On LL($k$) linear conjunctive grammars

Ilya Olkhovsky†  Alexander Okhotin‡

December 16, 2021

Abstract

Linear conjunctive grammars are a family of formal grammars with an explicit conjunction operation allowed in the rules, which is notable for its computational equivalence to one-way real-time cellular automata, also known as trellis automata. This paper investigates the LL($k$) subclass of linear conjunctive grammars, defined by analogy with the classical LL($k$) grammars: these are grammars that admit top-down linear-time parsing with $k$-symbol lookahead. Two results are presented. First, every LL($k$) linear conjunctive grammar can be transformed to an LL(1) linear conjunctive grammar, and, accordingly, the hierarchy with respect to $k$ collapses. Secondly, a parser for these grammars that works in linear time and uses logarithmic space is constructed, showing that the family of LL($k$) linear conjunctive languages is contained in the complexity class $L$.

Keywords: Linear conjunctive grammars, LL($k$) grammars, parsing, logarithmic space.

1 Introduction

LL($k$) parsing is perhaps the best known linear-time parsing method. An LL($k$) parser reconstructs a parse tree of the input string top-down, as it reads the string from left to right. At each step, the parser selects a rule to apply to a nonterminal symbol, looking ahead by at most $k$ symbols.

The LL($k$) parsing is applicable to a subclass of formal grammars known as the LL($k$) grammars; the main theoretical properties of these grammars have been established in the papers of Knuth [9], Lewis and Stearns[11], and Rosenkrantz and Stearns[19]. In particular, Rosenkrantz and Stearns[19] and Kurki-Suonio[10] proved that, for each $k$, the LL($k$ + 1) grammars can define more languages than the LL($k$) grammars, leading to a strict hierarchy of LL($k$) languages by $k$.

A natural subclass of LL($k$) linear grammars, which obey the LL($k$) restriction and allow at most one nonterminal symbol on the right-hand side of any rule, was first studied by Ibarra et al. [6] and by Holzer and Lange[3], who have characterized the computational complexity of the languages defined by these grammars. The language-theoretic properties of linear LL($k$) languages were recently investigated by Jirásková and Klíma[8]. Lately, the authors[18] have demonstrated that in the case of LL($k$) linear grammars, the hierarchy by $k$ collapses, that is, every language defined by an LL($k$) linear grammar for some $k$ can be defined by an LL(1) linear

---

*This work was supported by the Ministry of Science and Higher Education of the Russian Federation, agreement 075-15-2019-1619.
†Department of Mathematics and Computer Science, 14th Line V.O., 29, Saint Petersburg 199178, Russia and Leonhard Euler International Mathematical Institute at St. Petersburg State University, Saint Petersburg, Russia. E-mail: ilianolhin@gmail.com.
‡Department of Mathematics and Computer Science, 14th Line V.O., 29, Saint Petersburg 199178, Russia. E-mail: alexander.okhotin@spbu.ru.
grammar. This transformation incurs an exponential blow-up in the size of the grammar, and, furthermore, it was proved that this blow-up is unavoidable in the worst case [18].

The idea of LL($k$) parsing is applicable to several generalizations of ordinary (“context-free”) formal grammars. One of such extensions are conjunctive grammars, introduced by Okhotin [12], which enrich the expressive power of ordinary grammars by allowing a conjunction operation in the rules; a rule $A \to \alpha \& \beta$ defines all strings that can be represented both as $\alpha$ and as $\beta$. The subclass of LL($k$) conjunctive grammars and the associated linear-time parsing algorithm were defined [13, 16], but almost nothing is known about its theoretical properties.

This paper investigates LL($k$) parsing for a subclass of conjunctive grammars called linear conjunctive grammars, that is, grammars in which every conjunct in every rule may contain at most one nonterminal symbol. Linear conjunctive grammars are important for being equivalent to one-way real-time cellular automata, also known as trellis automata [14, 15], and the associated family of languages has received quite a lot of attention in the literature [3, 4, 7, 20], including some recent work on their expressive power [21, 22]. Turning to the LL($k$) subfamily of linear conjunctive grammars, it was proved that they cannot define a language as simple as \{a^n b^n | n ≥ 0, s ∈ \{a, b\}\} [17], but this is about all that is known about this family. However, in spite of these grammars’ inability to define some particular examples, this family may still contain some computationally hard specimens. Furthermore, it remains unknown whether these languages form a hierarchy by $k$.

This paper addresses both the computational complexity of LL($k$) linear conjunctive grammars, and the existence of a hierarchy by $k$. First, it is shown that, like for ordinary LL($k$) linear grammars, the hierarchy by $k$ collapses, and LL(1) linear conjunctive grammars are as powerful as LL($k$) linear conjunctive, for any $k$. Secondly, a parsing algorithm for LL($k$) linear conjunctive grammars is constructed, which not only works in linear time, but also uses logarithmic space. Accordingly, all languages defined by LL($k$) linear conjunctive grammars lie in the complexity class $L$, and therefore, under the standard assumption that $L \neq P$, these grammars cannot define any P-complete languages, unlike linear conjunctive grammars without the LL($k$) condition [7].

2 Definitions

Definition 1. A linear conjunctive grammar is a quadruple $G = (\Sigma, N, R, S)$ that consists of the following components:

1. $\Sigma$ is the alphabet of the language being defined.

2. $N$ is a finite set of nonterminals. Each nonterminal specifies some property that a given string from $\Sigma^*$ can have or not have.

3. $R$ is a finite set of rules, with each rule describing a possible structure of a string with a property $A \in N$. Each rule is either of the form $A \to u_1B_1v_1 \& u_2B_2v_2 \& \cdots \& u_rB_rv_r$, with $B_1, \ldots, B_r \in N$ and $u_1, v_1, \ldots, u_r, v_r \in \Sigma^*$, or of the form $A \to y$, with $y \in \Sigma^*$.

4. $S \in N$ is the initial nonterminal symbol.

In a rule $A \to u_1B_1v_1 \& u_2B_2v_2 \& \cdots \& u_rB_rv_r$, each string $u_iB_iv_i$ is called a conjunct. Such a rule intuitively means that if a string $w$ is representable as each conjunct—or, to be precise, $w = u_is_iv_i$, where $s_i$ has the property $B_i$, for each $i = 1, \ldots, r$—then the string $w$ has the property $A$. A rule of the form $A \to x$, where $x \in \Sigma^*$, naturally means that $x$ has the property $A$.

The language described by a conjunctive grammar can be naturally defined by generalizing parse trees used for ordinary (“context-free”) grammars. This generalization allows leaves to
Figure 1: A parse tree of $w$ as $A$

have multiple incoming edges, which correspond to representations of the same substring by different conjuncts.

Parse trees for conjunctive grammars are just like parse trees in ordinary ("context-free") grammars, but whenever a rule involving conjunction is used at a node, this node has a separate subtrees for each conjunct, and all these subtrees share the same set of leaves. Accordingly, these are, strictly speaking, directed acyclic graphs rather than trees, but only leaves may have multiple incoming edges.

**Definition 2.** Let $G = (\Sigma, N, R, S)$ be a linear conjunctive grammar. A parse tree of a string $w = a_1 \ldots a_n \in \Sigma^*$ as $A \in N$ has $n$ ordered leaves labelled with $a_1, \ldots, a_n$, a root node labelled with $A$, and

- either there is a rule $A \rightarrow w \in R$, and the rule node $A$ has all leaves as its immediate descendants,
- or there exists a rule $A \rightarrow u_1B_1v_1 \& \ldots \& u_mB_mv_m$, such that, for each $i$-th conjunct, $w = u_iy_iv_i$ for some $y_i$, and $A$ has $m$ groups of descendants corresponding to its conjuncts, with the group corresponding to each $u_iB_iy_i$ containing $|u_iB_iy_i|$ immediate descendants: the first $|u_i|$ leaves of $w$, a node labelled with $B_i$ spanning over the substring $y_i$, and the last $|v_i|$ leaves of $w$, and, furthermore, the subtree of $B_i$ is a parse tree of $y_i$ as $B_i$.

Figure 1 illustrates groups of descendants of a node $A$.

A parse tree of a string as $S$ is called simply a parse tree.

Each nonterminal $A$ defines a language $L_G(A) = L(A)$, which is the set of all strings $w$, for which there exists a tree of $w$ as $A$. The language defined by the grammar is the language defined by its initial symbol: $L(G) = L_G(S)$.

The language defined by a conjunction $\varphi = \alpha_1 \& \ldots \& \alpha_r$ is defined as $L(\varphi) = L(\alpha_1) \cap \ldots \cap L(\alpha_r)$.

**Definition 3.** Let $G = (\Sigma, N, R, S)$ be a linear conjunctive grammar, and let $\tau$ be a subtree of some parse tree, with the root of $\tau$ labelled with $A \in N$ (an $A$-subtree). Let $y$ be the string of all leaves located to the right of the rightmost leaf in $\tau$. Then it is said that $y$ follows the subtree $\tau$. This is illustrated in Figure 3(left).
Figure 2: String $y$ follows an $A$-subtree (left) and string $x$ defines the rule $\alpha$ (right)

The class of LL($k$) linear conjunctive grammar studied in this paper is defined by the following restriction.

**Definition 4.** An LL($k$)-table for a linear conjunctive grammar $G = (\Sigma, N, R, S)$ is a partial function $T: N \times \Sigma^\leq k \rightarrow R$, which satisfies the following condition. For every subtree $\tau$ of any parse tree, let $A \in N$ be the label of the root of $\tau$, and let $x \in \Sigma^\leq k$ be the first $k$ leaves starting from the first leaf of $\tau$; then, the rule applied to the root of $\tau$ must be $T(A, x)$, as illustrated in Figure 2(right).

If an LL($k$) table for a grammar $G$ exists, then $G$ is said to be LL($k$).

**Example 1.** The following LL($k$) linear conjunctive grammar defines the language $\{a^n b^n c^n \mid n \geq 0\}$.

$S \rightarrow A & C$
$A \rightarrow aA \mid D$
$D \rightarrow bDc \mid \varepsilon$
$C \rightarrow aCc \mid B$
$B \rightarrow bB \mid \varepsilon$

The LL(1) table for this grammar is given below.

|   | $\varepsilon$ | $a$ | $b$ | $c$ |
|---|---------------|-----|-----|-----|
| $S$ | $S \rightarrow A & C$ | $S \rightarrow A & C$ | $-$ | $-$ |
| $A$ | $A \rightarrow D$ | $A \rightarrow aA$ | $A \rightarrow D$ | $-$ |
| $D$ | $D \rightarrow \varepsilon$ | $-$ | $D \rightarrow bDc$ | $D \rightarrow \varepsilon$ |
| $C$ | $C \rightarrow B$ | $C \rightarrow aCc$ | $C \rightarrow B$ | $-$ |
| $B$ | $B \rightarrow \varepsilon$ | $-$ | $B \rightarrow bB$ | $B \rightarrow \varepsilon$ |

## 3 The aligned form of an LL($k$) linear conjunctive grammar

In this section it is shown that every LL($k$) linear conjunctive grammar can be transformed to a normal form called the **aligned form**, which is similar to the Greibach normal form for non-linear grammars.

**Definition 5.** A linear conjunctive grammar $G = (\Sigma, N, R, S)$ is called **aligned**, if each rule in $G$ is either of the form $A \rightarrow aC_1v_1 \& \ldots \& aC_mv_m$, with $a \in \Sigma$, $m \geq 1$, $C_1, \ldots, C_m \in N$ and $v_1, \ldots, v_m \in \Sigma^*$, or of the form $A \rightarrow y$, with $y \in \Sigma^*$.

In general, it is likely that some linear conjunctive grammars cannot be transformed to the aligned form (although, as to the authors’ knowledge, no proof has ever been presented). However, for the LL($k$) subclass, a transformation turns out to be possible.
The transformation consists of two steps: first, so called left-recursive rules are eliminated from the grammar, and then each rule \( A \rightarrow u_1C_1v_1 & \ldots & u_mC_mv_m \) is “aligned” by introducing new nonterminal symbols, so that in each conjunct there is exactly one symbol before a nonterminal.

**Definition 6.** A rule \( A \rightarrow \alpha_1 & \ldots & \alpha_r \) is called left-recursive, if at least one of its conjuncts is of the form \( \alpha_j = Bt \), for some \( B \in N \) and \( t \in \Sigma^* \).

Before describing the transformation, it is convenient to establish the uniqueness of parse trees in LL\((k)\) linear conjunctive grammars, which will be used many times throughout the paper. The result holds for all LL\((k)\) conjunctive grammars, not necessary linear. However, for simplicity, the proof is given only in the linear case.

**Lemma 1.** Let \( G \) be an LL\((k)\) linear conjunctive grammar, and let \( \tau_1 \) and \( \tau_2 \) be any two parse trees for a string \( w \in \Sigma^* \). Let \( \tau_1^A \) and \( \tau_2^A \) be two A-subtrees in \( \tau_1 \) and in \( \tau_2 \), respectively, such that the leaves to the left of each subtree form the same string \( s \in \Sigma^* \). Then, \( \tau_1^A \) and \( \tau_2^A \) are identical, and, in particular, define the same substring of \( w \).

**Proof.** Since the selected subtrees of \( \tau_1 \) and of \( \tau_2 \) are positioned within these trees after the same string of leaves \( s \), the first \( k \) leaves starting from the first leaf of \( \tau_1^A \) and the first \( k \) leaves starting from the first leaf of \( \tau_2^A \) form the same substring \( x \). Then, since the grammar \( G \) is LL\((k)\), the same rule \( T(A, x) \) is applied at the roots of both subtrees. The rule applied to the root of each subtree \( \tau_1^A \) and \( \tau_2^A \) is identical. This is proved by induction on the height of \( \tau_1^A \).

If \( \tau_1^A \) consists of a single rule \( A \rightarrow y \), the same rule is applied to the root of \( \tau_2^A \), and thus \( \tau_2^A \) also consists of a single rule \( A \rightarrow y \). This substring \( y \) is the one immediately following \( s \) in \( w \) (that is, \( w = sy\Sigma^* \)).

Now assume that the rule applied to the root of \( \tau_1^A \) is \( A \rightarrow u_1C_1v_1 \ldots u_mC_mv_m \). Then the rule applied to the root of \( \tau_2^A \) is also \( A \rightarrow u_1C_1v_1 \ldots u_mC_mv_m \). For each \( j \in \{1, \ldots, m\} \), denote by \( \tau_1^j \) and \( \tau_2^j \) the subtrees corresponding to the conjunct \( u_jC_jv_j \) in \( \tau_1^A \) and in \( \tau_2^A \), respectively. By the induction hypothesis, for each \( j \in \{1, \ldots, m\} \), the subtrees \( \tau_1^j \) and \( \tau_2^j \) are identical. Therefore, the subtrees \( \tau_1^A \) and \( \tau_2^A \) are also identical. \( \square \)

The next lemma establishes that it is possible to remove left-recursive rules from each LL\((k)\) linear conjunctive grammar.

**Lemma 2.** For every LL\((k)\) linear conjunctive grammar \( G = (\Sigma, N, R, S) \), there exists an LL\((k)\) linear conjunctive grammar \( G' = (\Sigma, N, R', S) \) without left-recursive rules that defines the same language as \( G \).

**Proof.** In the new grammar \( G' \), each rule will simulate a certain fragment of a parse tree in \( G \), comprised of node and a tree of all left-recursive chains coming out of this node in different conjuncts.

Let \( \tau \) be a parse tree in \( G \) with a selected A-subtree \( \tau_A \), and let \( x \) be the string formed by the first \( k \) leaves in the tree, starting from the first leaves of the A-subtree. Denote by \( w \) the string defined by \( \tau \), and denote by \( y \) the substring of \( w \) defined by \( \tau_A \).

Every such pair \((\tau, \tau_A)\) defines a rule \( A \rightarrow \varphi(\tau, \tau_A) \) in the new grammar. It will be shown later that the resulting set of rules is finite.

Let us call a conjunct \( \alpha \) normal if it is not left-recursive, that is, either \( \alpha = uCv \) for some \( C \in N \) and \( u \in \Sigma^*, v \in \Sigma^* \), or \( \alpha \in \Sigma^* \).

A left chain is a path \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m \) in a parse tree, wherein a left-recursive rule \( B_j \rightarrow \ldots \& B_{j+1}t_{j+1} \ldots \) is applied to each vertex \( v_j \) with \( j < m \), and \( v_{j+1} \) is the immediate descendant of \( v_j \) corresponding to the conjunct \( B_{j+1}t_{j+1} \).
Denote by $v_A$ the root of the $A$-subtree. A normal conjunct $\alpha$ is reachable from $A$ via a left chain, if there exists a left chain $v_0 = v_A \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$, wherein the rule applied to $v_m$ is $B_m \rightarrow \ldots & \alpha & \ldots$, as in Figure 3.

For convenience of the further proof, let us define left chains isomorphism.

**Definition 7.** Let $\tau$ and $\tau'$ be two parse trees in $G$, and let $v_0 \rightarrow \ldots \rightarrow v_m$ and $v'_0 \rightarrow \ldots \rightarrow v'_m$ be left chains in $\tau$ and in $\tau'$, respectively.

The left chains $v_0 \rightarrow \ldots \rightarrow v_m$ and $v'_0 \rightarrow \ldots \rightarrow v'_m$ are called isomorphic, if, for each $j \in \{0, \ldots, m\}$, the following conditions hold:

- The vertices $v_j$ and $v'_j$ are labelled with the same nonterminal.
- For $j < m$, if the rule applied to $v_j$ is $B_j \rightarrow \ldots & B_{j+1}t_{j+1} & \ldots$, where the conjunct $B_{j+1}t_{j+1}$ corresponds to the vertex $v_{j+1}$, then the same rule is applied to $v'_j$ in the other chain, and the immediate descendant of $v'_j$ corresponding to the conjunct $B_{j+1}t_{j+1}$ is $v'_{j+1}$.

The next claim establishes that the set of all left chains beginning in some vertex $v$ (and hence also the set of all normal conjuncts reachable from $v$ via left chains) is, up to isomorphism, defined just by the nonterminal at $v$ and by the first $k$ leaves starting from the first leaf of the subtree of $v$, and does not depend on the rest of the parse tree.

**Claim 1.** Let $\tau$ and $\tau'$ be two parse trees in $G$ with selected $A$-subtrees $\tau_A$ and $\tau'_A$, respectively. Assume that in both parse trees, the first $k$ leaves, starting from the first leaves of $A$-subtrees, form the same string $x$.

Then, for each left chain $v_0 = v_A \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$ in $\tau$, there exists an isomorphic left chain $v'_0 = v'_A \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_m$ in $\tau'$.

**Proof.** The claim is proved by induction on $m$, the length of a left chain in $\tau$.

For $m = 0$, both left chains consist of a single vertex $A$, which makes them isomorphic.

Now assume that $m > 0$, and let $v_0 = v_A \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$ be a left chain in $\tau$, wherein some rule $B_j \rightarrow \ldots & B_{j+1}t_{j+1} & \ldots$ is applied to each vertex $v_j$, with $j < m$.

By the induction hypothesis, there exists a left chain $v'_0 \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_{m-1}$ in $\tau'$, which is isomorphic to $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{m-1}$. Since each conjunct $B_jt_j$ begins with a nonterminal symbol, the subtrees of $v_{m-1}$ and $v'_{m-1}$ have the same first leaf, and hence the same string of first $k$ leaves beginning at this position. Therefore, the same rule $B_{m-1} \rightarrow \ldots & B_{m}t_{m} & \ldots$ is applied to both subtrees $v_{m-1}$ and $v'_{m-1}$.

Then $v'_m$ can be defined as the immediate descendant of $v'_{m-1}$ corresponding to the conjunct $B_{m}t_{m}$, and this makes the left chain $v'_0 \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_m$ isomorphic to $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$. \qed
A normal conjunct can be reachable from $A$ via several different left chains, and thus each normal conjunct $uCv$ can correspond to several different subtrees in $\tau_A$. The next claim establishes that all such subtrees are identical.

**Claim 2.** Let $\alpha = uCv$ be a normal conjunct reachable from $v_A$ via two left chains, $v_1^1 = v_A \rightarrow \ldots \rightarrow v_m^1$ and $v_0^2 = v_A \rightarrow \ldots \rightarrow v_m^2$, and let $\tau_C^1$ and $\tau_C^2$ be the subtrees corresponding to $\alpha$ in these left chains. Then the subtrees $\tau_C^1$ and $\tau_C^2$ are identical.

**Proof.** The roots of the subtrees $\tau_C^1$ and $\tau_C^2$ are both labelled with the same nonterminal $C$, and the leaves to the left of each subtree form the same string $su$, where $s$ is the string of all leaves before $\tau_A$. Then, by Lemma 1, the subtrees $\tau_C^1$ and $\tau_C^2$ are identical. \qed

Let $\alpha$ be a normal conjunct reachable from $A$ via some left chain. Denote by $t_\alpha$ the string of all leaves of $\tau_A$ following the subtree corresponding to $\alpha$.

Now the rule $A \rightarrow \varphi(\tau, \tau_A)$ can be defined.

Assume that there exists a normal conjunct without nonterminals, which is reachable from $A$ via a left chain. Then, fix any such conjunct and denote it by $\sigma \in \Sigma^*$. The rule $A \rightarrow \varphi(\tau, \tau_A)$ is then defined as $A \rightarrow \sigma t_\sigma$. Note that in this case $\sigma t_\sigma = y$, and hence the rule does not depend on the choice of the conjunct $\sigma$. However, one has to fix some conjunct $\sigma$, because the correctness proof requires a partition of the rule in the form $A \rightarrow \sigma t_\sigma$.

Otherwise, let $\alpha_1, \ldots, \alpha_k$ be all the conjuncts reachable from $A$ via left chains. Each $\alpha_j$ contains a nonterminal. The right-hand side $\varphi(\tau, \tau_A)$ is then defined as the conjunction $\alpha_1 t_{\alpha_1} \& \ldots \& \alpha_k t_{\alpha_k}$.

Accordingly, the rule $A \rightarrow \varphi(\tau, \tau_A)$ is defined by the set of all left chains $v_0 \rightarrow \ldots \rightarrow v_m$, by which normal conjuncts are reachable from $A$. The latter set, on the other hand, is defined by the string $x$ by Claim 1.

Therefore, the resulting rule $A \rightarrow \varphi(\tau, \tau_A)$ is also defined by $x$, and does not depend on the rest of $\tau$. In particular, the set of all rules $A \rightarrow \varphi(\tau, \tau_A)$ constructed for all possible pairs $(\tau, \tau_A)$ is finite.

Now the new grammar $G' = (\Sigma, N, R', S)$ is defined as follows. The set of nonterminals and the initial nonterminal of $G'$ are the same as those in $G$, and the set of rules $R'$ consists of all rules constructed for each possible pair $(\tau, \tau_A)$ and for each nonterminal $A \in N$.

By construction, each rule of the new grammar $G'$ is of the form $A \rightarrow \alpha_1 t_{\alpha_1} \& \ldots \& \alpha_k t_{\alpha_k}$. Let us fix such a partition for each rule; if the same rule can be obtained from different pairs $(\tau, \tau_A)$, then choose a partition corresponding to any pair.

By the construction, $G'$ is linear and does not contain any left-recursive rules.

The proof that $G'$ is LL($k$) and defines the same language as $G$ is based on a one-to-one correspondence between parse trees in $G$ and $G'$.

The following claim establishes how a parse tree in the new grammar can be obtained from a parse tree in the original grammar.

**Claim 3.** Every string defined in $G$ is also defined in $G'$.

**Proof.** Let $w$ be any string in $L(G)$ and fix its parse tree $\widehat{\tau}$. It is claimed that, for every $A$-subtree $\tau$ of $\widehat{\tau}$, which defines some substring $y$, the string $y$ is defined by $A$ in $G'$. The proof is given by induction on the height of $\tau$.

Assume that in $\tau$ there exists a nonterminal $\sigma \in \Sigma^*$ reachable from $A$ via a left chain, and let $t_\sigma$ be the leaves of $\tau$ following $\sigma$, so that $\tau$ defines the string $\sigma t_\sigma = y$. Then, by construction, $G'$ contains a rule $A \rightarrow \sigma t_\sigma$, and accordingly $y \in L_{G'}(A)$, as claimed. In particular, this argument covers of $\tau$ of minimal height, when it consists of a single $A \rightarrow \sigma$ with $\sigma \in \Sigma^*$, proving the the base case of induction.
Now let \( u_1C_1v_1, \ldots, u_rC_rv_r \) be all normal conjuncts reachable from \( A \) via left chains, and let \( t_1, \ldots, t_r \) be the “tails” corresponding to these conjuncts, so that for each \( j \in \{1, \ldots, r\} \), the \( C_j \)-subtree in \( \tau \) is followed by the string \( v_jt_j \). Denote by \( \tau_j \) the \( C_j \)-subtree in \( \tau \) corresponding to the conjunct \( u_jC_jv_j \). Let \( z_j \) be the substring defined in \( \tau_j \). By the induction hypothesis, the \( z_j \) is defined by \( C_j \) in \( G' \).

For each \( j \)-th conjunct, \( u_jz_jv_jt_j = y \). By the construction, \( G' \) contains a rule \( A \to u_1C_1v_1t_1 \& \ldots \& u_rC_rv_rt_r \). Then, \( y \in L_{G'}(A) \) by this rule.

Similarly, from each parse tree in the new grammar, one can construct a parse tree in the original grammar. In doing so, the correspondence between the vertices of the parse tree in the new grammar and the vertices of the parse tree in the original grammar is defined. In this correspondence, each vertex of the parse tree in the new grammar is mapped to a vertex of the parse tree in the original grammar, but some vertices of the parse tree in the original grammar do not occur as an image of any vertex in the parse tree in the new grammar.

**Claim 4.** There exists a function \( h' : \tau' \mapsto (\tau, \rho) \), which maps a parse tree \( \tau' \) of a string as a nonterminal \( D \) in \( G' \) to a parse tree \( \tau \) of the same string as \( D \) in \( G \) and to a mapping \( \rho \) from the set of vertices of \( \tau' \) to the set of vertices of \( \tau \), such that:

1. \( \rho \) maps the root of \( \tau' \) to the root of \( \tau \).
2. If a vertex \( v \) is labelled with a nonterminal \( A \), then the vertex \( \rho(v) \) is also labelled with \( A \).
3. The subtree of \( \rho(v) \) defines the same string as the subtree of \( v \).
4. The subtree of \( \rho(v) \) is followed by the same string as the subtree of \( v \).
5. (a) Suppose that the rule applied to \( v \) is \( A \to y \), and the partition corresponding to that rule is \( y = \sigma t_\sigma \). Then the conjunct \( \sigma \) is reachable from \( \rho(v) \) via a left chain, and the leaves of the subtree of \( \rho(v) \) following this conjunct form the string \( t_\sigma \).
(b) Suppose that the rule applied to \( v \) is \( A \to u_1C_1z_1 \& \ldots \& u_rC_rz_r \), where the partition fixed for each \( z_j \) is \( z_j = v_jt_j \).

Then the set of all normal conjuncts reachable from \( \rho(v) \) via left chains is \( \{u_1C_1v_1, \ldots, u_rC_rv_r\} \), and each \( C_j \)-subtree in the subtree of \( \rho(v) \), with \( j \in \{1, \ldots, r\} \), is followed by string \( v_jt_j \).

The function \( h' \) maps each parse tree \( \tau' \) to a pair \((\tau, \rho)\). However, in the following, for simplicity, \( h' \) will be used as if it maps parse trees to parse trees, and the mapping \( \rho \) exists separately from \( h' \).

**Proof.** The proof is carried out by induction on the height of \( \tau' \). Let \( \rho \) be the label of the root of \( \tau' \).

In the base case, \( \tau' \) consists of a single rule \( A \to y \), with \( y \in \Sigma^* \). By the construction of \( G' \), there is a partition \( y = \sigma t_\sigma \), such that there exists a parse tree in \( G \), with an \( A \)-subtree, wherein the normal conjunct \( \sigma \) is reachable from \( \tau \), and the leaves to the right of \( \sigma \) form the string \( t_\sigma \). Denote that \( A \)-subtree by \( \tau \), and define \( h' (\tau') = \tau \). The function \( \rho \) maps the root of \( \tau' \) to the root of \( \tau \).

Now assume that the rule applied to the root of \( \tau' \) is \( A \to u_1C_1z_1 \& \ldots \& u_rC_rz_r \), wherein each \( z_j \) is partitioned as \( z_j = v_jt_j \) according to the construction. For each \( j \in \{1, \ldots, r\} \), let \( \tau'_j \) denote the \( C_j \)-subtree of \( \tau' \) corresponding to the conjunct \( u_jC_jz_j \). By the induction hypothesis, for each \( j \in \{1, \ldots, r\} \) there is a parse tree \( h' (\tau'_j) \), which has the same root and defines the same string as \( \tau'_j \), as well as a mapping \( \rho_j \) from the set of vertices of \( \tau'_j \) to the set of vertices of \( h' (\tau'_j) \) satisfying the condition in Claim 4.
By construction, there exists a parse tree in $G$ with an $A$-subtree $τ_A$, such that the set of all normal conjuncts reachable from $A$ via left chains is $\{u_1C_1v_1, \ldots, u_rC_rv_r\}$, and for each $j \in \{1, \ldots, r\}$, the string following the conjunct $u_jC_jv_j$ in $τ_A$ is $t_j$.

Let $τ_1$ be the subtree in $τ_A$ corresponding to the conjunct $u_1C_1v_1$. Then the parse tree $h'(τ')$ is obtained from $τ_A$ by replacing each subtree $τ_j$ with the subtree $h'(τ'_j)$.

The mapping $ρ$ for $τ'$ is defined as follows: the root of $τ'$ is mapped to the root of $h'(τ')$, and vertices from each subtree $τ'_j$ are mapped to the corresponding vertices of $h'(τ'_j)$ by the mapping $ρ_j$.

The last claim immediately entails $L(G') \subseteq L(G)$, and therefore the equality $L(G) = L(G')$ is proved.

Consider any vertex $v$ in some parse tree $τ'$ in the new grammar. The rule $A → ϕ(τ,τ_A)$ applied to $v$ is obtained from some parse tree $τ$ in the original grammar, with a selected $A$-subtree $τ_A$. The function $h'$ from claim 4 on the other hand, matches $τ'$ to some parse tree $h'(τ')$, and matches the subtree in $τ'$ with the root $v$ to a subtree in $h'(τ')$ with the root $ρ(v)$.

The next claim states that the rule $A → ϕ(τ,τ_A)$ applied at a vertex $v$ coincides with the rule obtained from the parse tree $h'(τ')$ with the selected subtree $ρ(v)$.

Claim 5. Let $τ'$ be a parse tree in $G'$, let $v$ be a vertex in $τ'$, and let $A → ϕ$ be the rule applied at $v$. Then $ϕ = ϕ(h'(τ'), ρ(v))$, where $τ_{ρ(v)}$ is the subtree of $h'(τ')$ with the root $ρ(v)$.

Proof. First consider the case of $ϕ = y$, with $y \in Σ^*$. By the construction of $G'$, there is a partition $σ = στ_σ$ fixed for $y$. By Claim 5a, the conjunct $σ$ is reachable from $ρ(v)$ via a left chain, and the subtree $ρ(v)$ defines the string $y = στ_σ$. Then, by the construction, $ϕ(h'(τ'), ρ(v)) = y = ϕ$.

Now assume that $ϕ = u_1C_1v_1 \ldots & u_rC_rv_r$, and the partition fixed for each $z_j$ is $z_j = v_jt_j$. Then, by Claim 5b, the set of all normal conjuncts reachable from $ρ(v)$ via left chains equals $\{u_1C_1v_1, \ldots, u_rC_rv_r\}$, and for each $j \in \{1, \ldots, r\}$, the subtree corresponding to $u_jC_jv_j$ is followed in the subtree of $ρ(v)$ with with string $v_jt_j$. Therefore, by the construction, $ϕ(h'(τ'), τ_{ρ(v)}) = u_1C_1v_1t_1 \ldots & u_rC_rv_rt_r = ϕ$.

It remains to prove that $G'$ is LL(k).

Claim 6. The grammar $G'$ is LL(k).

Proof. Let $τ'_1$ and $τ'_2$ be two parse trees in $G'$, each containing an $A$-subtree, and sharing the same substring $a$ forming the first $k$ leaves, starting with the first leaves of $A$-subtrees in $τ'_1$ and $τ'_2$.

It is claimed that the rules applied to the roots of the $A$-subtrees are the same.

Let $τ_1 = h'(τ'_1)$ and $τ_2 = h'(τ'_2)$. Let $v_1$ and $v_2$ be the vertices corresponding to the $A$-subtrees in $τ'_1$ and $τ'_2$, respectively. Let the rule applied to $v_1$ in $τ'_1$ be $A → φ_1$, and let the rule applied to $v_2$ in $τ'_2$ be $A → φ_2$. By Claim 4 in both parse trees $τ_1$ and $τ_2$, the first $k$ leaves starting from the first leaves of subtrees of $ρ(v_1)$ and $ρ(v_2)$, respectively, form the same string $x$. Then, by the construction of rules in the new grammar, $ϕ(h'(τ'_1), τ_{ρ(v_1)}) = ϕ(h'(τ'_2), τ_{ρ(v_2)})$.

On the other hand, Claim 5 entails $ϕ_1 = ϕ(h'(τ'_1), τ_{ρ(v_1)})$ and $ϕ_2 = ϕ(h'(τ'_2), τ_{ρ(v_2)})$. Therefore, $ϕ_1 = ϕ_2$, and the proof is complete.

Once all left-recursive rules are removed from the grammar, the latter can be made aligned by a direct construction.

Lemma 3. For each LL(k) linear conjunctive grammar $G = (Σ, N, R, S)$, there exists an aligned LL(k) grammar $G'$ that defines the same language.
Proof. By Lemma 2, it may be assumed that $G$ does not contain left-recursive rules.

If $G$ is not aligned, then $G$ contains a rule of the form $A \to \ldots \& uA\& \ldots$, with $|u| \geq 1$. Then a new nonterminal $C$ with a single rule $C \to uBv$ is introduced, and the rule $A \to \ldots \& uA\& \ldots$ in $G$ is replaced with $A \to \ldots \& aC\& \ldots$. This is repeated until $G$ becomes aligned.

Such a substitution does not affect the language defined by grammar and the LL($k$) property. Thus, the resulting grammar is LL($k$) and defines the same language as $G$. It is aligned by construction.

4 Transforming an LL($k$) linear conjunctive grammar to LL(1) linear conjunctive

Theorem 1. For each LL($k$) linear conjunctive grammar $G = (\Sigma, N, R, S)$, there exists an aligned LL(1) grammar $G'$ that defines the same language.

The proof of the theorem is naturally split into several stages of construction. The main idea of the construction repeats the idea of construction in the analogous theorem for ordinary, non-conjunctive LL($k$) linear grammars [18, Thm. 1]: it splits into the same stages of transformation, and the actual construction is directly generalized. Some proofs are different from the non-conjunctive case only in the use of conjunction, and are accordingly omitted in this paper. Other parts of the argument, such as the verification of the LL property, require a more detailed analysis of parse trees; there proofs are presented in full.

The main idea of the construction is as follows. Every LL($k$) conjunctive grammar $G = (\Sigma, N, R, S)$ can be implemented in an LL($k$)-parser, which reads the input string symbol by symbol from left to right, and attempts to construct its parse tree along with reading it [13]. The parse tree is constructed top-down. At each step, the LL($k$) parser has the next $k$ input symbols available, and for each unprocessed node in the parse tree, it determines the rule to apply to the nonterminal symbol in this node by accessing the LL-table, indexed by the nonterminal symbols and the $k$ look-ahead symbols. The unprocessed nodes of the parse trees, whose subtrees have not been constructed yet, are stored in a so-called tree-structured stack; but these details of the general algorithm are beyond the scope of this paper.

The task is to reconstruct a given grammar $G$ to obtain an LL(1) linear conjunctive grammar $G'$ that defines the same language as $G$. A hypothetical LL(1)-parser for a grammar $G'$ should select a rule to use at a node of the parse trees, using only a single next input symbol. The main idea is to let the LL(1)-parser delay the choice of a rule until it reads all $k$ next input symbols, which uniquely determine the rule in the original grammar. At the same time, one should somehow remove the previously read substring $u$ from the rule $T(A, ua)$, and this may cause problems if the rule $T(A, ua)$ is “short”, that is, if $T(A, ua)$ is of the form $A \to y$, where $y \in \Sigma^*$ and $|y| < |u|$.

In order to avoid this problematic case, all such “short” rules are to be removed from the grammar $G$ beforehand. Thus, the entire construction consists of two stages: first, the short
rules are removed from the grammar, and then, using the resulting grammar free of short rules, an LL(1) linear conjunctive grammar is constructed using the above idea of buffering lookahead symbols in the nonterminal’s subscript.

The elimination of short rules does not use the LL(k) property of the grammar, and can be carried out for every linear conjunctive grammar. The transformation of a grammar without short rules to LL(1) in turn does not rely on the linearity of the grammar, and can be done for every LL(k) conjunctive grammar without short rules. Nevertheless, for simplicity, at each stage the grammar is assumed to be both linear and LL(k). Furthermore, by Lemma 3, the original grammar $G$ can be assumed to be aligned.

### 4.1 Short rules elimination

**Definition 8.** A rule $A \rightarrow y$ is called short if $|y| < k - 1$ and there exists a parse tree with an $A$-subtree followed by a nonempty string, as in Figure 4 (right).

**Lemma 4.** For each aligned LL(k) grammar $G = (\Sigma, N, R, S)$, there exists an aligned LL(k) grammar $G'$ without short rules that defines the same language.

**Proof.** Nonterminals in the new grammar $G' = (\Sigma, N', R', S_E)$ are of the form $A_u$, with $A \in N$ and $u \in \Sigma^{<k-1}$. The intention is to have $L_{G'}(A_u) = \{ yu \mid y \in L_G(A) \}$.

Each rule for a nonterminal $A_u \in N'$ is obtained by appending the suffix $u$ to the right-hand side of some rule of the original grammar.

For each rule $A \rightarrow y$ in the original grammar, the new grammar has a rule with the suffix $u$ appended.

$$A_u \rightarrow yu$$

For each rule $A \rightarrow aB_1^1v_1 \& \ldots \& aB_j^m v_m$ in the original grammar, the new grammar has a rule

$$A_u \rightarrow B_{s_1}^1 t_1 \& \ldots \& aB_{s_m}^m t_m$$

wherein for each $j \in \{1, \ldots, m\}$, the conjunct $aB_{s_j}^j t_j$ is obtained from the conjunct $aB_j^j v_j$ as follows. The string $s_j$ consists of the first $k - 1$ symbols of the string $v_j u$ (if $|v_j u| \leq k - 1$ then $s = v_j u$), and the string $t_j$ consists of the remaining suffix of $v_j u$, so that $s_j$ and $t_j$ satisfy $s_j = \text{First}_k(v_j u)$ and $s_j t_j = v_j u$. The intuition behind this is that string $u$ is first appended to the conjunct $aB_j^j v_j$, and then the longest possible prefix of $v_j u$ is moved to the subscript of $B_j^j$.

$$aB_j^j v_j \Rightarrow aB_j^j v_j u \Rightarrow aB_{s_j}^j t_j$$

The proof of correctness of the above construction is naturally split into checking several assertions: namely, that $G'$ is an aligned LL(k) grammar, defines the same language as $G$, and does not contain short rules.
Claim 7. If a string \( w \) is defined by a nonterminal \( A_u \) in the new grammar, then \( w = yu \), where \( y \) is defined by the nonterminal \( A \) in the original grammar.

*Proof.* Induction on the height of a parse tree for \( w \) as \( A_u \). \( \square \)

Claim 8. If a string \( y \) is defined by a nonterminal \( A \) in the original grammar, then, in the new grammar, the nonterminal \( A_u \) defines \( yu \).

*Proof.* Induction on the height of a parse tree for \( y \) as \( A \). \( \square \)

The next claim establishes the correspondence between parse trees in the original and the new grammar.

Claim 9. Assume that there is a parse tree in \( G' \), wherein a \( B_s \)-subtree defines \( ys \) by the rule \( B_s \to \varphi' \), which was obtained from the rule \( B \to \varphi \) in the original grammar, and assume that the \( B_s \)-subtree is followed by a string \( z \). Then, there is a parse tree in \( G \), with a \( B \)-subtree that defines \( y \) by the rule \( B \to \varphi \), and is followed by the string \( sz \).

Furthermore, if \( |s| < k - 1 \), then \( z = \varepsilon \).

*Proof.* Induction on the depth of the \( B_s \)-subtree in the parse tree. \( \square \)

Next, it is proved that \( G' \) does not contain any short rules.

Claim 10. There are no short rules in \( G' \).

*Proof.* There are no short rules for nonterminals \( A_u \) with \( |u| = k - 1 \), since, by Claim 7, all strings defined by \( A_u \) are of length at least \( k - 1 \).

And there are no short rules for nonterminals \( A_u \) with \( |u| < k - 1 \), since, by Claim 9, if \( |u| < k - 1 \), then each \( A_u \)-subtree is followed by the empty string. \( \square \)

Finally, it is proved that \( G' \) is \( LL(k) \).

Claim 11. Grammar \( G' \) is \( LL(k) \).

*Proof.* Let \( \tau'_1 \) and \( \tau'_2 \) be parse trees in the grammar \( G' \), each containing an \( A_u \)-subtree, and let the first \( k \) leaves starting from the first leaves of subtrees \( A_u \) form the same string \( x \) in both trees.
Figure 6: Parse trees $\tau_1'$ and $\tau_2'$ from Claim 11

For each $i \in \{1, 2\}$, let $y_iu$ be the string defined by the $A_u$-subtree in $\tau_i'$, let $A_u \to \varphi_i'$ be the rule applied to the root of this subtree, and let $z_i$ be the string following this subtree, as in Figure 6.

The rules $A_u \to \varphi_1'$ and $A_u \to \varphi_2'$ are obtained from some rules $A \to \varphi_1$ and $A \to \varphi_2$ of the original grammar. By Claim 9 there exist parse trees $\tau_1$ and $\tau_2$ in $G$, each containing an $A$-subtree, such that for each $i \in \{1, 2\}$ the $A$-subtree in $\tau_i$ defines the string $y_i$ by the rule $A \to \varphi_i$.

Then the first $k$ leaves of both parse trees, starting with the first leaves of the $A$-subtrees, form the same string $\text{First}_k(y_1uz_1) = \text{First}_k(y_2uz_2) = x$.

Since the grammar $G$ is LL($k$), the rules used in $\tau_1$ and in $\tau_2$ coincide ($\varphi_1 = \varphi_2$), and hence the rules of $G'$ obtained from these rules coincide as well ($\varphi_1' = \varphi_2'$).

Thus it has been shown that $G'$ defines the same language as $G$, is in LL($k$) and does not contain short rules. Also $G'$ is aligned by the construction.

4.2 Reduction to LL(1)

Once all short rules are eliminated from the grammar, it can be further transformed to satisfy LL(1) property.

Lemma 5. For each aligned LL($k$) grammar $G = (\Sigma, N, R, S)$ without short rules, there exists an aligned LL(1) grammar $G'$ that defines the same language.

Proof. Nonterminals of the new grammar $G' = (\Sigma, N', R', \varepsilon S)$ are of the form $uA$, with $A \in N$ and $u \in \Sigma^{\leq k-1}$.

The intention is to have $uA$ define strings from $L_G(A)$ with a prefix $u$ removed. However, the equality $L_G'(uA) = \{ x \mid ux \in L_G(A) \}$ generally does not hold, but it holds that the string $ux$ is defined by a nonterminal $A$ inside some parse tree if and only if the string $x$ is defined by $uA$ inside some parse tree.

The left subscript $u$ of a nonterminal $uA$ works as buffer which stores the last $k - 1$ symbols read by parser.

The initial nonterminal of $G'$ is $\varepsilon S$, which corresponds to $S$ with an empty buffer.

So $N' = \{ uA \mid A \in N, u \in \Sigma^{\leq k-1} \}$. The rules of the new grammar $G'$ are separated in three sets: $R_{buf}, R_G$ and $R_{empty}$.

Rules from $R_{buf}$ are responsible for filling the buffer. For each nonterminal $uA$ with $|u| < k-1$ and for each symbol $b \in \Sigma$, grammar $G'$ contains a rule attaching this symbol to the buffer.

$uA \to b_{ub}A$
Rules from $R_G$ are used when the buffer is filled and thus the parser can deduce which rule from the original grammar should be applied. For each nonterminal $uA \in N'$ and for each symbol $b \in \Sigma \cup \{\varepsilon\}$, where $|u| = k - 1$ and $T(A, ub)$ is defined, grammar $G'$ contains the rule obtained by removing string $u$ from the rule $T(A, ub)$. Suppose $T(A, ub)$ is of the form $A \rightarrow y$. Then, since $G$ does not contain short rules, $y = ux$ for some string $x \in \Sigma^*$ (note that short rules were eliminated exactly to make this part of construction work). Then the corresponding rule in $G'$ is

$uA \rightarrow x$

Now suppose $T(A, ub)$ is of the form $A \rightarrow aB^1v_1 & \ldots & aB^mv_m$. Then $u$ should begin with $a$, and the corresponding rule in $G'$ is

$au'A \rightarrow u'B^1v_1 & \ldots & u'B^mv_m$, where $au' = u$.

Finally, rules from $R_{\text{empty}}$ are for the case when the buffer is not yet filled, but the whole input string has already been consumed by the parser. Namely, for each $uA \in N'$, with $|u| < k - 1$ and with the entry $T(A, u)$ defined, grammar $G'$ contains an empty rule.

$uA \rightarrow \varepsilon$

Note that the sets $R_{\text{buf}}, R_G, R_{\text{empty}}$ are disjoint. For each rule $(uA \rightarrow \varphi') \in R_G$ it is always possible to uniquely determine the rule $A \rightarrow \varphi$ of the original grammar from which it was obtained. If $uA \rightarrow \varphi'$ is of the form $uA \rightarrow x$ with $x \in \Sigma^*$, then $A \rightarrow \varphi = A \rightarrow ux$, and if $uA \rightarrow \varphi'$ is of the form $au'A \rightarrow a'B^1v_1 & \ldots & a'B^mv_m$ then $A \rightarrow \varphi = aB^1v_1 & \ldots & aB^mv_m$.

The proof that $G'$ is $LL(1)$ and defines the same language as $G$ is given in a series of claims.

The correctness of the construction is proved in the following three claims.

Claim 12. Let $uA \in N'$. Let a parse tree of some string $w$ in $G$ contain an $A$-subtree that defines a string $ux$. Then there exists a parse tree of $w$ in $G'$ that contains a $uA$-subtree, which defines the string $x$.

Proof. Induction on the height of the parse tree for $ux$ as $A$.

Claim 13. If, in the grammar $G'$, a nonterminal $uA$ defines a string $x$, then, in the original grammar, the nonterminal $A$ defines the string $ux$.

Proof. Induction on the height of the parse tree for $x$ as $uA$.

Claim 14. Assume that there is a parse tree in $G'$ with an $uA$-subtree that defines a substring $x$ by the rule $uA \rightarrow \varphi'$, and is followed by a string $z$.

Then, there exists a parse tree in $G$ with an $A$-subtree that defines $ux$ and is followed by the string $z$. Moreover, if the rule $uA \rightarrow \varphi'$ is obtained from the rule $A \rightarrow \varphi$ of the original grammar then this $A$-subtree defines $ux$ by the rule $A \rightarrow \varphi$.

Proof. Induction on the depth of the $uA$-subtree.

The grammar $G'$ is linear conjunctive by construction, and Claims 14 and 12 together entail $L(G') = L_G(\varepsilon S) = L_G(S) = L(G)$. It remains to prove that $G'$ is $LL(1)$.

Claim 15. The grammar $G'$ is $LL(1)$.
**Proof.** Consider two parse trees \( \tau'_1 \) and \( \tau'_2 \) of the new grammar \( G' \), each containing an \( uA \)-subtree, and suppose that the strings starting from the first leaves of these subtrees either both begin with the same symbol or are both empty. Denote this symbol as \( b \) (if both strings are empty then \( b = \varepsilon \)).

For each \( i \in \{1, 2\} \), let \( x_i \) be the string defined by the \( uA \)-subtree in \( \tau'_i \), let \( uA \rightarrow \varphi'_i \) be the rule applied to its root, and let \( z_i \) be the string following the subtree \( s \), that \( b = \text{First}_1(x_iz_i) \) now it will be proved that \( \varphi'_1 = \varphi'_2 \).

The proof is given separately for nonterminals \( uA \) with \(|u| < k - 1 \), and for nonterminals \( uA \) with \(|u| = k - 1 \). First, let \(|u| < k - 1 \). Then, each of the rules \( uA \rightarrow \varphi'_1 \) and \( uA \rightarrow \varphi'_2 \) is either in \( R_{buf} \) or in \( R_{empty} \). Consider the cases.

- If both rules are in \( R_{empty} \), then \( \varphi'_1 = \varphi'_2 = \varepsilon \).
- If both rules are in \( R_{buf} \), then \( \varphi'_1 = \varphi'_2 = b \_A \).
- Suppose that one of the rules, say \( \varphi'_1 \), is in \( R_{buf} \), and the other is in \( R_{empty} \). Then \( \varphi'_1 = b \_A \) and \( \varphi'_2 = \varepsilon \). Hence \( b \neq \varepsilon \), because \( \varphi'_1 \in R_{buf} \).

On the other hand, since \( \varphi'_2 \in R_{empty} \), then \( \varepsilon \in L_{G'}(uA) \), and, by Claim [14], there is a parse tree in \( G \) with an \( A \)-subtree that defines the string \( ux_2 = u \), and the leaves to the right of the subtree form the string \( z_2 \). The grammar \( G \) does not contain short rules, and hence \(|u| < k - 1 \) entails \( z_2 = \varepsilon \), and therefore \( b = \text{First}_1(x_2z_2) = \varepsilon \). The contradiction obtained implies that this case is actually impossible.

Now suppose that \(|u| = k - 1 \). Then both rules \( uA \rightarrow \varphi'_1 \) and \( uA \rightarrow \varphi'_2 \) are in \( R_G \), and therefore are obtained from some rules \( A \rightarrow \varphi_1 \) and \( A \rightarrow \varphi_2 \) in the original grammar.

By Claim [14] there are parse trees \( \tau_1 \) and \( \tau_2 \) in \( G \), such that for each \( i \in \{1, 2\} \), the parse tree \( \tau_i \) contains an \( A \)-subtree that defines the string \( ux_i \), the rule applied to the root is \( A \rightarrow \varphi_i \), and the leaves to the right of the subtree form the string \( z_i \).

Then the first \( k \) leaves of these parse trees, starting with the first leaves of \( A \)-subtrees, form the same string \( \text{First}_k(ux_1z_1) = \text{First}_k(ux_2z_2) = ub \).

Since \( G \) is LL(\( k \)), this is the same rule \( (\varphi_1 = \varphi_2) \), and hence \( \varphi'_1 = \varphi'_2 \). \( \square \)

Now it has been proved that \( G' \) is an LL(1) linear conjunctive grammar that defines the same language as \( G \). Then by Lemma [3] there exists an aligned LL(1) grammar which defines the same language as \( G \), and therefore the proof of Lemma [5] is complete. \( \square \)

Together, Lemmata [4] and [5] constitute the proof of Theorem [1].

### 5 An efficient parser for aligned LL(1) linear conjunctive grammars

A parser for a grammar \( G \) is an algorithm that decides whether a given string \( w \in \Sigma^* \) is defined by the grammar. For an ordinary LL(\( k \)) grammar without conjunction, there exists a canonical parser that attempts to reconstruct a parse tree for the input string, while reading it from left to right, At each step the parser uses the next \( k \) input symbols to determine, which rule to apply. The parser uses stack memory, which contains a string of symbols from \( \Sigma \cup N \) representing the projected form of the remaining input string [9] [10] [19].

A classical LL(\( k \)) parser can be generalized to LL(\( k \)) conjunctive grammars, but the generalized parser, instead of a stack, requires a more complicated data structure: a tree-structured stack, which contains multiple top symbols and a single bottom [11] [2] [13] [16].

In this section it is shown that in the case of LL(\( k \)) linear conjunctive grammars, instead of a complicated tree-structured stack, it is sufficient to use a set of standard stacks. Moreover,
it will be proved that the number of stacks in the set never exceeds the number of nonterminal symbols in the grammar (Lemma \text{[3]}) and this fact will allow an implementation of this parser that uses logarithmic space (Theorem \text{[2]}).

Let \( G = (\Sigma, N, R, S) \) be an LL\((k)\) linear conjunctive grammar. By Lemmata \text{[1]} and \text{[3]} it may be assumed that \( G \) is aligned and LL\((1)\). Let \( w = a_1 \ldots a_n \) be an input string. At each step of the computation, the parser’s configuration is a pair \((Z, a_i \ldots a_n)\), where \( Z \) is a set of conjuncts of the form \( \{A_1 v_1, \ldots, A_k v_k\} \), called a stack set, and \( a_i \ldots a_n \) is an unread suffix of the input string. The following invariant is maintained: the entire input string \( w \) is defined by \( G \) if and only if the unread suffix \( a_i \ldots a_n \) is defined by each conjunct in \( Z \), that is, \( a_i \ldots a_n \in L_G(Av) \) for each \( Av \in Z \).

The parser’s initial configuration is a pair \((\{S\}, w)\): there is a single stack containing \( S \), and the whole input remains unread.

At each step of its computation, the parser reads the next input symbol and processes each conjunct in its stack set according to this symbol and the LL\((1)\) table. Let \((\{A_1 v_1, \ldots, A_k v_k\}, a_i \ldots a_n)\) be the current parser’s configuration. Let \( a = a_i \) be the next input symbol (if the whole input is already consumed, then \( a = \varepsilon \)). Then, for each conjunct \( A_j v_j \), the parser determines the correct rule for \( A_j \) and substitutes it for \( A_j \) as follows.

- If \( T(A_j, a) \) is not defined, then the parser reports a parse error and halts.
- If \( T(A_j, a) = A_j \rightarrow y_j \), then the parser checks that the unread suffix \( a_i \ldots a_n \) of the input coincides with \( y_j v_j \). If this is the case, then the parser removes the conjunct \( A_j v_j \) from the stack set, otherwise it reports a parsing error and halts.
- If \( T(A_j, a) = A_j \rightarrow aB_{j,1} v_{j,1} \& \ldots \& aB_{j,m_j} v_{j,m_j} \), then the parser replaces each conjunct \( A_j v_j \) from the stack set with the set of conjuncts \( \{B_{j,1} v_{j,1}, \ldots, B_{j,m_j} v_{j,m_j}, v_j\} \).

Assume that the conjuncts in the stack set are enumerated, so that the rules \( T(A_j, a) \) applied to the first \( r \) conjuncts contain nonterminals, while the rules for the remaining conjuncts \( A_{r+1} v_{r+1}, \ldots, A_k v_k \) are of the form \( T(A_j, a) = A_j \rightarrow y_j \). Altogether, the following rules are used.

\[
\begin{align*}
T(A_1, a) &= A_1 \rightarrow aB_{1,1} v_{1,1} \& \ldots \& aB_{1,m_1} v_{1,m_1} \\
&\vdots \\
T(A_r, a) &= A_r \rightarrow aB_{r,1} v_{r,1} \& \ldots \& aB_{r,m_r} v_{r,m_r} \\
T(A_{r+1}, a) &= A_{r+1} \rightarrow y_{r+1}: \ y_{r+1} v_{r+1} = a_i \ldots a_n \\
&\vdots \\
T(A_k, a) &= A_k \rightarrow y_k, \quad \text{where } y_k v_k = a_i \ldots a_n
\end{align*}
\]

Using these rules, the computation step proceeds as follows.

\[
\begin{align*}
\{A_1 v_1, \ldots, A_k v_k\}, a_i a_{i+1} \cdots a_n &\rightarrow \{B_{1,1} v_{1,1} v_1, B_{1,m_1} v_{1,m_1} v_1, \\
&\vdots \\
&b_{r,1} v_{r,1} v_{r}, \ldots, B_{r,m_r} v_{r,m_r} v_r, a_{i+1} \cdots a_n\}
\end{align*}
\]

If, at some step, the stack set happens to be empty, then the parser has actually already verified that the string is defined by the grammar. At the remaining steps, it switches to “idle mode” and reads the rest of the input symbols.
Since the parser reads one input symbol at each step, if the computation goes successfully, the parser reaches the configuration \((Z_n, \varepsilon)\) after exactly \(n = |w|\) steps. Then, at the last \((n + 1)\)-st step, the parser tries to apply to each conjunct \(Av \in Z_n\) the rule \(T(A, \varepsilon)\), which can only be of the form \(A \rightarrow \varepsilon\). If all these rules exist, the parser completes this last step in the configuration \((Z_{n+1}, \varepsilon)\). If \(Z_{n+1} = \emptyset\), then the computation is accepting. If either \(Z_{n+1} \neq \emptyset\), or the computation halted earlier, then the computation is rejecting.

Thus, the computation consists of exactly \(n + 1\) steps. At each step of the computation, rules are applied to each element from the stack set, and the next input symbol is read (at the last step, no symbol is read). As a result of rule application, conjuncts “spawn”, that is, are substituted with a (possibly empty) set of new conjuncts. Note that the total number of different conjuncts may decrease both because some old conjuncts have no descendants, and because some new conjuncts coincide.

Consider an accepting computation of the parser.

\[
(\{S\}, w) = (Z_0, a_1 \cdots a_n) \rightarrow (Z_1, a_2 \cdots a_n) \rightarrow \ldots \rightarrow (Z_{n+1}, \varepsilon) = (\emptyset, \varepsilon)
\]

Each conjunct from \(Z_{i+1}\) is a descendant of a conjunct from the previous stack set \(Z_i\). Formally, the notion of a descendant is defined as follows.

Let \(\alpha \in Z_i\) be any conjunct. The sets \(Z_j^{\alpha, i}\), with \(j \in \{i, \ldots, n + 1\}\) and \(Z_j^{\alpha, i} \subseteq Z_j\), are constructed inductively as follows.

For \(j = i\), let \(Z_i^{\alpha, i} = \{\alpha\}\). Now let us define the set \(Z_j^{\alpha, i+1}\) using the already constructed set \(Z_j^{\alpha, i}\).

Let \(j > i\), \(Z_j^{\alpha, i+1} = \{A_1v_1, \ldots, A_kv_k\}\), and let \(a = a_i\) be the next symbol of the input string (if the whole input is already read, then \(a = \varepsilon\)).

Each conjunct \(A_pv_p \in Z_j^{\alpha, i+1}\) gives rise to the set \(\text{Next}(A_pv_p) \subseteq Z_j\), which is defined as follows. If \(T(A_p, a) = A_p \rightarrow y\), then \(\text{Next}(A_pv_p) = \emptyset\). If \(T(A_p, a) = A_p \rightarrow aB_{p,1}v_{p,1} \& \ldots \& aB_{p,m_p}v_{p,m_p}\), then \(\text{Next}(A_pv_p) = \{B_{p,1}v_{p,1}v_p, \ldots, B_{p,m_p}v_{p,m_p}v_p\}\).

The set \(Z_j^{\alpha, i+1}\) is then defined as \(\bigcup_{p=1}^{n} \text{Next}(A_pv_p)\).

All conjuncts from the sets \(Z_i^{\alpha, i}, \ldots, Z_{n+1}^{\alpha, i}\) are called descendants of \(\alpha \in Z_i\).

Note, that since each conjunct from \(Z_j\) is a descendant of some conjunct from \(Z_{j-1}\), the stack set \(Z_j\) at the \(j\)-th configuration equals \(\bigcup_{\alpha \in Z_{j-1}} Z_j^{\alpha, j-1}\). However, some conjuncts from \(Z_j\) can at the same time be descendants of several conjuncts from \(Z_{j-1}\), so that the sets \(Z_j^{\alpha, j-1}\) can intersect for different \(\alpha\).

Each conjunct is a descendant of the conjunct \(S\) from the initial configuration, thus \(Z_j = Z_j^{S, 0}\). By the time the computation ends, each conjunct disappears from the stack set, hence, for all \(\alpha\) and for all \(j\), the set \(Z_n^{\alpha, j}\) is empty.

Now let us check the correctness of the above parsing algorithm, and also establish a correspondence between parse trees and accepting computations.

The next lemma states that each accepting computation on some string corresponds to a parse tree of that string.

**Lemma 6.** Let \(G\) be an aligned LL(1) linear conjunctive grammar, and let \((\{S\}, w) = (Z_0, a_1 \cdots a_n) \rightarrow (Z_1, a_2 \cdots a_n) \rightarrow \ldots \rightarrow (Z_{n+1}, \varepsilon) = (\emptyset, \varepsilon)\) be the accepting computation of the \(G\)-parser on the string \(w\). Then there exists a parse tree \(\tau\) for \(w\), and, for each \(i \in \{0, \ldots, n\}\), \(A \in N\) and \(v \in \Sigma^*\), the next two statements are equivalent:

- **There exists an** \(A\)-subtree in \(\tau\), such that the leaves to the left of the subtree form the string \(a_1 \cdots a_i\), while the leaves to the right of the subtree form the string \(v\).
- **The stack set** \(Z_i\) **contains the conjunct** \(Av\).
Proof. The proof introduces some notation for fragments of a parser’s computation evolving from a single conjunct occurring at some \(i\)-th step, and comprised of all its descendants. This is a kind of subcomputation that ignores all conjuncts other than the descendants of a chosen conjunct.

Let \(\alpha = A\nu\) be a conjunct in \(Z_i\). Then, an \((\alpha, i)\)-generated computation is defined as a sequence \((Z_1^{\alpha,i}, a_1 \cdots a_n), (Z_2^{\alpha,i}, a_1 \cdots a_n), \ldots, (Z_\ell^{\alpha,i}, a_1 \cdots a_n) = (\emptyset, a_\ell+1 \cdots a_n)\), where \(\ell\) is the first index of configuration, in which \(\alpha\) has no descendants.

By induction on the length of \((\alpha, i)\)-generated computation (that is, on \(\ell - i\)), it is proved that:

1. Each descendant of \(\alpha\) ends with \(\nu\), and hence is of the form \(u'A\nu'v\).

2. Let \(y\) be a string, such that \(y\nu = a_{i+1} \cdots a_n\). Then, there exists a parse tree \(\tau_\alpha\) for \(y\), with its root labelled with \(A\), such that, for each \(j \in \{i, \ldots, \ell\}\), the next two statements are equivalent:

   - There exists an \(A'\)-subtree in \(\tau_\alpha\), such that the leaves to the left of the subtree form the string \(u' = a_i \cdots a_j\), and the leaves to the right of the subtree form the string \(\nu'\).
   - The stack set \(Z_j^{(\alpha,i)}\) contains the conjunct \(A'\nu'\).

Note, that for the conjunct \(S\) from the initial configuration, point \(\Box\) is trivially satisfied since \(\nu = \varepsilon\), and point \(\Box\) is exactly the statement of the lemma.

In the base case of the induction, the length of a \((\alpha, i)\)-generated computation is 1, and at the \(i\)-th step the parser applies a rule \(A \rightarrow y\) to the conjunct \(A\nu\). Then, since the computation is accepting, it must hold that \(a_{i+1} \cdots a_n = y\nu\). Then the required parse tree for \(y\) consists of a single rule \(A \rightarrow y\) applied to the root.

Now assume that at the \(i\)-th step the parser applies a rule \(A \rightarrow aB_1v_1 & \cdots & aB_mv_m\) to the conjunct \(A\nu\), with \(a = a_i\). Then, \(B_1v_1, \ldots, B_mv_m\nu\) are all the descendants of \(A\nu\) from the \(i\)-th configuration. By the induction hypothesis, for each conjunct \(B_jv_j\nu\), there exists a string \(z_j\), such that \(z_jv_j\nu = a_{i+1} \cdots a_n\), all descendants of \(B_jv_j\nu\) end with \(v_j\nu\), and there exists a parse tree of \(z_j\) with its root labelled with \(B_j\), as in point \(\Box\).

Therefore, \(a_{i+1} \cdots a_n = y\nu\) for some string \(y = az_1v_1 = \cdots = az_mv_m\), and all descendants of \(A\nu\) end with \(\nu\).

Now, parse trees for the strings \(z_1, \ldots, z_j\) with the roots \(B_1, \ldots, B_j\) can be merged into one parse tree for \(y\) with the root \(A\), in which the rule \(A \rightarrow aB_1v_1 & \cdots & aB_mv_m\) is applied to the root.

The next lemma states, that each parse tree corresponds to an accepting computation.

Lemma 7. Let \(G\) be an aligned \(LL(1)\) grammar, and let \(\tau\) be a parse tree for a string \(w\). Then there exists a (unique) accepting computation \((\{S\}, w) = (Z_0, a_1 \cdots a_n), \ldots, (Z_{n+1}, \varepsilon) = (\emptyset, \varepsilon)\), and, for all \(i \in \{0, \ldots, n\}\), \(A \in N\) and \(v \in \Sigma^*\), the following statements are equivalent:

   - The stack set \(Z_i\) contains the conjunct \(A\nu\).
   - The parse tree \(\tau\) contains an \(A\)-subtree, with the leaves to the right of the subtree forming the string \(v\).

Proof. Let us inductively construct configurations \((Z_0, a_1 \cdots a_n), \ldots, (Z_{n+1}, \varepsilon)\) of the accepting computation, at each step assuming, that the last constructed configuration satisfies the condition in the lemma.
The base case $i = 0$ corresponds to the initial configuration $(\{ S \}, w)$. The conjunct $S \in Z_0$ in this case corresponds to the whole tree $\tau$.

Now suppose that $0 < i \leq n + 1$, and the configuration $(Z_{i-1}, t_{i-1})$ is already defined.

To define the next configuration $(Z_i, a_{i+1} \ldots a_n)$, it is sufficient to show, that, for each conjunct $Av \in Z_{i-1}$, the rule $T(A, a_i)$ is defined, and that, if that rule is of the form $T(A, a_i) = A \rightarrow y$, then $yw = a_i \ldots a_n$.

Consider any conjunct $Av \in Z_{i-1}$. By the induction hypothesis, there exists an $A$-subtree in $\tau$, such that the leaves to the left of the subtree form the string $a_1 \ldots a_{i-1}$, while leaves to the right of the subtree form the string $v$. Denote that subtree by $\tau_A$. Since the grammar $G$ is LL(1), the rule applied to the root of $\tau_A$ is $T(A, a)$, where $a = a_i$ ($a = \varepsilon$ if $i = n + 1$). In particular, $T(A, a)$ is defined. If $T(A, a) = A \rightarrow y$, then the existence of $\tau_A$ implies $yw = a_i \ldots a_n$.

Now assume the rule applied to $A$ is $T(A, a) = A \rightarrow aB_1v_1 \& \ldots \& aB_nv_m$. Then, $\tau$ contains subtrees with roots $B_1, \ldots, B_n$, and, for each $j \in \{1, \ldots, m\}$, the leaves to the left of the $B_j$-subtree form the string $a_1 \ldots a_i$, while the leaves to the right of the $B_j$-subtree form the string $v_jv$.

At the $i$-th step, the parser has to apply the same rule $A \rightarrow aB_1v_1 \& \ldots \& aB_nv_m$ to the conjunct $Av$, and therefore conjunct $Av$ gives rise to the set of descendants $Z_{i}^{Av,i-1} = \{ B_1v_1, \ldots, B_nv_m \}$. For every such descendant $B_jv_jv$, as it was mentioned, there is a subtree in $\tau$, such that the leaves to the left of that subtree form the string $a_1 \ldots a_i$, while the leaves to the right of the subtree form the string $v_jv$.

Therefore, the parser is able to perform the $i$-th step of the computation and update its configuration to $(Z_i, t_i)$, where $Z_i = \bigcup_{Av \in Z_{i-1}} Z_{i}^{Av,i-1}$. Each conjunct $B_jv_jv \in Z_i$ is a descendant of some conjunct $Av \in Z_{i-1}$, hence, for each conjunct, there exists a subtree from the statement of the lemma.

Vice versa, assume that $\tau$ contains a $B$-subtree $\tau_B$, with the leaves to the left of $\tau_B$ forming the string $a_1 \ldots a_i$, and with the leaves to the right of $\tau_B$ forming the string $v''$. Let $A$ be the nonterminal labelling the immediate ancestor of $\tau_B$, and let $v$ be the string following the $A$-subtree, so that $v'' = v'v$, for some string $v'$. Since the grammar is aligned, the leaves to the left of the $A$-subtree form the string $a_1 \ldots a_{i-1}$, and, by the induction hypothesis, the stack set $Z_{i-1}$ contains a conjunct $Av$ corresponding to the $A$-subtree. The rule applied to the root of the $A$-subtree and the rule applied to the conjunct $Av$ are both $T(A, a)$, thus conjuncts from $Z_{i}^{Av,i-1}$ one-to-one correspond to the immediate descendants of $A$. Therefore, the stack set $Z_{i}$ contains a conjunct $Bv'v \in Next(A, a)$ corresponding to $\tau_B$.

Thus, the sequence of configurations $(Z_0, t_0), (Z_1, t_1), \ldots, (Z_{n}, \varepsilon), (Z_{n+1}, \varepsilon)$ has been defined, and it remains to show that $Z_{n+1}$ is empty.

By construction, if $Av$ contains a conjunct $Av$, then there is a subtree in $\tau$, such that the leaves to the left of the subtree form the whole input string $w$, while the leaves to the right of the subtree form the string $v$. Then, of course, $v = \varepsilon$, and, since the grammar is aligned, $T(A, \varepsilon) = A \rightarrow \varepsilon$.

Therefore, $Z_{n}$ can contain only conjuncts consisting of a single nonterminal, and at the $n$-th step the parser is able to apply an empty rule to each of these nonterminals. Hence, $Z_{n+1} = \varnothing$, and thus the computation $(Z_0, t_0), (Z_1, t_1), \ldots, (Z_{n+1}, \varepsilon)$ is accepting. By construction, it satisfies the statement of the lemma. $\square$

Finally, it is possible to prove the main property of the described parser, which implies its efficiency: the size of the stack set is bounded by the number of nonterminals in the grammar.

**Lemma 8.** Let $G$ be an aligned LL(1) grammar, $w \in L(G)$, and let $s$ be a prefix of $w$. Assume that the parser’s stack set after reading the prefix $s$ is $Z = \{ A_1v_1, \ldots, A_kv_k \}$. Then, for every two elements $A_1v_1, A_2v_2 \in Z$, it holds that $A_1 = A_2 \Rightarrow v_1 = v_2$, and therefore $|Z| \leq |N|$.
Proof. By Lemma 1, there exists a parse tree $\tau$ for $w$, such that, for each conjunct $Av_i$, with $i \in \{1,2\}$, there exists an $A$-subtree with the leaves to the left of it forming the string $s$, and with the leaves to the right of the subtree forming the string $v_i$. Then, by Lemma 1, both subtrees define the same string $y$. Hence $w = yv_1 = yv_2$, and therefore $v_1 = v_2$. \hfill \square

6 Parsing in LOGSPACE

Lemma 5 proved in the previous section makes it possible to develop an improved implementation of a parser, which uses logarithmic space and still works in linear time.

Theorem 2. The language defined by each LL($k$) linear conjunctive grammar $G = (\Sigma, N, R, S)$ is decidable in logarithmic space and linear time.

Proof. By Theorem 1 there exists an aligned LL(1) linear conjunctive grammar $G'$ that defines the same language as $G$, hence it can be assumed that $G$ is aligned and LL(1).

Consider the LL(1)-parser for $G$ described in Section 5. Its data structures shall now be revised.

By definition, a configuration of an LL(1)-parser at each step of a computation is a pair $(Z,a_i\cdots a_n)$, where $Z$ is a stack set of the form $\{A_1v_1,\ldots ,A_kv_k\}$, and $a_i\cdots a_n$ is the unread suffix of the input string.

Instead of “tails” $v_1,\ldots ,v_k$, the logspace-parser stores only their lengths. Therefore, each conjunct $Av_i$ is encoded in the logspace-parser as the pair $(A,|v_i|)$. Instead of the suffix $a_i\cdots a_n$, the logspace-parser stores only the current position $i$.

Therefore, the corresponding configuration of the logspace-parser is a pair $(Z',i)$, where $Z' = \{(A_1,|v_1|),\ldots , (A_k,|v_k|)\}$.

Let $a = a_i$ be the next symbol of the input. At the $i$-th step of the computation, the LL(1)-parser applies the rule $T(A_j,a)$ to each conjunct $A_jv_j$. The logspace-parser implements this in the following way.

If $T(A_j,a) = A_j \rightarrow y$, with $y \in \Sigma^*$, then the logspace-parser checks that the substring $a_i\cdots a_{i+|y|-1}$ coincides with $y$. Note that since $|y|$ is bounded by the size of the grammar, this is done in constant time. If the strings indeed coincide, then the logspace-parser just removes the pair $(A_j,|v_j|)$ from the stack set, and otherwise the logspace-parser reports a parse error.

If $T(A_j,a) = A_j \rightarrow aB_jv_{j1}v_{j2}\cdots aB_{j,p_j}v_{j,p_j}$, then the logspace-parser checks that each of the strings $v_{j,1},\ldots ,v_{j,p_j}$ coincides with the corresponding substring of the input string, and replaces each pair $(A_j,|v_j|)$ with the set $\{(a,B_{j,1},|v_{j1}| + |v_{j1,i}|),\ldots , (a,B_{j,p_j},|v_j| + |v_{j,p_j}|)\}$. This is also done in constant time, since all $p_j$ and $|v_{j,i}|$ are bounded by the size of the grammar. \hfill \square

References

[1] T. Aizikowitz, M. Kaminski, “Conjunctive grammars and alternating pushdown automata” Acta Informatica, 50:3 (2013), 175–197.

[2] T. Aizikowitz, M. Kaminski, “Linear conjunctive grammars and one-turn synchronized alternating pushdown automata” International Journal of Foundations of Computer Science, 25:6 (2014), 781–802.

[3] K. Čulík II, J. Gruska, A. Salomaa, “Systolic trellis automata”, I and II, International Journal of Computer Mathematics, 15 (1984), 195–212, and 16 (1984), 3–22.

[4] C. Dyer, “One-way bounded cellular automata” Information and Control, 44:3 (1980), 261–281.
[5] M. Holzer, K.-J. Lange, “On the complexities of linear LL(1) and LR(1) grammars”, Fundamentals of Computation Theory (FCT 1993, Hungary, August 23–27, 1993), LNCS 710, 299–308.

[6] O. H. Ibarra, T. Jiang, B. Ravikumar, “Some subclasses of context-free languages in NC1”, Information Processing Letters, 29:3 (1988), 111–117.

[7] O. H. Ibarra, S. M. Kim, “Characterizations and computational complexity of systolic trellis automata”, Theoretical Computer Science, 29 (1984), 123–153.

[8] G. Jirásková, O. Klíma, “Deterministic bi-automata and subclasses of deterministic linear languages”, Language and Automata Theory and Applications—13th International Conference (LATA 2019, St. Petersburg, Russia, March 26–29, 2019), LNCS 11417, 315–327.

[9] D. E. Knuth, “Top-down syntax analysis” Acta Informatica, 1 (1971), 79–110.

[10] R. Kurki-Suonio, “Notes on top-down languages” BIT Numerical Mathematics, 9:3 (1969), 225–238.

[11] P. M. Lewis II, R. E. Stearns, “Syntax-directed transduction” Journal of the ACM, 15:3 (1968), 465–488.

[12] A. Okhotin, “Conjunctive grammars”, Journal of Automata, Languages and Combinatorics, 6:4 (2001), 519–535.

[13] A. Okhotin, “Top-down parsing of conjunctive languages” Grammars, 5:1 (2002), 21–40.

[14] A. Okhotin, “On the equivalence of linear conjunctive grammars to trellis automata”, RAIRO Informatique Théorique et Applications, 38:1 (2004), 69–88.

[15] A. Okhotin, “On the number of nonterminals in linear conjunctive grammars”, Theoretical Computer Science, 320:2–3 (2004), 419–448.

[16] A. Okhotin, “Recursive descent parsing for Boolean grammars” Acta Informatica, 44:3–4 (2007), 167–189.

[17] A. Okhotin, “Expressive power of LL(k) Boolean grammars” Theoretical Computer Science, 412:39 (*2011), 5132–5155.

[18] A. Okhotin, I. Olkhovsky, “On the transformation of LL(k)-linear grammars to LL(1)-linear”, Computer Science in Russia (CSR 2020, Ekaterinburg, Russia, 29 June–3 July 2020), LNCS 12159, 328–340.

[19] D. J. Rosenkrantz, R. E. Stearns, “Properties of deterministic top-down grammars” Information and Control, 17 (1970), 226–256.

[20] V. Terrier, “On real-time one-way cellular array” Theoretical Computer Science, 141:1–2 (1995), 331–335.

[21] V. Terrier, “Recognition of poly-slender context-free languages by trellis automata” Theoretical Computer Science, 692 (2017), 1–24.

[22] V. Terrier, “Some computational limits of trellis automata” Cellular Automata and Discrete Complex Systems (AUTOMATA 2017, Milan, Italy, 7–9 June 2017), LNCS 10248, 176–186.