Proceedings of Grammar Systems Week 2004, edited by E. Csuhaj-Varjú and Gy. Vaszil,
MTA SZTAKI, Budapest, pages 126 - 140, 2004.

Parallel Communicating Pushdown Transducer Systems

Liliana Cojocaru, Carlos Martín-Vide

Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
liliana.cojocaru@estudiants.urv.es, cmv@astor.urv.es

Abstract

In this paper a new variant of translating devices is presented. We propose a new model based on the interconnection of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. We focus on the strategy of communication by stacks, i.e., the content of the pushdown memory of each component of the system is shared with each other component, and transferred into the pushdown memory of the component that asked for it. They communicate also by the output tapes, i.e., whenever a query symbol appears on the top of a stack, the string yielded so far, on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that inquired for it. We call these devices *Parallel Communicating Pushdown Transducer Systems* (henceforth PCPTS). Depending on the protocol of communication we define several variants of PCPTS, e.g. returning or not returning, centralized or not. The strategy of exchanging data through the pushdown memory increases the complexity of the outputted languages. We investigate the computational power of these systems by taking into consideration the computational power of parallel communicating pushdown automata systems. Descriptional complexity reasons suggest us to use PCPTS in DNA computing. Several examples illustrate the manner in which this application is performed.

1 Introduction

From the very beginning, due to their practical applicability, translating systems turned out to be a rich and interesting subject area in the field of computational linguistics. The main goal of this paper is to describe a new variant of translating devices obtained by improving parallel communicating pushdown automata systems with output capabilities. The mechanism could have interesting applications in computational morphology and phonology, in speech recognition, artificial intelligence or communication among agents, in splicing systems and in DNA computing.

Finite-state transducers have been introduced for the first time in [11]. Since then they have increasingly developed and diversified due to their flexibility in representing and generating a large size of structural data, by using time and space optimal memory, e.g. the transducer minimization algorithm [21]. They have been applied

especially in natural language processing, in fields such as computational morphology and phonology [14], lexical analyzers [15] or speech recognition [20]. They have been extended to algebraic transducers [16] and weighted finite-state transducers [22]. Watson-Crick transducers have been developed in order to manipulate DNA molecules. They are described in [23]. If a pushdown automaton is provided with output capability, the resulting machine is a pushdown transducer. It comes in the literature from [9], [10] and [12], with many other succeeding papers. Parallel communicating pushdown automata are systems composed of several pushdown automata working independently but communicating with each other by stacks, under a specified protocol of cooperation. The protocol of cooperation consists in exchanging strings that have been generated up to the request moment through the pushdown memory. They have been defined and investigated in [4], [5], [17], [18], [19] and [27] with forerunner related papers [1], [13]. If a parallel communicating pushdown automata system is improved with output capability the resulting device is called in this paper Parallel Communicating Pushdown Transducer Systems. The new model is based on the interconnection of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. The communication is done through stacks, i.e., the content of the pushdown memory of each component of the system is shared with each other component, and transferred into the pushdown memory of the component that asked for it. They communicate also by the output tapes, i.e., whenever a query symbol appears on the top of a stack the string yielded so far, on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that asked for it. Depending on the protocol of communication, we propose several variants of PCPTS, e.g. returning or not-returning, centralized or not. The study of the computational power and possible applications of such systems in DNA computing is another goal of this paper. The theoretical approaches are accomplished by several examples that illustrate the manner in which PCPTSs work.

2 Preliminaries

Our aim in this section is to introduce the main concepts related to pushdown transducers and parallel communicating transducer systems. A setting of the notations of concepts from the topic covered in the paper will be given, too. We assume the reader to be familiar with the basic notions of formal language theory. Let Σ^* be the set of words over an alphabet Σ, composed of a finite set of letters, let λ be the empty word, and let w^R and $|w|$, $w \in \Sigma^*$, be the reverse and the length of the word w, respectively. We denote by $\Sigma^+ = \Sigma^* - \{\lambda\}$ and by $|\Sigma|$, the cardinality of the set Σ. $\bar{\Sigma}$ is the twin alphabet of Σ, i.e. $\bar{\Sigma} = \{\bar{x} | x \in \Sigma\}$. A mapping $h : \Sigma^* \longrightarrow T^*$, defined by $h(\lambda) = \{\lambda\}$ and $h(x_1x_2) = h(x_1)h(x_2)$, for x_1, $x_2 \in \Sigma^*$, is called a morphism. If $T \subseteq \Sigma$, then a projection associated with T, is a morphism $h_T : \Sigma^* \longrightarrow T^*$, defined by $h_T(x) = x$, for all $x \in T$, and $h_T(y) = \lambda$, for all $y \in \Sigma - T$.

2.1 Pushdown Transducers

A pushdown transducer is obtained by permitting a pushdown automaton (henceforth PDA) to emit a string of output symbols on each transition.

Definition 1 A pushdown transducer (PDT) is an 8-tuple:

$$P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$$

where:

- Q is a set of states, Σ is the input alphabet, Γ is the alphabet of pushdown memory and Δ is the output alphabet (all these sets are finite),
- δ is a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ into the finite subsets of $Q \times \Gamma^* \times \Delta^*$,
- q_0 the initial state, $Z_0 \in \Gamma$ is the start symbol from the pushdown memory and $F \subseteq Q$, the set of final states .

A configuration of P is a 4-tuple (q, x, α, y), where q represents the current state of the finite control, x is the unread portion of the input, α is the content of the pushdown memory and y is the output string emitted up to this point. The leftmost symbol of α is the topmost pushdown symbol .

If $\delta(q, a, Z)$ contains (p, α, z), then we write:

(1) $$(q, ax, Z\gamma, y) \vdash (p, x, \alpha\gamma, yz), \qquad \text{for all } x \in \Sigma^*,\ y \in \Delta^*, \text{ and } \gamma \in \Gamma^*.$$

If $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$ is a PDT, then the underlying pushdown automaton of P, is the pushdown automaton defined as:

$$A = (Q, \Sigma, \Gamma, \delta', q_0, Z_0, F),$$

where $(p, \alpha) \in \delta'(q, a, Z)$ if and only if $(p, \alpha, z) \in \delta(q, a, Z)$ for some $z \in \Delta^*$.

Using the relation (1), v is an output for $u \in L$, *by final states*, if:

(2) $$(q_0, u, Z_0, \lambda) \vdash^* (q, \lambda, \alpha, v), \quad \text{where } q \in F.$$

v is an output for $u \in L$, *by empty pushdown memory*, if:

(3) $$(q_0, u, Z_0, \lambda) \vdash^* (q, \lambda, \lambda, v), \quad \text{where } q \in Q.$$

Let P be a PDT and $L \subseteq \Sigma^*$ a language over Σ, recognized by the underlying pushdown automaton A. *The transduction realized by* P is a function

$$T_P : \Sigma^* \longrightarrow \Delta^* \quad \text{defined as:}$$

$T_P = \{(u, v) |$ such that $u \in \Sigma^*$ and $v \in \Delta^*$ satisfy one of the relations (2),(3)$\}$.

Thus the transduction function associated to the transducer P returns, for each input word u, the set of all words generated in the output tape of P at the end of a successful computation on u.

A PDT is *deterministic* (DPDT for short), if the following conditions hold, for all $q \in Q$:

- $|\delta(q, a, Z)| \leq 1$ for each $a \in \Sigma \cup \{\lambda\}$;
- if $|\delta(q, \lambda, Z)| \neq 0$ then for all $a \in \Sigma$, $|\delta(q, a, Z)| = 0$.

2.2 Parallel Communicating Pushdown Transducer Systems

Definition 2 A parallel communicating pushdown transducer system of degree n ($pcpt(n)$ for short) is a system:

$$\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K),$$

where:

1. Σ is the input alphabet,
2. Δ is the output alphabet,

3. Γ is the alphabet of pushdown symbols.

4. For each $1 \leq i \leq n$, $T_i = (Q_i, \Sigma, \Gamma, \Delta, \delta_i, q_i, Z_i, F_i)$ is a pushdown transducer where Q_i is the set of states, $q_i \in Q_i$ is the initial state of the transducer T_i, Σ is the input alphabet, Γ is the pushdown alphabet, Δ is the finite output alphabet, $Z_i \in \Gamma$ is the initial contents of the pushdown memory, $F_i \subseteq Q_i$ is the set of final states of the transducer T_i, and δ_i is the transition mapping defined from $Q_i \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q_i \times \Gamma^* \times \Delta^*$.

5. $K \subseteq \{K_1, K_2, \ldots, K_n\} \subseteq \Gamma$ is the set of query symbols.

The pushdown transducers $T_1, T_2, ..., T_n$ are called the *components* of the system $\mathcal{T}$. If there exists only one $1 \leq i \leq n$, such that only the transducer T_i is allowed to query, then the system is called *centralized* (henceforth *cpcpt*(n)). The master of this system is the component T_i.

Definition 3 A PCPTS is said to be *deterministic* (DPCPTS for short), if the next conditions hold, for all $q \in Q_i$, $1 \leq i \leq n$:

- $|\delta_i(q, a, Z)| \leq 1$ for each $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$;
- if $|\delta_i(q, \lambda, Z)| \neq 0$ then for all $a \in \Sigma$, $Z \in \Gamma$, then $|\delta(q, a, Z)| = 0$.

A configuration of a PCPTS of degree n is a $4n$-tuple:

$$(s_1, x_1, \alpha_1, \gamma_1, s_2, x_2, \alpha_2, \gamma_2, \ldots, s_n, x_n, \alpha_n, \gamma_n)$$

where for all $1 \leq i \leq n$:

- $s_i \in Q_i$ is the current state of the component T_i,
- $x_i \in \Sigma^*$ is the remaining part of the input word that has not been read yet by T_i,
- $\alpha_i \in \Gamma^*$ is the contents of the i-th stack,
- $\gamma_i \in \Delta^*$ is the output string emitted up to that point by T_i.

The transition relations are defined on the set of all configurations of $\mathcal{T}$ as follows:

(4) $(s_1, x_1, B_1\alpha_1, \gamma_1, \ldots, s_n, x_n, B_n\alpha_n, \gamma_n) \vdash (p_1, y_1, \beta_1\alpha_1, \gamma_1', \ldots, p_n, y_n, \beta_n\alpha_n, \gamma_n')$

where $B_i \in \Gamma, \alpha_i, \beta_i \in \Gamma^*$ and $\gamma_i, \gamma_i' \in \Delta^*$ for all $1 \leq i \leq n$, iff one of the following two conditions holds, for all $1 \leq i \leq n$:

(4.i) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in \Sigma \cup \{\lambda\}, \gamma_i' = \gamma_i z_i, (p_i, \beta_i, z_i) \in \delta_i(s_i, a_i, B_i)$,

(4.ii) for all $i, 1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K, \beta_i = B_{j_i}\alpha_{j_i}$, and $\gamma_i' = \gamma_i\gamma_{j_i}$. For all other $r, 1 \leq r \leq n, \beta_r = B_r$, and $y_t = x_t$, $p_t = s_t$, for all $t, 1 \leq t \leq n$.

(5) $(s_1, x_1, B_1\alpha_1, \gamma_1, \ldots, s_n, x_n, B_n\alpha_n, \gamma_n) \vdash_r (p_1, y_1, \beta_1\alpha_1, \gamma_1', \ldots, p_n, y_n, \beta_n\alpha_n, \gamma_n')$

where $B_i \in \Gamma, \alpha_i, \beta_i \in \Gamma^*, \gamma_i, \gamma_i' \in \Delta^*$, $1 \leq i \leq n$, if one of the following two conditions holds, for all $1 \leq i \leq n$:

(5.i) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in \Sigma \cup \{\lambda\}, \gamma_i' = \gamma_i z_i, (p_i, \beta_i, z_i) \in \delta_i(s_i, a_i, B_i)$,

(5.ii) for all $1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K, \beta_i = B_{j_i}\alpha_{j_i}, \beta_{j_i}\alpha_{j_i} = Z_{j_i}, \gamma_i' = \gamma_i\gamma_{j_i}$, and $\gamma_{j_i} = \lambda$. For all the other $r, 1 \leq r \leq n, \beta_r = B_r$, and $y_t = x_t$, $p_t = s_t$, for all $t, 1 \leq t \leq n$.

A *pcpt*(n) system whose moves are all based on the relation $\vdash_r$ is said to be *a returning* (henceforth *rpcpt*(n)), and non-returning otherwise.

Note that, according to the definition of transitions sketched in (4) and (5), a PCPTS functions as follows: whenever a component has a query symbol on top of the pushdown memory, it gets into communication with the component to which the

query symbol belongs. The communication consists of replacing the query symbol with the entire content of the pushdown memory of the interrogated component, that is the sender. Simultaneously, the string yielded so far on the output tape of the component to which the query symbol belongs, is transfered into the output tape of the component that queried for it. In the case of returning systems the pushdown memory of the component that sent the information returns to its initial symbol, while the output tape of the same component returns to the empty tape.
The reflexive and transitive closure of relation $\vdash$ ($\vdash_r$) is denoted by $\vdash^*$ ($\vdash_r^*$).
If the underlying pushdown automaton A_i, of the pushdown transducer T_i, is an extended one, for each $1 \leq i \leq n$, then the system $\mathcal{T}$, is said to be an extended PCPTS (henceforth EPCPTS). It means that the transition mappings δ_i are defined from finite subsets of $Q_i \times (\Sigma \cup \{\lambda\}) \times \Gamma^*$ to finite subsets of $Q_i \times \Gamma^* \times \Delta^*$, for all $1 \leq i \leq n$. Transitions for an EPCPT are defined in the same way as for PCPTS, with difference that extended systems may read words $B_i \in \Gamma^*$ instead of symbols. Note that PCPTS and EPCPTS are equivalent from a computational point of view.

Using the relation defined in (4), the set of words $v_i \in \Delta^*$, $1 \leq i \leq n$ is an output for $u \in \Sigma^*$, of a PCPTS with n components, if and only if:
(6) $(q_1, u, Z_1, \lambda, \ldots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \ldots, s_n, \lambda, \alpha_n, v_n), s_i \in F_i, 1 \leq i \leq n$ for a PCPTS accepting by final states, and :
(6') $(q_1, u, Z_1, \lambda, \ldots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \lambda, v_1, \ldots, s_n, \lambda, \lambda, v_n), s_i \in Q_i, 1 \leq i \leq n$ for a PCPTS accepting by empty pushdown memory .
Replacing the relation $\vdash^*$ with the relation $\vdash_r^*$ in (6) and in (6'), the definition of the output words, for a RPCPTS accepting by final states or by empty pushdown memory is obtained, respectively.

If $\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K)$ is $pcpt(n)$, then the underlying $pcpa(n)$ of $\mathcal{T}$ is defined as $\mathcal{A} = (\Sigma, \Gamma, A_1, A_2, \ldots, A_n, K)$, where A_i is the underlying PDA associated to T_i, for all $1 \leq i \leq n$. Let $\mathcal{T}$ be a PDT and $L \subseteq \Sigma^*$ a language over Σ, recognized by the underlying pushdown automaton $\mathcal{A}$.

Definition 4 Let $\mathcal{T} = (\Sigma, \Delta, \Gamma, T_1, T_2, \ldots, T_n, K)$, be a (nonreturning) PCPTS with n components. The **transduction function** of the i^{th} component of the system $\mathcal{T}$ is defined as :

$$\mathrm{T}_i(u) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \cdots, s_i, \lambda, \alpha_i, v_i, \cdots, s_n, \lambda, \alpha_n, v_n), s_j \in F_j, 1 \leq j \leq n\},$$

for any $1 \leq i \leq n$, if the word u is accepted by final states, and

$$\mathrm{T}_i(u) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \lambda, v_1, \cdots, s_i, \lambda, \lambda, v_i, \cdots, s_n, \lambda, \lambda, v_n), s_j \in Q_j, 1 \leq j \leq n\},$$

for any $1 \leq i \leq n$, if the word u is accepted by empty pushdown memory.

If in the above definition the relation $\vdash_r$ is used instead of $\vdash$ then the transduction function of the i^{th} component, for the returning case, denoted by TR_i is obtained. Thus each component T_i, $1 \leq i \leq n$, has associated a transduction mapping that returns for each input word u the set of all words yielded by that component.

The *language yielded by the component* T_i, $1 \leq i \leq n$ is defined as:

$$T_i(L) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^* (s_1, \lambda, \alpha_1, v_1, \cdots, s_i, \lambda, \alpha_i, v_i, \cdots, s_n, \lambda, \alpha_n, v_n), s_j \in F_j, u \in L, 1 \leq j \leq n\},$$ or by

$T_i(L) = \{v_i \in \Delta^* | (q_1, u, Z_1, \lambda, \cdots, q_i, u, Z_i, \lambda, \cdots, q_n, u, Z_n, \lambda) \vdash^*$
$(s_1, \lambda, \lambda, v_1, \cdots, s_i, \lambda, \lambda, v_i, \cdots, s_n, \lambda, \lambda, v_n), s_j \in Q_j, u \in L, 1 \leq j \leq n\}$.

If in the above definition the relation $\vdash_r$ is used instead of $\vdash$ then the definition of the language yielded by the i^{th} component, for the returning case, denoted by $TR_i(L)$ is obtained. Due to the definition of PCPTS different components can output different languages. The *transduction system mapping* of a PCPTS, with n components, is defined as: $\mathrm{T} = (\mathrm{T}_1, \mathrm{T}_2, \cdots, \mathrm{T}_n)$, for non-returning case, and as $\mathrm{TR} = (\mathrm{TR}_1, \mathrm{TR}_2, \cdots, \mathrm{TR}_n)$, for the returning case.

3 On the Computational Power of PCPTS

The problem that arises now, is to study the type of the possible languages yielded by a PCPTS having as input an arbitrary language $L \subseteq \Sigma^*$. Let RE, CS, CF, LIN, and REG be the families of languages generated by arbitrary, context-sensitive, context-free, linear and regular grammars, respectively. The next inclusions hold:

(7) $$REG \subset LIN \subset CF \subset CS \subset RE$$

For each parallel communicating pushdown automata system we can build a PCPTS, having the same structure as the underlying system. That is why pushdown transducer systems will be able to accept and to generate at least the same type of languages. In [4] has been proved that the family of languages accepted by a PCPAS with two components equals the RE languages. Therefore, the family of languages yielded by PCPTS with only two components covers the the family of RE languages. Next we center upon the answers of the following questions:

" *Given an input language from a class X of the hierarchy (7), could it be possible to generate a language from a class Y such that* $X \subset Y$ *?*" Furthermore,

" *Which is the smallest class of languages from the hierarchy (7) that could be used, as input language, by a PCPTS to cover the RE family ?*", and even more:

" *Which is the minimal number of components that a PCPTS should have in order to satisfy the above aims?*"

Answers to these questions will be given in the sequel. The following example is relevant for the computational power of PCPTS.

Example Let us consider a deterministic $pcpt(2)$, defined below:

$$\mathcal{T} = (\{a, c\}, \{Z_0, Z_1, a\}, T_1, T_2, \{K_1, K_2\}),$$

in which $T_i = (Q_i, \Sigma, \Gamma, \Delta, \delta_i, q_i, Z_i, F_i)$, $i \in \{1, 2\}$, are two pushdown automata, defined as: $T_1 = (\{q_1, s_1, r_1, p_1\}, \{a, c\}, \{Z_0, Z_1, a\}, \{a\}, \delta_1, q_1, Z_1, \{p_1\})$,

$T_2 = (\{q_2, s_2, p_2\}, \{a, c\}, \{Z_0, Z_1, a\}, \{a\}, \delta_2, q_2, Z_1, \{p_2\})$,

having the transition mappings :

1. $\delta_1(q_1, a, Z_1) = (s_1, Z_1, \lambda)$
2. $\delta_1(s_1, a, Z_1) = (s_1, aaZ_1, \lambda)$
3. $\delta_1(s_1, a, a) = (s_1, aaa, \lambda)$
4. $\delta_1(s_1, c, a) = (r_1, Z_0aaa, \lambda)$
5. $\delta_1(r_1, \lambda, a) = (r_1, \lambda, \lambda)$
6. $\delta_1(r_1, \lambda, Z_0) = (r_1, \lambda, \lambda)$
7. $\delta_1(r_1, \lambda, Z_1) = (p_1, K_2, \lambda)$
8. $\delta_1(p_1, \lambda, a) = (p_1, \lambda, a)$

1. $\delta_2(q_2, \lambda, Z_1) = (q_2, K_1, \lambda)$
2. $\delta_2(q_2, \lambda, a) = (q_2, K_1a, \lambda)$
3. $\delta_2(q_2, \lambda, Z_0) = (s_2, \lambda, \lambda)$
4. $\delta_2(s_2, a, a) = (s_2, \lambda, \lambda)$
5. $\delta_2(s_2, c, a) = (p_2, \lambda, \lambda)$
6. $\delta_2(p_2, \lambda, a) = (p_2, \lambda, \lambda)$
7. $\delta_2(p_2, \lambda, Z_1) = (r_2, \lambda, \lambda)$
8. $\delta_2(r_2, \lambda, a) = (r_2, \lambda, a)$

9. $\delta_1(p_1, \lambda, Z_1) = (p_1, \lambda, a)$ 9. $\delta_2(r_2, \lambda, Z_1) = (r_2, \lambda, a)$.

Taking as input the regular language: $L = \{a^n c \mid n \geq 1\}$ the above PCPTS will yield on both of the output tapes the following non-context-free language $L = \{a^{n^2-1} \mid n \geq 1\}$, by following the next transitions:

$$\underline{(q_1, a^n c, Z_1, \lambda, q_2, a^n c, Z_1, \lambda)} \vdash_{1,1} (s_1, a^{n-1}c, Z_1, \lambda, q_2, a^n c, K_1, \lambda)$$
$$\vdash_{2,1} (s_1, a^{n-2}c, aaZ_1, \lambda, q_2, a^n c, K_1, \lambda)$$
$$\vdash_{3,2}^{*(n-2)} (s_1, c, a^{2(n-1)}Z_1, \lambda, q_2, a^n c, a^{2(n-1)}Z_1 ... a^2 Z_1, \lambda)$$
$$\vdash_{4,2} (r_1, \lambda, Z_0 a^{2n} Z_1, \lambda, q_2, a^n c, K_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{6,3} (r_1, \lambda, a^{2n} Z_1, \lambda, s_2, a^n c, a^{2n} Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,4}^{*n} (r_1, \lambda, a^n Z_1, \lambda, s_2, c, a^n Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,5} (r_1, \lambda, a^{(n-1)} Z_1, \lambda, p_2, \lambda, a^{(n-1)} Z_1 a^{2(n-1)} Z_1 .. a^2 Z_1, \lambda)$$
$$\vdash_{5,6}^{*(n-1)} (r_1, \lambda, Z_1, \lambda, p_2, \lambda, Z_1 a^{2(n-1)} .. a^2 Z_1, \lambda)$$
$$\vdash_{7,7} (p_1, \lambda, K_2, \lambda, r_2, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda) \vdash$$
$$\vdash (p_1, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda, r_2, \lambda, a^{2(n-1)} Z_1 ... a^2 Z_1, \lambda)$$
$$\vdash_{8,8}^{*2(n-1)} (p_1, \lambda, Z_1 ... a^2 Z_1, a^{2(n-1)}, r_2, \lambda, Z_1 a ... a^2 Z_1, a^{2(n-1)})$$
$$\vdash_{9,9} (p_1, \lambda, a^{2(n-2)} Z_1 .. a^2 Z_1, a^{2(n-1)} a, r_2, \lambda, a^{2(n-2)} Z_1 .. a^2 Z_1, a^{2(n-1)} a)$$
$$\vdash_{8,8}^{*2(n-2)} (p_1, \lambda, Z_1 a_{...}^{2(n-3)} a^2 Z_1, a^{2(n-1)} a a^{2(n-2)}, r_2, \lambda, Z_1 a_{...}^{2(n-3)} a^2 Z_1, a^{2(n-1)} a a^{2(n-2)})$$
$$\vdash_{9,9} ... \vdash_{8,8}^{*2(n-3)} ... \vdash_{9,9} ... \vdash_{8,8}^{*2(n-4)} ... \vdash_{9,9} ... \vdash_{8,8}^{*4} ... \vdash_{9,9} ... \vdash_{8,8}^{*2} \vdash_{9,9}$$
$$\underline{(p_1, \lambda, \lambda, a^{(n-1)} a^{2(n-1)} a^{2(n-2)} .. a^4 a^2, r_2, \lambda, \lambda, a^{(n-1)} a^{2(n-1)} a^{2(n-2)} .. a^4 a^2)}$$

The output is: $a^{(n-1)} a^{2(n-1)} a^{2(n-2)} \ldots a^4 a^2 = a^{2(n-1)+...+4+2+n-1} = a^{n^2-1}$.

In order to express the computational power of PCPTS we will introduce several notations and basic results related to the *twin shuffle* operation on strings.

Definition 5 1) For two strings $x, y \in \Sigma^*$ and two symbols $a, b \in \Sigma$, the **shuffle** operation is defined as: (i) $x \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup x = \{x\}$,

(ii) $ax \sqcup\!\sqcup by = a(x \sqcup\!\sqcup by) \cup b(ax \sqcup\!\sqcup y)$.

2) The **shuffle of two languages** $L_1, L_2 \in \Sigma^*$ is:

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

3) The **twin shuffle language** over Σ, is defined as:

$$\mathrm{TS}_\Sigma = \bigcup_{x \in \Sigma^*} x \sqcup\!\sqcup \bar{x}.$$

The following representation of the family RE is well known from [23]:

Theorem 1.
Each recursively enumerable language $L \subseteq T^*$ can be written as $L = h_T(TS_\Sigma \cap R)$, where Σ is an alphabet, R is a regular language, and h_T is a projection associated with $T \subseteq \Sigma$.

Related to the computational power of PCPTS, the next result from [4] is useful in our considerations:

Theorem 2.
1. A language $L \subseteq T^*$ is recursively enumerable if and only if $L = h_T(Rec_r(\mathcal{A}))$, where $\mathcal{A}$ is a parallel communicating pushdown automata system[1] of type $x(2)$ with

[1]For the definition and more information related to these systems the reader is referred to [4].

$x \in \{rcpcpa, rpcpa\}$.
2. A language $L \subseteq T^*$ is recursively enumerable iff $L = h_T(Rec(\mathcal{A}))$, where $\mathcal{A}$ is a $cpcpa(2)$.

Due to the above theorems we have:

Theorem 3.
For each recursively enumerable language L there exists language $R \in REG$ and an automata system $\mathcal{T}$ of type $x(2)$, with $x \in \{rcpcpt, rpcpt, cpcpt\}$ such that $\mathrm{T}_{\mathcal{T}}(R) = (L, \lambda)$.

Proof. In the proof of Theorem 2 it has been shown that there exists a PCPAS of type $x(2)$, with $x \in \{rcpcpa, rpcpa, cpcpa\}$, that accepts the language $TS_\Sigma \cap R$, where $R \in REG$. With respects to Theorem 1, for each language $L \in RE$, there exists a projection h_T, $T \subseteq \Sigma$, and a language $R \in REG$, such that the image of $TS_\Sigma \cap R$ through h_T is the language $L \in RE$. With respect to Theorem 1 and Theorem 2 for each language $L \in RE$ there exists a language $R \in REG$, such that we can build a PCPTS, $\mathcal{T}$ of type $x(2)$, with $x \in \{rcpcpt, rpcpt, cpcpt\}$, that for each word $w \in R$, if $w \in TS_\Sigma$, $\mathcal{T}$ yields on its first output tape the word $h_T(w)$ (by simulating the projection h_T), and issues the empty word otherwise. The other component will output only the empty word. □

4 How to Apply PCPTS - A Descriptional Complexity Reasoning

As we have seen in the last section PCPTS are able to output RE languages having as input very simple REG languages. This result allows PCPTS to be used to simulate natural phenomena in which very simple data, seen as strings of REG languages, is processed into more complicated CS languages. Gene assembly in ciliates can be considered such a string processing phenomenon.

4.1 The Gene Assembly Process in Ciliates

Ciliates are single cell organisms that have two types of nuclei: *micronuclei* and *macronuclei*. The intramolecular process of transformation of a micronucleus into a macronucleus is known in literature as *gene assembly process in ciliates*. Micronuclear genes are composed of combinations of residual segments, called *Internal Eliminated Segments* (henceforth IES) and active segments, called *Macronuclear Destinated Segments* (henceforth MDS). The prominent feature of the gene assembly process consists in the very spectacular manner in which, during the transformation of micronuclear genes into macronuclear genes, the MDS regions are spliced and the IES portions are excised. The splicing process is done in some weaker points of the micronuclear genes, named *pointers*. These pointers are sequences of nucleotides, that bound the MDS components. The MDS breaking and reassembling is done such that at the end of the gene assembly process the yielded macronuclear genes will contain no residual segment. This procedure is performed with respect to three

molecular patterns that appear in the structure of a micronuclear gene. Each time a direct repeat pattern of pointers (p, p), i.e., no other pointer parts the two p pointers, appears in the structure of a micronuclear gene then the *ld*-(loop, direct repeat)-excision operation is performed. If an inverted repeat pattern of the form $(p, \bar{p})$ is found then the *hi*-(hairpin, inverted repeat)-excision/reinsertion operation is performed. In the case that the micronuclear gene contains an alternating direct repeat pattern of pointers (p, r), i.e., an overlapping structures of the form (p, p) and (r, r), then the *dlad*-(double loop, alternating direct repeat)-excision/reinsertion operation is carried on. From a computational point of view the three molecular operations *ld, hi*, and *dlad* have been studied in several papers, e.g. [8], [7], [24], [26], [6]. Our model has been inspired from the theoretical interpretation of the gene assembly process given in [7], and is briefly described below.

Let Π_k be the *pointers alphabet*, formed by the micronuclear gene pointers, usually denoted by $\Pi_k = \{2, 3, ..., k, \bar{2}, \bar{3}, ..., \bar{k}\}$. A string $\pi \in \Pi_k^*$ is called *legal* if each symbol p from π, is duplicated either by p, or by $\bar{p}$, and at most two occurrences of p (or $\bar{p}$) are allowed in π. Three operations can be defined on the sets of all legal strings over the alphabet Π_k:

1. If $\pi = \pi_1 pp\pi_2$ then $\mathbf{ld}_p$ (the *string negative rule* for p) is defined as: $\mathbf{ld}_p(\pi) = \pi_1\pi_2$.

2. If $\pi = \pi_1 p\pi_2\bar{p}\pi_3$ then $\mathbf{hi}_p$ (the *string positive rule* for p) is defined as: $\mathbf{hi}_p(\pi) = \pi_1\mathbf{rs}(\pi_2)\pi_3$, where $\mathbf{rs}(\pi_2)$ is the *reversed switch*[2] of π_2.

3. If $\pi = \pi_1 p\pi_2 r\pi_3 p\pi_4 r\pi_5$ then $\mathbf{dlad}_{p,r}$ (the *string double rule* for p and r) is defined as: $\mathbf{dlad}_{p,r}(\pi) = \pi_1\pi_4\pi_3\pi_2\pi_5$.

A *string pointer reduction system over* Π_k, denoted by $SPRS_{\Pi_k}$, is a set $SPRS_{\Pi_k} = \{\mathbf{ld}_p, \mathbf{hi}_p, \mathbf{dlad}_{p,r} | p, r \in \Pi_k\}$. We denote by $D = (\pi; \rho_1, \rho_2, ..., \rho_l)$, $l \geq 1$, $\rho_i \in SPRS_{\Pi_k}$, $\pi \in \Pi_k^*$, π a legal string, the *reduction of* π by applying the rules ρ_i, $0 \leq i \leq l$, in the order of their appearance in the scheme. We say that D is a *successful* reduction scheme for a legal string π, if the resulting string after the scheme application on π is the empty string. For more information related to properties of *reduction schemes*, the reader is referred to [7]. The string pointer reduction system described above is a formalization of the splicing phenomena that take place during the gene assembly in ciliates. In [7] it has been shown that the $\mathbf{ld}_p$ operation simulates the *ld* operation, $\mathbf{hi}_p$ simulates the *hi* operation, and finally $\mathbf{dlad}_{p,r}$ simulates the *dlad* operation. The successful scheme reduction of a legal string is convertible into a successful strategy for a realistic MDS descriptor (that is the theoretical representation of the IES/MDS structure of a micronuclear gene). A successful strategy for a realistic MDS descriptor represents a gene assembly procedure that has as result a successful transformation of a micronuclear gene into a macronuclear one, i.e., no pointer will appear, in the structure of the resulted macronuclear gene at the end of the molecular computation. In [2] we described a computational model based on Parallel Communicating Finite Transducer Systems (henceforth PCFTS[3]) that performs the above molecular operations. In the next section, descriptional reasonings suggest us that PCPTS can be used better than PCFTS to simulate the gene assembly process in ciliates.

[2]For instance $\mathbf{rs}(\bar{3}4\bar{2}234) = \bar{4}\bar{3}\bar{2}2\bar{4}3$.

[3]For the definition of PCFTS the reader is referred to [3].

4.2 PCPTS - A Descriptional Complexity Analysis

In the following we propose two $pcpt(2)$ that simulate **hi** and **dlad** operations. Because finite transducers/pushdown transducers can check the "legality" of a given string, next we will consider these operations defined only on legal strings.

Simulation 1 The $\mathbf{hi}_p$ operation
Let $T = (\Pi_k, \overline{\Pi_k, \{Z_0\}, T_1, T_2, K})$ be a *cpcpt*, with the components
$T_1 = (\{q_0, q_p, q_{\bar{p}}, q_{[p]}, q_{[\bar{p}]}, q_f\}, \Pi_k, \Pi_k, \Pi_k \cup \{Z_0, Z_1\}, \delta_1, q_0, Z_0, \{q_0, q_f\})$
$T_2 = (\{q_0, s_p, s_{\bar{p}}\}, \Pi_k, \Pi_k, \{Z_1\}, \delta_2, q_0, Z_0, \{q_0, s_p, s_{\bar{p}}\})$
and δ mappings are defined, for all $x \in \Pi_k$, as follows:

1. $\delta_1(q_0, x, Z_0) = \{(q_0, Z_0, x)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
1'. $\delta_2(q_0, x, Z_1) = \{(q_0, Z_1, \lambda)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
2. $\delta_1(q_0, p, Z_0) = \{(q_p, Z_0Z_1, \lambda)\}$,
2'. $\delta_2(q_0, p, Z_1) = \{(s_p, Z_1, p)\}$,
3. $\delta_1(q_0, \bar{p}, Z_0) = \{(q_{\bar{p}}, Z_0Z_1, \lambda)\}$,
3'. $\delta_2(q_0, \bar{p}, Z_1) = \{(s_{\bar{p}}, Z_1, \bar{p})\}$,
4. $\delta_1(q_p, x, Z_0) = \{(q_p, Z_0x, \lambda)\}$, $x \in \Pi_k - \{\bar{p}\}$;
4'. $\delta_2(s_p, x, Z_1) = \{(s_p, Z_1, x)\}$, $x \in \Pi_k$;
5. $\delta_1(q_p, \lambda, Z_0) = \{(q_f, K_2, \lambda)\}$;
5'. $\delta_2(s_{\bar{p}}, x, Z_1) = \{(s_{\bar{p}}, Z_1, x)\}$, $x \in \Pi_k$;
6. $\delta_1(q_{\bar{p}}, x, Z_0) = \{(q_{\bar{p}}, Z_0x, \lambda)\}$, $x \in \Pi_k - \{p\}$;
6'. $\delta_2(s_p, \lambda, Z_1) = \{(s_p, Z_1, \lambda)\}$;
7. $\delta_1(q_{\bar{p}}, \lambda, Z_0) = \{(q_f, K_2, \lambda)\}$;
7'. $\delta_2(s_{\bar{p}}, \lambda, Z_1) = \{(s_{\bar{p}}, Z_1, \lambda)\}$.
8. $\delta_1(q_p, \bar{p}, Z_0) = \{(q_{[\bar{p}]}, \lambda, \lambda)\}$;
9. $\delta_1(q_{[\bar{p}]}, \lambda, x) = \{(q_{[\bar{p}]}, \lambda, \bar{x})\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
10. $\delta_1(q_{[\bar{p}]}, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$;
11. $\delta_1(q_{\bar{p}}, p, Z_0) = \{(q_{[p]}, \lambda, \lambda)\}$;
12. $\delta_1(q_{[p]}, \lambda, x) = \{(q_{[p]}, \lambda, \bar{x})\}$, $x \in \Pi_k - \{p, \bar{p}\}$;
13. $\delta_1(q_{[p]}, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$;
14. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, $x \in \Pi_k - \{p, \bar{p}\}$;

The above system works as follows: for the beginning both components read symbols from the input tape until they reach p or $\bar{p}$. Only the first component outputs the symbols that have been read. If the pointers p and $\bar{p}$ do not exist in the input string, then the first component yields on its output tape the (whole) input string, while the second component yields the empty string. In this case, the system ends the computation in the pair of states: (q_0, q_0). When both components read the first occurrence of the pointer $p/\bar{p}$, the first component reaches the state $q_p/q_{\bar{p}}$, and the second one reaches the state $s_p/s_{\bar{p}}$. From now on, each symbol that will be read by the first component in the state $q_p/q_{\bar{p}}$, will be memorized in the pushdown memory, without any output. Each symbol that will be read by the second component (including $p/\bar{p}$) will be yielded on the second output tape. If the twin symbol of p (i.e. $\bar{p}$) or the twin symbol of $\bar{p}$ (i.e. p) is not found in the input string then the components communicate with each other in order to let the first component to

output the (whole) input on the first tape. Note that, in this case the first component cannot query during the computational process, otherwise the query symbol will be replaced by the initial symbol of the stack of the second component, fact that will block the computation, (the first mapping δ_1 not being defined in the state $q_p/q_{\bar{p}}$ for this symbol). In this situation the computation ends in the pair of states (q_f, s_p), if p is found, and in the pair of states $(q_f, s_{\bar{p}})$, if $\bar{p}$ is found. When the pair $(p, \bar{p})/(\bar{p}, p)$ is found the system changes the pair of states $(q_p, s_p)/(q_{\bar{p}}, s_{\bar{p}})$ into the pair $(q_{[\bar{p}]}, s_p)/(q_{[p]}, s_{\bar{p}})$. From now on the first component yields on its output tape the reversed switch of π_2 (that has been stored in its pushdown memory) reading no input symbol. When the first component reaches the bottom of the stack, it begins to read and to output the rest of the symbols from the input tape. During all this time the second component reads all the input symbols, having no output. The system ends the computation in the pair of states (q_f, s_p), if p is found before $\bar{p}$, and in the pair $(q_f, s_{\bar{p}})$, if $\bar{p}$ is found before p. Furthermore, the above system makes a simultaneous searching for both of the pairs $(p, \bar{p})$, and $(\bar{p}, p)$, while the $rpcft(2)$ presented in [2] performs this operation only for the pair $(p, \bar{p})$. We have to eliminate the rules 3, 6, 7, 11, 12, 13, 3', 5', 7', and the states $q_{\bar{p}}$, $q_{[p]}$, $s_{\bar{p}}$, in order to obtain a $cpcpt(2)$ that performs the same task as the $rpcft(2)$ presented in [2]. Therefore, in what follows we compare the descriptional complexity of the $rpcft(2)$ with the reduced model of the $cpcpt(2)$ presented above.

In the **hi** simulation displayed in [2], the $rpcft(2)$ that performs this operation needs 8 states for each component, $5|\Pi_k| + 3$ rules for the first component, and $4(|\Pi_k|+1)$ rules for the second one. The reduced model of the $cpcpt(2)$ presented here needs only 4 states for the first component, 2 states for the second one, $4|\Pi_k|-3$ rules for the first component, and $2|\Pi_k|$ rules for the second one. The $rpcft(2)$ needs $|\pi_2|+2$ communication steps, while the above $cpcpt(2)$ needs only one communication, used only in the situation when the twin symbol of p (or $\bar{p}$) is not found in the input string.

Simulation 2 The $\mathbf{dlad}_{p,r}$ operation

The next *rpcpt* system performs the **dlad** operation:

$T = (\Pi_k, \Pi_k, \{Z_0\}, T_1, T_2, K)$ be a *cpcpt*, with the components

$T_1 = (\{q_r, q_f, q_0, q_1, q_2, q_3, q_4, q_5\}, \Pi_k, \Pi_k \cup \{Z, Z_0, Z_1, Z_2, \bar{Z}, \bar{\bar{Z}}, \hat{Z}\}, \delta_1, q_0, Z_1, \{q_0, q_f\})$

$T_2 = (\{s_0, s_p, s_f, s_r, s_{[r]}, s_{[p]}, s_{[\bar{r}]}\}, \Pi_k, \Pi_k, \{Z, Z_2, \bar{\bar{Z}}\}, \delta_2, s_0, Z_2, \{s_0, s_f, s_{\bar{r}}\})$

and δ mappings defined, for all $x \in \Pi_k$, as follows:

1. $\delta_1(q_0, \lambda, Z_1) = \{(q_1, Z_0 Z_1, \lambda)\}$, 1'. $\delta_2(s_0, x, Z_2) = \{(s_0, Z_2, \lambda)\}$, $x \neq p$;
2. $\delta_1(q_1, \lambda, Z_0) = \{(q_1, Z_0, \lambda)\}$, 2'. $\delta_2(s_0, \lambda, Z_2) = \{(s_f, K_1, \lambda)\}$,
3. $\delta_1(q_1, \lambda, Z_1) = \{(q_f, Z_1, \lambda)\}$, 3'. $\delta_2(s_0, p, Z_2) = \{(s_p, Z_2, \lambda)\}$,
4. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, 4'. $\delta_2(s_p, x, Z_2) = \{(s_p, Z_2, \lambda)\}$,$x \notin \{p, \bar{p}\}$
5. $\delta_1(q_1, \lambda, Z_0 Z_1) = \{(q_r, K_2, \lambda)\}$, 5'. $\delta_2(s_p, \{p, \bar{p}\}, Z_2) = \{(s_f, K_1, \lambda)\}$,
6. $\delta_1(q_r, \lambda, Z) = \{(q_r, Z, \lambda)\}$, 6'. $\delta_2(s_f, x, Z_0) = \{(s_f, Z_2, \lambda)\}$, $x \neq p$;
7. $\delta_1(q_r, \lambda, Z_1) = \{(q_r, K_2, \lambda)\}$, 7'. $\delta_2(s_f, x, Z_2) = \{(s_f, Z_2, \lambda)\}$, $x \neq p$;
8. $\delta_1(q_r, \lambda, \bar{Z}) = \{(q_f, Z_1, \lambda)\}$, 8'. $\delta_2(s_f, x, \bar{\bar{Z}}) = \{(s_f, \bar{\bar{Z}}, \lambda)\}$;
9. $\delta_1(q_r, \lambda, \bar{\bar{Z}}) = \{(q_1, \bar{\bar{Z}}, \lambda)\}$, 9'. $\delta_2(s_f, \lambda, Z_2) = \{(s_f, Z_2, \lambda)\}$,
10. $\delta_1(q_1, \lambda, \bar{\bar{Z}}) = \{(q_1, \bar{\bar{Z}}, \lambda)\}$, 10'. $\delta_2(s_p, r, Z_2) = \{(s_r, Z_2, \lambda)\}$,
11. $\delta_1(q_1, x, \bar{\bar{Z}}) = \{(q_1, \hat{Z}, x)\}$, 11'. $\delta_2(s_r, x, Z_2) = \{(s_{[r]}, Zx, \lambda)\}$,

12. $\delta_1(q_1, x, \hat{Z}) = \{(q_1, \hat{Z}, x)\}$, 12'. $\delta_2(s_{[r]}, x, Z_2) = \{(s_{[r]}, K_1 x, \lambda)\}$,
13. $\delta_1(q_1, p, \hat{Z}) = \{(q_2, K_2, \lambda)\}$, 13'. $\delta_2(s_{[r]}, \lambda, Z) = \{(s_{[r]}, Z, \lambda)\}$,
14. $\delta_1(q_2, \lambda, Z_2) = \{(q_3, \lambda, \lambda)\}$, 14'. $\delta_2(s_{[r]}, \{\bar{p}, r\}, Z) = \{(s_f, \bar{Z}, \lambda)\}$,
15. $\delta_1(q_3, \lambda, x) = \{(q_3, \lambda, x)\}$, 15'. $\delta_2(s_{[r]}, p, Z) = \{(s_{[p]}, \bar{\bar{Z}}, \lambda)\}$,
16. $\delta_1(q_3, x, Z_1) = \{(q_3, Z_1, x)\}$, $x \neq r$, 16'. $\delta_2(s_{[p]}, x, Z_2) = \{(s_{[p]}, Z_2, x)\}$,
17. $\delta_1(q_3, r, Z_1) = \{(q_4, \lambda, \lambda)\}$, 17'. $\delta_2(s_{[p]}, \bar{r}, Z_2) = \{(s_f, K_1, \lambda)\}$,
18. $\delta_1(q_4, x, Z_1) = \{(q_4, Z_1, \lambda)\}$, $x \neq r$, 18'. $\delta_2(s_{[p]}, r, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
19. $\delta_1(q_4, r, Z_1) = \{(q_5, Z_1, \lambda)\}$, 19'. $\delta_2(s_{[\bar{r}]}, x, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
20. $\delta_1(q_5, x, Z_1) = \{(q_f, Z_1, x)\}$, 20'. $\delta_2(s_{[\bar{r}]}, \lambda, Z_2) = \{(s_{[\bar{r}]}, Z_2, \lambda)\}$,
21. $\delta_1(q_f, x, Z_1) = \{(q_f, Z_1, x)\}$, 21'. $\delta_2(s_{[\bar{r}]}, \lambda, Z_2) = \{(s_{[\bar{r}]}, K_1 Z_2, \lambda)\}$.

Briefly, the above system works in two stages. In the first stage the second transducer makes a search, over the whole input string, for the alternating direct repeat pattern of pointers (p, r). During all this time the first transducer reads no symbol, and has no output. If this pattern is not found the first transducer outputs the whole input string on the first output tape. When this pattern is found the second transducer yields the result of the $\mathbf{dlad}_{p,r}$ operation. This will be done in the second stage of the simulation.

In more details, the system works as follows: for the beginning only the second component reads symbols from the input tape until the pointer p is found, without any output. If this pointer is not found, then the (whole) input string will be yielded on the output tape of the first component (due to rules 2', 3, and 4). When the symbol p is reached the current state of the second component is changed into the state s_p (rule 3'). From now on the second component continues the searching of r. If the pointers p or $\bar{p}$ are found before r, then the $\mathbf{dlad}_{p,r}$ operation cannot be performed. Thus the (whole) input string will be outputted by the first pushdown transducer (due to rules 5', 3, and 4). If r is found (before of p or $\bar{p}$), then the state s_p is changed into s_r. Afterward, the string placed between the pointers r and p, i.e., π_3 (see Section 4.1), is memorized in the pushdown memory of the second transducer. This process is performed as follows: in the state s_r the second transducer memories the first symbol from π_3, and changes the state s_r into the state $s_{[r]}$ (due to rule 11'). In this moment the first pushdown transducer is obliged to query (rule 5), the second pushdown transducer not being defined in the state $s_{[r]}$ with the top of pushdown memory set on the pushdown symbol Z. Due to the query process, the content of the pushdown memory of the second transducer will be discharged into the pushdown memory of the first transducer, and the pushdown memory of the second transducer is reduced to Z_2, due to the returning character of the system. In the state $s_{[r]}$, having on the top of pushdown memory the symbol Z_2, the second transducer is able now to memorize the next symbol from π_3 and to ask for the content of the first pushdown memory (rule 12'), in order to build in its memory the image of the string π_3, and not the mirror image of it, as it usually happens due to the structure of the pushdown memory. After the query step is performed, the pushdown memory of the first transducer returns to Z_1, due to the returning character of the system. In the next step the first transducer asks for the content of the pushdown memory of the second transducer (rule 7), in order to give freedom to the second one to memories the next symbol from π_3 and to make the concatenation

between the symbol that has been currently read and the substring of π_3 that had been already read and memorized into the pushdown memory of the first transducer (rule 12'). This process continues until the whole substring π_3 has been read and memorized into the pushdown memory of the second transducer. Afterwards, if in the search procedure of the second transducer the pointer $\bar{p}/r$ is found before the second occurrence of p the $\mathbf{dlad}_{p,r}$ operation cannot be performed. So that the (whole) input string will be outputted by the first pushdown transducer (due to rules 14', 8, 7, and 4). When the second occurrence of p is found, the state $s_{[r]}$ is changed into $s_{[p]}$ and the second transducer starts to yield the image of a possible π_4 (see Section 4.1). If the pointer $\bar{r}$ is found instead of r, then the second transducer queries the first one in order to let it output the whole input string. If the second occurrence of r is found, then an alternating direct repeat pattern of pointers (p, r) exists in the string, so that the result of the $\mathbf{dlad}_{p,r}$ operation will be yielded on the first output tape, according to the rules from 11 to 21.

If we compare the above $rpcpt(2)$ system with the $rpcft(2)$ presented in [2], from a descriptional complexity point of view, we will observe that the former one is more efficient than the latter one. Thus $rpcft(2)$, presented in [2], needs $|\Pi_k| + 7$ states, and $8|\Pi_k| + A_k^2 + 5$ rules for the first component, 9 states and $6|\Pi_k| + 9$ rules for the second component. The number of communications is fixed to 3. The above $rpcpt(2)$ needs 8 states and $8|\Pi_k| + 11$ rules for the first component, 7 states and $9(|\Pi_k| + 1)$ rules for the second one, and $|\pi_3| + 2$ communication steps. Furthermore, the system makes a complete searching for the pointers p and r, and it yields the output for the all situations.

All the observations done along the systems described in this section, argue that PCPTS turns out to be very efficient in implementing complex molecular operations, from a descriptional point of view.

5 Conclusions

In this paper we have introduced a new translating device called Parallel Communicating Pushdown Transducer Systems. They are systems composed of several pushdown transducers working in parallel, in a synchronized manner and communicating with each other by request. We focus on the strategy of communication by stacks, and by the output tapes. The protocol of collaboration is controlled by query symbols. They specify which component has to send the content of the pushdown memory and the content of the output tape, and which component has to receive them. The strategy of data exchanging through the pushdown memory and through the output tape allows each component to output complex information. Therefore the complexity of the yielded languages is substantially increased. The computational power of the new device has been investigated by taking into consideration the computational power of parallel communicating pushdown automata systems. A comparison between them and parallel communicating finite transducer systems has been done, too. Even if they are able to yield at least the same classes of languages as parallel communicating finite transducer systems do, they are more efficient from a descriptional complexity point of view.

References

[1] A. O. Buda. Multiprocessor Automata, *Information Processing Letters*, 25:257-261, 1977.

[2] L. Cojocaru. Simulating the Process of Gene Assembly in Ciliates. In *Proceedings of the* 9^{th} *International Conference on Implementation and Application of Automata*, Queen's University, Kingston, Ontario, Canada, July 22-24, 2004.

[3] E. Csuhaj-Varjú, C. Martín-Vide, and V. Mitrana. Parallel Communicating Finite Transducer Systems. In *Language and Computers: Studies in Practical Linguistics*, 47, 9-23, 2002.

[4] E. Csuhaj-Varjú, C. Martin-Vide, V. Mitrana, and G. Vaszil. Parallel Communicating Pushdown Automata Systems. *International Journal of Foundations of Computer Science* 11, No. 4, 633-650, 2000.

[5] J. Dassow and V. Mitrana. Stack Cooperation in Multi-Stack Pushdown Automata, *Journal of Computer and System Sciences* 58, 611-621, 1999.

[6] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, and G. Rozenberg. *Computation in Living Cells, Gene Assembly in Ciliates.* Springer-Verlag, 2004.

[7] A. Ehrenfeucht, I. Petre, D.M. Prescott, and G. Rozenberg. Universal and Simple Operations for Gene Assembly in Ciliates. In *Where Mathematics, Computer Science, Linguistics and Biology Meet*, C.Martin-Vide, V. Mitrana (eds.), 329-342, Kluwer Academic Publishers, Dordrecht/Boston, 2001.

[8] A. Ehrenfeucht, D.M. Prescott, and G. Rozenberg. Computational Aspects of Gene (Un)scrambling in Ciliates. In *Evolution as Computation*, L. Landweber, E. Winfree (eds.), 45-86, Springer Verlag, Berlin, Heidelberg, 2001.

[9] R. J. Evey. The Theory and Application of Pushdown Store Machines, Ph.D. Thesis and Research Report, *Mathematics, Linguistics and Automata.* Trans. Project, Harvard University, NSF-10, May 1963.

[10] P. C. Fischer. On Computability by Certain Classes of Restricted Turing machines, In *Proc. Fourth IEEE Symp. on Switch. Circuit Theory and Logical Design*, 1963.

[11] S. Ginsburg. Example of Abstract Machines, *IEEE Trans. on Electronic Computers* 111: 2, 132-135, 1962.

[12] S. Ginsburg and G. F. Rose. Preservation of Languages by Transducers, *Information and Control* 9:2, 153-176, 1966.

[13] O. H. Ibarra. One Two-Way Multihead Automata, *J. Comput. System Sci.* 7, 28-36, 1973.

[14] R. M. Kaplan and M. Kay. Regular Models of Phonological Rule Systems, *Computational Linguistics*, 20(3), 1994.

[15] L. Karttunen. Finite-State Lexicon Compiler. *Technical Report Xerox PARC* P93-00077, Xerox PARC, 1993.

[16] W. Kuich and A. Salomaa. Semirings, Automata, Languages, *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, Germany, 1986.

[17] C. Martín-Vide and V. Mitrana. Parallel Communicating Automata Systems, A Survey. *Korean Journal of Computational Applied Mathematics*, Vol. 7, No. 2, 237-257, 2000.

[18] C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel Finite Automata Systems Communicating by States. *International Journal of Foundation of Computer Science*, Vol. 13 No. 5, 733-749, 2002.

[19] V. Mitrana. On the Degree of Communication in Parallel Communicating Finite Automata Systems. Descriptional Complexity of Automata, Grammars and Related Structures (Magdeburg, 1999). *Journal of Automata Language and Comb.* Vol. 5, No. 3, 301-314, 1999.

[20] M. Mohri. Finite-State Transducers in Language and Speech Processing, *Computational Linguistics*, 23(2), 269-311, 1997.

[21] M. Mohri. Minimization Algorithms for Sequential Transducers, *Theoretical Computer Science*, 234: 177201, 2000.

[22] M. Mohri, F. Pereira, and M. Riley. Weighted Automata in Text and Speech Processing, In *ECAI-96 Workshop*, Budapest, Hungary, 1996.

[23] G. Păun, G. Rozenberg, and A. Salomaa. DNA Computing: new Computing Paradigms, Springer-Verlag, Berlin, Germany, 1998.

[24] D. M. Prescott, A. Ehrenfeucht, and G. Rozenberg. Molecular Operations for Processing in Hypotrichous Ciliates. *European Journal of Protislology*, 2001.

[25] E. Roche and Y. Schabes. *Finite-State Language Processing.* MIT Press, Cambridge, 1997.

[26] G. Rozenberg. Gene Assembly in Ciliates: Computing by Folding and Recombination. In *A half-century of automata theory: celebration and inspiration*, 93-130. World Scientific Publishing, 2001.

[27] M. Sakthi Balan, K. Krithivasan, and M. Madhu. Some Variants in Communication of Parallel Communicating Pushdown Automata. *Journal of Automata, Languages and Combinatorics* 8, 3:401-406, 2003.