On the Complexity of Coordinated Table Selective Substitution Systems

Liliana Cojocaru
Faculty of Computer Science
Alexandru Ioan Cuza University
liliana.cojocaru@info.uaic.ro

Abstract

We investigate computational resources used by Turing machines (TMs) and alternating Turing machines (ATMs) to accept languages generated by coordinated table selective substitution systems with two components. We prove that the class of languages generated by real-time (RL; 0S)-systems, an alternative device to generate λ-free labeled marked Petri net languages, can be accepted by nondeterministic TMs in $O(\log n)$ space and $O(n \log n)$ time. Consequently, this proper subclass of Petri nets languages (known also as $L$-languages) is included in $NSPACE(\log n)$. The class of languages generated by (RL; RB)-systems for which the nonterminal alphabet of the RL-grammar is composed of only one symbol and the nonterminal alphabet of the RB-grammar is composed of two symbols, can be accepted by ATMs in $O(\log n)$ time and space. Consequently, this proper subclass of one counter languages generated by one counter machines with only one control state is included in $U_{E^*}$-uniform $NC^1$, hence in $SPACE(\log n)$.

Keywords: coordinated table selective substitution systems, Petri nets, one counter languages, alternating Turing machines, $NSPACE(\log n)$, $NC^1$

1 Introduction

Coordinated table selective substitution systems (cts systems) were introduced in [18] as an alternative framework to simulate grammars and automata. At the basis of cts systems stands the notion of selective substitution grammar (s-grammar) which is a unifying model for several rewriting systems and grammars [17], [18]. The main idea in an s-grammar is to substitute various kinds of rules by only context-free (CF) rules, that at a certain step of derivation must rewrite a certain combination of symbols occurring in the sentential form. This combination is provided by the so called selector of the s-grammar, built in such a way it simulates the respective rewriting formalism. Informally, an s-grammar $G$ is composed of a set of CF productions $P$ and a selector $K$, which is a language over an alphabet $\Sigma \cup \overline{\Sigma}$, where $\Sigma$ is the alphabet of $G$ and $\overline{\Sigma} = \{\overline{a} | a \in \Sigma\}$, $\Sigma \cap \overline{\Sigma} = \emptyset$. A word $y \in \Sigma^*$ is derived from $x \in \Sigma^*$, if and only if there exists $z \in K \subseteq (\Sigma \cup \overline{\Sigma})^*$ such that $x = z$, after replacing all letters in $\overline{\Sigma}$ occurring in $z$ by their unbarred counterparts in $\Sigma$, and each barred letter in $z$
is rewritten in the usual manner by rules in $P$. This is a derivation step in an s-grammar $G$. The language of $G$ is the set of all words derived from a fixed symbol $S$, called the \textit{axiom} of $G$, by iteratively applying the derivation step described above. If $K \subseteq \Sigma^* \Sigma^*$, then $K$ is called \textit{sequential}. The sequential selectors we are concerned with are the \textit{right-boundary selector} $K = \Sigma^* \bar{\Sigma}$ and the \textit{0-sequential selector} $K = \Sigma^* \Sigma \Sigma^*$.

A cts system \cite{1} consists of $n$ s-grammars $G_1, ..., G_n$ and a set $R \subseteq P_1 \times ... \times P_n$ of rewrites. Each $P_i$, $1 \leq i \leq n$, is the set of productions of $G_i$. An $n$-tuple $(y_1, ..., y_n)$ can be derived from $(x_1, ..., x_n)$, where $x_i$ and $y_i$ are words over the alphabet of $G_i$, if there exists a rewrite $r = (r_1, ..., r_n) \in R$ such that each $y_i$ can be derived from $x_i$ in $G_i$ using $r_i$. The language of the cts system consists of all words generated on the first component, such that the system starts the computation with a fixed $n$-tuple of words called the \textit{system axiom}, and ends up the computation in an $n$-tuple composed of empty words excepting the first component (containing the system’s word). Cts systems with one component ($n = 1$) simulate the work of grammars, while cts systems with $n \geq 2$ components simulate automata machines. Cts systems with two coordinates that uses a right-boundary selector, denoted by $(RL; RB)$-systems, are an alternative model for pushdown automata. Cts systems with two coordinates using a 0-sequential selector, denoted by $(RL; 0S)$-systems, are an alternative model for generating Petri nets (PN) languages. The computational power of cts systems and their relation to classes of languages in the Chomsky hierarchy have been studied in \cite{5}, \cite{13}, and \cite{18}. However, in none of these papers the authors investigate the computational complexity of these systems, since this may have interesting consequences on the complexity of some subclasses of languages in the Chomsky hierarchy (e.g. one counter languages). In this paper we try to cover this gap.

We investigate computational resources used by (alternating) Turing machines to accept languages generated by $(RL; 0S)$ and $(RL; RB)$-systems. We prove in Section 3 that real-time $(RL; 0S)$-languages can be accepted by nondeterministic on-line TMs in $O(\log n)$ space. Consequently, the class of languages generated by $\lambda$-free labeled marked PN (known as $\mathcal{L}$-languages \cite{8}) is included in $\text{NSPACE}(\log n)$. This shows the existence of a substantial gap between the general ($\text{EXPSPACE}$-hard) PN languages, known as $\mathcal{L}^0$-languages \cite{8}, and $\mathcal{L}$-languages. The class of \textit{one counter languages} can be generated by $(RL; RB)$-systems for which the nonterminal alphabet of the $RB$-grammar is composed of only two symbols, the axiom and a working symbol. Denote by $(RL; RB_c)$ these systems. In Section 4 we provide an alternative proof, via $(RL; RB_c)$-systems, of the inclusion of one-counter languages in $\text{NSPACE}(\log n)$. If we further restrict the $RL$-grammar to be composed of only one nonterminal then the resulting $(RL; RB)$-systems are equivalent with \textit{one counter machines with one control state}. We prove that the class of languages generated by one state one counter machines can be accepted by indexing ATMs in $O(\log n)$ time and space. Since $\text{ALOGTIME}$, the class of languages recognizable by indexing ATMs in logarithmic time, equals $U_{E^*}\text{-uniform NC}^1$ \cite{2}, \cite{19} we conclude that this proper subclass of one counter languages belongs to $\mathcal{NC}^1$, hence to $\text{SPACE}(\log n)$. In Section 5 we provide several examples of languages generated by $(RL; RB_c)$-systems.
2 Prerequisites

We introduce the basic concepts related to (RL; 0S) and (RL; RB)-systems. We assume
the reader to be familiar with basic notions of automata and formal language theory \[20\].

For an alphabet $\Sigma$, $\Sigma^*$ denotes the free monoid generated by $\Sigma$, $\lambda$ is the empty string, $|x|_a$ denotes the number of occurrences of the letter $a$ in the string $x \in \Sigma^*$, $|\Sigma|$ is the cardinality
of $\Sigma$, while $|x|$ the length of $x \in \Sigma^*$. We denote by $id_\Sigma$ the homomorphism from $(\Sigma \cup \Sigma)^*$ into $\Sigma^*$, defined by $id_\Sigma(a) = a$ and $id_\Sigma(a) = a$, for all $a \in \Sigma$. From \[11\] and \[18\] we have

**Definition 2.1.** (1) A selector over $\Sigma$ is a subset of $(\Sigma \cup \Sigma)^* \Sigma (\Sigma \cup \Sigma)^*$.

(2) A table is a triple $T = (\Sigma, P, K)$, where $\Sigma$ is the alphabet of $T$, $P \subseteq \Sigma \times \Sigma^*$ is a finite nonempty set of productions, and $K$ is a selector over $\Sigma$.

(3) Let $T = (\Sigma, P, K)$ be a table. For $x, y \in \Sigma^*$ we say that $x$ directly derives $y$ in $T$, denoted by $x \Rightarrow_T y$, if $x = x_1 \ldots x_n$, $y = y_1 \ldots y_m$, $n \geq 1$, $x_i \in \Sigma$, $y_i \in \Sigma^*$, $1 \leq i \leq n$, and there exists $z \in \Sigma^*$ such that $id_\Sigma(z) = x$ and, for $1 \leq i \leq n$, if $z_i \in \Sigma$, then $x_i = y_i$, and if $z_i \in \Sigma$, then $(x_i, y_i) \in P$.

**Definition 2.2.** (1) A right-linear grammar (denoted RL-grammar) is a 5-tuple $G = (\Sigma, P, S, \Delta, K)$, where $T = (\Sigma, P, K)$ is the table of $G$, $\Sigma = \text{al}(G)$ is the alphabet of $G$, $\Delta = \text{tal}(G) \subseteq \Sigma$ is the terminal alphabet of $G$, $\Delta = \text{ntal}(G)$ is the nonterminal alphabet of $G$, $S = ax(G) \in \text{ntal}(G)$ is the axiom of $G$, $P = \text{prod}(G)$ is the set of productions in $T$ such that to any $(X, \alpha) \in \text{prod}(T)$ it corresponds a rule of the form $X \rightarrow \alpha$, $X \in \text{ntal}(G)$, $\alpha \in \Sigma \cup \{\lambda\} \cup (\text{tal}(G)\text{ntal}(G))$, $K = (\text{tal}(G))^*(\text{ntal}(G))$.

(2) A 0-sequential grammar (denoted 0S-grammar) or a right-boundary grammar (denoted RB-grammar) is a 4-tuple $G = (\Sigma, P, S, K)$, where $T = (\Sigma, P, K)$ is the table of $G$, $\Sigma = \text{al}(G) = \text{ntal}(G)$ is the alphabet of $G$, $S = ax(G) \in \Sigma$ is the axiom of $G$, $P = \text{prod}(G)$ is the set of productions in $T$ such that to any $(X, Y) \in P$ it corresponds a rule of the form $X \rightarrow Y$, $X \in \Sigma^*$, $Y \in \Sigma^*$, $K = \Sigma^* \Sigma \Sigma^*$ for a 0S-grammar, and respectively $K = \Sigma^* \Sigma$ for an RB-grammar.

**Definition 2.3.** (1) A right-linear 0-sequential system, denoted (RL; 0S)-system, is a triple $G = (G_1, G_2, R)$, where $G_1$ is an RL-grammar, $G_2$ is an 0S-grammar, and $R = \text{rew}(G) \subseteq \text{prod}(G_1) \times \text{prod}(G_2)$ is the set of rewrites of $G$.

(2) A right-linear right-boundary system, abbreviated (RL; RB)-system, is a triple $G = (G_1, G_2, R)$, where $G_1$ is an RL-grammar, $G_2$ is an RB-grammar, and $R = \text{rew}(G) \subseteq \text{prod}(G_1) \times \text{prod}(G_2)$ is the set of rewrites of $G$.

**Definition 2.4.** Let $G = (G_1, G_2, R)$ be an X-system, $X \in \{(RL; 0S), (RL; RB)\}$, and $x = (x_1, x_2)$, $y = (y_1, y_2) \in (\text{al}(G_1))^* \times (\text{al}(G_2))^*$.

(1) We say that $x$ directly derives $y$ in $G$ by using $r = (r_1, r_2) \in \text{rew}(G)$, denoted by $x \Rightarrow_T^* G y$, if $x_1 \Rightarrow_{G_1}^* y_1$, and $x_2 \Rightarrow_{G_2}^* y_2$. Let $\Rightarrow_T^* G$ be the reflexive and transitive closure of $\Rightarrow_T G$. If $x \Rightarrow_T^* G y$, then we say that $y$ is derived from $x$ in $G$.

(2) Let $(x, y) \in (\text{al}(G_1))^+ \times (\text{al}(G_2))^+$. An $(x, y)$-derivation in $G$ is a sequence $\rho = \rho(0), \ldots, \rho(n)$ of elements in $(\text{al}(G_1))^+ \times (\text{al}(G_2))^+$, $n \geq 0$, such that $\rho(0) = (x, y)$ and for $1 \leq i \leq n$, $\rho(i-1) \Rightarrow_G \rho(i)$. In the sequel $\rho$ is called an n-step derivation and $\rho(i-1) \Rightarrow_G \rho(i)$ is called the ith step derivation of $\rho$. Each $\rho(i)$, $0 \leq i \leq n$, is called a snapshot of $\rho$. 

3
(3) Let \( \rho = \rho(0), ..., \rho(n) \), \( n \geq 0 \), be a derivation in \( G \). We say that \( \rho \) is a successful derivation if \( \rho(0) = (ax(G_1), ax(G_2)) \) and \( \rho(n) = (w, \lambda) \) where \( w \in \Delta^* \).

(4) The language generated by \( G \) is defined as \( L(G) = \{ w \in (tal(G_1))^*(ax(G_1), ax(G_2)) \Rightarrow_{\beta}^* (w, \lambda) \} \). \( L(G) \) is referred to as an \( X \)-language, \( X \in \{(RL;0S), (RL;RB)\} \).

The class of all \( X \)-languages is denoted by \( L(X) \), \( X \in \{(RL;0S), (RL;RB)\} \). The strings \( x_1, y_1 \in (al(G_1))^* \), \( x_2, y_2 \in (al(G_2))^* \) (in Definition 2.4) are called sentential forms (s.forms) of the \( RL \)-, \( 0S \)- and respectively, \( RB \)-grammars.

An \( RL \)-grammar \( G \) is real-time if \( prod(G) \) contains no chain rules, i.e., rules of the form \( X \rightarrow Y \), where \( X, Y \in ntal(G_1) \). Let \( G = (G_1, G_2, R) \) be an \( X \)-system. \( G \) is real-time if \( G_1 \) is real-time. The class of real-time \( X \)-languages is denoted by \( \mathcal{L}_{rt}(X) \), \( X \in \{(RL;0S), (RL;RB)\} \).

Henceforth, in any reference to an \( X \)-system \( G = (G_1, G_2, R) \), \( X \in \{(RL;0S), (RL;RB)\} \), \( R = rew(G) = \{r_1, ..., r_k\} \) is considered to be the ordered set of productions in \( prod(G_1) \times prod(G_2) \), \( A_1 = ax(G_1) \), \( B_1 = ax(G_2) \), \( ntal(G_1) = \{A_1, ..., A_k\} \) and \( al(G_2) = ntal(G_2) = \{B_1, ..., B_k\} \) the ordered set of nonterminals in \( G_1 \) and \( G_2 \), respectively. Each \( r_i \in rew(G) \), \( 1 \leq i \leq k \), is a pair of the form \( (r_i, G_1), (r_i, G_2) \), where \( r_i, G_1 \in prod(G_1) \) and \( r_i, G_2 \in prod(G_2) \). Each rule \( r_i, G_1 \) is of the form \( \alpha_{r_i, G_1} \rightarrow \beta_{r_i, G_1}, \alpha_{r_i, G_1} \in ntal(G_1) \) and \( \beta_{r_i, G_1} \in (tal(G_1) \cup \{\lambda\})(ntal(G_1) \cup \{\lambda\)) \). Each \( r_i, G_2 \) is of the form \( \alpha_{r_i, G_2} \rightarrow \beta_{r_i, G_2}, \alpha_{r_i, G_2} \in ntal(G_2) \) and \( \beta_{r_i, G_2} \in (ntal(G_2))^* \). The net effect of rule \( r_i, G_2 \), with respect to \( \beta_1 \in ntal(G_2) \), is provided by \( df_{B_1}(r_i,G_2) = |\beta_{r_i,G_2}|_B - |\alpha_{r_i,G_2}|_{B_1} \), \( 1 \leq l \leq 2 \). To each rule \( r_i, G_2 \in prod(G_2) \) we associate a vector \( V(r_i,G_2) \in \mathbb{Z}^2 \) defined by \( V(r_i,G_2) = (df_{B_1}(r_i,G_2), df_{B_2}(r_i,G_2), ..., df_{B_2}(r_i,G_2)) \).

3 On the Complexity of \((RL;0S)\)-Systems and Petri Nets

Petri Nets (PN) [8], [12], [16], equivalent with Vector Addition Systems with States (VASS) [11], are a language modeling framework to simulate reactive and distributed systems, with practical applications in concurrent business processes and economy, in chemical or biological processes. Formally we have

**Definition 3.1.** A labeled marked Petri net, abbreviated \( \text{lmPN} \), is a 6-tuple \( P = (P, T, F, \Sigma, l, M_0) \), where \( P \) is the set of places, \( T \) is the set of transitions, \( F \subseteq (P \times T) \cup (T \times P) \) is the flow relation, \( \Sigma \) is the alphabet, \( l \) is the labeling function \( l : T \rightarrow \Sigma \cup \{\lambda\} \) that assigns to each transition either a letter or \( \lambda \), and \( M_0 \) is the initial marking (a marking of a PN is a function from \( P \) to \( \mathbb{N} \)). If to a \( \text{lmPN} \) we associate a finite set of (final) markings then the resulted PN is called a labeled marked Petri net with final markings, abbreviated \( \text{lmPNf} \). If \( l(t) \neq \lambda \) for each \( t \in T \), then \( P \) is a \( \lambda \)-free \( \text{lmPNf} \).

The language generated by a \( \lambda \)-free \( \text{lmPNf} \) is the set of all labeled firing-sequences of \( P \) from \( M_0 \) to any marking. The corresponding class of languages is denoted in [8] and [12] by \( L \). The language generated by a \( \text{lmPNf} \) is the set of all labeled firing-sequences of \( P \) from \( M_0 \) to some final markings. The class of languages generated by \( \text{lmPNf} \) with final markings is denoted in [8] by \( L_0^\lambda \). From [8] we have \( L \subset L_0^\lambda \). The membership problem for \( L_0^\lambda \) is many-one equivalent to the reachability problem for VASS [8], which was proved to be \( \text{EXPSPACE-hard} \) [13], as a lower bound. Nowadays this problem is known to have a
non-elementary complexity \cite{21}, i.e., a tower of exponentials of time of height depending elementarily on the size of the input, as lower bound provided by TOWER = \text{F}_3 \cite{3}, and a non-primitive recursive upper bound provided by ACKERMANN = \text{F}_\omega \cite{14}. On the other hand the reachability problem for 1-VASS (VASS in dimension 1-VASS is \text{NP}-complete \cite{1}), i.e., a tower of exponentials of time of height depending non-elementary complexity, \cite{TOWER}. If binary encoding is used, then the reachability problem for 1-VASS is \text{NP}-complete \cite{10}. We prove that, in spite of the \mathcal{L}_\lambda^\omega \text{s non-elementary complexity, the membership problem of } \mathcal{L}, \text{ in binary encoding, has a sub-linear space complexity, i.e., } \mathcal{L} \subseteq \text{NSPACE}(\log n). \text{ We have }

\textbf{Theorem 3.2.} Each language } L \in \mathcal{L}_{rt}(RL; 0S) \text{ can be recognized by an on-line nondeterministic Turing machine in } \mathcal{O}(\log n) \text{ space and } \mathcal{O}(n \log n) \text{ time (i.e., } \mathcal{L}_{rt}(RL; 0S) \subseteq \text{NSPACE}(\log n)\text{).}

\textbf{Proof.} \text{ Let } G = (G_1, G_2, R) \text{ be a real-time (RL; 0S)-system and } L(G) \text{ the language generated by } G. \text{ Let } T \text{ be an on-line nondeterministic TM (with stationary positions) composed of an input tape that stores an input } w \in (\text{tal}(G_1))^*, \text{ } w = a_1 a_2 ... a_n \text{ of length } n, \text{ and one (read-write) working tape.}

\text{From an initial state } q_0, \text{ in stationary positions, } T \text{ records (by using constant time, related to the length of the input) the net effects of the rules in } prod(G_2), \text{ and it books } l_2 \text{ Parikh blocks } B_l, 1 \leq l \leq l_2, \text{ of } \mathcal{O}(\log n) \text{ cells (to record the Parikh vector of the s.forms of } G_2). \text{ At the beginning of the computation, on the first cell of each } B_l, \text{ } T \text{ records the Parikh vector } V_{G_2}^0 \text{ of the axiom } B_1 = ax(G_2), \text{ i.e., } V_{G_2,B_1}^0 = 1 \text{ and } V_{G_2,B_l}^0 = 0, 2 \leq l \leq l_2. \text{ Denote by } q_s \text{ the state reached at the end of this procedure. In brief, from now on } T \text{ proceeds as follows. For each scanned symbol } a_i, \text{ } T \text{ checks whether there exists at least one rule in } \text{rew}(G) \text{ such that the rule in } G_1 \text{ rewrites the single nonterminal existing in the s.form of } G_1, \text{ producing } a_i, \text{ and whether the nonterminal in the left-hand side of the rule in } G_2 \text{ exists in the s.form of } G_2. \text{ } T \text{ memorizes in states the information regarding the rules applied at this step and updates the Parikh vector of the s.form of } G_2. \text{ It accepts the input if } T \text{ reaches the end of } w \text{ with no nonterminals in the s.forms of } G.

\text{Formally, } T \text{ works as follows. From } q_s, \text{ } T \text{ enters in a state } q_{a_1,X_1}^{a_1,X_1} \text{ by reading } a_1, \text{ where } r_{A_1}^{a_1,X_1} = (r_{A_1,G_1}, \mathcal{L}_{G_2}) \text{ and } r_{A_1,G_1}^{a_1,X_1} \text{ is a rule of the form } A_1 \rightarrow a_1 X_1. \text{ As } X_1 \text{ may vary, there may exist several states of this type (the nondeterminism). If there is no rule } r_{A_1}^{a_1,X_1} \in \text{rew}(G) \text{ then the computation is blocked, hence } w \text{ is not accepted. On } q_{a_1,X_1}^{a_1,X_1} \text{ } T \text{ checks (in stationary positions) whether } r_{G_2}^{a_1,X_1} \text{ rewrites } B_1. \text{ If this holds, } T \text{ adds the net effect of } \mathcal{L}_{G_2}, \text{ with respect to } B_l, \text{ to } V_{G_2,B_l}^0, 1 \leq l \leq l_2, \text{ i.e., } T \text{ computes, in binary, } \text{sd}f_{(1,G_2)} = V_{G_2,B_l}^0 + \text{df}_{(1,G_2)}(\mathcal{L}_{G_2}). \text{ The binary value of } \text{sd}f_{(1,G_2)} \text{ is recorded on the } l^{th} \text{ Parikh block of } G_2. \text{ From state } q_{a_1,X_1}^{a_1,X_1} \text{, } T \text{ enters in a state of the form } q_{a_2,X_2}^{a_1,X_1} \text{ by reading } a_2, \text{ where } r_{X_1}^{a_2,X_2} = (r_{X_1,G_1}, \mathcal{L}_{G_2}), r_{X_1,G_1}^{a_2,X_2} \text{ is of the form } X_1 \rightarrow a_2 X_2, \text{ and } \mathcal{L}_{G_2} \text{ is the rule in } prod(G_2) \text{ such that } r_{X_1}^{a_2,X_2} \in \text{rew}(G). \text{ If there is no rule in } prod(G_1) \text{ that rewrites } X_1, \text{ then the}

\footnote{1}{By dimension of a VASS it is understood the number of counters of VASS.}

\footnote{2}{Otherwise, i.e., \mathcal{L}_{G_2} \text{ does not rewrite } B_1, \text{ then the computation is blocked on this state, but may continue on another state of type } q_{a_1,X_1}^{a_1,X_1}, \text{ for a different } X_1.}
computation is blocked. On \( q_{_{X_1 \rightarrow X_2}, G_2} \) \( \mathcal{T} \) checks whether the nonterminal rewritten by \( \alpha_{_{G_2}} \) exists in the s.form of \( G_2 \), i.e., whether \( \text{sd}f_{_{\alpha_{_{G_2}}}}(t) > 0 \), where \( \alpha_{_{G_2}} \) is the left-hand side of \( \text{sd}f_{_{G_2}} \). If it holds, \( \mathcal{T} \) adds the net effect of \( \text{sd}f_{_{G_2}} \) with respect to \( B_l \), to \( \text{sd}f_{_{B_l}}(t) = \text{sd}f_{_{B_l}}(t) + \text{df}_{_{\mathcal{T}}}(t) \). The space used by \( \mathcal{T} \) to record \( \text{sd}f_{_{B_l}}(t) \) is now used to record \( \text{sd}f_{_{B_l}}(t) \).

The computation continues in this way until all symbols \( a_i \), \( 1 \leq i \leq n \), are read and checked as explained above. Besides, for the last symbol \( a_n \), \( \mathcal{T} \) also checks whether the s.form of \( G_2 \) contains no nonterminal, i.e., \( \text{sd}f_{_{B_l}}(n, G_2) = \text{sd}f_{_{B_l}}(n-1, G_2) + \text{df}_{_{\mathcal{T}}}(t) = 0, 1 \leq l \leq l_2 \).

The input is accepted if for each \( a_i \), \( 1 \leq i \leq n \), there exists at least one state of the form \( q_{_{X_i \rightarrow X_i}, G_2} \), where \( r_{_{X_i \rightarrow X_i, G_2}} = (r_{_{X_i \rightarrow X_i, G_2}}) \in \text{rew}(G) \), \( r_{_{X_i \rightarrow X_i, G_2}} \) is of the form \( X_{i-1} \rightarrow a_i X_i \), respectively \( X_{n-1} \rightarrow a_n \). \( \text{sd}f_{_{G_2}} \) rewrites an arbitrary occurrence of \( \alpha_{_{G_2}} \) in the s.form of \( G_2 \), and at the end of computation \( \mathcal{T} \) reaches a final state with no nonterminal in the s.forms of \( G \).

Recall that \( w \in L(G) \) if for each snapshot there exists a strategy of marking a symbol in the s.form of \( G_2 \) such that, rewriting this symbol, at the end of computation, the s.form of \( G_2 \) equals \( \lambda \). Since \( K = \Sigma^* \Sigma^* \), it is evident that the position of the marked nonterminal (hence, the order in which the nonterminals are rewritten) in the s.form of \( G_2 \) is not important. There is no context that may control the marked symbol. Hence, the condition for a rule \( \alpha_{_{G_2}} \), \( 2 \leq i \leq n \), to rewrite an arbitrary occurrence of \( \alpha_{_{G_2}} \) suffices, as long as this nonterminal exists in the s.form of \( G_2 \) (checked by the condition \( \text{sd}f_{_{\alpha_{_{G_2}}}}(t) > 0 \)).

Since \( G_2 \) is a CF grammar, the length of each s.form in \( G_2 \) is linearly bounded by the length of the input. Hence, \( \mathcal{O}(\log n) \) cells are enough in order to record the number of occurrences of each nonterminal \( B_l \), \( 1 \leq l \leq l_2 \), on the s.form of \( G_2 \), and the space used by \( \mathcal{T} \) is \( \mathcal{O}(\log n) \). Because each time, reading an input symbol, \( \mathcal{T} \) has to visit \( \mathcal{O}(\log n) \) cells in the working tape, and the constant number of auxiliary (summation) operations with binary numbers, performed at each step of derivation in \( G \), require \( \mathcal{O}(\log n) \) time, \( \mathcal{T} \) performs the whole computation in \( \mathcal{O}(n \log n) \) time. Hence \( L_{rt}(RL; 0S) \subseteq \NSPACE(\log n) \).

In [4] it is proved that the class of languages generated by real-time \( (RL; 0S) \)-systems is equal with the class of languages generated by \( \lambda \)-free labeled marked Petri nets, denoted in [5] as \( \mathcal{L} \). Hence, from Theorem 3.2 we have

**Corollary 3.3.** \( \mathcal{L} = L_{rt}(RL; 0S) \subseteq \NSPACE(\log n) \).

### 4 On the Complexity of \((RL; RB)\)-Systems and One Counter Machines

\((RL; RB)\)-systems are alternative models of pushdown automata [4], [5], [6]. Pushdown automata accept exactly the CF languages. If the pushdown alphabet is restricted to only two symbols, a symbol \( S_2 \) to mark the bottom of the stack and a working symbol \( Z_2 \), then the resulting machine is a one counter automaton (OCA). A counter can be increased by

---

3Recall that \( \alpha_{_{G_2}} \) is the nonterminal placed on the left-hand side of rule \( \alpha_{_{G_2}} \).
1, decreased by 1, left unchanged, or tested for 0 (the bottom marker test). The class of languages accepted by OCAs, denoted by \( \mathcal{L}(OC) \), is a proper subset of CF languages [9]. If OCAs are not allowed to perform 0-tests then the resulting machines are called one counter nets automata (OCN). The class of languages accepted by OCN are denoted by \( \mathcal{L}(OCN) \). It is known that \( \mathcal{L}(OCN) = \mathcal{L}(1-\text{VASS}) \subset \mathcal{L}(OC) \). We prove that the membership problem for some particular subclasses of \( \mathcal{L}(OC) \) and \( \mathcal{L}(OCN) \) are in \( \mathcal{NC}^1 \).

**Definition 4.1.** (1) A cts-system \( G = (G_1, G_2, R) \) be an \( (RL; RB_c) \)-counter system, denoted by \( (RL; RB_c) \), if \( G_1 \) is an \( RL \)-grammar, \( G_2 \) is an \( RB \)-grammar, \( al(G_2) = ntal(G_2) = \{S_2, Z_2\} \), and \( prod(G_2) \subseteq \{S_2 \rightarrow S_2, S_2 \rightarrow S_2Z_2, Z_2 \rightarrow Z_2, Z_2 \rightarrow Z_2Z_2, Z_2 \rightarrow \lambda, S_2 \rightarrow \lambda\} \).

(2) A cts-system \( G = (G_1, G_2, R) \) is an \( (RL; RB_c) \)-counter nets system, denoted by \( (RL^0; RB_c) \), if \( G_1 \) is an \( RL \)-grammar, \( G_2 \) is an \( RB \)-grammar, \( al(G_2) = ntal(G_2) = \{Z_2\} \), and \( prod(G_2) \subseteq \{Z_2 \rightarrow Z_2, Z_2 \rightarrow Z_2Z_2, Z_2 \rightarrow \lambda\} \).

From [18] we have \( \mathcal{L}(RL;RB_c) = \mathcal{L}(OC) \) and \( \mathcal{L}(RL^0;RB_c) = \mathcal{L}(OCN) \).

**Definition 4.2.** Let \( G = (G_1, G_2, R) \) be an \( (RL; RB_c) \)-system. The state diagram \( D_G = (V, E) \) of \( G \), is a directed graph in which \( V = \{(X, Y)|X \in ntal(G_1), Y \in ntal(G_2)\} \cup \{F\} \) is the set of nodes. The nodes labeled by \((S_1, S_2)\) and by \( F \) are the initial node, and respectively, the final node of \( D_G \). Let \( r = (r_{G_1}, r_{G_2}) \) be an arbitrary rule in \( R \), the set \( E \) of edges is built as follows.

- If \( r_{G_1} \) is of the form \( X \rightarrow xY, x \in tal(G_1) \cup \{\lambda\}, X, Y \in ntal(G_1) \), and \( r_{G_2} \) is of the form \( S \rightarrow S, S \in \{S_2, Z_2\} \), then \( E \) contains an edge, labeled by \( x/0 \), from the node labeled by \((X, S)\) to the node labeled by \((Y, S)\).

- If \( r_{G_1} \) is of the form \( X \rightarrow xY, x \in tal(G_1) \cup \{\lambda\}, X, Y \in ntal(G_1) \) and \( r_{G_2} \) is of the form \( S \rightarrow S, S \in \{S_2, Z_2\} \), then \( E \) contains an edge, labeled by \( x/+ \), from the node labeled by \((X, S)\) to the node labeled by \((Y, Z_2)\).

- If \( r_{G_1} \) is of the form \( X \rightarrow xY, x \in tal(G_1) \cup \{\lambda\}, X, Y \in ntal(G_1) \) and \( r_{G_2} \) is of the form \( Z_2 \rightarrow \lambda \), then \( E \) contains two edges, labeled by \( x/- \), one from the node labeled by \((X, Z_2)\) to the node labeled by \((Y, Z_2)\), and the other one from \((X, Z_2)\) to the node labeled by \((Y, S_2)\).

- If \( r_{G_1} \) is of the form \( X \rightarrow x, x \in tal(G_1) \cup \{\lambda\}, X \in ntal(G_1) \) and \( r_{G_2} \) is of the form \( S \rightarrow S, S \in \{S_2, Z_2\} \), then \( E \) contains an edge, labeled by \( x \), from the node labeled by \((X, S_2)\) to a final node labeled by \( F \).

Next we prove that walking through the state diagram \( D_G \), from the initial node to the final node, with an additional counting procedure for the number of \( Z_2 \)’s existing in the s.form of \( G_2 \), we can simulate the work of \( G \) by using (nondeterministic) logarithmic space. More precisely, we have

**Theorem 4.3.** Each \( L \in \mathcal{L}(RL; RB_c) \) can be recognized by a nondeterministic TM in \( O(\log n) \) space and \( O(n^k \log n) \) time \( (\mathcal{L}(RL; RB_c) \subseteq \text{NSPACE}(\log n)) \).

**Proof.** Let \( G = (G_1, G_2, R) \) be an \( (RL; RB_c) \)-system with \( |ntal(G_1)| \geq 2 \), and \( D_G \) its state diagram. Let \( T \) be an on-line nondeterministic TM (with stationary positions) composed of an input tape that stores an input \( w \in (tal(G_1))^*, w = x_1x_2...x_n \), of length \( n \), and one
read-write $O(\log n)$-space restricted working tape used to store in binary the number $n_z$ of $G_2$'s occurring in the s.form of $G_2$ at any snapshot. At the beginning of the computation $n_z$ is set to 0. Each node $(X,Y) \in V$ corresponds to a state $q_{(X,Y)}$ of $T$, denoted by $q_{(X,Y)}$. Besides, $T$ has a particular state $q_5$ in which it enters each time $T$ reaches a state $q_{(X,S_2)}$ with a non-empty counter ($n_z > 0$). In $q_5$, the $\delta$ (transition) relation is left undefined.

Simulating $G$, $T$ starts the computation in $q_{(S_1,S_2)}$ and nondeterministically applies one of the rules that rewrites $(S_1,S_2)$, i.e., from $q_{(S_1,S_2)}$ $T$ enters in one of the states $q_{(X,Y)}$ such that $E$ contains the edge $(S_1,S_2) - (X,Y)$. When passing from $q_{(S_1,S_2)}$ to $q_{(X,Y)}$, $T$ reads $x$ from the input tape or reads no symbol if $x = \lambda$, where $x$ is the label of the edge $(S_1,S_2) - (X,Y)$. Besides, if $(S_1,S_2) - (X,Y)$ is labeled by $+/−/0$ then $n_z$ is increased by $1/0$ or by $1/0$. From state $q_{(X,Y)}$, $T$ enters in the state $q_{(X',Y')}$ such that $(X,Y) - (X',Y') \in E$, it reads $x$ from the input tape, where $x$ is the label of the edge $(X,Y) - (X',Y')$, and increases by $1/0$ or by $1/0$. The computation continues in this way until all symbols from the input tape are read. The input $w$ is accepted if at the end of computation $T$ enters in the final state $q_F$ with no $Z_2$'s on the s.form of $G_2$ (i.e., $n_z = 0$) and the input head placed on a rightmost delimiter of the input (e.g. the blank symbol).

Note that walking on $D_G$ one can reach again the node $(S_1,S_2)$. If at this point $n_z > 0$ then the computation is blocked due to the existence of state $q_5$. If $n_z = 0$ (i.e., the bottom of the stack is reached) and there exists a rule of the form $S_1 \rightarrow x$, $x \in \text{tail}(G_1) \cup \{\lambda\}$, then from $q_{(S_1,S_2)}$, $T$ enters in the state $q_F$, and it accepts the input only by reading the blank symbol, otherwise the computation is blocked. If $S_1 \rightarrow x$ does not exist, $T$ may continue with a rule of the form $S_1 \rightarrow xX$, by recharging the counter. Furthermore, the simulation allows recharging of the counter each time $T$ gets into a state of the form $q_{(X,S_2)}$, with $n_z = 0$, and there exists a rule of the form $X \rightarrow xY$ and unread symbols on the input tape. If $T$ reaches $q_{(X,S_2)}$ with a nonempty counter ($n_z > 0$) then again the computation is blocked due to state $q_5$.

The time used by $T$ to perform the computation is $O(n^k\log n)$, where $k \geq 1$ is a constant. This is because when $G_1$ contains no chain rules, $n_z$ (recorded in binary) is linearly bounded by the length of the input (as $G_2$ is a CF grammar). The time complexity may increase for the case when the $(RL; RB_c)$-system (through $G_1$) contains chain rules. In this situation, if all chain rules are invoked in loops composed of at least one non-chain rule, then at each run of such a loop at least one symbol is read. Hence, the length of computation depends linearly on the length of the input. Certainly, the same situation occurs when chain rules are sparsely applied during a derivation (in the sense that none of them is used during the execution of a loop). If all chain rules are invoked in loops composed only of chain rules, then the running of such a loop is overflowing for the counter and useless for the computation. If the $Z_2$’s symbols, added through such loops, are deleted by non-chain rules, i.e., meantime the system scans some input symbols, then the running time depends linearly on the length of the input. Otherwise, if the $Z_2$’s symbols are deleted

---

4The structure of $D_G$ is depicted by the manner in which the transition relation $\delta$ of $T$ is defined. Hence, no other recordings on the working tape are needed.
only by chain rules, then this loops are useless for computation and they can be forced to run of a polynomial number of times by imposing $O(\log n)$ restriction on the working tape (without altering the language). Hence, the number of $Z_2$’s gathered on the s.form of $G_2$, can be at most polynomial. The time complexity is $O(n^k \log n)$, $k \geq 1$, while the space complexity is $O(\log n)$.

It is widely known that $\mathcal{L}(\text{OC})$ is $\text{NSPACE}(\log n)$-complete [22]. Therefore, the inclusion $\mathcal{L}(\text{OC}) \subseteq \text{SPACE}(\log n)$ tracks the equality $\text{SPACE}(\log n) = \text{NSPACE}(\log n)$. Both these statements are still open problems in complexity theory, and therefore a $\text{SPACE}(\log n)$ upper bound for the whole class of $\mathcal{L}(\text{OC})$ remains a challenging problem. Next we prove that an $\text{NC}^1$, hence a $\text{SPACE}(\log n)$, upper bound can be obtained for languages generated by $(RL; RB_c)$-systems for which $\text{ntal}(G_1)$ is restricted to only one nonterminal. Denote by $(RL_1; RB_c)$ these particular $(RL; RB_c)$-systems and by $\mathcal{L}(\text{OC}_1)$ the class of one counter languages generated by OCA’s with only one control state. We have $\mathcal{L}(RL_1; RB_c) = \mathcal{L}(\text{OC}_1)$.

An indexing ATM is an ATM composed of an input tape, $k$ working tapes ($k \geq 1$), and an index tape. The indexing ATM has $k + 1$ heads corresponding to the working tapes and the index tape. There is no need of a head for the input tape, since the indexing ATM reads a symbol from the input tape, through the index tape as follows. It writes an integer $i$, $i \leq n$, in binary, on the index tape, where $n$ is the length of the input, and enters in a state requiring the symbol placed at the $i^{th}$ location on the input tape to be visible to the machine. From the initial state or a given node in the computation tree of the ATM, an indexing ATM may proceed either with an universal configuration (of which state is an universal state) or with an existential configuration (of which state is an existential state).

Since an ATM is a non-deterministic machine, there will be one or more such universal (or existential) configurations placed at the same level in the computation tree. All edges going from a certain node to universal (existential) configurations, placed at the same level, are called universal (existential) branches. Sometimes, when in an universal (existential) branch the ATM makes some more complex operations (such as simulating another ATM), we call the corresponding branch universal (existential) process.

The Nick class ($\text{NC}$) is defined as the class of all functions computable by a family of uniform Boolean circuits with polynomial size and depth bounded by a polynomial in $\log n$. For each integer $i$, we denote by $\text{NC}^i$ the class of functions computable by polynomial size Boolean circuits with depth $\mathcal{O}(\log^i n)$ and fan-in two. We have $\text{NC}^i \subseteq \text{NC}^{i+1}$, $i \geq 1$, and $\text{NC} = \bigcup_{i \geq 1} \text{NC}^i$. Depending on the type of the uniformity restrictions, i.e., speedups on time and space imposed on (alternating) TMs simulating a family of circuits, we obtain several $\text{NC}$ “uniform” classes. The uniformity condition on which we are interested in this paper is the $U_{E_{\log n}}$-uniformity. It concerns logarithmic restrictions on the time and space needed by an ATM to simulate a family of circuits of size $s(n)$ and depth $t(n)$ [19]. Depending on the type of the uniformity, several characterizations of the $\text{NC}$ class in terms of ATM resources are presented in [19]. For $i \geq 2$, all $\text{NC}^i$ classes behave similarly, no matter which uniformity restriction is imposed on circuits, i.e., $\text{NC}^i = \text{ASPACE}, \text{TIME}(\log n, \log^i n)$, where $\text{ASPACE}, \text{TIME}(s(n), t(n))$ denotes the class of languages acceptable by ATMs in simultaneous space $s(n)$ and time $t(n)$. For $i = 1$ this equality holds only for the $U_{E_{\log n}}$-uniformity, more precisely, $U_{E_{\log n}}-\text{uniform} \text{NC}^1 = \text{ASPACE}, \text{TIME}(\log n, \log n) = \text{ALOGTIME}$. Fur-
thermore, $NC^1 \subseteq SPACE(\log n) \subseteq NSPACE(\log n)$ [2]. Recall that an $(RL_1; RB_c)$-counter system performs a zero-test (0-test) each time the $RB_c$-grammar rewrites $S_2$ after some deletions of $Z_2$'s. After a 0-test the counter can be recharged. We have

**Theorem 4.4.** Each language $L \in \mathcal{L}(RL_1; RB_c)$ can be recognized by an indexing ATM in $O(\log n)$ time and space (i.e., $\mathcal{L}(RL_1; RB_c) \subseteq \text{ALOGTIME}$).

*Proof.* Let $G = (G_1, G_2, R)$ be an $(RL_1; RB_c)$-system, $L(G)$ the language generated by $G$, $\text{tal}(G_1) = \{a_1, \ldots, a_m\}$, $\text{ntal}(G_1) = \{S_1\}$, and $\text{ntal}(G_2) = \{S_2, Z_2\}$. $R = \text{rew}(G)$ can be composed of the following rules (where $x \in \text{tal}(G_1)$)

$$S = \begin{cases} 
\psi_1^x = (S_1 \rightarrow xS_1, S_2 \rightarrow S_2), & \psi_2^x = (S_1 \rightarrow xS_1, S_2 \rightarrow S_2Z_2), \\
\psi_3^x = (S_1 \rightarrow xS_1, Z_2 \rightarrow Z_2Z_2), & \psi_4^x = (S_1 \rightarrow xS_1, Z_2 \rightarrow Z_2), \\
\psi_5^x = (S_1 \rightarrow xS_1, Z_2 \rightarrow \lambda), & \psi_6^x = (S_1 \rightarrow x, S_2 \rightarrow \lambda), \\
\psi_7 = (S_1 \rightarrow S_1, S_2 \rightarrow S_2Z_2), & \psi_8 = (S_1 \rightarrow S_1, Z_2 \rightarrow Z_2Z_2), \\
\psi_9 = (S_1 \rightarrow S_1, Z_2 \rightarrow \lambda), & \psi_{10} = (S_1 \rightarrow \lambda, S_2 \rightarrow \lambda). 
\end{cases}$$

Let $A$ be an ATM composed of an index tape, a (read-write) working tape divided into three blocks $B_1$, $B_2$, and $B_3$, and an input tape that stores an input word $w \in (\text{tal}(G_1))^*$, $w = x_1\ldots x_n$, of length $n$. The first block $B_1$, of constant size (related to the length of the input) is used to store the data concerning the types of the rules in $R$, the second block $B_2$, of $O(\log n)$ size, is used to record in binary the number $n_z$ of $Z_2$'s existing on the s.form of $G_2$ at any snapshot, while the last block $B_3$ is used for auxiliary computations.

In brief, we eliminate in turn some of the rules in $S$ and we study whether the rest of the rules form a valid $(RL_1; RB_c)$-system. For a valid $(RL_1; RB_c)$-system we prove that $w \in (\text{tal}(G_1))^*$ can be accepted by $A$ in logarithmic time and space. We consider only those $(RL_1; RB_c)$-systems that may rise interesting computations, depending on the types of the rules that compose $R = \text{rew}(G)$.

Any successful derivation in a “general” $(RL_1; RB_c)$-system runs upon two main “subroutines”. Rules applied on the first subroutine cannot be applied on the second one. On the first subroutine the system applies, arbitrarily many times, a rule of type $\psi_1^x$. Applying one time a rule of type $\psi_6^x$ or $\psi_{10}$ the system ends up with a successful derivation, or it may enter in the second subroutine by applying (one time) a rule of type $\psi_2^x$ or $\psi_7$. Now it can apply, arbitrarily many times, rules of the forms $\psi_3^x$, $\psi_4^x$, $\psi_8$, $\psi_9$. In a successful derivation the second subroutine must be canceled by the application of a rule of type $\psi_5^x$ or $\psi_9$, to delete the last $Z_2$ symbol existing on the s.form of $G_2$ (the counter stack) performing a 0-test. Then it may finish the computation (by using a rule of type $\psi_6^x$ or $\psi_{10}$) or it may recharge the counter, by passing either into the first subroutine or into the second one. In the sequel, rules of types $\psi_8$ and $\psi_9$ are called correction rules, because they do not produce terminals, but they add more $Z_2$'s on the s.form of $G_2$ (to allow applications of non-chain rules in $G_1$ associated with deleting rules in $G_2$) or they delete an exceeding number of $Z_2$'s.

In stationary positions $A$ records on $B_1$ the rules in $R$, while $n_z$ is set to 0 and recorded on $B_2$. In an existential state $A$ guesses the length of $w$, i.e., writes on the index tape $n$ in binary, and checks whether the $n^{th}$ cell of the input tape contains a symbol in $\text{tal}(G_1)$ and the cell $n + 1$ contains no symbol.
Denote by \( \varphi_i \) the \((RL_1;\ RB_c)\)-system composed of some rules described in \( S \). There are finitely many such systems and most of them are invalid (do not generate a language). Next we discuss the solution for the most important ones.

- \( \varphi_1 \): \((\psi_1^1, \psi_1^2)\) is an \((RL_1;\ RB_c)\)-system composed of rules of types \( \psi_1^1 \) and \( \psi_1^2 \). \( A \) checks whether \( R \) contains a rule of type \( \psi_1^2 \) that produces \( x_n = x \), and whether for each \( x_i \) occurring in \( x_1x_2...x_{n-1} \), \( R \) contains the rule \( \psi_1^{x_i} \) that produces \( x_i \). These can be done by universally branching all \( x_i \), \( 1 \leq i \leq n \).

- \( \varphi_2 \): \((\psi_1^1, \psi_2)\) \( A \) universally checks whether for each terminal \( x_i \) occurring in \( x_1x_2...x_{n} \) there exists in \( R \) a rule of the form \( \psi_1^{x_i} \), \( 1 \leq i \leq n \).

- \( \varphi_3 \): \((\psi_1^3, \psi_1^4, \psi_5, \psi_6, \psi_7)\) The case when each \( x \) occurring in \( w \) is produced in the same time by \( \psi_3^3 \) and \( \psi_5^5 \). \( A \) checks whether \( I \). Each \( x_i \) in \( w \) is produced (simultaneously) by rules of types \( \psi_3^3 \) and \( \psi_5^5 \), \( 1 \leq i \leq n \) (\( n \) universal branches).

If \( n \) is an odd number then a strategy that \( G \) may choose, to derive \( w \), is to apply first rules \( \psi_5^5 \) of \( \frac{n-1}{2} \) times, and then rules \( \psi_5^5 \) of \( \frac{n+1}{2} + 1 \) times (in order to delete all \( Z_2 \)'s from the s.form of \( G_2 \)). It ends the derivation with rule \( \psi_{10} \).

If \( n \) is even then a strategy that \( G \) may choose is to apply alternatively the rules \( \psi_5^5 \) and \( \psi_7 \), performing \( n \) zero-tests and ending with \( \psi_{10} \). Hence, any combination of terminals in \( \tau(G_1) \) that satisfy \( I \) is accepted (regular language).

- \( \varphi_4 \): \((\psi_7, \psi_5^5, \psi_1^4, \psi_10)\) The system performs \( n \) 0-tests, similar for \((\psi_7, \psi_5^5, \psi_5^5, \psi_10)\), while for \((\psi_7, \psi_5^5, \psi_7, \psi_10)\) one 0-test suffices (A returns 1), \((\psi_7, \psi_5^5, \psi_10)\) or \((\psi_5^5, \psi_5^5, \psi_{10})\) are invalid systems (too many of \( Z_2 \)'s, \( A \) returns 0).

- \( \varphi_5 \): \((\psi_5^5, \psi_5^5, \psi_10)\) I. If each \( x \) occurring in \( w \) is produced in the same time by \( \psi_5^5 \) and \( \psi_5^5 \), then \( A \) performs \( n \) 0-tests. II. If each \( x \) occurring in \( w \) is produced either by \( \psi_5^5 \) or by \( \psi_5^5 \), then \( A \) checks (through universal branches) whether each \( x_i \), where \( i \) is odd, is produced by a rule \( \psi_5^5 \) and each \( x_i \), where \( i \) is even, is produced by rule \( \psi_5^5 \). III. If there are \( x \) occurring in \( w \) produced in the same time by rules of \( \psi_5^5 \) and \( \psi_5^5 \), and there are \( x \)'s occurring in \( w \) produced only by a rule of type \( \psi_5^5 \) or \( \psi_5^5 \), then \( A \) checks whether \( w \) contains at least one substring composed of consecutive symbols produced only by rules of type \( \psi_5^5 \) or of type \( \psi_5^5 \) (but not both). If it holds, \( A \) does not accept the input (it returns 0, and 1 otherwise).

- \( \varphi_6 \): \((\psi_7, \psi_5^5, \psi_9, \psi_{10})\) Due to rule \( \psi_9 \), \( A \) always accepts as long as each \( x_i \) in \( w \) is produced by a rule of type \( \psi_5^5 \), \( 1 \leq i \leq n \) (universal branches).

- \( \varphi_7 \): \((\psi_7, \psi_5^5, \psi_5^5, \psi_9, \psi_{10})\) each \( x \) occurring in \( w \) is produced in the same time by \( \psi_5^5 \) and \( \psi_5^5 \). \( A \) proceeds similar as in \( \varphi_3 \). It also checks if \( n \) is produced by \( \psi_5^5 \).

- \( \varphi_8 \): \((\psi_7, \psi_5^5, \psi_5^5, \psi_5^5, \psi_10)\) I. If \( \psi_5^5 \) and \( \psi_5^5 \) produce the same terminals, but distinct than the terminals produced by \( \psi_5^5 \) or if \( \psi_5^5 \), \( \psi_5^5 \), and \( \psi_5^5 \) produce the same terminals, then \( A \) proceeds similar as in \( \varphi_3 \), as the rules of type \( \psi_5^5 \) do not modify the number of \( Z_2 \)’s on the s.form of \( G_2 \) (there will always be a derivation of \( w \) in \( G \) by the nondeterminism and 0-tests). II. If each \( x \) in \( w \) is produced only by a rule in \( \{\psi_5^5, \psi_5^5, \psi_5^5\} \) then \( w \) is of the form \( w = \alpha_1\beta_1\alpha_2\beta_2...\alpha_k\beta_k\alpha_{k+1} \) in which each substring \( \alpha_i \) is composed of terminals produced by rules of type \( \psi_5^5 \) and additionally of type \( \psi_5^5 \) (however these rules may be missing), and each substring \( \beta_i \) is composed of terminals produced by rules of type \( \psi_5^5 \), and additionally, of type \( \psi_5^5 \). Denote by \( \eta_i^{(5)} \) the number of terminals in \( \alpha_i \) produced by rules of type \( \psi_5^5 \),

---

5For a system \((\psi_1^4, \psi_5^5, \psi_{10})\) the procedure works similar as described in \( \varphi_1 \) and \( \varphi_2 \).
and by \( \eta_i^{(3)} \) the number of terminals in \( \beta_i \) produced by rules of type \( \psi_i^x \). Note that \( \eta_i^{(5)} \) may be 0. If \( \eta_i^{(5)} > 0 \) then the \((RL_1; RB_k)\)-system should perform \( \eta_i^{(5)} \) 0-tests, by alternatively applying a rule \( \psi_i^x \), followed by \( \psi_7 \) (to recharge the counter). The same happens each time, at a certain snapshot (point of derivation), the number of terminals produced by rules of type \( \psi_i^x \) is smaller than the number of terminals produced by rules of type \( \psi_i^5 \) (the rule \( \psi_7 \) allows the counter to be recharged at any 0-test). Hence, for this kind of system, an input \( w \) can be decided when checking the \( \beta_k \) segment. Note that, if \( \eta_i^{(5)} = 0 \) then \( w \) cannot be generated by \( G \), as the counter cannot be emptied. Since a rule of type \( \psi_i^x \) does not modify the number of \( Z_2 \)'s on the s.form of \( G_2 \), \( A \) only checks whether terminals in \( w \) that are not produced by rules of type \( \psi_i^x \) and \( \psi_i^5 \), can be produced by rules of type \( \psi_i^\gamma \). Below we describe a sequential procedure that decides \( w \).

First a variable \( S \) is set to \( \eta_i^{(3)} \), \( \eta_{k+1}^{(3)} = 0 \), and \( i = 2 \), while \( i \leq k + 1 \) do (if \( S + 1 \leq \eta_i^{(5)} \) then \( S \) is set to \( \eta_i^{(3)} \) (because an \( S + 1 \) number of \( Z_2 \)'s are deleted by rules of type \( \psi_i^x \) and \( \eta_i^{(5)} \) - 1 - \( S \) 0-tests are performed, hence \( S \) is set to the next \( \eta_i^{(3)} \)) else \( S \) is set to \( S - \eta_i^{(5)} + \eta_i^{(3)} \) (the counter contains \( S - \eta_i^{(5)} + \eta_i^{(3)} + 1 \) of \( Z_2 \)'s), \( i \) is set to \( i + 1 \). The procedure returns \( S \).

The input \( w \) can be derived by \( G \) if \( S = 0 \), as \( S \) is set to \( \eta_k^{(3)} = 0 \), only if the previous value of \( S \) is smaller than \( \eta_{k+1}^{(5)} - 1 \). For \( A \) to decide the input only the part of the routine starting from the point when \( i \) is set to the maximum \( j \), \( j \leq k \), such that \( S \) is set to \( \eta_j^{(3)} \), after the last 0-test, is important (no matter how many times the counter is recharged). To do so, \( A \) spawns \( k \) existential processes \( \partial_j \), \( 1 \leq j \leq k - 1 \), each of which holds a suffix of \( w \) of the form \( w_{j,k+1} = \beta_j \alpha_j \beta_{j+1} \ldots \alpha_k \beta_k \alpha_{k+1} \). On each \( \partial_j \), \( A \) checks whether \( \sum_{i=j}^k \eta_i^{(3)} - \sum_{i=j+1}^k \eta_i^{(5)} + 1 \leq \eta_{k+1}^{(5)} \), i.e., whether the number of \( Z_2 \)'s gathered on the counter, over the \( \beta_l \) segments, \( j \leq l \leq k + 1 \), can be deleted by rules of type \( \psi_i^5 \), used over the \( \alpha \) segments, as long as no other 0-test is performed during the scanning of the suffix \( w_{j,k+1} \). That is, whether \( \sum_{i=j}^q \eta_i^{(3)} - \sum_{i=j+1}^q \eta_i^{(5)} + 1 > \eta_{q+1}^{(5)} \) for each \( j \leq q \leq k - 1 \). These inequalities should be universally checked on each existential branch, through \( O(k - j + 1) \) universal processes, each of which testing only one inequality. The fan-out of the computation tree of \( A \) on the \( k \) existential processes \( \partial_j \), or on the \( k - j + 1 \) universal branches \( 1 \leq j \leq k - 1 \), is \( O(n) \). It can be converted (using a divide and conquer procedure) into a computation tree of fan-out 2 and height \( O(\log n) \). The (iterative) summations and comparison operations in binary require \( O(\log n) \) time and space, as these operations\(^6\) are in \( NC^1 \). Hence, \( A \) decides \( w \) in \( O(\log n) \) time and space.

When the correction rule \( \psi_9 \) is added to \( \varphi_8 \) \( A \) accepts any combination of \( a_i \in tal(G_1) \) in \( w \) as the surplus of \( Z_2 \)'s on the counter can be deleted by \( \psi_9 \) at the end of computation. In this case \( A \) universally checks whether each \( x_i \) in \( w = x_1x_2...x_n \) is produced solely by a single rule in \( \varphi_8 \). Other cases, e.g., when \( \psi_7 \) is added to \( \varphi_8 \), and \( \psi_7^5 \) and \( \psi_7^5 \), or \( \psi_7^5 \) are dropped from \( \varphi_8 \), such that each \( x \) is produced only by a rule in \( \varphi_8 \), are treated in a similar manner.

- \( \varphi_9: (\psi_7, \psi_7^5, \psi_7^5, \psi_10) \) and each \( x \) in \( w \) is produced only by a rule in \{ \psi_7^5, \psi_7^5 \} then \( A \) proceeds similar as in \( \varphi_8 \). II.

\(^6\)Multiplication and addition of \( n \) \( n \)-bit numbers are in \( NC^1 \).
- $\varphi_{10}$: $(\psi_7, \psi_{\bar{7}}^x, \psi_{\bar{5}}^x, \psi_{10})$ and there exist $x$'s in $w$ produced only by a rule in $\{\psi_3^x, \psi_5^x\}$ and there exist $x$'s produced in the same time by $\psi_3^x$ and $\psi_5^x$.

Denote by $I = \{a_{i_1}, \ldots, a_{i_r}\}, H = \{a_{h_1}, \ldots, a_{h_p}\}$ and $L = \{a_{l_1}, \ldots, a_{l_q}\}$ the sets of terminals in $\text{tal}(G_1)$ such that each $a_{i_j}, 1 \leq j \leq r$, is produced only by the rule $\psi_{a_{i_j}}^o$, each $a_{h_j}, 1 \leq j \leq p$, is produced only by $\psi_{a_{h_j}}^o$, and each $a_{l_j}, 1 \leq j \leq q$, is produced in the same time by $\psi_{a_{l_j}}^3$ and $\psi_{a_{l_j}}^5$. Obviously, $H \cup I \cup L$ is a partition of $\text{tal}(G_1)$. In this case an input $w$ may be of the form $w = L_0 H_0 L_1 \alpha_1 \ldots \alpha_j \ldots \alpha_k \bar{L}_0 \bar{H}_0 \bar{L}_1$, where $\alpha_j = I_j L_{j_1} H_{j_1} L_{j_2} H_{j_2} \ldots L_{j_4} H_{j_4}$, $1 \leq j \leq k$, each $H_-$ is composed of terminals in $H$, each $I_-$ is composed of terminals in $I$, and each $L_-$ is composed of terminals in $L$. Substrings of type $H_-$ may be resolved by 0-tests and rule $\psi_7$, substrings of type $L_-$ may be resolved as in $\psi_3$. The only substrings that may block the computation are of type $I_-$.

To check whether the $Z_2$'s symbols added on the counter by segments of type $I_-$ can be “deleted in due time”, as in the case of systems of type $\varphi_8$, $A$ spawns $k$ existential processes $\delta_j$, $1 \leq j \leq k$, each of which holds a suffix of $w$ of the form $w_{j,k} = \alpha_j \ldots \alpha_k \bar{L}_0 \bar{H}_0 \bar{L}_1$, $1 \leq j \leq k - 1$. $A$ checks for $w_{j,k}$ similar conditions as in $\varphi_8$, where the number of terminals produced by rules of type $\psi_3^-$ are counted, then when it is necessary to delete the exceeding $Z_2$'s produced by rules of type $\psi_3^-$ used in $I_-$, also over the segments of type $L_-$. If the number $\eta_1^{(3)}$ of $Z_2$'s produced by rules of type $\psi_3^-$ to generate a certain $I_t$ segment is smaller than the number $\eta_2^{(5)}$ of $Z_2$'s deleted by rules of type $\psi_5^-$ (used in $H_{l_1}, \ldots, H_{l_q}$ and $L_-$ segments), then denote by $S_t^- = \eta_2^{(5)} - \eta_1^{(3)}$ (computed in binary). $A$ performs $S_t^-$ 0-tests for some of the $H_{l_1}, \ldots, H_{l_q}$ segments and applies the procedure described in $\varphi_3$ for the $L_-$ segments, or it may use some of the $\psi_5^-$ rules to delete the surplus of $Z_2$'s produced in segments of type $\alpha_5$ occurring in $w_{j,k}$ before $\alpha_1$ (if this surplus exists).

If the number $\eta_1^{(3)}$ of $Z_2$'s produced by rules of type $\psi_3^-$ to generate $I_t$ is greater than the number $\eta_2^{(5)}$ of $Z_2$'s deleted by rules of type $\psi_5^-$ used to generate the $H_{l_1}, \ldots, H_{l_q}$ and $L_-$ segments, then denote by $S_t^+ = \eta_1^{(3)} - \eta_2^{(5)}$ the surplus of $Z_2$'s (computed in binary). $A$ checks further for the next segments of type $\alpha_5$ occurring in $w_{j,k}$, after $\alpha_1$, such that $\eta_1^{(3)} < \eta_2^{(5)}$. Denote by $S_t^+ = \eta_t^{(3)} - \eta_t^{(5)}$. $A$ computes $S_t^+ - S_t^-$, if $S_t^+ > S_t^-$, $l < l'$, and $S_t^+$ is set to $S_t^+ - S_t^-$. Otherwise $S_t^+$ is set to 0 and $A$ performs $S_t^+ - S_t^-$ 0-tests on the $\alpha_5$ segment.

- $\varphi_{11}$: $(\psi_2^x, \psi_{\bar{2}}^x, \psi_{\bar{7}}^x, \psi_{10})$ There exist $x$’s in $w$ produced only by a rule in $\{\psi_3^x, \psi_5^x\}$ and there exist $x$’s produced in the same time by $\psi_3^x$ and $\psi_5^x$.

$\varphi_{11}$ I. Any $x$ produced by a rule of type $\psi_2^x$ can be also produced by a rule of type $\psi_3^x$ or $\psi_5^x$, or both. Denote by $I = \{a_{i_1}, \ldots, a_{i_r}\}$, $H = \{a_{h_1}, \ldots, a_{h_p}\}$, and $L = \{a_{l_1}, \ldots, a_{l_q}\}$ the sets of terminals in $\text{tal}(G_1)$ such that each $a_{i_j}, 1 \leq j \leq r$, is produced only by the rule $\psi_{a_{i_j}}^o$, each $a_{h_j}, 1 \leq j \leq p$, is produced only by $\psi_{a_{h_j}}^o$, and each $a_{l_j}, 1 \leq j \leq q$, is produced in the same time by $\psi_{a_{l_j}}^3$ and $\psi_{a_{l_j}}^5$. For the sake of simplicity, we refer to a terminal $x \in X, X \in \{I, H, L\}$, that occurs in $w$, as $x \in X \cap w$. $A$ spawns $n$ universal processes $\delta_j, 1 \leq j \leq n$.

- On $\partial_1$, $A$ returns 1 if there exists a rule $\psi_{x_1}^o$ producing $x_1$ and 0 otherwise.

---

7Some of the substrings of type $H_-, I_-$, or $L_-$ may be empty strings.

8The beginning of each $L_-$ segment may be considered as a marker in $w$. 

13
\[ \text{On any } \partial_j, 2 \leq j \leq n - 1, \text{ such that } x_j \in I \cup L, A \text{ returns } 1 \text{ by default.} \]

\[ \text{On any } \partial_j, 2 \leq j \leq n - 1, \text{ such that } x_j \in H, A \text{ counts the number of terminals } \]

\[ x \in H \cup I \text{ occurring in } w_j = x_1 x_2 \ldots x_{j-1}. \text{ Suppose there exist } \eta_j \text{ terminals in } I \cap w_j \text{ and } \eta_j^{(5)} \text{ terminals in } H \cap w_j. A \text{ checks whether } \eta_j^{(3)} - \eta_j^{(5)} \geq 0, \text{ i.e., at the application of rule } \]

\[ \psi_j (\text{producing } x_j) \text{ the s.form of } G_2 \text{ contains at least one } Z_2. \text{ If it holds } \partial_j \text{ returns } 1. \text{ If } \]

\[ \eta_j^{(5)} - \eta_j^{(3)} > 1, A \text{ must use a rule of type } \psi_j^- \text{ to produce a terminal in } L \cap w_j. \text{ Furthermore, } \]

\[ \hat{A} \text{ makes such an election for each process } \partial_m, m < j, \text{ handling a terminal } x_m \in H \cap w_j, \]

\[ \text{with the property that } \eta_m^{(5)} - \eta_m^{(3)} > 1. \text{ Hence, at process } \partial_j, \text{ when choosing an } y \in L \cap w_j \]

\[ \text{to be rewritten by } \psi_3^y, A \text{ must assure that the number of terminals in } L \cap w_j \text{ is greater than the number of terminals handled by processes of type } \partial_m, \text{ with } \eta_m^{(5)} - \eta_m^{(3)} > 1. \text{ To do so, } A \text{ counts the number of terminals } x \in H \cap w_j, \]

\[ \text{with the property that for each fixed position } x_i \text{ of } x \text{ in } w \text{ (i.e., } x_i = x \text{) } \eta_i^{(5)} - \eta_i^{(3)} > 1 \text{ holds. Suppose there exist } \eta_j^{(c,j)} \text{ terminals } x \text{ with this property (including } x_j), \text{ i.e., up to the } j^{th} \text{ symbol in } w, A \text{ must choose } \eta_j^{(c,j)} - 1 \text{ terminals in } L \text{ to be produced by rules of type } \psi_j^- . \]

\[ A \text{ checks whether } \eta_j^{(c,j)} \leq \eta_j^{(c,j)}, \text{ where } \eta_j^{(c,j)} \text{ is the number of terminals in } L \cap w_j. \text{ On the other hand, each time } \]

\[ \eta_j^{(5)} - \eta_j^{(3)} - 1 = 0, \text{ a } 0\text{-test is performed, hence } A \text{ must check whether } x_j \text{ can be produced by a rule of type } \psi_j^x \]

\[ (\text{no matter whether } x_j \text{ can be also produced by a rule of type } \psi_j^- \text{ or } \psi_j^-, \text{ or both). Process } \]

\[ \partial_j \text{ returns } 1 \text{ if these conditions hold.} \]

\[ \circ \text{ On } \partial_n \text{ if } x_n \in I, \text{ then } \partial_n \text{ returns } 0. \]

\[ \circ \text{ Case } \partial_n^1 \text{ if } x_n \in H \text{ and } \eta_n^{(5)} - \eta_n^{(3)} = 1, \text{ then after applying } \psi_n^x \text{ there will be no } Z_2 \text{’s on the s.form of } G_2, \text{ and } G \text{ may apply rule } \psi_{10} \text{ to empty the stack. However, this is conditioned by the acceptance of the string composed of all terminals in } w \cap L \text{ that have not been chosen to be produced by rules of type } \psi_3. \]

\[ \text{Denote by } w_L \text{ the string composed of these terminals (no matter which the order is). The acceptance of } w_L \text{ is decided as in } \varphi_3. \]

\[ \text{We have } |w_L| = \eta_L^{(n)} - \eta_L^{(c,n)}, \text{ where } \eta_L^{(n)} \text{ is the number of terminals in } L \cap w, \text{ and } \eta_L^{(c,n)} \text{ is the number of terminals in } L \cap w \text{ chosen to be produced by rules of type } \psi_3^- . \text{ Note that, } \]

\[ \eta_L^{(c,n)} - \eta_L^{(c,k)} = \eta_L^{(c,k)}, \text{ where } k \text{ is the rightmost terminal in } H \cap w \text{ with the property } \eta_k^{(5)} - \eta_k^{(3)} > 1. \]

\[ \circ \text{ Case } \partial_n^2 \text{ if } x_n \in H \text{ and } \eta_n^{(5)} - \eta_n^{(3)} > 1, A \text{ proceeds as in process } \partial_j (x_j \in H, \eta_j^{(5)} - \eta_j^{(3)} > 1) \]

\[ \text{to check whether there exist enough terminals in } L \cap w \text{ to be produced by rules } \psi_3^-- \text{, and then as in } \partial_n^1 \text{ to check whether } w_L \text{ is accepted by } A. \]

\[ \circ \text{ Case } \partial_n^3 \text{ if } x_n \in H \text{ and } \eta_n^{(3)} - \eta_n^{(5)} \geq 0, \text{ then the exceeding } Z_2 \text{’s existing in the s.form of } G_2, \text{ after “spelling” the string composed of terminals in } (H \cup I) \cap w \text{ (preserving their order in } w) \text{ must be deleted by using rules of type } \psi_5^- \text{ that produce terminals in } L \cap w. \]

\[ \text{To do so, } A \text{ checks whether } \eta_L^{(n)} - \eta_L^{(c,n)} - 1 \geq \eta_L^{(3)} - \eta_L^{(5)} \text{ and the string } w_L, \text{ of length } |w_L| = \eta_L^{(n)} - \eta_L^{(c,n)} - 1 - (\eta_L^{(3)} - \eta_L^{(5)}) \text{ is valid as in process } \varphi_3, \text{ where } \eta_L^{(n)} \text{ is the number of terminals in } L \cap w, \text{ and } \eta_L^{(c,n)} \text{ is the number of terminals in } L \cap w \text{ chosen to be produced by rules of type } \psi_3^- .
\]

\[ ^9] \text{That is there are enough terminals in } L \cap w_j \text{ that can be produced by rules of type } \psi_j^- \text{ in order to cover the lack of } Z_2 \text{’s that must be deleted by rules of type } \psi_5^- \text{ that produce only terminals in } H, \text{ up to the } x_j \text{ symbol.} \]
Case $\partial_n^1$ If $x_n \in L$, then denote by $\eta_n^{(3)}$ the number of terminals in $I \cap w$, $\eta_n^{(5)}$ the number of terminals in $H \cap w$. $A$ computes $|w_L| = \eta_n^{(n)} - \eta_n^{(c,n)} - 1 - (\eta_n^{(3)} - \eta_n^{(5)})$ and it checks whether $w_L$ is accepted by $A$ (as in $\varphi_3$). Recall that $G$ must apply rule $\psi_5^{x_n}$ to produce $x_n$. It ends the derivation with rule $\psi_10$.

Note that, the reasoning for which a process $\partial_j$, that handles a terminal $x_j \in I \cup L$ always returns 1, is the following. If $x_j$ is not preceded in $w$ by any $y \in H \cap w$, then applying $\psi_3^{x_j}$ to produce $x_j$ arises no problem ($\partial_j$ may return 1). If $x_j$ is preceded in $w$ by some $y \in H \cap w$, then the application of $\psi_3^{x_j}$ may be disabled by the lack of $Z_2$’s in the s.form of $G_2$. However, this is resolved when checking if $\eta_l^{(3)} - \eta_l^{(5)} \geq 0$, where $y = x_l$ is the very last terminal that precedes $x_j$, occurring also in $H$. If $\eta_l^{(3)} - \eta_l^{(5)} < 0$, then at process $\partial_l$, $A$ chooses a terminal in $L \cap w$ to be produced by a rule of type $\psi_3^-$, allowing the application of rule $\psi_3^{x_j}$ (and again $\partial_j$ may return 1). If $\partial_l$ cannot find enough terminals in $L \cap w$ to be produced by a rule of type $\psi_3^-$, then $\partial_l$ returns 0, which rectifies the possible error obtained when $\partial_j$ is set to 1 ($\partial_j$ are universal processes).

$\varphi_{11}$ II. If $x$ is produced only by a rule of type $\psi_5^-$, then $x$ can be used as a 0-test marker in $w$ and $A$ checks whether the strings between these markers can be generated by a system composed only of rules of types $\psi_5^-$ and $\psi_5^+$.

Other cases, e.g., when $\psi_6^+$ or $\psi_9$ are added to $\varphi_8$, $\varphi_9$, $\varphi_{10}$ or $\varphi_{11}$ and $\psi_{10}$ is dropped, are treated in a similar manner. If in all the above cases $\psi_1^+$ is added, and $x$ is produced only by $\psi_3^+$, then again $x$ may be used as a marker in $w$ and $A$ checks whether the substrings of $w$ placed between these markers can be generated by an $(RL_1; RB_c)$-system discussed above. If $x$ is produced in the same time by $\psi_1^+$, $\psi_3^+$ or $\psi_5^+$ etc., then $\psi_1^+$ may be left to be used only for those $x_i$ in $w$ when 0-tests may be produced by reading $x_i$, $x_{i+1} = x$ and then when using $\psi_1^+$ prevents too many of $Z_2$’s to be added on the counter, if instead the rule $\psi_3^+$ would be used, or in order to prevent less $Z_2$’s to be deleted, if instead the rule $\psi_5^+$ would be used. Otherwise, if $\psi_1^+$ has no “corrections” to perform on the number of $Z_2$’s in the s.form of $G_2$, then rule $\psi_1^+$ is simply ignored, and the system is judged as in the cases already discussed.

The computation tree of $A$ has a constant number of (existential or universal) levels (related to the length of the input) of fan-out $O(n)$. Each level, in which $A$ spawns $O(n)$ existential or universal branches, can be unfolded (through a divide and conquer procedure) into a binary tree of height $O(\log n)$. Auxiliary operations such as counting, addition, multiplication of binary numbers, require $O(\log n)$ time and space (as these are in $\mathcal{NC}^1$). Hence, the computation tree of $A$ has depth $O(\log n)$ and bounded fan-out.

$O(\log n)$ space restriction on $B_2$ forces $\psi_7$, $\psi_8$ or $\psi_9$ to be applied of at most polynomial number of times (related to the length of the input) without altering the language. If $\psi_7$ or $\psi_8$ are applied in $G$ of an exponential number of times, then if the surplus of $Z_2$’s is deleted only by $\psi_9$ through at least one loop that does not contain any non-chain rule, $A$ cuts off the (sequential) running time (i.e. space for the ATM) of these (unproductive) loops. Otherwise, the number of $Z_2$’s on the s.form of $G_2$ is linearly related to the length of the input, hence $n_z = \mathcal{O}(n)$, and $O(\log n)$ space suffices to record in binary the number

---

10 The correction is done when running, through another universal branch, the process $\partial_i$, case when $x_i \notin H$ (already described).
of Z₂’s on the s.form of G₂ at any snapshot.

**Corollary 4.5.** \( L(OC_1) = L(RL_1; RB_c) \subseteq NC^1 \subseteq SPACE(\log n) \).

Denote by \((RL_1^0; RB_c)\) the \((RL_0; RB_c)\)-counter nets systems for which \(ntal(G_1)\) is restricted to only one nonterminal, i.e., \((RL_1; RB_c)\)-counter systems that are not allowed to perform 0-tests, and by OCN₁ one counter nets automata with only one control state. \((RL_1^0; RB_c)\)-systems may be composed only of rules of the forms \(\varphi_{x_3}, \varphi_{x_4}, \varphi_{x_5}, \varphi_8, \varphi_9\). Hence we have

**Corollary 4.6.** \( L(OCN_1) = L(RL_1^0; RB_c) \subseteq NC^1 \).

## 5 Useful Examples

**Example 5.1.** The \((RL_0; RB_c)\)-counter nets system for which \(tal(G_1) = \{a, b\}, ntal(G_1) = \{S_1, X\}, al(G_2) = \{Z_2\}, ax(G_1) = S_1, ax(G_2) = Z_2\), and \(R = \{(S_1 \rightarrow aS_1, Z_2 \rightarrow Z_2Z_2), (S_1 \rightarrow X, Z_2 \rightarrow \lambda), (X \rightarrow bX, Z_2 \rightarrow \lambda), (X \rightarrow b, Z_2 \rightarrow \lambda)\}\) generates the OCN language \(L = \{a^n b^n \mid n \geq 1\}, \ L \notin L(OCN_1)\).

**Example 5.2.** The \((RL; RB_c)\) system with \(al(G_1) = \{a, b\}, ntal(G_1) = \{S_1, X\}, al(G_2) = \{S_2, Z_2\}, ax(G_1) = S_1, ax(G_2) = S_2\) and \(R = \{(S_1 \rightarrow S_1, S_2 \rightarrow S_2Z_2), (S_1 \rightarrow aS_1, Z_2 \rightarrow Z_2Z_2), (S_1 \rightarrow X, Z_2 \rightarrow \lambda), (X \rightarrow bX, Z_2 \rightarrow \lambda), (X \rightarrow \lambda, S_2 \rightarrow \lambda)\}\) generates the OCA language \(L = \{a^n b^n \mid n \in \mathbb{N}\}, \ L \notin L(OC_1)\).

**Example 5.3.** The \((RL_1^0; RB_c)\) system for which \(tal(G_1) = \{a, b\}, ntal(G_1) = \{S_1\}, al(G_2) = \{Z_2\}\), and \(R = \{(S_1 \rightarrow aS_1, Z_2 \rightarrow Z_2Z_2), (S_1 \rightarrow bS_1, Z_2 \rightarrow \lambda), (S_1 \rightarrow \lambda, Z_2 \rightarrow \lambda)\}\) generates the OCN₁ language \(L = \{a^n b^m a^n b^{m_2} \cdots a^{n_{k-1}} b^{m_{k-1}} a^{n_k} b^{m_k} \mid \sum_{i=1}^{j} n_i \geq \sum_{i=1}^{j} m_i, 1 \leq j \leq k-1, \sum_{i=1}^{k} n_i = \sum_{i=1}^{k} m_i, k \geq 1, n_i, m_i \in \mathbb{N}, n_i, m_i \geq 1, 1 \leq i \leq k\}\).

Note that, if rules \((S_1 \rightarrow aS_1, S_2 \rightarrow S_2Z_2)\) and \((S_1 \rightarrow \lambda, S_2 \rightarrow \lambda)\) are added to \(R\), and rule \((S_1 \rightarrow \lambda, Z_2 \rightarrow \lambda)\) is removed from \(R\), then the resulted system is an \((RL_1; RB_c)\)-counter system generating the same language.

## 6 Conclusions

Cts-systems were introduced in [17] as a unifying framework for several rewriting systems and automata. The main advantage of a cts-system is that it splits the simulated device into several subcomponents, each of which having to accomplish a precise task, such that all tasks together carry out the system’s work. This modularization facilitates deeper investigations into the nature of the simulated device, from a generative power and computational complexity perspective. In this paper, we have materialized this idea for Petri nets and counter machines.

We proved, via cts-systems, that the class of languages generated by \(\lambda\)-free labeled marked Petri nets, known as \(L\)-languages [3], is included in \(NSPACE(\log n)\), while the class of languages generated by one-counter machines with only one control state (with or without 0-tests) is included in \(U_{E^*}\)-uniform \(NC^1\). All upper bounds are tight and they
bring new insights into the computational complexity of several subclasses of one counter languages, languages generated by VASS in fix dimension, and Petri nets languages.

One of our future aims is to extend this line of research upon the complexity of one counter (nets) languages, in order to tighten the known upper bound $\text{NSPACE}(\log n)$ to $\mathcal{N}^1$. Even if sophisticated, we hope to combine methods in Theorem 4.4 with the state diagram used in Theorem 4.3 to reach a $\text{SPACE}(\log n)$ upper bound for $L(OC)$ or $L(OCN)$. The method described in Theorem 3.2 combined with a proper state diagram defined for the general case of Petri nets similar to that provided by Definition 4.2, may lead to some improvements on the non-elementary upper bound of the $L^\lambda_0$ class, as $L^\lambda_0 = L(RL; 0S)$ [1].

References

[1] Aalbersberg, I.J.J., Rozenberg, G.: CTS Systems and Petri Nets, Theoretical Computer Science 40, 149–162 (1985).
[2] Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity, Springer-Verlag (1990).
[3] Czerwiński, W., Lasota, S., Lazić, R., Leroux, J., Mazowiecki, F.: The Reachability Problem for Petri Nets is Not Elementary, STOC 2019, 24–33 (2019).
[4] Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G.: Computations in Coordinated Pair Systems, Technical Report CU-CS-260-83, Univ. of Colorado at Boulder (1984).
[5] Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G.: On Coordinated Rewriting, Fundamentals of Computation Theory (1985).
[6] Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G.: On the Active and Full Use of Memory in Right-Boundary Grammars and Push-Down Automata, Theoretical Computer Science 48(3): 201–228 (1986).
[7] Englert, M., Lazić, R., Totzke, P.: Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-complete. LICS 2016, 477–484. ACM (2016).
[8] Hack, M.: Petri Net Languages, Technical Report 159, Lab. for Computation Science, MIT Cambridge, MA (1976).
[9] Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, (1978).
[10] Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in Succinct and Parametric One-Counter Automata. CONCUR 2009, Concurrency Theory, LNCS 5710, 369–383, Springer (2009).
[11] Hopcroft, J.E., Pansiot, J.J.: On the reachability problem for 5-dimensional vector addition systems. Theoretical Computer Science 8, 135–159 (1979).
[12] Jantzen, M.: On the Hierarchy of Petri Net Languages. RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Application 13.1: 19–30 (1979).
[13] Kleijn, H.C.M., Rozenberg, G.: On the Role of Selectors in Selective Substitution Grammars, *Fundamentals of Computation Theory*, 190–198 (1981).

[14] Leroux, J., Schmitz, S.: Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. *LICS 2019*, 1–13. ACM (2019).

[15] Lipton, R.: The Reachability Problem Requires Exponential Space, *Technical Report 62*, Yale University, Dept. of CS. (1976).

[16] Petri, C.A.: Kommunikation mit Automaten. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM nr. 2 (1962).

[17] Rozenberg, G.: Selective Substitution Grammars (Towards a Framework for Rewriting Systems), Part I: Definitions and Examples, *Elektron. Informationsverarbeit Kybernetik* 13, 455–463 (1977).

[18] Rozenberg, G.: On Coordinated Selective Substitutions: Towards a Unified Theory of Grammars and Machines, *Theoretical Computer Science* 37, 31–50 (1985).

[19] Ruzzo, W.: On Uniform Circuit Complexity, *Journal of Computer and System Sciences*, 22(3), 365–383 (1981).

[20] Salomaa, A.: *Formal Languages*. Academic Press, London-New York (1973).

[21] Schmitz, S.: Complexity Hierarchies Beyond ELEMENTARY. *ACM Transactions and Computation Theory* 8(1), 3:1–3:36 (2016).

[22] Sudborough, I.H.: On Tape-Bounded Complexity Classes and Multihead Finite Automata, *Journal of Computer and System Sciences*, 10(1): 62–76 (1975).