ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **10**, Number 4, 2007, 315–322

Ciliate Operations without Context in a Membrane Computing Framework

A. ALHAZOV

Department of Information Technologies, Åbo Akademi University
Turku Center for Computer Science, FIN-20520 Turku, Finland
aalhazov@abo.fi

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
aartiom@math.md

Abstract. We study the computational power of string processing systems with excision and insertion rules with communication. The strings are distributed in different regions, and the rules are defined by cutting out a substring flanked by specific repeated symbols and a reverse operation; the rule only specifies the repeated symbol and the regions of reactants and products. It turns out that they can generate all recursively enumerable sets of non-negative integers.

1. Introduction

Ciliates are a group of unicellular eucaryotes with a unique feature: the gene assembly. During the sexual reproduction the genetic information in ciliates is being rearranged: blocks of DNA surrounded by certain repeated sequences called pointers are being reordered, and some of them are inverted. A few pointer-based mathematical models have been introduced to explain this process, such as the intermolecular model, see [6], [7], and the intramolecular model, see [2], [8].

Hence, we can say that the gene assembly in ciliates inspired a number of theoretical string-processing operations. Ciliate operations are string processing rules based on recombination at specific repeated substrings.

To introduce the operations in both models we need a few notions. Let V be a finite alphabet, then the set of words (also called strings) over V is denoted by

V^*. The circular string associated to a string $x \in V^*$ is denoted by $\bullet x$; two circular strings are considered identical if and only if they are associated to xy and yx for some $x, y \in V^*$. Consider a fixed bijective complement function $c : V \longrightarrow V$ such that $c(c(a)) = a$ for all $a \in V$. Then the complement $c(a_n) \cdots c(a_2)c(a_1)$ of a mirror image of a word $x = a_1 a_2 \cdots a_n$ is denoted by $\overline{x}$.

Essentially the intramolecular model has rules exc, ins:

$$upv, \bullet pw \Rightarrow^{\mathsf{ins}_p} upvpw, \quad (1)$$
$$\bullet upv, \bullet pw \Rightarrow^{\mathsf{ins}_p} \bullet upvpw, \quad (2)$$
$$upvpw \Rightarrow^{\mathsf{exc}_p} upv, \bullet pw, \quad (3)$$
$$\bullet upvpw \Rightarrow^{\mathsf{exc}_p} \bullet upv, \bullet pw \quad (4)$$
for strings $u, v, w \in V^*$,

and the intermolecular model has rules ld, hi, dlad:

$$uppw \Rightarrow^{\mathsf{ld}_p} uw, \quad (5)$$
$$pvp \Rightarrow^{\mathsf{ld}_p} \bullet v, \quad (6)$$
$$upv\overline{p}w \Rightarrow^{\mathsf{hi}_p} u\overline{v}w, \quad (7)$$
$$upvqwpxqy \Rightarrow^{\mathsf{dlad}_{p,q}} uxwvy \quad (8)$$
for strings $u, v, w, x, y \in V^*$.

Notice some "conservation laws" in the rules above: all parts of the result appear in the reactants. In case one considers the first model in the computational framework, one might assume the presence of an infinite supply of certain strings as resources for computation. In the latter model, there is only one string considered, and the computation is limited to reordering, inverting and deleting its parts, so when one speaks about its computational power, some variant of increasing the workspace is assumed.

Some universality results have been obtained for intermolecular model, see [7] and for intramolecular model, see [5] and [4]. All these results rely on contexts: whether some rule is applicable to the corresponding pointers depends on the contexts surrounding them. We now eliminate the control by contexts by placing strings in different regions and letting rules control their movement between the regions. It turns out that such systems are still very powerful.

To describe such devices we use the framework of P systems. P systems are parallel distributed computational devices of biochemical inspiration, introduced by Gh. Păun, see [10] for a systematic overview and [12] for comprehensive bibliography. A system consists of a graph, objects placed in its nodes and processed in parallel by the rules. The basic framework specifies neither the nature of the objects, nor the kind of rules used to process these objects. Throughout this paper we speak about (linear and circular) strings and intermolecular ciliate operations.

The closest systems considered in the literature are splicing P systems, introduced by Gh. Păun and T. Yokomori in [11], see also [10]. Although a ciliate operation may be viewed as two synchronized splicing operations, splicing seems to provide much more control.

The structure of the paper is the following. In Section 2 we recall the tools we use and give a formal definition of P systems with ciliate operations, and in Section 3 we show the power of these systems. We conclude in Section 4 by a discussion on possible variants of the model and open questions.

2. Preliminaries

2.1. Basics

We denote the set of circular strings over O by $O^\bullet$.

The class of all recursively enumerable languages is denoted by RE. The class of all recursively enumerable sets of nonnegative integers is denoted by NRE.

2.2. Register machines

A (non-deterministic) register machine is a tuple $M = (n, Q, q_0, q_f, I)$ where

- n is the number of registers;
- I is a set of instructions bijectively labeled by elements of Q, of the form $(q : io, q', q'')$, where $1 \leq i \leq n$, $o \in \{+, -\}$, and $q, q', q'' \in Q$;
- $q_0 \in Q$ is the initial label;
- $q_f \in Q$ is the final label.

The allowed instructions are:

- $(q : i+, q', q'')$ - add one to the contents of register i and proceed to instruction q' or q'';
- $(q : i-, q', q'')$ - jump to instruction q'' if the contents of register i is zero, otherwise subtract one from it and proceed to instruction q';
- $(q_f : halt)$ - finish the computation; this is a unique instruction with label q_f.

If a register machine starts from instruction q_0 with all registers containing zero and arrives to the instruction q_f with the first register containing n_1, we say that it generates the number n_1. Any recursively enumerable set of nonnegative integers can be generated by a register machine; three registers suffice.

2.3. Operations and P systems

Our systems have multiple regions (cells) and a population of (linear and circular) strings associated to each region. The operations are defined by specifying the regions of each reactant, the regions of each product, and the pointer defining the place of excision or insertion.

An excision operation $i \rightarrow_p j/k$ is applicable to a string $upvpw$ (or $\bullet upvpw$) in region i; it yields strings upw (or $\bullet upw$, respectively) in region j and $\bullet pv$ in region k.

An insertion operation $j/k \rightarrow_p i$ is applicable to a pair of strings upw (or $\bullet upw$) in region j and $\bullet pv$ in region k; it yields a string $upvpw$ (or $\bullet upvpw$, respectively) in region i.

Definition 2.1. *A (tissue) P system with ciliate operations* is a tuple

$$\Pi = (O, C, R, i_0), \text{ where}$$

- O is a finite set of symbols;
- C is a finite set of cells; each cell $c \in C$ we associate a finite set $\mathsf{inf}(c)$ of strings from $O^* \cup O^\bullet$ present in infinite supply, and a finite multiset $\mathsf{cfg}(c)$ of strings from $O^* \cup O^\bullet$, representing its initial contents;
- R is a finite set of excision and insertion rules;
- $i_0 \in C$ is the output cell.

Remark 2.1. We additionally require that R cannot contain rule $i \rightarrow_p j/k$ if $\inf(i)$ contains some string of the form $upvpw$, $u, v \in O^*$. We also requite that R cannot contain rule $i/j \rightarrow_p k$ if both $\inf(i)$ contains some string of the form upv, $u, v \in O^*$ and $\inf(j)$ contains some string of the form xpy, $x, y \in O^*$.

This requirement is needed to make sure that only finite multisets of rules are applicable to any configuration.

The rules are applied non-deterministically, in a maximally parallel way (the set of applications of rules cannot be extended). The computation halts when no rules are applicable. The result of a halting computation is a set of strings (or lengths of strings) in cell i_0. The set generated by Π is the union of the results over all halting computations; we denote it by $L(\Pi)$ (or $N(\Pi)$, respectively). If instead of maximally parallel application of rules we consider asynchronous way (any number of rules can be applied), then we add superscript asyn to L or N in the notation.

We denote the set of numbers (N) generated by asynchronous (asyn) tissue P systems with at most m membranes (tP_m) with linear and circular strings ($S_\bullet$) and ciliate operations with pointers of length at most k ($\mathsf{exc}_k, \mathsf{ins}_k$) and targets ($\mathsf{tar}$) by $N^{\mathsf{asyn}} S_\bullet tP_m(\mathsf{exc}_k, \mathsf{ins}_k, \mathsf{tar})$.

We replace m by $*$ if we do not restrict it. We replace N by L if we consider generation of languages. We remove asyn if we consider maximally parallel systems. We replace tar by tar_2 if at most two different regions are present in specifications of rules, or by ntar if non-distributed systems are considered.

Remark 2.2. Since a rule involves three regions, one might argue that instead of tissue it is more appropriate to call the underlying structure a network of cells, see also [3].

3. Results

Theorem 3.1. $N^{\mathsf{asyn}}S_{\bullet}tP_{*}(\mathsf{exc}_1, \mathsf{ins}_1, \mathsf{tar}) = NRE$.

Proof. Let $L \in NRE$. Then there exists a register machine $M = (3, Q, q_0, q_f, I)$ generating $\{n \geq 0 \mid n+11 \in L\}$ in the first register, halting with other registers being empty. Consider the following P system:

$$\begin{aligned}
\Pi &= (O, C, R, q_f), \text{ where} \\
O &= \{\#\} \cup \{A_i, a_i \mid 1 \leq i \leq 3\}, \\
C &= \{q, z_q \mid q \in Q\} \cup \{s_{a_i}, s_{A_i} \mid 1 \leq i \leq 3\} \cup \{test, fail\}, \\
\mathsf{cfg}(q_0) &= \{\#A_1a_1A_1A_2a_2A_2A_3a_3A_3\#\}, \\
\mathsf{cfg}(q_f) &= \{a_1^n \mid n < 11, n \in L\}, \\
\mathsf{cfg}(c) &= \emptyset,\ c \notin \{q_0, q_f\}, \\
\mathsf{inf}(s_{a_i}) &= \{\bullet a_i\},\ 1 \leq i \leq 3, \\
\mathsf{inf}(s_{A_i}) &= \{\bullet A_i a_i\},\ 1 \leq i \leq 3, \\
\mathsf{inf}(c) &= \emptyset,\ c \notin \{s_{a_i}, s_{A_i} \mid 1 \leq i \leq 3\},
\end{aligned}$$

and the rules are listed and explained below.

Consider the string from $\mathsf{cfg}(q_0)$. As it evolves throughout the computation, its part between copies of A_i holds the value of register i plus one, represented by symbols a_i in unary.

For each instruction $(q : i+, q', q'') \in I$ we have the following rules in R:

(+1) $q/s_{a_i} \rightarrow_{a_i} q'$,

(+2) $q/s_{a_i} \rightarrow_{a_i} q''$,

For each instruction $(q : i-, q', q'') \in I$ we have the following rules in R:

(-1) $q \rightarrow_{a_i} q'/test$,

(-2) $q \rightarrow_{A_i} z_q/test$,

(-3) $z_q/s_{A_i} \rightarrow q''$.

The set R also has the following rules:

(*1) $test \rightarrow_{a_i} fail/fail$,

(*2) $fail/fail \rightarrow_{a_i} test$.

(*3) $q_0 \rightarrow_{\#} test/test$.

Suppose that the value of register j is n_j, for $1 \leq j \leq 3$, and the state is q. Then the string $s = \#A_1a_1^{n_1+1}A_1A_2a_2^{n_2+1}A_2A_3a_3^{n_3+1}A_3\# = s_1A_ia_i^{n_i+1}A_is_2$ is present in cell q.

If q is assigned an instruction incrementing register i, then one copy of $\bullet a_i$ from cell s_{a_i} is inserted somewhere into the part $a_i^{n_i+1}$ of s, and the resulting string $s_1A_ia_i^{n_i+2}A_is_2$ is sent to either cell q' by (+1), or to cell q'' by (+2).

Consider the case when q is assigned a conditional subtract instruction of register i.

If $n_1 > 0$, then the correct simulation is the following:

$$s_1 A_i a_i^{n_i+1} A_i s_2 \text{ in } q \Rightarrow^{(-1)} s_1 A_i a_i^{n_i+1} A_i s_2 \text{ in } q'.$$

A string $\bullet a_i$ is sent to cell *test*. If rule (-1) is applied to non-adjacent occurrences of a_i, then a string of more than one symbol a_i appears in cell *test*, and the computation never halts because of the rules $(*1)$ and $(*2)$. Notice that if $n_1 = 0$, then only one occurrence of a_1 is present, so (-1) is not applicable.

If $n_1 = 0$, then the correct simulation is the following:

$$s_1 A_i a_i A_i s_2 \text{ in } q \Rightarrow^{(-2)} s_1 A_i s_2 \text{ in } z_q \Rightarrow^{(-3)} s_1 A_i a_i A_i s_2 \text{ in } q_5.$$

A string $\bullet A_i a_i$ is sent to cell *test*. Then, an identical string from cell s_{A_i} is inserted, restoring the shape of the string. Notice that if $n_1 > 0$, then then a string of one A_1 and more than one symbol a_1 appears in cell *test*, and the computation never halts because of the rules $(*1)$ and $(*2)$.

Finally, when the computation of M reaches the final state, the string representing the final configuration of M in the form

$$\#A_1 a_1^{n_1+1} A_1 A_2 a_2 A_2 A_3 a_3 A_3 \#$$

is present in cell q_f; its length is $n_1 + 11$, where n_1 is the number generated by M.

Hence, Π has a halting computation generating a number $n + 11$, $n \geq 0$ if and only if M has a halting computation generating n, i.e., if and only if $n + 11 \in L$. As for numbers smaller than 11, they can be generated by Π (and only they) because the corresponding strings are already present in q_f. We only need to mention that they are considered as a part of the result since the computation may always halt, i.e., by $(*3)$.

The maximally parallel nature of the computation is not required for the theorem above to hold. Indeed, any string in q_f is generated from a linear initial string by inserting strings present in infinite copies, and excising strings that are never inserted back; the only interaction between them is by making sure that they do not lead to an infinite computation. This is equally true for asynchronous systems. □

4. Discussion

We showed the computational completeness of distributed systems with ciliate operations without contexts. Certain related questions are currently open and present interest.

From the point of view of biological inspiration, it is interesting to consider a dedicated region, called environment, and require that in all other regions only finite multisets of strings are present.

Problem 4.1. *Would this decrease the generative power of such systems?*

Since P systems with ciliate operations are string-processing devices, it is natural to ask the following question.

Problem 4.2. *What is the class of* languages *characterized by P systems with ciliate operations.*

Clearly, Theorem 3.1. already implies non-recursiveness. However, with ciliate operations it seems to be very difficult to organize the output in linear order.

Other interesting questions come from an observation that regulation by targets, i.e., specifying the regions of reactants and products of each operation, provides a rather strong control. How does the power of P systems with ciliate operations depend on the "amount" of control, e.g., consider the following question.

Problem 4.3. *What is the computational power of ciliate operations of the form* $i \to_p i/j$, $i \to_p j/i$, $i \to_p j/j$, $i/i \to_p j$, $i/j \to_p i$, $i/j \to_p j$?

We expect such systems to be still quite powerful.

Problem 4.4. *What is the computational power of ciliate operations without targets?*

In these case, it seems unlikely that some nontrivial computation can be performed, since insertion operation has no context in either string, while excision operation cannot control the excised segment.

Should the demands or restrictions in the problems stated above be "too strong", the following extensions of the systems may be considered.

Extension 4.1. *Using maximal parallelism.*

Extension 4.2. *Allowing pointers to be strings as opposed to symbols.*

Extension 4.3. *Allowing the two copies of a pointer in the semantics of the rule to be different, and specifying both in the rule.*

We also mention some particular related observations.

Remark 4.1. It is easy to see that Theorem 3.1. holds also if rules of the form $i/j \to_p k$ are replaced by rules $i(u) \to_p k$ ("in a string from cell i duplicate a substring p and insert between these two copies of p a *specific* string u"). No strings would then need to be present in infinite supply. Moreover, it would then be explicit that different strings in the system do not interact.

Remark 4.2. Excision and insertion can be defined on linear strings only, in style of synchronized insertion/deletion, see [1]: rule $j/k \to_p i$, applied to upw in region j and vp in region k, yields a string $upvpw$ in region i, while rule $i \to_p j/k$ has opposite effect. It is easy to see that Theorem 3.1. then holds with linear strings only.

Remark 4.3. Deterministic simulation of register machines is possible under the following assumptions: (a) excision is not applicable to the occurrences of a pointer that do not contain other occurrences of the same pointer; (b) the operations are only defined as excising a circular string from a linear string and inserting a circular string in a linear string; (c) maximal parallelism.

Remark 4.4. In case of rules inserting and deleting *specific* strings, the computational completeness is achieved even in the non-distributed case, see [9].

Acknowledgements. The author gratefully acknowledges the support by Academy of Finland, project 203667, and the Science and Technology Center in Ukraine, project 4032. The author thanks Ion Petre, Sergey Verlan and Tseren-Onolt Ishdorj for useful discussions.

References

[1] DALEY M., KARI L., MCQUILLAN I., *Families of Languages Defined by Ciliate Bio-Operations*, Theoretical Computer Science, **320**(1), 2004, pp. 51–69.

[2] EHRENFEUCHT A., PRESCOTT D.M., ROZENBERG G., *Computational Aspects of Gene (Un)scrambling in Ciliates*, in L. F. Landweber, E. Winfree (Eds.) *Evolution as Computation*, Springer, Berlin, 2001, pp. 216-256.

[3] FREUND R., VERLAN S., *A Formal Framework for P Systems*, in G. Eleftherakis, P. Kefalas, Gh. Păun (Eds.) *Preproceedings of Eight Workshop on Membrane Computing (WMC8)*, Thessaloniki, Greece, pp. 317–330.

[4] ISHDORJ T.-O., PETRE I., *Computing Through Gene Assembly*, in S. G. Akl et al. (Eds.), *Unconventional Computation* 2007, *Lecture Notes in Computer Science* **4618**, Springer, 2007, pp. 91–105, and *TUCS Technical Report* **816**, 2007.

[5] ISHDORJ T.-O., PETRE I., ROGOJIN V., *Computational Power of Intramolecular Gene Assembly*, International Journal of Foundations of Computer Science, **18** (5), 2007, pp. 1123–1136, and *TUCS Technical Report* **815**, 2007.

[6] LANDWEBER L. F., KARI L., *The Evolution of Cellular Computing: Nature's Solution to a Computational Problem*, in *Proceedings of the 4th DIMACS Meeting on DNA-Based Computers*, Philadelphia, PA, 1998, pp. 3–15.

[7] LANDWEBER L. F., KARI L., *Universal Molecular Computation in Ciliates*, in L. F. Landweber and E. Winfree (Eds.) *Evolution as Computation*, Springer, Berlin, 2002.

[8] PRESCOTT D. M., EHRENFEUCHT A., ROZENBERG G., *Molecular Operations for DNA Processing in Hypotrichous Ciliates*, Europ. J. Protistology, **37**, 2001, pp. 241–260.

[9] MARGENSTERN M., PĂUN GH., ROGOZHIN YU., VERLAN S., *Context-Free Insertion-Deletion Systems*, Theoretical Computer Science, **330**(2), 2005, pp. 339–348.

[10] PĂUN GH., *Membrane Computing. An Introduction.* Springer, Berlin, 2002.

[11] PĂUN GH., YOKOMORI T., *Membrane Computing Based on Splicing*, in E. Winfree, D. Gifford (Eds.) *Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers*, MIT, 1999, pp. 213–227.

[12] The P systems web page. `http://psystems.disco.unimib.it/`.