Parallel Communicating Finite Transducer Systems

Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, and Victor Mitrana

Computer and Automation Research Institute, Hungarian Academy of Sciences
Research Group in Mathematical Linguistics, Rovira i Virgili University
Faculty of Mathematics and Computer Science, University of Bucharest

Abstract

A parallel communicating finite transducer system is a translating device where several finite transducers work in parallel, in a synchronized manner, and communicate with each other by requests. The communicated data are the current state of the transducer and the current contents of its output tape. Each computation step in such a system is either a usual translating step or a communication step; moreover, the communication steps have priority over the translating ones. Furthermore, whenever a component requests some data, that data must be communicated. We investigate the computational power of these systems. Then we consider systems restricted to subsequential transducers, as components, and compare these systems with the general ones. These devices turned out to be useful in computational linguistics. A short discussion on a possible relevance in the theory of discourse parsing and some directions for further work closes the paper.

1 Introduction

In many areas of computer science (parallel and distributed computing, computer networks, natural language processing, artificial intelligence - problem solving, human computer interaction, expert systems, computer supported cooperative work, and recently in DNA computing and quantum computing) various models based on cooperation and communication among agents have been considered. Many of these areas are multidisciplinary being placed at various crossroads of mathematics, computer science, biology, computational linguistics, and cognitive psychology. Formal language theory has been involved in most of these circumstances e.g. in modelling aspects whose essence can be captured at the level of abstract symbol systems (Csuhaj-Varjú et al. 1994, Durfee et al. 1989, Păun 1998). Thus, Csuhaj-Varjú and Dassow (1990) introduced *cooperating distributed (CD) grammar systems*, inspired by the so-called "blackboard model" in problem solving theory (Durfee et al. 1989). Several grammars working together, following a prescribed strategy form a grammar system. Two essentially different architectures, depending on the protocols of cooperation and communication among the components, have been studied during the years (see e.g. Csuhaj-Varjú et al. 1994). In the case of *cooperating distributed grammar systems* the cooperation is done by means of the sentential form; components may rewrite, in turn, the sentential form according to their own strategies. When a component is active, all the others are inactive. Quite different is the cooperation in *parallel communicating (PC) grammar systems*, where the components work in parallel on their own sentential forms, and from time to time some components ask, by means of query symbols, for the work of other ones. The contacted components have to send their current work

(sentential form) to those components which asked for it.

An entire theory has been developed for both types of grammar systems (see the monograph (Csuhaj-Varjú et al. 1994) and more recently the chapter (Dassow et al. 1997) in (Rozenberg and Salomaa 1997)). Connections between grammar systems and natural language processing and modelling were presented in (Csuhaj-Varjú 1994, 2000, Csuhaj-Varjú et al. 1999) and in (Jiménez-López 2000).

The idea of considering several automata which cooperate in the aim of recognizing a word, following different strategies, can be found in many papers though it is not explicitly asserted. We mention here some of them: *multi-head automata* (Ibarra 1973), *multiprocessor automata* (Buda 1987), *parallel communicating automata systems* (Csuhaj-Varjú et al. 2000, Martín-Vide et al. 2002, Martín-Vide and Mitrana 2001), or *cooperating multi-stack pushdown automata* (Dassow and Mitrana 1999).

In Martín-Vide et al. (2002) systems of finite automata work in parallel on the same input tape and communicate with each other by states, in order to recognize the word placed on the common input tape. These systems have components which communicate with each other under similar protocols to those considered for parallel communicating grammar systems mentioned above. Every component is entitled to request the state of any other component; the contacted component communicates its current state and either remains in the same state (in the case of the non-returning strategy) or enters again the initial state (in the case of the returning strategy). In centralized systems only one component (the master of the system) is allowed to ask a state from the others. We want to stress that each step in an automata system is either a usual accepting step or a communication step; moreover, the communication steps have priority to the accepting ones. We also mention that whenever a component requests a state, the state must be communicated.

In this paper, we propose a new approach by extending the concepts of parallelism and communication from the grammar systems area to systems of finite transducers.

Sequential string-to-string transducers have been successfully used in the representation of large-scale dictionaries, computational morphology, and local grammar and syntax (Mohri 1996, 1997b, 1997a). The reader may also consult (Krauwer and des Tombe 1981). Some algebraic aspects related to finite transducers have been reported in (Choffrut 1977, Reutenauer 1993, Schützenberger 1977).

There are a few approaches for constructing a finite transducer which has the required properties for being used in some applications. A nice and comprehensive picture of the state of the art in using finite-state methods in language processing is the collective volume edited by Roche and Schabes (1997). We present two of them very briefly and informally. One consists of building a series of cascaded transducers. The output of a transducer is feed in the input of the next transducer, i.e., to write a series of transducers from the most general one to the most specific. This approach is based to the closure of finite transducers under composition. However, composition of finite transducers generally outputs a transducer with a large number of states which might be hardly manipulated, in spite of

the fact that in many applications one needs not expand a cascade of transducers (Kempe 2000, Abney 1996).

Another way of building a desired transducer consists of approximating iteratively through a sequence of intersections. This approach successively approximates the desired transducer by constructing a transducer for each specific phenomenon of the problem which is to be solved and then intersects all these transducers such that a common behavior is achieved. However, the most general variants of transducers are not closed under intersection. Actually, approximation techniques have been an important topic in natural language processing in recent times (Johnson 1998, Evans 1997, Mohri and Nederhof 2001).

The new model we propose in this paper is based on a different view to computation, that is, it makes use of cooperation and communication. A parallel communicating finite transducer system is a translating device based on communication between finite transducers working in parallel. It consists of several finite transducers working independently but communicating with each other by request. The strategy of cooperation of finite automata systems is slightly modified for finite transducer systems: the contents of the output tape is communicated together with the current state and appended to the contents of the receiver output tape.

We expect to achieve two main goals: to increase the computational power of the components by cooperation and communication and to decrease the complexity of the different tasks by distribution and parallelism.

The achieved results demonstrate that cooperation and communication considerably enhance the computational power, parallel communicating finite transducer systems, even with a very small number of components and a simple input language over an alphabet of not more than four letters, are able to identify any recursively language. Thus, the translation process can be made significantly more effective with the help of distribution and communication. For example, some well-known mildly context-sensitive languages can be obtained with these tools, even if we use only one-letter alphabets, that is, a signal. This means that some important non-context-free structures of natural languages can be effectively handled by these very restricted distributed translating devices.

2 Preliminaries

We assume the reader is familiar with the basic concepts of formal language theory and automata theory, in particular with the notions of grammars and finite automata (Rozenberg and Salomaa 1997).

An alphabet is a finite nonempty set of symbols. The set of all words over an alphabet V is denoted by V^*. The empty word is written as ε; and, $V^+ = V^* - \{\varepsilon\}$. Sometimes, for a given alphabet V and a word $x = a_1 a_2 \dots a_n$, $a_i \in V, 1 \leq i \leq n$, we write $\bar{V} = \{\bar{a} | a \in V\}$ and $\bar{x} = \bar{a_1}\bar{a_2}\dots\bar{a_n}$. Here $\bar{V}$ is a disjoint copy of V.

A parallel communicating finite transducer system (a PCFTS, in short) of size n, where $n \geq 1$, is a construction

$$\mathcal{A} = (V, U, A_1, A_2, \dots, A_n, K),$$

where V and U are the input and the output alphabets, respectively, and

- $A_i = (Q_i, V, U, f_i, q_i, F_i), 1 \leq i \leq n$, are finite transducers with the set of states Q_i, $q_i \in Q_i$ (the initial state of transducer A_i), $F_i \subseteq Q_i$ (the set of final states of A_i), and f_i is the transition-and-output mapping of transducer A_i from $Q_i \times (V \cup \{\varepsilon\})$ to finite subsets of $Q_i \times U^*$. Notice that Q_i, $1 \leq i \leq n$ are not necessarily disjoint sets. If one discards the output alphabet U, and the mapping f_i is restricted to a mapping from $Q_i \times (V \cup \{\varepsilon\})$ to finite subsets of Q_i, then one obtains a finite automaton which will be termed as the underlying finite automaton of A_i. If f_i is a function from $Q_i \times V$ to finite subsets of $Q_i \times U^*$, i.e., A_i reads exactly one symbol at each transition, then A_i is said to be a *sequential transducer* (in other works these devices are called *generalized sequential machines*, see e.g. (Eilenberg 1974)).
- $K = \{K_1, K_2, \ldots, K_n\} \subseteq \bigcup_{i=1}^{n} Q_i$ is the set of query states. K_i, for $1 \leq i \leq n$, is the query symbol pointing to A_i in $\mathcal{A}$.

The finite transducers $A_1, A_2, \ldots, A_n$ are called the *components* of the system $\mathcal{A}$. We refer to A_i as the ith component or component i of $\mathcal{A}$, $1 \leq i \leq n$. If there exists just one $1 \leq i \leq n$ such that $K \subseteq Q_i$, then the system is said to be *centralized*, the master of this system being the component i. For the sake of simplicity, whenever a system is centralized, the first component is its master. If the following conditions:

$$\begin{aligned} &(i) && card(f_i(s,a)) \leq 1 \text{ for all } s \in Q_i \text{ and } a \in V \cup \{\varepsilon\}, \\ &(ii) && \text{if } card(f_i(s,\varepsilon)) \neq 0 \text{ for some } s \in Q_i, \text{ then} \\ & && card(f_i(s,a)) = 0 \text{ for all } a \in V, \end{aligned}$$

hold for all $1 \leq i \leq n$, then the system is said to be *deterministic*.

Given a PCFTS $\mathcal{A}$, the system consisting in the underlying finite automata of all components is called the underlying parallel communicating finite automata system (*pcfas*, in short) of $\mathcal{A}$.

By a configuration of a parallel communicating finite transducer system, as above, we mean an $3n$-tuple

$$(s_1, x_1, y_1, s_2, x_2, y_2, \ldots, s_n, x_n, y_n)$$

where

- $s_i \in Q_i$ is the current state of the component i,
- $x_i \in V^*$ is the remaining part of the input word which has not been read yet by the component i,
- $y_i \in U^*$ is the contents of the output tape of the component i, $1 \leq i \leq n$.

We define two binary relations on the set of all configurations of $\mathcal{A}$ in the following way:

$$(s_1, x_1, y_1, s_2, x_2, y_2, \ldots, s_n, x_n, y_n) \vdash (s'_1, x'_1, y'_1, s'_2, x'_2, y'_2, \ldots, s'_n, x'_n, y'_n)$$

iff one of the following two conditions holds:

(i) $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ and
$x_i = a_i x_i', a_i \in V \cup \{\varepsilon\}$, $y_i' = y_i z_i$, $(s_i', z_i) \in f_i(s_i, a_i)$, $1 \leq i \leq n$

(ii) for all $1 \leq i \leq n$ such that $s_i = K_{j_i}$ and $s_{j_i} \notin K$ we put $s_i' = s_{j_i}$, and $y_i' = y_i y_{j_i}$, $s_r' = s_r$, and $y_r' = y_r$, for all the other $1 \leq r \leq n$, and $x_t' = x_t$, $1 \leq t \leq n$,

and

$$(s_1, x_1, y_1, s_2, x_2, y_2, \ldots, s_n, x_n, y_n) \vdash_r (s_1', x_1', y_1', s_2', x_2', y_2', \ldots, s_n', x_n', y_n')$$

iff one of the following two conditions holds:

(i) $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ and
$x_i = a_i x_i', a_i \in V \cup \{\varepsilon\}$, $y_i' = y_i z_i$, $(s_i', z_i) \in f_i(s_i, a_i)$, $1 \leq i \leq n$

(ii) for all $1 \leq i \leq n$ such that $s_i = K_{j_i}$ and $s_{j_i} \notin K$ we put $s_i' = s_{j_i}$, $s_{j_i}' = q_{j_i}$, $y_i' = y_i y_{j_i}$, and $y_{j_i}' = \varepsilon$, $s_r' = s_r$, and $y_r' = y_r$, for all the other $1 \leq r \leq n$, and $x_t' = x_t$, $1 \leq t \leq n$.

The difference between the two relations defined above can easily be noticed when the current states of some components are query states: these components enter into communication with those components which are identified by the appearing query states. The component identified by the query state is forced to send its current state to the requesting one, supposing that it is not a query state, and this state becomes the new current state of the receiver component. Moreover, this contacted component sends also the contents of its output tape which is appended to the contents of the output tape of the receiver. The new current state and the contents of the output tape of the sender component remains the same in the case of relation $\vdash$ whereas it becomes the initial state and empty string when relation $\vdash_r$ has been applied. In other words, in the returning case, the sender components restart their work from the current input symbol.

A PCFTS with moves based only on the relation $\vdash_r$ is said to be *returning*, while a PCFTS with moves based only on the relation $\vdash$ is called *non-returning*.

In the following we shall denote the reflexive and transitive closure of relations $\vdash$ and $\vdash_r$ by $\vdash^*$ and $\vdash_r^*$, respectively.

Remember that if A is a finite transducer, then it defines a transduction function T_A which returns for each input word x the set of all words which occurred on the output tape of A at the end of a successful computation on x. If there is no such successful computation on x, then $T_A(x) = \emptyset$.

Now, each PCTFS $\mathcal{A}$ of size n as above defines n transduction functions. Informally, the transduction of x by the ith component of $\mathcal{A}$ consists of all strings $y_i \in U^*$ such that the system starts in an initial configuration $(q_1, x, \varepsilon, q_2, x, \varepsilon, \ldots, q_n, x, \varepsilon)$ and reaches a final configuration, that is, a configuration of the form $(s_1, \varepsilon, y_1, s_2, \varepsilon, y_2, \ldots, s_n, \varepsilon, y_n)$ with $s_i \in F_i$.

Formally, if $\mathcal{A}$ is a non-returning PCTFS, then

$$\begin{aligned} T_{\mathcal{A}}^{(i)}(x) &= \{y_i \in U^* \mid (q_1, x, \varepsilon, q_2, x, \varepsilon, \ldots, q_n, x, \varepsilon) \vdash^* \\ &\quad (s_1, \varepsilon, y_1, s_2, \varepsilon, y_2, \ldots, s_n, \varepsilon, y_n), s_j \in F_j, 1 \leq j \leq n\} \\ &\quad \text{for any } 1 \leq i \leq n. \end{aligned}$$

If if $\mathcal{A}$ is a returning PCTFS, then

$$\begin{aligned} TR_{\mathcal{A}}^{(i)}(x) &= \{y_i \in U^* \mid (q_1, x, \varepsilon, q_2, x, \varepsilon, \ldots, q_n, x, \varepsilon) \vdash_r^* \\ &\quad (s_1, \varepsilon, y_1, s_2, \varepsilon, y_2, \ldots, s_n, \varepsilon, y_n), s_j \in F_j, 1 \leq j \leq n\} \\ &\quad \text{for any } 1 \leq i \leq n. \end{aligned}$$

Note that because of the possible communication, $T_{\mathcal{A}}^{(i)}$ is not necessarily equal to T_{A_i}. The above transduction mapping can be extended to a language $L \subseteq V^*$ by

$$T_{\mathcal{A}}^{(i)}(L) = \bigcup_{x \in L} T_{\mathcal{A}}^{(i)}(x), \qquad TR_{\mathcal{A}}^{(i)}(L) = \bigcup_{x \in L} TR_{\mathcal{A}}^{(i)}(x).$$

Furthermore, we define the

$$\begin{aligned} \textit{transduction system mapping: } T_{\mathcal{A}} &= (T_{\mathcal{A}}^{(1)}, T_{\mathcal{A}}^{(2)}, \ldots, T_{\mathcal{A}}^{(n)}), \\ TR_{\mathcal{A}} &= (TR_{\mathcal{A}}^{(1)}, TR_{\mathcal{A}}^{(2)}, \ldots, TR_{\mathcal{A}}^{(n)}). \end{aligned}$$

We shall denote by:

- $RCPCFTS(n)$ the class of all returning centralized parallel communicating finite transducer systems of size n;
- $RPCFTS(n)$ the class of all returning parallel communicating finite transducer systems of size n;
- $CPCFTS(n)$ the class of all non-returning centralized parallel communicating finite transducer systems of size n;
- $PCFTS(n)$ the class of all non-returning parallel communicating finite transducer systems of size n.

Clearly, $RCPCFTS(n) \subseteq RPCFTS(n)$ and $CPCFTS(n) \subseteq PCFTS(n)$ for any $n \geq 1$. We add the prefix D to the notation in order to denote deterministic variants and replace F by S when we refer to parallel communicating sequential transducer systems.

We present an example for the above notions.

Example 1 *Let* $\mathcal{A} = (\{x\}, \{a, b, c\}, A_1, A_2, A_3, \{K_1, K_2, K_3\})$, *be a non-returning and non-centralized PCFTS where*

$$\begin{aligned} A_1 &= (\{q_1, K_2, K_3, r\}, \{x\}, \{a, b, c\}, f_1, q_1, \{r\}), \\ A_2 &= (\{q_2, K_3, r\}, \{x\}, \{b\}, f_2, q_2, \{r\}), \\ A_3 &= (\{q_3, r\}, \{x\}, \{c\}, f_3, q_3, \{r\}), \end{aligned}$$

with the following transition-and-output functions

$$f_1(q_1,\varepsilon) = \{(q_1,a)\} \quad f_2(q_2,\varepsilon) = \{(q_2,b)\} \quad f_3(q_3,\varepsilon) = \{(q_3,c)\},$$
$$f_1(q_1,x) = \{(K_2,\varepsilon)\} \quad f_2(q_2,x) = \{(K_3,\varepsilon)\} \quad f_3(q_3,x) = \{(r,\varepsilon)\}.$$

It is easy to see that $T_{\mathcal{A}}^{(1)}(\{x\}) = \{a^n b^n c^n \mid n \geq 1\}$. *That is, a parallel communicating finite transducer system is able to compute a non-context-free language by reading an input consisting of a symbol only.*

3 Computational Power

Parallel communicating finite transducer systems turn out to be powerful computational devices. Among other things, it can be shown that these systems, even with a very small number of components and with relatively simple input languages over a four-letter alphabet, are are able to determine any recursively enumerable language.

In the sequel, we define two operations on words and languages useful in our considerations concerning the computational power of PCFTSs. A homomorphism which erases some symbols and leaves unchanged the others is said to be a *projection*. For two disjoint alphabets V and V', mapping $h : (V \cup V')^* \longrightarrow V^*$ is a projection, since it erases the symbols from V'. We denote this mapping by pr_V. The other operation is a well-known operation in formal language theory and in parallel programming theory, called the *shuffle* operation. A shuffle of two strings is an arbitrary interleaving of the substrings of the original strings, like shuffling two decks of cards. More precisely, for two strings $x, y \in V^*$ and two symbols $a, b \in V$,

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

For two languages L_1, L_2 we define $L_1 \sqcup\!\sqcup L_2 = \bigcup_{x \in L_1, y \in L_2} x \sqcup\!\sqcup y$.

Consider an alphabet V and its barred copy $\bar{V} = \{\bar{a} \mid a \in V\}$. The *twin shuffle* language over V is defined by

$$TS(V) = \bigcup_{x \in V^*} x \sqcup\!\sqcup \bar{x},$$

where each word $\bar{x}$ is obtained from x by replacing each letter from V by its barred copy from $\bar{V}$.

The following representation of recursively enumerable languages is well known (Engelfriet and Rozenberg 1980):

Theorem 1 *Each recursively enumerable language* $L \subseteq T^*$ *can be written as* $L = pr_T(TS(V) \cap R)$, *where* V *is an alphabet including* T *and* R *is a regular language.*

Based on this result and the proof of Theorem 4 in Martín-Vide et al. (2002) one can immediately infer the next theorem.

Theorem 2 *Let L be a recursively enumerable language over an alphabet T. Then, there exists an alphabet V including T such that the following statements hold:*

1. *There exists $\mathcal{A} \in RCPCFTS(3)$ such that $TR_{\mathcal{A}}(V^*) = (L, \{\varepsilon\}, \{\varepsilon\})$.*
2. *There exists $\mathcal{A} \in CPCFTS(3)$ such that $T_{\mathcal{A}}(V^*) = (L, \{\varepsilon\}, \{\varepsilon\})$.*

Proof. In the proof of Theorem 4 in Martín-Vide et al. (2002) for an arbitrary alphabet V and an arbitrary regular language R one gives a returning and centralized *pcfas* and a centralized *pcfas*, both with three components, each of them accepting the language $TS(V) \cap R$. Now, it suffices to transform the master component into a finite transducer such that at each step it writes the current symbol, if that symbol is in T, and the empty word, otherwise, and to transform the other components in finite transducers which write always the empty word. ■

We do not know whether or not two components suffice in any of the variants. This remains an open problem.

In this representation, all the components of the system depend on the language L since both V and R depend on it. This can be avoided by making use of the following representation of recursively enumerable languages which can be easily derived from the previous one. We denote by $\mathbf{B}$ the alphabet consisting of symbols 0 and 1.

Theorem 3 *For each recursively enumerable language L there exists a finite transducer A_L such that $L = T_{A_L}(TS(\mathbf{B}))$.*

This representation allows us to give the following characterizations of recursively enumerable languages in terms of PCFTSs which have only one component that depends on the given language.

Theorem 4 *Let L be a recursively enumerable language over an alphabet V. Then, there exists $\mathcal{A} \in RCPCFTS(3)$ such that $TR_{\mathcal{A}}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = (L, \{\varepsilon\}, \{\varepsilon\})$.*

Proof. Let $A = (Q, \mathbf{B} \cup \bar{\mathbf{B}}, V, \delta, s_0, F)$ be a finite transducer such that $L = T_A(TS(\mathbf{B}))$. We define the returning and centralized PCFTS

$$\mathcal{A} = (\mathbf{B} \cup \bar{\mathbf{B}}, A_1, A_2, A_3, \{K_1, K_2, K_3\}),$$

where

$$\begin{aligned} A_1 &= (\{q_1, K_2, K_3, r_0, r_1\} \cup F, \mathbf{B} \cup \bar{\mathbf{B}}, V, f_1, q_1, F), \\ A_2 &= (\{q_2, r_0, r_1\}, \mathbf{B} \cup \bar{\mathbf{B}}, V, f_2, q_2, \{q_2\}), \\ A_3 &= (Q, \mathbf{B} \cup \bar{\mathbf{B}}, V, f_3, s_0, \{s_0\}), \end{aligned}$$

with the transition-and-output functions defined as follows

$$\begin{array}{ll} f_1(q_1,\varepsilon) = \{(q_1,\varepsilon),(K_2,\varepsilon)\} & f_2(q_2,\varepsilon) = \{(q_2,\varepsilon)\} \\ f_1(q_1,\bar{a}) = \{(q_1,\varepsilon),(K_2,\varepsilon)\},\ a \in \mathbf{B} & f_2(q_2,a) = \{(q_2,\varepsilon)\},\ a \in \mathbf{B} \\ f_1(r_a,a) = \{(q_1,\varepsilon)\},\ a \in \mathbf{B} & f_2(q_2,\bar{a}) = \{(r_a,\varepsilon)\},\ a \in \mathbf{B} \\ f_1(q_2,\varepsilon) = \{(K_3,\varepsilon)\} & \end{array}$$

$$\begin{aligned} f_3(s,a) &= \delta(s,a),\ s \in Q, a \in \mathbf{B} \cup \bar{\mathbf{B}} \\ f_3(s,\varepsilon) &= \delta(s,\varepsilon) \cup \{(s,\varepsilon)\},\ s \in Q. \end{aligned}$$

It is easy to observe that the *pcfas* consisting of the underlying finite automata of the first two components of $\mathcal{A}$ accepts a word x over $\mathbf{B} \cup \bar{\mathbf{B}}$ if and only if the word obtained from x by erasing all symbols $0, 1$ is the barred copy of the word obtained from x by removing all symbols $\bar{0}, \bar{1}$. When this task was successfully accomplished, then the first component is in q_1 and has ε on its output tape. Now, the first component, the master, checks whether the second component is in its final state, that is, in q_2, and if this is the case, then it checks whether the third component is in a state from F. The current state as well as the contents of the output tape of this component are sent to the master. Since the set of final states of the first component is F, we infer that the computation is successful if and only if all these checking processes finish successfully. Therefore, $TR_{\mathcal{A}}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = (L, \{\varepsilon\}, \{\varepsilon\})$. ■

Theorem 5 *Let L be a recursively enumerable language over an alphabet V. Then, there exists $\mathcal{A} \in CPCFTS(4)$ such that $T_{\mathcal{A}}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = (\{\varepsilon\}, \{\varepsilon\}, \{\varepsilon\}, L)$.*

Proof. We give an informal proof. First, a *cpcfas* of size 3 which accepts the language $TS(\mathbf{B})$ is constructed as follows. The first two components check whether or not the input string is in $x \sqcup\!\sqcup \bar{y}$, where x is a scattered substring (subsequence) of y whereas the first and the third component checks whether or not the input string is in $x \sqcup\!\sqcup \bar{y}$, where y is a scattered substring of x. Moreover, whenever, a component enters a final state, it stops reading since the transition mapping is not defined for final states. The construction in the second case of the proof of Theorem 4 from Martín-Vide et al. (2002) can easily be modified to satisfy the above requirements. Now, we transform these finite automata into finite transducers which write always the empty word on the output tape and add the third component of the PCFTS from the previous proof without changing the set of final states. In spite of the fact that this new component works independently from the other ones and does not exchange any information with them, due to the system synchronization it follows that whenever the input word is from $TS(\mathbf{B})$ and the last component is in a final state, the contents of the output tape of the last component is a word in L and vice versa. ■

It is easy to modify the centralized PCFTS from the proof above such that $T_{\mathcal{A}}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = (L, \{\varepsilon\}, \{\varepsilon\}, L)$. On the other hand, the second or the third component (or both) of the above system can be modified in the aim of simulating both the work of the automaton and the work of the last transducer. This can be done here because the system is non-returning so that the current state of the modified component can encode also the current state of the fourth component. Thus, after removing the fourth component we obtain a centralized PCFTS with three components $\mathcal{A}'$ such that

$$T_{\mathcal{A}'}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = \begin{cases} (L_1, L, \{\varepsilon\}) \text{ for some language } L_1, \\ \text{if the second component was modified} \\ (L_2, \{\varepsilon\}, L) \text{ for some language } L_2, \\ \text{if the third component was modified} \\ (L_3, L, L) \text{ for some language } L_3, \\ \text{if both components were modified} \end{cases}$$

It is known (Martín-Vide et al. 2002) that $TS(\mathbf{B})$ can be accepted by a *pcfas* (non-returning, non-centralized) with two components only. Therefore, we have

Theorem 6 *Let L be a recursively enumerable language over an alphabet V. Then, there exists $\mathcal{A} \in PCFTS(3)$ such that $T_{\mathcal{A}}((\mathbf{B} \cup \bar{\mathbf{B}})^*) = (\{\varepsilon\}, \{\varepsilon\}, L)$.*

The last three results, unexpected in a certain sense, might have some biological relevance. In this respect, we remember that genetic information of any organism is encoded in DNA molecules which are sequences formed by four nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T). Furthermore, these four nucleotides are grouped in two pairs of Watson-Crick complementary bases (A,T) and (C,G). Therefore, one may consider any word in $(\mathbf{B} \cup \bar{\mathbf{B}})^+$ as encoding of a DNA molecule in a very natural way (see also Păun 1998). In this view, the last three results may be interpreted in the following way: Any recursively enumerable language is nothing else than the translation of the set of all genoms via a parallel communicating finite transducer system. These observations lead to the following interesting questions. How to interpret the well-known mildly context-sensitive structures in natural languages in this frame, that is, how to describe them as translations of genom sequences and which are the properties of the genoms convenient for this purpose.

Finally, a natural question arises, namely: Do the above results remain valid for parallel communicating finite transducer systems with a smaller number of components?

3.1 One-Letter Input Alphabet

So far we have examined the power of parallel communicating finite transducer systems with input alphabets having at least four letters. A natural question is,

whether or not by using alphabets with fewer symbols, especially one-letter input alphabet, we are able to reach remarkable computational power. If the input alphabet consists of only one letter, then the input string can be considered as a sequence of a signal, and in this respect, only the length of the string is important. We shall see that even with one-letter input alphabet, these systems demonstrate considerable computational power.

When the input alphabet V consists of one letter, one can easily prove that each language generated by a parallel communicating grammar systems with right-linear components (Csuhaj-Varjú et al. 1994) is the transduction of V^+ done by the first component of a PCFTS.

A parallel communicating grammar system of size $n \geq 1$ is a construct

$$\gamma = (N, K, T, (S_1, P_1), (S_2, P_2), \ldots, (S_n, P_n)),$$

where N, T are two disjoint alphabets, S_i, $1 \leq i \leq n$ are the axioms of the components of γ, P_i, $1 \leq i \leq n$, are finite sets of production rules over $N \cup T$, and $K = \{Q_1, Q_2, \ldots, Q_n\}$ is the set of *query symbols*; their indices point to the components of γ. Moreover, N, T, K are mutually disjoint.

Now we give the informal definition of the functioning of the *centralized* and *returning* (to axioms) variants of parallel communicating grammar systems.

For two n-tuples $(x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n), x_1, y_1 \in (N \cup K \cup T)^*$, $x_i, y_i \in (N \cup T)^*$, $2 \leq i \leq n$, the derivation in a parallel communicating grammar system as above is defined as follows:

$$(x_1, x_2, \ldots, x_n) \Longrightarrow_\gamma (y_1, y_2, \ldots, y_n)$$

if one of the next two cases holds:

- no query symbol appears in x_1, and then we have a component-wise derivation, $x_i \Longrightarrow_{P_i} y_i, 1 \leq i \leq n$, except in the case when $x_i \in T^*$ and then $y_i = x_i$;
- in the case of query symbols appearing, a communication step is performed as these symbols impose: each occurrence of Q_j in x_1 is replaced by x_j, supposing that x_j does not contain any query symbol, and, after that, the j^{th} component resumes working from its axiom. Moreover, the communication has priority over the effective rewriting.

To avoid the many technical details, we have preferred this informal definition instead of a formal one. The reader interested in the formal definition can consult (Csuhaj-Varjú et al. 1994). Following the definition of the parallel communicating finite transducer systems, the reader can easily infer the definitions of *non-centralized* or/and *non-returning* variants.

The language generated by a system γ as above is

$$L(\gamma) = \{x \in T^* | (S_1, S_2, \ldots, S_n) \Longrightarrow^* (x, \alpha_2, \ldots, \alpha_n), \alpha_i \in (N \cup T)^*, 2 \leq i \leq n\}.$$

As we have stated above, each language generated by a parallel communicating grammar systems with right-linear components (Csuhaj-Varjú et al. 1994) is the transduction of a^+ done by the first component of a PCFTS. Furthermore, the input alphabet can be reduced to a singleton set formed by a string of length one in the case of non-centralized variants. We do not know whether or not there are PCFTSs which can translate a^+ in languages which cannot be generated by parallel communicating grammar systems with right-linear components. However, if the components are sequential transducers, we have the next result:

Theorem 7 *1. A language L is generated by a returning parallel communicating grammar system γ with n right-linear components if and only if there exists a returning PCSTS of size n, $\mathcal{A}$, such that $TR_{\mathcal{A}}(a^+) = (L, L_2, L_3, \ldots, L_n)$ holds for some languages L_i, $2 \leq i \leq n$. Moreover, $\mathcal{A}$ is centralized if γ is.*

2. A language L is generated by a non-returning parallel communicating grammar system γ with n right-linear components if and only if there exists a non-returning PCSTS of size n, $\mathcal{A}$, such that $T_{\mathcal{A}}(a^+) = (L, L_2, L_3, \ldots, L_n)$ holds for some languages L_i, $2 \leq i \leq n$. Moreover, $\mathcal{A}$ is centralized if γ is.

By this result, it can be shown that certain non-context-free constructions which appear in both natural and programming languages (Gazdar and Pullum 1985, Joshi 1985) can be obtained as a translation of the universal language over the one-letter alphabet via a PCSTS. Example 1 in Section 2 is one of these structures. This result presents a connection between parallel communicating generative devices and translating devices. A comparison of the size complexity of the above two types of computational tools would be of interest.

4 Conclusions and Further Work

We start this last section with a brief discussion on a possible relevance of the parallel communicating finite transducer systems introduced in this paper in the theory of discourse parsing. It is known that text is not just a simple sequence of clauses and sentences, but it rather follows a highly elaborated structure. Most of the current natural language processing systems process an unrestricted text on a sentence-by-sentence basis. One can easily imagine two sequences of sentences which differ from each other in the order of the sentences, only, such that most of the natural language processing systems derive in both cases syntactic trees and construct semantic representations for each of the individual sentences without noticing any anomalies. However, only one sequence is fully acceptable. If we would like to build a proficient natural language processing system, it seems, therefore, obvious that we need to enable these systems to derive inferences that pertain not only to the intra-sentential level, but to the extra-sentential level as well. A transducer system which outputs trees appears to be a good candidate for such a task. Each component might be used for deriving intra-sentential inferences while the communication among them might enable the inference derivation to the extra-sentential level.

Parallel communicating finite transducer systems provide more interesting problems for further study. One possibility is when we put the cascade of transducers in the parallel communicating frame, that is, the transducers work in parallel, in a synchronized manner, and whenever a query symbol appears at some component, then both the state and the recent output of the connected component is communicated, but this string will be appended to the left of the input string of the receiver (this will be read first). This model will give information on the possible distribution of the work in cascade systems.

Since communication is not always complete in real life, those transducer systems which communicate incomplete information are of interest as well. In this case only a (proper) subword of the output string is communicated. For parallel communicating grammar systems, there is no difference between communicating complete and incomplete information (Csuhaj-Varjú and Vaszil 2002), but it is an important question whether or not the same holds for parallel communicating transducer systems. These investigations can have a lot of practical relevance.

We close with some technical problems which appear of interest to us. Due to space limitations, we just mention them as possible directions for further work.

1. Given a finite transducer A and a PCFTS of size n, $\mathcal{A}$, is

$$T_A \circ T_{\mathcal{A}} = (T_A \circ T_{\mathcal{A}}^{(1)}, T_A \circ T_{\mathcal{A}}^{(2)}, \ldots, T_A \circ T_{\mathcal{A}}^{(n)})$$

   a transduction system function?

2. Is the union of two transduction system mappings still a transduction system function?

3. What one can say about the total transduction mapping of a PCFTS $\mathcal{A}$ defined by

$$\begin{aligned} UT_{\mathcal{A}} &= T_{\mathcal{A}}^{(1)} \cup T_{\mathcal{A}}^{(2)} \cup \ldots \cup T_{\mathcal{A}}^{(n)} \\ UTR_{\mathcal{A}} &= TR_{\mathcal{A}}^{(1)} \cup TR_{\mathcal{A}}^{(2)} \cup \ldots \cup TR_{\mathcal{A}}^{(n)}? \end{aligned}$$

4. What is the complexity of minimizing transducer systems?

Acknowledgements

The work of the first author was supported in part by the NATO Scientific Committee in Spain, in the frame of a fellowship in 2003, while the work of the third author was partly supported by the Centre of Excellence in Information Technology, Computer Science and Control, ICA1-CT-2000-70025, HUN-TING project, WP5.

References

Abney, S. (1996), Partial parsing with finite-state cascades, *Proceedings of the ESSLLI'96 Robust Parsing Workshop*, Prague, pp. 8–15.

Buda, A. (1987), Multiprocessor automata, *Information Processing Letters* **25** (4), 257–261.

Choffrut, C. (1977), Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationelles, *Theoretical Computer Science* **5**, 325–338.

Csuhaj-Varjú, E. (1994), Grammar systems: a framework for natural language generation, *in* G. Păun (ed.), *Mathematical Aspects of Natural and Formal Languages*, World Scientific Publishing, Singapore, pp. 63–78.

Csuhaj-Varjú, E. (2000), Colonies: a multi-agent approach to language generation, *in* A. Kornai (ed.), *Extended Finite-State Models of Language*, Cambridge University Press, Cambridge, pp. 208–225.

Csuhaj-Varjú, E. and Dassow, J. (1990), On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics* **26**, 49–63.

Csuhaj-Varjú, E., Dassow, J., Kelemen, J. and Păun, G. (1994), *Grammar Systems. A grammatical approach to distribution and cooperation*, Gordon and Breach, Amsterdam.

Csuhaj-Varjú, E., Jiménez-López, M. and Martín-Vide, C. (1999), Pragmatic eco-rewriting systems, *in* G. Păun and A. Salomaa (eds), *Grammatical Models of Multi-Agent Systems*, Gordon Breach, Amsterdam, pp. 262–283.

Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V. and Vazil, G. (2000), Parallel communicating pushdown automata systems, *International Journal of the Foundations of Computer Science* **11** (4), 633–650.

Csuhaj-Varjú, E. and Vaszil, G. (2002), Parallel communicating grammar systems with incomplete information communication, *in* A. Kuich and G. Rozenberg (eds), *Proc. Developments in Language Theory*, LNCS 2295, Springer-Verlag, Berlin, pp. 359–368.

Dassow, J. and Mitrana, V. (1999), Stack operation un multi-stack pushdown automata, *Journal of Computer System Science* **58**, 611–621.

Dassow, J., Păun, G. and Rozenberg, G. (1997), Grammar systems, *in* Rozenberg and Salomaa (1997).

Durfee, E., Lesser, V. and Corkill, D. (1989), Cooperative distributed problem solving, *in* P. Barr, R. Cohen and E. Feigenbaum (eds), *The Handbook of AI*, Vol. 4, Addison-Wesley, Boston.

Eilenberg, S. (ed.) (1974), *Automata, Languages, and Machines*, Vol. A, Academic Press, New York.

Engelfriet, J. and Rozenberg, G. (1980), Fixed point languages, equality languages, and representations of recursively enumerable languages, *Journal of ACM* **27**, 499–518.

Evans, E. (1997), Approximating context-free grammars with a finite-state calculus, *Proceedings of ACL 1997*, Madrid, pp. 452–459.

Gazdar, G. and Pullum, G. (1985), Computationally relevant properties of natural languages and their grammars, *New Generation Computing* **3**, 273–306.

Ibarra, O. (1973), On two-way multihead automata, *Journal of Computer System Science* **7**, 28–36.

Jiménez-López, M. (2000), *Grammar Systems: A Formal Language-theoretic*

Framework for Linguistics and Cultural Evolution, PhD thesis, Rovira i Virgili University, Tarragona.

Johnson, M. (1998), Finite-state approximation of constraint-based grammars using left-corner grammar transforms, *Proceedings of COLING-ACL'98*, Montreal, pp. 619–623.

Joshi, A. (1985), How much context-sensitivity is required to provide reasonable structural descriptions? Tree Adjoining Grammars, *in* D. Dowty, L. Karttunen and A. Zwicky (eds), *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, Cambridge University Press, Cambridge, pp. 206–250.

Kempe, A. (2000), Reduction of intermediate alphabets in finite-state transducer cascades, *Proceedings of TALN 2000*, Lausanne, pp. 207–215.

Krauwer, S. and des Tombe, L. (1981), Transducers and grammars as theories of language, *Theoretical Linguistics* **8**, 173–202.

Martín-Vide, C., Mateescu, A. and Mitrana, V. (2002), Parallel finite automata systems communicating by states, *International Journal of the Foundations of Computing Science* **13** (5), 733–749.

Martín-Vide, C. and Mitrana, V. (2001), Some undecidable problems for parallel communicating finite automata systems, *Information Processing Letters* **77** (5–6), 239–245.

Mohri, M. (1996), On some applications of finite-state automata theory to natural language processing, *Journal of Natural Language Engineering* **2**, 1–20.

Mohri, M. (1997a), Finite-state transducers in language and speech processing, *Computational Linguistics* **23** (2), 269–312.

Mohri, M. (1997b), On the use of sequential transducers in natural language processing, *in* Roche and Schabes (1997).

Mohri, M. and Nederhof, M. (2001), Regular approximation of context-free grammars through transformation, *in* J. Junqua and G. van Noord (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, Dordrecht, pp. 153–163.

Păun, G. (1998), *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin.

Reutenauer, C. (1993), Subsequential functions, *Proceedings of the International Meeting of Young Computer Scientists*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 62–79.

Roche, E. and Schabes, Y. (eds) (1997), *Finite-State Language Processing*, MIT Press, Cambridge.

Rozenberg, G. and Salomaa, A. (eds) (1997), *The Handbook of Formal Languages*, Springer-Verlag, Berlin.

Schützenberger, M. (1977), Sur une variante des fonctions séquentielles, *Theoretical Computer Science* **4**, 45–51.