Collapsing hierarchies in parallel communicating grammar systems with communication by command

Lucian Ilie

Turku Centre for Computer Science

TUCS Technical Reports

No 12, May 1996

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

May 1996

ISBN 951-650-754-9
ISSN 1239-1891

Abstract

We investigate here, mainly from the point of view of the hierarchies generated by different classes of systems, two variants of the parallel communicating grammar systems with communication by command: the multiple and, respectively, the single communication case. We show that the hierarchies for regular and linear components collapse in the single communication case and the hierarchy for context-sensitive components collapses in both multiple and single communication cases. By a result in [3], it will follow from our result on systems with context-sensitive components that also the hierarchy for context-free components collapses in both cases. Some open problems are also formulated.

Keywords: Grammar systems, parallel communication, communication by command.

TUCS Research Group
"Mathematical Structures of Computer Science"

1 Introduction

A parallel communicating grammar system, as introduced in [10], consists of several grammars which work synchronously, each of them rewriting its own sentential form, the communication being made by *request*: when a component introduces a query symbol (from a special set) for another component, then the latter one sends its current sentential form to the former which rewrites it in place of the query symbol. The language generated by the system is the set of terminal strings generated (using communication or not) by a distinguished component called master. (For further results and references see also [2].)

Another kind of parallel communicating grammar systems, with communication by *command*, is introduced in [3] with suggestions from the WAVE paradigm for data flow in highly parallel machines ([4], [5], [12]), Boltzmann machine ([6]), the Connection Machine ([7], [13]), and other well known parallel machines.

The communication by command means that when the current sentential form derived in a component coresponds to another component, i.e. belongs to the regular language associated to the respective component or fits the pattern (in the sense of [1], [9]) associated to that component, then the sentential form is sent to the other component. The language generated by the system is also the set of terminal strings generated by a component designed as master.

There are many possible variants (see [3]) but in the followings we shall concentrate only on two of them: the multiple communication case, when, if a component can receive from more than one component, then it will receive the concatenation of the sentential forms in a well defined order, and the single communication case when any component can receive from at most one component at the same time.

2 Basic definitions and examples

We shall denote by V^* the set of all finite strings over the alphabet V, the empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$. The set of regular, linear, context-free, and context-sensitive languages will be denoted by REG, LIN, CF, CS respectively. For further elements of formal language theory we refer to [8] and [11].

A *parallel communicating grammar system with communication by command* ($CCPC$ grammar system for short) of degree $n \geq 1$ is a construct of the form

$$\Gamma = (N, T, (S_1, P_1, R_1), \ldots, (S_n, P_n, R_n)),$$

where N is the *nonterminal* alphabet, T is the *terminal* alphabet, and (S_i, P_i, R_i), $1 \leq i \leq n$, are the *components* of the system: S_i is the *axiom*, P_i is the *set of rules*, $P_i \subset (N \cup T)^+ \times (N \cup T)^+$ (we do not allow λ-rules), and $R_i \in REG$ is the *selector* language for the component i.

Such a system works as follows:

– start from the initial configuration $(S_1, S_2, \ldots, S_n)$,

– at each step, the configuration of the system will be described by an n-tuple $(x_1, x_2, \ldots, x_n) \in ((N \cup T)^*)^n$,

– the configuration of the system can be modified either by *rewriting* steps or by *communication* steps,

– rewriting steps are performed componentwise and the derivation must be maximal in each component (that is the component can not rewrite its sentential form any longer),

– communication steps are performed as follows: when (after maximal derivations) some components $S_{i_1}, S_{i_2}, \ldots, S_{i_k}, 1 \leq i_1 < i_2 < \cdots < i_k \leq n$, have derived the strings $w_1, w_2, \ldots, w_k \in R_i$, for some $1 \leq i \leq n, i \notin \{i_1, i_2, \ldots, i_k\}$ (a component may not communicate with itself) and these are all the components, at that moment, able to communicate their sentential forms to the component i, then the string $w_1 w_2 \ldots w_k$, for the *multiple communication* case, or the string w_1, for the *single communication* case, respectively, will *replace* the sentential form of the component i becoming the current sentential form of this component; the components which send their sentential forms will *restart* from the initial symbol,

– if, after a sequence of rewriting/communication steps, the string on the first position in the current configuration is a terminal one, then it belongs to the generated language (so the master is always the first component).

Formally, a *derivation step* is

$$(x_1, \ldots, x_n) \Longrightarrow (y_1, \ldots, y_n) \text{ iff } \begin{array}{l} x_i \Longrightarrow^* y_i \text{ in } P_i \text{ and there is no} \\ z_i \in (N \cup T)^* \text{ with } y_i \Longrightarrow z_i \text{ in } P_i. \end{array}$$

In order to define a communication step, let us denote

$$\delta(x_i, j) = \begin{cases} \lambda, & \text{if } x_i \notin R_j \text{ or } i = j, \\ x_i, & \text{if } x_i \in R_j \text{ and } i \neq j, \end{cases}$$

for $1 \leq i, j \leq n$,

$$\Delta(i) = \delta(x_1, i)\delta(x_2, i) \ldots \delta(x_n, i),$$

for $1 \leq i \leq n$ (the message received by the component i in the multiple case),

$$\Delta_1(i) = \begin{cases} \delta(x_k, i) = x_k, & \text{if there is a } k, 1 \leq k \leq n, \text{ such that} \\ & \delta(x_l, i) = \lambda, \text{ for } 1 \leq l \leq k-1, \text{ and } \delta(x_k, i) \neq \lambda, \\ \lambda, & \text{otherwise,} \end{cases}$$

for $1 \leq i \leq n$ (the message received by the component i in the single case),

$$\delta(i) = \delta(x_i, 1)\delta(x_i, 2)\ldots\delta(x_i, n),$$

for $1 \leq i \leq n$ (the message sent by the component i in both cases).
A *communication step* is:
in the multiple case:

$$(x_1, \ldots, x_n) \vdash (y_1, \ldots, y_n) \text{ iff } y_i = \begin{cases} \Delta(i), & \text{if } \Delta(i) \neq \lambda, \\ x_i, & \text{if } \Delta(i) = \lambda \text{ and } \delta(i) = \lambda, \\ S_i, & \text{if } \Delta(i) = \lambda \text{ and } \delta(i) \neq \lambda, \end{cases}$$

in the single case:

$$(x_1, \ldots, x_n) \vdash_1 (y_1, \ldots, y_n) \text{ iff } y_i = \begin{cases} \Delta_1(i), & \text{if } \Delta_1(i) \neq \lambda, \\ x_i, & \text{if } \Delta_1(i) = \lambda \text{ and } \delta(i) = \lambda, \\ S_i, & \text{if } \Delta_1(i) = \lambda \text{ and } \delta(i) \neq \lambda, \end{cases}$$

The generated language is

$$\begin{aligned} L_c(\Gamma) = \{w \in T^* \mid & (S_1, \ldots, S_n) \Longrightarrow (x_1^{(1)}, \ldots, x_n^{(1)}) \vdash_c (y_1^{(1)}, \ldots, y_n^{(1)}) \Longrightarrow \\ & (x_1^{(2)}, \ldots, x_n^{(2)}) \vdash_c (y_1^{(2)}, \ldots, y_n^{(2)}) \Longrightarrow \cdots \Longrightarrow (x_1^{(k)}, \ldots, x_n^{(k)}), \\ & \text{for some } k \geq 1 \text{ such that } w = x_1^{(k)}\}, \end{aligned}$$

where, for the multiple case $c = \lambda$ and we identify $L_\lambda(\Gamma)$ with $L(\Gamma)$, and $\vdash_\lambda$ with $\vdash$, and, for the single case, $c = 1$ so we have $L_1(\Gamma)$ and $\vdash_1$.

We denote by $CCPC_nX$ the family of languages $L(\Gamma)$, generated by CCPC grammar systems of degree at most $n, n \geq 1$, with components of type $X \in \{REG, LIN, CF, CS\}$, working in the multiple communication way, and by $1CCPC_nX$ the family of languages $L_1(\Gamma)$, generated by CCPC grammar systems of degree at most $n, n \geq 1$, with components of type X, working in the single communication way. When the number of components is arbitrary, we write $CCPC_\infty X$ and, respectively, $1CCPC_\infty X$.

Here are some examples (we shall denote the nonterminals by upper case letters and the terminals by lower case letters).

Example 1.

$$
\begin{aligned}
\Gamma_1 &= (N, T, (S_1, P_1, R_1), \ldots, (S_{k+1}, P_{k+1}, R_{k+1})),\\
&P_1 = \{S_1 \longrightarrow A|a, B \longrightarrow A|a\},\\
&R_1 = B^+,\\
&P_i = \{A \longrightarrow B\}, 2 \le i \le k+1,\\
&R_i = A^+, 2 \le i \le k+1.
\end{aligned}
$$

for some $k \ge 2$. It is easy to see that

$L(\Gamma_1) = \{a^{k^n} \mid n \ge 0\} \notin CF$ (even not in the family of matrix languages),
$L_1(\Gamma_1) = \{a\} \in REG$ (even finite).

Example 2.

$$
\begin{aligned}
\Gamma_2 &= (N, T, (S_1, P_1, R_1), \ldots, (S_{k+1}, P_{k+1}, R_{k+1})),\\
&P_1 = \{S_1 \longrightarrow aS_1|bS_1|A|B\},\\
&R_1 = \{a, b\}^*,\\
&P_i = \{A \longrightarrow a, B \longrightarrow b\}, 2 \le i \le k+1,\\
&R_i = \{A, B, a, b\}^*, 2 \le i \le k+1.
\end{aligned}
$$

for some $k \ge 2$. We get

$$
\begin{aligned}
L(\Gamma_2) &= \{x^k \mid x \in \{a, b\}^+\} \notin CF,\\
L_1(\Gamma_2) &= \{a, b\}^+ \in REG.
\end{aligned}
$$

Example 3.

$$
\begin{aligned}
\Gamma_3 &= (N, T, (S_1, P_1, R_1), (S_2, P_2, R_2)),\\
&P_1 = \{S_1 \longrightarrow A^k|B^k, A \longrightarrow aC|aD|aE, B \longrightarrow bC|bD|bE\},\\
&R_1 = \{A, a, b\}^* \cup \{B, a, b\}^*,\\
&P_2 = \{C \longrightarrow A|B, D \longrightarrow a, E \longrightarrow b\},\\
&R_2 = \{C, a, b\}^* \cup \{D, a, b\}^* \cup \{E, a, b\}^*.
\end{aligned}
$$

for a fixed $k \ge 1$. We obtain

$$L(\Gamma_3) = L_1(\Gamma_3) = \{x^k \mid x \in \{a, b\}^+\} \notin CF.$$

Observe that in examples 1 and 2, using several regular components, very complex languages (non-context-free and even non-matrix languages) can be obtained (even with three components) but only in the multiple communication case; in the single communication case the obtained languages is regular or finite.

In fact, in what concerns regular and linear cases, this multiple communication seems to be the most powerful tool in generating complex languages and, as a strong argument for that, it will be proved in the next section that, in the single communication case, all systems with regular (linear) components can generate only regular (linear) languages.

Observe also that the difference seems not to be so evident in the context-free case where, only with two components, thus in both multiple and single communication cases, non-context-free languages can be obtained (see example 3 above).

3 Results

From the definitions we get:

Lemma 3.1 *For any family* X
1. $X \subseteq CCPC_1X \subseteq CCPC_2X \subseteq \cdots \subseteq CCPC_\infty X$,
2. $X \subseteq 1CCPC_1X \subseteq 1CCPC_2X \subseteq \cdots \subseteq 1CCPC_\infty X$.

Lemma 3.2 *For any families* $X \subseteq Y$
1. $CCPC_nX \subseteq CCPC_nY$,
2. $1CCPC_nX \subseteq 1CCPC_nY$.

Lemma 3.3 *For any family* X, $CCPC_2X = 1CCPC_2X$.

The next theorem shows that in the single communication case no non-regular language can be obtained with regular components.

Theorem 3.1 $REG = 1CCPC_\infty REG$.

Proof. Let $n \geq 2$ be a natural number and take a system with n regular components

$$\Gamma = (N, T, (S_1, P_1, R_1), \ldots, (S_n, P_n, R_n)).$$

For every $i, 1 \leq i \leq n$, let $A_i = (Q_i, N \cup T, q_{0,i}, \delta_i, F_i)$ be a deterministic finite automaton recognizing R_i.

We construct the following regular grammar

$$G = (N', T, S, P)$$

where

$$N' = \{[q_1, q_2, \ldots, q_n, X, i] \mid q_i \in Q_i, 1 \le i \le n, X \in N, i \in \{1, \ldots, n\}\} \cup \{S\}$$

and P contains the following sets of productions:

1. $S \longrightarrow [q_{0,1}, q_{0,2}, \ldots, q_{0,n}, S_i, i]$,
   for $1 \le i \le n$
   (the derivation may start by simulating any component of the given system),

2. $[q_1, q_2, \ldots, q_n, X, i] \longrightarrow w[q'_1, q'_2, \ldots, q'_n, Y, i]$,
   for $1 \le i \le n, X \longrightarrow wY \in P_i, q'_i = \delta_i(q_i, w)$
   (a step in component i),

3. $[q_1, q_2, \ldots, q_n, X, i] \longrightarrow [q_1, q_2, \ldots, q_n, X, j]$,
   for $1 \le i, j \le n, i \ne j$; there is no rule $X \longrightarrow \alpha \in P_i, \delta_j(q_j, X) \in F_j$, and, for any fixed j, i (if any) is minimal with the properties above
   (if the derivation is maximal in the component i, then the obtained string may be sent to the component j only if i is the minimal index component which can send its sentential form to j),

4. $[q_1, q_2, \ldots, q_n, X, 1] \longrightarrow w$,
   for $X \longrightarrow w \in P_1, w \in T^*$
   (the derivation may be finished anytime in the master component),

5. $[q_1, q_2, \ldots, q_n, X, i] \longrightarrow w$,
   for $2 \le i \le n, X \longrightarrow w \in P_i, w \in T^*, \delta_1(q_1, w) \in F_1$, i being minimal with these properties
   (if the obtained string is to be sent to the master component, then the derivation may be finished in any other component).

Informally speaking, G simulates Γ as follows: on the first n positions in the current nonterminal in a derivation in G are the states in which the automata $A_i, 1 \le i \le n$, are lead from the initial states by the terminal string already obtained. On the position $n+1$ is the current nonterminal and on the last position is the number of the component currently simulated. The main idea is that because no multiple communication can be performed in the derivation process in the system Γ, the current sentential form is only passed from a component to another and this can can be simulated by putting together all the rules in Γ's components and remember all the time (by the position $n+2$ in the current nonterminal in G) which component is active. It is easy to see now that $L(G) = L_1(\Gamma)$. $\square$

Corollary 3.1 $REG = 1CCPC_1REG = 1CCPC_2REG = \cdots$
$= 1CCPC_\infty REG.$

Using a similar idea as above, we can prove the same result in the linear case. The difference is that here the current string has two parts: one at the beginning (say w_1) and another at the end (say w_2). That is why we have to remember all the time two possible states for each finite automaton: the first, reached by the automaton from the initial state using w_1 (the automaton is deterministic), and the second one which can be any state from which the automaton can reach, using w_2, a final state.

Theorem 3.2 $LIN = 1CCPC_\infty LIN.$

Proof. Take a system with $n \geq 2$ linear components

$$\Gamma = (N, T, (S_1, P_1, R_1), \dots, (S_n, P_n, R_n)).$$

For every $i, 1 \leq i \leq n$, let $A_i = (Q_i, N \cup T, q_{0,i}, \delta_i, F_i)$ be a deterministic finite automaton recognizing R_i.

We construct the following linear grammar

$$G = (N', T, S, P)$$

with

$$N' = \{[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \dots, q_{1,n}, q_{2,n}, X, i] \mid q_{1,i}, q_{2,i} \in Q_i, 1 \leq i \leq n, X \in N\} \cup \{S\}$$

and P containing the following sets of productions:

1. $S \longrightarrow [q_{0,1}, q_1^F, q_{0,2}, q_2^F, \dots, q_{0,n}, q_n^F, S_i, i]$,
   for $1 \leq i \leq n, q_i^F \in F_i$,

2. $[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \dots, q_{1,n}, q_{2,n}, X, i] \longrightarrow$
   $w_1[q'_{1,1}, q'_{2,1}, q'_{1,2}, q'_{2,2}, \dots, q'_{1,n}, q'_{2,n}, Y, i]w_2$,
   for $1 \leq i \leq n, X \longrightarrow w_1 Y w_2 \in P_i, q'_{1,i} = \delta_i(q_{1,i}, w_1), \delta(q'_{2,i}, w_2) = q_{2,i}$,

3. $[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \dots, q_{1,n}, q_{2,n}, X, i] \longrightarrow$
   $[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \dots, q_{1,n}, q_{2,n}, X, j]$,
   for $1 \leq i, j \leq n, i \neq j$; there is no rule $X \longrightarrow \alpha \in P_i, \delta(q_{1,j}, X) = q_{2,j}$, and, for any fixed j, i (if any) is minimal with the properties above,

4. $[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \dots, q_{1,n}, q_{2,n}, X, 1] \longrightarrow w$,
   for $X \longrightarrow w \in P_1, w \in T^*$,

5. $[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \ldots, q_{1,n}, q_{2,n}, X, i] \longrightarrow w$,
   for $2 \leq i \leq n, X \longrightarrow w \in P_i, w \in T^*, \delta_i(q_{1,i}, w) = q_{2,i}$, i being minimal with these properties.

It is easy to see that if the current sentential form is $w_1[q_{1,1}, q_{2,1}, q_{1,2}, q_{2,2}, \ldots, q_{1,n}, q_{2,n}, X, i]w_2$, and $\delta_i(q_{1,i}, X) = q_{2,i}$ then the current string w_1Xw_2 is in the regular language R_i. Using this comment and those made in the construction of the set of production for the regular case it should be clear that $L(G) = L_1(\Gamma)$. □

Corollary 3.2 $LIN{=}1CCPC_1LIN{=}1CCPC_2LIN{=}\cdots{=}1CCPC_\infty LIN.$

Also in the context-sensitive case, a similar result holds but, in what follows, we are going to show that, in this case, the hierarchy collapses even in the multiple communication case.

Theorem 3.3 $CS = CCPC_\infty CS.$

Proof. Take a system with $n \geq 2$ context-sensitive components

$$\Gamma = (N, T, (S_1, P_1, R_1), \ldots, (S_n, P_n, R_n))$$

where $R_i \in REG, 1 \leq i \leq n$.

We intend to construct a linear bounded automaton A accepting the language $L(\Gamma)$. Our construction will be rather informal because it takes too long to formalize it entirely and the formalism can be understood from the explanations.

The automaton A works mainly by trying, using the backtracking method, to find whether the current string was obtained either by rewriting in the current component or by communication from the others (at the beginning, the current component is the master and the current string is the given string over T). In the first case, A works in the same way in which a linear bounded automaton simulates a context-sensitive grammar (see, for instance, Theorem 9.5 in [8]), i.e. A tries to identify inside the current string right hand side members of productions in the respective component and, when it succeeds, replaces such a member with a corresponding left-hand side one. For the communication, A tries all possibilities for the current string to be communicated by the other components.

The set of the states of A will contain the states (together with some aditional states needed to perform some steps not formalized here)

$$S = \{(i, s) \mid 1 \leq i \leq n, s \in \{r, c, m, b, f\}\}$$

where i means the number of the current component, r stands for rewriting, c for communication, m for maximal, b for back, and f for finish.

The meaning of the state is:

(i, r) – A is checking all possible rewritings of the current string in the component i,

(i, c) – A is checking all possible decompositions of the current string in strings over R_i and also, for each such decomposition, all possibilities in which the current string could have been communicated,

(i, m) – after supposing that the component i has received a part of the current string from another component, A is checking whether the derivation in that component was maximal before the sending,

(i, b) – if the current branch failed, then A goes back to the nearest vertex in the backtracking tree,

(i, f) – if the current branch succeeded (i.e. an initial symbol is obtained), then A is trying to finish the job by checking the entire content of the tape at that moment).

The set of used markers will be

$$M = \{[i] \mid 1 \leq i \leq n\} \cup \{[ij] \mid 1 \leq i < j \leq n\}$$

(if we have at some moment on the the tape $\ldots [i_1]w_1[i_1i_2]w_2[i_2i_3]\ldots$, then that means that A supposed that w_1 was sent by the component i_1, w_2 by i_2, etc.).

We also denote

$$F = \{S_{i_1}S_{i_2}\ldots S_{i_j} \mid 1 \leq j \leq n, 1 \leq i_1 < i_2 < \cdots < i_j \leq n\}$$

the set of contents of the tape in which A accepts.

More detailed, A follows the next algorithm.

0 start with the first component as the current component and with the given string as the current string,

1 if it supposed that the current string was obtained by

- **(a)** derivation in the current component – then go to step 1.1,
- **(b)** communication from the other components – then go to step 1.2,

(always suppose a communication, if unsettled otherwise)

1.1 enter the state (i, r) (i is the number of the current state and r means that A supposes that there was a rewriting)

1.1.1 try all possibilities by which the current string could have been rewritten in one step in the current component, i. e. search for any occurrence of a right hand side member of a rule of the current component,

1.1.2 for any possibility found at step 1.1.1 (if any) replace the right hand side member by a corresponding left one,

1.1.3 if the respective left hand side member introduced was S_i, then (trying to finish) enter the state (i, f) and check whether on the tape, in that moment, excepting the markers, is a string in F. If yes, then **accept**, else go to step 1 with the right neighbour part of the tape as the current string,

1.1.4 if the respective left hand side member is not S_i, then go to step 1 with the obtained string as the current string (because it is possible that some communication has been performed at this point),

1.2 enter the state (i, c) (i is the number of the current component and c means that A supposes that there was a communication),

1.2.1 try all possibilities by which the current string could have been communicated by the other components, i. e. all possibilities in which the current string can be decomposed in strings in R_i,

1.2.2 for any decomposition obtained at step 1.2.1 (if any) insert markers on the tape in such a way that it is clear where each part of the string was supposed to came from. For instance, if the current string (say w) can be decomposed as $w = w_1w_2w_3, w_1, w_2, w_3 \in R_i$, and A supposes that w_1 was sent by the component number 2, w_2 by 4, and w_3 by 5, then A insert $[2]w_1[24]w_2[45]w_3[5]$,

1.2.3 enter the state (i, m) in order to check whether all subwords supposed as sent to the component i were obtained after some maximal derivations in the respective components,

1.2.4 if yes, then go to step 1 with the current component the one which was supposed to have sent the leftmost substring of the current string (in the decomposition just analyzed) and with this substring as the current string,

1.2.5 if not, then try another decomposition (go to step 1.2.2) (if any) and, if all decompositions have been analyzed, then enter the state (i, b) for going back in the backtracking tree and go to step 1 to try another rewritting,

2 this step is reached if and only if all branches in the backtracking tree have failed so, in this case, A **rejects**.

Observe that we can make the following assumption: if a component S_i sends a string w to another component S_j, then it is not possible that S_j sends w back to S_i without modifying it and, in that moment, S_j is the only component in the system which can send its sentential form to S_i. As it can be seen, our assumption does not reduce the generative power of the system because, anyway, if S_i receives w unmodified, then it will not be able to rewrite it any longer since the derivation just before sending w to S_j was maximal. Using this assumption and the fact that any string sent by a component to another cannot be empty, it is easy to see that the backtracking tree (so the number of possibilities to be checked by A) is finite and the number of markers which can be introduced is linearly bounded by the length of the initial string.

Consequently, we get that A does well our job, and the proof is over. □

It is easy to see that a similar (but simpler) proof could be followed for the single communication case. Therefore we have:

Corollary 3.3 *1.* $CS = 1CCPC_1CS = 1CCPC_2CS = \cdots = 1CCPC_\infty CS$.
2. $CS = CCPC_1CS = CCPC_2CS = \cdots = CCPC_\infty CS$.

In [3] is proved that $CS \subseteq CCPC_2CF$. Consequently, using also Lemma 3.3, we have the following result.

Corollary 3.4 *1.* $CF = CCPC_1CF \subset CCPC_2CF = CCPC_\infty CF = CS$,
2. $CF = 1CCPC_1CF \subset 1CCPC_2CF = 1CCPC_\infty CF = CS$.

This means that also the hierarchy of languages generated by systems with context-free components collapses in both cases.

The problem whether or not the hierarchy for systems with regular components working in the multiple communication way collapses remains **open**. We believe that the answer is affirmative.

Acknowledgement

I would like to thank to my advisor, Arto Salomaa, for useful comments on an earlier version of this paper.

References

[1] D. Angluin, Finding patterns common to a set of strings, *J. Comput. Syst. Sci.*, 21 (1980), 46 – 62.

[2] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[3] E. Csuhaj-Varju, J. Kelemen, Gh. Păun, Parallel communicating grammar systems with WAVE-like communication, submitted, 1995.

[4] L. Errico, *WAVE: An Overview of the Model and the Language*, CSRG, Dept. Electronic and Electr. Eng., Univ. of Surrey, UK, 1993.

[5] L. Errico, C. Jesshope, Towards a new arhitecture for symbolic processing, in vol. *Artificial Inteligence and Information-Control Systems of Robots '94* (I. Plander, ed.), World Sci. Publ., Singapore, 1994, 31 – 40.

[6] S. E. Fahlman, G. E. Hinton, T. J. Seijnowski, Massively parallel arhitectures for AI: NETL, THISTLE and Boltzmann machines, in *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109 – 113.

[7] W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.

[8] J. E. Hopcroft, J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Mass., 1979.

[9] T. Jiang, E. Kinber, A Salomaa, K. Salomaa, S. Yu, Pattern languages with and without erasing, *Intern. J. Computer Math.*, 50(1994), 147 – 163.

[10] Gh. Păun, L. Sântean, Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Math.-Informatics Series*, 38, 2 (1989), 55 – 63.

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

[12] P. S. Sapaty, *The WAVE Paradigm*, Internal Report 17/92, Dept. Informatics, Univ. of Karlsruhe, Germany, 1992.

[13] ***, *Connection Machine, Model CM-2. Tehnical Summary*, Thinking Machines T. R. HA 87 – 4, MIT, Cambridge, USA, 1987.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

University of Turku
- **Department of Mathematical Sciences**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**