# Some Remarks on Superposition Based on Watson-Crick-Like Complementarity*

Florin Manea[1], Victor Mitrana[1,2], and Jose M. Sempere[2]

[1] Faculty of Mathematics and Computer Science,
University of Bucharest,
Str. Academiei 14, 70109, Bucharest, Romania
flmanea@gmail.com, mitrana@fmi.unibuc.ro
[2] Department of Information Systems and Computation
Technical University of Valencia,
Camino de Vera s/n. 46022 Valencia, Spain
jsempere@dsic.upv.es

**Abstract.** In this note we solve a problem left open in [2]: namely, we show that the iterated superposition of a regular language is regular. The proof of this result is based on two facts: (i) the iterated superposition of a language equals the restricted iterated superposition of the same language, (ii) the sequence formed by iteratively applying the restricted superposition can be precisely defined. We then define the restricted superposition distance between a word and a language and prove that this distance can be computed in time $\mathcal{O}(n^2 f(n))$, where the language is accepted in time $\mathcal{O}(f(n))$ in the RAM model. Finally, we briefly discuss the necessity of the $n^2$ factor for the classes of regular and context-free languages.

## 1 Introduction

A DNA molecule consists of a double strand, each DNA single strand being composed of 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 bonds with G. This paradigm of *Watson-Crick complementarity* is one of the main concepts used in defining the formal operation of superposition investigated in [2].

Two other biological principles used as sources of inspiration in that paper are those of *annealing* and *lengthening DNA by polymerase*, respectively. 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

* Work partially supported by the PN II projects 11052(GlobalComp) and 11056(CellSim).

V. Diekert and D. Nowotka (Eds.): DLT 2009, LNCS 5583, pp. 372–383, 2009.

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 polymerase 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 polymerase. 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 both words, namely the upper one, formed by the prolongation of $x$, and lower one, formed by the prolongation of $y$, are 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' in DNA biochemistry) as well as in both, despite that in biology almost all polymerase extend in the direction from 5' to 3'.

As shown in [2], this operation resembles some other operations on words: *sticking* investigated in [5,3,9] (particular polyominoes with sticky ends are combined provided that the sticky ends are Watson-Crick complementary), *PA-matching* considered in [7] which is related to both the *splicing* and the *annealing* operations, and the *superposition* operation introduced in [1] (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 reader interested in related bio-inspired operations is referred to [8] and [6].

In this work we propose a slightly different variant of the operation considered in [2]. It turns out that both operations coincide when they are applied to a language. This variant allows us to solve a problem left open in [2]: namely, we show that the iterated superposition of a regular language is regular. The proof of this result is based on two facts: (i) a sort of normal form which implies that the iterated superposition of a language equals the restricted iterated superposition of the same language, (ii) the sequence formed by iteratively applying the restricted superposition can be precisely defined. We then define the restricted superposition distance between a word and a language and prove that this distance can be computed in time $\mathcal{O}(n^2 f(n))$, where the language is accepted in

time $\mathcal{O}(f(n))$ in the RAM model. Finally, we briefly discuss the necessity of the $n^2$ factor for the classes of regular and context-free languages.

## 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 [10].

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

Let $\Omega$ be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of $\Omega$. In other words, $\Omega$ is the *universe* of the languages in this paper, i.e., all words and languages are over alphabets that are subsets of $\Omega$. An *involution* over a set $S$ is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution $\sigma$ on $\Omega$ 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 polymerase, where the Watson-Crick complementarity plays an important role. Let $\overline{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from $\Omega^*$ to $\Omega^*$ 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 $\overline{A} = T$, $\overline{C} = G$.

### 2.1 Non-iterated Superposition

Given two words $x, y \in V^+$ we define the following operations:

$$
\begin{aligned}
x \blacktriangleright y &= \{uw\overline{v}, \overline{uw}v \mid x = uw, y = \overline{w}v \text{ for some } u, v \in V^*, w \in V^+\} \\
x \blacktriangleleft y &= \{\overline{u}wv, u\overline{wv} \mid x = wv, y = u\overline{w} \text{ for some } u, v \in V^*, w \in V^+\} \\
x \blacktriangle y &= \{\overline{u}x\overline{v}, y \mid y = u\overline{x}v \text{ for some } u, v \in V^*\} \\
x \blacktriangledown y &= \{x, \overline{u}y\overline{v} \mid x = u\overline{y}v \text{ for some } u, v \in V^*\}.
\end{aligned}
$$

Clearly, $x \blacktriangleright y = y \blacktriangleleft x$ and $x \blacktriangle y = y \blacktriangledown x$ for any pair of words $x, y$. Despite this redundancy we prefer to work with these definitions because they allow a simplification of the arguments we are to discuss.

We now define the *superposition* operation applied to the pair of words $x, y \in V^+$ as above, denoted by $\blacklozenge$, as follows:

$$x \blacklozenge y = (x \blacktriangleright y) \cup (x \blacktriangleleft y) \cup (x \blacktriangle y) \cup (x \blacktriangledown y).$$

The result of this operation applied to the two words $x$ and $y$ as above which might be viewed as two single stranded molecules is the pair of words formed by $z$ and its complement $\overline{z}$ which form a double stranded molecule $\frac{z}{\overline{z}}$. 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 [8]. Note the difference between this operation and the superposition operation defined in [2], where only the upper word is considered to be the result of superposition.

We stress from the very beginning the mathematical character of the definition proposed in [2]: nature cannot distinguish which is the upper or the lower strand in the process of constructing a double stranded molecule from two single strands. This drawback is avoided by the definition proposed here. 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

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

We write $\blacklozenge(L)$ instead of $L \blacklozenge L$. It is plain that the superposition operation proposed in [2] and that proposed here coincide when they are applied to a language.

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

### 2.2 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 $\blacklozenge_u^*(L)$ and defined by

$$\begin{aligned} \blacklozenge_u^0(L) &= L, \\ \blacklozenge_u^{i+1}(L) &= \blacklozenge_u^i(L) \cup \blacklozenge(\blacklozenge_u^i(L)), i \geq 0, \\ \blacklozenge_u^*(L) &= \bigcup_{i \geq 0} \blacklozenge_u^i(L). \end{aligned}$$

Clearly, $\blacklozenge_u^*(L)$ is the smallest language containing $L$ and closed under superposition. More precisely, it is the smallest language $K$ such that $L \subseteq K$ and $\blacklozenge(K) \subseteq K$. In words, one starts with the words in $L$ and applies superposition iteratively 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 $\blacklozenge_u^*(L)$.

We say that a family $\mathcal{F}$ of languages is closed under unrestrictedly iterated superposition if $\blacklozenge_u^*(L)$ is in $\mathcal{F}$ for any language $L \in \mathcal{F}$.

We now recall from [2] another superposition closure of a language which may be viewed as a "normal form" of iterated superposition. The *restricted superposition closure of* $L$ denoted by $\blacklozenge_r^*(L)$ is defined in the following way:

$$\begin{aligned} \blacklozenge_r^0(L) &= L, \\ \blacklozenge_r^{i+1}(L) &= ((\blacklozenge_r^i(L))\blacklozenge L) \cup (\blacklozenge_r^i(L)) \\ \blacklozenge_r^*(L) &= \bigcup_{i \geq 0} \blacklozenge_r^i(L). \end{aligned}$$

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.

## 3 Iterated Superposition Preserves Regularity

Note that $\blacklozenge_r^*(L) \subseteq \blacklozenge_u^*(L)$ for any language $L$. Surprisingly enough (remember that $\blacklozenge$ is not associative), we have an equality between the two superposition closures of any language.

**Theorem 1.** [Normal Form Theorem][2] $\blacklozenge_r^*(L) = \blacklozenge_u^*(L)$ *for any language* $L$.

This theorem allows us to use the notation $\blacklozenge^*$ when the way of iterating the superposition does not matter. The problem of closure under iterated superposition of the class of regular languages was left open in [2]. We propose here an affirmative answer to this question. To this aim, we need some preliminary results. The redundancy introduced in the definition of the superposition operation turns out to be useful now. For two languages $L_1, L_2$ we define

$$\begin{aligned} (i)\ & \blacktriangleleft (L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} x \blacktriangleleft y, \\ (ii)\ & \blacktriangleleft^0 (L_1, L_2) = L_1, \\ (iii)\ & \blacktriangleleft^{i+1} (L_1, L_2) = \blacktriangleleft ((\blacktriangleleft^i (L_1, L_2)), L_2), i \geq 0, \\ (iv)\ & \blacktriangleleft^* (L_1, L_2) = \bigcup_{i \geq 0} \blacktriangleleft^i (L_1, L_2). \end{aligned}$$

The language $\blacktriangleright^* (L_1, L_2)$ is defined analogously.

**Lemma 1.** *For every language* $L$ *and any integer* $k \geq 1$*, the following relations hold:*

$$(\blacklozenge_r^k(L) \backslash L) = \bigcup_{0 \leq n+m < k} \blacktriangleright^n (\blacktriangleleft^m (\blacklozenge(L), L), L) = \bigcup_{0 \leq n+m < k} \blacktriangleleft^n (\blacktriangleright^m (\blacklozenge(L), L), L).$$

*Proof.* We prove the first relation only; the second one can be easily proved analogously. It is plain that

$$\bigcup_{0\le n+m<k} \blacktriangleright^n (\blacktriangleleft^m (\blacklozenge(L), L), L) \subseteq ((\blacklozenge_r^k(L)) \setminus L).$$

Let $w \in (\blacklozenge_r^k(L) \setminus L)$, we prove that $w \in \blacktriangleright^n (\blacktriangleleft^m (\blacklozenge(L), L), L)$ for some $0 \le n + m < k$ by induction on $k$. The assertion is immediately true for $k = 1$. A simple observation makes the assertion true for $k = 2$ as well. Indeed, it is clear that $((x\blacktriangle y) \cup (x\blacktriangledown y)) \subseteq \blacklozenge(L)$ for any $x \in \blacklozenge(L)$ and $y \in L$.

Let $w \in \blacklozenge_r^{k+1}(L)$ for some $k \ge 2$; there exist $x \in \blacklozenge_r^k(L)$ and $y \in L$ such that $w \in x\blacklozenge y$. We distinguish four cases:

1. $w \in x \blacktriangleright y$. By the induction hypothesis, $x \in \blacktriangleright^n (\blacktriangleleft^m (\blacklozenge(L), L), L)$ for some $0 \le n+m < k$, hence $w \in \blacktriangleright^{n+1} (\blacktriangleleft^m (\blacklozenge(L), L), L)$ with $0 \le n+1+m < k+1$ holds.
2. $w \in x \blacktriangleleft y$. If $x \in \blacktriangleright^0 (\blacktriangleleft^m (\blacklozenge(L), L), L)$, then $w \in \blacktriangleright^0 (\blacktriangleleft^{m+1} (\blacklozenge(L), L), L)$ holds. Assume that $x \in \blacktriangleright^n (\blacktriangleleft^m (\blacklozenge(L), L), L)$ with $n \ge 1$; it follows that there exist $u \in \blacktriangleright^{n-1} (\blacktriangleleft^m (\blacklozenge(L), L), L)$ and $v \in L$ such that $x \in u \blacktriangleright v$. It further follows that $\{u, \overline{u}\} \subseteq \blacklozenge_r^{n+m}$. As $n + m < k$, we infer that $\blacktriangleleft (\{u, \overline{u}\}, y) \subset \blacklozenge_r^k(L)$. Further, $\blacktriangle(\{u, \overline{u}\}, y) \subset \blacklozenge_r^k(L)$ (see also the next item) and $w \in s \blacktriangleright v$ for some $s \in \blacktriangleleft (\{u, \overline{u}\}, y)$. By the induction hypothesis, $s \in \blacktriangleright^p (\blacktriangleleft^q (\blacklozenge(L), L), L)$ holds for some $0 \le p + q < k$, therefore $w \in \blacktriangleright^{p+1} (\blacktriangleleft^q (\blacklozenge(L), L), L)$ with $0 \le p + 1 + q < k + 1$ holds as well.
3. $w \in x\blacktriangle y$. This case immediately leads to $w \in \blacklozenge_r^k(L)$.
4. $w \in x\blacktriangledown y$. This case immediately leads to $w \in \blacklozenge_r^k(L)$ and we are done.

□

A direct consequence of this lemma is the following corollary.

**Corollary 1.** *For every language $L$, the following relations hold:*

$$\blacklozenge_r^*(L) = \blacktriangleright^* (\blacktriangleleft^* (\blacklozenge(L), L), L) \cup L = \blacktriangleleft^* (\blacktriangleright^* (\blacklozenge(L), L), L) \cup L.$$

We still need one more result. We start with some additional notation. For two words $x, y$ we denote

$$x \blacktriangleright_{\frac{1}{2}} y = \{uw\overline{v} \mid x = uw, y = \overline{w}v \text{ for some } u \in V_1^*, w \in V_1^+, v \in V_2^*\}$$
$$x \blacktriangleleft_{\frac{1}{2}} y = \{\overline{u}wv \mid x = wv, y = u\overline{w} \text{ for some } u \in V_2^*, w \in V_1^+, v \in V_1^*\}.$$

For two languages $L_1, L_2$ we define

$$(i)\ \blacktriangleleft_{\frac{1}{2}} (L_1, L_2) = \bigcup_{x\in L_1, y\in L_2} x \blacktriangleleft_{\frac{1}{2}} y,$$
$$(ii)\ \blacktriangleleft_{\frac{1}{2}}^0 (L_1, L_2) = L_1,$$
$$(iii)\ \blacktriangleleft_{\frac{1}{2}}^{i+1} (L_1, L_2) = \blacktriangleleft_{\frac{1}{2}} ((\blacktriangleleft_{\frac{1}{2}}^i (L_1, L_2)), L_2), i \ge 0,$$
$$(iv)\ \blacktriangleleft_{\frac{1}{2}}^* (L_1, L_2) = \bigcup_{i\ge 0} \blacktriangleleft_{\frac{1}{2}}^i (L_1, L_2).$$

The language $\blacktriangleright_{\frac{1}{2}}^* (L_1, L_2)$ is defined analogously.

**Lemma 2.** *For every language $L$ the following relations hold:*
*1.* $\blacktriangleleft^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L}) = \blacktriangleleft^* (\blacklozenge(L), L)$.
*2.* $\blacktriangleright^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L}) = \blacktriangleright^* (\blacklozenge(L), L)$.

*Proof.* We prove the first relation only; the second one can be shown analogously. We first consider the inclusion $\blacktriangleleft^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L}) \subseteq \blacktriangleleft^* (\blacklozenge(L), L)$.

Let $w \in \blacktriangleleft^k_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L})$; the inclusion immediately holds for $k = 0$. Assume $w \in \blacktriangleleft^{k+1}_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L})$, there exist $x \in \blacktriangleleft^k_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L})$ and $y \in (L \cup \overline{L})$ such that $w \in x \blacktriangleleft_{\frac{1}{2}} y$. If $y \in L$, then $w \in \blacktriangleleft^* (\blacklozenge(L), L)$ by the induction hypothesis. If $y \in \overline{L}$, then $\overline{y} \in L$ and $w \in \overline{x} \blacktriangleleft \overline{y}$ which concludes the proof of this part as soon as we note that $\overline{x} \in \blacktriangleleft^* (\blacklozenge(L), L)$.

Conversely, let $w \in \blacktriangleleft^k (\blacklozenge(L), L)$. The converse inclusion holds for $k = 0$. Assume $w \in \blacktriangleleft^{k+1} (\blacklozenge(L), L)$; there exist $x \in \blacktriangleleft^k (\blacklozenge(L), L)$ and $y \in L$ such that $w \in x \blacktriangleleft y$. If $w = \overline{u}wv$, where $x = wv, y = u\overline{w}$, then $w \in \blacktriangleleft^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L})$. If $w = u\overline{wv}$, where $x = wv, y = u\overline{w}$, then $w \in \overline{x} \blacktriangleleft_{\frac{1}{2}} \overline{y}$. Since $\overline{y} \in \overline{L}$ and $\overline{x} \in \blacktriangleleft^k (\blacklozenge(L), L)$, hence $x \in \blacktriangleleft^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L})$ by the induction hypothesis. The proof is now complete. □

**Corollary 2.** *For every language $L$, the following relations hold:*

$$\blacklozenge^*_r(L) = \blacktriangleright^*_{\frac{1}{2}} (\blacktriangleleft^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L}), L \cup \overline{L}) \cup L = \blacktriangleleft^*_{\frac{1}{2}} (\blacktriangleright^*_{\frac{1}{2}} (\blacklozenge(L), L \cup \overline{L}), L \cup \overline{L}) \cup L.$$

We are now ready to prove one of the main results of this note.

**Theorem 2.** *$\blacklozenge^*(L)$ is always regular for any regular language $L$. In other words, the class of regular languages is closed under iterated superposition.*

*Proof.* We start by recalling a result proved in [2], namely $\blacklozenge(L)$ is regular for every regular language $L$. By the previous corollary, it suffices to prove that $\blacktriangleright^*_{\frac{1}{2}} (E, L \cup \overline{L})$ is regular provided that $E$ is regular.

Let $L \subseteq V^*$ be a regular language; we assume that the deterministic finite automaton $A = (Q, V, \delta, q_0, F)$ accepts $L \cup \overline{L}$. For every state $q \in Q$ we define the regular language $R(q) = (V \cup \overline{V})^* \{\overline{w} \mid w \in V^+, \delta(q_0, w) = q\}$ accepted by the (not necessarily deterministic) finite automaton $A^{(q)} = (Q^{(q)}, V \cup \overline{V}, \delta^{(q)}, s_0^{(q)}, \{s_f^{(q)}\})$. Note that all automata $A^{(q)}$, $q \in Q$, have a single final state.

We now define the following left-linear grammar $G = (N, V \cup \overline{V} \cup \{Z_X \mid X \subseteq (Q^{(q)} \times Q^{(q)}), q \in Q\}, S, P)$, where

$$N = \{S\} \cup (\bigcup_{q \in Q} \{[X, q], [X, q, q'] \mid X \subseteq (Q^{(q)} \times Q^{(q)}), q' \in Q\})$$
$$\cup (\bigcup_{q \in Q} \{\langle X, q\rangle, \langle X, q, q'\rangle \mid X \subseteq (Q^{(q)} \times Q^{(q)}), q' \in Q\}).$$

The set of productions $P$ contains the following rules (each set of rules is accompanied with some explanations):

1. For every $q \in Q$, we add the complement of $w$ as the suffix to the current word provided that $\delta(q, w) \in F$ and the current word lies in $R(q)$. We memorize to check the last condition in the pair $(s_0^{(q)}, s_f^{(q)})$ which must be completed to obtain a non-deterministically guessed correct recognition.

   $S \to [\{(s_0^{(q)}, s_f^{(q)})\}, q]$ for all $q \in Q, X \subseteq (Q^{(q)} \times Q^{(q)})$
   $[X, q] \to [X, q, q']\overline{a},$ if $\delta(q', a) \in F,$ for all $q, q' \in Q, X \subseteq (Q^{(q)} \times Q^{(q)})$
   $[X, q, q'] \to [X, q, q'']\overline{a},$ if $q' \in \delta(q'', a),$ for all $q, q', q'' \in Q, X \subseteq (Q^{(q)} \times Q^{(q)})$
   $[X, q, q] \to X$ for all $q \in Q, X \subseteq (Q^{(q)} \times Q^{(q)})$.

2. For every $q \in Q$, we add the complement of $w$ as the suffix to the current word provided that $\delta(q, w) \in F$ and the current word lies in $R(q)$. Beside memorizing to check the last condition in the pair $(s_0^{(q)}, s_f^{(q)})$ as above we simultaneously continue guessing the appropriate paths in all automata $A^{(q')}$ for $q' \in Q$.

   $$X \to \langle X, q\rangle \text{ for all } q \in Q, X \subseteq (Q^{(q)} \times Q^{(q)})$$
   $$\langle X, q\rangle \to \langle X', q, q'\rangle\overline{a}, \text{ where}$$

   – $\delta(q', a) \in F$,
   – if $(s^{(r)}, t^{(r)}) \in X$ for some $r \in Q$, then $X'$ contains one pair $(s^{(r)}, p^{(r)})$ such that $t^{(r)} \in \delta^{(r)}(p^{(r)}, \overline{a})$,

   $$\langle X, q, q'\rangle \to \langle X', q, q''\rangle\overline{a}, \text{ where}$$

   – $\delta(q'', a) = q'$,
   – if $(s^{(r)}, t^{(r)}) \in X$ for some $r \in Q$, then $X'$ contains one pair $(s^{(r)}, p^{(r)})$ such that $t^{(r)} \in \delta^{(r)}(p^{(r)}, \overline{a})$,

   $$\langle X, q, q\rangle \to X \cup \{(s_0^{(q)}, s_f^{(q)})\}.$$

3. $X \to Z_X$.

We claim $\blacktriangleright_{\frac{1}{2}}^{*} (E, L \cup \overline{L}) = s(L(G))$, where $s$ is a substitution $s : (V \cup \overline{V} \cup \{Z_X \mid X \subseteq (Q^{(q)} \times Q^{(q)}), q \in Q\})^* \longrightarrow 2^{(V \cup \overline{V})^*}$ defined by $s(a) = a$ for any $a \in V \cup \overline{V}$ and

$$s(Z_X) = \{w \in (V \cup \overline{V})^* \mid \forall q \in Q, \forall (s^{(q)}, t^{(q)}) \in X, t^{(q)} \in \delta^{(q)}(s^{(q)}, w)\} \cap E.$$

For $s$ is a substitution by regular languages, it follows that the language $\blacktriangleright_{\frac{1}{2}}^{*}$ $(E, L \cup \overline{L})$ is regular. □

## 4 Restricted Superposition Distance

In this section we discuss the following problem: Given a language $L$ and a word $w \in \blacklozenge_r^*(L)$, compute the minimal value $i$ such that $w \in \blacklozenge_r^i(L)$. To this aim, we first recall a well-known problem whose solution will be useful, namely the so-called *Range Minimum (Maximum) Query Problem*: Given an array $A$ with entries over a totally ordered set can we preprocess $A$ (i.e., produce some additional data structures), such that we can answer efficiently queries like $RmQ_A(i,j) = \text{argmin}_{i \leq t \leq j} A[t]$ or $RMQ_A(i,j) = argmax_{i \leq t \leq j} A[t]$? In both cases, when there is a tie, the leftmost possible value is returned. Harel and Tarjan [4] proposed a solution for this problem such that, if $A$ has $n$ elements, then the preprocessing is performed in time $\mathcal{O}(n)$ (and a data structure of size $\mathcal{O}(n)$, called RmQ or RMQ, respectively, is produced), while the answer to any query can be obtained in time $\mathcal{O}(1)$.

Note that in the following we will use the word "derivation" with the meaning "application of the Watson-Crick superposition".

**Theorem 3.** *Let $L$ be a language over the alphabet $V$ accepted in time $\mathcal{O}(f(n))$ on the RAM model, and $w \in V^*$. One can compute the minimum value $i$ such that $w \in \blacklozenge_r^i(L)$ in time $\mathcal{O}(|w|^2 f(|w|))$.*

*Proof.* Let $i$ be the minimum value such that $w \in \blacklozenge_r^i(L)$. It follows that there exists the sequence $w_0, w_1, \ldots, w_i$, such that $w_0 \in L$, $w_i = w$, and $w_t \in (w_{t-1} \circ_t z_t)$, where $\circ_t \in \{\blacktriangleright, \blacktriangleleft, \blacktriangle, \blacktriangledown\}$ and $z_t \in L$, for all $t \in \{1, \ldots, i\}$. It is clear that both $w_t$ and $\overline{w_t}$, for all $t \geq 1$, can be obtained in $t$ steps starting from $w_0$. Without loss of generality, we may assume that $w_0$ is a subword of $w$. By Lemma 1, we may further assume that $\circ_1 \in \{\blacktriangleright, \blacktriangleleft, \blacktriangle, \blacktriangledown\}$ and $\circ_t = \blacktriangleleft$, $2 \leq t \leq p$, $\circ_t = \blacktriangleright$, $p+1 \leq t \leq i$, for some $1 \leq p \leq i$. Note that no operation $\blacktriangleleft$ or $\blacktriangleright$ is applied when $p = 1$ or $p = i$, respectively. Moreover, by the proof of Lemma 1 we may assume that $\circ_t = \blacktriangleright$ for all $2 \leq t \leq i$ provided that $\circ_1 = \blacktriangleright$.

We now focus our discussion on the way the words from $L$ on which we can apply these operations could be identified. The idea is quite simple, and it corresponds to a greedy strategy: in the first step we identify all the subwords of $w$ that are words in $L$. Then, we add to these words all the subwords of $w$ that can be obtained from the words got in the first step using the $\blacktriangle$ and $\blacktriangledown$ operations together with their complements. Then, we try to obtain $w$ starting from each of these words, and compute the minimum number of operations needed to do this. Consequently, we proceed as follows: if the current word is not a prefix of $w$, or the complement of such a prefix, we choose a word from $L$ which we can apply the operation $\blacktriangleleft$ to, such that at least one of the obtained words is a subword of $w$ and the length of this word is maximal among all the words that can be obtained by applying the $\blacktriangleleft$ operation to the current word.

If the current word is a prefix of $w$, then we choose a word from $L$ which we can apply the operation $\blacktriangleright$ to, such that at least one of the words we obtain is a subword of $w$ and the length of this word is maximal between all the words that can be obtained by applying the $\blacktriangleright$ operation to the current word.

It is plain that this strategy that is actually dynamic programming works as the new current word is always a subword of $w$. In the following, we show how this strategy can be implemented in time $\mathcal{O}(n^2 f(n))$. If the input word $w$ belongs to $\blacklozenge_r^*(L)$, then the algorithm outputs the minimal $i$ such that $w \in \blacklozenge_r^i(L)$, otherwise it outputs $\infty$. Data structures used by the algorithm are:

– The 2-dimensional arrays $M$, $C$, $T$, $CT$, $D$ and $N$, with $|w|$ rows and columns.

– The 1-dimensional arrays $Left$, $Right$, $CLeft$ and $CRight$, with $|w|$ positions. The values stored in these arrays are initially set to 0.

– The queue $Q$ that is initially empty.

**Algorithm 1**

function $Distance(w, L)$;
begin
$n := |w|$;
for $l = 1$ to $n$ do
  for $i = 1$ to $n - l + 1$ do
    $j = i + l - 1$;
    if $w[i..j] \in L$ then $M[i][j] = 1$; $Right[i] = j, Left[j] = i$;
      Add $([i, j], 1, 0)$ to $Q$, $D[i][j] = 1$;
    endif
    if $\overline{w[i..j]} \in L$ then $C[i][j] = 1, CRight[i] = j, CLeft[j] = i$;
  endfor
endfor
if $(D[1][n] = 1)$ then return 0;
for $l = 1$ to $n$ do
  for $i = 1$ to $n - l + 1$ do
    $j = i + l - 1$;
    $CT[i][j] = \max\{C[i][j], CT[i-1][j], CT[i][j-1]\}$;
    $T[i][j] = \max\{M[i][j], T[i-1][j], T[i][j-1]\}$;
    if $(C[i][j] = 1$ & $T[i][j] = 1$ & $D[i][j] \neq 1)$ then Add $([i, j], 1, 1), ([i, j], -1, 1)$ to $Q$,
      $N[i][j] = 1, D[i][j] = 1$;
    endif
    if $(M[i][j] = 1$ & $CT[i][j] = 1$ & $N[i][j] \neq 1)$ then Add $([i, j], -1, 1)$ to $Q$,
      $N[i][j] = 1$;
    endif
  endfor
endfor
Compute *the RmQ data structures for the arrays* $Left, CLeft$;
Compute *the RMQ data structures for the arrays* $Right, CRight$;
$b = false$
while $(b = false$ & $Q$ *not empty)* do
  Extract $([i, j], x, k)$ from $Q$
  if $(x = 1$ & $i \neq 1)$ then
    $t = RmQ_{CLeft}(i, j)$;
    if $D[t][j] \neq 1$ then Add $([t, j], 1, k + 1)$ to $Q$, $D[t][j] = 1$;

```
    if N[t][j] ≠ 1 then Add ([t, j], −1, k + 1) to Q, N[t][j] = 1;
  endif
  if (x = 1 & i = 1) then
    t = RMQ_CRight(i, j);
    if D[i][t] ≠ 1 then Add ([i, t], 1, k + 1) to Q, D[i][t] = 1;
    if N[i][t] ≠ 1 then Add ([i, t], −1, k + 1) to Q, N[i][t] = 1;
  endif
  if (x = −1 & i ≠ 1) then
    t = RmQ_Left(i, j);
    if D[t][j] ≠ 1 then Add ([t, j], 1, k + 1) to Q, D[t][j] = 1;
    if N[t][j] ≠ 1 then Add ([t, j], −1, k + 1) to Q, N[t][j] = 1;
  endif
  if (x = −1 & i = 1) then
    t = RMQ_Right(i, j);
    if D[i][t] ≠ 1 then Add ([i, t], 1, k + 1) to Q, D[i][t] = 1;
    if N[i][t] ≠ 1 then Add ([i, t], −1, k + 1) to Q, N[i][t] = 1;
  endif
  if (D[1][n] = 1) then b = true;
endwhile;
  if (D[1][n] = 1) then find in Q the first tuple ([1, n], 1, i), ∀i; return i;
  else return ∞;
end.;
```

Informally, the algorithm works as follows:

- First, the subwords of $w$ that are words from $L$ are identified. If $w[i..j]$ is such a word then we insert the item $([i, j], 1, 0)$ in the queue $Q$. These are the only words that can be obtained from a subword of $w$ in 0 derivation steps. Also, we use the arrays $Left$ (and $CLeft$) to store the starting position in $w$ of the longest word from $L$ (respectively $\overline{L}$) ending on a certain position in $w$, while the arrays $Right$ (and $CRight$) are used to store the ending position of the longest word from $L$ (respectively $\overline{L}$) starting on a certain position.
- Then, we identify the words that can be obtained from a subsequence of $w$ using a single application of the rules ▲, ▼: if $\overline{w}[i..j]$ can be obtained we add $([i, j], -1, 1)$ to $Q$, if $w[i..j]$ can be obtained we add $([i, j], 1, 1)$ to $Q$.
- Finally, using the queue $Q$ (in which the tuples $([i, j], x, k)$ are ordered increasingly according to the value $k$), we try to obtain new words (actually, the longest words) that can be derived from a subword of $w$, and still remain subwords of $w$ or $\overline{w}$. Each time when a new item $([i, j], 1, k)$ is extracted from $Q$ we try to extend it as much as possible to the left (if $i \neq 1$), or to the right (if $i = 1$), and add items corresponding to the newly obtained words to the queue. The same strategy is used in the case when an item of the form $([i, j], -1, k)$ is extracted from $Q$.
- The algorithm ends in two situations (i) no more words can be obtained and $w$ was not obtained yet, when it returns $\infty$, (ii) $w$ was obtained, when it returns the minimum number of derivation steps used to obtain this word.

It is clear that the first two for cycles can be executed in $\mathcal{O}(n^2 f(n))$ time, while the next two ones can be executed in $\mathcal{O}(n^2)$ time. Next, $Q$ contains an item with the first component $[i, j]$ at most two times during the computation, thus the maximum number of elements that may enter in $Q$ is $\mathcal{O}(n^2)$. Consequently, the while cycle is executed at most $\mathcal{O}(n^2)$ times and every computation done in this cycle is executed in constant time. This shows that the overall time complexity of the above algorithm is $\mathcal{O}(n^2 f(n))$. The space needed by this algorithm is $\mathcal{O}(n^2 S(n))$, given that $L$ is accepted by a RAM in $\mathcal{O}(f(n))$ time and $\mathcal{O}(S(n))$ space. □

It is clear that using some preprocessing which replaces the part of the algorithm consisting of the first two for cycles we can obtain an overall complexity of $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space for regular languages. In the case of context-free languages we can obtain an overall complexity of $\mathcal{O}(n^3)$ time, using the Cocke-Younger-Kasami algorithm in the preprocessing phase, and $\mathcal{O}(n^2)$ space.

As a corollary of the previous theorem we can state:

**Theorem 4.** *The class* **P** *is closed under iterated superposition.*

## References


1. Bottoni, P., Mauri, G., Mussio, P., Păun, G.: Grammars working on layered strings. Acta Cybernetica 13, 339–358 (1998)
2. Bottoni, P., Labella, A., Manca, V., Mitrana, V.: Superposition based on Watson-Crick-like complementarity. Theory of Computing Systems 39, 503–524 (2006)
3. Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Bidirectional sticker systems. In: Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E. (eds.) Third Annual Pacific Conf. on Biocomputing, Hawaii, pp. 535–546. World Scientific, Singapore (1998)
4. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
5. Kari, L., Păun, G., Rozenberg, G., Salomaa, A., Yu, S.: DNA computing, sticker systems, and universality. Acta Informatica 35(5), 401–420 (1998)
6. Kari, L., Rozenberg, G.: The many facets of Natural Computing. Communications of the ACM 51(10), 72–83 (2008)
7. Kobayashi, S., Mitrana, V., Păun, G., Rozenberg, G.: Formal properties of PA-matching. Theoretical Comput. Sci. 262(1-2), 117–131 (2001)
8. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New Computing Paradigms. Springer, Berlin (1998); Tokyo, 1999
9. Păun, G., Rozenberg, G.: Sticker systems. Theoret. Comput. Sci. 204, 183–203 (1998)
10. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 3. Springer, Heidelberg (1997)