# Circular contextual insertions/deletions with applications to biomolecular computation[1]

Mark Daley, Lila Kari
Department of Computer Science
University of Western Ontario
London, Ontario, N6A 5B7 Canada
{daley,lila}@csd.uwo.ca

Greg Gloor
Department of Biochemistry
University of Western Ontario
London, Ontario, N6A 5C1 Canada
ggloor@julian.uwo.ca

Rani Siromoney
Madras Christian College
Department of Computer Science
Madras, 600 059, India

## Abstract

*Insertions and deletions of small circular DNA strands into long linear DNA strands are phenomena that happen frequently in nature and thus constitute an attractive paradigm for biomolecular computing. This paper presents a new model for DNA-based computation that involves circular as well as linear molecules, and that uses the operations of insertion and deletion. After introducing the formal model we investigate its properties and prove in particular that the circular insertion/deletion systems are capable of universal computation. We also give the results of an experimental laboratory implementation of our model. This shows that rewriting systems of the circular insertion/deletion type are viable alternatives in DNA computation.*

## 1 Introduction

Early models of DNA recombination, the *splicing systems*, have already been defined by [4]. They aimed to describe the action of restriction enzymes and ligases on DNA molecules which resulted in cleavage and reassociation of DNA strands. Almost a decade later, [1] reported the first experiment that practically used DNA molecules for computation. The experiment consisted in finding a laboratory solution of an instance of the Directed Hamiltonian Path Problem solely by manipulation of DNA strands in test tubes.

Since then, research in DNA computing has embraced both theoretical studies of models of DNA computation and experiments giving DNA solutions to mathematical problems, [3], [5], [6], [8].

In particular, [9] proposed a model of DNA computation based on insertions and deletions that could be implemented in the laboratory by using a technique called PCR site-specific oligonucleotide mutagenesis.

Insertions and deletions of small circular strands of DNA into/from long linear strands happen frequently in all types of cells and constitute also one of the methods used by some viri to infect a host. We investigate here a generalization of insertions and deletions of words that aims to model these processes. (Note that circular DNA strings have been studied in the literature in the context of the splicing system model in [5], [11], [14], [15].)

Given a pair $(x, y)$ of words called a *context* and a pair of words $(\alpha, \beta)$ called a *guide*, the *circular contextual guided insertion* of the circular string $\bullet v$ into the linear string $u$ is performed as follows. First, the circular word $\bullet v$ is linearized by cutting it between $\alpha$ and $\beta$ (provided $\alpha\beta$ occurs as a subword in $v$) and reading it clockwise starting from $\alpha$ and ending at $\beta$. The resulting linear strand is then inserted into $u$, between $x$ and $y$. If $xy$ does not occur as a subword in $u$ no insertion can take place.

The $(x, y)$-deletion of $v$ from $u$ accomplishes the excision of the linear strand $v$ from $u$, provided $v$ occurs in $u$ flanked by $x$ on its left side and by $y$ on its right side.

We study closure properties of some well-known language families under various types of circular insertions. Moreover, we prove that every Turing machine can be simulated by a system based entirely on circular insertions and

[1]This research was partially supported by Academic Development Fund grant 97-115 of The University of Western Ontario and Grants R2824A01 and OGP0007877 of the Natural Sciences and Engineering Research Council of Canada.

linear deletions. Finally, we describe an experimental laboratory implementation of a rewriting system based in circular insertions and deletions. This proves the feasibility of performing computations in vitro by means of circular insertions and deletions.

Throughout this paper, $X$ represents an alphabet (a finite nonempty set), $\lambda$ represents the empty word (the word containing 0 letters), $\bullet v$ represents a circular string $v$ (a set containing every circular permutation of the linear string $v$). The length of a word $v$, denoted by $|v|$, is the number of occurrences of letters in $v$, counting repetitions. For a language $L$, by $\bullet L$ we denote the set of all words $\bullet v$ where $v \in L$. For further formal language definitions and notations the reader is referred to [12], [13].

## 2 Circular contextual guided insertions

Besides being fundamental in formal language theory, the operations of insertion and deletion have recently become of interest in connection with the topic of biomolecular computing. The insertion operation has been studied in [7] as a generalization of catenation. Given words $u$ and $v$, the *insertion* of $v$ into $u$ consists of all words that can be obtained by inserting $v$ in an arbitrary position of $u$:

$$u \longleftarrow v = \{u_1 v u_2 |\ u = u_1 u_2, u_1, u_2 \in X^*\}.$$

In the context of bimolecular computing, the insertion operation is too nondeterministic to model the type of insertions occurring in DNA strands. Consequently, a modified version, *contextual insertion*, was defined in [9] to capture the fact that insertions of DNA strands are context-sensitive. Given a pair of words $(x, y) \in X^*$, called a context, the $(x, y)$- contextual insertion of $v$ into $u$ is defined as

$$u \overset{(x,y)}{\longleftarrow} v = \{u_1 x v y u_2 |\ u_1, u_2 \in X^*, u = u_1 x y u_2\}.$$

Contextual insertion models insertion of linear DNA strands into linear DNA strands. To formalize insertion of circular DNA strands we define *circular contextual insertion* as follows. Given a pair $(x, y) \in X^* \times X^*$ called a *context*, the *circular* $(x, y)$*-contextual insertion* of $\bullet v$, $v \in X^*$ into $u \in X^*$ is defined as

$$u \overset{(x,y)}{\longleftarrow} \bullet v = \{u_1 x v' y u_2 | u = u_1 x y u_2, v' \in \bullet v, u_1, u_2 \in X^*\}.$$

In other words, the circular $(x, y)$-contextual insertion of $\bullet v$ into $u$ consists of inserting, between the subwords $x$ and $y$ of $u$, of each of the linear words representing a circular permutation of $v$. Note that if $u$ does not contain $xy$ as a subword then the result of the circular $(x, y)$-contextual insertion of any word into $u$ is the empty set. The cardinality of $u \overset{(x,y)}{\longleftarrow} \bullet v$ ranges thus from 0 to the number of occurrences of $xy$ in $u$ times $|v|$ (the number of circular permutations of $v$).

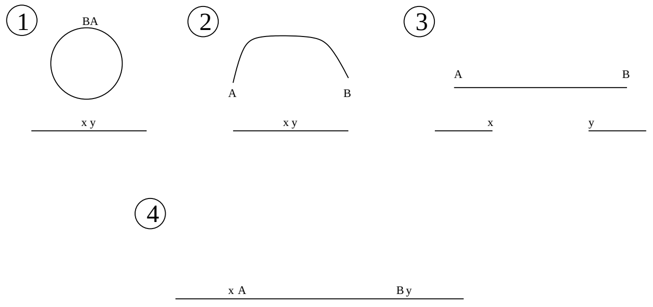

**Figure 1. Graphical representation of a circular insertion in the context $(x, y)$, where the circular string is cut at the site $(A, B)$.**

We can extend the circular contextual insertion to the situation where we are given apriori a list of contexts instead of a single one. Given a set $C \subseteq X^* \times X^*$ of contexts, we define the *circular C-contextual insertion of* $\bullet v$ *into* $u$ as:

$$u \overset{C}{\longleftarrow} \bullet v = \bigcup_{v' \in \bullet v} \{u_1 x v' y u_2 | (x, y) \in C, u = u_1 x y u_2, u_1, u_2 \in X^*\}.$$

Defining circular insertions this way allows inserting any of the circular permutations of the circular string. If we talk in terms of DNA strands, this entails being able to cut a circular string at any position, followed by its insertion into the linear strand. In practice, however, circular DNA strands can only be cut by restriction enzymes provided the recognition site of the enzyme is present. To make our operations closer to the reality of DNA strand bio-operations we modify our definition to only allow cutting of the circular strands at designated sites.

Formally, we introduce a *guide set* $G$, which is a set of locations at which circulars string may be cut and read clockwise to produce linear strands. By using this refinement, the circular insertion becomes:

$$u \overset{C,G}{\longleftarrow} \bullet v = \{u_1 x \alpha w \beta y u_2 | (x, y) \in C, (\alpha, \beta) \in G, u = u_1 x y u_2, \bullet v = \bullet \alpha w \beta, u_1, u_2, w \in X^*\}.$$

The existence of a guide set allows thus for more control over the insertion operation and makes the formal model a more accurate reflection of the biological reality of the insertion of plasmids (circular DNA strands) into linear DNA. A graphical example of this operation can be seen in **Fig. 1** .

A *circular insertion scheme* is a triple $I = (X, C, G)$ where $X$ is an alphabet, $C \subseteq X^* \times X^*$ is a context set and $G \subseteq X^* \times X^*$ is a guide set. (Both $C$ and $G$ are finite.)

**Definition 1** *Given two words $u, v \in X^*$, the circular contextual guided insertion of $\bullet v$ into $u$ according to the circular insertion scheme $I$ is defined as:*

$$u \overset{I}{\longleftarrow} \bullet v = \{u_1 x \alpha w \beta y u_2 | (x,y) \in C, (\alpha,\beta) \in G, u = u_1 x y u_2, \bullet v = \bullet \alpha w \beta, u_1, u_2, w \in X^*\}.$$

Informally, the insertion of $\bullet v$ into $u$ according to the insertion scheme $I = (X, C, G)$ means that the circular strand $\bullet v$ may be cut at a certain location specified by the guide set $G$; the linear strand obtained by reading the circular strand clockwise from the location of the cut may then be inserted into $u$ at a position specified by the context set $C$.

The definition can easily be generalized to languages. If $I = (X, C, G)$ is a circular insertion scheme and $L_1, L_2 \subseteq X^*$ are two languages, the circular insertion of $\bullet L_2$ into $L_1$, according to $I$, is defined as

$$L_1 \overset{I}{\longleftarrow} \bullet L_2 = \bigcup_{u \in L_1, v \in L_2} u \overset{I}{\longleftarrow} \bullet v$$

Note that if the context set $C$ equals $\{\lambda, \lambda\}$ then the insertion $u \overset{I}{\longleftarrow} \bullet v$ is in fact context-free, in the sense that insertion is allowed irrespective of any subwords occurring in the target word $u$. Similarly, if the guide set $G$ equals $\{\lambda, \lambda\}$ then cutting the circular word $\bullet v$ is restriction free, i.e., we can cut it at any point.

Depending on whether or not the insertion is context-free and/or the cutting is restriction-free, we can have four different types of circular insertion.

If $C = G = \{(\lambda, \lambda)\}$ then we call the context-free guide-free operation simply *circular insertion* and in this case $L_1 \overset{I}{\longleftarrow} \bullet L_2$ amounts to $L_1 \longleftarrow$ circ$(L_2)$, where circ$(L_2) = \{uv | vu \in L_2\}$ and $\longleftarrow$ is the insertion operation defined at the beginning of this section.

If $G = \{(\lambda, \lambda)\}$ then $L_1 \overset{I}{\longleftarrow} \bullet L_2$, amounts to the contextual insertion, [9], of circ$(L_2)$ into $L_1$ and is consequently called *circular contextual insertion*.

If $C = \{(\lambda, \lambda)\}$ then the operation becomes a *circular guided insertion* of $\bullet L_2$ into $L_1$.

Finally, if neither $C$ nor $G$ equal $\{\lambda, \lambda\}$, we obtain the *circular contextual guided insertion*.

## 3 Closure properties of families of languages under circular insertion

In this section we study closure properties of some well-known families of languages under various types of circular insertion. A language family $\mathcal{L}$ is said to be closed under a binary operation $\diamond$ if for any two languages $L_1, L_2 \in \mathcal{L}$, $L_1 \diamond L_2$ is also in $\mathcal{L}$. Let REG,CF,CS,RE denote the families of regular languages, context-free languages, context-sensitive languages and recursively enumerable languages respectively.

With the biological motivation in mind, the operation of *circular insertion* can be thought of as consisting of two distinct phases. In the first phase, a circular string is cut at some location to form a linear string. The second phase involves inserting the newly generated linear string into a specific location on a previously existing linear string.

To show that a given family of languages is closed under circular insertion, it suffices thus to show that it is closed under cyclic permutation (the first phase), and also under linear insertion (the second phase). As it is known that REG,CF,CS and RE are closed under cyclic permutation [2], and under ordinary insertion [7], it follows that they are closed also under circular insertion ($C = G = \{\lambda, \lambda\}$).

As REG, CF, CS, RE are all closed under contextual insertion [9] and circular permutation, it follows that they are closed under circular contextual insertion ($G = \{\lambda, \lambda\}$) as well.

To show the closure of the above families under circular guided insertion ($C = \{\lambda, \lambda\}$), we need an additional notion. Given $L \subseteq X^*$ and $G \subseteq X^* \times X^*$ we define the *circular closure of $L$ guided by $G$* as

$$\text{circ}_G(L) = \{\alpha w' \beta | \alpha w' \beta \in \bullet w, w \in L, (\alpha, \beta) \in G\}.$$

**Lemma 1** *For any $G$, circ$_G(L) \subseteq$ circ$(L)$.*

**Lemma 2** *circ$_G(L) =$ circ$(L) \cap (\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta)$.*

*Proof:* Suppose $v \in$ circ$_G(L)$. By Lemma 1, $v \in$ circ$(L)$. Furthermore, since $v$ is of the form $\alpha w' \beta$ for some $(\alpha, \beta) \in G$, it also belongs to $\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta$. Thus $v \in$ circ$(L) \cap (\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta)$ and circ$_G(L) \subseteq$ circ$(L) \cap (\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta)$.

Assume now that $v \in$ circ$(L) \cap (\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta)$. This implies that $v \in$ circ$(L)$ and moreover $v$ is of the form $\alpha X^* \beta$ for some $(\alpha, \beta) \in G$. Since $v \in$ circ$(L)$, there exists $w \in L$ such that $v \in \bullet w$. As $(\alpha, \beta) \in G$, $v$ now meets the definition of an element in circ$_G(L)$. Thus, circ$_G(L) \supseteq$ circ$(L) \cap (\bigcup_{(\alpha,\beta)\in G} \alpha X^* \beta)$. ♣

**Corollary 1** *REG,CF,CS,RE are closed under guided circular closure.*

*Proof:* It follows from Lemma 2 and the closure of REG,CF,CS and RE under circular closure and intersection with regular languages ♣

**Proposition 1** *REG,CF,CS,RE are closed under circular guided insertion ($C = \{(\lambda, \lambda)\}$).*

*Proof:* Let $L_1, L_2$ be two languages in one of the above families, and $I = (X, C, G)$ be an insertion scheme with $C = \{\lambda, \lambda\}$. The closure of the REG, CF, CS, RE under guided circular closure follows from the equality

$$L_1 \overset{I}{\longleftarrow} \bullet L_2 = L_1 \longleftarrow \text{circ}_G(L_2),$$

Corollary 1, and the closure of REG,CF,CS and RE under (linear) insertion [7]. ♣

**Proposition 2** *REG,CF,CS,RE are closed under circular guided contextual insertion.*

*Proof:* Let $L_1, L_2$ be two languages in REG (respectively CF,CS, RE) and let $I = (X, C, G)$ be a circular insertion scheme where $C, G \subseteq X^* \times X^*$ are the context respectively the guide set. For each pair $(x, y) \in C$ denote

$$L_{x,y} = (X^* x \# y X^*) \cap (L_1 \amalg \{\#\}),$$

where $\#$ is a new symbol not belonging to $X$ and $\amalg$ denotes the shuffle operation defined as

$$u \amalg v = u_1 v_1 u_2 v_2 ... u_n v_n, u = u_1 u_2 ... u_n,$$

$$v = v_1 v_2 ... v_n,\ u_i, v_i \in X^*,\ 1 \leq i \leq n, n \geq 1.$$

Note now that

$$\bigcup_{(x,y)\in C} L_{x,y} = \{u_1 x \# y u_2 | u = u_1 x y u_2 \in L_1,$$

$$(x, y) \in C, u_1, u_2 \in X^*\}.$$

If we consider the substitution defined by

$$s(\#) = \text{circ}_G(L_2)/\{\lambda\}$$

$$s(a) = a \text{ for } a \in X, a \neq \#$$

then we have

$$L_1 \overset{I}{\longleftarrow} \bullet L_2 = s(L_2) \text{ if } \lambda \notin L_2,$$

$$L_1 \overset{I}{\longleftarrow} \bullet L_2 = s(L_2) \cup [L_1 \cap (\bigcup_{(x,y)\in C} X^* x y X^*)]$$

$$\text{if } \lambda \in L_2.$$

The theorem now follows as REG,CF,CS and RE are closed under shuffle, guided circular closure, $\lambda$-free substitution, intersection with regular languages and union. ♣.

Note that the closure properties of the families in the Chomsky hierarchy under circular insertion, contextual circular insertion and guided circular insertion follow now from Proposition 2. However, separate justifications were given for each type of circular insertion as an illustration of the proof techniques that can be used to prove other closure properties.

## 4 Computing with circular insertions and (linear) deletions

In the preceding section we have studied properties of guided contextual circular insertions. As rewriting mechanisms, insertion-type rules alone cannot achieve the computational power of Turing machines: some deletion-type operations are needed. We will combine circular insertions with linear deletions to obtain the necessary kit of rewriting rules that can achieve Turing machine computational power.

In the style of [9], we define a *circular insertion/deletion system* as a tuple

$$ID^\bullet = (X, T, I^\bullet, D, A)$$

where $X$ is an alphabet, card$(X) \geq 2$, $T \subseteq X$ is the terminal alphabet, $I^\bullet \subseteq (X^*)^5$ is the finite set of circular insertion rules, $D \subseteq (X^*)^3$ is the finite set of deletion rules, and $A \subseteq X^+$ is a linear strand called the *axiom*.

A circular insertion rule in $I^\bullet$ is written as $(c_1, g_1, \bullet x, g_2, c_2)_I$ where $(c_1, c_2)$ represents the context of the insertion, $\bullet x$ is the string to be inserted and $(g_1, g_2)$ are the guides, i.e. the location where $\bullet x$ is cut.

A deletion rule in $D$ is written as $(c_1, x, c_2)_D$ where $(c_1, c_2)$ represents the context of deletion and $x$ is the string to be deleted.

If $u, v \in X^*$, we say that $u$ derives $v$ according to ID$^\bullet$ and we write $u \Rightarrow v$ if $v$ is obtained from $u$ by either a guided contextual circular insertion or by a linear contextual deletion, i.e.,

– either $u = \alpha c_1 c_2 \beta$, $v = \alpha c_1 g_1 x' g_2 c_2 \beta$ and $I^\bullet$ contains the circular insertion rule $(c_1, g_1, \bullet x, g_2, c_2)_I$ where $g_1 x' g_2 \in \bullet x$, or

– $u = \alpha c_1 x c_2 \beta$, $v = \alpha c_1 c_2 \beta$ and $D$ contains the linear deletion rule $(c_1, x, c_2)_D$.

A sequence of direct derivations

$$u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k, k \geq 0$$

is denoted by $u_1 \Rightarrow^* u_k$ and $u_k$ is said to be derived from $u_1$.

The language $L(ID^\bullet)$ accepted by the circular insertion/deletion system ID$^\bullet$ is defined as

$$L(ID^\bullet) = \{v \in T^* |\ v \Rightarrow^* A, A \text{ is the axiom }\}$$

Recall that, [13], a rewriting system $(S, X \cup \{\#\}, F)$ is called a *Turing machine* iff the following conditions are satisfied.

(i) $S$ and $X \cup \{\#\}$ (with $\# \notin X$ and $X \neq \emptyset$) are two disjoint alphabets referred to as the *state* and *tape* alphabet.

(ii) Elements $s_0 \in S$, $\flat \in X$, and a subset $S_f \subseteq S$ are specified, namely, the *initial state*, the *blank symbol*, and the *final state set*. A subset $V_f \subseteq X$ is specified as the *final alphabet*.

(iii) The productions in $F$ are of the forms

(1) $s_i a \rightarrow s_j b$ overprint
(2) $s_i a c \rightarrow a s_j c$ move right
(3) $s_i a \# \rightarrow a s_j \flat \#$ move right and extend workspace
(4) $c s_i a \rightarrow s_j c a$ move left
(5) $\# s_i a \rightarrow \# s_j \flat a$ move left and extend workspace

where $s_i, s_j \in S$ and $a, b, c \in X$. Furthermore, for each $s_i, s_j \in S$ and $a \in X$, $F$ either contains no productions (2) and (3) (resp. (4) and (5)) or else contains both (2) and (3) (respectively (4), (5)) for every $c \in X$. For no $s_i \in S$ and $a \in X$, the word $s_i a$ is a subword of the left side in two productions of the forms (1), (3) and (5).

We say that a word $sw$, where $s \in S$ and $w \in (X \cup \{\#\})^*$ is *final* iff $w$ does not begin with a letter $a$ such that $sa$ is a subword of the left side of some production in $F$. The language *accepted* by a Turing machine TM is defined by

$$L(TM) = \{w \in V_f^* | \ \#s_0 w\# \ \Rightarrow^* \ \#w_1 s_f w_2\# \text{ for some}$$
$$s_f \in S_f, w_1, w_2 \in X^* \text{ such that } s_f w_2\# \text{ is final}\}$$

where $\Rightarrow$ denotes derivation according to the rewriting rules (1) – (5) of the Turing machine. A language is *acceptable* by a Turing machine iff $L = L(TM)$ for some TM. It is to be noted that TM is *deterministic*: at each step of the rewriting process, at most one production is applicable.

**Proposition 3** *If a language is acceptable by a Turing machine TM, then there exists a circular insertion/deletion system ID$^\bullet$ accepting the same language.*

*Proof.* Let TM be a Turing machine $TM = (S, X \cup \{\#\}, F)$ as described above. We will construct a circular insertion/deletion (shortly insdel) system $ID^\bullet = (N, T, I^\bullet, D, Y_0)$ such that the language accepted by the insdel system is $L(ID^\bullet) = L(TM)$. The alphabet of ID$^\bullet$ is $N = S \cup X \cup \{\#\} \cup \{O, L, R, Y_0, Y_1, Y_2\}$, where $O, L, R, Y_0, Y_1, Y_2$ are new symbols not appearing in $S \cup X$. The terminal alphabet is $T = V_f$, the axiom is $Y_0$, and the guided contextual insertion and contextual linear deletion rules are defined as follows.

**(a)** For each rule of the Turing machine $TM$, circular insertion and deletion rules are added to the circular insdel system in the following fashion, where $a, b, c$ are letters in $X$, $x, y \in X \cup X^2 \cup X^3 \cup \{\#\}$, $z \in X^2 \cup \{\#\}X$, and $r, t \in X^*$:

**a1.** For each *rule (1)* $s_i a \ \rightarrow \ s_j b$ (overprint) of $F$, we add to the insdel system the rules $(xs_i a, s_j, \bullet s_j Ob, b, y)_I$, $(x, s_i a, s_j Oby)_D$ and $(zs_j, O, by)_D$.

Hence, if $u = \#rxs_i ayt\#$, then rule *(1)* of TM can be simulated by the following derivation in ID$^\bullet$:

$$\#rxs_i ayt\# \ \Rightarrow \ \#rxs_i as_j Obyt\# \ \Rightarrow$$
$$\#rxs_j Obyt\# \ \Rightarrow \ \#rxs_j byt\#$$

**a2.** For each *rule (2)* $s_i ac \ \rightarrow \ as_j c$ (move right) of $F$, we add to ID$^\bullet$ the rules $(xs_i a, s_j, \bullet s_j R, R, cy)_I$, $(x, s_i, as_j Rcy)_D$ and $(zas_j, R, cy)_D$.

Hence, if $u = \#rxs_i acyt\#$, then rule *(2)* of TM can be simulated by the following derivation in ID$^\bullet$:

$$\#rxs_i acyt\# \ \Rightarrow \ \#rxs_i as_j Rcyt\# \ \Rightarrow$$
$$\#rxas_j Rcyt\# \ \Rightarrow \ \#rxas_j cyt\#.$$

**a3.** For each *rule (3)* $s_i a\# \ \rightarrow \ as_j \flat\#$ (move right and extend workspace) of $F$, we add to ID$^\bullet$ the rules $(xs_i a, s_j, \bullet s_j R\flat, \flat, \#)_I$, $(x, s_i, as_j R\flat\#)_D$, and $(xas_j, R, \flat\#)_D$.

If $u = \#rxs_i a\#$, then rule *(3)* of TM can be simulated by the following derivation in ID$^\bullet$:

$$\#rxs_i a\# \ \Rightarrow \ \#rxs_i as_j Rflat\ \# \ \Rightarrow$$
$$\#rxas_j R\flat\# \ \Rightarrow \ \#rxas_j \flat\#.$$

**a4.** For each *rule (4)* $cs_i a \ \rightarrow \ s_j ca$ (move left) of $F$, we add to ID$^\bullet$ the rules $(x, s_j, \bullet s_j L, L, cs_i ay)_I$, $(xs_j Lc, s_i, ay)_D$, $(xs_j, L, cay)_D$.

Hence, if $u = \#rxcs_i ayt\#$, then rule *(4)* of TM can be simulated by the following derivation in ID$^\bullet$:

$$\#rxcs_i ayt\# \ \Rightarrow \ \#rxs_j Lcs_i ayt\# \ \Rightarrow$$
$$\#rxs_j Lcayt\# \ \Rightarrow \ \#rxs_j cayt\#.$$

**a5.** For each *rule (5)* $\#s_i a \ \rightarrow \ \#s_j \flat a$ (move left and extend workspace) of $F$ we add to ID$^\bullet$ the rules $(\#, s_j, \bullet s_j L\flat, \flat, s_i ay)_I$, $(\#s_j L\flat, s_i, ay)_D$, $(\#s_j, L, \flat ay)_D$.

Hence, if $u = \#s_i ayt\#$ then rule *(5)* of TM can be simulated by the following derivation in ID$^\bullet$:

$$\#s_i ayt\# \ \Rightarrow \ \#s_j L\flat s_i ayt\# \ \Rightarrow$$
$$\#s_j L\flat ayt\# \ \Rightarrow \ \#s_j \flat ayt\#.$$

In addition to the rules above, that simulate the rewriting of the Turing machine by circular insertions and deletion rules, we introduce the following rules **(b)**:

$(b1)$ $(\lambda, \#, \bullet\#s_0, s_0, b)_I, (b, \#, \bullet\#, \#, \lambda)_I$
$(b2)$ $(s_f, Y_1, \bullet Y_1, Y_1, a)_I, (s_f, Y_1, \bullet Y_1, Y_1, \#)_I$
$(b3)$ $(c, s_f, Y_1)_D, (\#, s_f, Y_1)_D$
$(b4)$ $(Y_1, b, c)_D, (Y_1, b, \#)_D$
$(b5)$ $(b, Y_2, \bullet Y_2, Y_2, Y_1\#)_I, (\#, Y_2, \bullet Y_2, Y_2, Y_1\#)_I$
$(b6)$ $(Y_2, Y_1, \#)_D$
$(b7)$ $(b, c, Y_2)_D, (\#, b, Y_2)_D$
$(b8)$ $(\#, Y_0, \bullet Y_0, Y_0, Y_2\#)_I$
$(b9)$ $(\lambda, \#, Y_0)_D, (Y_0, Y_2\#, \lambda)_D$
$(b10)$ $(\lambda, \#, \bullet\#s_0\#, \#, \lambda)_I, (\#s_f, Y_2, \bullet Y_2, Y_2, \#)_I$, $(\#, s_f, Y_2\#)_D$

where $s_f$ ranges over $S_f$, $b, c$ range over $X$, and for each $s_f$, $a$ ranges over such elements of $X$ that $s_f a$ is final. It can now be verified that $L(ID^\bullet) = L(TM)$.

Indeed, if $w \in L(TM)$ then $w \in T^*$ and there exists a derivation

$$\#s_0 w\# \ \Rightarrow^* \ \#w_1 s_f w_2\# \qquad (*)$$

for some $s_f \in S_f$, $w_1, w_2 \in X^*$, $s_f w_2\#$ final. To show that $w \in L_a(ID^\bullet)$ we must find a derivation $w \Rightarrow^* Y_0$ according to the rules of ID$^\bullet$. If $w \neq \lambda$, such a derivation is the following:

$$w \overset{(b1)}{\Rightarrow} \#s_0w\# \overset{(a)^*}{\Rightarrow} \#w_1s_fw_2\# \overset{(b2)}{\Rightarrow} \#w_1s_fY_1w_2\# \overset{(b3)}{\Rightarrow}$$

$$\#w_1Y_1w_2\# \overset{(b4)}{\Rightarrow} \#w_1Y_1\# \overset{(b5)}{\Rightarrow} \#w_1Y_2Y_1\# \overset{(b6)}{\Rightarrow}$$

$$\#w_1Y_2\# \overset{(b7)}{\Rightarrow} \#Y_2\# \overset{(b8)}{\Rightarrow} \#Y_0Y_2\# \overset{(b9)}{\Rightarrow} Y_0,$$

where $\overset{(a)^*}{\Rightarrow}$ represents a simulation of the derivation (*) of the Turing machine by the rules **(a)**.

If $w = \lambda$, the required derivation is the following:

$$\lambda \overset{(b10)}{\Rightarrow} \#s_0\# \overset{(b10)}{\Rightarrow} \#Y_2\# \overset{(b8)}{\Rightarrow} \#Y_0Y_2\# \overset{(b9)}{\Rightarrow} Y_0.$$

Assume, conversely that $w \in L(ID^\bullet)$. If $w = \lambda$ there is a derivation according to ID$^\bullet$ from $\#s_f\#$ to $Y_0$ where $s_f \in S_f$ and $s_f = s_0$. This implies that $\lambda \in L(TM)$. If $w \neq \lambda$ then, according to the way the rules **(a)** were constructed, there is a derivation according to ID$^\bullet$ from

$$\#w_1s_faw_2'\#, s_f \in S_f, a \in X, w_1, w_2' \in X^*, s_fa \text{ final}, (**)$$

to $Y_0$, and also a derivation from $w$ to $(**)$, according to rules (b). This implies that $w \in L(TM)$. ♣

## 5 Experimental implementation of a circular insertion/deletion system

In order to be able to describe our molecular biology laboratory implementation of a circular insertion/deletion system we need a brief introduction of some basic molecular biology notions. For further details of molecular biology terminology, the reader is referred to [10].

DNA (deoxyribonucleic acid) is found in every cellular organism as the storage medium for genetic information. It is composed of units called nucleotides, distinguished by the chemical group, or base, attached to them. The four bases are *adenine, guanine, cytosine* and *thymine*, abbreviated as $\mathcal{A}$, $\mathcal{G}$, $\mathcal{C}$, and $\mathcal{T}$. (The names of the bases are also commonly used to refer to the nucleotides that contain them.) Single nucleotides are linked together end–to–end to form DNA strands. A short single-stranded polynucleotide chain, usually less than 30 nucleotides long, is called an *oligonucleotide* (or, shortly, oligo). The DNA sequence has a *polarity*: a sequence of DNA is distinct from its reverse. The two distinct ends of a DNA sequence are known under the name of the $5'$ end and the $3'$ end, respectively. Taken as pairs, the nucleotides $\mathcal{A}$ and $\mathcal{T}$ and the nucleotides $\mathcal{C}$ and $\mathcal{G}$ are said to be *complementary*. Two complementary single-stranded DNA sequences with opposite polarity will join together to form a double helix in a process called *base-pairing* or *annealing*. The reverse process – a double helix coming apart to yield its two constituent single strands – is called *melting*.

A single strand of DNA can be likened to a string consisting of a combination of four different symbols, $\mathcal{A}$, $\mathcal{G}$, $\mathcal{C}$, $\mathcal{T}$. Mathematically, this means we have at our disposal a 4-letter alphabet $X = \{\mathcal{A}, \mathcal{G}, \mathcal{C}, \mathcal{T}\}$ to encode information. As concerning the operations that can be performed on DNA strands, the existing models of DNA computation are based on various combinations of the following primitive *bio-operations*, [8]:

– *Synthesizing* a desired polynomial-length strand.
– *Mixing:* pour the contents of two test-tubes into a third.
– *Annealing (hybridization):* bond together two single-stranded complementary DNA sequences by cooling the solution.
– *Melting (denaturation):* break apart a double-stranded DNA into its single-stranded components by heating the solution.
– *Amplifying (copying):* make copies of DNA strands by using the Polymerase Chain Reaction (PCR).
– *Separating* the strands by size using a technique called gel electrophoresis.
– *Extracting* those strands that contain a given pattern as a substring by using affinity purification.
– *Cutting* DNA double-strands at specific sites by using commercially available restriction enzymes.
– *Ligating:* paste DNA strands with compatible sticky ends by using DNA ligases.
– *Substituting:* substitute, insert or delete DNA sequences by using PCR site-specific oligonucleotide mutagenesis.
– *Detecting and Reading* a DNA sequence from a solution.

To test the empirical validity of our theoretical model, we implemented a small circular insertion/deletion system in the laboratory. The purpose of this implementation was to show that in vitro circular insertion is possible and not overwhelmingly difficult.

The following circular insertion/deletion system was chosen:

$$ID^\bullet = (X, T, I^\bullet, D, u)$$

where the alphabets are $T, X = \{\mathcal{A}, \mathcal{C}, \mathcal{G}, \mathcal{T}\}$, there are no deletion rules, i.e. $D = \emptyset$, the axiom $u$ is a small DNA segment from the *Drosophila Melanogaster* genome and $I^\bullet = (\mathcal{G}, \mathcal{G}, \bullet v, \mathcal{TCGAC}, \mathcal{TCGAC})$ where $\bullet v$ is a commercially available plasmid (circular strand). Note that $\mathcal{A}, \mathcal{C}, \mathcal{G}, \mathcal{T}$ correspond to the four bases that occur in natural DNA, and that the sequence $\mathcal{G}|\mathcal{TCGAC}$ is the restriction (cut) site for the *Sal I* enzyme.

To begin the experiment, we synthesized the linear axiom $u$ in which we would then insert. This was accomplished by taking DNA from *Drosophila* (fruit fly) and performing PCR with the primers $BC^+$ and $cd^-$. The result was the amplification of a particular 682bp (basepair) linear sequence of DNA which became the axiom $u$ of the circular insertion/deletion system. The 682bp linear strand was chosen to contain exactly one restriction site for the enzyme *Sal I*, corresponding to the context of insertion $(\mathcal{G}, \mathcal{TCGAC})$.

For the circular string $\bullet v$ to be inserted we chose pK18h, a commercially available plasmid having one restriction site for *Sal I*, corresponding to the guides $(\mathcal{G}, \mathcal{TCGAC})$ in the insertion rule.

After verifying that the PCR had worked correctly and we had indeed obtained the desired 682bp linear axiom $u$, we cut $u$ with *Sal I*, cleaving it into two new linear strands denoted by $L$ and $R$, i.e. $u = L|R$. The product was checked by gel electrophoresis to ensure the presence of bands corresponding to the sizes of $L$ (188bp) and $R$ (493bp), as seen from the first band in the gel of **Fig. 2**. The plasmid $\bullet v$ was also cut and linearized in the same fashion resulting in the linear strand $v$.

At this point the linear strands $L$ and $R$ were combined with the linearized pK18h, i.e. $v$, and ligase was added to reconnect the strands of DNA. After allowing time for ligation, a gel was run to determine the products. The second band from the gel shown in **Fig. 2** indicates that in addition to the desired $L|$plasmid$|R$, we also obtain $R|R$, $L|R$, plasmid$|$plasmid and even plasmid$|$plasmid$|$plasmid.

Note that the band corresponding to the approximate size of $L|$plasmid$|R$ can be seen as a smear. This could suggest the presence of $R|$plasmid$|R$ or of any other combination of two linear fragments and a plasmid which failed to separate clearly from one another due to the large size. Thus further analysis was required to ensure the presence of the desired product $L|$plasmid$|R$.

In order to amplify the amount of DNA available at this point, the DNA was recircularized and introduced into *E. Coli* bacteria. (The complex details of this process are omitted here.)

Prior to sequencing, a quick restriction digest[2] was performed on small amounts of product isolated from each of the several bacterial colonies. If the starting sample were a heterogeneous mixture of DNA molecules, each colony would yield a different product. Consequently, the restriction digest of DNA samples (each isolated from a particular colony) with enzymes *Sal I*, *Stu I* and *Xba* , resulted in bands indicating different size distributions. Of these, one sample corresponded to the size of $L|$plasmid$|R$ and the identity of the product was confirmed by sequencing.

This experiment demonstrates that it is possible to insert a plasmid into a linear strand in vitro, implementing thus a circular insertion/deletion system. Future experimental work would ideally include a much larger system to test the scalability of this approach.

[2]A restriction digest consists of taking an unknown product, cutting it with a series of restriction endonucleases and running each product on a gel. If the restriction sites for the desired product are known, then we can compute the expected sizes of the molecules formed after the digest. If the expected sizes match the bands found on the gel, then there is a good chance the unknown product is the desired product. By repeating this operation with different enzymes, we can increase our confidence in this result.

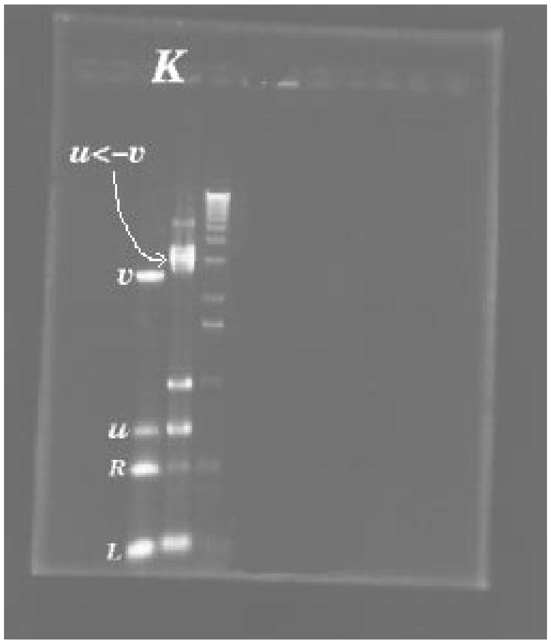

**Figure 2. The first vertical lane of this gel consists of bands corresponding to the unreacted linearized plasmid $v$, the linear strand $u$, and the two fragments of the cut linear strand ($R$, respectively $L$). The second vertical lane shows a band corresponding to the product obtained after reaction ($u \leftarrow v$). The third lane contains a standard 1kb (kilobase) ladder used to measure the others.**

## 6 Conclusions and future work

The circular insertion/deletion systems provide a theoretically interesting extension to the linear systems originally described in [9]. Moreover, since the operation of inserting plasmids into linear strands of DNA is a common occurrence in biology, the study of such systems is a natural next step in the development of an insertion/deletion-based model of cellular genetic manipulation.

From a computer science and language theory standpoint, the construction given here is capable of universal computation and can thus theoretically implement any algorithm that can be run on an electronic computer. One interesting possibility for future work would be an analysis of what type of algorithms are most easily and efficiently implemented with such a system. Such a study would yield some insight into what types of “algorithms” are being used by cells themselves.

The experimental results we have obtained show that rewriting systems based on circular insertions and deletions are viable in vitro alternatives in DNA computation. They are momentarily limited due to the fact that only a small size problem could be implemented. Future work on this model will include a larger and more complex experiment to help identify the sources and severity of biochemical errors.

This paper has presented a new model for DNA based computing using circular strands by extending the insertion/deletion systems presented in [9]. The model has been shown to be capable of universal computation and a small size problem was actually implemented in DNA.

We believe that continued effort to provide formal models of actual biological processes will yield not only insight into biological systems but perhaps also provide new ideas about the nature of computation itself.

## 7 Acknowledgments

We thank Gabriel Thierrin for discussion, Jürgen Dassow for pointing out two critical papers, Gordon Alford for comments and Esther Males for assistance and a great deal of patience in the lab.

## References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] A. Brandstadt. Closure properties of certain families of formal languages with respect to a generalization of cyclic closure. *R.A.I.R.O. Theoretical Informatics*, 15(3):233–252, 1981.

[3] M. Daley and M. Eramian. Formal models of DNA computation. *Manuscript*, 1998.

[4] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49, 1987.

[5] T. Head. Splicing schemes and DNA. *Lindenmayer systems*, pages 371–383, 1991.

[6] T. Head, G. Paun, and D. Pixton. Language theory and molecular genetics. *Handbook of Formal Languages*, 2:295–358, 1997.

[7] L. Kari. *On insertions and deletions in formal languages*. PhD thesis, University of Turku, Finland, 1991.

[8] L. Kari. DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer*, 19:9–22, 1997.

[9] L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, 131:47–61, 1996.

[10] J. Kendrew et al., editor. *The Encyclopedia of Molecular Biology*. Blackwell Science, Oxford, 1994.

[11] D. Pixton. Linear and circular splicing systems. *Proceedings of the First International Symposium on Intelligence in Neural and Biological Systems*, pages 181–188, 1995.

[12] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer, Berlin, 1997.

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

[14] R. Siromoney, K.G. Subramanian, and Dare Rajkumar. Circular DNA and splicing systems. *Parallel Image Analysis. Lecture Notes in Computer Science 654*, pages 260–273, 1992.

[15] T. Yokomori, S. Kobayashi, and C. Ferretti. Circular splicing systems and DNA computability. *Proc. of IEEE International Conference on Evolutionary Computation'97*, pages 219–224, 1997.