Parallel Communicating Grammar Systems with Terminal Transmission

Henning Fernau

WSI-2000-15

Henning Fernau

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany

E-Mail: fernau@informatik.uni-tuebingen.de
Telefon: (07071) 29-77565
Telefax: (07071) 29-5061

ISSN 0946-3852

Parallel Communicating Grammar Systems with Terminal Transmission

Henning Fernau
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
fernau@informatik.uni-tuebingen.de

August 1, 2000

Abstract

We introduce a new variant of PC grammar systems, called PC grammar systems with terminal transmission, PCGSTT for short. We show that right-linear centralized PCGSTT have nice formal language theoretic properties: they are closed under gsm mappings (in particular, under intersection with regular sets and under and homomorphisms) and union; a slight variant is, in addition, closed under concatenation and star; their power lies between that of n-parallel grammars introduced by Wood and that of matrix languages of index n, and their relation to equal matrix grammars of degree n is discussed. We show that membership for these language classes is complete for NL. In a second part of the paper, we discuss questions concerning grammatical inference of these systems. More precisely, we show that PCGSTT whose component grammars are terminal distinguishable right-linear, a notion introduced by Radhakrishnan and Nagaraja in [29, 30], are identifiable in the limit if certain data communication information is supplied in addition.

Part 0: Motivation, Definitions and Examples

1 Introduction

Parallel communicating grammar systems (PCGS, for short) were introduced in [27] in order to investigate concepts like parallelism, synchronization and data communication with formal language theoretic means.

Unfortunately, the language families introduced in this fashion are rather intricate from a formal language point of view, even if one restricts oneself to right-linear grammar components, as has been done, for instance, it the already quoted introductory PCGS paper of Păun and Santean. Only recently, several closure properties have been shown by Autebert [4] by means of special techniques. There are few non-trivial known inclusion properties with respect

to other language families studied in the literature. As regards to complexity questions, in particular the fixed membership problem, the inclusion of so-called centralized regular PCGS within the complexity class NL (defined by languages acceptable by nondeterministic Turing machines whose work tape is only of logarithmic length in terms of the length of the input word) has been shown by Cai in [10] by using a special construction. On the other hand, The technique used by Abrahamson, Cai and Gordon to show NL hardness for so-called coherent PCGS heavily makes use of non-centralized features and is, hence, not applicable to our case [1]. In fact, NL hardness of the fixed membership problem for non-centralized regular PCGS with two components already follows by its inclusion of the linear languages proved in [12] in combination with the well-known NL hardness result for linear languages due to Sudborough [38].

We will discuss a variant of PCGS which we call PCGS with terminal transmission, PCGSTT for short, a model where only transmissions of terminal strings are allowed in order to exclude the influence of the queried component grammar on the querying component grammar. Systems with this property were already investigated by Păun in [28] under the name *terminally synchronized* PCGS. They appear to have rather weak descriptive capacity. In our version, we enhance the power of the mechanism by the simple trick that we consider the so-called query symbols formally as terminal symbols (and not as nonterminal symbols). We further show that right-linear PCGS with terminal transmission (in our definition) have rather nice formal language properties, including simple hierarchical relations to well-known regulated formal language classes and complexity classes, contrary to the case of the classical variant of regular PCGS. We will treat these questions in the first part of this paper.

Moreover, there is also another motivation for discussing this variant which is intrinsic to PCGS: Parallel synchronized computation should be free from side-effects, since this allows the calling processor to continue its work without waiting for the end of the transmission of the results from the called processor. Otherwise, it is not reasonable to assume that communication (such as derivation) takes only one step. We can also think of PCGSTT as modelling data independence features by way of grammar systems, a notion well-studied in the parallel complexity community under the name of *owner read, owner write PRAM* (see [23, 24, 34]). Observe that data independence is a particularly useful and reasonable assumption when considering both derivation steps and communication steps of a grammar system to happen in unit time steps since, then, a possibly long communicated terminal string can be safely "buffered" somewhere, and the derivation of the system may proceed, since the communicated string has no influence on the further development of the system.

Another aim and motivation of the present study was to investigate the inferrability of certain right-linear PCGS language families, given positive samples only, within the learning model "identification in the limit" proposed by Gold [19]. This is an important issue from two different angles: (1) People who like to apply Gold's learning model should be given the possibility to choose among a great variety of grammar or automata formalisms the one which meets their needs best. Concerning PCGS language families, applications where par-

allelism, synchronization and data communication are involved are good candidates. (2) It is always a challenge for theoreticians to make their work as applicable as possible. For reasons explained below, we did not forsee good opportunities to develop a learning theory for the PCGS formalism as defined classically. Therefore, we introduced the present model. In particular, the data independence feature seems to be needed for inference purposes. We will come to this issue in the second part of this paper. We conclude that part with a detailed possible application scenario originating from discussions with people working in the hardware manufacturing industry.

A preliminary version of this paper is included in the proceedings of the conference Grammar Systems 2000, see [15].

2 Definition of PCGS with Terminal Transmission

Definition 1 A *right-linear parallel communicating grammar system with terminal transmission* with n *components*, where $n \geq 1$ (a PCGSTT for short), is an $(n+3)$-tuple $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$, where N is a *nonterminal alphabet*, Σ is a *terminal alphabet* and $K = \{q_1, q_2, \ldots, q_n\}$ is an alphabet of *query symbols*. N, Σ and K are pairwise disjoint sets, $G_i = (N_i, \Sigma \cup K, P_i, S_i)$, $1 \leq i \leq n$, called the *components* of Γ, are usual Chomsky grammars with nonterminal alphabet $N_i \subseteq N$, terminal alphabet $\Sigma \cup K$, a set of productions P_i and *axiom* (or start symbol) S_i. The rules are of the form $A \to wB$, where $w \in \Sigma^* K^*$ and $B \in \{\lambda\} \cup N$. We require that $N = N_1 \cup \ldots \cup N_n$. G_1 is said to be the *master grammar* (or *master*) of Γ.

Observe that query symbols are formally considered as additional terminal symbols. Of course, it would also be possible to admit context-free or linear grammar components, but since we are going to restrict our considerations to the regular case in the following, we only defined right-linear PCGSTT above.

Definition 2 Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$, $n \geq 1$, be a PCGSTT. An n-tuple $(x_1, \ldots, x_n)$, where $x_i \in (\Sigma_i \cup N_i \cup K)^*$, $1 \leq i \leq n$, is called a *configuration* of Γ. $(S_1, \ldots, S_n)$ is said to be the *initial configuration*.

PCGSTT change their configurations by performing direct derivation steps.

Definition 3 Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$, $n \geq 1$, be a PCGSTT and let $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ be two configurations of Γ. We say that $(x_1, \ldots, x_n)$ *directly derives* $(y_1, \ldots, y_n)$, denoted by $(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$, if one of the following two cases hold:
1. There is no x_i which contains any query symbol, that is, $x_i \in (N_i \cup \Sigma)^*$ for $1 \leq i \leq n$. In this case, y_i is obtained from x_i by a direct derivation step in G_i, that is, $x_i \Rightarrow_{G_i} y_i$. For $x_i \in \Sigma^*$, we have $x_i = y_i$ in the normal mode. In the *fully synchronized mode*, we block the derivation of the grammar system if $x_i \in \Sigma^*$.

2. There is some x_i, $1 \leq i \leq n$, which contains at least one occurrence of query symbols. Let x_i be of the form $x_i = zq_{i_1}q_{i_2}\dots q_{i_t}A$, where $z \in \Sigma^*$, $A \in N \cup \{\lambda\}$ and $q_{i_l} \in K$, $1 \leq l \leq t$. In this case, $y_i = z_1 x_{i_1} x_{i_2} \dots x_{i_t} A$ if, for all $1 \leq l \leq t$, $x_{i_l} \in (\Sigma_{i_l} \cup K)^*$. If a would-be communicated string x_{i_l} contains a nonterminal symbol, the derivation blocks.

In the *returning mode*, all non-queried components are reset, i.e., $y_i = S_i$ in those cases.

In the *non-returning mode*, all non-querying components continue their work, i.e., $y_i = x_i$ in those cases.

The first case is the description of a rewriting step: if no query symbol is present in any of the sentential forms, then each grammar component uses one of its rewriting rules, except for those which have already produced a terminal string. If a terminal string has been produced, we distinguish two cases. Observe that the derivation is blocked if a sentential form of some component grammar is not a terminal string, but no rule can be applied to it.

The second case describes a communication: if some query symbol, say q_i, appears in a sentential form, then rewriting stops and a communication step must be performed. The symbol q_i must be replaced by the current terminal sentential form of component G_i, say x_i. Observe that in classical PCGS, query symbols are not communicated. Instead, first only components querying other components whose sentential form does not contain any query symbol may get their queries satisfied, and the other components whose sentential forms contain query symbols stay with these same sentential forms until they, in the next steps, eventually may satisfy their queries. It is seen quite easily that our model is indeed equivalent to this "classical" definition.

To finish a communication step, the components are reset according to the choice between returning or non-returning mode. Note that this is a slight difference to the "classical" model of returning PCGS, where only queried components resume their work on the axiom, but when considering terminal transmission, it seems to be unreasonable to assume that queried components maintain their (terminal) sentential forms; at least, they should restart.

If a circular query appears, the derivation continues forever without producing a terminal string as a result.

Let $\Rightarrow_{rew}$ and $\Rightarrow_{com}$ denote a rewriting step and a communication step, respectively. Let $\Rightarrow^*$ denote the reflexive transitive closure of $\Rightarrow := \Rightarrow_{rew} \cup \Rightarrow_{com}$.

Definition 4 Let $\Gamma = (N, K, \Sigma, G_1, ..., G_n)$ be a PCGSTT with master grammar G_1 and let $(S_1, ..., S_n)$ denote the initial configuration of Γ. The *language* generated by the PCGSTT Γ is

$$L(\Gamma) = \{\alpha_1 \in T^* \mid (S_1, \dots, S_n) \Rightarrow^* (\alpha_1, \dots, \alpha_n)\}.$$

Thus, the generated language consists of the terminal strings appearing as sentential forms of the master grammar G_1.

Our studies in the present paper will focus on returning PCGSTT which are *centralized*, i.e., only the master grammar may introduce query symbols. Hence,

circular queries will never occur. Furthermore, the communication in these parallel communicating grammar systems can be called truly data-independent, since both the (indirect) influences on the calling grammar via communicated query symbols and via nonterminals is explicitly excluded.

Let the class of returning centralized right-linear PCGSTT with at most n right-linear components, as well as the class of languages generated by these systems, be denoted by $\mathrm{PC}_n\mathrm{GSTT}$. When an arbitrary number of components is considered, we use $*$ in the subscript instead of n. We add fs in our notation if we consider full synchronization, so that, for example, we arrive at the language class $\mathrm{PC}_*\mathrm{GSTTfs}$.

3 Examples of PCGSTT

Example 5 Consider a PCGSTT with 3 components, where the master grammar contains the rules $S_1 \to aS_1$ and $S_1 \to aq_2q_3$, G_2 has the rules $S_2 \to bS_2$ and $S_2 \to b$ and G_3 has the rules $S_3 \to cS_3$ and $S_3 \to c$. This PCGSTT generates

$$\{\, a^k b^\ell c^m \mid 1 \leq \ell, m \leq k \,\}$$

in the normal mode and

$$\{\, a^m b^m c^m \mid 1 \leq m \,\}$$

when viewed as fully synchronized.

Example 6 Consider a PCGSTT with 2 components, where the master grammar contains the rules $S_1 \to S_1$ and $S_1 \to q_2q_2$ and G_2 has the rules $S_2 \to aS_2$, $S_2 \to bS_2$ and $S_2 \to \lambda$. This PCGSTT generates

$$\{ww \mid w \in \{a,b\}^*\},$$

both in normal and fully synchronized mode.

These examples show that particularly the full synchronization mode can be of help to define languages which are "complicated" when considering its Chomsky type.

Example 7 Consider the following example of a PCGSTT with three components: $\Gamma = (\{S, A\}, \{q_2, q_3\}, \{a\}, G_1, G_2, G_3)$, where G_i has rule set P_i with: $P_1 = \{S \to S, S \to q_3A, A \to A, A \to \$q_2q_3q_2A, A \to q_3\}$, $P_2 = \{S \to S, S \to \$A, A \to aA, A \to a\}$ and $P_3 = \{S \to S, S \to \$S, S \to aS, S \to \$, S \to a\}$.

Due to the looping rules $S \to S$, the fully synchronized mode is equivalent, in this case, to the normal mode. Now, consider the language $L = L(\Gamma) \cap (\$a^+\$\$a^+)^+\$$. Let $w \in L$. w encodes an instance of the graph accessibility problem for directed graphs in the following way:

- The nodes of the graph are coded as subwords $\$a^i\$$, designating node number i in this way.

- Each subword of the form $\$a^i\$\$a^j$ codes an arc from node i to node j.
- The graph accessibility problem asks whether there is a directed path from some specified start node i_0 to some specified target node i_t. In our coding, we simply assume that the first coded node i_0, which is represented as the prefix of the form $\$a^{i_0}\$$ in w, is the start node, and that the last coded node i_t, which is represented as the suffix of the form $\$a^{i_t}\$$ in w, is the target node.

Now, if $w \in L$, then w encodes an instance of the graph accessibility problem; in particular, this means that there exists at least one directed path from the encoded start node to the target node of the graph. This is guaranteed by the two calls of the second grammar component. The third component is only necessary to "hide away" the created path within other edges generated by the third component.

On the other hand, any instance of the graph accessibility problem can be coded in this way.

This third example shows that our systems can generate languages of a higher computational complexity than any single component may create, since regular languages are complete for the complexity class NC^1, while the graph accessibility problem is the paradigmatic hard problem for NL, nondeterministic logarithmic space.

Lemma 8 *For all* $n \geq 1$, $PC_nGSTT \subseteq PC_nGSTTfs$.

PROOF. In each non-master component, one has simply to introduce a "looping rule" of the form $S \to S$, where S is the start symbol of that component. In this way, the component may defer its termination to the required synchronization point. □

Part 1: Formal Language Issues

4 Closure Properties

Closure properties are known for regular PCGS only by recent results, see [4]. Their proofs generally require the combination of several techniques specific to PCGS. We give here positive closure results for PCGSTT, basically using only standard arguments.

First, recall the notion of a generalized sequential machine (gsm): a gsm $\gamma = (Q, \Sigma, \Delta, \delta, q_0, Q_f)$ has state alphabet Q, input alphabet Σ, output alphabet Δ, start state $q_0 \in Q$, a set of final states $Q_f \subseteq Q$ and a finite transition relation $\delta \subseteq Q \times \Sigma \times \Delta^* \times Q$, whose rules are also written as $qa \to wq'$, where $q, q' \in Q$, $a \in \Sigma$ and $w \in \Delta^*$. These rules define a rewriting system: a string of the form $uqav$ with $u \in \Delta^*$, $q \in Q$, $a \in \Sigma$ and $v \in \Sigma^*$ yields $uwq'v$, written $uqav \Rightarrow_\gamma uwq'v$, when the rule $qa \to wq'$ is applied. Then, for $L \subseteq \Sigma^*$, let

$\gamma(L) = \{ w \in \Delta^* \mid \exists v \in \Sigma^* \exists q_f \in Q_f q_0 v \stackrel{*}{\Rightarrow}_\gamma w q_f \}$, where $\stackrel{*}{\Rightarrow}_\gamma$ is the reflexive transitive closure of $\Rightarrow_\gamma$.

Theorem 9 PC$_*$GSTT *and* PC$_*$GSTTFs *are closed under gsm mappings.*

PROOF. A variant of the "pair construction" (remembering the original plus the current state of the simulated gsm) may be used for the proof. The non-master components are multiplied (times the square of the number of states of the considered gsm), so that the appropriate gsm simulating component can be queried by the master.

More precisely, let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a PCGSTT(fs).

Without loss of generality, we can assume that all non-master components generate infinite languages since, otherwise, calls to components generating finite languages can be incorporated directly in the master component and, furthermore, blockages for the grammar derivation due to components generating finite languages can be simulated within the master component by simple counting.

For simplicity, we may assume $K = \{2, \ldots, n\}$, i.e., $i \in K$ can be considered as label of G_i. Consider a gsm $\gamma = (Q, \Sigma, \Delta, \delta, q_0, Q_f)$.We define a PCGSTT(fs)

$$\Gamma' = (N', K' = K \times Q \times Q, \Delta, G'_1, \ldots, G'_{n'})$$

, with $n' = 1 + (n-1)|Q|^2$.

The new master grammar G'_1 contains, for each rule p of G_1, several rules according to the following two cases.
Case 1: $p = A \to w i_1 \ldots i_k B$, $w \in \Sigma^*$, $i_j \in K$, $B \in N$. Then, G'_1 contains all rules of the form

$$(A, q) \to w'(i_1, q_1, q_2) \ldots (i_k, q_k, q_{k+1})(B, q_{k+1}),$$

where $q, q_1, \ldots, q_{k+1} \in Q$ and $qw \stackrel{*}{\Rightarrow}_\gamma w' q_1$.
Case 2: $p = A \to w i_1 \ldots i_k$, $w \in \Sigma^*$, $i_j \in K$. Then, G'_1 contains all rules of the form

$$(A, q) \to w'(i_1, q_1, q_2) \ldots (i_k, q_k, q_{k+1}),$$

where $q, q_1, \ldots, q_{k+1} \in Q$, $q_{k+1} \in Q_f$ and $qw \stackrel{*}{\Rightarrow}_\gamma w' q_1$.
If S is the start symbol of G_1, then (S, q_0) is the start symbol of G'_1.

Consider a non-master grammar G'_j with label (i.e., corresponding query symbol $(i, \bar{q}, \bar{q}')$). Then, G'_j simulates G_i and, in parallel, it simulates γ starting from state $\bar{q}$ and terminating in state $\bar{q}'$. If $A \to wB$, $w \in \Sigma^*$, $B \in N$ is a rule in G_i, then $(A, q) \to w'(B, q')$ is a rule in G'_j, where $q, q' \in Q$ and $qw \stackrel{*}{\Rightarrow}_\gamma w' q'$. If $A \to w$ with $w \in \Sigma^*$ is a rule in G_i, then $(A, q) \to w'$ is a rule in G'_j, where $q \in Q$ and $qw \stackrel{*}{\Rightarrow}_\gamma w' \bar{q}'$. If S is the start symbol of G_i, then $(S, \bar{q})$ is the start symbol of G'_j. □

We refrain from giving a formal inductive proof argument of the construction in the previous proof. Instead, we mention the following observations:

1. The returning mode is essential for the correctness of the construction since, otherwise, the synchronized simulation of one grammar component G_i of the original grammar by a number of grammar components G'_j of the simulating grammar couldn't be guaranteed.

2. The simulation works both in normal and in fully synchronized mode.

3. The construction does not transfer to the more general case of arbitrary rational transductions, since desynchronization effects due to λ-transition might occur. Only subsequential transductions where, basically, a regular set may be appended at the end of a string translation by a gsm, can be simulated; this result immediately follows from the gsm-closure shown above and the catenation closure proved below. For different notions of transductions, we refer to [5].

By making use of standard constructions, one can show:

Theorem 10 *For each* $n \geq 1$, $\mathrm{PC}_n\mathrm{GSTT}$ *and* $\mathrm{PC}_n\mathrm{GSTTfs}$ *are closed under homomorphisms.* □

Theorem 11 $\mathrm{PC}_*\mathrm{GSTT}$ *and* $\mathrm{PC}_*\mathrm{GSTTfs}$ *are closed under union.*

Sketch of Proof: Assume that L_1 is generated by a PCGS G_1 with n_1 components with axioms $A_1, \ldots, A_{n_1}$, and that L_2 is generated by a PCGS G_2 with n_2 components with axioms $B_1, \ldots, B_{n_2}$. We may assume that the nonterminal alphabets of G_1 and G_2 are disjoint. The PCGS G generating the union $L = L_1 \cup L_2$, has $n = n_1 + n_2 - 1$ components. Assuming a new axiom S_1 for the master component, the master grammar has all rules of the master grammars of G_1 and G_2 (where the query symbols have to be adapted in an obvious manner) plus the new start rules (which play the role of nondeterministic choice rules in the classical textbook constructions) $S_1 \to w$, for all rules $A_1 \to w$ of the master component of G_1 and for all rules $B_1 \to w$ of the master component of G_2. The second through n_1th components are the non-master components of G_1, and the (n_1+1)th through (n_1+n_2-1)th components are the non-master components of G_2. Querying rules of G_1 and G_2 (which are now collected in the new master component of G) have to be updated accordingly. □

Theorem 12 $\mathrm{PC}_*\mathrm{GSTT}$ *and* $\mathrm{PC}_*\mathrm{GSTTfs}$ *are closed under plus and star operations.*

Sketch of Proof: We only show the construction for the plus operation. Due to the closure under union, the claim concerning the star operation follows immediately.

Let a PCGS generating L consist of n components. We describe a PCGS for L^+ with $n+1$ components in the following. The $(n+1)$th component has just the rules $S_{n+1} \to S_{n+1}$ and $S_{n+1} \to \lambda$ and is used as a "synchronizer". For every rule $A \to w$ where $w = sq$ with $s \in \Sigma^*$ and $q \in K^*$, put $A \to sqq_{n+1}S_1$

as an additional rule into the master grammar. In this way, all components (besides the master component) reset their work and start new derivations from their start symbols again. □

Observe that the "blow-up" of the number of components is only due to the fact that we have to synchronize all components by querying a "dummy component". This is not necessary if all terminal rules $A \to w$ of the original master contain query symbols. This can be seen best by the following example:

Example 13 $\{a^n b^n \mid n > 0\}^* \in \mathrm{PC}_2\mathrm{GSTTFs}$.

Proof. Consider $\Gamma = (N, K, \Sigma, G_1, G_2)$, where G_1 has the rules $S_1 \to \lambda$, $S_1 \to aA$, $S_1 \to aq_2S_1$, $A \to aA$ and $A \to aq_2S_1$, and G_2 has the rules $S_2 \to bS_2$ and $S_2 \to b$. □

Again, observe that the returning mode feature is essential for the correctness of the construction in Theorem 12; otherwise, the "synchronizer" would not work.

Theorem 14 $\mathrm{PC}_*\mathrm{GSTT}$ *and* $\mathrm{PC}_*\mathrm{GSTTFs}$ *are closed under concatenation.*

Sketch of Proof: It is clear that the language families in question are closed under concatenation with finite languages; this can be easily done within the master component alone.

Hence, we can assume that both languages which are to be concatenated are infinite. This allows us to combine the proofs given for the closure under union and under star: We start simulating both involved grammar systems in parallel (as in the union construction), but with the master component first simulating the first PCGS master component and then switching (as in the star construction) to the simulation of the second PCGS master component. In this switching phase, all non-master components are reset by querying a dummy component generating the empty string as in the star construction. We need the assumption that the original languages are infinite in order to prevent unwanted derivation blockages in the full synchronization mode. □

Since the synchronization feature of PCGS poses some problems, it is not known whether the corresponding language classes are closed under inverse homomorphism. In particular, it is open whether $\mathrm{PC}_*\mathrm{GSTTFs}$ or $\mathrm{PC}_*\mathrm{GSTT}$ forms a full AFL (abstract family of languages). Furthermore, closure under intersection and complementation is not fully explored.

5 Hierarchy Relations

We can prove that $\mathrm{PC}_*\mathrm{GSTTFs}$ strictly include the parallel right-linear grammars defined by Wood and Rosebrugh [31, 32, 33, 44] and are strictly included in the class describable by context-free matrix grammars of finite index. This immediately yields that the fixed membership problem for right-linear PCGSTT is in NL, a result which has also been proven in the classical case by Cai, see [10]

and, more generally, [9]. We elaborate on this in the following section. Moreover, we strongly conjecture that $PC_*GSTTfs$ are incomparable with right-linear simple matrix languages, an issue discussed more thoroughly in the first part of the conclusions.

Definition 15 (n-parallel right-linear grammar) For $n > 0$, an *n-parallel right-linear grammar* (n-PRLG) is an $(n+3)$-tuple $G = (N_1, \ldots, N_n, T, S, P)$, where N_i, $1 \leq i \leq n$, are pairwise mutually disjoint nonterminal alphabets, T is the terminal alphabet, $S \notin N \cup T$ is the start symbol where $N = N_1 \cup \ldots \cup N_n$, and

$$P \subseteq N \times (T^*N \cup T^+) \cup \{S\} \times (N_1 \ldots N_n \cup T^*)$$

is a finite set of rules, written $X \to x$.

The yield relation is defined as follows: for $x, y \in (N \cup T \cup \{S\})^*$, $x \Rightarrow y$ iff either $x = S$ and $S \to y \in P$, or $x = y_1X_1 \ldots y_nX_n$, $y = y_1x_1 \ldots y_nx_n$, where $y_i \in T^*$, $x_i \in T^*N_i \cup T^+$, $X_i \in N_i$ and $X_i \to x_i \in P$, $1 \leq i \leq n$. We extend $\Rightarrow$ to $\overset{*}{\Rightarrow}$ in the usual way. $L \subseteq T^*$ is an *n-parallel right-linear language* (n-PRLL) iff there exists an n-PRLG G generating L. The family of n-PRLL is denoted by $\mathcal{R}_n$. Let $\mathcal{R}_* = \bigcup_{n \geq 1} \mathcal{R}_n$.

Rosebrugh and Wood have shown that $\mathcal{R}_n$ is strictly contained in the family of n-right-linear simple matrix languages, a class which we do not introduce formally here. We refer the reader to [11, 21, 37].[1] Below, instead, we introduce matrix languages of index n which, in turn, generalize the n-right-linear simple matrix languages.

Loosely speaking, one can think of an n-PRLG as consisting of n right-linear grammars working in parallel. There is one subtlety one has to observe: at the very beginning, there is some sort of coordination between the grammars, since the start points of the grammars are selected in a coordinated manner. If this feature is excluded, one arrives at the so-called n-parallel finite state languages which form the proper subclass $\mathcal{J}_n$ of the class $\mathcal{R}_n$, see [43, 44].

We would like to discuss briefly the relations to terminally synchronized centralized PCGS as introduced by [28, Section 5] shortly, supplementing [28] in this way. Without giving definitions and details here, we mention the following properties:

1. For every terminally synchronized centralized PCGS with right-linear components, there exists an equivalent system of the same kind with only two components. Namely, since there will be only (at most) one query of some non-master component at the end of the derivation, one non-master component can simulate a bunch of non-master components by nondeterministic choice at the very beginning of its computation. This observation somehow sharpens [28, Theorem 6] for our purpose.

[1] The formalism of right-linear tuple languages is equivalent to right-linear simple matrix languages, see [25, 33]. The language class has been named "equal matrix languages" by Siromoney [37].

2. The language class derivable by terminally synchronized centralized PCGS with right-linear components are strictly included in the class $\mathrm{PC}_2\mathrm{GSTT}$. Due to the first property, we may assume that the terminally synchronized centralized PCGS which we have to simulate has only two components. Now, interpreting the query symbol as a terminal symbol proves the inclusion. The strictness may be seen through Example 6, since the systems according to Păun's definition can only generate context-free languages, see [28, Corollary 1].

3. The language class derivable by terminally synchronized centralized PCGS with right-linear components are included in the class $\mathcal{R}_2$. Actually, an argument as in the previous point applies.

Theorem 16 $\mathcal{R}_* \subset \mathrm{PC}_*\mathrm{GSTTfs}$.

PROOF. Let $G = (N_1, \ldots, N_n, T, S, P)$ be an n-PRLG. Let $N = \bigcup_{i=1}^{n} N_i$. We give a simulating PCGSTTfs with $|N| + 1$ components, namely a master component M and components G_A, where $A \in N$. Let q_A denote the query symbol for component G_A. M has the following rules:

- a "waiting rule" $S \to S$;
- for every rule $S \to y$ with $y \in T^*$, $S \to y$ is contained in the master's rule set; and
- for every rule $S \to X_1 \ldots X_n$ with $X_i \in N_i$, $S \to q_{X_1} \ldots q_{X_n}$ is in the master's rule set.

Component G_A, where $A \in N_i$, has the rule set $P \cap N_i \times (T^* N_i \cup T^+)$.

The strictness is easily seen by making use of Example 13. The fact that that language is not contained in $\mathcal{R}_*$ can be seen by noticing that that language is not even a linear simple matrix language, see [11, Lemma 1.5.6(iii)]. □

It is moreover possible to characterize $\mathcal{R}_*$ languages in terms of $\mathrm{PC}_*\mathrm{GSTTfs}$:

Theorem 17 *$L \in \mathcal{R}_*$ iff there exists a PCGSTTfs Γ whose non-master components are grouped into n disjoint sets whose query symbols are from K_i, with $L(\Gamma) = L$ such that Γ's master contains only one form of rules besides the wait rule $S \to S$, namely, rules of the form $S \to k$, where S is the master's start symbol and $k \in K_1 \ldots K_n$ is a string of query symbols.*

Sketch of Proof: The previous theorem shows that each $\mathcal{R}_*$ language can be simulated by a PCGSTTfs of the required form.

On the other hand, if we are given a PCGSTTfs of the required form, let us assume that all nonterminal alphabets of all components are pairwise disjoint. We then put all rules of all non-master components into the simulating PRLG rule set plus rules of the form $S \to X_1 \ldots X_n$ iff $S \to q_{i_1} \ldots q_{i_n}$ is a query rule in the master component of the simulated PCGSTTfs and X_j is the start symbol of component i_j for $1 \leq j \leq n$. □

Furthermore, we mention without proof:

Theorem 18 *$L \in \mathcal{J}_n$ iff there exists a $\Gamma \in \mathrm{PC}_n\mathrm{GSTTFs}$, with $L(\Gamma) = L$ such that Γ's master contains only one form of query rules, namely, rules of the form $A \to \gamma q_2 \dots q_n$, where A is some nonterminal symbol of the master and γ is some terminal string.* □

Definition 19 (matrix language of index n) A *matrix grammar* (cf. [11]) is a quintuple $G = (N, T, M, S)$, where N, T, and S are defined as in Chomsky grammars (the alphabet of nonterminals, the terminal alphabet and the axiom) and M is a finite set of *matrices*, each of which is a finite sequence $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \dots, \alpha_r \to \beta_r)$, $r \geq 1$, of context-free rewriting rules over $V = N \cup T$. For some words x and y in V^* and a matrix $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \dots, \alpha_r \to \beta_r)$ in M, we write $x \underset{m}{\Longrightarrow} y$ (or simply $x \Rightarrow y$ if there is no danger of confusion) iff there are strings $x_0, x_1, \dots, x_r$ (called *intermediate sentential forms*) such that $x_0 = x$, $x_r = y$, and for $1 \leq i \leq r$,

$$x_{i-1} = z_{i-1}\alpha_i z'_{i-1},\ x_i = z_{i-1}\beta_i z'_{i-1} \text{ for some } z_{i-1},\ z'_{i-1} \in V^*.$$

The language generated by G is defined as $L(G) = \{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$.

G is *of index n* (cf. [11, 26]) if every terminal word derivable in G has a derivation where each of its intermediate sentential forms has less than $n + 1$ occurrences of nonterminal symbols. The corresponding language class is denoted by $\mathcal{MAT}_n(\mathrm{CF})$ or $\mathcal{MAT}_*(\mathrm{CF})$ if the index is not important. $L \in \mathcal{MAT}_*(\mathrm{CF})$ is also called a matrix language of *finite index*.

Theorem 20 $\mathrm{PC}_*\mathrm{GSTTFs} \subseteq \mathcal{MAT}_*(CF)$.

PROOF. We sketch the simulating matrix grammar in the following. Let $\Gamma = (N, K, \Sigma, G_1, \dots, G_n)$ be a $\mathrm{PC}_n\mathrm{GSTTFs}$. We may assume that all nonterminal alphabets of the components of the simulated PCGSTT Γ are pairwise disjoint. Moreover, without loss of generality we can assume that all non-master components when considered "stand-alone" (i.e., as usual right-linear grammars) generate infinite languages. Let S_i be the axiom of the ith component G_i. Now, we can describe the simulating matrices:

Start rules: If there is a rule $A \to sq_{i_1} \dots q_{i_k} B$ in the master grammar of the simulated PCGSTT where $A \in N$ and $B \in N \cup \{\lambda\}$, $s \in \Sigma^*$ and $q_{i_j} \in K$, then we put a start matrix of the form

$$(S \to (S_1, i_1 \dots i_k, s) s S_{i_1} \dots S_{i_k} E)$$

into the matrix set of the simulating grammar, where $E = \lambda$ if $B = \lambda$ and E is a new distinguished nonterminal, otherwise. In this way, a nondeterministic guess of the querying rule the master will choose some time in the future is made. This starts the simulation of the grammar components which are going to be queried in the correct way. Note that we do not have to worry about simulating grammar components which

are not going to be called, since they will restart after the future query (of other components) due to the returning mode.

If there is a rule $A \to s$ in the master grammar of the simulated PCGSTT where $A \in N$ and $s \in \Sigma^*$, then we put a start matrix of the form $(S \to S_1)$, as well as a terminating matrix of the form $(A \to s)$, into the simulating matrix grammar.

Simulation of a normal rewriting step: Assume rules $A_1 \to \alpha_1 B_1$ from G_1, $r(j) = A_{i_j} \to \alpha_{i_j} B_{i_j}$ from G_{i_j}, $A_1, A_{i_1}, \ldots, A_{i_k}, B_1, B_{i_1}, \ldots, B_{i_k} \in N$, where identical rules $r(j) = r(j')$ are selected if $i_j = i_{j'}$. Such rules are simulated in the matrix grammar by the matrix

$$((A_1, i_1 \ldots i_k, s) \to \alpha_1(B_1, i_1 \ldots i_k, s), A_{i_1} \to \alpha_{i_1} B_{i_1}, \ldots, A_{i_k} \to \alpha_{i_k} B_{i_k}),$$

where the indices $i_1, \ldots i_k$ and the terminal string s come from the first case discussed in the start rules;

If there is a rule $A \to s$ in the master grammar of the simulated PCGSTT, we add the matrix $(A_1 \to \alpha_1 B_1)$ to the matrix set.

Simulation of a communication step of the form $A \to s q_{i_1} \ldots q_{i_k} B$ in the master grammar of the simulated PCGSTT, where $A \in N$ and $B \in N \cup \{\lambda\}$, $s \in \Sigma^*$ and $q_{i_j} \in K$: Let $r(j) = A_{i_j} \to \alpha_{i_j}$ be terminating rules of G_{i_j}, i.e., $\alpha_{i_j} \in T^*$, where identical rules $r(j) = r(j')$ are selected if $i_j = i_{j'}$.

In the case $B \neq \lambda$, consider the matrix

$$((A, i_1 \ldots i_k, s) \to \lambda, A_{i_1} \to \alpha_{i_1}, \ldots, A_{i_k} \to \alpha_{i_k}, E \to (B, j_1 \ldots j_\ell, t) t S_{j_1} \ldots S_{j_\ell} E'),$$

where we assume that there is a rule $B \to t q_{j_1} \ldots q_{j_\ell} C$ in the master grammar of the simulated PCGSTT, where $B \in N$ and $C \in N \cup \{\lambda\}$, $t \in \Sigma^*$ and $q_{j_i} \in K$, and $E' = \lambda$ if $B = \lambda$ with $E' = E$ otherwise.

In the case $B = \lambda$, consider the matrix

$$((A, i_1 \ldots i_k, s) \to \lambda, A_{i_1} \to \alpha_{i_1}, \ldots, A_{i_k} \to \alpha_{i_k}).$$

Observe that the simulating matrix grammar has finite index, but this index need not correspond to the number of components of the simulated PCGSTT, as the following example shows. □

Example 21 The language sequence $L_n = \{ (a^m b)^n \mid m > 0 \}$ has an arbitrarily large matrix grammar index (refer to the proof of Theorem 3.1.7 in [11]), but each of these languages lies in PC_2GSTTfs (more specifically in PC_2GSTT), since the master needs only to query the same second component n times.

In view of the results of the preceding section and keeping in mind the fact that $\mathcal{MAT}_*(\mathrm{CF})$ forms an AFL, it would be interesting to see a proof of the

conjectured strict inclusion of $PC_*GSTTfs$ within $\mathcal{MAT}_*(CF)$, as well as of the strictness of the inclusion of the smallest AFL containing $PC_*GSTTfs$ within $\mathcal{MAT}_*(CF)$.

Due to [11, Lemma 3.1.5], we may conclude:

Corollary 22 *Each language in* $PC_*GSTTfs$ *is semilinear.*

6 Remarks on Complexity

We now turn to the complexity of the membership problem. Actually, we deal with the so-called fixed membership and the non-emptiness problems. The first of these problems fits profits from the language-theoretic hierarchy considerations of the preceding section. More precisely, we consider the following membership problems:

Fixed membership with fixed number of components in the case of PCGS:
Let a PCGS G with n components be fixed.
For given word w, is $w \in L(G)$?

Fixed membership with fixed index in the case of matrix grammars of finite index: Let a matrix grammar G of index n be fixed.
For given word w, is $w \in L(G)$?

NL denotes the class of languages which can be accepted by nondeterministic Turing machines whose working tape is logarithmically space bounded.

It is known that the fixed membership problem with fixed index is NL complete in the case of programmed grammars of finite index, see [16, Fig. 1]. This implies, in particular, that $\mathcal{MAT}_*(CF)$ is contained in NL due to [11, 35]. Fortunately, the classical techniques developed by Sudborough are applicable [39, 38] to our language classes, so that we can show NL-completeness for our variant rather straightforwardly. Namely, we have shown in Example 7 how to encode the graph accessibility problem of directed graphs into a PCGSTT with only three components. Hence, we can conclude:

Corollary 23 *For each* $n \geq 3$, PC_nGSTT *and* $PC_nGSTTfs$ *are complete for* NL. □

Since $PC_1GSTT = PC_2GSTT$ coincides with the regular languages, we may state:

Corollary 24 PC_1GSTT *and* $PC_1GSTTfs$ *are complete for* NC^1. □

(The complexity class NC^1 is defined via special forms of circuits. For further information, the reader is referred to any modern textbook on computational complexity.)

When analyzing Example 7, we notice that the third component is only needed to "hide" the path which is created by the second component within a

list of further edges, so that a simulating logspace Turing machine needs to be able to make nondeterministic choices. Therefore, by making use of a similar construction, hardness of PC_2GSTT for the class L, i.e., deterministic logspace, can be shown.

Corollary 25 PC_2GSTT *and* PC_2GSTTfs *are hard for* L. □

Unfortunately, we do not know whether any of these two classes is contained within L. We also do not know containment in superclasses of L which are subclasses of NL (e.g., so-called symmetric logspace or unambiguous logspace), see, e.g., [23].

Now, we turn to the non-emptiness problem.

Theorem 26 *For each* $n \in \mathbb{N}$, *the non-emptiness problem for* PC_nGSTT *and for* PC_nGSTTfs *grammar systems is complete for* NL.

PROOF. Since the non-emptiness problem is complete for the right-linear grammars which coincide with $\mathrm{PC}_1\mathrm{GSTT} = \mathrm{PC}_1\mathrm{GSTTfs}$ in our notation, the claimed NL-hardness follows immediately.

We need to show how an instance of the non-emptiness problem can be solved on a nondeterministic Turing machine with logarithmic work tape. We only sketch this in the following.

Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a PC_nGSTTfs. $L(\Gamma) \neq \emptyset$ iff there are communication points $k_1, \ldots, k_r \in K^+$ such that, for some terminal strings $w_1, \ldots, w_{r+1}$ and nonterminals $A_0, \ldots, A_r$ of the master (where A_0 is the master's axiom),

$$A_0 \stackrel{*}{\Rightarrow} w_1 k_1 A_1 \stackrel{*}{\Rightarrow} w_2 k_2 A_2 \stackrel{*}{\Rightarrow} \ldots \stackrel{*}{\Rightarrow} w_r k_r A_r \stackrel{*}{\Rightarrow} w_{r+1}$$

is a valid derivation (of the master grammar). We assume that "intermediately" (i.e., in derivation steps indicated by $\stackrel{*}{\Rightarrow}$), there are no communications. This validity of course depends on the derivability of strings in a certain number of derivation steps in the called components. Let $K_i \subseteq K$ be the set of components called, possible more than once, through k_i. Clearly, the derivation sketched above is valid if all called components in each of the sets K_i yields some terminal string. The simulating nondeterministic Turing machine proceeds as follows:

REPEAT FOREVER

Guess whether there will be a next communication point.

IF not: Verify non-emptiness as for right-linear grammars (only consider the master component); in parallel: verify that all other components† may make at least as many derivation steps as the simulated master component did. IF successful, answer: YES (to the non-emptiness question).

IF so: Simulate in parallel all components, i.e., start with all components with the start symbols, and then repeatedly update all nonterminals of all n components† according to the grammar rules. In such a simulation, only the

nonterminal symbols need to be recorded and not the terminal strings. Of course, the communication points are also of interest. If, in the course of such a simulation, the master introduces some $k_i \in K^+$, then all queried components must be able to generate a terminal string at this point.
END REPEAT

Observe that a dagger symbol † indicates a place where it is necessary that we fix the number of components in order to stay in NL, i.e., treat the non-emptiness problem for PC_nGSTT and not for PC_*GSTT. □

Question: Is non-emptiness for PC_*GSTTfs hard for P? Observe that non-emptiness is even PSPACE-complete for $\mathcal{MAT}_*$(CF), see [16, Fig. 1].
Acknowledgments: We thank Markus Holzer for discussions regarding this section.

7 Concluding Remarks 1

We introduced a new class of right-linear PCGS which fit nicely into previously defined language models (it lies between PRLL languages and matrix languages of finite index) and possesses attractive language theoretic properties. Anyhow, many things remain to be clarified from a formal language point of view. We mention, especially, the exact relations to obviously related (but probably incomparable) classes like right-linear simple matrix languages and regular valence languages. $\{ a^n b^n \mid n \geq 0 \}^*$ is an example of a PCGSTTfs language (see Example 13) which is neither a right-linear simple matrix language nor a regular valence language, see [21, 22]. On the other hand, we conjecture that the right-linear simple matrix language $\{ wh(w) \mid w \in \{a,b\}^* \}$, where h, the morphism defined by $h(a) = b$ and $h(b) = a$, is not a PCGSTTfs language and that the regular valence language $\{ a^n b^m a^n b^m \mid n, m > 0 \}$ is not a PCGSTTfs language.

Moreover, decidability questions have been tackled only superficially.

We just mention here that a result of Wood [44] can be sharpened by stating that each 2-parallel right-linear language is a regular additive valence language or, equivalently, a language acceptable by a nondeterministic finite automaton with one blind counter [20]. That result can be generalized to the following fact which we state without proof:

Theorem 27 *$\mathcal{R}_*$ is contained in the class of regular multiplicative valence languages*[2]. □

On the other hand, we remark that the family of languages acceptable by nondeterministic finite automata equipped with one blind counter which is allowed to make at most one turn (i.e., one change between an incrementing and a decrementing phase), a family which can be characterized as the least trio containing

[2]or, equivalently, in the class of nondeterministic finite automata with multiplication without (intermediate) equality tests, cf. [22]

$\{a^n b^n \mid n \geq 0\}$, see [8], is included in the family of 2-parallel languages. The simulation itself is rather straightforward; the "first grammar" has only to memorize its target state (which is reached exactly at the point where the simulated grammar [or automaton] makes its turn) which is the interface to the "second grammar" working in parallel (hence, simulating the counter applications).

The most important language theoretic issue would be to develop criteria for proving non-containment of certain languages in PC_*GSTT and $PC_*GSTTfs$.

Let us mention, finally, that it is quite easy to develop "analyzing" or automata-theoretic models equivalent, in particular, to $PC_*GSTTfs$ (also, in the case when generalizing those systems by allowing arbitrary context-free components), a task which has been proved to be quite hard for the "classical" PCGS definition, see [7].

Part 2: Language Identification

8 Machine Learning

The topic of Machine Learning possesses an ever-growing field of applications, in particular triggered with the advent of the internet as a mass medium and the need to collect pieces of information automatically, leading to newly coined catchwords like "data mining" and "discovery science".

There are various models of machine learning around, i.e., ways to formalize an automatic induction process. The simplest one is certainly "identification in the limit from positive samples", which was introduced by Gold in 1967 [19] and further studied by Blum & Blum in [6] and Angluin [2]. The identification machine IM gets an infinite stream of words $(w_1, w_2, \ldots)$ (possibly with repetitions) from the (unknown) target language $L = \{w_1, w_2, \ldots\}$ and outputs a hypothesis device (grammar or automaton in most cases) D_M after having received the words $w_1, \ldots, w_M$. L is identified by IM if the sequence of devices $D_1, D_2, \ldots$ converges to some D with $L(D) = L$ in the discrete topological space of devices, i.e., there is a constant n (depending on the enumeration of L) such that for all $i \geq n$, $D_i = D_n$. IM is called a *learner* for a language class $\mathcal{L}$ iff IM identifies any language from $\mathcal{L}$ correctly and independent of its enumeration order (which may also contain repetitions).

Unfortunately, the learning model is very weak from a formal language point of view. For example, Gold has shown that there is no learner for a "superfinite" language class, i.e., a class containing all finite and at least one infinite language. This seems to exclude any reasonable class of formal languages, since nearly all "nice" language properties concerning closure operations are lost. In particular, all classes of the Chomsky hierarchy are not identifiable.

¿From a philosophical viewpoint, the non-learnability of any superfinite language class is not such bad news: obviously, for the task of induction, it is necessary to generalize from some given finite sample set towards a concept, i.e., an infinite language. This means that any finite sublanguage of that concept which contains the given finite sample set (which is also called a "characteristic

sample (set)" of the concept) cannot be learned, since it would immediately be generalized to the (hopefully intended) concept.

The weakness of the model results, in particular, from four facts: (1) The such-defined learning algorithms are deterministic (i.e., they do not contain stochastic elements), (2) exact (not approximate) identification is required, (3) no further information (like "negative samples") is given, and (4) the IM is "passive" and cannot ask questions to a "teacher" on its own. On the other hand, the first three points make the language classes amenable to analysis using tools from formal language theory.

9 Identification of PCGS Languages

One way to show identifiability is to transfer known results on learning by formal language constructs. This has been successfully undertaken by using control languages [13, 18, 40, 41] and certain context conditions [17]. We try to pursue a similar method in the case of PCGS here.

In order to explain the difficulties that arise in this approach when taking the "classical" PCGS definition, let us describe a typical identifiable subclass of regular languages.

9.1 Terminal Distinguishable Right-Linear Languages

Let $x \in \Sigma^*$. Then, define $\mathrm{Ter}(x) = \{a \in \Sigma \mid uav = x \text{ and } u, v \in \Sigma^*\}$ and $\mathrm{Ter}(L) = \bigcup_{w \in L} \mathrm{Ter}(w)$.

Definition 28 Let $G = (N, T, P, S)$ be a right-linear grammar with

$$P \subseteq (N \setminus \{S\}) \times (T(N \setminus \{S\})) \cup \{S\} \times (N \setminus \{S\}).$$

G is called *terminal distinguishable right-linear*, or TDRL for short, if it has the following properties:
1. G is *backward deterministic*. ($B \to w$ and $C \to w$ implies $B = C$).
2. For all $A \in N \setminus \{S\}$ and for all $x, y \in L(A)$, $\mathrm{Ter}(x) = \mathrm{Ter}(y)$ holds.
3. If (a) $S \to B$ and $S \to C$ are in P with $B \neq C$ or
if (b) $A \to aB$ and $A \to aC$ are in P with $B \neq C$, then $\mathrm{Ter}(B) \neq \mathrm{Ter}(C)$.

It has been shown in [13] based on [14]:

Theorem 29 *A language is* TDRL *for short iff there is a* TDRL *grammar generating it.* □

In actual fact, Radhakrishnan and Nagaraja [29, 30] had already claimed a similar grammatical characterization for TDRL, but they gave no proof. Our grammatical definition of TDRL mainly adds point 3(a) to their definition, but this turns out to be an essential point in the equivalence proof. For further discussions, we refer to [13].

How does an inference algorithm for TDRL work? We need some auxiliary notions to explain its behaviour. The algorithm receives an input sample set $S^+ = \{w_1, \ldots, w_M\}$. Let $w_i = a_{i1} \ldots a_{in_i}$, where $a_{ij} \in \Sigma$, $1 \leq i \leq M$, $1 \leq j \leq n_i$. The *skeletal grammar* for the sample set is defined as

$$\begin{aligned} G_{S^+} &= (N_{S^+} \cup \{S\}, \Sigma, P_{S^+}, S), \quad \text{where} \\ N_{S^+} &= \{\, N_{ij} \mid 1 \leq i \leq M, 1 \leq j \leq n_i \,\} \quad \text{and} \\ P_{S^+} &= \{\, S \to N_{i1} \mid 1 \leq i \leq M \,\} \\ &\cup \{\, N_{ij} \to a_{ij} N_{i,j+1} \mid 1 \leq i \leq M, 1 \leq j < n_i \,\} \\ &\cup \{\, N_{in_i} \to a_{in_i} \mid 1 \leq i \leq M \,\}. \end{aligned}$$

The *frontier string* of N_{ij} is defined as $\mathrm{FS}(N_{ij}) = a_{ij} \ldots a_{in_i}$. This means that $L(N_{ij}) = \{\mathrm{FS}(N_{ij})\}$. The *head string* of N_{ij} is defined by the equation $\mathrm{HS}(N_{ij})\mathrm{FS}(N_{ij}) = w_i$, i.e., $\mathrm{HS}(N_{ij}) = a_{i1} \ldots a_{i,j-1}$.

Now, consider to nonterminals $N_{i,j}$ and $N_{k,l}$ of a right-linear skeletal grammar as TDRL-*equivalent*, denoted by $N_{i,j} \equiv_{\mathrm{TDRL}} N_{k,l}$, iff $\equiv_{\mathrm{TDRL}} = \rightleftharpoons^+_{\mathrm{TDRL}}$, i.e., the transitive closure of the reflexive symmetric relation $\rightleftharpoons_{\mathrm{TDRL}}$, which is given by $N_{i,j} \rightleftharpoons_{\mathrm{TDRL}} N_{k,l}$ iff

1. $\mathrm{FS}(N_{i,j}) = \mathrm{FS}(N_{k,l})$ and $\mathrm{Ter}(\mathrm{HS}(N_{i,j}) = \mathrm{Ter}(\mathrm{HS}(N_{k,l}))$, or
2. $\mathrm{HS}(N_{i,j}) = \mathrm{HS}(N_{k,l})$.

Without formal proof, we state:

Lemma 30 *$\equiv_{\mathrm{TDRL}}$ is an equivalence relation on the set of nonterminals of the skeletal grammar G_{S^+}.* □

Now, define the grammar G as having nonterminals $[N]_{\mathrm{TDRL}}$, where $[N]_{\mathrm{TDRL}}$ is the set of those nonterminals of G_{S^+} which are TDRL-equivalent to N, and the following rules:

initial rules $S \to [N_{i,1}]_{\mathrm{TDRL}}$ for some $1 \leq i \leq M$;

transition rules $[N_{i,j}]_{\mathrm{TDRL}} \to a_{i,j}[N_{i,j+1}]_{\mathrm{TDRL}}$ for some $1 \leq i \leq M$, $1 \leq j < n_i$; and

terminal rules $[N_{i,n_i}]_{\mathrm{TDRL}} \to a_{i,n_i}$ for some $1 \leq i \leq M$.

Since G is reduced, G is isomorphic to the canonical objects for TDRL defined in [13, 14] in terms of product automata. Hence, the sketched procedure is indeed an identification algorithm for TDRL.

As exhibited in [14] (based on the implementation of the 0-reversible language identification algorithm due to Angluin [3]), the algorithm sketched above can be implemented to run in nearly linear time, i.e., in time $O(\alpha^{-1}(2^\ell n)2^\ell n)$, where ℓ is the size of the input alphabet (this will become somewhat important in the following investigations), n is the total input size and α^{-1} is the inverse of the Ackermann function as defined by Tarjan [42].

Example 31 The skeletal grammar of $S^+ = \{w_1 = ab, w_2 = aab\}$ is:

$$\begin{aligned} G_{S^+} \quad = \quad & (\{S, N_{11}, N_{12}, N_{21}, N_{22}, N_{23}, N_{24}\}, \{a, b\}, \\ & \{S \to N_{11}, N_{11} \to aN_{12}, N_{12} \to b, \\ & S \to N_{21}, N_{21} \to aN_{22}, N_{22} \to aN_{23}, N_{23} \to b\}, S)\,. \end{aligned}$$

$\in N_S$	HS	FS	Ter(FS)
N_{11}	λ	ab	$\{a, b\}$
N_{12}	a	b	$\{b\}$
N_{21}	λ	aab	$\{a, b\}$
N_{22}	a	ab	$\{a, b\}$
N_{23}	aa	b	$\{b\}$

The reader may verify that the inferred grammar is

$$G = (\{S, A, B\}, \{a, b\}, \{S \to A, A \to aA, A \to aB, B \to b\}, S)\,,$$

where $A = [N_{11}]_{\mathrm{TDRL}}$ and $B = [N_{12}]_{\mathrm{TDRL}}$. More precisely, $A = \{N_{11}, N_{21}, N_{22}\}$ and $B = \{N_{12}, N_{23}\}$. G generates $\{a^n b \mid n > 0\}$.

In addition, "structural information" is sometimes provided to the learning algorithm (see [36] in the case of TDRL). It is argued that in applications, certain structural information is always known.

9.2 Structural Information in PCGS

In the case of PCGS, typical structural information would concern the data communication. Having this information somewhat supplied, a natural idea would be to "learn" every component language separately. The main difficulty in this approach lies in the fact that classical PCGS allow the communication of "state information", i.e., nonterminals, from one component to another. It is not clear how to cope with these symbols without telling the IM beforehand which nonterminals will occur in the grammar, information which is generally regarded as part of the inference task.

Therefore, we introduced a variant of right-linear PCGS to avoid this drawback.

10 Concerning Learnability

We discuss PCGSTT whose grammar components are, in some sense, TDRL languages. To this end, we must include the information of where transmission of terminal strings occurred explicitly within the sample words.

In order to define the inclusion of the transmission information rigorously, let us reconsider the definition of a derivation step between two configurations $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ of a PCGSTT $\Gamma = (N, K, \Sigma, G_1, ..., G_n)$. We concentrate on centralized returning PCGSTT in the following. When x_1 contains no query symbol, the relations $\Rightarrow_Q$ and $\Rightarrow_{Q'}$ we are going to define coincide with

$\Rightarrow$. If x_1 contains at least one occurrence of query symbols, we obtain a difference. Let x_1 be of the form $x_1 = zq_{i_1}q_{i_2}\dots q_{i_t}A$, where $z \in \Sigma^*$, $A \in N \cup \{\lambda\}$ and $q_{i_l} \in K$, $1 \leq l \leq t$. In this case, $y_1 = z_1q_{i_1}q_{i_2}\dots q_{i_t}x_{i_1}x_{i_2}\dots x_{i_t}A$ in the case of the relation $\Rightarrow_Q$ and $y_1 = z_1q_{i_1}q_{i_2}\dots q_{i_t}A$ in the case of the relation $\Rightarrow_{Q'}$. The other parts of the definition of $\Rightarrow_Q$ and $\Rightarrow_{Q'}$ are as in the definition of $\Rightarrow$, especially concerning the differences between the fully synchronized and normal mode of derivation.

Definition 32 Let $\Gamma = (N, K, \Sigma, G_1, ..., G_n)$ be a PCGSTT with master grammar G_1 and let $(S_1, ..., S_n)$ denote the initial configuration of Γ. The *Q-language* generated by the PCGSTT Γ is

$$L_Q(\Gamma) = \{\alpha_1 \in T^* \mid (S_1, \dots, S_n) \Rightarrow_Q^* (\alpha_1, \dots, \alpha_n)\}.$$

The *Q'-language* generated by the PCGSTT Γ is

$$L_{Q'}(\Gamma) = \{\alpha_1 \in T^* \mid (S_1, \dots, S_n) \Rightarrow_{Q'}^* (\alpha_1, \dots, \alpha_n)\}.$$

Therefore, an idea would be to design an inference machine for $L_Q(\Gamma)$ instead of $L(\Gamma)$. In this way, we formalize what we mean by explicit information about the communication structure. Since the contributions of the non-master components cannot be uniquely reconstructed from this representation, we concentrate on what we call even PCGSTT. A PCGSTT is called *even* if every right-hand side α of every rule in the PCGSTT contains exactly one symbol from the terminal alphabet Σ.

Consider the following example:

Example 33 $\Gamma = (\{S_1, S_2, S_3\}, \{q_2, q_3\}, \{a, b\}, G_1, G_2, G_3)$, where G_1 contains the rules $S_1 \to aS_1$, $S_1 \to aq_2q_3$, G_2 contains the rules $S_2 \to bS_2$ and $S_2 \to b$, and G_3 contains the rules $S_3 \to aS_3$ and $S_3 \to a$. Obviously, Γ is an even PCGSTT. Considered in the full synchronization mode, Γ generates

$$L(\Gamma) = \{a^m b^m a^m \mid m > 0\}.$$

Moreover, we find that

$$L_Q(\Gamma) = \{a^m q_2 q_3 b^m a^m \mid m > 0\} \text{ and}$$

$$L_{Q'}(\Gamma) = \{a^m q_2 q_3 \mid m > 0\}.$$

Given some word $a^m q_2 q_3 b^m a^m$ in the example above, it is quite easy to tell what the exact contributions of the non-master components are, since Γ was even and worked fully synchronized; namely, since the word derived by the master component up to the first query situation has length m, each of the two words queried from components 2 and 3 must have length m, too. This means that b^m has been contributed by the second component and the postfix a^m has been contributed by the third component. Moreover, it is easy to see that the word $a^m q_2 q_3$ from $L_{Q'}(\Gamma)$ "corresponding" to $a^m q_2 q_3 b^m a^m$ in $L_Q(\Gamma)$

can be deduced in this fashion, as well. Observe that $L_{Q'}$ collects the genuine contributions of the master grammar.

In general, we obtain the following decomposition algorithm.

Algorithm 34 (Decomposition algorithm)
Let $w = v_1 k_1 v_2 k_2 \dots v_\ell k_\ell v_{\ell+1}$ be in the Q-language of some even PCGSTTfs, where $v_i \in \Sigma^*$ and $k_i \in K^+$; more precisely, let $k_i = q_{i,1} \dots q_{i,j_i}$, with $q_{i,j} \in K$. Then v_2 can be decomposed as $v_2 = v_{1,1} \dots v_{1,j_1} v_2'$, where $|v_1| = |v_{1,1}| = \dots = |v_{1,j_1}|$, since $v_{1,m}$ is the contribution of the mth component queried by the master in step m; therefore, $|v_{1,m}| = |v_1|$. Similarly, v_3 can be decomposed as $v_3 = v_{2,1} \dots v_{2,j_2} v_3'$, where $|v_2'| = |v_{2,1}| = \dots = |v_{2,j_1}|$ and so forth. Finally, $v_{\ell+1}$ is a part which is derived by the master grammar only (without further queries to other components).

This decomposition allows us to use the following inference algorithm (based here on the inferrability of TDRL; of course, any inferrable regular language subclass could be "plugged in" instead) for the class which we are going to call TDRL$-$PC$_n$GSTTfs.[3] More precisely, let TDRL$-$PC$_n$GSTTfs denote the class of languages which will be output by the inference algorithm given below. The corresponding languages contain query symbols as guidance for the inference procedure.

Algorithm 35 (Inference of TDRL $-$ PC$_n$GSTTfs**)**

1. Every input word w can be decomposed in order to obtain sets of words W_i generated by component i.
2. W_i can be given to a TDRL algorithm for the ith component.

We have to be a bit careful with the master component, due to the presence of query symbols in words from $L_{Q'}$.

We suggest the following variant for coping with query symbols:
A word

$$w_i = a_{i,1} \dots a_{i,i_1} k_{i,1} a_{i,i_1+1} \dots a_{i,i_2} k_{i,2} \dots k_{i,\ell} a_{i,i_\ell+1} \dots a_{i,i_{\ell+1}}$$

from $L_{Q'}$ with $a_{i,j} \in \Sigma$ and $k_{i,j} \in K^+$ can be generated by using the rules

- $S \to N_{i,1}$,
- $N_{i,j} \to a_{i,j} N_{i,j+1}$ for $j \neq i_\nu$, $1 \leq \nu \leq \ell + 1$,
- $N_{i,i_\nu} \to a_{i,i_\nu} k_{i,\nu} N_{i,i_\nu+1}$ for $1 \leq \nu \leq \ell$, and

[3] Let us point the reader to a subtle minor point we will not discuss further: The TDRL grammars which are inferred by the algorithm given above are not "even" according to our PCGSTT definition, since the start symbol plays an extra role. However, this does not affect the decomposition algorithm just described and may, hence, be neglected.

- $N_{i,i_{\ell+1}} \to a_{i,i_{\ell+1}}$.

The merging conditions are as before, now considering "symbols" which are in fact words from ΣK^*. This means, in particular, that the "alphabet" one has to consider is not fixed beforehand but may keep on growing for some time when new examples (with new sequences of query symbols) emerge.

Let us explore this phenomenon a bit more formally. Consider some language $L_{Q'}$. Let $Q = \{q \in K^+ \mid \exists uaqbv \in L_{Q'}, a, b \in \Sigma\} \cup \{\lambda\}$. Consider $[\Sigma Q] = \Sigma Q$ as a new alphabet and the natural homomorphism $h : [\Sigma Q]^* \to (\Sigma \cup K)^*$ defined as the trivial decoding $h(aq) = aq$ for $a \in \Sigma$ and symbols q from Q. Then, $L_{Q'}$ will be inferred by the suggested algorithm iff $h^{-1}(L_{Q'}) \subseteq [\Sigma Q]^+$ is a TDRL language.

Let us continue our example in order to clarify our ideas:

Consider as input words aq_2q_3ba and aaq_2q_3bbaa.

This means that the words $[aq_2q_3]$ and $[a][aq_2q_3]$ are passed to the TDRL inference algorithm of the master grammar. That inference algorithm would yield $S_1 \to N_{1,1,1}$, $N_{1,1,1} \to aN_{1,1,1}$ and $N_{1,1,1} \to aq_2q_3$. (We use an additional index in the nonterminals to distinguish the different grammar components.)

Similarly, the words b and bb are given to the TDRL inference algorithm of the second component, yielding the rules $S_2 \to N_{2,1,1}$, $N_{2,1,1} \to bN_{2,1,1}$ and $N_{2,1,1} \to b$.

Analogously, the TDRL inference algorithm for the third component outputs the rules $S_3 \to N_{3,1,1}$, $N_{3,1,1} \to aN_{3,1,1}$ and $N_{3,1,1} \to a$.

In this way, a correct PCGSTTfs is induced.[4]

Is there a way of characterizing the language class TDRL $-$ PC$_*$GSTTfs somehow? It is tempting to assume that $L_Q \in$ TDRL $-$ PC$_*$GSTTfs iff the "component languages" (as they could be defined by the decomposition algorithm given above) lie in TDRL. There is one subtle thing that this conjecture might overlook: It is possible that not all words of the language generated by a non-master component are actually queried by the master; more precisely, we have to single out the length of words which could be queried by the master. Let M_i be the set of lengths of words generated by the ith component which can be queried by the master.

In order to understand this difficulty, consider the following example:

Example 36 $\Gamma = (\{S_1, S_1', S_2, S_3\}, \{q_2, q_3\}, \{a, b\}, G_1, G_2, G_3)$ where G_1 contains the rules $S_1 \to aS_1'$, $S_1' \to aS_1$ and $S_1' \to aq_2q_3$, G_2 contains the rules $S_2 \to bS_2$ and $S_2 \to b$, and G_3 contains the rules $S_3 \to aS_3$ and $S_3 \to a$. Obviously, Γ is an even PCGSTTfs. We have

$$L(\Gamma) = \{a^{2n}b^{2n}a^{2n} \mid n > 0\}.$$

The inference algorithm would yield —after the trivial conversion into the even PCGSTTfs form discussed in the footnotes above— the system

$$\Gamma' = (\{S_1, S_1', S_2, S_2', S_3, S_3'\}, \{q_2, q_3\}, \{a, b\}, G_1, G_2, G_3),$$

[4]It is trivial to convert this induced PCGSTTfs into an equivalent even PCGSTTfs.

where G_1 contains the rules $S_1 \to aS_1'$, $S_1' \to aS_1$ and $S_1' \to aq_2q_3$, G_2 contains the rules $S_2 \to bS_2'$, $S_2' \to bS_2$ and $S_2' \to b$, and G_3 contains the rules $S_3 \to aS_3'$, $S_3' \to aS_3$ and $S_3' \to a$. Obviously, Γ' is an even PCGSTTfs which is equivalent to Γ.

Nevertheless, we can state:

Theorem 37 *Fix $n \in \mathbb{N}$. $L_Q \in \mathrm{TDRL} - \mathrm{PC}_n\mathrm{GSTTfs}$ iff*

- *$h^{-1}(L_{Q'}) \in \mathrm{TDRL}$, and*
- *for all $2 \leq i \leq n$, there exists a language $L_i \in \mathrm{TDRL}$ such that*

$$L_i \cap \{\, w \in \Sigma^* \mid |w| \in M_i \,\}$$

is the language queried by the master component.

PROOF. Consider a grammar system which is output by the identification algorithm. Obviously, the L_Q language of such a system is of the form required by the theorem.

On the other hand, consider a language L_Q satisfying the above requirements. Observe that the length sets M_i can be computed from the words in L_Q. Now, let $h^{-1}(L_{Q'})$ be enumerated as input of the TDRL-identification algorithm of the master component. Obviously, since $h^{-1}(L_{Q'}) \in \mathrm{TDRL}$ and due to the identifiability of TDRL, there will be some time step $n_{1,0}$ of this algorithm from which on the hypothesis grammar for the master component will not change any more and the corresponding generated language equals $h^{-1}(L_{Q'})$.

Now, consider some enumeration of $L_i' = L_i \cap \{\, w \in \Sigma^* \mid |w| \in M_i \,\}$ to the TDRL identification algorithm (of the ith component). Due to [14, Lemma 9], the finite automata sequence corresponding to the output sequence of right-linear grammars evoked by the enumeration of L_i' consists solely of sub-automata of the canonical finite automaton A of L_i (where "canonical" refers to the definitions in [13, 14]). Since the identification algorithm is conservative in the sense that it does not change its hypothesis as long as it is consistent with all the enumerated samples, and since the search space (the subautomata of A) is finite, the algorithm will converge, i.e., the output will be constant from a certain time step $n_{i,0}$ onward, yielding some $\bar{L}_i$ satisfying $L_i' = \bar{L}_i \cap \{\, w \in \Sigma^* \mid |w| \in M_i \,\}$. In conclusion, the proposed identification algorithm will converge, and the obtained $\mathrm{TDRL} - \mathrm{PC}_n\mathrm{GSTTfs}$ system generates L_Q. □

Corollary 38 *For each $n \geq 1$, the class $\mathrm{TDRL} - \mathrm{PC}_n\mathrm{GSTTfs}$ is identifiable in the limit from positive samples.* □

Let us mention that the canonical objects required to define the convergence of the identification algorithm properly can be easily derived from the previous theorem and the canonical objects for TDRL-languages, as exhibited in [13, 14]. Moreover, in this way, suitable characteristic samples for languages in $\mathrm{TDRL} - \mathrm{PC}_n\mathrm{GSTTfs}$ can be obtained.

It is easy to see that in this way, one extends the class of identifiable languages (compared to the basic class TDRL) considerably at the cost of providing the additional transmission information contained in L_Q. Of course, one can consider "standardized" or uniform transmission information. For example, as suggested by Theorems 17 or 18, one can define learnable subclasses of $\mathcal{R}_*$.

11 Concluding Remarks 2

In the second part of this paper, we discussed identifiability of language classes defined via right-linear PCGS with terminal transmission. We focussed on the notion of terminal distinguishability. Naturally, similar results can be obtained based on f-distinguishable languages for some arbitrary so-called distinguishing function f, see [13]. Referring to the formal language problems discussed in the first section, we mention that these questions could be discussed for subclasses such as TDRL $-$ PC$_*$GSTTFs, as well.

Of course, the practical applicability of the sketched learning approach has to be verified. Here, it is natural to make experiments with identifiable subclasses of the regular languages different from TDRL, as well.

We only sketch one possible application scenario, motivated by discussion with people working in the hardware manufacturing industry:

Consider some hardware where some central unit collects its own error protocol, as well as the protocols of other components. Based on these common error protocols of the whole machine (each common error protocol corresponds to one run of the machinery), an engineer has to extract the "typical errors" and express these as regular expressions (for each of the hardware units). Assuming the different hardware components to be finite automata (which is, indeed, reasonable from a practical viewpoint) and assuming, further, that the protocols allow for distinguishing the origin of different error messages (as can be done in the formalization developed above by means of the decomposition algorithm), some inference algorithm for PC$_n$GSTTFs can be used to aid the engineer in his/her task, where n is the total number of hardware components involved.

References

[1] K. Abrahamson, L. Cai, and S. Gordon. A grammar characterization of logarithmic-space computation. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages*, volume 1218 of *LNCS*, pages 247–255. Springer, 1997.

[2] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[3] D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.

[4] J.-M. Autebert. Some results about centralized PC grammar system. *Theoretical Computer Science*, 215, 1999.

[5] J. Berstel. *Transductions and Context-Free Languages*, volume 38 of *LAMM*. Teubner, 1979.

[6] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[7] H. Bordihn, J. Dassow, and G. Vaszil. Grammar systems as language analyzers and recursively enumerable languages. In G. Ciobanu and Gh. Păun, editors, *Fundamentals of Computation Theory, Proc. FCT'99*, volume 1684 of *LNCS*, pages 136–147. Springer, 1999.

[8] F.-J. Brandenburg. Intersections of some families of languages. In *Automata, Languages and Programming, Proc. ICALP'86*, volume 226 of *LNCS*, pages 60–69. Springer, 1986.

[9] L. Cai. The complexity of linear PCGS. *Computers and Artificial Intelligence*, 15:199–210, 1996.

[10] L. Cai. The computational complexity of PCGS with regular components. In J. Dassow, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory II; at the crossroads of mathematics, computer science and biology*, pages 209–219. World Scientific, 1996.

[11] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1989.

[12] S. Dumitrescu and Gh. Păun. On the power of parallel communicating grammar systems with right-linear components. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 31(4):331–354, 1997.

[13] H. Fernau. Identifying terminal distinguishable languages. Submitted revised version of the extended abstract which appeared in the Proceedings of AMAI 2000, see `http://rutcor.rutgers.edu/~amai/AcceptedCont.htm`.

[14] H. Fernau. On learning function distinguishable languages. Technical Report WSI–2000–13, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 2000. A conference version will appear at ALT 2000.

[15] H. Fernau. PC grammar systems with terminal transmission. In R. Freund and A. Kelemenová, editors, *Proc. International Workshop on Grammar Systems*, pages 229–252. Silesian University at Opava, 2000.

[16] H. Fernau and M. Holzer. Conditional context-free languages of finite index. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages*, volume 1218 of *LNCS*, pages 10–26. Springer, 1997.

[17] H. Fernau and M. Holzer. External contextual and conditional languages. To appear in a book edited by Gh. Păun, 1999.

[18] H. Fernau and J. M. Sempere. Permutations and control sets for learning non-regular language families. To appear in the Proc. of ICGI 2000.

[19] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[20] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.

[21] O. Ibarra. Simple matrix languages. *Information and Control*, 17:359–394, 1970.

[22] O. H. Ibarra, S. K. Sahni, and C. E. Kim. Finite automata with multiplication. *Theoretical Computer Science*, 2:271–296, 1976.

[23] K.-J. Lange. Complexity and structure in formal language theory. *Fundamenta Informaticae*, 25(3):327–352, 1996.

[24] K.-J. Lange and R. Niedermeier. Data independence of read, write and control structures in PRAM computations. *Journal of Computer and System Sciences*, 60:109–144, 2000.

[25] H. A. Maurer and W. Kuich. Tuple languages. In W. D. Itzfeld, editor, *Proc. of the ACM International Computing Symposium*, pages 882–891. German Chapter of the ACM, 1970.

[26] Gh. Păun. On the family of finite index matrix languages. *Journal of Computer and System Sciences*, 18(3):267–280, 1979.

[27] Gh. Păun and L. Santean. Parallel communicating grammar systems: the regular case. *Ann. Univ. Bucharest, Ser. Matem.-Inform.*, 38(2):55–63, 1989.

[28] Gh. Păun. On the synchronization in parallel communicating grammar systems. *Acta Informatica*, 30:351–367, 1993.

[29] V. Radhakrishnan. *Grammatical Inference from Positive Data: An Effective Integrated Approach.* PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay (India), 1987.

[30] V. Radhakrishnan and G. Nagaraja. Inference of regular grammars via skeletons. *IEEE Transactions on Systems, Man and Cybernetics*, 17(6):982–992, 1987.

[31] R. D. Rosebrugh and D. Wood. A characterization theorem for n-parallel right linear languages. *Journal of Computer and System Sciences*, 7:579–582, 1973.

[32] R. D. Rosebrugh and D. Wood. Image theorems for simple matrix languages and n-parallel languages. *Mathematical Systems Theory*, 8:150–155, 1974.

[33] R. D. Rosebrugh and D. Wood. Restricted parallelism and right linear grammars. *Utilitas Mathematica*, 7:151–186, 1975.

[34] P. Rossmanith. The owner concept for PRAMs. In *8th Annual Symposium on Theoretical Aspects of Computer Science STACS'91*, volume 480 of *LNCS*, pages 172–183. Springer, 1991.

[35] G. Rozenberg and D. Vermeir. On the effect of the finite index restriction on several families of grammars. *Information and Control*, 39:284–302, 1978.

[36] J. M. Sempere and G. Nagaraja. Learning a subclass of linear languages from positive structural information. In V. Honavar and G. Slutski, editors, *Proceedings of the Fourth International Colloquium on Grammatical Inference (ICGI-98)*, volume 1433 of *LNCS/LNAI*, pages 162–174. Springer, 1998.

[37] R. Siromoney. On equal matrix languages. *Information and Control*, 14:133–151, 1969.

[38] I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the Association for Computing Machinery*, 22(4):499–500, 1975.

[39] I. H. Sudborough. On tape-bounded complexity classes and multi-head finite automata. *Journal of Computer and System Sciences*, 10(1):62–76, 1975.

[40] Y. Takada. Learning even equal matrix languages based on control sets. In A. Nakamura et al., editors, *Parallel Image Analysis, ICPIA'92*, volume 652 of *LNCS*, pages 274–289. Springer, 1992.

[41] Y. Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123:138–145, 1995.

[42] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the Association for Computing Machinery*, 22(2):215–225, 1975.

[43] D. Wood. Properties of n-parallel finite state languages. Technical Report 73/4, McMaster University, Hamilton, Canada, 1973.

[44] D. Wood. Bounded parallelism and regular languages. In G. Rozenberg and A. Salomaa, editors, *L Systems*, volume 15 of *LNCS*, pages 292–301. Springer, 1974.