), 339–358.

[3] J. Castellanos, V. Mitrana, Some remarks on hairpin and loop languages, *Words, Semigroups, and Tranlations*, (M. Ito, Gh. Păun, S. Yu, eds.), World Scientific, Singapore, 2001 47–59.

[4] K. Culik II, T. Harju, The regularity of splicing systems and DNA, *Proc. ICALP 1989, LNCS 372*, 1989, 222–233.

[5] R. Deaton, R. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, Good encodings for DNA-based solutions to combinatorial problems, *Proc. of DNA-based computers II*, (L.F. Landweber, E. Baum, eds.), DIMACS Series, vol. 44, 1998, 247–258.

[6] R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, Bidirectional sticker systems, *Third Annual Pacific Conf. on Biocomputing*, Hawaii, 1998 (R.B. Altman, A.K. Dunker, L. Hunter, T.E. Klein, eds.), World Scientific, Singapore, 1998, 535–546.

[7] M. Garzon, R. Deaton, P. Neathery, R.C. Murphy, D.R. Franceschetti, E. Stevens, On the encoding problem for DNA computing, *The Third DIMACS Workshop on DNA-Based Computing*, Univ. of Pennsylvania, 1997, 230–237.

[8] M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens Jr., M. Wittner, Genome encoding for DNA computing, *Proc. Third Genetic Programming Conference*, Madison, MI, 1998, 684–690.

[9] J. Hartmanis, On the weight of computations, *Bulletin of the EATCS*, 55(1995), 136–138.

[10] T. Head, Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors, *Bulletin of Mathematical Biology*, 49(1987), 737–759.

[11] L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu, DNA computing, sticker systems, and universality, *Acta Informatica*, 35, 5(1998), 401–420.

[12] S. Kobayashi, V. Mitrana, Gh. Păun, G. Rozenberg, Formal properties of PA-matching, *Theoretical Comput. Sci.*, 262, 1-2(2001), 117–131.

[13] V. Manca, S. Di Gregorio, D. Lizzari, G. Vallini, C. Zandron, A DNA algorithm for 3-SAT(11,20), *Proc. 7th Intern. Meeting on DNA Based Computers* (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 167–177.

[14] Gh. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998, Tokyo, 1999.

[15] Gh. Păun, G. Rozenberg, Sticker systems, *Theoret. Comput. Sci.*, 204(1998), 183–203.

[16] Gh. Păun, G. Rozenberg, T. Yokomori, Hairpin languages, *Intern. J. Found. Comp. Sci.* 12, 6(2001), 837–847.

[17] D. Pixton, Regularity of splicing languages, *Discrete Applied Mathematics* 69, 1-2(1996), 101–124.

[18] J.H. Reif, Parallel molecular computation: Models and simulations, *Proc. of Seventh Annual ACM Symp. on Parallel Algorithms and Architectures*, Santa Barbara, 1995, 213–223.

[19] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems.* Academic Press, New York, 1980.

[20] G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, Heidelberg, 1997.

[21] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya, Molecular computation by DNA hairpin formation, *Science* 288(2000), 1223–1226.

[22] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.