Liliana Cojocaru

Linear Context-Free Languages Are in NC^1

DEPARTMENT OF COMPUTER SCIENCES
UNIVERSITY OF TAMPERE

A-2009-2

TAMPERE 2009

ISBN 978-951-44-7687-7
ISSN 1459-6903

Linear Context-Free Languages Are in NC^1

Liliana Cojocaru

Rovira I Virgili University of Tarragona
Research Group on Mathematical Linguistics
Pl. Imperial Tarraco 1, 43005, Spain
liliana.cojocaru@estudiants.urv.cat

Abstract. This paper concerns *linear context-free languages* (LIN). We prove that LIN $\subseteq$ NC^1 (under $U_{E^*}-$*uniformity reduction*). We introduce a new normal form for context-free grammars, called *Dyck normal form*. Using this new normal form we prove that for each context-free language L there exist an integer K and a homomorphism φ such that $L = \varphi(D'_K)$, where $D'_K \subseteq D_K$, and D_K is the one-sided Dyck language over K letters. Based on these results we prove that each linear context-free language can be recognized in $\mathcal{O}(\log n)$ time and space by an *indexing alternating Turing machine* (ATM). Since the class of languages recognizable by an indexing ATM in logarithmic time equals the U_{E^*}-uniform NC^1 class, result proved in [15], we obtain that LIN $\subseteq$ NC^1, and consequently, LIN $\subseteq$ L (where L is the class of languages recognizable by a deterministic Turing machine in logarithmic space). On the other hand, according to [17], each language in LIN belongs to L if and only if L = NL (where NL is the class of languages recognizable by a nondeterministic Turing machine in logarithmic space). Hence, besides the inclusion of linear context-free languages in NC^1, problem left open in [12], we also resolve the long-standing open question L $\stackrel{?}{=}$ NL in the favor of L = NL.

1 Introduction

Due to their ability in covering a large variety of computational phenomena occurring in parallel computing, *Boolean circuits* have been proved to be one of the most suitable models of parallel computation. This paper does not directly deal with circuits but indirectly through the notions of the NC^1 class and alternating Turing machine (ATM). Therefore, in this section we provide only a brief description of circuits and uniformity restrictions imposed on them. For the formal definition of Boolean circuits, other notions and results concerning circuit complexity and Boolean functions, the reader is referred to [1], [15], [18], and [19].

Informally, a Boolean circuit is an assembly composed of a fixed number n of Boolean input variables x_1, ..., x_n, a finite number of gates over a basis set of Boolean functions $\mathcal{B}$, and a fixed number m of outputs y_1, ..., y_m. The graphical representation of a Boolean circuit is viewed as a finite directed acyclic graph

(V, E) in which nodes are labeled by input variables, gates, and outputs [18]. Edges in this graph are rigorously partitioned and ordered according to the type of Boolean functions belonging to the basis $\mathcal{B}$, such that edges belonging to the same partition connect nodes corresponding to arguments of the same Boolean function. The in-degree (out-degree) called fan-in (fan-out) of each node depends on the type of the variable that labels that node and on the partition defined on the set E. Each output element y_i, $1 \leq i \leq m$, not necessarily labeling a node with fan-out 0, gives the i^{th} bit of the output $y = (y_1, ..., y_m)$.

Since the input of a Boolean circuit has a fixed length, this model builds for each input length a circuit of its own. Therefore, for arbitrarily large number of inputs, whose lengths cover an infinite set of instances of a problem, there would be an infinite number of circuits solving the problem. Due to this characteristic Boolean circuits are *non-uniform* models of computation, in opposite with finite state automata or Turing machines, for which one device is sufficient to deal with all inputs of arbitrary length. This is the reason why a uniformity restriction should be imposed on Boolean circuits in order to rank them among the realistic models of computations, i.e., to make them algorithmically describable. This can be done by restructuring an infinite family of circuits according to several rules that make them "regular" and "uniformly constructible". Under uniformity conditions Boolean circuits are called *uniform Boolean circuits*. This makes it possible to compare Boolean circuits with other parallel or sequential models of computations such as alternating or sequential Turing machines.

The main complexity measures concerning Boolean circuits are the *size* of a circuit, i.e., the number of gates (that corresponds to the time complexity of a Turing machine, and to the space complexity of an ATM), and the *depth* of the circuit (that corresponds to the space of a Turing machine, and to the time of an ATM). Connections between the complexity measures of Boolean circuits and Turing machines or ATMs can be found in [2] and [15], respectively.

The NC class 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 $\log n$. If for each integer i, we denote by NC^i the class of functions computable by polynomial size Boolean circuits, with depth $\mathcal{O}(\log^i n)$ and fan-in two, then we have $\mathrm{NC}^i \subseteq \mathrm{NC}^{i+1}$, $i \geq 1$, and $\mathrm{NC} = \bigcup_{i \geq 1} \mathrm{NC}^i$.

Depending on the type of the uniformity restriction, i.e., speedups on time and space imposed on (alternating) Turing machines simulating a family of circuits, we obtain several NC "uniform" classes. The uniformity condition on which we are interested in this paper is the U_{E^*}-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 $t(n)$ (see [15]). This gives birth to the so-called U_{E^*}-uniform NC^1 class. Depending on the type of the uniformity several characterizations of the NC class in terms of ATMs are presented in [15]. It is proved that for $i \geq 2$, all NC^i classes behaves the same no matter which uniformity restriction is imposed on circuits, i.e., $\mathrm{NC}^i = \mathrm{ASPACE}, \mathrm{TIME}(\log n, \log^i n)$, for all $i \geq 2$, where $\mathrm{ASPACE}, \mathrm{TIME}(s(n), t(n))$ denotes the class of languages acceptable by an ATM in simultaneous space $s(n)$ and time $t(n)$. For $i = 1$, this

equality holds only for the U_{E^*}-uniformity, more precisely, U_{E^*}-uniform $\mathrm{NC}^1 =$ ASPACE, TIME(log n, log n)= ALOGTIME.

The following relationships hold between NC^1, NC^2, and (nondeterministic) Turing time and space complexity classes: $\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{NC}^2 \subseteq \mathrm{NC} \subseteq \mathrm{P}$. The inclusion $\mathrm{NC}^1 \subseteq \mathrm{L}$ comes from [2] where it is proved that DEPTH(log n) $\subseteq$ DSPACE(log n) = L, where DEPTH(log n) is a log-time uniformity condition imposed on the depth of Boolean circuits. Later on it has been proved that ALOGTIME = DEPTH(log n), which traces the above inclusion for the U_{E^*} uniformity condition. However, NC^1 is contained in L no matter which uniformity restriction is imposed to circuits. Hence, $\mathrm{NC}^1 \subseteq \mathrm{L}$ "universally" holds.

Concerning the Chomsky hierarchy, it is well known that REG $\subset \mathrm{NC}^1$, and CFL $\subseteq \mathrm{NC}^2$ [15]. Up to now this is the best upper bound for the whole class of CFLs. In [12] it is proved that several CFLs, such as D_k languages, i.e., one-sided Dyck language over K letters, $k \geq 1$, structural, bracketed and deterministic linear CFLs are contained in NC^1 (under U_{E^*}-uniformity). It is a long-standing open problem whether the entire class of CFLs, or at least the LIN class, is included in the U_{E^*}-uniform NC^1. Proving that CFL, or at least LIN, is contained in NC^1 would imply that L equals NL. This holds due to the fact that for both classes LIN and CFL there exists a "hardest" language which is NL-complete.

Let A and B be two languages over a finite alphabet Σ. A is $\log(n)$-*space reducible* to B, denoted as $A \leq_{log} B$, if there exists a $\log(n)$-space computable function f, such that for each $x \in \Sigma^*$, it holds that $x \in A$ if and only if $f(x) \in B$. A language $L \subseteq \Sigma^*$ is $\log(n)$-*complete* if $L \in$ NL and, for each $L' \in$ NL, $L' \leq_{log} L$, i.e., the language L is *universal* in the class NL (each other language from NL is reducible to it).

In [17] it is proved that there exists an NL-complete linear language (the *hardest linear language* in the sense of completeness), while in [16] and [13] it is proved that there exists an NL-complete context-free language (the *hardest context-free language*). Proving that LIN is contained in NC^1 implies LIN $\subseteq$ L, i.e., L contains also the hardest linear language. Hence, L contains all languages from NL reducible to the hardest language (see [16]), i.e., NL $\subseteq$ L. The same reasoning holds for CFLs. In this paper we focus on linear context-free languages and we prove that LIN $\subseteq \mathrm{NC}^1$. Therefore, according to [17], we have settled the *log-space* problem in the favor of L = NL.

In order to prove the inclusion LIN $\subseteq \mathrm{NC}^1$ we first introduce in Section 2 a new normal form for CFLs, called *Dyck normal form*. We have been inspired to develop this new normal form by the general theory of Dyck words and Dyck languages, that turned out to play a crucial role in the description and characterization of CFLs (see [7], [8], and [12]).

The Dyck normal form is a syntactical restriction of the Chomsky normal form, in which the two nonterminals occurring on the right-hand side of a rule are paired nonterminals, in the sense that each left (right) nonterminal of a pair has a unique right (left) pairwise. This pairwise property imposed on the structure of the right-hand side of each rule induces a nested structure on the derivation tree of each word generated by a grammar in Dyck normal form. More precisely,

each derivation tree of a word generated by a grammar in Dyck normal form, read in the depth-first order is a Dyck word, hence the name of the normal form. This property, along with several other terminal rewriting conditions imposed to a grammar in Dyck normal form, stands at the basis of our intuition that at least for linear languages, the derivation tree of each word might be "rebuilt" by an ATM in logarithmic time and space.

Using the Dyck normal form we give a new characterization of CFLs in terms of Dyck languages. We prove, in Section 3, that for each language $L \in \text{CFL}$, there exist an integer K and a homomorphism φ such that $L = \varphi(D'_K)$, where D'_K is a subset of the one-sided Dyck language over K letters. We apply the Dyck normal form for linear languages and using the homomorphism φ, we prove in Section 4 the main result of this paper, i.e., each language in LIN can be recognized in $\mathcal{O}(\log n)$ time and space by an indexing ATM. With respect to [15], we have LIN $\subseteq$ NC^1. As a consequence, NL becomes "reducible" to L. On the other hand, it is well known from [5] that D0L $\subset$ AC^0, which together with LIN $\subseteq$ NC^1, provides the strict containment of linear context-free languages in NC^1.

2 Dyck normal form

A normal form for context-free grammars (CFGs) consists of several restrictions imposed to the structure of context-free productions, especially on the number of terminals and nonterminals allowed on the right-hand side of a context-free rule. In order to be correct a normal form for CFGs should generate the whole class of CFLs. For formal definitions, results, and surveys on normal forms, the reader is referred to [10].

Definition 1. *A context-free grammar $G = (N, T, P, S)$ is said to be in Dyck normal form if it satisfies the following conditions:*

1. *G is in Chomsky normal form,*
2. *if $A \to a \in P$, $A \in N$, $A \neq S$, $a \in T$, then no other rule in P rewrites A,*
3. *for each $A \in N$ such that $X \to AB \in P$ ($X \to BA \in P$) there is no other rule in P of the form $X' \to B'A$ ($X' \to AB'$),*
4. *for each rules $X \to AB$, $X' \to A'B$ ($X \to AB$, $X' \to AB'$), we have $A = A'$ ($B = B'$).*

Note that the Dyck normal form is a Chomsky normal form on which several restrictions on the positions of nonterminals occurring on the right hand-side of each context-free production are imposed. The reasons for which we introduce these restrictions (items 2, 3, and 4 in Definition 1) are the following. The second condition in Definition 1 allows to make a partition between those nonterminals rewritten by nonterminals, and those nonterminals rewritten by terminals. This enables us, in Section 3, to define a homomorphism from Dyck words to words generated by a grammar in Dyck normal form. The third and fourth conditions allow us to split the set of nonterminals into pairwise nonterminals, and thus to introduce bracketed pairs. This restriction (or pairwise property of nonterminals

occurring on the right-hand side of a context-free rule) is intensively used in Section 4 during several guessing procedures that estimate, by using only logarithmic time and space, the structure of the derivation tree associated with a word belonging to a linear language. The next theorem proves that the Dyck normal form is correct.

Theorem 1. *For each CFG $G = (N, T, P, S)$ there exists a grammar $G' = (N', T, P', S)$ such that $L(G) = L(G')$ and G' is in Dyck normal form.*

Proof. Suppose that G is a CFG in Chomsky normal form. Otherwise, using the algorithm as described in [10] or [14], we can convert G into Chomsky normal form. In order to convert G from Chomsky normal form into Dyck normal we proceed as follows.

Step 1 We check whether P contains two (or more) rules of the form $A \to a$, $A \to b$, $a \neq b$. If it does, then for each rule $A \to b$, $a \neq b$, we introduce a new variable A_b. We add the new rule $A_b \to b$, and we remove the rule $A \to b$. For each rule of the form $X \to AB$ ($X \to BA$) we add the new rule $X \to A_bB$ ($X \to BA_b$), while for a rule of the form $X \to AA$ we add three new rules $X \to A_bA$, $X \to AA_b$, $X \to A_bA_b$, without removing the initial rules. We call this procedure an *A_b-terminal substitution* of A.

For each rule $A \to a$, $a \in T$, we check whether a rule of the form $A \to B_1B_2$, $B_1, B_2 \in N$, exists in P. If it does, then a new nonterminal A_a is introduced and we perform an A_a-terminal substitution of A for the rule $A \to a$. ◇

Step 2 Suppose that there exist two (or more) rules of the form $X \to AB$ and $X' \to B'A$. If we have agreed on preserving only the left occurrences of A in the right-hand sides, then according to condition 3 of Definition 1, we have to remove all right occurrences of A. To do this we introduce a new nonterminal ${}_ZA$ and all right occurrences of A, preceded at the left side by Z, in the right-hand side of a rule, are substituted by ${}_ZA$. For each rule that rewrites A, $A \to Y$, $Y \in N^2 \cup T$, we add a new rule of the form ${}_ZA \to Y$, preserving the rule $A \to Y$. We call this procedure an *${}_ZA$-nonterminal substitution* of A. According to this procedure, for the rule $X' \to B'A$, we introduce a new nonterminal ${}_{B'}A$, we add the rule $X' \to B'{}_{B'}A$, and remove the rule $X' \to B'A$. For each rule that rewrites A, of the form[1] $A \to Y$, $Y \in N^2 \cup T$, we add a new rule of the form ${}_{B'}A \to Y$, preserving the rule $A \to Y$. ◇

Step 3 Finally, for each two rules $X \to AB$, $X' \to A'B$ ($X \to BA$, $X' \to BA'$) with $A \neq A'$, a new nonterminal ${}_{A'}B$ ($B_{A'}$) is introduced to replace B from the second rule, and we perform an ${}_{A'}B(B_{A'})$-nonterminal substitution of B, i.e., we add $X' \to A'{}_{A'}B$, and remove $X' \to A'B$. For each rule that rewrites B, of the form $B \to Y$, $Y \in N^2 \cup T$, we add a new rule ${}_{A'}B \to Y$ by preserving the rule $B \to Y$. In the case that A' occurs on the right-hand side of another rule, such that A' matches at the right side with another nonterminal different of ${}_{A'}B$, then the procedure described above is repeated for A', too. ◇

[1] This case deals with the possibility of having $Y = B'{}_{B'}A$, too.

If one of the conditions 2, 3, and 4 in Definition 1, has been settled, we do not have to resolve it once again in further steps of this procedure. The new grammar G' built as described in steps 1, 2, and 3 has the set of nonterminals N' and the set of productions P' composed of all nonterminals from N and productions from P, plus/minus all nonterminals and productions, respectively introduced/removed according to the substitutions performed during the above steps. Next we prove that grammars $G = (N, T, P, S)$, in Chomsky normal form, and $G' = (N', T, P', S)$, in Dyck normal form, generate the same language. In this order, consider the homomorphism $h_d : N' \cup T \to N \cup T$ defined by $h_d(x) = x$, $x \in T$, $h_d(X) = X$, for $X \in N$, and $h_d(X') = X$ for $X' \in N' - N$, $X \in N$ such that X' is a (transitive[2]) X'-substitution of X, terminal or not, in the above construction of the grammar G'.

To prove that $L(G') \subseteq L(G)$ we extend h_d to a homomorphism from $(N' \cup T)^*$ to $(N \cup T)^*$ defined on the classical concatenation operation. It is straightforward to prove by induction, that for each $\alpha \Rightarrow^*_{G'} \delta$ we have $h_d(\alpha) \Rightarrow^*_G h_d(\delta)$. This implies that for any derivation of a word $w \in L(G')$, i.e., $S \Rightarrow^*_{G'} w$, we have $h_d(S) \Rightarrow^*_G h_d(w)$, i.e., $S \Rightarrow^*_G w$, or equivalently, $L(G') \subseteq L(G)$.

To prove that $L(G) \subseteq L(G')$ we make use of the CYK (Cocke-Younger-Kasami) algorithm as described in [14].

Let $w = a_1 a_2 ... a_n$ be an arbitrary word from $L(G)$, and V_{ij}, $i \leq j$, $i, j \in \{1, ..., n\}$, be the triangular matrix of size $n \times n$ built with the CYK algorithm (see [14]). Since $w \in L(G)$, we have $S \in V_{1n}$. We prove that $w \in L(G')$, i.e., $S \in V'_{1n}$, where V'_{ij}, $i \leq j$, $i, j \in \{1, ..., n\}$ forms the triangular matrix obtained by applying the CYK algorithm to w according to G' productions.

We consider two relations $\hat{h}_t : N \cup T \to N' \cup T$ and $\hat{h}_{\neg t} : N \to N'$. The first relation is defined by $\hat{h}_t(x) = x$, $x \in T$, $\hat{h}_t(S) = S$, if $S \to t$, $t \in T$, is a rule in G, and $\hat{h}_t(X) = X'$, if X' is a (transitive) X'-terminal substitution[3] of X, and $X \to t$ is a rule in G. Finally, $\hat{h}_t(X) = X$ if $X \to t \in P$, $t \in T$. The second relation is defined as $\hat{h}_{\neg t}(S) = S$, $\hat{h}_{\neg t}(X) = \{X\} \cup \{X' | X'$ is a (transitive) X'-nonterminal substitution of $X\}$ and $\hat{h}_{\neg t}(X) = X$, if there is no substitution of X and no rule of the form $X \to t$, $t \in T$, in G. Notice that $\hat{h}_x(X_1 \cup X_2) = \hat{h}_x(X_1) \cup \hat{h}_x(X_2)$, for $X_i \subseteq N$, $i \in \{1, 2\}$, $x \in \{t, \neg t\}$. Furthermore, using the relation $\hat{h}_t$, each rule $X \to t$ in P has a corresponding set of rules $\{X' \to t | X' \in \hat{h}_t(X), X \to t \in P\}$ in P'. Each rule $A \to BC$ in P has a corresponding set of rules $\{A' \to B'C' | A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), B'$ and C' are pairwise nonterminals, $A \to BC \in P\}$ is in P'.

Consider $V'_{ii} = \hat{h}_t(V_{ii})$ and $V'_{ij} = \hat{h}_{\neg t}(V_{ij})$, $i < j$, $i, j \in \{1, ..., n\}$. We claim that V'_{ij}, $i, j \in \{1, ..., n\}$, $i \leq j$, defined as before, forms the triangular matrix obtained by applying the CYK algorithm to rules that derive w in G'.

---

[2] There exist $X_k \in N$, such that X' is an X'-substitution of X_k, X_k is an X_k-substitution of X_{k-1},..., and X_1 is an X_1-substitution of X. All of them substitute X.

[3] There may exist several (distinct) terminal (nonterminal) substitutions for the same nonterminal X. This property makes $\hat{h}_t$ ($\hat{h}_{\neg t}$) to be a relation.

First, observe that for $i = j$, we have $V'_{ii} = \hat{h}_t(V_{ii}) = \{A|A \to a_i \in P'\}$, $i \in \{1, ..., n\}$, due to the definition of the relation $\hat{h}_t$. Now let us consider $k = j-i$, $k \in \{1, ..., n-1\}$. We want to compute V'_{ij}, $i < j$.

By definition, we have $V_{ij} = \bigcup_{l=i}^{j-1}\{A|A \to BC, B \in V_{il}, C \in V_{l+1j}\}$, so that $V'_{ij} = \hat{h}_{\neg t}(V_{ij}) = \hat{h}_{\neg t}(\bigcup_{l=i}^{j-1}\{A|A \to BC, B \in V_{il}, C \in V_{l+1j}\}) = \bigcup_{l=i}^{j-1}\hat{h}_{\neg t}(\{A|A \to BC, B \in V_{il}, C \in V_{l+1j}\}) = \bigcup_{l=i}^{j-1}\{A'|A' \to B'C', A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}$, B' and C' are pairwise nonterminals, $A \to BC \in P\}$. Let us explicitly develop the last union.

If $k = 1$, then $l \in \{i\}$. For each $i \in \{1, ..., n-1\}$ we have $V'_{ii+1} = \{A'|A' \to B'C', A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{ii}, C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{i+1i+1}$, B' and C' are pairwise nonterminals, $A \to BC \in P\}$. Due to the fact that $B \in V_{ii}$ and $C \in V_{i+1i+1}$, B' is a terminal substitution of B, while C' is a terminal substitution of C. Therefore, we have $B' \notin \hat{h}_{\neg t}(B)$, $C' \notin \hat{h}_{\neg t}(C)$, so that $B' \in \hat{h}_t(B)$, for all $B \in V_{ii}$, and $C' \in \hat{h}_t(C)$, for all $C \in V_{i+1i+1}$, i.e., $B' \in \hat{h}_t(V_{ii}) = V'_{ii}$ and $C' \in \hat{h}_t(V_{i+1i+1}) = V'_{i+1i+1}$. Therefore, $V'_{ii+1} = \{A'|A' \to B'C', B' \in V'_{ii}, C' \in V'_{i+1i+1}\}$.

If $k \geq 2$, then $l \in \{i, i+1, ..., j-1\}$, and $V'_{ij} = \bigcup_{l=i}^{j-1}\{A'|A' \to B'C', A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}$, B' and C' are pairwise nonterminals, $A \to BC \in P\}$. We now compute the first set of the above union, i.e., $V'_i = \{A'|A' \to B'C', A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{ii}, C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{i+1j}$, B' and C' are pairwise nonterminals, $A \to BC \in P\}$. By the same reasoning as before, the condition $B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{ii}$, is equivalent with $B' \in \hat{h}_t(V_{ii}) = V'_{ii}$.

Because $i + 1 \neq j$, C' is a nonterminal substitution of C. Therefore, $C' \notin \hat{h}_t(C)$, and the condition $C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{i+1j}$ is equivalent with $C' \in \hat{h}_{\neg t}(V_{i+1j}) = V'_{i+1j}$. So that $V'_i = \{A'|A' \to B'C', B' \in V'_{ii}, C' \in V'_{i+1j}\}$.

Using the same method for each $l \in \{i+1, ..., j-1\}$ we have $V'_l = \{A'|A' \to B'C', A' \in \hat{h}_{\neg t}(A), B' \in \hat{h}_{\neg t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{\neg t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}$, B' and C' are pairwise nonterminals, $A \to BC \in P\} = \{A'|A' \to B'C', B' \in V'_{il}, C' \in V'_{l+1j}\}$. In conclusion, $V'_{ij} = \bigcup_{l=i}^{j-1}\{A'|A' \to B'C', B' \in V'_{il}, C' \in V'_{l+1j}\}$, for each $i, j \in \{1, ..., n\}$, i.e., V'_{ij}, $i \leq j$, contains the nonterminals of the $n \times n$ triangular matrix computed by applying the CYK algorithm to rules that derive w in G'. Because $w \in L(G)$, we have $S \in V_{1n}$. That is equivalent with $S \in V'_{1n} = \hat{h}_t(V_{1n})$, if $n = 1$, and $S \in V'_{1n} = \hat{h}_{\neg t}(V_{1n})$, if $n > 1$, i.e., $w \in L(G')$. □

Corollary 1. *Let G be a CFG in Dyck normal form. Any terminal derivation in G producing a word of length n, $n \geq 1$, takes $2n-1$ steps.*

Proof. If G is a CFG in Dyck normal form, then it is also in Chomsky normal form, and all properties of the latter hold. □

Corollary 2. *If $G = (N, T, P, S)$ is a grammar in Chomsky normal form, and $G' = (N', T, P', S)$ its equivalent in Dyck normal form, then there exists a homomorphism $h_d : N' \cup T \to N \cup T$, such that any derivation tree of $w \in L(G)$ is the homomorphic image of a derivation tree of the same word in G'.*

Proof. Consider the homomorphism $h_d : N' \cup T \to N \cup T$ defined as $h_d(A_t) = h_d({}_ZA) = h_d(A_Z) = A$, for each A_t-terminal or ${}_ZA(A_Z)$-nonterminal substitution of A, and $h_d(t) = t$, $t \in T$. The claim is a direct consequence of the way in which the new nonterminals A_t, ${}_ZA$, and A_Z have been chosen. □

Let G be a grammar in Dyck normal form. In order to emphasis the pairwise brackets occurring on the right-hand side of each rule, each nonterminal A and B occurring on a rule of the form $X \to AB$, is replaced by a left and right bracket $[_A$ and $]_B$, respectively. In each rule that rewrites A or B, we replace A by $[_A$, and B by $]_B$, respectively. Next, we present an example of the conversion procedure described in Theorem 1, along with the homomorphism that lays between the grammars in Chomsky and Dyck normal forms.

Example 1 Consider the CFG $G = (\{E, T, R\}, \{+, *, a\}, E, P)$ with $P = \{E \to a/T * R/E + T, T \to a/T * R, R \to a\}$.

The Chomsky normal form of G is $G' = (\{E_0, E, E_1, E_2, T, T_1, T_2, R\}, \{+, *, a\}, E_0, P')$ in which $P' = \{E_0 \to a/TT_1/EE_1, E \to a/TT_1/EE_1, T \to a/TT_1, T_1 \to T_2R, E_1 \to E_2T, T_2 \to *, E_2 \to +, R \to a\}$.

We now convert G' into Dyck normal form. To do this, with respect to Definition 1, item 2, we first have to remove $E \to a$ and $T \to a$. Then, according to item 3, we remove the right occurrence of T from the rule $E_1 \to E_2T$, along with other transformations that may be required after completing these procedures. Let E_3 and T_3 be two new nonterminals. We remove $E \to a$ and $T \to a$, and add the rules $E_3 \to a$, $T_3 \to a$, $E_0 \to E_3E_1$, $E_0 \to T_3T_1$, $E \to E_3E_1$, $E \to T_3T_1$, $E_1 \to E_2T_3$, $T \to T_3T_1$.

Let T' be the new nonterminal that replaces the right occurrence of T. We add the rules $E_1 \to E_2T'$, $T' \to TT_1$, $T' \to T_3T_1$, and remove $E_1 \to E_2T$. We repeat the procedure with T_3 (added in the previous step), i.e., we introduce a new nonterminal T_4, remove $E_1 \to E_2T_3$, add $E_1 \to E_2T_4$ and $T_4 \to a$.

Due to the new nonterminals E_3, T_3, T_4, item 4 does not hold. To have accomplished this condition too, we introduce three new nonterminals E_4 to replace E_2 in $E_1 \to E_2T_4$, E_5 to replace E_1 in $E_0 \to E_3E_1$ and $E \to E_3E_1$, and T_5 to replace T_1 in $E_0 \to T_3T_1$ and $E \to T_3T_1$. We remove all the above rules and add the new rules $E_1 \to E_4T_4$, $E_4 \to +$, $E_0 \to E_3E_5$, $E \to E_3E_5$, $E_5 \to E_2T'$, $E_5 \to E_4T_4$, $E_0 \to T_3T_5$, $E \to T_3T_5$, and $T_5 \to T_2R$. The Dyck normal form of G', using the bracket notation, is

$G'' = (\{E_0, [_E,]_{E_1}, [_{E_2}, [_{E_3}, [_{E_4},]_{E_5},]_{T'}, [_T,]_{T_1}, [_{T_2}, [_{T_3},]_{T_4},]_R\}, \{+, *, a\}, E_0, P'')$,

$P'' = \{E_0 \to a/[_T\,]_{T_1}/[_E\,]_{E_1}/[_{E_3}\,]_{E_5}/[_{T_3}\,]_{T_5},\ [_E \to [_T\,]_{T_1}/[_E\,]_{E_1}/[_{E_3}\,]_{E_5}/[_{T_3}\,]_{T_5},$
$]_{E_1} \to [_{E_2}\,]_{T'}/[_{E_4}\,]_{T_4},\ ]_{E_5} \to [_{E_2}\,]_{T'}/[_{E_4}\,]_{T_4},\ [_T \to [_T\,]_{T_1}/[_{T_3}\,]_{T_5},$
$]_{T'} \to [_T\,]_{T_1}/[_{T_3}\,]_{T_5},\ ]_{T_1} \to [_{T_2}\,]_R,\ ]_{T_5} \to [_{T_2}\,]_R,$
$[_{T_2} \to *,\ [_{T_3} \to a,\ ]_{T_4} \to a,\ [_{E_2} \to +,\ [_{E_3} \to a,\ [_{E_4} \to +,\ ]_R \to a\}$.

The homomorphism h_d is defined as $h_d : N' \cup T \to N'' \cup T$, $h_d(E_0) = E_0$, $h_d([_E) = h_d([_{E_3}) = E$, $h_d(]_{E_1}) = h_d(]_{E_5}) = E_1$, $h_d([_{E_2}) = h_d([_{E_4}) = E_2$, $h_d([_T) = h_d(]_{T'}) = h_d([_{T_3}) = h_d(]_{T_4}) = T$, $h_d(]_{T_1}) = h_d(]_{T_5}) = T_1$, $h_d([_{T_2}) = T_2$, $h_d(]_R) = R$, $h_d(t) = t$, for each $t \in T$.

The string $w = a*a*a+a$ is a word in $L(G'') = L(G)$ generated, for instance, by a leftmost derivation D in G'' as follows.

$D : E_0 \Rightarrow [_E\ ]_{E_1} \Rightarrow [_T\ ]_{T_1}\ ]_{E_1} \Rightarrow [_{T_3}\ ]_{T_5}\ ]_{T_1}\ ]_{E_1} \Rightarrow a\ ]_{T_5}\ ]_{T_1}\ ]_{E_1} \Rightarrow a\ [_{T_2}\ ]_R\ ]_{T_1}\ ]_{E_1}$
$\Rightarrow\ a *]_R\ ]_{T_1}\ ]_{E_1}\ \Rightarrow\ a\ *\ a\ ]_{T_1}\ ]_{E_1}\ \Rightarrow\ a\ *\ a\ [_{T_2}\ ]_R\ ]_{E_1}\ \Rightarrow\ a\ *\ a *]_R\ ]_{E_1}\ \Rightarrow$
$a\ *\ a\ *\ a\ ]_{E_1} \Rightarrow a\ *\ a\ *\ a\ [_{E_4}\ ]_{T_4} \Rightarrow a\ *\ a\ *\ a +]_{T_4} \Rightarrow a\ *\ a\ *\ a\ +\ a.$

Applying h_d to the derivation D of w in G'' we obtain a derivation of w in G'. If we consider $\mathcal{T}$ the derivation tree of w in G, and $\mathcal{T}'$ the derivation tree of w in G'', then $\mathcal{T}$ is the homomorphic image of $\mathcal{T}'$ through h_d. ♣

A normal form is correct if it preserves the language generated by the original grammar. This condition is called *the weak equivalence*, i.e., a normal form preserves the language but may lose several other (important) syntactical or semantical properties of the original grammar. It is well known that the Chomsky normal form is a "strong" normal form in this respect. It preserves almost all syntactical and semantical properties of the original grammar.

The Dyck normal form is at least as powerful as the Chomsky normal form, not only because of the "tree homomorphism" existing between a grammar in Chomsky normal form and its equivalent in Dyck normal, but also because the Dyck normal form, due to the pairwise structure of the derivation, makes visible possible correlations between linear languages or CFLs, and Dyck words and Dyck languages. It is known that Dyck words have very interesting combinatorial structures and properties (see [7], [8]), while Dyck languages are of a low complexity, i.e., they belong to the NC^1 class (see [12]). The main aim of this paper is to prove that an homomorphic relationship between Dyck languages and linear languages makes LIN to collapse inside NC^1.

This shows that looking for new normal forms can still be considered a vivid topic in formal language theory. It is challenging to see how far can we go with a normal form in order to preserve or emphasis syntactical properties (like derivation tree structures) and semantical properties (like ambiguities) of the original grammars. Actually, these preservation properties make a normal form stronger and more useful in fields such as descriptional and computational complexity, pattern matchings and parsing, inference and learning theory.

3 Characterizations of CFLs by Dyck languages

Definition 2. Let $G_k = (N_k, T, P_k, S)$ be a CFG in Dyck normal form with $|N_k - \{S\}| = 2k$. Let $D : S \Rightarrow u_1 \Rightarrow u_2 \Rightarrow ... \Rightarrow u_n = w$, $n \geq 2$, be a leftmost derivation of $w \in L(G)$. The *trace-word* of w associated with the derivation D, denoted as $t_{w,D}$, is defined as the concatenation of nonterminals consecutively rewritten in D, excluding the axiom. The *trace-language* associated with G_k, denoted as $Ł_k$, is $Ł_k = \{t_{w,D} | D$ is a leftmost derivation of $w, w \in L(G_k)\}$.

The trace-word associated to w and the leftmost derivation D in Example 1 is $t_{a*a*a+a,D} = [_E\ [_T\ [_{T_3}\ ]_{T_5}\ [_{T_2}\ ]_R\ ]_{T_1}\ [_{T_2}\ ]_R\ ]_{E_1}\ [_{E_4}\ ]_{T_4}$.

Definition 3. A one-sided Dyck language over k letters, $k \geq 1$, is a context-free language defined by the grammar $\Gamma_k = (\{S\}, T_k, P, S)$, where $T_k = \{[_1,\ [_2, ..., [_k,\ ]_1,\ ]_2, ...,\ ]_k\}$ and $P = \{S \to [_i\ S\ ]_i, S \to SS, S \to [_i\ ]_i \mid 1 \leq i \leq k\}$.

Let $G_k = (N_k, T, P_k, S)$ be a CFG in Dyck normal form. In order to emphasize possible relations between the structure of trace-words from Ł_k and the structure of words belonging to the Dyck language, and also to keep control of each bracketed pair occurring in the right-hand side of each rule from G_k, we fix $N_k = \{S, [_1,\ [_2, ..., [_k,\ ]_1,\ ]_2, ...,\ ]_k\}$, and P_k to be composed of rules the of the form $X \to [_i\ ]_i$, $1 \leq i \leq k$, and $Y \to t$, $X, Y \in N_k$, $t \in T$.

Note that $t_{w,D}$, $w \in L(G)$, can also be read from the derivation tree in depth-first order starting with the root, but ignoring the root and the leaves. In [12] it is proved that D_k can be recognized by an alternating Turing machine (ATM) in $\mathcal{O}(\log n)$ time and space. The proof is based on a characterization of D_k, $k \geq 1$, presented in this paper through Definition 4, and Lemmas 1 and 2.

Definition 4. For a string w, let $w_{i:j}$ be its substring starting at the i^{th} position and ending at the j^{th} position. Let h be a homomorphism defined as follows:

$h([_1) = h([_2) = ... = h([_k) = [_1, \qquad h(]_1) = h(]_2) = ... = h(]_k) =]_1$.

Let $w \in D_k$, $1 \leq i \leq j \leq |w|$, where $|w|$ is the length of w. We say that $(i,\ j)$ is a *matched pair* of w, if $h(w_{i:j})$ is *balanced*, i.e., $h(w_{i:j})$ has an equal number of $[_1$'s and $]_1$'s and, in any prefix of $h(w_{i:j})$, the number of $[_1$'s is greater than or equal to the number of $]_1$'s.

Lemma 1. *A string $w \in \{[_1\ ,]_1\}^*$ is in D_1 if and only if it is balanced.*

Consider the homomorphisms defined as follows (where λ is the empty string)

$h_1([_1) = [_1, \quad h_1(]_1) =]_1, \quad h_1([_2) = h_1(]_2) = ... = h_1([_k) = h_1(]_k) = \lambda,$
$h_2([_2) = [_1, \quad h_2(]_2) =]_1, \quad h_2([_1) = h_2(]_1) = ... = h_2([_k) = h_2(]_k) = \lambda,$
. . . . . .
$h_k([_k) = [_1, \quad h_k(]_k) =]_1, \quad h_k([_1) = h_k(]_1) = ... = h_k([_{k-1}) = h_k(]_{k-1}) = \lambda.$

Lemma 2. *We have $w \in D_k$, $k \geq 2$, if and only if the following conditions hold: i) $(1,\ |w|)$ is a matched pair, and ii) for all matched pairs $(i,\ j)$, $h_k(w_{i:j})$ are in D_1, where $k \geq 1$.*

Definition 5. Let $w \in D_k$, $(i,\ j)$ is a *nested pair* of w if $(i,\ j)$ is a matched pair, and either $j = i + 1$, or $(i + 1,\ j - 1)$ is a matched pair.

Definition 6. Let $w \in D_k$, and $(i,\ j)$ be a matched pair of w. We say that $(i,\ j)$ is *reducible* if there exists an integer j', $i < j' < j$, such that $(i,\ j')$ and $(j' + 1,\ j)$ are matched pairs of w.

Consider $w \in D_k$, if $(i,\ j)$ is a nested pair of w then $(i,\ j)$ is an irreducible pair. If $(i,\ j)$ is a nested pair of w then $(i + 1,\ j - 1)$ may be a reducible pair.

Theorem 2. *The trace-language associated with a CFG, $G = (N_k, T, P_k, S)$ in Dyck normal form, with $|N_k| = 2k + 1$, is a subset of the D_k.*

Proof. Let $N_k = \{S, [_1, ..., [_k, \ ]_1, ..., \ ]_k\}$ be the set of nonterminals, $w \in L(G)$, and D a leftmost derivation of w. We show that any subtree of the derivation tree, read in the depth-first order, by ignoring the root and the terminal nodes, corresponds to a matched pair in $t_{w,D}$. In particular, $(1, |t_{w,D}|)$ will be a matched pair. Denote by $t_{w,D_{i:j}}$ the substring of $t_{w,D}$ starting at the i^{th} position and ending at the j^{th} position of $t_{w,D}$. We show that for all matched pairs (i, j), $h_{k'}(t_{w,D_{i:j}})$ belong to D_1, $1 \leq k' \leq k$. We prove these claims by induction on the height of subtrees.

Basis. Certainly, any subtree of height $n = 1$, read in the depth-first order, looks like $[_i \ ]_i$, $1 \leq i \leq k$. Therefore, it satisfies the above conditions.

Induction step. Assume that the claim is true for all subtrees of height h, $h < n$, and prove it for $h = n$. Each subtree of height n can have one of the following structures. The level 0 of the subtree is marked by a left or right bracket. This bracket will not be considered when we read the subtree. Denote by $[_m$ the left son of the root. Then the right son is labeled by $]_m$. They are the roots of a left and right subtree, for which at least one has the height $n - 1$.

Suppose that both subtrees have the height $1 \leq h \leq n - 1$. By the induction hypothesis, let us suppose further that the left subtree corresponds to the matched pair (i_l, j_l), and the right subtree corresponds to the matched pair (i_r, j_r), $i_r = j_l + 2$, because the position $j_l + 1$ is taken by $]_m$.

As h is a homomorphism, we have $h(t_{w,D_{i_l-1:j_r}}) = h([_m t_{w,D_{i_l:j_l}}]_m t_{w,D_{j_l+2:j_r}}) = h([_m)h(t_{w,D_{i_l:j_l}})h(]_m)h(t_{w,D_{j_l+2:j_r}})$. Therefore, $h(t_{w,D_{i_l-1:j_r}})$ satisfies all conditions in Definition 4, and thus $(i_l - 1, j_r)$ that corresponds to the considered subtree of height n, is a matched pair.

Also, by the induction hypothesis, $h_{k'}(t_{w,D_{i_l:j_l}})$ and $h_{k'}(t_{w,D_{i_r:j_r}})$ are in D_1, $1 \leq k' \leq k$. Hence, $h_{k'}(t_{w,D_{i_l-1:j_r}}) = h_{k'}([_m)h_{k'}(t_{w,D_{i_l:j_l}})h_{k'}(]_m)h_{k'}(t_{w,D_{j_l+2:j_r}}) \in \{h_{k'}(t_{w,D_{i_l:j_l}})h_{k'}(t_{w,D_{j_l+2:j_r}}), [_1 h_{k'}(t_{w,D_{i_l:j_l}})]_1 h_{k'}(t_{w,D_{j_l+2:j_r}})\}$ belong to D_1.

Note that in this case the matched pair $(i_l - 1, j_r)$ is reducible into $(i_l - 1, j_l + 1)$ and $(j_l + 2, j_r)$, where $(i_l - 1, j_l + 1)$ corresponds to the substring $t_{w,D_{i_l-1:j_l+1}} = [_m t_{w,D_{i_l:j_l}}]_m$. We refer to this structure as the *left embedded subtree*, i.e., $(i_l - 1, j_l + 1)$ is a nested pair. A similar reasoning is applied for the case when one of the subtrees has the height 0. Analogously, it can be shown that the initial tree corresponds to the matched pair $(1, |t_{w,D}|)$, i.e., the first condition of Lemma 2 holds.

So far, we have proved that each subtree of the derivation tree, and also each left embedded subtree, corresponds to a matched pair (i, j) and (i_l, j_l), such that $h_{k'}(t_{w,D_{i:j}})$ and $h_{k'}([_m t_{w,D_{i_l:j_l}}]_m)$, $1 \leq k' \leq k$, are in D_1.

Next we show that all matched pairs from $t_{w,D}$ correspond only to subtrees, or left embedded subtrees, from the derivation tree. To derive a contradiction, let us suppose that there exists a matched pair (i, j) in $t_{w,D}$, that does not correspond to any subtree, or left embedded subtree, of the derivation tree read in the depth-first order. We show that this leads to a contradiction.

Since (i, j) does not correspond to any subtree, or left embedded subtree, there exist two adjacent subtrees θ_1 (a left embedded subtree) and θ_2 (a right subtree) such that (i, j) is composed of two adjacent "subparts" of θ_1 and θ_2.

In terms of matched pairs, if θ_1 corresponds to the matched pair (i_1, j_1) and θ_2 corresponds to the matched pair (i_2, j_2), such that $i_2 = j_1 + 2$, then there exists a suffix $s_{i_1-1:j_1+1}$ of $t_{w,D_{i_1-1:j_1+1}}$, and a prefix $p_{i_2:j_2}$ of $t_{w,D_{i_2:j_2}}$, such that $t_{w,D_{i:j}} = s_{i_1-1:j_1+1}p_{i_2:j_2}$. Furthermore, without loss of generality, we assume that (i_1, j_1) and (i_2, j_2) are nested pairs. Otherwise, the matched pair (i, j) can be "narrowed" until θ_1 and θ_2 are characterized by two nested pairs. If (i_1, j_1) is a nested pair, then so is $(i_1 - 1, j_1 + 1)$.

Since $s_{i_1-1:j_1+1}$ is a suffix of $t_{w,D_{i_1-1:j_1+1}}$ and $(i_1 - 1, j_1 + 1)$ is a matched pair, with respect to Definition 4, the number of $]_1$'s in $h(s_{i_1-1:j_1+1})$ is greater than or equal to the number of $[_1$'s in $h(s_{i_1-1:j_1+1})$. On the other hand, $s_{i_1-1:j_1+1}$ is also a prefix of $t_{w,D_{i:j}}$, because (i, j) is a matched pair, by our hypothesis. Therefore, the number of $[_1$'s in $h(s_{i_1-1:j_1+1})$ is greater than or equal to the number of $]_1$'s in $h(s_{i_1-1:j_1+1})$

Hence, the only possibility for $s_{i_1-1:j_1+1}$ to be a suffix for $t_{w,D_{i_1-1:j_1+1}}$ and a prefix for $t_{w,D_{i:j}}$ is the equality between the number of $[_1$'s and $]_1$'s in $h(s_{i_1-1:j_1+1})$. This property holds if and only if $s_{i_1-1:j_1+1}$ corresponds to a matched pair in $t_{w,D_{i_1-1:j_1+1}}$, i.e., if i_s and j_s are the start and the end positions of $s_{i_1-1:j_1+1}$ in $t_{w,D_{i_1-1:j_1+1}}$, then (i_s, j_s) is a matched pair. Thus, (i_1-1, j_1+1) is a reducible pair into $(i_1 - 1, i_s - 1)$ and (i_s, j_s), where $j_s = j_1 + 1$. We have reached a contradiction, i.e., $(i_1 - 1, j_1 + 1)$ is reducible.

Therefore, the matched pairs in $t_{w,D}$ correspond to subtrees, or left embedded subtrees, in the derivation tree. For these matched pairs we have already proved that they satisfy Lemma 2. Accordingly, $t_{w,D} \in D_k$, and consequently the trace-language associated with G is a subset of D_k. □

Theorem 3. *Given a CFG G there exist an integer K, a homomorphism φ, and a subset D'_K of the Dyck language D_K, such that $L(G) = \varphi(D'_K)$.*

Proof. Let G be a CFG and $G_k = (N_k, T, P_k, S)$ be the Dyck normal form of G, such that $N_k = \{S, [_1, [_2, ..., [_k,]_1,]_2, ...,]_k\}$. Let Ł$_k$ be the trace-language associated with G_k. If $\{t_{k+1}, ..., t_{k+p}\}$ is an ordered subset of T, such that $S \to t_{k+i} \in P$, $1 \leq i \leq p$, then let $N_{k+p} = N_k \cup \{[_{t_{k+1}}, ..., [_{t_{k+p}},]_{t_{k+1}}, ...]_{t_{k+p}}\}$, and $P_{k+p} = P_k \cup \{S \to [_{t_{k+i}}]_{t_{k+i}}, [_{t_{k+i}} \to t_{k+i},]_{t_{k+i}} \to \lambda | S \to t_{k+i} \in P, 1 \leq i \leq p\}$. Certainly, the new grammar $G_{k+p} = (N_{k+p}, T, P_{k+p}, S)$ generates the same language as G_k.

Let $\varphi : (N_{k+p} - \{S\})^* \to T^*$ be the homomorphism defined by $\varphi(N) = \lambda$, for each rule of the form $N \to XY$, $N, X, Y \in N_k - \{S\}$, and $\varphi(N) = t$, for each rule of the form $N \to t$, $N \in N_k - \{S\}$, and $t \in T$, $\varphi([_{k+i}) = t_{k+i}$, and $\varphi(]_{k+i}) = \lambda$, for each $1 \leq i \leq p$.

It is evident that $L = \varphi(D'_K)$, where $K = k + p$, $D'_K = $ Ł$_k \cup L_p$, and $L_p = \{[_{t_{k+1}}\,]_{t_{k+1}}, ..., [_{t_{k+p}}\,]_{t_{k+p}}\}$. □

4 LIN is in NC[1]

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary CFG in Dyck normal form, with $N_k = \{S, [_1, [_2, ..., [_k, \ ]_1,]_2, ...,]_k\}$. Let $\varphi : N_k^* \to (T \cup \{S\})^*$ be a variant of the homomorphism introduced in the proof of Theorem 3, defined as follows: $\varphi(\lambda) = \lambda$, $\varphi(S) = S$, $\varphi(N) = \lambda$, for each rule of the form $N \to XY$, $N, X, Y \in N_k$, $N \neq S$, and $\varphi(N) = t$, for each rule of the form $N \to t$, $N \in N_k$, $N \neq S$, $t \in T$. Next we divide N_k into three main sets $N^{(1)}$, $N^{(2)}$, $N^{(3)}$ as follows

1. $[_i$ and $]_i$ belong to $N^{(1)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = t'$, $t, t' \in T$,
2. $[_i$ and $]_i$ belong to $N^{(2)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = \lambda$, or vice versa $\varphi([_i) = \lambda$ and $\varphi(]_i) = t$, $t \in T$,
3. $[_i,]_i \in N^{(3)}$ if and only if $\varphi([_i) = \lambda$ and $\varphi(]_i) = \lambda$.

Certainly, $N_k - \{S\} = N^{(1)} \cup N^{(2)} \cup N^{(3)}$ and $N^{(1)} \cap N^{(2)} \cap N^{(3)} = \emptyset$. The set $N^{(2)}$ is further divided into $N_l^{(2)}$ and $N_r^{(2)}$. The subset $N_l^{(2)}$ contains those pairs $([_i,]_i) \in N^{(2)}$ such that $\varphi([_i) \neq \lambda$, while $N_r^{(2)}$ contains those pairs $([_i,]_i) \in N^{(2)}$ such that $\varphi(]_i) \neq \lambda$. We have $N^{(2)} = N_l^{(2)} \cup N_r^{(2)}$ and $N_l^{(2)} \cap N_r^{(2)} = \emptyset$. For the sake of simplicity, for each two brackets $[_i$ and $]_i$ belonging to X, $X \in \{N^{(1)}, N^{(2)}, N^{(3)}\}$, we use the notation $([_i,]_i) \in X$. In this section we deal only with regular and linear context-free languages, and grammars in right, left and linear-Dyck normal form defined below. A linear CFG is denoted as LCFG.

Definition 7. *A grammar G_k is in right-Dyck normal form if G_k is in Dyck normal form and $N^{(3)} = N_r^{(2)} = \emptyset$. A grammar G_k is in left-Dyck normal form if G_k is in Dyck normal form and $N^{(3)} = N_l^{(2)} = \emptyset$. A grammar G_k is in linear-Dyck normal form if G_k is in Dyck normal form and $N^{(3)} = \emptyset$.*

Lemma 3. *For each right-linear grammar G, there exits a grammar G_k in right-Dyck normal form such that $L(G) = L(G_k)$.*

Proof. Each right-linear grammar G can be written in an equivalent standard form by using only rules of the type $X \to \lambda$, $X \to t$, $X \to tY$, $t \in T$, $X, Y \in N$. Transforming G from standard form into Chomsky normal form, and then into the Dyck normal form, we obtain a grammar G_k in right-Dyck normal form. Since each standard, Chomsky, and Dyck normal form are weakly equivalent with the initial grammar we obtain $L(G) = L(G_k)$. □

Lemma 4. *For each left-linear grammar G, there exits a grammar G_k in left-Dyck normal form such that $L(G) = L(G_k)$.*

Corollary 3. *REG is equal with the family of languages generated by grammars in right-Dyck normal form or in left-Dyck normal form.*

Lemma 5. *For each linear grammar G, there exits a grammar G_k in linear-Dyck normal form such that $L(G) = L(G_k)$.*

Proof. Each linear grammar G can be written in an equivalent standard form by using only rules of the form $X \to \lambda$, $X \to t$, $X \to t_1Y$, $X \to Yt_2$, $X \to t_1Yt_2$, $t, t_1, t_2 \in T$, X, $Y \in N$. Transforming G from standard form, into Chomsky normal form, and then into the Dyck normal form, we obtain a grammar G_k in linear-Dyck normal form. Since each standard form for linear languages, Chomsky, and Dyck normal form are weakly equivalent we obtain $L(G) = L(G_k)$. □

Corollary 4. *LIN is equal with the family of languages generated by grammars in linear-Dyck normal form.*

In what follows we consider more closely the structure of the derivation tree of a LCFG in linear-Dyck normal form.

Let G be a LCFG and $G_k = (N_k, T, P_k, S)$ be the linear-Dyck normal form of G, with $N_k = \{S, [_1, [_2, ..., [_k, \ ]_1,]_2, ..., \ ]_k\}$, such that both $N_r^{(2)}$ and $N_l^{(2)}$ are non-empty sets. Let $L(G_k)$ be the language generated by G_k. Each word $w = a_1a_2...a_n$, of an arbitrary length n, belonging to $L(G_k)$, has the property that there exists an index n_t, $1 \leq n_t \leq n-1$, and a pair $([_j^t, \ ]_j^t) \in N^{(1)}$, such that $[_j^t \to a_{n_t}$ and $]_j^t \to a_{n_t+1}$. Using the homomorphism φ, this is equivalent with $\varphi([_j^t) = a_{n_t}$ and $\varphi(]_j^t) = a_{n_t+1}$. For the position n_t already "marked", there is no other position in w with the above property.

We call the index n_t the *core index* of w, and $a_{n_t}a_{n_t+1}$ the *core segment* of w. However, because linear languages are ambiguous, there may exist several other "unique" positions inside w in which the core index can be reached. If we develop the derivation tree of w, for a core index already "decided", it may be observed that the pair $([_j^t,]_j^t)$ generating $a_{n_t}a_{n_t+1}$, is placed at the bottom of the derivation tree of w. Each symbol in this tree is generated within two steps of derivation (because the set $N_l^{(3)} = \emptyset$), therefore the height of the derivation tree is $n-1$. Each level, excepting the last one, is composed either of a bracketed pair from $N_l^{(2)}$ or of a bracketed pair from $N_r^{(2)}$. The last level contains the bracketed pair from $N^{(1)}$, i.e., the two symbols from w holding on the core segment. The prefix of w, $w_{pf} = a_1a_2...a_{n_t-1}$ can be read from the left side of the derivation tree, while the suffix of w, $w_{sf} = a_{n_t+2}...a_{n-1}a_n$ can be read from the right side of the derivation tree. The set of all prefixes and suffixes of w, of type w_{pf} and w_{sf}, respectively, when $w \in L(G_k)$, i.e., the languages

$L_{pf} = \{w_{pf} | w_{pf} = a_1a_2...a_{n_t-1}, w \in L(G_k), n_t \text{ is the core index of } w\}$, and

$L_{sf} = \{w_{sf} | w_{sf} = a_{n_t+2}...a_{n-1}a_n, \ w \in L(G_k), n_t \text{ is the core index of } w\}$,

are regular. Furthermore, each of the left and right regular set, L_{pf} and L_{sf}, is a "shuffle" of a finite number of left and right regular languages, respectively. More precisely, each linear grammar in linear-Dyck normal form can be split into a finite number of right and left linear grammars in right and left Dyck normal form, respectively, such that L_{pf} is a "shuffle" of the finite number of languages generated by right-linear grammars and L_{sf} is a "shuffle" of the finite number of languages generated by left-linear grammars.

The idea is to use an indexing ATM, denoted as $\mathcal{A}$, such that if $w \in T^*$, $w = a_1a_2...a_n$ is a string of an arbitrary length (that we would like to have it in $L(G_k)$), then $\mathcal{A}$ first guesses the core index n_t. Then $\mathcal{A}$ guesses possible words

belonging to the right-linear languages that are subwords of $w_{pf} = a_1 a_2 ... a_{n_t-1}$, and possible words belonging to the left-linear languages that are subwords of $w_{sf} = a_{n_t+2} ... a_{n-1} a_n$. For each guess that matches, $\mathcal{A}$ checks whether the solution does not spoil the derivation tree, i.e., the concatenation of $a_1 a_2 ... a_{n_t-1}$, $a_{n_t} a_{n_t+1}$, and $a_{n_t+2} ... a_{n-1} a_n$ gives indeed a word belonging to the linear language generated by G_k. More precisely, we check whether the shuffle of the right-linear languages and the shuffle of the left-linear languages are well synchronized. Because there may exist an infinite number of places inside w_{pf} where the right-linear languages shuffle, and also an infinite number of places inside w_{sf} where the left-linear languages shuffle, $\mathcal{A}$ must guess an infinite number of integers that represents the length of each subword[4].

A procedure that guesses an infinite number of arbitrarily large integers, and checks the correctness of this guess, requires more than logarithmic time and space (even if the integers are stored in binary, see Algorithm 1). In order to overcome this impediment we define the *dependency graph* associated with a LCFG in linear-Dyck normal form. Using the description of a directed graph through regular expressions given in [6] we reduce the number of arbitrarily large integers that $\mathcal{A}$ has to guess to a finite number. This makes it possible the recognitions of linear languages by indexing ATMs in logarithmic time and space (Algorithms 2, 3 and 4).

Recall that an indexing ATM is an alternating Turing machine that is allowed to write any binary number on a special tape, called *index* tape. This number is interpreted as an address of a location on the input tape. Having an integer i written in binary on the index tape, $\mathcal{A}$ is able to read the input symbol placed on the i^{th} cell of the input tape. By using universal states to relate different branches on the computation, we can effectively read an input string of length n in $\mathcal{O}(\log\, n)$ time.

We recall that for each linear grammar in linear-Dyck normal form G_k, the derivation tree of a word $w \in L(G_k)$ has only "one bottom" reached in a bracketed pair from $N^{(1)}$. Therefore, the trace-word $t_{w,D}$ associated with w and the derivation D of w, contains only one nested segment of the form $[_j^t\ ]_j^t$, where $([_j^t,]_j^t) \in N^{(1)}$. The image through φ of $[_j^t\ ]_j^t$ is the core segment of w.

As explained above, $\mathcal{A}$ first guesses the position of the single bracketed pair $([_j^t,]_j^t) \in N^{(1)}$ in $t_{w,D}$. Depending on this position, and on the rules of G_k, $\mathcal{A}$ performs several guessing procedures in order to rebuild from the input word w, the structure of $t_{w,D}$, and thus the derivation tree of w. If no guessing procedure leads to a possible trace-word associated with the input string, then w is rejected as not belonging to $L(G_k)$.

In the sequel, by $D_{l,r}$ we denote the derivation of w that works as a rightmost derivation each time a bracketed pair $([_i,]_i) \in N_r^{(2)}$ occurs in a sentential form, i.e., first the rule that rewrites $]_i$ is applied, and afterwards the rule that rewrites $[_i$, and as a leftmost derivation for each bracketed pair $([_j,]_j) \in N_l^{(2)} \cup N^{(1)}$, i.e., first the rule that rewrites $[_j$ is applied, and afterwards the rule that rewrites

[4] Note that $\mathcal{A}$ does not have to guess also the position of each subword inside w. Knowing the length of each subword these positions can be computed.

$]_j$. The derivation tree of a word $w \in L(G_k)$ does not depend on the derivation order, thence a left-most derivation, or an arbitrary other derivation D of w, including $D_{l,r}$, have the same derivation tree, that read in the depth-first order gives a unique trace-word $t_{w,D}$.

We denote by $\overset{\triangleleft\,(2)}{N_r}$ and $\overset{\triangleright\,(2)}{N_r}$ the sets of all left-brackets $[_i$ and right-brackets $]_i$, respectively, such that $([_i,]_i) \in N_r^{(2)}$. The aim is to build the finite number of left-regular languages whose shuffle equals the language L_{sf}. This can be performed by using the rules of the grammar G_k. More precisely, for each $([^t_{j_p},]^t_{j_p}) \in N^{(1)}$, and $([_{j_q},]_{j_q})$, $([_{j_{q'}},]_{j_{q'}}) \in N_l^{(2)}$, j_q, $j_{q'}$ not necessarily different, we build, according to all possible derivations in G_k, four types of regular sets, i.e., sets of type $\mathcal{L}([_{j_q})$, $\mathcal{L}([^t_{j_p})$, $\mathcal{L}([_{j_q}[_{j_{q'}})$, and $\mathcal{L}([_{j_q}[^t_{j_p})$.

In the sets of type $\mathcal{L}([_{j_q})$ we collect all prefixes of all possible trace-words from Ł$_k$ that are generated until the left bracket $[_{j_q}$ occurs for the first time in a sentential form, such that before $[_{j_q}$ is derived, no left bracket from $N_l^{(2)}$ occurs within the derivation. All these prefixes are composed of only left-brackets belonging to $N_r^{(2)}$. In the sets of type $\mathcal{L}([^t_{j_p})$ we collect all prefixes of possible trace-words generated until the left bracket $[^t_{j_p}$ occurs in a sentential form, such that before $[_{j_q}$ is derived, no left bracket from $N_l^{(2)}$ occurs within the derivation.

In the sets $\mathcal{L}([_{j_q}[_{j_{q'}})$ we collect all possible subwords of trace-words from Ł$_k$, composed of only brackets belonging to $\overset{\triangleleft\,(2)}{N_r}$ generated between any two brackets $[_{j_q}$ and $[_{j_{q'}}$. Finally, the sets $\mathcal{L}([_{j_q}[^t_{j_p})$ gather all subwords of trace-words from Ł$_k$, composed of only brackets belonging to $\overset{\triangleleft\,(2)}{N_r}$ generated between any two left-brackets $[_{j_q}$ and $[^t_{j_p}$ during the generative process of $L(G_k)$.

$\mathcal{L}([_{j_q}) = \{x | S \Rightarrow^{(p)} x_1]_{j_q} y, [_{j_q} \rightarrow x_1 \in P, x \in (\overset{\triangleleft\,(2)}{N_r})^*, x$ is the prefix of $t_{w,D}$ generated until the step $p, p > 2$, of $D_{l,r}, y \in T^*\} \cup \{S | S \Rightarrow^{(2)} x_1]_{j_q}, [_{j_q} \rightarrow x_1 \in P\}$,

$\mathcal{L}([^t_{j_p}) = \{x | S \Rightarrow^{(2n-2)} x_1]^t_{j_p} y, [^t_{j_p} \rightarrow x_1,]^t_{j_p} \rightarrow x_2 \in P, x \in (\overset{\triangleleft\,(2)}{N_r})^*, x$ is a prefix of $t_{w,D}$ generated until the step $2n-2, n > 2$, of $t_{w,D}, y \in T^*\} \cup \{S | S \Rightarrow^{(2)} x_1]^t_{j_p}, [^t_{j_p} \rightarrow x_1 \in P\}$,

$\mathcal{L}([_{j_q}[_{j_{q'}}) = \{x | S \Rightarrow^{(p_1)} x_1 ... x_j]_{j_q} y, S \Rightarrow^{(p_2)} x_1 ... x_j x_{j+1}]_{j_{q'}} y', [_{j_q} \rightarrow x_j, [_{j_{q'}} \rightarrow x_{j+1} \in P, x \in (\overset{\triangleleft\,(2)}{N_r})^*, x$ is the subword of $t_{w,D}$ generated between the steps $p_1 + 1$ and $p_2 - 1$ of $D_{l,r}, y, y' \in T^*\} \cup \{\lambda | S \Rightarrow^{(p_1)} x_1 ... x_j]_{j_q} y, S \Rightarrow^{(p_1+2)} x_1 ... x_j x_{j+1}]_{j_{q'}} y', [_{j_q} \rightarrow x_j,]_{j_q} \rightarrow [_{j_{q'}}]_{j_{q'}}, [_{j_{q'}} \rightarrow x_{j+1} \in P, y, y' \in T^*\}$,

$\mathcal{L}([_{j_q}[^t_{j_p}) = \{x | S \Rightarrow^{(p)} x_1 ... x_{n_t-1}]_{j_q} y, S \Rightarrow^{(2n-2)} x_1 ... x_{n_t-1} x_{n_t}]^t_{j_p} y', [_{j_q} \rightarrow x_{n_t-1}, [^t_{j_p} \rightarrow x_{n_t},]^t_{j_p} \rightarrow x_{n_t+1} \in P, x \in (\overset{\triangleleft\,(2)}{N_r})^*, x$ is the subword of $t_{w,D}$ generated beween the steps $p+1$ and $2n-2$ of $D_{l,r}, y, y' \in T^*\} \cup \{\lambda | S \Rightarrow^{(2n-4)} x_1 ... x_{n_t-1}]_{j_q} y, S \Rightarrow^{(2n-2)} x_1 ... x_{n_t-1} x_{n_t}]^t_{j_p} y', [_{j_q} \rightarrow x_{n_t-1},]_{j_q} \rightarrow [_{j_{q'}}]_{j_{q'}}, [_{j_{q'}} \rightarrow x_{n_t} \in P, y, y' \in T^*\}$.

Let Ψ_r and ψ_r be the homomorphisms defined as $\Psi_r : (\overset{\triangleleft (2)}{N_r})^* \to (\overset{\triangleright (2)}{N_r})^*$, such that $\Psi_r([_i) =]_i$, and $\psi_r : (\overset{\triangleright (2)}{N_r})^* \to T^*$, $\psi_r(S) = \lambda$, $\psi_r(\lambda) = \lambda$, $\psi_r(]_i) = t$, for each rule of the form $]_i \to t$, $([_i,]_i) \in N_r^{(2)}$, $t \in T$. Now consider $\mathcal{L}^{{\Psi_r}^r}([_{j_q})$, $\mathcal{L}^{{\Psi_r}^r}([^t_{j_p})$, $\mathcal{L}^{{\Psi_r}^r}([_{j_q}[_{j_{q'}})$, and $\mathcal{L}^{{\Psi_r}^r}([_{j_q}[^t_{j_p})$ the mirror image[5] of the above sets, in which each left bracket $[_i$, such that $([_i,]_i) \in N_r^{(2)}$, is switched into its pairwise $]_i$, i.e., Ψ_r is applied first. Let $\mathcal{L}^{\psi_r}([_{j_q})$, $\mathcal{L}^{\psi_r}([^t_{j_p})$, $\mathcal{L}^{\psi_r}([_{j_q}[_{j_{q'}})$, and $\mathcal{L}^{\psi_r}([_{j_q}[^t_{j_p})$ be the image of the above $'\mathcal{L}^{{\Psi_r}^r}{}'$ sets through ψ_r. Since regular languages are closed under reverse and homomorphism, $\mathcal{L}^{{\Psi_r}^r}([_{j_q})$, $\mathcal{L}^{{\Psi_r}^r}([^t_{j_p})$, $\mathcal{L}^{{\Psi_r}^r}([_{j_q}[_{j_{q'}})$, $\mathcal{L}^{{\Psi_r}^r}([_{j_q}[^t_{j_p})$, $\mathcal{L}^{\psi_r}([_{j_q})$, $\mathcal{L}^{\psi_r}([^t_{j_p})$, $\mathcal{L}^{\psi_r}([_{j_q}[_{j_{q'}})$, and $\mathcal{L}^{\psi_r}([_{j_q}[^t_{j_p})$ are regular.

With the above regular sets already built, we have all prerequisites needed to explain the way in which a "primary" indexed ATM can "guess" linear languages (see Algorithm 1). This is not yet a logarithmic solution of recognition of linear languages, but supply an idea of how an ATM should be used and where the strategy of using it must be improved in order to reach our "logarithmic" aims. In parallel with the description of this very first algorithm we provide an explicit labeling procedure of the computation tree associated with an ATM, with respect to [1] and [3]. Let $\mathcal{A}$ be an indexing ATM composed of an input tape that stores an input word, $w \in T^*$ of length n, an index tape to guess input symbols, and one work tape divided into four tracks, to record the positions of the input symbols and several other values used during the computation. These numbers are stored on the tracks of the work tape in binary. At the beginning of the computation the tracks of the work tape are empty.

Algorithm 1 (*The Primary Algorithm*) Let $w \in T^*$, $w = a_1a_2...a_n$, be an input word of length n

Level 1 (*Existential*) In an existential state $\mathcal{A}$ guesses the length of w and verifies the correctness of this guess, i.e., writes on the index tape n, and checks whether the n^{th} cell of the input tape contains a terminal symbol and the cell $(n+1)$ contains no symbol. The correct value of n is recorded in binary on the first track of the work tape. *This procedure requires* $\mathcal{O}(\log n)$ *time and space.* $\triangle$

Level 2 (*Existential*) Using existential states $\mathcal{A}$ branches all i between 1 and n, and tries to localize the position of $([^t_j,]^t_j) \in N^{(1)}$ inside w, such that $[^t_j \to a_i$, and $]^t_j \to a_{i+1} \in P$. Denote by n_t the core index i for which a_i, and a_{i+1},

[5] If $s = s_1s_2...s_n$ is an arbitrary string then the mirror image of s, denoted as s^r, is the reverse of s, i.e., $s^r = s_n...s_2s_1$. The mirror image of the above sets can be also built directly during the derivation process in the same way we built the sets $\mathcal{L}([_{j_q})$, $\mathcal{L}([^t_{j_p})$, $\mathcal{L}([_{j_q}[_{j_{q'}})$, $\mathcal{L}([_{j_q}[^t_{j_p})$. This can be done, if instead of collecting brackets from $\overset{\triangleleft (2)}{N_r}$ we collect brackets from $\overset{\triangleright (2)}{N_r}$. In the right-side of the nested segment $[^t_j$ $]^t_j$ in $t_{w,D}$, with $\varphi([^t_j\ ]^t_j) = a_{n_t}a_{n_t+1}$, the right-brackets are generated in the reverse order they occur in the left side of $[^t_j\ ]^t_j$. We chose to collect left-brackets from $N_r^{(2)}$ in order to make the connection between brackets from the trace-word and brackets from $\overset{\triangleright (2)}{N_r}$ occurring in the dependency graph, further defined (Definition 8).

$1 \leq i \leq n-1$, have been guessed with this property. Once the core segment $a_{n_t}a_{n_t+1}$, $1 \leq n_t \leq n-1$, has been localized, the computation continues on that existential branch with Level 3. At this level there may exist several existential branches through which the computation can (correctly) continue. All these branches are labeled by 1. If no two symbols a_{n_t}, a_{n_t+1}, with the above property can be found, the input w is rejected. For each existential branch, holding on the core segment $a_{n_t}a_{n_t+1}$, the value n_t is stored in binary on the second track of the work tape. Then $\mathcal{A}$ computes $n-n_t-1$ and $\log(n-n_t-1)$ space is allocated for the third track. This allocation stays as a space-measure to control the limits of computation, i.e., no positive integer greater than $n-n_t-1$, written in binary, can be stored on this track. The out-degree of a node at this level is at most n. The operation to convert the computation tree into a tree with out-degree 2 increases the height of the computation tree with log n. Therefore, *the cost of the computation at this level takes* $\mathcal{O}(\log n)$ *time and space*, too. $\triangle$

Level 3 Depending on the position of $a_{n_t}a_{n_t+1}$ in w we may have:

1) (*Universal* case) If $n_t = 1$, then the trace-word $t_{w,D}$, has the structure[6]:

$$t_{w,D} = \begin{array}{cccccccccccc} [_{i_{n-1}} & [_{i_{n-2}} & \cdots & [_{i_2} & [^t_{j_1} & &]^t_{j_1} & &]_{i_2} & \cdots &]_{i_{n-2}} &]_{i_{n-1}} \\ | & | & & | & | & & | & & | & & | & | \\ \lambda & \lambda & \ldots & \lambda & a_1 = a_{n_t} & a_2 = a_{n_t+1} & & a_3 \ldots & a_{n-1} & & a_n & \end{array}$$

Consider $w_{n_t} = a_3a_4...a_{n-1}a_n$. In this case $\mathcal{A}$ checks whether the axiom S or w_{n_t} belongs to the language $\mathcal{L}^{\psi_r}([^t_{j_1})$. Because $\mathcal{L}^{\psi_r}([^t_{j_1})$ is a regular language, the membership problem is decidable in $\mathcal{O}(\log n)$ time (in terms of the length of the input string) by an ATM. If neither the axiom S nor w_{n_t} belongs to $\mathcal{L}^{\psi_r}([^t_{j_1})$, then w is rejected on this branch of the computation tree.

2) (*Universal* case) If $n_t = n-1$, then the trace-word $t_{w,D}$, is of the form:

$$t_{w,D} = \begin{array}{cccccccccccc} [_{j_1} &]_{j_1} & [_{j_2} &]_{j_2} & \cdots & [_{j_{n-2}} &]_{j_{n-2}} & & [^t_{j_{n-1}} & &]^t_{j_{n-1}} & \\ | & | & | & | & & | & | & & | & & | & \\ a_1 & \lambda & a_2 & \lambda & \ldots a_{n-2} & & \lambda & a_{n-1} = a_{n_t} & & a_n = a_{n_t+1} & & \end{array}$$

Then $\mathcal{A}$ universally branches all i between 1 and $n-1$, and in parallel checks whether the following conditions hold: 2.*i*) there exists $[_{j_1} \rightarrow a_1 \in P$ such that $S \in \mathcal{L}([_{j_1})$, i.e., $S \rightarrow [_{j_1}]_{j_1} \in P$; 2.*ii*) for each i, $1 \leq i \leq n-2$, whether there exist $[_{j_i} \rightarrow a_i \in P$, $[_{j_{i+1}} \rightarrow a_{i+1} \in P$, and $\lambda \in \mathcal{L}([_{j_i}[_{j_{i+1}})$; 2.*iii*) whether there exist $[_{j_{n-2}} \rightarrow a_{n-2} \in P$, $[^t_{j_{n-1}} \rightarrow a_{n-1} \in P$, and $\lambda \in \mathcal{L}([_{j_{n-2}}[^t_{j_{n-1}})$.

If for at least one i, $1 \leq i \leq n-1$, the above conditions do not hold w is rejected (there exists at least one branch in an universal fork labeled by 0).

3) (*Alternating from Universal to Existential*) If $1 < n_t < n-1$ then $t_{w,D}$ is a matched segment of the form listed below, in which sets of left-brackets $[_{i_1}...[_{i_{k_1}}$, $[_{i_{k_{j-1}+1}}...[_{i_{k_j}}$, $2 \leq j \leq n_t$, may be missing.

[6] By vertical lines we emphasize the image through the homomorphism φ of each bracket occurring in $t_{w,D}$.

$$t_{w,D} = \begin{array}{cccccccccccccccc} [_{i_1} & ... & [_{i_{k_1}} & [_{j_1} &]_{j_1} & [_{i_{k_1+1}} & ... & [_{i_{k_2}} & [_{j_2} &]_{j_2} & ... & [_{j_{n_t-1}} &]_{j_{n_t-1}} & [_{i_{k_{n_t-1}+1}} & ... \\ | & ... & | & | & | & | & ... & | & | & | & ... & | & | & | & ... \\ \lambda & ... & \lambda & a_1 & \lambda & \lambda & ... & \lambda & a_2 & \lambda & ... & a_{n_t-1} & \lambda & \lambda & ... \end{array}$$

$$\begin{array}{cccccccccccccc} ... & [_{i_{k_{n_t}}} & [^t_{j_{n_t}} &]^t_{j_{n_t}} &]_{i_{k_{n_t}}} & ... &]_{i_{k_{n_t-1}}} & ... &]_{i_{k_2}} & ... &]_{i_{k_1}} & ... &]_{i_1} \\ ... & | & | & | & | & ... & | & ... & | & ... & | & ... & | \\ ... & \lambda & a_{n_t} & a_{n_t+1} & a_{n_t+2} & . & a_{n-k_{n_t-1}...-k_1+1} & & a_{n-k_2-k_1+1} & ... & a_{n-k_1+1} & ... & a_n \end{array}$$

In this case, $\mathcal{A}$ existentially guesses n_t natural numbers k_1, k_2, ..., k_{n_t-1}, k_{n_t}, $0 \leq k_i \leq n - n_t - 1$, $1 \leq i \leq n_t$, such that $k_1 + k_2 + ... + k_{n_t-1} + k_{n_t} = n - n_t - 1$, and for each such an n_t-tuple universally branch on all $[_{j_i}$ left-brackets, and $[^t_{j_{n_t}}$, $([_{j_i},]_{j_i}) \in N_l^{(2)}$, $([^t_{j_{n_t}},]^t_{j_{n_t}}) \in N^{(1)}$, $1 \leq i \leq n_t - 1$, and let $\mathcal{A}$ to proceed further as follows:

1) verify whether there exists a rule $[_{j_1} \to a_1 \in P$, and for each such a rule check whether the axiom S or the suffix of w, $w_{n-k_1+1,n} = a_{n-k_1+1}...a_{n-1}a_n$ belongs to the regular set $\mathcal{L}^{\psi_r}([_{j_1})$;

2) for $1 \leq i \leq n_t - 1$, check whether there exist two rules $[_{j_i} \to a_i \in P$, and $[_{j_{i+1}} \to a_{i+1} \in P$, and verify whether λ, or the substring $w_{n-k_{s_{i+1}}+1, n-k_{s_i}} = a_{n-k_{s_{i+1}}+1}...a_{n-k_{s_i}}$ of w, belongs to the set $\mathcal{L}^{\psi_r}([_{j_i}[_{j_{i+1}})$, where $k_{s_{i+1}} = k_1 + k_2 + ... + k_{i+1}$ and $k_{s_i} = k_1 + k_2 + ... + k_i$.

3) for $i = n_t$, check whether there exist two rules $[_{j_{n_t-1}} \to a_{n_t-1} \in P$ and $[^t_{j_{n_t}} \to a_{n_t} \in P$, and verify whether λ, or the suffix $w_{n-k_{s_{n_t}}+1, n-k_{s_{n_t-1}}} = a_{n-k_{s_{n_t}}+1}...a_{n-k_{s_{n_t-1}}} = a_{n_t+2}...a_{n_t+k_{n_t}+1}$ of w, belongs to the set $\mathcal{L}^{\psi_r}([_{j_{n_t-1}}[^t_{j_{n_t}})$, where $k_{s_{n_t}} = k_1 + k_2 + ... + k_{n_t}$ and $k_{s_{n_t-1}} = k_1 + k_2 + ... + k_{n_t-1}$.

Note that for each step described above, $\mathcal{A}$ first searches existentially for rules in the grammar G_k that "pump" in w terminal symbols. If there is no solution for this search, i.e., all branches in this existential fork are labeled by 0, w is rejected. Otherwise, having the n_t-tuple stored in the fourth track of the ATM work tape, $\mathcal{A}$ universally checks, for each of the 1) 2) and 3) steps, whether the guessed n_t-tuple is a correct solution. Each guessed n_t-tuple is taken by an universal branch. Now it is easy to observe that the above method goes beyond $\mathcal{O}(\log n)$ time and space, because the number of all possible (existential) guesses for all n_t-tuples with $n_t = \mathcal{O}(n)$, arises to $\mathcal{O}(n^n)$. The operation to convert the tree into a tree with (finite) out-degree 2 will increase the height, and thus the alternating time, of the computation tree with $\mathcal{O}(n\log n)$. The space required to record the n_t-tuple is also $\mathcal{O}(n\log n)$. Therefore, this strategy is not suitable for our approaches. However, this "primary" algorithm provides an idea of how an indexing ATM, guessing linear languages, works. △

◇

In the sequel we focus on finding a suitable strategy for an indexing ATM, such that based on the method explained in Algorithm 1, we can reduce the computational resources of $\mathcal{A}$ to $\mathcal{O}(\log n)$ time and space. To accomplish this aim, we introduce a structural framework that consists in a graphical representation of a grammar in linear-Dyck normal form through a directed graph called the *dependency graph* of the grammar.

By making use of this graphical representation of a grammar in linear-Dyck normal form the dimension of the tuples, of arbitrarily large positive integers, that $\mathcal{A}$ has to guess can be substantially reduced to a finite number. This, substantially decreases the number of guessings performed by $\mathcal{A}$ from $\mathcal{O}(n^n)$ to $\mathcal{O}(n^c)$, where c is a constant. In this way the height of the computation tree associated with $\mathcal{A}$ becomes comparable with $\mathcal{O}(c{\cdot}\log\, n)$.

Definition 8. Let $G_k = (N_k, T, P_k, S)$ be an arbitrary LCFG in linear-Dyck normal form. The *dependency graph* of G_k, denoted as $\mathcal{G}$, is a directed graph in which vertices are labeled with variables from N_k, and for each rule $X \rightarrow [_i\ ]_i \in P$, $([_i,\ ]_i) \in N_l^{(2)}$, $\mathcal{G}$ contains a directed edge from X to $]_i$, and for each rule $X \rightarrow [_i\ ]_i \in P$, $([_i,\ ]_i) \in N_r^{(2)}$, $\mathcal{G}$ contains a directed edge from X to $[_i$. If $([_i^t,\ ]_i^t) \in N^{(1)}$ and $X \rightarrow [_i^t\ ]_i^t \in P$, then $\mathcal{G}$ contains a directed edge from X to $[_i^t$, and $[_i^t$ is called a *final vertex*. The vertex labeled by the axiom is called the *initial vertex*.

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary LCFG in linear-Dyck normal form, and let $\mathcal{G}$ be the dependency graph associated with G_k. Next, we build the set of all possible paths in $\mathcal{G}$ starting from the initial vertex to a final vertex. We call such a path a *terminal path*. We refer to an edge or a path from v to v as a (*cycle*) *loop*. We are interested only on those grammars for which $\mathcal{G}$ contains at least one cycle. Otherwise, the language generated by G_k is finite.

The *cycle rank* of a graph is a measure of the loop complexity[7] formally defined and studied in [4], [6], and [9]. In [6] it is proved that from each two vertices m and n belonging to a digraph of cycle rank k, there exists a regular expression of star-height k that describes the set of paths from m to n. According to this result the set of terminal paths in the dependency graph $\mathcal{G}$ can be characterized by regular expressions of finite star-height.

If $v_1v_2...v_kv_1$ is a loop (cycle), where v_i, $1 \leq i \leq k$, are vertices in $\mathcal{G}$, then arbitrarily many repetitions of this loop in $\mathcal{G}$ is denoted as $(v_1v_2...v_k)^{\diamond}$, where $\diamond$ may be $*$ or $+$ Kleene closures. A loop within a loop (or a loop of diamond-height two) is encoded as $(v_{j_1}...v_{j_p}(v_{i_1}...v_{i_k})^{\diamond}v_{j_{p+1}}...v_{j_r})^{\diamond}$, where $v_{i_a} \neq v_{i_b}$, $v_{j_c} \neq v_{j_d}$, $a \neq b$, $c \neq d$. An arbitrary loop of diamond-height m is called in this paper *multilevel loop*. Since N_k is finite, i.e., the dependency graph is composed of a finite number of vertices, there will be only a finite number of distinct loops in a loop. Also, each terminal path will be composed of a finite number of distinct loops, otherwise a multilevel loop is created. The "height" of the nested loops can be at most equal with the cycle rank of the underlying dependency graph. With these considerations we can divide the infinite set of terminal paths from the initial vertex to a final vertex in $\mathcal{G}$ into a finite number of classes of terminal paths. All paths belonging to the same class are characterized by the same regular expression in terms of $*$ and $+$. Hence, there exists a finite number of regular expressions based on concatenation, $*$ and $+$ Kleene operations, that characterize

[7] It is trivial to prove, by induction on the number of nodes, that the cycle rank of a digraph, with a finite number of nodes, is finite.

all classes of terminal paths in $\mathcal{G}$. It is easy to observe that each regular expression of a finite star-height can be described as a finite union of regular expressions in terms of only + Kleene closure[8]. Therefore, by enlarging the number of regular expressions readable from the dependency graph associated with a LCFG in linear-Dyck normal form, without loss of generality we can assume to work only with regular expressions in terms of +. Next we give an example that deals with the dependency graph of a linear grammar and with regular expressions, in terms of + Kleene closure, readable from this dependency graph.

Example 2 Let $G_4 = (N_4, T, P_4, S)$ be a LCFG in linear-Dyck normal form such that $P_4 = \{S \rightarrow [_1\,]_1, [_1 \rightarrow [_2\,]_2, [_1 \rightarrow [_3\,]_3,]_2 \rightarrow [_1\,]_1,]_3 \rightarrow [_2\,]_2,]_3 \rightarrow [_4^t\,]_4^t,]_1 \rightarrow a, [_2 \rightarrow b, [_3 \rightarrow c, [_4^t \rightarrow d,]_4^t \rightarrow d\}$.

The dependency graph of G_4 is the directed graph composed of the set $V = \{S,\ [_1,\ ]_2,\ ]_3,\ [_4^t\}$ of vertices, and the set $E = \{S\ [_1, [_1\ ]_2, [_1\ ]_3,]_2\ [_1,]_3\ ]_2,]_3\ [_4^t\}$ of edges. The initial vertex is S and the final vertex is $[_4^t$. For this grammar $\mathcal{G}$ is characterized by a single regular expression, i.e., $\Re = S[_1((]_2\ [_1)^*(]_3(]_2\ [_1)^+)^*)^*]_3[_4^t$. More than one class of terminal paths, in terms of $*$, would exist for instance, if $\mathcal{G}$ would be composed of two or more disjoint graphs connected with each other only through the initial vertex S. Furthermore, $\Re$ can be expressed as a finite union of regular expressions in terms of +, as follows $\Re = \Re_1 \cup \Re_2 \cup \Re_3 \cup \Re_4$, where $\Re_1 = S[_1]_3[_4^t$, $\Re_2 = S[_1((]_2\ [_1)^+)^+]_3[_4^t$, $\Re_3 = S[_1((]_3(]_2\ [_1)^+)^+)^+]_3[_4^t$, $\Re_4 = S[_1((]_2\ [_1)^+(]_3(]_2\ [_1)^+)^+)^+]_3[_4^t$ ♣

The plus-height, i.e., the number of nested + occurring in the description of a regular expression, can be defined analogous to the star-height of a regular expression in [11].

Definition 9. Let Σ be a finite alphabet. The plus-height $h(\Re)$ of a regular expression $\Re$ is defined recursively as follows:

1. $h(\lambda) = h(\emptyset) = h(a) = 0$ for $a \in \Sigma$;
2. $h(\Re_1 \cup \Re_2) = h(\Re_1\Re_2) = \max\{h(\Re_1), h(\Re_2)\}$, and $h(\Re^+) = h(\Re) + 1$.

From [11] we have also adopted the next definition of a regular expression in string form, in terms of + Kleene closure.

Definition 10. The class $SREG^+(\Sigma)$ of regular expressions in string form over a finite alphabet Σ, in terms of plus-height, is defined recursively as follows. For any regular expression $\Re$,

1. if $h(\Re) = 0$, then $\Re \in SREG^+(\Sigma)$ iff $\Re = w_1 \cup ... \cup w_m$ for some $m \geq 1$, and $w_i \in \Sigma^*$, $1 \leq i \leq m$;
2. if $h(\Re) > 0$, then $\Re \in SREG^+(\Sigma)$ iff $\Re = \Re_1 \cup ... \cup \Re_m$ for some $m \geq 1$, $\Re_i \in SREG^+(\Sigma)$, $1 \leq i \leq m$, and there exists at least one j, $1 \leq j \leq m$, such that $\Re_j$ is of the form $\Re_j = w_{j_1}(\Re_{j_1})^+ w_{j_2}(\Re_{j_2})^+ ... w_{j_{p_j}}(\Re_{j_{p_j}})^+ w_{j_{p_j+1}}$, $p_j \geq 1$, $w_{j_l} \in \Sigma^*$, $1 \leq l \leq p_j + 1$, and $\Re_{j_k} \in SREG^+(\Sigma)$, $1 \leq k \leq p_j$.

[8] The union operator, that may occur inside a $*$ loop, can be eliminated by using the relation $(r_1 \cup r_2)^* = ((r_1)^*(r_2)^*)^*$, where r_1 and r_2 are regular expressions.

Since each regular expression based on $*$ Kleene operation can be expressed as a finite union of regular expressions in terms of $+$ Kleene operation, without loss of generality, we assume that we can divide the set of terminal paths in $\mathcal{G}$ into a finite number of classes described only by $+$ Kleene operation.

Recall that $\overset{\triangleleft\,(2)}{N_r}$ is the set of all left-brackets $[_i$, $([_i$, $]_i) \in N_r^{(2)}$, and $\overset{\triangleright\,(2)}{N_r}$ is the set of all right-brackets $]_i$, such that $([_i$, $]_i) \in N_r^{(2)}$. Analogously, we define $\overset{\triangleleft\,(2)}{N_l}$ the set of all left-brackets $[_j$, $([_j$, $]_j) \in N_l^{(2)}$, and $\overset{\triangleright\,(2)}{N_l}$ the set of all right-brackets $]_j$, such that $([_j$, $]_j) \in N_l^{(2)}$. Our aim is to describe terminal paths in $\mathcal{G}$, in terms of plus-height loops induced only by elements from $\overset{\triangleright\,(2)}{N_l}$, i.e., nested plus-loops that begin with a bracket[9] from $\overset{\triangleright\,(2)}{N_l}$[10]. This is always possible by "rotating" a loop until it starts with a bracket from $\overset{\triangleright\,(2)}{N_l}$. This operation does not change the regular language associated with the regular expression. It only enlarges the length of the regular expression, as in Example 3.

Example 3 Let $\Re = S([_5[_3[_4(]_1(]_2[_4)^+[_3[_4)^+)^+t$ be a regular expression over the set of brackets $\overset{\triangleright\,(2)}{N_l} = \{]_1,]_2\}$, $\overset{\triangleleft\,(2)}{N_r} = \{[_3, [_4, [_5\}$.

In terms of loops induced by elements from $\overset{\triangleright\,(2)}{N_l}$, $\Re$ can be rewritten as $\Re = S[_5[_3[_4\big((]_1(]_2[_4)^+[_3[_4)^+[_5[_3[_4\big)^*(]_1(]_2[_4)^+[_3[_4)^+t$. In terms of loops induced by elements from $\overset{\triangleright\,(2)}{N_l}$ and $+$ Kleene operation $\Re$ becomes $\Re = \Re_0^* \cup \Re_0^+$, $\Re_0^* = S[_5[_3[_4(]_1(]_2[_4)^+[_3[_4)^+t$, $\Re_0^+ = S[_5[_3[_4\big((]_1(]_2[_4)^+[_3[_4)^+[_5[_3[_4\big)^+(]_1(]_2[_4)^+[_3[_4)^+t$. ♣

Let $\mathcal{R}_\mathcal{G}$ be the set of all regular expressions, in terms of $+$ Kleene operation, readable from $\mathcal{G}$. Let $\Re$ be an arbitrary element of $\mathcal{R}_\mathcal{G}$, and $\wp$ an arbitrary path belonging to the class of terminal paths characterized by $\Re$. Comparing the regular structure of $\Re$ with the cyclicity structure of $\wp$, we observe that both $\Re$ and $\wp$ are composed of the same three main types of segments:

1. segments that do not contain any bracket from $\overset{\triangleright\,(2)}{N_l}$ (but only left-brackets from $\overset{\triangleleft\,(2)}{N_r}$),
2. "constant" segments that contain a finite number of distinct brackets from $\overset{\triangleright\,(2)}{N_l}$ (and between them an arbitrary large number of left-brackets from $\overset{\triangleleft\,(2)}{N_r}$),
3. and one-level or multilevel loops (nested loops of different plus-heights) induced by brackets from $\overset{\triangleright\,(2)}{N_l}$.

Depending on the type of the above segments composing $\Re$ and $\wp$, we continue to build, using the concatenation and $+$ Kleene operations, several other more complex regular sets similar to the sets $'\mathcal{L}^{\psi_r}{}'$.

---

[9] We are not yet interested in the loops induced in $\mathcal{G}$ by elements from $\overset{\triangleleft\,(2)}{N_r}$.

[10] A similar procedure can be outlined if we consider terminal paths in terms of loops induced only by elements from $\overset{\triangleleft\,(2)}{N_r}$. In this case each routine should be symmetrically changed in terms of brackets belonging to the sets $N_r^{(2)}$ and $N_l^{(2)}$.

The first case considered above has been discussed when we built the sets of type $'\mathcal{L}^{\psi_r}'$. Case 1, considered below, deals with "constant" segments that do not contain any loop induced only by elements from $\overset{\triangleright\,(2)}{N_l}$. Loops are investigated according to their complexity, i.e., the plus-height of the loop, see Case 2. Each terminal path belonging to a certain class, characterized by a certain regular expression, will be a finite combination of "empty" segments in terms of brackets from $\overset{\triangleright\,(2)}{N_l}$, "constant" segments and loops of "constant" length and different plus-heights, induced by brackets from $\overset{\triangleright\,(2)}{N_l}$.

Before dealing with the above segments and loops readable from $\mathcal{G}$, recall that Ψ_r and ψ_r are homomorphisms defined as $\Psi_r : (\overset{\triangleleft\,(2)}{N_r})^* \rightarrow (\overset{\triangleright\,(2)}{N_r})^*$, $\Psi_r(]_i) =]_i$, and $\psi_r : (\overset{\triangleright\,(2)}{N_r})^* \rightarrow T^*$, $\psi_r(S) = \lambda$, $\psi_r(]_i) = t$, for each rule of the form $]_i \rightarrow t$.

We further define Ψ_l and ψ_l as $\Psi_l : (\overset{\triangleright\,(2)}{N_l})^* \rightarrow (\overset{\triangleleft\,(2)}{N_l})^*$, with $\Psi_l([_j) = [_j$, and $\psi_l : (\overset{\triangleleft\,(2)}{N_l})^* \rightarrow T^*$, $\psi_l(S) = \lambda$, $\psi_l([_j) = t$, for each rule of the form $[_j \rightarrow t$.

Case 1 Let $]_{c_1}X_1]_{c_2}X_2...X_{s-1}]_{c_s}$ be an arbitrary "constant" segment occurring in $\mathcal{G}$, composed of a constant number of brackets from $\overset{\triangleright\,(2)}{N_l}$, such that each segment X_i, $1 \leq i \leq s$, $s \geq 2$, is composed of an arbitrary number of brackets from $\overset{\triangleleft\,(2)}{N_r}$. For each such segment we build a pair of sets[11] in which the first element of the couple is $[_{c_1}[_{c_2}...[_{c_s}$, and the second element is composed of all words obtained by concatenating words belonging to previously built sets of type $\mathcal{L}([_{j_q}[_{j_{q'}})$. Formally, we have $\mathcal{L}^c([_{c_1}...[_{c_s}) = \{([_{c_1}...[_{c_s}, \ell_1\ell_2...\ell_{s-1}) | \ell_i \in \mathcal{L}([_{c_i}[_{c_{i+1}}), 1 \leq i \leq s-1, s \geq 2\}$.

If $(x_1,\ x_2)$ is a pair of two strings, and X is an arbitrary set of such pairs, then we denote by X_l the set composed of the first elements of pairs, and by X_r the set composed of the second elements of pairs, i.e., if $X = (X_l,\ X_r)$, then $X_l = \{x_1 | (x_1, x_2) \in X\}$, and $X_r = \{x_2 | (x_1, x_2) \in X\}$.

Further on, we compute $\mathcal{L}^{c^{\psi_r}}([_{c_1}...[_{c_s}) = \{(x, y) | x = \psi_l([_{c_1}[_{c_2}...[_{c_s}), y = \psi_r((\Psi_r(\ell))^r), \ell \in \mathcal{L}^c_r([_{c_1}...[_{c_s})\}$. $\surd$

Case 2 In order to study multilevel loops we first investigate the simplest loops determined by brackets from $\overset{\triangleright\,(2)}{N_l}$, i.e., loops of plus-height one and two, and make a generalization afterwards. In what follows, specific structures, especially indexes, occurring inside languages generated by brackets from $\overset{\triangleright\,(2)}{N_l}$, are marked by symbols $'\check{}'$. Specific structures, characterizing languages generated by brackets from $\overset{\triangleleft\,(2)}{N_r}$, are marked by symbols $'\hat{}'$. Whenever it is known, the plus-height of a regular expression is denoted as $+_i$, meaning that the plus-height of that expression is i. In general, instead of "plus-height" we use the notation $+_\hbar$. Associated with each $+_\hbar$ there is an arbitrarily large positive integer, called the *value* of $+_\hbar$, i.e., the number of times that loop is repeated. Upper indexes of integer values are always enclosed into parenthesis.

[11] To avoid any ambiguity, that may occur in the guessing and searching procedures of $\mathcal{A}$, by $[_{c_1}[_{c_2}...[_{c_s}$ occurring in the further notation $\mathcal{L}([_{c_1}...[_{c_s})$ we denote the finite enumeration of all brackets that occur between $[_{c_1}$ and $[_{c_s}$.

Case 2. 1 (Loops of plus-height one induced by brackets from $\overset{\triangleright^{(2)}}{N_l}$) The simplest loop of plus-height one is

$$(\rfloor_{j_1^{(1)}} X_{i_1^{(1)}} \rfloor_{j_2^{(1)}} X_{i_2^{(1)}} ... X_{i_{\check{q}^{(1)}-1}^{(1)}} \rfloor_{j_{\check{q}^{(1)}}^{(1)}} X_{i_{\check{q}^{(1)}}^{(1)}})^{+} \tag{1}$$

where $\rfloor_{j_p^{(1)}} \in \overset{\triangleright^{(2)}}{N_l}$, and $X_{i_p^{(1)}}$ is composed of brackets from $\overset{\triangleleft^{(2)}}{N_r}$, more precisely, $X_{i_p^{(1)}} \in \mathcal{L}(\lfloor_{j_p^{(1)}} \lfloor_{j_{p+1}^{(1)}})$, $1 \leq p \leq \check{q}^{(1)} - 1$, $X_{i_{\check{q}^{(1)}}^{(1)}} \in \mathcal{L}(\lfloor_{j_{\check{q}^{(1)}}^{(1)}} \lfloor_{j_1^{(1)}})$.

Note that for each loop of plus-height greater than one, the concatenation of brackets from $\overset{\triangleright^{(2)}}{N_l}$ (i.e., the segment $\rfloor_{j_1^{(1)}} \rfloor_{j_2^{(1)}} ... \rfloor_{j_{\check{q}^{(1)}}^{(1)}}$ for the case of loop (1)) has to be well synchronized with the segment composed of brackets from $\overset{\triangleleft^{(2)}}{N_r}$ (i.e., the segment $X_{i_1^{(1)}} X_{i_2^{(1)}} ... X_{i_{\check{q}^{(1)}}^{(1)}}$ for the loop (1)). In less words, if $\rfloor_{j_1^{(1)}} \rfloor_{j_2^{(1)}} ... \rfloor_{j_{\check{q}^{(1)}}^{(1)}}$ is repeated k times in (1) then so is $X_{i_1^{(1)}} X_{i_2^{(1)}} ... X_{i_{\check{q}^{(1)}}^{(1)}}$.

Therefore, from now on instead of building "single" regular languages based on the concatenation operation, we build "couples" of regular languages based on concatenation and "synchronized" + Kleene operation.

For the loop (1) we build the set of all couples in which the first element of the couple is the concatenation of sequences of brackets from $\overset{\triangleright^{(2)}}{N_l}$, and the second element of the couple is a concatenation of all brackets from $\overset{\triangleleft^{(2)}}{N_r}$, generated in $\mathcal{G}$ between any two consecutive right-brackets from $\overset{\triangleright^{(2)}}{N_l}$ occurring in (1). Formally, using the concatenation, homomorphism, and the reverse operations we compute

$$\mathcal{L}^{c_\psi^{(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}) = \{(x, y) | x = \psi_l(\Psi_l(\rfloor_{j_1^{(1)}} ... \rfloor_{j_{\check{q}^{(1)}}^{(1)}})), y = \psi_r((\Psi_r(\ell_{i_1^{(1)}} ... \ell_{i_{\check{q}^{(1)}}^{(1)}}))^r),$$
$$\ell_{i_p^{(1)}} \in \mathcal{L}(\lfloor_{j_p^{(1)}} \lfloor_{j_{p+1}^{(1)}}), 1 \leq p \leq \check{q}^{(1)} - 1, \ell_{i_{\check{q}^{(1)}}^{(1)}} \in \mathcal{L}(\lfloor_{j_{\check{q}^{(1)}}^{(1)}} \lfloor_{j_1^{(1)}})\}.$$

Considering that the loop (1) is performed $\check{k}^{(1)}$ times, we go further by building the + Kleene closure of this loop, i.e.,

$$\mathcal{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}) = \{(x^{\check{k}^{(1)}}, y^{\check{k}^{(1)}}) | x \in {\mathcal{L}_l}^{c_\psi^{(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}), y \in {\mathcal{L}_r}^{c_\psi^{(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}),$$
$$\check{k}^{(1)} \geq 1\}.$$

The set $\mathcal{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ behaves rather as a context-free language than a regular language[12]. However, separately considered the sets ${\mathcal{L}_l}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}})$

[12] If for each pair from $\mathcal{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ the first element is concatenated with the second element, i.e., we compute the language $\bar{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}) = \{x^{\check{k}^{(1)}} y^{\check{k}^{(1)}} | x \in {\mathcal{L}_l}^{c_\psi^{(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}), y \in {\mathcal{L}_r}^{c_\psi^{(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}}), \check{k}^{(1)} \geq 1\}$, then $\bar{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}} ... \lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ is not regular.

and ${\mathcal{L}_r}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}}...\lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ are regular (because regular languages are closed under concatenation, homomorphism, reverse, and plus Kleene operations, and there is no "synchronization" between them). Since $\mathcal{L}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}}...\lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ is not regular, $\mathcal{A}$ has only to guess $\check{k}^{(1)}$ that characterizes the set ${\mathcal{L}_l}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}}...\lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ and afterwords to check the synchronization. In order to do this we need also the structure of the loops occurring in $\mathcal{G}$ between each two consecutive brackets $\lfloor_{j_{\check{q}^{(1)}}^{(1)}}$, $\lfloor_{j_p^{(1)}}$ and $\lfloor_{j_p^{(1)}}$, $\lfloor_{j_{p+1}^{(1)}}$, $1 \leq p \leq \check{q}^{(1)} - 1$. More precisely, we need the entire structure of the loop provided in (1). Note that, when we build the sets ${\mathcal{L}_r}^{c_\psi^{+(1)}}(\lfloor_{j_1^{(1)}}...\lfloor_{j_{\check{q}^{(1)}}^{(1)}})$ we concatenate all segments $X_{i_p^{(1)}}$, $1 \leq p \leq \check{q}^{(1)}$. Therefore, we are interested in the structure of the whole segment $X^{(1)} = X_{i_1^{(1)}}...X_{i_{\check{q}}^{(1)}}$ and not in the structure of each particular segment $X_{i_p^{(1)}}$, $1 \leq p \leq \check{q}^{(1)}$. At its turn $X^{(1)}$ can be expressed in terms of loops containing only brackets from $\overset{\triangleleft}{N}_r^{(2)}$.

According to Definition 10, an "universal" structure of the segment $X^{(1)}$ can be given by $X^{(1)} = c_1 O_1^+ c_2 O_2^+ ... c_{m_i} O_{m_i}^+ c_{m_i+1}$, where c_k, $1 \leq k \leq m_i + 1$, are "constant" segments that do not contain any loop, and O_j are multilevel loops, $1 \leq j \leq m_i$. One of the structure, in terms of concatenation and $+$ Kleene operations, that O_j may have, is given by (2).

$$(\lfloor_{o_1^{(m)}}\lfloor_{o_2^{(m)}}...(\lfloor_{o_1^{(m-1)}}\lfloor_{o_2^{(m-1)}}...(...(\lfloor_{o_1^{(2)}}\lfloor_{o_2^{(2)}}...\lfloor_{o_{p_{(2)}}^{(2)}}(\lfloor_{o_1^{(1)}}\lfloor_{o_2^{(1)}}...\lfloor_{o_{\check{q}^{(1)}}^{(1)}})^{+_1}$$
$$...\lfloor_{o_{\check{q}^{(2)}}^{(2)}})^{+_2}...)^{+_{m-2}}...\lfloor_{o_{\check{q}^{(m-1)}}^{(m-1)}})^{+_{m-1}}...\lfloor_{o_{\check{q}^{(m)}}^{(m)}})^{+_m}. \quad (2)$$

It can be seen now that an ATM $\mathcal{A}$ that tries to guess the positions of the two subwords generated during the execution of the loop (1), placed at the left and right side of the core segment $a_{n_t}a_{n_t+1}$, and synchronize them according to each repetition of the loop (1), performed of $\check{k}^{(1)}$ times, and each repetition of a loop placed at the level q of (2), performed of $\hat{k}^{(q)}$ times, where $\check{k}^{(1)}$ and $\hat{k}^{(q)}$, $1 \leq q \leq m$, are arbitrarily large, $\mathcal{A}$ has to guess $\check{k}^{(1)} \prod_{q=1}^{m} \hat{k}^{(q)}$ arbitrarily large integers[13] (that are values of plus-heights $+_q$, $1 \leq q \leq m$, from (2)). This "naive" procedure makes it impossible to store all these numbers in only $\mathcal{O}(\log n)$ space. Next we show that we can do a well synchronization by guessing only a finite number of arbitrarily large positive integers.

Let $\hat{k}^{(2)}$ be the value of $+_2$ in (2). At a very first glance for the first level $\mathcal{A}$ should guess $\hat{k}^{(2)}$ arbitrarily large integers greater than one, i.e., $\hat{k}_1^{(1)}$, $\hat{k}_2^{(1)}$, ..., $\hat{k}_{\hat{k}^{(2)}}^{(1)}$. The sum of all these values is

$$\hat{k}_1^{(1)} + \hat{k}_2^{(1)} + ... + \hat{k}_{\hat{k}^{(2)}}^{(1)} \geq \hat{k}^{(2)} \geq 1. \quad (3)$$

[13] In order for $\mathcal{A}$ to guess these positions, $\mathcal{A}$ has to guess the length of these substrings.

Therefore, guessing only one large integer, say $\hat{s}^{(1)}$, that covers the above sum, such that $\hat{s}^{(1)} \geq \hat{k}^{(2)}$, is enough to have a well synchronization of repetitions of the loop placed at the first level according to repetitions of the loop placed at the second level. Indeed, any arbitrarily large number $\hat{s}^{(1)}$, $\hat{s}^{(1)} \geq \hat{k}^{(2)}$, can be decomposed into a sum of form (3) in which all terms are greater than one. Going further with upper levels, let $\hat{k}^{(3)}$ be the arbitrarily large value of $+_3$. Then for the second level $\mathcal{A}$ should guess $\hat{k}^{(3)}$ arbitrarily large integers greater than one, i.e., $\hat{k}_1^{(2)}$, $\hat{k}_2^{(2)}$, ..., $\hat{k}_{\hat{k}^{(3)}}^{(2)}$. For the first level $\mathcal{A}$ should guess $\Sigma_{i=1}^{\hat{k}^{(3)}} \hat{k}_i^{(2)}$ arbitrarily large integers greater than one, i.e., $\hat{k}_{1,1}^{(1)}$, $\hat{k}_{2,1}^{(1)}$, ..., $\hat{k}_{\hat{k}_1^{(2)},1}^{(1)}$, $\hat{k}_{1,2}^{(1)}$, $\hat{k}_{2,2}^{(1)}$, ..., $\hat{k}_{\hat{k}_2^{(2)},2}^{(1)}$, ..., $\hat{k}_{1,\hat{k}^{(3)}}^{(1)}$, $\hat{k}_{2,\hat{k}^{(3)}}^{(1)}$, ..., $\hat{k}_{\hat{k}_{\hat{k}^{(3)}}^{(2)},\hat{k}^{(3)}}^{(1)}$. The sums of all integers guessed for the second and the first level are, respectively,

$$\hat{k}_1^{(2)} + \hat{k}_2^{(2)} + ... + \hat{k}_{\hat{k}^{(3)}}^{(2)} \geq \hat{k}^{(3)} \geq 1, \tag{4}$$

$$\text{and } \Sigma_{j=1}^{\hat{k}^{(3)}} (\Sigma_{i=1}^{\hat{k}_j^{(2)}} (\hat{k}_{i,j}^{(1)})) \geq \hat{k}_1^{(2)} + \hat{k}_2^{(2)} + ... + \hat{k}_{\hat{k}^{(3)}}^{(2)} \geq \hat{k}^{(3)} \geq 1. \tag{5}$$

Guessing only two arbitrarily large integers $\hat{s}^{(2)}$ and $\hat{s}^{(1)}$ that cover the sum (4) and (5), respectively, such that $\hat{s}^{(1)} \geq \hat{s}^{(2)} \geq \hat{k}^{(3)} \geq 1$, is enough to have a well synchronization of repetitions of the loops placed at the first level according to repetitions of loops placed at the second and the third level. Indeed, any arbitrarily large numbers $\hat{s}^{(1)}$ and $\hat{s}^{(2)}$, with $\hat{s}^{(1)} \geq \hat{s}^{(2)} \geq \hat{k}^{(3)} \geq 1$, can be decomposed[14] into a sum of the form (4) or (5), without spoiling neither the structure of the loop (2) nor the structure of the loop (1).

By induction on m, i.e., the $+_\hbar$ of loop (2), it can be proved that it is sufficient for $\mathcal{A}$ to guess only m arbitrarily large integers $\hat{s}^{(p)}$, $1 \leq p \leq m$, together with $\check{k}^{(1)}$, for the "mother" loop (1), such that $\hat{s}^{(1)} \geq \hat{s}^{(2)} \geq ... \geq \hat{s}^{(m)} \geq \check{k}^{(1)} \geq 1$ in order to have a well synchronization of all loops composing (2) and (1).

So far we have considered only $+_\hbar$ nested loops, i.e., no concatenation of two or more loops occur inside a particular level of a loop. However, in order to make a correct synchronization of the shuffle of languages from the left side of the core segment with the shuffle of languages from the right side of the core segment of the input word, we must provide a full description of all loops at any level of a $+_\hbar$ nested loop. In order to accomplish this aim we introduce a tree representation of any arbitrary $+_\hbar$ regular expression in string form.

Let $\hat{\Re}_m$ be an arbitrary regular expression in string form of $+_m$ over a finite alphabet Σ. According to Definition 10 this can be described as follows

[14] Inequalities (4) and (5) assures that the decomposition of $\hat{s}^{(1)}$ and $\hat{s}^{(2)}$ into terms greater than one is possible. Correct decompositions are guaranted by a membership checking of the string, generated by the corresponding loop, into regular sets of type $'\mathcal{L}_l^{c_\psi^{+(1)}}{}'$ and $'\mathcal{L}_r^{c_\psi^{+(1)}}{}'$, built for the loop (1), and in general of type $'\mathcal{L}_l^{c_\psi^{+(m)}}{}'$ and $'\mathcal{L}_r^{c_\psi^{+(m)}}{}'$, built for loops of $+_m$, $m \geq 1$, (see Case 2.2) executed during the Algorithms 2, 3, and 4. All these regular sets are built on the base of only + Kleene closures.

$$\hat{\Re}_m = w_1^{(1)}(O_1)^{+_{m_1}} w_2^{(1)}(O_2)^{+_{m_2}} ... w^{(1)}_{p_1^{(1)}}(O_{p_1^{(1)}})^{+_{m_{p_1^{(1)}}}} w^{(1)}_{p_1^{(1)}+1} \quad (6)$$

in which each $w_j^{(1)} \in \Sigma^*$, contains no loops, $1 \leq j \leq p_1^{(1)}+1$, and each $(O_i)^{+_{m_i}}$ is a loop of plus-height m_i, $1 \leq i \leq p_1^{(1)}$, $m = \max\{m_1, ..., m_{p_1^{(1)}}\}$. According to the recursive Definition 10, each O_i, $1 \leq i \leq p_1^{(1)}$, can be further described as

$$O_i = w_{i,1}^{(2)}(O_{i,1})^{+_{h_{i,1}}} w_{i,2}^{(2)}(O_{i,2})^{+_{h_{i,2}}} ... w^{(2)}_{i,p_i^{(2)}}(O_{i,p_i^{(2)}})^{+_{h_{i,p_i^{(2)}}}} w^{(2)}_{i,p_i^{(2)}+1}$$

where $h_{i,j}$ is the $+_\hbar$ of the loop $O_{i,j}$, $1 \leq j \leq p_i^{(2)}$, $h_{i,j} \leq m_i - 1$, $1 \leq i \leq p_1^{(1)}$. Next, each $O_{i,j}$, $1 \leq i \leq p_1^{(1)}$, $1 \leq j \leq p_i^{(2)}$, is defined as

$$O_{i,j} = w_{i,j,1}^{(3)}(O_{i,j,1})^{+_{h_{i,j,1}}} w_{i,j,2}^{(3)}(O_{i,j,2})^{+_{h_{i,j,2}}} ... w^{(3)}_{i,j,p_j^{(3)}}(O_{i,j,p_j^{(3)}})^{+_{h_{i,j,p_j^{(3)}}}} w^{(3)}_{i,j,p_j^{(3)}+1}$$

where $h_{i,j,k}$ is the $+_\hbar$ of the loop $O_{i,j,k}$, $1 \leq k \leq p_j^{(3)}$, $h_{i,j,k} \leq h_{i,j} \leq m_i - 2$, $1 \leq j \leq p_i^{(2)}$, $1 \leq i \leq p_1^{(1)}$. The recursion continues until the last step, when no more loops can be defined, i.e., the recursion ends up with loops of the form

$$O_{i_1,i_2,...,i_{m_i-1}} = w^{(m_i)}_{i_1,i_2,...,i_{m_i-1},1}(O_{i_1,i_2,...,i_{m_i-1},1})^{+_{h_{i_1,i_2,...,i_{m_i-1},1}}} w^{(m_i)}_{i_1,i_2,...,i_{m_i-1},2}$$

$$...w^{(m_i)}_{i_1,...,i_{m_i-1},p^{(m_i)}_{i_{m_i-1}}}(O_{i_1,...,i_{m_i-1},p^{(m_i)}_{i_{m_i-1}}})^{+_{h_{i_1,...,i_{m_i-1},p^{(m_i)}_{i_{m_i-1}}}}} w^{(m_i)}_{i_1,...,i_{m_i-1},p^{(m_i)}_{i_{m_i-1}}+1}$$

where $h_{i_1,i_2,...,i_{m_i}} \leq 1$, $1 \leq i_j \leq p^{(j)}_{i_{j-1}}$, $2 \leq j \leq m_i$, $1 \leq i_1 \leq p_1^{(1)}$, and

$$O_{i_1,i_2,...,i_{m_i}} = w^{(m_i+1)}_{i_1,i_2,...,i_{m_i},1} O_{i_1,i_2,...,i_{m_i},1} w^{(m_i+1)}_{i_1,i_2,...,i_{m_i},2} O_{i_1,i_2,...,i_{m_i},2}$$

$$...w^{(m_i+1)}_{i_1,i_2,...,i_{m_i},p^{(m_i+1)}_{i_{m_i}}} O_{i_1,i_2,...,i_{m_i},p^{(m_i+1)}_{i_{m_i}}} w^{(m_i+1)}_{i_1,i_2,...,i_{m_i},p^{(m_i+1)}_{i_{m_i}}+1}$$

where each $O_{i_1,i_2,...,i_{m_i},i_{m_i+1}}$, $1 \leq i_{m_i+1} \leq p^{(m_i+1)}_{i_{m_i}}$ contains no loop.

To each loop $O_{i_1,i_2,...,i_j}$, $1 \leq i_j \leq p^{(j)}_{i_{j-1}}$, $2 \leq j \leq m_i - 1$, $1 \leq i_1 \leq p_1^{(1)}$, we associate three parameters, the name of the loop, encoded into the lower indexes, i.e., $i_1, i_2, ..., i_j$, the height of the loop, i.e., $h_{i_1,i_2,...,i_j}$, $h_{i_1,i_2,...,i_j} \leq h_{i_1,i_2,...,i_{j-1}} \leq m_{i_1} - (j-1)$, and the length of the loop defined as $\lg(O_{i_1,i_2,...,i_j}) = |w^{(j+1)}_{i_1,i_2,...,i_j,1}| + ... + |w^{(j+1)}_{i_1,i_2,...,i_j,p^{(j+1)}_{i_j}+1}|$.

The regular expression $\hat{\Re}_m$ is characterized by $\hat{\Re}_m$ as name, $+_m$ as $+_\hbar$, and $\lg(\hat{\Re}_m) = |w_1^{(1)}| + ... + |w^{(1)}_{p_1^{(1)}+1}|$ as length. Loops that cancel the above recursion[15], i.e., $O_{i_1,...,i_{m_i}}$, $1 \leq i_{m_i} \leq p^{(m_i)}_{i_{m_i-1}}$, are characterized by the name of the loop, i.e., $i_1, ..., i_{m_i}$, the height of the loop, i.e., $h_{i_1,...,i_{m_i}} = 1$, and the length of the loop

[15] These loops correspond to the loop of type $([_{o_1^{(1)}}[_{o_2^{(1)}}...[_{o^{(1)}_{\hat{q}^{(1)}}})^{+_1}$ from (2).

defined as $\lg(O_{i_1,...,i_{m_i}}) = |w^{(m_i+1)}_{i_1,...,i_{m_i},1}| + |O_{i_1,...,i_{m_i},1}| + ... + |O_{i_1,...,i_{m_i},p^{(m_i+1)}_{i_{m_i}}}| + |w^{(m_i+1)}_{i_1,...,i_{m_i},p^{(i_{m_i}+1)}_{i_{m_i}}+1}|$.

To each regular expression $\hat{\Re}_m$ we associate a finite tree $\hat{\mathcal{T}}(\hat{\Re}_m) = (\mathcal{V}, \mathcal{E})$, called the *coverability tree* of $\hat{\Re}_m$. Nodes in this tree correspond to loops $O_{i_1,...,i_j}$, $1 \leq i_j \leq p^{(j)}_{i_{j-1}}$, $1 \leq j \leq m_i - 1$. Each node is indexed by the name of the loop, i.e., $i_1, ..., i_j$, and it is labeled by the parameters that characterize that loop, i.e., the height $h_{i_1,...,i_j}$, and length $\lg(O_{i_1,...,i_j})$. If $O_{i_1,...,i_j}$ is an arbitrary loop composed of the loops $O_{i_1,...,i_j,i_{j+1}}$, $1 \leq i_{j+1} \leq p^{(j+1)}_{i_j}$, then there exists an edge in $\hat{\mathcal{T}}(\hat{\Re}_m)$ from the node indexed by $i_1, ..., i_j$ to each node indexed by $i_1, ..., i_j, i_{j+1}$.

Definition 11. Let $\hat{\Re}_m$ be an arbitrary regular expression defined in (6). The *coverability tree* associated with $\hat{\Re}_m$ is a rooted tree $\hat{\mathcal{T}}(\hat{\Re}_m) = (\mathcal{V}, \mathcal{E})$, in which the set of vertices $\mathcal{V}$, and the set of edges $\mathcal{E}$, are defined as follows:

1. the root r_0 is indexed by $\hat{\Re}_m$ and it is labeled by the parameters that characterize $\hat{\Re}_m$, i.e., the $+_{\hbar} = +_m$ and the length $\lg(\hat{\Re}_m)$, i.e., $r_0 = v_{\hat{\Re}_m}(+_m, \lg(\hat{\Re}_m))$. The root r_0 has $p^{(1)}_1$ children. The i^{th} child of the root corresponds to the i^{th} loop O_i, $1 \leq i \leq p^{(1)}_1$. The node associated with the i^{th} loop is indexed by i and labeled by the parameters that characterize O_i, i.e., $(+_{m_i}, \lg(O_i))$, $1 \leq i \leq p^{(1)}_1$. Therefore, there is an edge in $\mathcal{E}$ from $v_{\hat{\Re}_m}(+_m, \lg(\hat{\Re}_m))$ to each $v_i(+_{m_i}, \lg(O_i))$, $1 \leq i \leq p^{(1)}_1$.
2. if $v_{i_1,...,i_j}(+_{h_{i_1,...,i_j}}, \lg(O_{i_1,...,i_j}))$ is the node associated with the j^{th} loop $O_{i_1,...,i_j}$, $1 \leq i_j \leq p^{(j)}_{i_{j-1}}$, $2 \leq j \leq m_i - 1$, then $v_{i_1,...,i_j}$ has $p^{(j+1)}_{i_j}$ ordered children. The child i_{j+1} of the node $v_{i_1,...,i_j}$, $1 \leq i_{j+1} \leq p^{(j+1)}_{i_j}$, $2 \leq j \leq m_i - 1$, corresponds to the loop $O_{i_1,...,i_{j+1}}$. This node is encoded by $v_{i_1,...,i_{j+1}}(+_{h_{i_1,...,i_{j+1}}}, \lg(O_{i_1,...,i_{j+1}}))$ and a new edge from $v_{i_1,...,i_j}$ to $v_{i_1,...,i_{j+1}}$ is added in $\mathcal{E}$.
3. the leaves are nodes $v_{i_1,...,i_{m_i}}(+_{h_{i_1,...,i_{m_i}}} = +_1, \lg(O_{i_1,...,i_{m_i}}))$ associated with loops of the form $O_{i_1,...,i_{m_i}}$, $1 \leq i_{m_i} \leq p^{(m_i)}_{i_{m_i-1}}$.

Example 4 The loop $\hat{\Re}_4 = S[_1((]_2\ [_1)^+(]_3(]_2\ [_1)^+)^+)^+]_3[^t_4$ of Example 2, has the coverability tree $\hat{\mathcal{T}}(\hat{\Re}_4) = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is composed of the nodes $v_{\hat{\Re}_4}(+_3, 4)$, $v_1(+_3, 0)$ associated with the loop $((]_2\ [_1)^+(]_3(]_2\ [_1)^+)^+)^+$, $v_{1,1}(+_1, 2)$ associated with the loop $(]_2\ [_1)^+$, $v_{1,2}(+_2, 1)$ associated with the loop $(]_3(]_2\ [_1)^+)^+$, and $v_{1,2,1}(+_1, 2)$ associated with the loop $(]_2\ [_1)^+$. The set of edge is $\mathcal{E} = \{v_{\hat{\Re}_4}v_1, v_1v_{1,1}, v_1v_{1,2}, v_{1,2}v_{1,2,1}\}$. ♣

According to Definition 11, the loop O_j described in (2) is characterized by a linear tree in which the root of the tree corresponding to the loop O_j, is characterized by $(+_m, 0)$. The child of the root is the loop placed at the m^{th} level, denoted as $O^{+_m}_j$. It is characterized by the parameters $(+_m, \hat{q}^{(m)})$. The child of the loop $O^{+_m}_j$ is the loop placed at the level $m-1$, denoted as $O^{+_{m-1}}_j$,

and it is characterized by $(+_{m-1}, \hat{q}^{(m-1)})$. The very last child is the loop $O_j^{+_2}$ characterized by the parameters $(+_2, \hat{q}^{(2)})$. It has as child (that is also the leaf of the tree) the loop $O_j^{+_1}$ characterized by $(+_1, \hat{q}^{(1)})$.

Consider now the situation, where $\mathcal{A}$ guesses the $+_\hbar$ value, $\check{k}^{(1)}$, of the loop (1). Then the length of the subword generated by the loop (1) at the left side of the core segment $a_{n_t}a_{n_t+1}$ is $\check{q}_s^{(1)}\check{k}^{(1)}$. In order to guess the subword generated by the loop (1) at the right side of the core segment $a_{n_t}a_{n_t+1}$, as explained before, $\mathcal{A}$ has to guess m arbitrarily large integers $\hat{s}^{(p)}$, $1 \leq p \leq m$. Recall that each $\hat{s}^{(p)}$, $1 \leq p \leq m$, stands for the sum of all $+_\hbar$ values of loops placed at the level p of (2). On the other hand if the value of the plus-height of the loop (1) is $\check{k}^{(1)}$ then the loop (2) must be "repeated" of at least $\check{k}^{(1)}$ times. According to the above explanations the positive integers guessed so far must fulfill the inequalities

$$\hat{s}^{(1)} \geq \hat{s}^{(2)} \geq ... \geq \hat{s}^{(m-1)} \geq \hat{s}^{(m)} \geq \check{k}^{(1)} \geq 1. \tag{7}$$

The length of the subword generated by (1) in the right side of $a_{n_t}a_{n_t+1}$ is given by $\hat{q}^{(m)}\hat{s}^{(m)} + \hat{q}^{(m-1)}\hat{s}^{(m-1)} + ... + \hat{q}^{(1)}\hat{s}^{(1)}$, in which $\hat{q}^{(p)}$, $1 \leq p \leq m$, stands for the length of the loop placed at the level p of (2).

In this moment we have a general view of the structure of a loop through the graphical representation provided by the coverability tree associated with that loop. The loop (2) is a particular case for which the associated tree has a linear representation. It can be considered as a proper branch (a path from the root to a leaf) of a most general tree as introduced in Definition 11. Using this framework we can easily generalize condition (7) for the whole tree, i.e., conditions of type (7) must be imposed along all branches (paths from the root to each leaf) of the coverability tree. Consider a loop having a tree representation given by all paths from the root v_0 to each leaf, of the form:

$$\left.\begin{array}{l} \hat{\wp}_1 = \hat{v}_0, \hat{v}_{1,1}, \hat{v}_{1,2}, ..., \hat{v}_{1,\hat{f}_1}, \\ \hat{\wp}_2 = \hat{v}_0, \hat{v}_{2,1}, \hat{v}_{2,2}, ..., \hat{v}_{2,\hat{f}_2}, \ ..., \\ \hat{\wp}_{\hat{m}} = \hat{v}_0, \hat{v}_{\hat{m},1}, \hat{v}_{\hat{m},2}, ..., \hat{v}_{\hat{m},\hat{f}_{\hat{m}}}, \end{array}\right\} \tag{8}$$

where each node $\hat{v}_{i,j}$ is characterized by the parameters $(+_j, \hat{l}_{i,j})$, in which j is the $+_\hbar$ (that gives the ranking place in the path) and $\hat{l}_{i,j}$ is the length of the loop characterized by $\hat{v}_{i,j}$. For the whole tree $\mathcal{A}$ must guess at most $\hat{f}_1 + ... + \hat{f}_{\hat{m}}$ arbitrarily large numbers of integers[16], i.e., for each node $\hat{v}_{i,j}$, $\mathcal{A}$ guesses an arbitrarily large integer $\hat{s}^{i,j}$, $1 \leq i \leq \hat{m}$, $1 \leq j \leq \hat{f}_i$, and for each path checks whether the next inequalities hold:

$$\hat{s}^{i,\hat{f}_i} \geq \hat{s}^{i,\hat{f}_i - 1} \geq ... \geq \hat{s}^{i,2} \geq \hat{s}^{i,1} \geq \hat{s}^0 \geq \check{k}^{(1)} \geq 1 \tag{9}$$

where $\hat{s}^0$ is the arbitrarily large value associated with the root.

Notice that, for common branches $\mathcal{A}$ guesses single values. If for instance, two paths fork in a node $\hat{v}_{i_1}$, i.e., $\hat{\wp}_1 = \hat{v}_0, \hat{v}_1, ..., \hat{v}_{i_1}, \hat{v}_{i_{1,2}}, ..., \hat{v}_{i_{1,\hat{f}_1}}$, $\hat{\wp}_2 = \hat{v}_0, \hat{v}_1, ..., \hat{v}_{i_1}$,

[16] The plus-height of the loop described by (8) is given by $m = \max\{\hat{f}_1, ..., \hat{f}_{\hat{m}}\}$.

$\hat{v}_{i_{2,2}}, ..., \hat{v}_{i_{2,\hat{f}_2}}$, then up to the bifurcation point $\hat{v}_{i_1}$, $\mathcal{A}$ guesses a single value for each node, even if there are two different paths. The same restriction is imposed when we compute the length of the string generated by the loop. If a node belongs to different paths, when we compute the length of the string generated by the entire loop, the arbitrarily large integer guessed for that node, multiplied by the length of the loop associated with that node, is added into the final length only once. Thus, a loop represented by a tree composed of two paths $\wp_1$ and $\wp_2$ generates a string of length $\ell_{\wp_1,\wp_2} = (\hat{s}^0\hat{l}_0 + \hat{s}^1\hat{l}_1 + ... + \hat{s}^{i_1}\hat{l}_{i_1}) + (\hat{s}^{i_{1,2}}\hat{l}_{i_{1,2}} + ... + \hat{s}^{i_{1,\hat{f}_1}}\hat{l}_{i_{1,\hat{f}_1}}) + (\hat{s}^{i_{2,2}}\hat{l}_{i_{2,2}} + ... + \hat{s}^{i_{2,\hat{f}_2}}\hat{l}_{i_{2,\hat{f}_2}})$. A sum operation with this restriction, i.e., each term is added only once, is denoted as $\bigoplus$. We use this notation whenever there is no possibility to make distinction between different nodes in a coverability tree associated with a loop. With these considerations the length of the string generated by loop (2) is computed by

$$\bigoplus_{i=1}^{m}(\hat{s}^0\hat{l}_0 + \hat{s}^{i,1}\hat{l}_{i,1} + \hat{s}^{i,2}\hat{l}_{i,2} + ... + \hat{s}^{i,\hat{f}_i}\hat{l}_{i,\hat{f}_i}). \tag{10}$$

Reconsider the loop (1) for which the segment $X^{(1)} = X_{i_1^{(1)}}X_{i_2^{(1)}}...X_{i_{\check{q}^{(1)}}^{(1)}}$ has an "universal" representation through a tree composed of all paths from the root v_0 to each leaf, i.e., $\hat{\wp}_1$, $\hat{\wp}_2$,..., $\hat{\wp}_{\hat{m}}$, for which the conditions (9) hold, then for the loop (1) we have already built the regular languages

$\mathcal{L}_l{}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}}) = \{x^{\check{k}^{(1)}} | x \in \mathcal{L}_l{}^{c_\psi^{(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}}),\ \check{k}^{(1)} \geq 1\}$, and

$\mathcal{L}_r{}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}}) = \{y^{\hat{k}^{(1)}} | y \in \mathcal{L}_r{}^{c_\psi^{(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}}),\ \hat{k}^{(1)} \geq 1\}$, where

$\mathcal{L}^{c_\psi^{(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}}) = \{(x,y) | x = \psi_l(\Psi_l(]_{j_1^{(1)}}...]_{j_{\check{q}^{(1)}}^{(1)}})), y = \psi_r((\Psi_r(\ell_{i_1^{(1)}}...\ell_{i_{\check{q}^{(1)}}^{(1)}}))^r)$, $\ell_{i_p^{(1)}} \in \mathcal{L}([_{j_p^{(1)}}[_{j_{p+1}^{(1)}})$, $1 \leq p \leq \check{q}^{(1)} - 1$, $\ell_{i_{\check{q}^{(1)}}^{(1)}} \in \mathcal{L}([_{j_{\check{q}^{(1)}}^{(1)}}[_{j_1^{(1)}})\}$.

Consider now the particular case of a LCFG in linear-Dyck normal form for which the dependency graph $\mathcal{G}$ has only one class of terminal paths represented by a regular expression that starts with S (the axiom), ends with t (a terminal node), and in between is composed of a loop of type (1). Let $\hat{\mathcal{T}}$ be the coverability tree associated with the segment $X^{(1)}$, described by terminal paths given in (8), with the conditions (9) upon its nodes. With the above information recorded, by using constant space, Algorithm 1 can be improved as follows.

Algorithm 2 Let $w \in T^*$ be an input word of length n of the form $a_1a_2...a_n$. First $\mathcal{A}$ existentially proceeds similar as in Level 1 and Level 2 of Algorithm 1. Then, $\mathcal{A}$ existentially guesses an arbitrarily large integer $\check{k}^{(1)}$ and at most $\hat{f}_1 + ... + \hat{f}_{\hat{m}}$ arbitrarily large numbers of positive integers, i.e., for each node $\hat{v}_0$ and $\hat{v}_{i,j}$ from (8), $\mathcal{A}$ guesses arbitrarily large integers, $\hat{s}^0$ and $\hat{s}^{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq \hat{f}_i$, and for each path checks whether the inequalities (9) hold. Then,

1. $\mathcal{A}$ computes $\ell_l = \check{q}^{(1)}\check{k}^{(1)}$, checks whether $n_t - 1 = \ell_l$, and whether the prefix of length ℓ_l of w, i.e., $a_1...a_{\ell_l} = a_1...a_{n_t-1}$ belongs to the regular set ${\mathcal{L}_l}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$;
2. $\mathcal{A}$ computes $\ell_r = \bigoplus_{i=1}^{m}(\hat{s}^0\hat{l}_0 + \hat{s}^{i,1}\hat{l}_{i,1} + \hat{s}^{i,2}\hat{l}_{i,2} + ... + \hat{s}^{i,\hat{f}_i}\hat{l}_{i,\hat{f}_i})$, checks whether $n - n_t - 1 = \ell_r$, and whether the suffix of length ℓ_r of w, i.e., $a_{n-\ell_r+1}...a_n = a_{n_t+2}...a_n$ belongs to the regular set ${\mathcal{L}_r}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$. $\diamond$

Since $\hat{f}_1 + ... + \hat{f}_{\hat{m}}$ is finite (being bounded above by $m \cdot k$, where m is the finite $+_\hbar$ of loop (1) and k is the maximum number of different loops occurring at any level), there will be $\mathcal{O}(n^s)$ existential branches corresponding to the values of all plus-heights interfering in loop (1), where $s = \mathcal{O}(\hat{f}_1 + ... + \hat{f}_{\hat{m}})$. Therefore, the computation tree associated with $\mathcal{A}$ in this case can be converted into a binary tree of height at most $\mathcal{O}(s\log\, n)$. Furthermore, all operations performed during Algorithm 2 (check out inequalities of type (9), product or sum of a finite number of $\mathcal{O}(\log\, n)$ binary numbers) are functions belonging to NC^1.

Algorithm 2 is correct, in the sense that it accepts only words belonging to the language generated by a LCFG in linear-Dyck normal form, characterized by a dependency graph $\mathcal{G}$ composed of only one class of terminal paths represented by a regular expression that starts with S (the axiom), ends with t (a terminal node) such that the pair $[_j^t\ ]_j^t$ generates the core segment $a_{n_t}a_{n_t+1}$, and in between is composed of a loop of type (1) whose coverability tree $\hat{\mathcal{T}}$ is described in (8).

Indeed, for a well synchronization of a word in ${\mathcal{L}_l}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$ with a word in ${\mathcal{L}_r}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$, it is enough to check whether knowing that the first loop is performed of $\check{k}^{(1)}$ times, all loops inside the "mother" loop (1) are performed of at least $\check{k}^{(1)}$ times, too, i.e., to check the inequalities (9). The other inequalities between proper values associated with nodes in (8) concern well synchronization of loops induced only by brackets from $\overset{\triangleleft\,(2)}{N}_r$.

A main question arises concerning the $\mathcal{O}(\hat{f}_1 + ... + \hat{f}_{\hat{m}})$ arbitrarily large integers that $\mathcal{A}$ in Algorithm 2, needs to guess for a correct solution of the membership problem. Why does not $\mathcal{A}$ guess only the core index n_t, and checks whether the prefix of w, $w_{pf} = a_1...a_{n_t-1}$ belongs to ${\mathcal{L}_l}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$, and the suffix $w_{sf} = a_{n_t+2}...a_n$ belongs to ${\mathcal{L}_r}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$? Can the above strategy be a correct solution? The answer is, of course, No. First, because even if we have a positive answer for the membership searching of $w_{pf} \in {\mathcal{L}_l}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$ and $w_{sf} \in {\mathcal{L}_l}^{c_\psi^{+(1)}}([_{j_1^{(1)}}...[_{j_{\check{q}^{(1)}}^{(1)}})$, when we concatenate $w_{pf}a_{n_t}a_{n_t+1}w_{sf}$, we may accept one w_{pf} that is generated of infinitely $\check{k}^{(1)}$ many times by the loop (1) and one w_{sf} that, inside (1), is generated of only of a finite number of times. Which is clearly not correct. Therefore, a synchronization at the level of the "mother"

loop (1) must be checked. On the other hand a correct synchronization of all loops in (2) according to repetitions of loop (1) can be done only if we know all values $\hat{s}^0$ and $\hat{s}^{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq \hat{f}_i$, associated with the nodes $\hat{v}_0$ and $\hat{v}_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq \hat{f}_i$, in (8). This observation is valid for any kind of multilevel loop induced by brackets from $\overset{\triangleright\,(2)}{N}_l$. The above strategy described at the level of a loop of plus-height one, induced by brackets from $\overset{\triangleright\,(2)}{N}_l$ stands as a basis of an "universal" loop induced by brackets from $\overset{\triangleright\,(2)}{N}_l$, case considered below.

$\surd$

Case 2. 2 (Loops of plus-height greater than one induced by brackets from $\overset{\triangleright\,(2)}{N}_l$) A multilevel loop, similar to the loop (2), induced by brackets from $\overset{\triangleright\,(2)}{N}_l$, looks like (11)

$$(]_{j_1^{(m)}} X_{i_1^{(m)}}]_{j_2^{(m)}} X_{i_2^{(m)}} ... (]_{j_1^{(m-1)}} X_{i_1^{(m-1)}}]_{j_2^{(m-1)}} X_{i_2^{(m-1)}} ... (...(]_{j_1^{(1)}} X_{i_1^{(1)}} ...]_{j_{\check{q}^{(1)}}^{(1)}} X_{i_{\check{q}^{(1)}}^{(1)}})^{+_1}$$

$$...)^{+_{m-2}} ... X_{i_{\check{q}^{(m-1)}-1}^{(m-1)}}]_{j_{\check{q}^{(m-1)}}^{(m-1)}} X_{i_{\check{q}^{(m-1)}}^{(m-1)}})^{+_{m-1}} ... X_{i_{\check{q}^{(m)}-1}^{(m)}}]_{j_{\check{q}^{(m)}}^{(m)}} X_{i_{\check{q}^{(m)}}^{(m)}})^{+_m} \qquad (11)$$

where $]_{j_p^{(l)}} \in \overset{\triangleright\,(2)}{N}_l$, and $X_{i_p^{(l)}}$ is composed of brackets from $\overset{\triangleleft\,(2)}{N}_r$, more precisely, $X_{i_p^{(l)}} \in \mathcal{L}([_{j_p^{(l)}} [_{j_{p+1}^{(l)}})$, $1 \leq p \leq \check{q}^{(l)}-1$, $X_{i_{\check{q}^{(l)}}^{(l)}} \in \mathcal{L}([_{j_{\check{q}^{(l)}}^{(l)}} [_{j_1^{(l)}})$, $1 \leq p \leq \check{q}^{(l)}$, $1 \leq l \leq m$. If $X^{(l)} = X_{i_1^{(l)}} X_{i_2^{(l)}} ... X_{i_{\check{q}^{(l)}}^{(l)}}$, $1 \leq l \leq m$, then $X^{(l)}$ is a regular expression of type (6).

The coverability tree associated with $X^{(l)}$, defined in Definition 11, is denoted by $\hat{\mathcal{T}}(X^{(l)}) = (\mathcal{V}_{X^{(l)}}, \mathcal{E}_{X^{(l)}})$. The coverability tree associated with the regular expression (11), denoted by $\check{\Re}$, is a finite tree $\check{\mathcal{T}}(\check{\Re}) = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V} = \mathcal{V}_{\check{\Re}} \cup \mathcal{V}_{X^{(1)}} \cup ... \cup \mathcal{V}_{X^{(m)}}$, where $\mathcal{V}_{\check{\Re}} = \{v_{(0)}, v_{(1)}, ..., v_{(m)}\}$, each node $v_{(l)}(+_l, \check{q}^{(l)})$ corresponds to the loop placed at the level l, $1 \leq l \leq m$, composed of only brackets from $\overset{\triangleright\,(2)}{N}_l$. The set of edges is $\mathcal{E} = \mathcal{E}_{\check{\Re}} \cup \mathcal{E}_{X^{(1)}} \cup ... \cup \mathcal{E}_{X^{(m)}}$, where $\mathcal{E}_{\check{\Re}} = \{v_{(1)} v_{X^{(1)}}, v_{(1)} v_{(2)}, v_{(2)} v_{X^{(2)}}, v_{(2)} v_{(3)}, ..., v_{(m-1)} v_{X^{(m-1)}}, v_{(m-1)} v_{(m)}, v_{(m)} v_{X^{(m)}}\}$, in which $v_{X^{(l)}}$, $1 \leq l \leq m$, is the root of $\hat{\mathcal{T}}(X^{(l)}) = (\mathcal{V}_{X^{(l)}}, \mathcal{E}_{X^{(l)}})$.

Let $\check{k}_m$ be the value of the $+_m$, $\check{s}_l$, $1 \leq l \leq m-1$, be the sum of all $+_l$'s values, $1 \leq l \leq m-1$, and $\hat{s}_l$, $1 \leq l \leq m$, be the sum of all $+_l$ values of the root of each tree $\hat{\mathcal{T}}(X^{(l)})$. Then the conditions (7) become

$$\left.\begin{array}{l} \check{s}^{(1)} \geq \check{s}^{(2)} \geq ... \geq \check{s}^{(m-1)} \geq \check{k}^{(m)} \geq 1, \\ \hat{s}^{(l)} \geq \check{s}^{(l)}, 1 \leq l \leq m-1, \\ \hat{s}^{(m)} \geq \check{k}^{(m)}. \end{array}\right\} \qquad (12)$$

Furthermore, each $\hat{s}^{(l)}$, $1 \leq l \leq m$, corresponds to the root $v_{X^{(l)}}$, and satisfies proper inequalities of type (9) corresponding to the coverability tree $\hat{\mathcal{T}}(X^{(l)})$. The length of the string generated by (11) on the left side of the core index is

$$\check{k}^{(m)} \check{q}_{(m)} + \check{s}^{(m-1)} \check{q}_{(m-1)} + ... + \check{s}^{(2)} \check{q}_{(2)} + \check{s}^{(1)} \check{q}_{(1)}. \qquad (13)$$

The length of the string generated by the loop (11) on the right side of the core index is a sum of type (10) considered over all trees $\hat{\mathcal{T}}(X^{(l)})$.

To have a generalization of loop (11) let us consider $\check{\Re}_m$ an arbitrary regular expression in string form of $+_m$ over a finite alphabet Σ. According to Definition 10, this can be described as

$$\begin{aligned}\check{\Re}_m = (\ ]_{j_1^{(m)}} X_{i_1^{(m)}} \ ... \ ]_{j_{l_1}^{(m)}} X_{i_{l_1}^{(m)}} (O_1)^{+_{m_1}}]_{j_{l_1+1}^{(m)}} X_{i_{l_1+1}^{(m)}} \ ... \ ]_{j_{l_1+l_2}^{(m)}} X_{i_{l_1+l_2}^{(m)}} (O_2)^{+_{m_2}} \ ... \\ ]_{j_{l_1+...+l_{p^{(m)}-1}+1}^{(m)}} X_{i_{l_1+...+l_{p^{(m)}-1}+1}^{(m)}} ...]_{j_{l_1+...+l_{p^{(m)}}}^{(m)}} X_{i_{l_1+...+l_{p^{(m)}}}^{(m)}} (O_{p^{(m)}})^{+_{m_{p^{(m)}}}} \\ ]_{j_{l_1+...+l_{p^{(m)}}+1}^{(m)}} X_{i_{l_1+...+l_{p^{(m)}}+1}^{(m)}} ...]_{j_{l_1+...+l_{p^{(m)}+1}}^{(m)}} X_{i_{l_1+...+l_{p^{(m)}+1}}^{(m)}})^{+_m} \qquad (14)\end{aligned}$$

in which each $O_i^{+_{m_i}}$ is a loop of height $+_{m_i}$, $1 \leq i \leq p^{(m)}$, $m = \max\{m_1, ..., m_{p^{(m)}}\}$, and has a similar structure as loop $\check{\Re}_m$. Analogously as for the loop (6), we can recursively continue to define each loop $O_i^{+_{m_i}}$, $1 \leq i \leq p^{(m)}$, until loops of $+_\hbar = 0$. For each loop $O_i^{+_{m_i}}$ we consider the segment $X_i^{+_{m_i}} = X_{i_1^{(m_i)}} ... X_{i_{l_1+...+l_{p^{(m_i)}}+1}^{(m_i)}}$, $1 \leq i \leq p^{(m)}$. For $\check{\Re}_m$ the above segment becomes $X^{+_m} = X_{i_1^{(m)}} ... X_{i_{l_1+...+l_{p^{(m)}}+1}^{(m)}}$.

Each $X_i^{+_{m_i}}$ is composed of only brackets from $\overset{\triangleleft\,(2)}{N_r}$. It may be an empty or constant segment (a loop of $+_\hbar = 0$) or it may have a structure similar to the loop (6). To each such a segment we associate a coverability tree $\hat{\mathcal{T}}(X_i^{+_{m_i}})$, $1 \leq i \leq p^{(m)}$, as defined in Definition 11. To each loop of form (14) we can also associate a coverability tree, denoted $\check{\mathcal{T}}(\check{\Re}_m)$, similar to the coverability tree defined in Definition 11, or to the coverability tree associated with the loop (11). The parameters that characterize each node associated with each loop $O_i^{+_{m_i}}$ in $\check{\mathcal{T}}$, are the same as in the case of Definition 11, i.e., $+_\hbar = +_{m_i}$ and the length $l_1 + ... + l_{p^{(m_i)}}$, i.e., the number of brackets from $\overset{\triangleright\,(2)}{N_l}$ that composes $O_i^{+_{m_i}}$.

Since each segment $X_i^{+_{m_i}}$ contains no bracket from $\overset{\triangleright\,(2)}{N_l}$, the coverability tree[17] associated with $X_i^{+_{m_i}}$, i.e., $\hat{\mathcal{T}}(X_i^{+_{m_i}})$, does not contain any (coverability) subtree of type $\check{\mathcal{T}}$. Thus, each subtree of type $\hat{\mathcal{T}}$ can be considered as a "leaf" subtree of $\check{\mathcal{T}}$. Now we can state conditions of type (9) and (10) for a general loop of type (14). The loop (14) has a tree representation given by all paths from the root v_0 to each leaf of the form

$$\left.\begin{aligned}&\check{\wp}_1 = \check{v}_0, \check{v}_{1,1}, \check{v}_{1,2}, ..., \check{v}_{1,\check{f}_1-1}, \hat{t}_{1,\check{f}_1}, \\ &\check{\wp}_2 = \check{v}_0, \check{v}_{2,1}, \check{v}_{2,2}, ..., \check{v}_{2,\check{f}_2-1}, \hat{t}_{2,\check{f}_2}, \ ..., \\ &\check{\wp}_{\check{m}} = \check{v}_0, \check{v}_{\check{m},1}, \check{v}_{\check{m},2}, ..., \check{v}_{\check{m},\check{f}_{\check{m}}-1}, \hat{t}_{\check{m},\check{f}_{\check{m}}},\end{aligned}\right\} \qquad (15)$$

in which each node $\check{v}_{i,j}$, $1 \leq j \leq \check{f}_i - 1$, $1 \leq i \leq \check{m}$, is characterized by the parameters $(+_j, \check{l}_{i,j})$, where j is the $+_\hbar$ of the loop associated with the node $\check{v}_{i,j}$ (that gives the ranking place in that path) and $\check{l}_{i,j}$ is the length of the loop.

[17] If a segment of type $X_i^{+_{m_i}}$ is empty, then $\hat{\mathcal{T}}(X_i^{+_{m_i}})$ will be an empty tree, too.

Each node $\hat{t}_{i,\check{f}_i}$, $1 \leq i \leq \check{m}$, stands for the root of a coverability tree of type $\hat{\mathcal{T}}_{\check{f}_i}$, which may also be an empty tree.

For the whole tree $\check{\mathcal{T}}(\check{\Re}_m)$, $\mathcal{A}$ must guess at most[18] $\check{f}_1+...+\check{f}_{\check{m}}-\check{m}$ arbitrarily large numbers of integers corresponding to nodes[19] of type $\check{v}_{i,j}$, i.e., for each such node $\mathcal{A}$ guesses an arbitrarily large integer $\check{s}^{i,j}$, $1 \leq j \leq \check{f}_i - 1$, $1 \leq i \leq \check{m}$. For each path $\check{\wp}_i$, $1 \leq i \leq \check{m}$, $\mathcal{A}$ checks whether the next inequalities hold:

$$\hat{s}^{i,\check{f}_i} \geq \check{s}^{i,\check{f}_i-1} \geq ... \geq \check{s}^{i,2} \geq \check{s}^{i,1} \geq \check{s}^0 \geq 1 \tag{16}$$

where $\check{s}^0$ is the arbitrarily large value associated with the root $\check{v}^0$, and $\hat{s}^{i,\check{f}_i}$, $1 \leq i \leq \check{m}$, is the arbitrarily large value associated with the root of each "leaf" tree of type $\hat{\mathcal{T}}_{\check{f}_i}$. The length of the string generated by the loop of type (14) on the left side of the core index, is given by

$$\bigoplus_{i=1}^{\check{m}} (\check{s}^0 \check{l}_0 + \check{s}^{i,1}\check{l}_{i,1} + \check{s}^{i,2}\check{l}_{i,2} + ... + \check{s}^{i,\check{f}_i-1}\check{l}_{i,\check{f}_i-1}). \tag{17}$$

Each tree of type $\hat{\mathcal{T}}_{\check{f}_i}$, rooted by $\hat{t}_{i,\check{f}_i}$, $1 \leq i \leq \check{m}$, has a description in terms of terminal paths given by a system of type (8) of the form

$$\left.\begin{array}{l} \hat{\wp}_1^{\check{f}_i} = \hat{t}_{i,\check{f}_i}, \hat{v}_{1,1}^{\check{f}_i}, \hat{v}_{1,2}^{\check{f}_i}, ..., \hat{v}_{1,\hat{f}_1^{(\check{f}_i)}}^{\check{f}_i}, \\ \hat{\wp}_2^{\check{f}_i} = \hat{t}_{i,\check{f}_i}, \hat{v}_{2,1}^{\check{f}_i}, \hat{v}_{2,2}^{\check{f}_i}, ..., \hat{v}_{2,\hat{f}_2^{(\check{f}_i)}}^{\check{f}_i}, \ ..., \\ \hat{\wp}_{\hat{m}_i}^{\check{f}_i} = \hat{t}_{i,\check{f}_i}, \hat{v}_{\hat{m}_i,1}^{\check{f}_i}, \hat{v}_{\hat{m}_i,2}^{\check{f}_i}, ..., \hat{v}_{\hat{m}_i,\hat{f}_{\hat{m}_i}^{(\check{f}_i)}}^{\check{f}_i}. \end{array}\right\} \tag{18}$$

Therefore, for each "leaf" subtree of type $\hat{\mathcal{T}}_{\check{f}_i}$[20], characterized by (18), $\mathcal{A}$ must guess other $\hat{f}_1^{(\check{f}_i)}+...+\hat{f}_{\hat{m}_i}^{(\check{f}_i)}$, $1 \leq i \leq \check{m}$, finite numbers, of arbitrarily large integers that satisfy the conditions:

$$\hat{s}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}} \geq \hat{s}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}-1} \geq ... \geq \hat{s}_{\check{f}_i}^{j,2} \geq \hat{s}_{\check{f}_i}^{j,1} \geq \hat{s}^{i,\check{f}_i} \tag{19}$$

where $\hat{s}^{i,\check{f}_i}$ is the arbitrarily large value associated with the root $\hat{t}_{i,\check{f}_i}$, $1 \leq j \leq \hat{m}_i$, and $1 \leq i \leq \check{m}$. The length of the string generated by the loop (14) at the right side of the the core index, is given by

$$\bigoplus_{i=1}^{\check{m}} \bigoplus_{j=1}^{\hat{m}_i} (\hat{s}^{i,\check{f}_i} \cdot \hat{l}^{i,\check{f}_i} + \hat{s}_{\check{f}_i}^{j,1} \cdot \hat{l}_{\check{f}_i}^{j,1} + \hat{s}_{\check{f}_i}^{j,2} \cdot \hat{l}_{\check{f}_i}^{j,2} + ... + \hat{s}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}} \cdot \hat{l}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}}). \tag{20}$$

For each loop of type (14) we build the following left and right regular languages

[18] For each common node, belonging to different paths, $\mathcal{A}$ guesses only a single value.

[19] The plus-height of (14), without considering the plus-height of each $X_i^{+_{m_i}}$ segment, is $m = \max\{\check{f}_1, ..., \check{f}_{\check{m}}\}$.

[20] The height of $\hat{\mathcal{T}}_{\check{f}_i}$ is equal with the plus-height of the segment $X_i^{+_{m_i}}$, i.e., $+_{m_i} = \max\{\hat{f}_1^{(\check{f}_i)}, ..., \hat{f}_{\hat{m}_i}^{(\check{f}_i)}\}$, $1 \leq i \leq \check{m}$.

$\mathcal{L}_l{}^{c_\psi^{+(m)}}(\check{\Re}_m) = \psi_l(\Psi_l(\eta_{\lambda_l}(\check{\Re}_m)))$ and $\mathcal{L}_r{}^{c_\psi^{+(m)}}(\check{\Re}_m) = \psi_r((\Psi_r(\eta_{\lambda_r}(\check{\Re}_m)))^r))$
where η_{λ_l}, η_{λ_r}, ψ_l, ψ_r, Ψ_l, and Ψ_r are homomorphisms defined as

$\eta_{\lambda_l}\colon (\overset{\triangleright\,(2)}{N_l} \cup \overset{\triangleleft\,(2)}{N_r} \cup\{S\})^* \to (\overset{\triangleright\,(2)}{N_l})^*$, $\eta_{\lambda_l}(S) = \lambda$, $\eta_{\lambda_l}([_i) = \lambda$, $\eta_{\lambda_l}(]_j) =]_j$, for $([_i,]_i) \in N_r^{(2)}$ and $([_j,]_j) \in N_l^{(2)}$,
$\eta_{\lambda_r}\colon (\overset{\triangleright\,(2)}{N_l} \cup \overset{\triangleleft\,(2)}{N_r} \cup\{S\})^* \to (\overset{\triangleleft\,(2)}{N_r})^*$, $\eta_{\lambda_r}(S) = \lambda$, $\eta_{\lambda_r}(]_j) = \lambda$, $\eta_{\lambda_r}([_i) = [_i$, for $([_i,]_i) \in N_r^{(2)}$ and $([_j,]_j) \in N_l^{(2)}$;
$\Psi_l : (\overset{\triangleright\,(2)}{N_l})^* \to (\overset{\triangleleft\,(2)}{N_l})^*$, with $\Psi_l(]_j) = [_j$, and $\Psi_r : (\overset{\triangleleft\,(2)}{N_r})^* \to (\overset{\triangleright\,(2)}{N_r})^*$, $\Psi_r([_i) =]_i$;
$\psi_l : (\overset{\triangleleft\,(2)}{N_l})^* \to T^*$, $\psi_l([_j) = t$, for each rule of the form $[_j \to t$, and
$\psi_r : (\overset{\triangleright\,(2)}{N_r})^* \to T^*$, $\psi_r(]_i) = t$, for each rule of the form $]_i \to t$.

Consider now the particular case of a LCFG in linear-Dyck normal form for which the dependency graph $\mathcal{G}$ has only one class of terminal paths represented by a regular expression that starts with S (the axiom), ends with t (a terminal node), and in between is composed of a loop of type (14). Let $\check{\mathcal{T}}$ be the coverability tree described by terminal paths given in (15) and (18), with the conditions (16) and (19) upon its nodes. Having the above information recorded, by using constant space, on the work tape of $\mathcal{A}$, Algorithm 2 can be generalized as follows.

Algorithm 3 Let $w \in T^*$ be an input word of length n of the form $a_1a_2...a_n$. First $\mathcal{A}$ existentially proceeds similar as in Level 1 and Level 2 in Algorithm 1. Then, $\mathcal{A}$ existentially guesses at most $\check{f}_1 + ... + \check{f}_{\check{m}} - \check{m}$ arbitrarily large numbers of integers corresponding to the sum of plus-height values associated with nodes in the coverability tree $\check{\mathcal{T}}(\check{\Re}_m)$ described by (15), and at most $\sum_{i=1}^{\check{m}}(\hat{f}_1^{(\check{f}_i)} + ... + \hat{f}_{\hat{m}_i}^{(\check{f}_i)})$ arbitrarily large integers corresponding to the sum of plus-height values associated with nodes in each "leaf" coverability tree of type $\hat{\mathcal{T}}$ described by (18). Then $\mathcal{A}$ computes

1. $\ell_l = \bigoplus_{i=1}^{\check{m}}(\check{s}^0\check{l}_0 + \check{s}^{i,1}\check{l}_{i,1} + \check{s}^{i,2}\check{l}_{i,2} + ... + \check{s}^{i,\check{f}_i-1}\check{l}_{i,\check{f}_i-1})$, checks whether $n_t - 1 = \ell_l$, and whether the prefix of length ℓ_l of w, i.e., $a_1...a_{\ell_l} = a_1...a_{n_t-1}$ belongs to the regular set $\mathcal{L}_l{}^{c_\psi^{+(m)}}(\check{\Re}_m)$;
2. $\ell_r = \bigoplus_{i=1}^{\check{m}}\bigoplus_{j=1}^{\hat{m}_i}(\hat{s}^{i,\check{f}_i} \cdot \hat{l}^{i,\check{f}_i} + \hat{s}_{\check{f}_i}^{j,1} \cdot \hat{l}_{\check{f}_i}^{j,1} + \hat{s}_{\check{f}_i}^{j,2} \cdot \hat{l}_{\check{f}_i}^{j,2} + ... + \hat{s}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}} \cdot \hat{l}_{\check{f}_i}^{j,\hat{f}_j^{(\check{f}_i)}})$, checks whether $n - n_t - 1 = \ell_r$, and whether the suffix of length ℓ_r of w, i.e., $a_{n-\ell_r+1}...a_n = a_{n_t+2}...a_n$ belongs to the regular $\mathcal{L}_r{}^{c_\psi^{+(m)}}(\check{\Re}_m)$. $\diamond$

Notice that, since both sums $\check{f}_1 + ... + \check{f}_{\check{m}} - \check{m}$ and $\sum_{i=1}^{\check{m}}(\hat{f}_1^{(\check{f}_i)} + ... + \hat{f}_{\hat{m}_i}^{(\check{f}_i)})$ are finite, there will always be $\mathcal{O}(n^{s_1+s_2})$ existential branches corresponding to the values of all plus-heights interfering in loop (14), where $s_1 = \mathcal{O}(\check{f}_1 + ... + \check{f}_{\check{m}} - \check{m})$, and $s_2 = \mathcal{O}(\sum_{i=1}^{\check{m}}(\hat{f}_1^{(\check{f}_i)} + ... + \hat{f}_{m_i}^{(\check{f}_i)}))$. Therefore, the computation tree associated with $\mathcal{A}$ can be converted into a binary tree of height at most $\mathcal{O}((s_1 + s_2)\log n)$. $\surd$

The observations concerning the correctness of Algorithm 3 are the same as for Algorithm 2. The procedure described in Algorithm 3 is universal concerning the manner we deal with multilevel loops induced by brackets from $\overset{\triangleright\,(2)}{N_l}$. Now we are ready to prove the main result. $\surd$

Theorem 4. *Each language $L \in LIN$ can be recognized by an indexing ATM in $\mathcal{O}(\log n)$ time and space.*

Proof. Let $G = (N, T, P, S)$ be an arbitrary LCFG, and $G_k = (N_k, T, P_k, S)$ be the linear-Dyck normal form of G, with $N_k = \{S, [_1, [_2, ..., [_k, \ ]_1,]_2, ..., \ ]_k\}$. Let $L(G_k)$ be the language generated by G_k, and $\mathcal{G}$ the dependency graph of G_k. Without loss of generality we assume that G does not contain rules of the from $S \to t$, $t \in T$. Otherwise, by using a similar procedure as described in Theorem 3, the sets N_k and P_k are enlarged to N_{k+p} and P_{k+p} (defined in Theorem 3), respectively. Then it is trivial to check whether words of length one belong to the language generated by the new grammar G_{k+p}.

Let $\mathcal{A}$ be an indexing ATM composed of an input tape that stores an input word, $w \in T^*$ of an arbitrary length n, an index tape to guess input symbols, and one working tape divided into four tracks, used to record the positions of the input symbols and several other values used during the computation. These numbers are stored on the work tapes in binary. At the beginning of the computation the tracks of the work tape are empty.

Briefly, $\mathcal{A}$ first guesses the core index. Then depending on the structure of $\mathcal{G}$, $\mathcal{A}$ performs several guessing and checking procedures for each "constant" segment or multilevel loop induced by brackets from $\overset{\triangleright\,(2)}{N_l}$, similar to Algorithm 3. Denote by $\mathcal{P}$ the set of all classes of terminal paths, in terms of + multilevel loops induced by brackets from $\overset{\triangleright\,(2)}{N_l}$, found in the dependency graph of G_k. As explained, with each element of $\mathcal{P}$ we associate a regular expression in terms of + Kleene operation. Denote the set of all regular expressions, in string form, readable from $\mathcal{G}$ by $\mathcal{R}_\mathcal{G}$. Then the cardinality of $\mathcal{P}$ and $\mathcal{R}_\mathcal{G}$ are equal and finite, i.e., $|\mathcal{P}| = |\mathcal{R}_\mathcal{G}|$.

Algorithm 4 (*The General Algorithm*) Let $w \in T^*$, $w = a_1 a_2 ... a_n$, be an input word of length n stored on the input tape.

Level 1 (*Existential*) In an *existential* state $\mathcal{A}$ guesses the length of w and verifies the correctness of this guess, as in Level 1 of Algorithm 1. The correct value of n is recorded in binary on the first track of the work tape. *This procedure requires $\mathcal{O}(\log n)$ time and space.* $\triangle$

Level 2 (*Existential*) Using *existential* states $\mathcal{A}$ branches all i between 1 and n, and tries to localize the position of the core index, i.e., the index n_t such that there exists $([_j^t, \ ]_j^t) \in N^{(1)}$ with $[_j^t \to a_{n_t}$, and $]_j^t \to a_{n_t+1} \in P_k$. Once the core segment $a_{n_t} a_{n_t+1}$, $1 \leq n_t \leq n-1$, is localized, the computation continues on that existential branch with Level 3. The value n_t is stored in binary on the second track of the work tape. Then $\mathcal{A}$ computes $n - n_t - 1$ and $\log(n - n_t - 1)$

space is allocated for the third track. This allocation is used to control the limits of computation, i.e., no positive integer greater than $n-n_t-1$, written in binary, can be stored on this track. As explained in Level 2 of Algorithm 1, *the cost of the computation at this level is of* $\mathcal{O}(\log n)$ *time and space*, too. $\triangle$

Level 3 (*Existential*$_1$ | *Existential*$_2$ | *Universal*$_1$)

Let $|\mathcal{R}_\mathcal{G}|$ be the cardinality of $\mathcal{R}_\mathcal{G}$. Recall that each element in $\mathcal{R}_\mathcal{G}$, is uniquely associated with a single class of terminal paths in $\mathcal{P}$. Then $\mathcal{A}$ spawns $|\mathcal{R}_\mathcal{G}|$ = $|\mathcal{P}|$ *existential*$_1$ branches, each branch holding on the "structure" of one of the regular expression in $\mathcal{R}_\mathcal{G}$. With respect to Definition 10, and our bracketed considerations, a general element $\Re \in \mathcal{R}_\mathcal{G}$, can be of the form

$$\Re = Sz_1c_1z_2o_1^+z_3c_2z_4o_2^+...z_{2\xi_\Re-1}c_{\xi_\Re}z_{2\xi_\Re}o_{\xi_\Re}^+z_{2\xi_\Re+1}c_{\xi_\Re+1}t.$$

In this structure, each segment z_α, $1 \leq \alpha \leq 2\xi_\Re + 1$, contains only brackets from $\overset{\triangleleft\,(2)}{N}_r$, "organized" either into constant segments, or into multilevel + loops. The segments c_β, $1 \leq \beta \leq \xi_\Re + 1$, are "constant" segments in terms of brackets from $\overset{\triangleright\,(2)}{N}_l$ studied at Case 1, and o_δ^+, $1 \leq \delta \leq \xi_\Re$, are "universal" loops studied at Case 2.

Each segment $c_\beta, 1 \leq \beta \leq \xi_\Re+1$, is of the form $c_\beta =]_{c_1^{(\beta)}} X_1^{(\beta)} ...]_{c_{s_\beta-1}^{(\beta)}} X_{s_\beta-1}^{(\beta)}]_{c_{s_\beta}^{(\beta)}}$, in which each $X_i^{(\beta)}$, $1 \leq i \leq s_\beta - 1$, are segments induced by brackets from $\overset{\triangleleft\,(2)}{N}_r$. Let $]_l^{c_\beta}$ and $]_r^{c_\beta}$ be the leftmost and rightmost right-brackets from c_β, i.e., $]_l^{c_\beta} =]_{c_1^{(\beta)}}$ and $]_r^{c_\beta} =]_{c_{s_\beta}^{(\beta)}}$. Subwords, placed at the right-side of the core segment $a_{n_t}a_{n_t+1}$, generated by brackets from $\overset{\triangleright\,(2)}{N}_r$, such that their pairwise from $\overset{\triangleleft\,(2)}{N}_r$, occur in the dependency graph $\mathcal{G}$, inside any "empty" or "constant" segment z_α, $1 \leq \alpha \leq 2\xi_\Re+1$, or c_β, $1 \leq \beta \leq \xi_\Re+1$, are words belonging to regular languages (already built in Case 1) of type $'\mathcal{L}^{\psi_r}{}'$ and $\mathcal{L}^{c^{\psi_r}}([_l^{c_\beta}...[_r^{c_\beta})$, $1 \leq \beta \leq \xi_\Re + 1$.

Each o_δ^+ segment, $1 \leq \delta \leq \xi_\Re$, denotes an one-level or multilevel loop, determined by brackets from $\overset{\triangleright\,(2)}{N}_l$. Let us consider $]_l^{o_\delta}$ and $]_r^{o_\delta}$, the leftmost and rightmost brackets from o_δ^+, respectively. Subwords, placed at the right-side of the core segment $a_{n_t}a_{n_t+1}$, generated by brackets from $\overset{\triangleleft\,(2)}{N}_r$ occurring in the loop o_δ^+, $1 \leq \delta \leq \xi_\Re$, are words belonging to regular languages of type $\mathcal{L}_r{}^{c_\psi^{+(m)}}(o_\delta^+)$ (already built in Case 2.2). Subwords, placed at the left-side of the core segment $a_{n_t}a_{n_t+1}$, generated by brackets from $\overset{\triangleright\,(2)}{N}_l$ occurring in the loop o_δ^+, $1 \leq \delta \leq \xi_\Re$, are words belonging to regular languages of type $\mathcal{L}_l{}^{c_\psi^{+(m)}}(o_\delta^+)$.

For each *existential*$_1$ branch, holding on a regular expression of type $\Re$, $\mathcal{A}$ proceeds as follows.

For the combination of "empty" and "constant" segments in terms of brackets from $\overset{\triangleright\,(2)}{N}_l$, occurring in $\Re$, $\mathcal{A}$ *existentially*$_2$ guesses $3\xi_\Re + 3$ positive integers $\dot{k}_1$, ..., $\dot{k}_{2\xi_\Re+1}$, $\bar{k}_1$, ..., $\bar{k}_{\xi_\Re+1}$, and k_t, where each $\dot{k}_\alpha$, $1 \leq \alpha \leq 2\xi_\Re + 1$, stands for the length of z_α, and each $\bar{k}_\beta$, $1 \leq \beta \leq \xi_\Re + 1$, stands for the length of all sequences of segments composed of brackets from $\overset{\triangleleft\,(2)}{N}_r$, generated during the execution of

c_β, i.e., $\bar{k}_\beta = |X_1^{(\beta)}...X_{s_\beta-1}^{(\beta)}|$. The last value k_t stands for the number of brackets from $\overset{\triangleleft\ (2)}{N_r}$ generated between the rightmost bracket from $c_{\xi_\Re+1}$ and t, the label of the final vertex from the considered terminal path.

For each loop o_δ^+, $1 \leq \delta \leq \xi_\Re$, as explained in Algorithm 3, $\mathcal{A}$ *existentially*$_2$ guesses at most $\check{f}_1^{(\delta)} + ... + \check{f}_{\check{m}^{(\delta)}}^{(\delta)} - \check{m}^{(\delta)}$, and at most $\hat{f}_1^{(\check{f}_i^{(\delta)})} + ... + \hat{f}_{\hat{m}_i^{(\delta)}}^{(\check{f}_i^{(\delta)})}$, $1 \leq i \leq \check{m}^{(\delta)}$, arbitrarily large numbers of integers corresponding to "accumulative sums" associated with nodes of type $\check{v}_{i,j}^{\delta}$, $1 \leq j \leq \check{f}_i^{(\delta)}$, $1 \leq i \leq \check{m}^{(\delta)}$, defining trees of type $\check{\mathcal{T}}^\delta(o_\delta^+)$ in (15), and with nodes of type $\hat{v}_{i,j}^{\check{f}_i^{(\delta)}}$, $1 \leq j \leq \hat{f}_{\hat{m}_i^{(\delta)}}^{(\check{f}_i^{(\delta)})}$, $1 \leq i \leq \hat{m}_i^{(\delta)}$, defining "leaf" trees of type $\hat{\mathcal{T}}_{\check{f}_i}^{\delta}$ in (18), respectively. For each path $\check{\wp}_i^\delta$, $1 \leq i \leq \check{m}^{(\delta)}$, from (15) $\mathcal{A}$ checks whether inequalities of types (16) hold, and computes a sum of type (17)

$$\ell_l^{(\delta)} = \bigoplus_{i=1}^{\check{m}^{(\delta)}} (\check{s}_\delta^0 \check{l}_0^{(\delta)} + \check{s}_\delta^{i,1} \check{l}_{i,1}^{(\delta)} + \check{s}_\delta^{i,2} \check{l}_{i,2}^{(\delta)} + ... + \check{s}_\delta^{i,\check{f}_i-1} \check{l}_{i,\check{f}_i-1}^{(\delta)}). \tag{21}$$

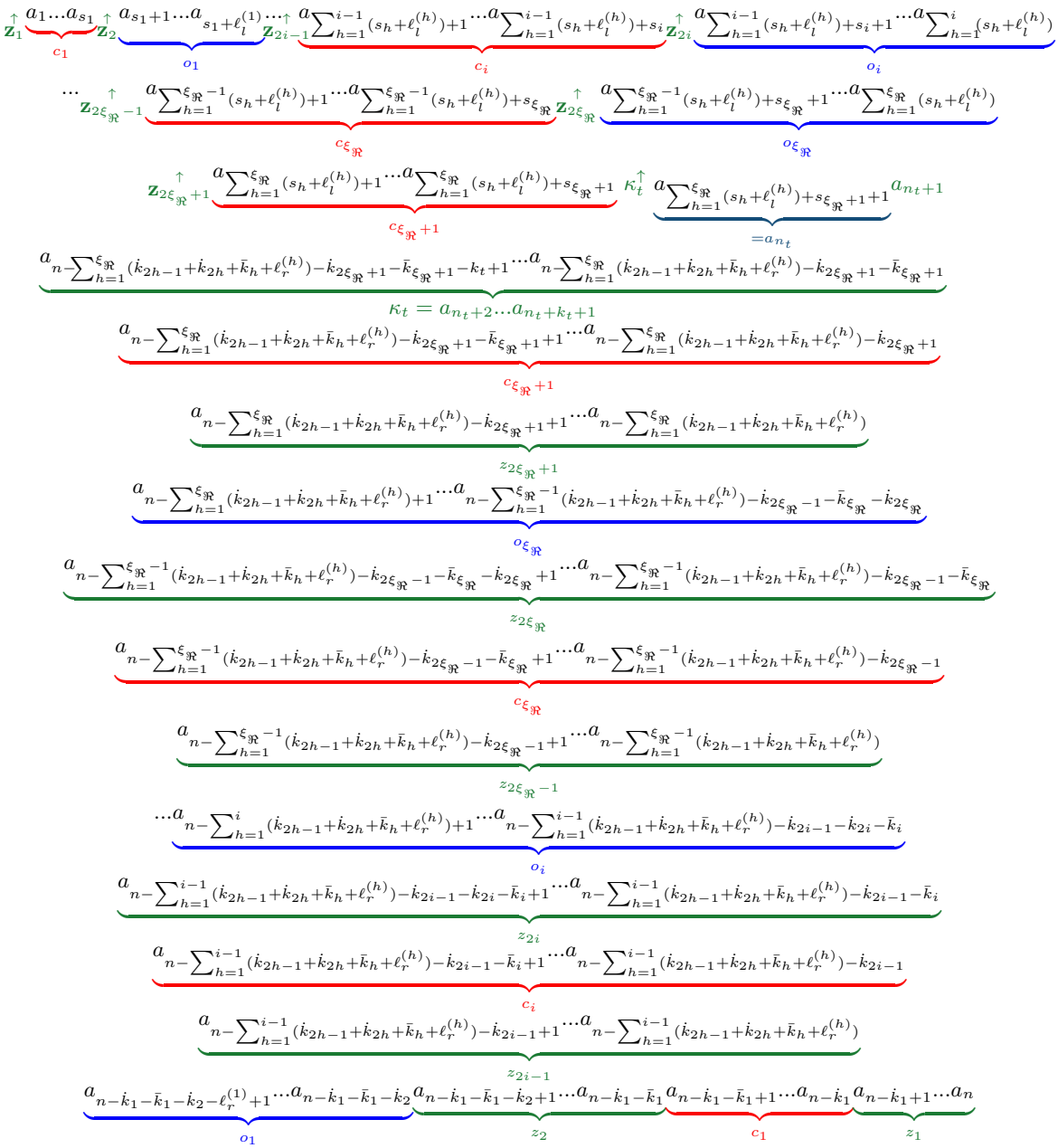
For each path $\hat{\wp}_i^{\check{f}_i^{(\delta)}}$, $1 \leq i \leq \hat{m}_i^{(\delta)}$, from (18) $\mathcal{A}$ checks whether inequalities of types (19) hold, and computes a sum of type (20). More precisely, $\mathcal{A}$ computes

$$\ell_r^{(\delta)} = \bigoplus_{i=1}^{\check{m}^{(\delta)}} \bigoplus_{j=1}^{\hat{m}_i^{(\delta)}} (\hat{s}^{i,\check{f}_i^{(\delta)}} \cdot \hat{l}^{i,\check{f}_i^{(\delta)}} + \hat{s}_{\check{f}_i^{(\delta)}}^{j,1} \cdot \hat{l}_{\check{f}_i^{(\delta)}}^{j,1} + \hat{s}_{\check{f}_i^{(\delta)}}^{j,2} \cdot \hat{l}_{\check{f}_i^{(\delta)}}^{j,2} + ... + \hat{s}_{\check{f}_i^{(\delta)}}^{j,\hat{f}_j^{\check{f}_i^{(\delta)}}} \cdot \hat{l}_{\check{f}_i^{(\delta)}}^{j,\hat{f}_j^{\check{f}_i^{(\delta)}}}). \tag{22}$$

Each $5\xi_\Re+3$-tuple of positive and arbitrarily large integers guessed within the *existential*$_2$ branch, i.e., $\dot{k}_\alpha$, $1 \leq \alpha \leq 2\xi_\Re+1$, $\bar{k}_\beta$, $1 \leq \beta \leq \xi_\Re+1$, k_t, $\ell_l^{(\delta)}$ and $\ell_r^{(\delta)}$, $1 \leq \delta \leq \xi_\Re$, the last two δ-tuples computed with respect to (21) and (22), can be stored, checked, and computed into the fourth track of the alternating machine. Each of them must satisfy the conditions $0 \leq \dot{k}_\alpha \leq n - n_t - 1$, $1 \leq \alpha \leq 2\xi_\Re + 1$, $0 \leq \bar{k}_\beta \leq n - n_t - 1$, $1 \leq \beta \leq \xi_\Re + 1$, $0 \leq k_t \leq n - n_t - 1$, $0 \leq \ell_l^{(\delta)} \leq n_t - 1$, $0 \leq \ell_r^{(\delta)} \leq n - n_t - 1$, $1 \leq \delta \leq \xi_\Re$, $\sum_{i=1}^{2\xi_\Re+1} \dot{k}_i + \sum_{i=1}^{\xi_\Re+1} \bar{k}_i + \sum_{i=1}^{\xi_\Re} \ell_r^{(i)} + k_t = n - n_t - 1$, and $\sum_{i=1}^{\xi_\Re} (\ell_l^{(i)} + s_i) + s_{\xi_\Re+1} = n_t - 1$.

Since we deal with operations that concern a finite number of arbitrarily large integers of $\mathcal{O}(\log n)$ bits, all the procedures to store and check inequalities of type (16) and (19), or compute sums of type (21) and (22), are elementary functions belonging to NC^1. Therefore, the time and space needed to check the above inequalities, and to compute in binary the sums described above, are not more than $\mathcal{O}(\log n)$ (using for instance the third track of a length $log(n-n_t-1)$ as a space-clock). Furthermore, due to the finite dimension of the tuples, of

arbitrarily large positive integers, $\mathcal{A}$ has to guess, the height of the computation tree associated with $\mathcal{A}$ is still of $\mathcal{O}(\log n)$.



Segments marked by the same color and the same index, are subwords generated within the same partition in $\mathcal{G}$, i.e., loops, "empty", and "constant" segments in terms of elements from $N_l^{\triangleright(2)}$ and $N_r^{\triangleleft(2)}$. They are universally branched and checked according to items 1, 2, and 3. Figure 1.

For each *existential*$_2$ branch holding on an $5\xi_{\Re}+3$-tuple, $\mathcal{A}$ *universally*$_1$ branches substrings of the input string corresponding to the c_i, and o_i segments (generated by brackets from $\overset{\triangleleft\ (2)}{N_l}$) placed at the left-side of the core segment $a_{n_t}a_{n_t+1}$ with substrings of the input string corresponding to the segments z_{2i-1}, c_i, z_{2i}, o_i, $1 \leq i \leq \xi_{\Re}+1$, (generated by brackets from $\overset{\triangleright\ (2)}{N_r}$) placed at the right-side of the core segment as emphasized in Figure 1. Each universal branch checks whether the guessed integer, corresponding to the type of the enclosed segment, can indeed be the length of a substring placed at the left or right side of the core segment. More precisely, in parallel, for each i, $1 \leq i \leq \xi_{\Re}+1$, $\mathcal{A}$ *universally*$_1$ proceeds with the checking procedures described in 1), 2), and 3):

1) For $i=1$, $\mathcal{A}$ checks whether

- there exists $[_l^{c_1} \rightarrow a_1 \in P_k$ and whether S or the suffix of w of length $\dot{k}_1$, $a_{n-\dot{k}_1+1}...a_{n-1}a_n$ belongs to $\mathcal{L}^{\psi_r}([_l^{c_1})$,
- there exists $[_l^{c_1} \rightarrow a_1 \in P_k$, $[_r^{c_1} \rightarrow a_{s_1} \in P_k$, and $(a_1...a_{s_1}, a_{n-\dot{k}_1-\bar{k}_1+1}...a_{n-\dot{k}_1})$ $\in \mathcal{L}^{c^{\psi_r}}([_l^{c_1}...[_r^{c_1})$,
- there exists $[_r^{c_1} \rightarrow a_{s_1} \in P_k$, $[_l^{o_1} \rightarrow a_{s_1+1} \in P_k$, and whether λ or the subword of w of length $\dot{k}_2$, $a_{n-\dot{k}_1-\bar{k}_1-\dot{k}_2+1}...a_{n-\dot{k}_1-\bar{k}_1}$ belongs to $\mathcal{L}^{\psi_r}([_r^{c_1}[_l^{o_1})$,
- there exists $[_l^{o_1} \rightarrow a_{s_1+1} \in P_k$, $[_r^{o_1} \rightarrow a_{s_1+\ell_l^{(1)}} \in P_k$, $a_{s_1+1}...a_{s_1+\ell_l^{(1)}} \in \mathcal{L}^{c^{\psi_r}}_l([_l^{o_1}[_r^{o_1})$ and $a_{n-\dot{k}_1-\bar{k}_1-\dot{k}_2-\ell_r^{(1)}+1}...a_{n-\dot{k}_1-\bar{k}_1-\dot{k}_2} \in \mathcal{L}^{c^{\psi_r}}_r([_l^{o_1}[_r^{o_1})$.

2) For each i, $2 \leq i \leq \xi_{\Re}$, $\mathcal{A}$ checks whether

- there exist $[_r^{o_{i-1}} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})} \in P_k$, $[_l^{c_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+1} \in P_k$, and whether λ or the subword of w of length $\dot{k}_{2i-1}$, $a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2i-1}+1}...$ $a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})}$ belongs to $\mathcal{L}^{\psi_r}([_r^{o_{i-1}}[_l^{c_i})$,
- there exist $[_l^{c_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+1} \in P_k$, and $[_r^{c_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i} \in P_k$, $a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+1}...a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i} \in \mathcal{L}^{c^{\psi_r}}_l([_l^{c_i}...[_r^{c_i})$, and a subword of length $\bar{k}_i$, $a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2i-1}-\bar{k}_i+1}...a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2i-1}}$ $\in \mathcal{L}^{c^{\psi_r}}_r([_l^{c_i}...[_r^{c_i})$,
- there exist $[_r^{c_{s_i}} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i} \in P_k$, $[_l^{o_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i+1} \in P_k$, and whether λ or the subword of w, $a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2i-1}-\bar{k}_i-\dot{k}_{2i}+1}$ $...a_{n-\sum_{h=1}^{i-1}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2i-1}-\bar{k}_i} \in \mathcal{L}^{\psi_r}([_r^{c_i}[_l^{o_i})$,

– there exist $\lceil_l^{o_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i+1} \in P_k$, $\lceil_r^{o_i} \rightarrow a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i+\ell_l^{(i)}} \in P_k$,

$a_{n-\sum_{h=1}^{i}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})+1} \ldots a_{n-\sum_{h=1}^{i}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})+\ell_r^{(i)}} \in \mathcal{L}_r^{\psi_r}(\lceil_l^{o_i} \lceil_r^{o_i})$,

and $a_{\sum_{h=1}^{i-1}(s_h+\ell_l^{(h)})+s_i+1} \ldots a_{\sum_{h=1}^{i}(s_h+\ell_l^{(h)})} \in \mathcal{L}_l^{\psi_r}(\lceil_l^{o_i} \lceil_r^{o_i})$.

3) For $i = \xi_{\Re}+1$, $\mathcal{A}$ checks whether

– there exist $\lceil_r^{o_{\xi_{\Re}}} \rightarrow a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})} \in P_k$, $\lceil_l^{c_{\xi_{\Re}+1}} \rightarrow a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+1} \in P_k$, and whether λ or the subword of length $\dot{k}_{2\xi_{\Re}+1}$ of w, i.e.,
$a_{n-\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2\xi_{\Re}+1}+1} \ldots a_{n-\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h+1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})}$ belongs to $\mathcal{L}^{\psi_r}(\lceil_r^{o_{\xi_{\Re}}} \lceil_l^{c_{\xi_{\Re}+1}})$,

– there exist $\lceil_l^{c_{\xi_{\Re}+1}} \rightarrow a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+1} \in P_k$, $\lceil_r^{c_{\xi_{\Re}+1}} \rightarrow a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+s_{\xi_{\Re}+1}} \in P_k$,

such that $a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+1} \ldots a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+s_{\xi_{\Re}+1}} \in \mathcal{L}c_l^{\psi_r}(\lceil_l^{c_{\xi_{\Re}+1}} \lceil_r^{c_{\xi_{\Re}+1}})$ and

$a_{n\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2\xi_{\Re}+1}-\bar{k}_{\xi_{\Re}+1}+1} \ldots a_{n\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2\xi_{\Re}+1}}$
$\in \mathcal{L}c_r^{\psi_r}(\lceil_l^{c_{\xi_{\Re}+1}} \lceil_r^{c_{\xi_{\Re}+1}})$,

– there exist $\lceil_r^{c_{\xi_{\Re}+1}} \rightarrow a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+s_{\xi_{\Re}+1}} \in P_k$, where $a_{\sum_{h=1}^{\xi_{\Re}}(s_h+\ell_l^{(h)})+s_{\xi_{\Re}+1}} = a_{n_t-1}$, $\lceil_{j_{n_t}}^{t} \rightarrow a_{n_t} \in P_k$, and $a_{n-\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2\xi_{\Re}+1}-\bar{k}_{\xi_{\Re}+1}-k_t+1}$
$\ldots a_{n-\sum_{h=1}^{\xi_{\Re}}(\dot{k}_{2h-1}+\dot{k}_{2h}+\bar{k}_h+\ell_r^{(h)})-\dot{k}_{2\xi_{\Re}+1}-\bar{k}_{\xi_{\Re}+1}} = a_{n_t+2} \ldots a_{n_t+k_t+1}$

The input $w \in T^*$ is accepted as belonging to $L(G_k)$, if at least one existential branch from the first existential fork is labeled by 1, which is actually 'decided' at the end of the computation. This means that all universal branches from the last universal fork are labeled by 1, i.e., the above conditions 1), 2), and 3) hold. Going up in the computation tree, there exists at least one $5\xi_{\Re}+3$-tuple, that satisfies the condition 1), 2), and 3), i.e., there exists at least one branch in the second existential fork labeled by 1. This implies that there exists at least one existential branch in the first existential fork holding on a regular expression, or a path in the dependency graph of the grammar G_k that generates w. △

◇

5 Conclusions

In this paper we have presented a new normal form for CFGs, called Dyck normal form, and a characterization of CFLs in terms of Dyck languages. Based on these two main results we have developed a "scenario" of an indexing alternating Turing machine that recognizes linear context-free languages in logarithmic time and space. Hence, linear languages are a proper subclass of NC^1 class. According to results from Sudborough [16] and [17], this implies the equality of L (deterministic log-space) and NL (non-deterministic log-space).

The presentation of the solution we have proposed for the log-space problem is very detailed, and follows, step by step, our progresses, in thinking and dealing with this problem, reached during the elaboration of this implementation.

6 Acknowledgments

The author is indebted to Professor Hendrik Jan Hoogeboom for a very first checking of the correctness of the normal form for context-free grammars introduced in this paper.

I would like to express my deep gratitudes to Professor Erkki Mäkinen for his patience in reading this paper, and for many technical and stylistic suggestions I received from him, and to Professor Laurenţiu Ferucio Ţiplea for precious and valuable discussions we had upon this topic.

I am also deeply indebted to Professor Juraj Hromkovič for technical comments and suggestions upon a previous version of this paper.

Part of the work reported here was possible due to a research stay at the University of Tampere, department of Computer Science, supported by the European Science Foundation (ESF) project "Automata: from Mathematics to Applications".

References

1. J.L. Balcázar, J. Díaz, J. Gabarró. Structural Complexity, Vol. II. *Springer-Verlag, Berlin-Heidelberg*, 1990.
2. A. Borodin. On Relating Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, **6** (4), 733–743, 1977.
3. A. Chandra, D. Kozen, L. Stockmeyer. Alternation. *Journal of Association for Computing Machinery*, **28** (1), 114–133, 1981.
4. R.S. Cohen. Techniques for Establishing Star Height of Regular Sets. *Mathematical Systems Theory*, vol. 5, 97–114, 1971.
5. C. Damm, M. Holzer, K.-J. Lange, P. Rossmanith. *Deterministic 0L Languages are of Very Low Complexity: D0L is in AC^0*. Developments in Language Theory, 305–313, 1993.
6. L.C. Eggan. Transition graphs and the star height of regular events. *Michigan Mathematical Journal*, **10** (4), 385-397, 1963.

7. A. Ehrenfeucht, H.J. Hoogeboom, G. Rozenberg. *Computations in Coordinated Pair Systems*, Tech. Rep. CU-CS-260-84, Dept. of Computer Science, University of Colorado at Boulder, 1984.
8. A. Ehrenfeucht, H.J. Hoogeboom, G. Rozenberg. *Coordinated Pair Systems; Part II: Sparse Structure of Dyck Words and Ogden's Lemma*, Tech. Rep. CU-CS-276-84, Dept. of Computer Science, University of Colorado at Boulder, 1984.
9. H. Gruber, M. Holzer. Finite Automata, Digraph Connectivity, and Regular Expression Size. *Proceedings of Automata, Languages and Programming, 35th International Colloquium, ICALP 2008*, Reykjavik, Iceland, LNCS 5126, 2008.
10. M.A. Harrison. Introduction to Formal Language Theory. *Addison-Wesley Publishing Company, Inc., Reading, Massachusetts*, 1978.
11. K. Hashiguchi. Algorithms for Determining Relative Star Height and Star Height. *Information and Computation*, **78**, 124–169, 1988.
12. O.H. Ibarra, T. Jiang, B. Ravikumar. Some Subclasses of Context-free Languages are in NC^1. *Journal of Information Processing Letters*, **29**, 111–117, 1988.
13. V.A. Nepomnyashchii. Spatial Complexity of Recognition of Context-Free Languages. *Cybernetics and Systems Analysis*, **11** (5), 736–741, 1975.
14. P. Linz. An Introduction to Formal Languages and Automata. *Jones and Bartlett Publishers, Inc., Sudbury, Massachusetts*, 2001.
15. W. Ruzzo. On Uniform Circuit Complexity. *Journal of Computer and System Sciences*, **22**, 365–383, 1981.
16. I.H. Sudborough. On Tape-Bounded Complexity Classes and Multi-Head Finite Automata. *Journal of Computer and System Sciences*, **10** (1), 62–76, 1975.
17. 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.
18. H. Vollmer. Introduction to Circuit Complexity A Uniform Approach. *Springer-Verlag, Berlin-Heidelberg*, 1999.
19. I. Wegener. The complexity of Boolean functions. *Wiley-Teubner Series in Computer Science, New York-Stuttgart*, 1987.