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 fo 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 [5], 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

[†]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 α and as β. 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 $\{\, a^n b^n s \mid n \geqslant 0,\ s \in \{a, b\} \,\}$ [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. *Σ 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 Σ^* 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 $A \to u_1 B_1 v_1 \ \& \ u_2 B_2 v_2 \ \& \ \cdots \ \& \ u_r B_r v_r$, 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 $A \to u_1 B_1 v_1 \ \& \ u_2 B_2 v_2 \ \& \ \cdots \ \& \ u_r B_r v_r$, each string $u_i B_i v_i$ is called a *conjunct*. Such a rule intuitively means that if a string w is representable as each conjunct—or, to be precise, $w = u_i s_i v_i$, 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
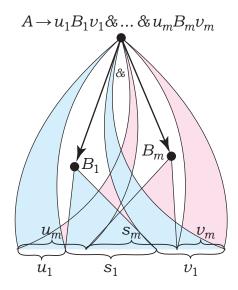
Figure 1: A parse tree of w as A

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 $A \to w \in R$, and the rule node A has all leaves as its immediate descendants,*

- *or there exists a rule $A \to u_1 B_1 v_1 \& \ldots \& u_m B_m v_m$, such that, for each i-th conjunct, $w = u_i y_i v_i$ for some y_i, and A has m groups of descendants corresponding to its conjuncts, with the group corresponding to each $u_i B_i v_i$ containing $|u_i B_i v_i|$ 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.*
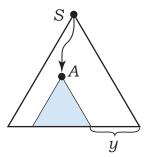
    Figure 1 illustrates groups of descendants of a node A.

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 τ be a subtree of some parse tree, with the root of τ 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 τ. Then it is said that y follows the subtree τ. This is illustrated in Figure 2(left).*
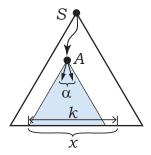
3

Figure 2: String y follows an A-subtree (left) and string x defines the rule α (right)

The class of LL(k) linear conjunctive grammars 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 $T\colon N \times \Sigma^{\leqslant k} \to R$, which satisfies the following condition. For every subtree τ of any parse tree, let $A \in N$ be the label of the root of τ, and let $x \in \Sigma^{\leqslant k}$ be the first k leaves starting from the first leaf of τ; then, the rule applied to the root of τ 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 $\{\, a^n b^n c^n \mid n \geqslant 0 \,\}$.*

$$
\begin{aligned}
S &\;\rightarrow\; A \,\&\, C \\
A &\;\rightarrow\; aA \mid D \\
D &\;\rightarrow\; bDc \mid \varepsilon \\
C &\;\rightarrow\; aCc \mid B \\
B &\;\rightarrow\; bB \mid \varepsilon
\end{aligned}
$$

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

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

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 $A \to aC_1v_1 \,\&\, \ldots \,\&\, aC_mv_m$, with $a \in \Sigma$, $m \geqslant 1$, $C_1, \ldots, C_m \in N$ and $v_1, \ldots, v_m \in \Sigma^*$, or of the form $A \to y$, 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 $A \to u_1 C_1 v_1 \& \ldots \& u_m C_m v_m$ is "aligned" by introducing new nonterminal symbols, so that in each conjunct there is exactly one symbol before a nonterminal.

Definition 6. *A rule $A \to \alpha_1 \& \ldots \& \alpha_r$ 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 $\mathrm{LL}(k)$ linear conjunctive grammars, which will be used many times throughout the paper. The result holds for all $\mathrm{LL}(k)$ 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 τ_1 and τ_2 be any two parse trees for a string $w \in \Sigma^*$. Let τ_1^A and τ_2^A be two A-subtrees in τ_1 and in τ_2, respectively, such that the leaves to the left of each subtree form the same string $s \in \Sigma^*$. Then, τ_1^A and τ_2^A are identical, and, in particular, define the same substring of w.*

Proof. Since the selected subtrees of τ_1 and of τ_2 are positioned within these trees after the same string of leaves s, the first k leaves starting from the first leaf of τ_1^A and the first k leaves starting from the first leaf of τ_2^A form the same substring x. Then, since the grammar G is $\mathrm{LL}(k)$, the same rule $T(A, x)$ is applied at the roots of both subtrees. the rule applied to the root of each subtree τ_1^A and τ_2^A is $T(A, x)$.

Now it is claimed that τ_1^A and τ_2^A are identical. This is proved by induction on the height of τ_1^A.

If τ_1^A consists of a single rule $A \to y$, n the same rule is applied to the root of τ_2^A, and thus τ_2^A also consists of a single rule $A \to y$. This substring y is the one immediately following s in w (that is, $w \in sy\Sigma^*$).

Now assume that the rule applied to the root of τ_1^A is $A \to u_1 C_1 v_1 \& \ldots \& u_m C_m v_m$. Then the rule applied to the root of τ_2^A is also $A \to u_1 C_1 v_1 \& \ldots \& u_m C_m v_m$. For each $j \in \{1, \ldots, m\}$, denote by τ_1^j and τ_2^j the subtrees corresponding to the conjunct $u_j C_j v_j$ in τ_1^A and in τ_2^A, respectively. By the induction hypothesis, for each $j \in \{1, \ldots, m\}$, the subtrees τ_1^j and τ_2^j are identical. Therefore, the subtrees τ_1^A and τ_2^A are also identical. $\square$

The next lemma establishes that it is possible to remove left-recursive rules from each $\mathrm{LL}(k)$ 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 τ be a parse tree in G with a selected A-subtree τ_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 τ, and denote by y the substring of w defined by τ_A.

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

Let us call a conjunct α *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 $v_0 \to v_1 \to \ldots \to v_m$ in a parse tree, wherein a left-recursive rule $B_j \to \ldots \& B_{j+1} t_{j+1} \& \ldots$ 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 $B_{j+1} t_{j+1}$.
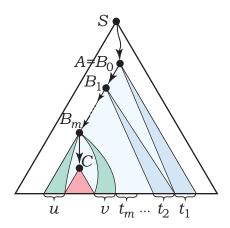
Figure 3: Reachability of a normal conjunct $\alpha = uCv$ from A by a left chain.

Denote by v_A the root of the A-subtree. A normal conjunct α *is reachable from A via a left chain*, if there exists a left chain $v_0 = v_A \to v_1 \to \ldots \to v_m$, wherein the rule applied to v_m is $B_m \to \ldots \& \alpha \& \ldots$, as in Figure 3.

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

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

The left chains $v_0 \to \ldots \to v_m$ and $v'_0 \to \ldots \to v'_m$ 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 $B_j \to \ldots \& B_{j+1}t_{j+1} \& \ldots$, 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 τ and τ' be two parse trees in G with selected A-subtrees τ_A and τ'_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 $v_0 = v_A \to v_1 \to \ldots \to v_m$ in τ, there exists an isomorphic left chain $v'_0 = v'_A \to v'_1 \to \ldots \to v'_m$ in τ'.

Proof. The claim is proved by induction on m, the length of a left chain in τ.

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

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

By the induction hypothesis, there exists a left chain $v'_0 \to v'_1 \to \ldots \to v'_{m-1}$ in τ', which is isomorphic to $v_0 \to v_1 \to \ldots \to v_{m-1}$. Since each conjunct $B_j t_j$ 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 $B_{m-1} \to \ldots \& B_m t_m \& \ldots$ 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 $v'_0 \to v'_1 \to \ldots \to v'_m$ isomorphic to $v_0 \to v_1 \to \ldots \to v_m$. $\quad \square$

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 τ_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, $v_0^1 = v_A \to \ldots \to v_{m_1}^1$ and $v_0^2 = v_A \to \ldots \to v_{m_2}^2$, and let τ_C^1 and τ_C^2 be the subtrees corresponding to α in these left chains. Then the subtrees τ_C^1 and τ_C^2 are identical.*

Proof. The roots of the subtrees τ_C^1 and τ_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 τ_A. Then, by Lemma 1, the subtrees τ_C^1 and τ_C^2 are identical. $\square$

Let α be a normal conjunct reachable from A via some left chain. Denote by t_α the string of all leaves of τ_A following the subtree corresponding to α.

Now the rule $A \to \varphi(\tau, \tau_A)$ 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 $A \to \varphi(\tau, \tau_A)$ is then defined as $A \to \sigma t_\sigma$. Note that in this case $\sigma t_\sigma = y$, and hence the rule does not depend on the choice of the conjunct σ. However, one has to fix some conjunct σ, because the correctness proof requires a partition of the rule in the form $A \to \sigma t_\sigma$.

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

Accordingly, the rule $A \to \varphi(\tau, \tau_A)$ is defined by the set of all left chains $v_0 \to \ldots \to v_m$, 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 $A \to \varphi(\tau, \tau_A)$ is also defined by x, and does not depend on the rest of τ. In particular, the set of all rules $A \to \varphi(\tau, \tau_A)$ constructed for all possible pairs (τ, τ_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 (τ, τ_A) and for each nonterminal $A \in N$.

By construction, each rule of the new grammar G' is of the form $A \to \alpha_1 t_{\alpha_1} \& \ldots \& \alpha_r t_{\alpha_r}$. Let us fix such a partition for each rule; if the same rule can be obtained from different pairs (τ, τ_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 τ 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 τ.

Assume that in τ there exists a normal conjunct $\sigma \in \Sigma^*$ reachable from A via a left chain, and let t_σ be the leaves of τ following σ, so that τ defines the string $\sigma t_\sigma = y$. Then, by construction, G' contains a rule $A \to \sigma t_\sigma$, and accordingly $y \in L_{G'}(A)$, as claimed. In particular, this argument covers of τ of minimal height, when it consists of a single $A \to \sigma$ with $\sigma \in \Sigma^*$, proving the base case of induction.

Now let $u_1 C_1 v_1$, ..., $u_r C_r v_r$ 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 τ is followed by the string $v_j t_j$. Denote by τ_j the C_j-subtree in τ corresponding to the conjunct $u_j C_j v_j$. Let z_j be the substring defined in τ_j. By the induction hypothesis, the z_j is defined by C_j in G'.

For each j-th conjunct, $u_j z_j v_j t_j = y$. By the construction, G' contains a rule $A \to u_1 C_1 v_1 t_1 \& \ldots \& u_r C_r v_r t_r$. Then, $y \in L_{G'}(A)$ by this rule. $\hfill \square$

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 nd 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 τ' of a string as a nonterminal D in G' to a parse tree τ of the same string as D in G and to a mapping ρ from the set of vertices of τ' to the set of vertices of τ, such that:*

1. *ρ maps the root of τ' to the root of τ.*

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 σ 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_σ.*

   (b) *Suppose that the rule applied to v is $A \to u_1 C_1 z_1 \& \ldots \& u_r C_r z_r$, where the partition fixed for each z_j is $z_j = v_j t_j$.*

   Then the set of all normal conjuncts reachable from $\rho(v)$ via left chains is $\{u_1 C_1 v_1, \ldots, u_r C_r v_r\}$, and each C_j-subtree in the subtree of $\rho(v)$, with $j \in \{1, \ldots, r\}$, is followed by string $v_j t_j$.

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

Proof. The proof is carried out by induction on the height of τ'. Let A be the label of the root of τ'.

In the base case, τ' 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 σ is reachable from A, and the leaves to the right of σ form the string t_σ. Denote that A-subtree by τ, and define $h'(\tau') = \tau$. The function ρ maps the root of τ' to the root of τ.

Now assume that the rule applied to the root of τ' is $A \to u_1 C_1 z_1 \& \ldots \& u_r C_r z_r$, wherein each z_j is partitioned as $z_j = v_j t_j$ according to the construction. For each $j \in \{1, \ldots, r\}$, let τ'_j denote the C_j-subtree of τ' corresponding to the conjunct $u_j C_j z_j$. 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 τ'_j, as well as a mapping ρ_j from the set of vertices of τ'_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 $\{u_1 C_1 v_1, \ldots, u_r C_r v_r\}$, and for each $j \in \{1, \ldots, r\}$, the string following the conjunct $u_j C_j v_j$ in τ_A is t_j.

Let τ_j be the subtree in τ_A corresponding to the conjunct $u_j C_j v_j$. Then the parse tree $h'(\tau')$ is obtained from τ_A by replacing each subtree τ_j with the subtree $h'(\tau_j')$.

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

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 $A \to \varphi(\tau, \tau_A)$ 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 $h'(\tau')$, and matches the subtree in τ' with the root v to a subtree in $h'(\tau')$ with the root $\rho(v)$.

The next claim states that the rule $A \to \varphi(\tau, \tau_A)$ applied at a vertex v coincides with the rule obtained from the parse tree $h'(\tau')$ with the selected subtree $\rho(v)$.

Claim 5. *Let τ' be a parse tree in G', let v be a vertex in τ', and let $A \to \psi$ be the rule applied at v. Then $\psi = \varphi(h'(\tau'), \tau_{\rho(v)})$, where $\tau_{\rho(v)}$ is the subtree of $h'(\tau')$ with the root $\rho(v)$.*

Proof. First consider the case of $\psi = y$, with $y \in \Sigma^*$. By the construction of G', there is a partition $y = \sigma t_\sigma$ fixed for y. By Claim 5a, the conjunct σ is reachable from $\rho(v)$ via a left chain, and the subtree $\rho(v)$ defines the string $y = \sigma t_\sigma$. Then, by the construction, $\varphi(h'(\tau'), \tau_{\rho(v)}) = y = \psi$.

Now assume that $\psi = u_1 C_1 z_1 \, \& \, \ldots \, \& \, u_r C_r z_r$, and the partition fixed for each z_j is $z_j = v_j t_j$. Then, by Claim 5b, the set of all normal conjuncts reachable from $\rho(v)$ via left chains equals $\{u_1 C_1 v_1, \ldots, u_r C_r v_r\}$, and for each $j \in \{1, \ldots, r\}$, the subtree corresponding to $u_j C_j v_j$ is followed in the subtree of $\rho(v)$ with with string $v_j t_j$. Therefore, by the construction, $\varphi(h'(\tau'), \tau_{\rho(v)}) = u_1 C_1 v_1 t_1 \, \& \, \ldots \, \& \, u_r C_r v_r t_r = \psi$. $\qquad \square$

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 $\tau_1 = h'(\tau_1')$ and $\tau_2 = h'(\tau_2')$. 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 $A \to \varphi_1$, and let the rule applied to v_2 in τ_2' be $A \to \varphi_2$. By Claim 4, in both parse trees τ_1 and τ_2, the first k leaves starting from the first leaves of subtrees of $\rho(v_1)$ and $\rho(v_2)$, respectively, form the same string x. Then, by the construction of rules in the new grammar, $\varphi(h'(\tau_1'), \tau_{\rho(v_1)}) = \varphi(h'(\tau_1'), \tau_{\rho(v_2)})$.

On the other hand, Claim 5 entails $\varphi_1 = \varphi(h'(\tau_1'), \tau_{\rho(v_1)})$ and $\varphi_2 = \varphi(h'(\tau_2'), \tau_{\rho(v_2)})$. Therefore, $\varphi_1 = \varphi_2$, and the proof is complete. $\qquad \square$

$\qquad \square$

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 $G = (\Sigma, N, R, S)$, 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 $A \to \ldots \& auBv\& \ldots$, with $|u| \geqslant 1$. Then a new nonterminal C with a single rule $C \to uBv$ is introduced, and the rule $A \to \ldots \& auBv\& \ldots$ 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.
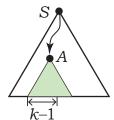
The task is to reconstruct a given grammar G to obtain an $LL(1)$ linear conjunctive grammar G' that defines the same language as G. A hypothetical $LL(1)$-parser for a grammar G' should select a rule to use at a node of the parse trees, using only a single next input symbol. The main idea is to let 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 G.
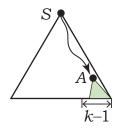
This is done by attaching a *buffer* of at most $k-1$ symbols to each nonterminal A of the original grammars. Accordingly, the nonterminals in G' are of the form $_uA$, where $A \in N$ and $u \in \Sigma^{\leqslant k-1}$. Until the buffer of a nonterminal $_uA$ is not yet filled, the parser applies the following rules for filling up the buffer.

$$_uA \to a\,_{ua}A \qquad\qquad\qquad (|u| < k-1)$$

Once the buffer of $_uA$ is filled, that is, $|u| = k-1$, the $k-1$ symbols of the buffer, together with the next input symbol a available to the $LL(1)$-parser, together form the k symbols necessary to determine the rule $T(A, ua)$, which should be applied to A 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
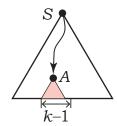
10

Figure 4: The third rule is short, while the first two are not.

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 $A \to y$ 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 $G' = (\Sigma, N', R', S_\varepsilon)$ are of the form A_u, with $A \in N$ and $u \in \Sigma^{\leqslant k-1}$. 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 $A \to y$ in the original grammar, the new grammar has a rule with the suffix u appended.
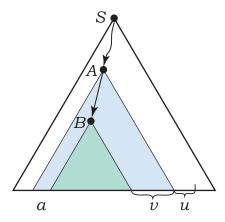
$$A_u \to yu$$

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

$$A_u \to a\, B^1_{s_1}\, t_1 \,\&\, \ldots \,\&\, a\, B^m_{s_m}\, t_m$$

wherein for each $j \in \{1, \ldots, m\}$, the conjunct $a\, B^j_{s_j} t_j$ is obtained from the conjunct $aB^j v_j$ as follows. The string s_j consists of the first $k - 1$ symbols of the string $v_j u$ (if $|v_j u| \leqslant k - 1$ 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 $s_j = \mathrm{First}_k(v_j u)$ and $s_j t_j = v_j u$. The intuition behind this is that string u is first appended to the conjunct $aB^j v_j$, and then the longest possible prefix of $v_j u$ is moved to the subscript of B^j.

$$aB^j v_j \Rightarrow aB^j v_j u \Rightarrow a\, B^j_{s_j}\, t_j$$

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.
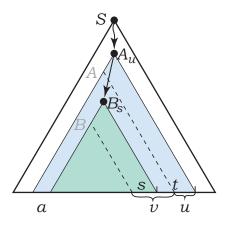
11

Figure 5: The rule $A_u \to \ldots \& a B_s t \& \ldots$ in G' is obtained from the rule $A \to \ldots \& a B v \& \ldots$ in G

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. $\qquad\square$

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. $\qquad\square$

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. $\qquad\square$

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 as 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. $\qquad\square$

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

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

Proof. Let τ_1' and τ_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.
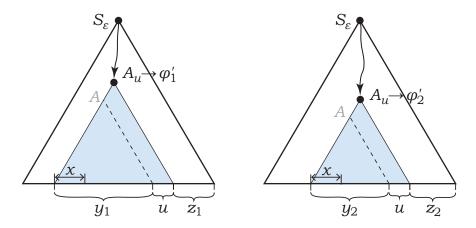
Figure 6: Parse trees τ_1' and τ_2' from Claim 11

For each $i \in \{1,2\}$, let $y_i u$ be the string defined by the A_u-subtree in τ_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 τ_1 and τ_2 in G, each containing an A-subtree, such that for each $i \in \{1,2\}$ the A-subtree in τ_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 $\mathrm{First}_k(y_1 u z_1) = \mathrm{First}_k(y_2 u z_2) = x$.

Since the grammar G is LL(k), the rules used in τ_1 and in τ_2 coincide ($\varphi_1 = \varphi_2$), and hence the rules of G' obtained from these rules coincide as well ($\varphi_1' = \varphi_2'$). $\qquad\square$

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. $\qquad\square$

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 ${}_u A$, with $A \in N$ and $u \in \Sigma^{\leqslant k-1}$.

The intention is to have ${}_u A$ define strings from $L_G(A)$ with a prefix u removed. However, the equality $L_{G'}({}_u A) = \{\, x \mid ux \in L_G(A) \,\}$ 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 ${}_u A$ inside some parse tree.

The left subscript u of a nonterminal ${}_u A$ 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 $N' = \{{}_u A \mid A \in N,\, u \in \Sigma^{\leqslant k-1}\}$. 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 ${}_u A$ with $|u| < k-1$ and for each symbol $b \in \Sigma$, grammar G' contains a rule attaching this symbol to the buffer.

$${}_u A \to b \,{}_{ub} A$$

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 $A \rightarrow aB^1v_1 \& \dots \& aB^mv_m$. Then u should begin with a, and the corresponding rule in G' is

$$_{au'}A \rightarrow {}_{u'}B^1v_1 \& \dots \& {}_{u'}B^mv_m, \quad \text{where } au' = u.$$

Finally, rules from R_{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_{buf}, R_G, R_{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 $_{au'}A \rightarrow {}_{u'}B^1v_1 \& \dots \& {}_{u'}B^mv_m$ then $A \rightarrow \varphi = aB^1v_1 \& \dots \& aB^mv_m$.

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 τ_1' and τ_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 τ_i', let $_uA \to \varphi_i'$ be the rule applied to its root, and let z_i be the string following the subtree s, that $b = \mathrm{First}_1(x_i z_i)$ 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 $_uA \to \varphi_1'$ and $_uA \to \varphi_2'$ 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 $\varphi_1' = \varphi_2' = b\,_{ub}A$.

- Suppose that one of the rules, say φ_1, is in R_{buf}, and the other is in R_{empty}. Then $\varphi_1' = b\,_{ub}A$ 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 $b = \mathrm{First}_1(x_2 z_2) = \varepsilon$. The contradiction obtained implies that this case is actually impossible.

Now suppose that $|u| = k - 1$. Then both rules $_uA \to \varphi_1'$ and $_uA \to \varphi_2'$ are in R_G, and therefore are obtained from some rules $A \to \varphi_1$ and $A \to \varphi_2$ in the original grammar.

By Claim 14, there are parse trees τ_1 and τ_2 in G, such that for each $i \in \{1, 2\}$, the parse tree τ_i contains an A-subtree that defines the string ux_i, the rule applied to the root is $A \to \varphi_i$, 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 $\mathrm{First}_k(ux_1 z_1) = \mathrm{First}_k(ux_2 z_2) = ub$.

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

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. $\qquad\square$

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 [1, 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 8), and this fact will allow an implementation of this parser that uses logarithmic space (Theorem 2).

Let $G = (\Sigma, N, R, S)$ be an LL(k) linear conjunctive grammar. By Lemmata 1 and 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 $(Z, a_i \cdots a_n)$, where Z is a set of conjuncts of the form $\{A_1 v_1, \ldots, A_k v_k\}$, called a *stack set*, and $a_i \cdots a_n$ is an unread suffix of the input string. The following invariant is maintained: the entire input string w is defined by the grammar if and only if the unread suffix $a_i \cdots a_n$ is defined by each conjunct in Z, that is, $a_i \cdots a_n \in L_G(Av)$ 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 $T(A_j, a) = A_j \to y_j$, then the parser checks that the unread suffix $a_i \cdots a_n$ 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 $T(A_j, a) = A_j \to a B_{j,1} v_{j,1} \& \ldots \& a B_{j,m_j} v_{j,m_j}$, then the parser replaces each conjunct $A_j v_j$ from the stack set with the set of conjuncts $\{B_{j,1} v_{j,1} v_j, \ldots, B_{j,m_j} v_{j,m_j} v_j\}$.

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 $T(A_j, a) = A_j \to y_j$. Altogether, the following rules are used.

$$T(A_1, a) = A_1 \to a B_{1,1} v_{1,1} \& \ldots \& a B_{1,m_1} v_{1,m_1}$$
$$\vdots$$
$$T(A_r, a) = A_r \to a B_{r,1} v_{r,1} \& \ldots \& a B_{r,m_r} v_{r,m_r}$$

$$T(A_{r+1}, a) = A_{r+1} \to y_{r+1}; \quad y_{r+1} v_{r+1} = a_i \cdots a_n$$
$$\vdots$$
$$T(A_k, a) = A_k \to y_k, \qquad\qquad\qquad \text{where } y_k v_k = a_i \cdots a_n$$

Using these rules, the computation step proceeds as follows.

$$(\{A_1 v_1, \ldots, A_k v_k\}, a_i a_{i+1} \cdots a_n) \to (\{B_{1,1} v_{1,1} v_1, \ldots, B_{1,m_1} v_{1,m_1} v_1,$$
$$\vdots$$
$$B_{r,1} v_{r,1} v_r, \ldots, B_{r,m_r} v_{r,m_r} v_r\}, a_{i+1} \cdots a_n)$$

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, ε) 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 $A \to \varepsilon$. If all these rules exist, the parser completes this last step in the configuration (Z_{n+1}, ε). If $Z_{n+1} = \varnothing$, then the computation is accepting. If either $Z_{n+1} \neq \varnothing$, 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.

$$(\{S\}, w) = (Z_0, a_1 \cdots a_n) \to (Z_1, a_2 \cdots a_n) \to \ldots \to (Z_{n+1}, \varepsilon) = (\varnothing, \varepsilon)$$

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 $Z_{j+1}^{\alpha,i}$ using the already constructed set $Z_j^{\alpha,i}$.

Let $j > i$, $Z_{j-1}^{\alpha,i} = \{A_1 v_1, \ldots, A_k v_k\}$, 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 $A_p v_p \in Z_{j-1}^{\alpha,i}$ gives rise to the set $Next(A_p v_p) \subseteq Z_j$, which is defined as follows. If $T(A_p, a) = A_p \to y$, then $Next(A_p v_p) = \varnothing$. If $T(A_p, a) = A_p \to aB_{p,1}v_{p,1} \& \ldots \& aB_{p,m_p}v_{p,m_p}$, then $Next(A_p v_p) = \{B_{p,1}v_{p,1}v_p, \ldots, B_{p,m_p}v_{p,m_p}v_p\}$.

The set $Z_j^{\alpha,i}$ is then defined as $\bigcup_{p=1}^{k} Next(A_p v_p)$.

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 α.

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 α and for all j, the set $Z_{n+1}^{\alpha,j}$ 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 $(\{S\}, w) = (Z_0, a_1 \cdots a_n) \to (Z_1, a_2 \cdots a_n) \to \ldots \to (Z_{n+1}, \varepsilon) = (\varnothing, \varepsilon)$ be the accepting computation of the G-parser on the string w. Then there exists a parse tree τ 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 τ, 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 $\alpha = Av$ be a conjunct in Z_i. Then, an (α, i)-*generated computation* is defined as a sequence $(Z_i^{\alpha,i}, a_{i+1} \cdots a_n), (Z_{i+1}^{\alpha,i}, a_{i+1} \cdots a_n), \ldots, (Z_\ell^{\alpha,i}, a_{\ell+1} \cdots a_n) = (\varnothing, a_{\ell+1} \cdots a_n)$, where ℓ is the first index of configuration, in which α has no descendants.

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

1. Each descendant of α ends with v, and hence is of the form $u'A'v'v$.

2. Let y be a string, such that $yv = a_{i+1} \cdots a_n$. Then, there exists a parse tree τ_α 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 τ_α, such that the leaves to the left of the subtree form the string $u' = a_{i+1} \ldots a_j$, and the leaves to the right of the subtree form the string v'.
   - The stack set $Z_j^{(\alpha,i)}$ contains the conjunct $A'v'v$.

Note, that for the conjunct S from the initial configuration, point 1 is trivially satisfied since $v = \varepsilon$, and point 2 is exactly the statement of the lemma.

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

Now assume that at the i-th step the parser applies a rule $A \to aB_1v_1 \& \ldots \& aB_mv_m$ to the conjunct Av, with $a = a_i$. Then, $B_1v_1v, \ldots, B_mv_mv$ are all the descendants of Av from the i-th configuration. By the induction hypothesis, for each conjunct B_jv_jv, there exists a string z_j, such that $z_jv_jv = a_{i+1} \cdots a_n$, all descendants of B_jv_jv end with v_jv, and there exists a parse tree of z_j with its root labelled with B_j, as in point 2.

Therefore, $a_i \cdots a_n = yv$ for some string $y = az_1v_1 = \ldots = az_mv_m$, and all descendants of Av end with v.

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 $A \to aB_1v_1 \& \ldots \& aB_mv_m$ is applied to the root, $\qquad \square$

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 τ be a parse tree for a string w. Then there exists a (unique) accepting computation $(\{S\}, w) = (Z_0, a_1 \cdots a_n), \ldots, (Z_{n+1}, \varepsilon) = (\varnothing, \varepsilon)$, 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 Av.*

- *The parse tree τ 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 τ.

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

To define the next configuration $(Z_i, a_{i+1} \cdots a_n)$, 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 $T(A, a_i) = A \to y$, then $yv = a_i \dots a_n$.

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

Now assume the rule applied to A is $T(A, a) = A \to aB_1v_1 \& \dots \& aB_mv_m$. Then, τ contains subtrees with roots $B_1, \dots, B_m$, and, for each $j \in \{1, \dots, m\}$, the leaves to the left of the B_j-subtree form the string $a_1 \dots a_i$, 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 $A \to aB_1v_1 \& \dots \& aB_mv_m$ to the conjunct Av, and therefore conjunct Av gives rise to the set of descendants $Z_i^{Av, i-1} = \{B_1v_1v, \dots, B_mv_mv\}$. For every such descendant B_jv_jv, as it was mentioned, there is a subtree in τ, such that the leaves to the left of that subtree form the string $a_1 \cdots a_i$, 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 τ contains a B-subtree τ_B, with the leaves to the left of τ_B forming the string $a_1 \dots a_i$, and with the leaves to the right of τ_B forming the string v''. Let A be the nonterminal labelling the immediate ancestor of τ_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 $a_1 \dots a_{i-1}$, 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 $Bv'v \in Next(Av, a)$ corresponding to τ_B.

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

By construction, if A_n contains a conjunct Av, then there is a subtree in τ, 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, $T(A, \varepsilon) = A \to \varepsilon$.

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 $(Z_0, t_0), (Z_1, t_1), \dots, (Z_{n+1}, \varepsilon)$ is accepting. By construction, it satisfies the statement of the lemma. $\qquad\square$

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 $Z = \{A_1v_1, \dots, A_kv_k\}$. 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 $|Z| \leqslant |N|$.*

Proof. By Lemma 6, there exists a parse tree τ 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 $v_1 = v_2$. $\qquad\square$

6   Parsing in LOGSPACE

Lemma 8 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 is encoded in the logspace-parser as the pair $(A, |v|)$. 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 $T(A_j, a) = A_j \to y$, 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 $T(A_j, a) = A_j \to aB_{j,1}v_{j,1} \& \ldots \& aB_{j,p_j}v_{j,p_j}$, 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 $\{(a, B_{j,1}, |v_j| + |v_{j,1}|), \ldots, (a, B_{j,p_j}, |v_j| + |v_{j,p_j}|)\}$. This is also done in constant time, since all p_j and $|v_{j,i}|$ are bounded by the size of the grammar. $\quad\square$

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 NC^1", *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 biautomata 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.