

Expressive power of $LL(k)$ Boolean grammars

Alexander Okhotin²

Department of Mathematics, University of Turku, Turku FI-20014, Finland

Academy of Finland, Helsinki, Finland

Abstract

The paper studies the family of *Boolean LL languages*, generated by Boolean grammars and usable with the recursive descent parsing. It is demonstrated that over a one-letter alphabet, these languages are always regular, while Boolean LL subsets of Σ^*a^* obey a certain periodicity property, which, in particular, makes the language $\{a^n b^{2^n} \mid n \geq 0\}$ non-representable. It is also shown that linear conjunctive LL grammars cannot generate any language of the form $L \cdot \{a, b\}$, with L non-regular, and that no languages of the form $L \cdot c^*$, with non-regular L , can be generated by any linear Boolean LL grammars. These results are used to establish a detailed hierarchy and closure properties of these and related families of formal languages.

Key words: Boolean grammars, conjunctive grammars, context-free grammars, LL grammars, language equations, parsing, recursive descent.

1. Introduction

Boolean grammars [16] extend the definition of the context-free grammars by allowing explicit Boolean operations in the rules. While standard context-free grammars can combine syntactical conditions using only disjunction, effectively specified by multiple rules for a single symbol, *conjunctive grammars* [13] additionally allow conjunction, and *Boolean grammars* further support negation. At the same time, Boolean grammars preserve the most important property of the context-free grammars—that of defining the properties of strings inductively. Accordingly, the main context-free parsing algorithms, such as the Cocke–Kasami–Younger, the recursive descent and the generalized LR, can be extended to Boolean grammars without an increase in computational complexity [16, 17, 19]. The extended expressive power of Boolean grammars and their intuitive clarity make them a powerful tool for specifying languages, which can replace standard context-free grammars in some applications.

Though practical properties of Boolean grammars seem to be as good as in the case of standard context-free grammars, theoretical questions for Boolean grammars present a greater challenge. Already a formal definition of the semantics of Boolean grammars involves certain theoretical problems [11, 16]. A major gap in the knowledge on these grammars is the lack of methods of proving non-representability of languages [16]. Even though the family generated by Boolean grammars was proved to be contained in $DTIME(n^{2.376}) \cap DSPACE(n)$ [16, 22], there is still no proof that any particular language from $DTIME(n^{2.376}) \cup DSPACE(n)$ is not generated by any Boolean grammar, and no methods for constructing such proofs are known.

Results of the latter kind are hard to obtain for many interesting classes of automata and grammars. Consider the family of *trellis automata* [5], also known as one-way real-time cellular automata, which were studied since the 1970s, and which were eventually proved to be equal in power to a subclass of Boolean

^{*}A preliminary version of this paper was presented at the 16th International Symposium on Fundamentals of Computation Theory (FCT 2007) held in Budapest, Hungary on August 27–30, 2007.

^{**}Supported by the Academy of Finland under grants 118540 and 134860.

Email addresses: alexander.okhotin@utu.fi (Alexander Okhotin)

grammars [15]. No methods of establishing non-representability of languages in this family had been known for two decades, until the first such result by Yu [31], who established a pumping lemma for a special case. Only a decade later a general non-representability argument for trellis automata was discovered by Terrier [28], who used it to present the first context-free language not recognized by these automata. Another example is given by the growing context-sensitive languages, for which a method of proving non-representability was discovered by Jurdzinski and Lorys [10] twenty years after the model was proposed.

The purpose of this paper is to establish some limitations of the expressive power of the subclass of Boolean grammars, to which the recursive descent parsing is applicable: the *LL(k) Boolean grammars* [19]. Already for this class, obtaining non-representability proofs presents a challenge: consider that there exists an LL(1) linear conjunctive grammar for the language of computations of any Turing machine [18], which rules out a general pumping lemma. There also exists an LL(1) Boolean grammar for a P-complete language [23], which shows computational non-triviality of this class. This paper proposes several methods for proving non-representability of languages by these grammars, which become the first results of such a kind in the field of Boolean grammars.

Following a definition of Boolean grammars in Section 2, recursive descent parsers for Boolean grammars and their simple formal properties are described in Sections 3 and 4. In Section 5, it is proved that Boolean LL grammars over a unary alphabet generate only regular languages, in contrast with conjunctive grammars of the general form, which are known to have formidable expressive power in the domain of one-letter languages [6, 7, 8, 9]. Section 6 considers subsets of Σ^*a^* representable by Boolean LL grammars and establishes a periodicity property of such languages, which, in particular, implies non-representability of the language $\{a^n b^{2^n} \mid n \geq 0\}$. Stronger non-representability results for two subclasses of Boolean LL grammars with linear concatenation are obtained in Sections 7 and 8. Based on these results, in Section 9, a detailed hierarchy of language families is obtained. Closure properties of these families are determined and compared to the known closure properties of context-free LL languages [27, 30] in Section 10.

2. Boolean grammars and their non-left-recursive subset

Let Σ be a finite non-empty set used as an *alphabet*, let Σ^* be the set of all finite strings over Σ . For a string $w = a_1 \dots a_\ell \in \Sigma^*$ with $a_i \in \Sigma$, the *length* of the string is denoted by $|w| = \ell$. The unique string of length 0 is denoted by ε . For a string $w \in \Sigma^*$, for every partition $w = uv$, u is a *prefix* of w and v is its *suffix*; furthermore, for every partition $w = xyz$, the string y is a *substring* of w .

Any subset of Σ^* is a *language* over Σ . The common operations on languages are *concatenation* $K \cdot L = \{uv \mid u \in K, v \in L\}$ and the Boolean set operations: union $K \cup L$, intersection $K \cap L$ and complementation \bar{L} . *Boolean grammars* are a variant of the context-free grammars, in which all these operations can be explicitly specified.

Definition 1. [16] *A Boolean grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite non-empty sets of terminal and nonterminal symbols respectively; P is a finite set of rules of the form*

$$A \rightarrow \alpha_1 \& \dots \& \alpha_m \& \neg \beta_1 \& \dots \& \neg \beta_n, \quad (1)$$

where $m + n \geq 1$, $\alpha_i, \beta_i \in (\Sigma \cup N)^*$; $S \in N$ is the initial symbol of the grammar.

In this paper, it is further assumed that $m \geq 1$ and $n \geq 0$ in every rule (1). If $m = 1$ and $n = 0$ in every such rule, then a *standard context-free grammar* is obtained. An intermediate family of *conjunctive grammars* [13] has $m \geq 1$ and $n = 0$ in every rule. *Linear* subclasses of Boolean, conjunctive and standard context-free grammars are defined by the additional requirement that $\alpha_i, \beta_i \in \Sigma^* \cup \Sigma^* N \Sigma^*$.

For each rule (1), the objects α_i and $\neg \beta_j$ (for all i, j) are called *conjuncts*, positive and negative respectively. The notation $\pm \alpha_i$ and $\pm \beta_j$ is used to refer to a conjunct of an unspecified sign.

The intuitive semantics of a Boolean grammar is fairly clear: a rule (1) specifies that every string that satisfies each of the conditions α_i and none of the conditions β_i is therefore generated by A . However, constructing a mathematical definition of a Boolean grammar has proved to be a relatively non-trivial task.

Generally, a grammar is interpreted as a system of language equations in variables N , in which the equation for each $A \in N$ is

$$A = \bigcup_{A \rightarrow \alpha_1 \& \dots \& \alpha_m \& \neg \beta_1 \& \dots \& \neg \beta_n \in P} \left[\bigcap_{i=1}^m \alpha_i \cap \bigcap_{j=1}^n \overline{\beta_j} \right]. \quad (2)$$

The vector $(\dots, L_G(A), \dots)$ of languages generated by the nonterminals of the grammar is defined by a solution of this system. If a grammar is conjunctive or standard context-free, then the complementation operation is never used in this system, and hence the system is known to have a *least solution*: this solution is used to define the grammar. But for a Boolean grammar using negation, the system (2) may have no solutions or multiple solutions, and hence the definition requires more precise conditions, which have been a subject of research [11, 16].

Fortunately, for the subclass of Boolean grammars studied in this paper, the formal definition is much simplified. For a recursive descent parser to work correctly, a grammar needs to satisfy a much stronger requirement than solution uniqueness. First, define the following auxiliary notion:

Definition 2. [19] *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar. The relation of reachability in one step, \rightsquigarrow , is a binary relation on the set of strings with a marked substring $\{\alpha \langle \beta \rangle \gamma \mid \alpha, \beta, \gamma \in (\Sigma \cup N)^*\}$, defined as*

$$\alpha \langle \beta A \gamma \rangle \delta \rightsquigarrow \alpha \beta \eta \langle \sigma \rangle \theta \gamma \delta,$$

for all $\alpha, \beta, \gamma, \delta, \eta, \sigma, \theta \in (\Sigma \cup N)^*$, $A \in N$ and for all conjuncts $\pm \eta \sigma \theta$ that occur in any rules for A . Denote its reflexive and transitive closure by \rightsquigarrow^* and its transitive closure by \rightsquigarrow^+ .

This reachability relation is meant to trace logical dependence of the languages generated by different nonterminal symbols on each other, while disregarding the actual Boolean operations in the grammar. It reflects a fragment of a computation of a recursive descent parser investigating this logical dependence, though ignores the data processed in this computation. Some particular cycles in such dependencies correspond to loops in the computation, and hence they must be disallowed:

Definition 3. [19] *A Boolean grammar G is said to be non-left-recursive, if $\varepsilon \langle A \rangle \varepsilon \rightsquigarrow^+ \theta \langle A \rangle \eta$, for any $A \in N$ and $\theta, \eta \in (\Sigma \cup N)^*$, implies $\varepsilon \notin L_G(\theta)$.*

In a strongly non-left-recursive Boolean grammar, the condition $\varepsilon \notin L_G(\theta)$ is strengthened to $\varepsilon \notin L_{G_+}(\theta)$, where $G_+ = (\Sigma, N, P_+, S)$ is a conjunctive grammar defined by removing all negative conjuncts from every rule in G : that is, for every rule (1) in P , the set P_+ contains the rule $A \rightarrow \alpha_1 \& \dots \& \alpha_m$.

For every strongly non-left-recursive grammar, the corresponding system of equations (2) has a unique solution [19]. Then, for every $A \in N$, the language $L_G(A)$ is defined as the value of A in this solution. Let $L(G) = L_G(S)$. When the grammar G is clear from the context, $L_G(A)$ will be abbreviated to $L(A)$.

Definition 4. *Let $G = (\Sigma, N, P, S)$ be a strongly non-left-recursive Boolean grammar. The height of a nonterminal $A \in N$, denoted $h(A)$, is the greatest number of steps in a reachability sequence $\varepsilon \langle A \rangle \varepsilon \rightsquigarrow \dots \rightsquigarrow \theta \langle B \rangle \eta$, for any $B \in N$ and $\theta, \eta \in (\Sigma \cup N)^*$ with $\varepsilon \in L_{G_+}(\theta)$. The height of the grammar is $h(G) = \max_{A \in N} h(A)$.*

Consider the following three simple examples of Boolean grammars:

Example 1. *The following strongly non-left-recursive linear conjunctive grammar (left column) generates the language $\{a^n b^n c^n \mid n \geq 0\}$.*

$S \rightarrow A \& C$	$S = A \cap C$	$L_G(S) = \{a^n b^n c^n \mid n \geq 0\}$
$A \rightarrow aA \mid D$	$A = aA \cup D$	$L_G(A) = \{a^i b^j c^k \mid j = k\}$
$D \rightarrow bDc \mid \varepsilon$	$D = bDc \cup \varepsilon$	$L_G(D) = \{b^m c^m \mid m \geq 0\}$
$C \rightarrow aCc \mid B$	$C = aCc \cup B$	$L_G(C) = \{a^i b^j c^k \mid i = k\}$
$B \rightarrow bB \mid \varepsilon$	$B = bB \cup \varepsilon$	$L_G(B) = b^*$

The middle column contains the corresponding system of equations, and the unique solution of this system is given in the right column.

The grammar is based upon the representation of the language $\{a^n b^n c^n \mid n \geq 0\}$ as an intersection of two context-free languages:

$$\underbrace{\{a^n b^n c^n \mid n \geq 0\}}_{L(S)} = \underbrace{\{a^i b^j c^k \mid j = k\}}_{L(A)} \cap \underbrace{\{a^i b^j c^k \mid i = k\}}_{L(C)}.$$

Example 2. [23] *The following strongly non-left-recursive Boolean grammar generates a P-complete language:*

$$\begin{aligned} S &\rightarrow E \& \neg A b S \& \neg C S \\ A &\rightarrow a A \mid \varepsilon \\ C &\rightarrow a C A b \mid b \\ E &\rightarrow a E \mid b E \mid \varepsilon \end{aligned}$$

This grammar defines the set of yes-instances for a variant of the circuit value problem. In this variant, every gate x_i computes the function $x_i = \neg x_{i-1} \wedge \neg x_{j_i}$, where the first argument is always the preceding gate, and the second argument can be any of the previous gates, $j_i < i$. Such an n -gate circuit is represented as a string $a^{n-1-j_n} b a^{n-2-j_{n-1}} b \dots a^{2-j_3} b a^{1-j_2} b$. The grammar defines the membership of such strings inductively on the number of gates. The conjunct AbS represents the circuits, in which the $(n-1)$ -th gate has value 1, the conjunct CS specifies that the gate number j_n , pointed by the string a^{n-1-j_n} , has value 1, and the negation of both conditions given in the rule for S implements the inductive definition. Since the entire family generated by Boolean grammars is contained in $DTIME(n^{2.376}) \subset P$ [22], this language is among the hardest of its kind.

Example 3. [13] *The following strongly non-left-recursive conjunctive grammar generates the language $\{w c w \mid w \in \{a, b\}^*\}$:*

$$\begin{aligned} S &\rightarrow C \& D \\ C &\rightarrow X C X \mid c \\ X &\rightarrow a \mid b \\ D &\rightarrow a A \& a D \mid b B \& b D \mid c E \\ A &\rightarrow X A X \mid c E a \\ B &\rightarrow X B X \mid c E b \\ E &\rightarrow a E \mid b E \mid \varepsilon \end{aligned}$$

The essence of this grammar is in the nonterminal D , which generates the language $\{u c z u \mid u, z \in \{a, b\}^*\}$. The membership of each $u c z u$ in $L(D)$ can be shown inductively upon the length of u . The base case is given by the rule $D \rightarrow c E$, which generates $\{c z \mid z \in \{a, b\}^*\}$; the rules $D \rightarrow a A \& a D$ and $D \rightarrow b B \& b D$ are used to extend a string $u c z u \in L(D)$ with a and b , respectively, and the nonterminals A and B ensure that z ends with the same symbol. Finally,

$$\underbrace{\{x c y \mid x, y \in \{a, b\}^*, |x| = |y|\}}_{L(C)} \cap \underbrace{\{u c z u \mid u, z \in \{a, b\}^*\}}_{L(D)} = \{w c w \mid w \in \{a, b\}^*\}.$$

3. Boolean recursive descent parser

Recursive descent parsers for Boolean grammars were defined and proved correct in a recent paper [19]. This definition and the statement of correctness are essential for establishing the subsequent results, and they will be presented in this section.

A parser operates according to a parsing table, which is similar in itself to the well-known context-free LL table. Let $k \geq 1$. For a string w , define

$$First_k(w) = \begin{cases} w, & \text{if } |w| \leq k \\ \text{first } k \text{ symbols of } w, & \text{if } |w| > k \end{cases}$$

This definition is extended to languages as $First_k(L) = \{First_k(w) \mid w \in L\}$. Define $\Sigma^{\leq k} = \{w \mid w \in \Sigma^*, |w| \leq k\}$.

Definition 5. [19] A string $v \in \Sigma^*$ is said to follow $\sigma \in (\Sigma \cup N)^*$ if $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* \theta\langle \sigma \rangle \eta$ for some $\theta, \eta \in (\Sigma \cup N)^*$ with $v \in L_G(\eta)$.

Definition 6. [19] Let $G = (\Sigma, N, P, S)$ be a strongly non-left-recursive Boolean grammar, let $k \geq 1$. An $LL(k)$ table for G is any such function $T_k : N \times \Sigma^{\leq k} \rightarrow P \cup \{-\}$, that for every rule $A \rightarrow \varphi$, for every string $u \in L_G(\varphi)$ and for every string v that follows A , the value of $T_k(A, \text{First}_k(uv))$ is $A \rightarrow \varphi$.

A Boolean grammar is said to be $LL(k)$, if such a table exists.

Both grammars in Examples 1–2 are $LL(1)$. An $LL(1)$ table for the grammar in Example 1 is given below. This table is the smallest one for this grammar, in the sense that each of its non-empty entries must be there according to Definition 6. Filling any empty entries with arbitrary rules would lead to a valid table as well.

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

For instance, $T_1(B, b) = B \rightarrow bB$ and $T_1(B, c) = B \rightarrow \varepsilon$, because $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^+ a\langle B \rangle c$ and $b, c \in \text{First}_k(L_G(Bc))$. On the other hand, since no string beginning with a follows B or is generated by B , Definition 6 imposes no requirements on $T_1(B, a)$, so it can be anything in $\{-, B \rightarrow \varepsilon, B \rightarrow bB\}$. The known algorithm for constructing $LL(k)$ tables for Boolean grammars [19] would set $T_1(S, b) = T_1(S, c) = S \rightarrow A\&C$, because both $L_G(A)$ and $L_G(C)$ contain strings beginning with b and c , and understanding that these are disjoint sets of strings is much beyond the analysis done by the algorithm.

In contrast, the grammar in Example 3 is not $LL(k)$ for any k , because there will always be an ambiguity in the choice between $E \rightarrow \varepsilon$ and $E \rightarrow a$ (or $E \rightarrow b$). Suppose this grammar has an $LL(k)$ table T_k for some k and consider the reachability $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* aX^{k-1}c\langle E \rangle aX^{k-1}$. On the one hand, for the rule $E \rightarrow \varepsilon$ and the strings $u = \varepsilon \in L_G(\varepsilon)$ and $v = a^k \in L_G(aX^{k-1})$, it follows that $E \rightarrow \varepsilon \in T_k(E, a^k)$. On the other hand, taking $E \rightarrow aE$, $u = a \in L_G(aE)$ and $v = a^k \in L_G(aX^{k-1})$, one obtains $E \rightarrow aE \in T_k(E, a^k)$. Since $E \rightarrow \varepsilon \neq E \rightarrow aE$, this yields a contradiction. It remains unknown whether the language generated by this grammar, $\{wcv \mid w \in \{a, b\}^*\}$, is generated by any Boolean $LL(k)$ grammar.

Now the recursive descent parser for a given Boolean $LL(k)$ grammar is defined as follows. As in the context-free case, it contains a procedure for each terminal and nonterminal symbol. There are two global variables used by all procedures: the input string $w = w_1w_2 \dots w_{|w|}$ and a positive integer p pointing at a position in this string. Each procedure $s()$, where $s \in \Sigma \cup N$, begins its computation with some initial value of this pointer, $p = i$, and eventually either returns, setting the pointer to $p = j$ (where $i \leq j \leq |w|$), or *raises an exception*, in the sense of an exception handling model, such as the one in C++.

The procedure corresponding to every terminal $a \in \Sigma$ [19] is defined, as in the standard case, as

$a()$ $\{$ if $w_p = a$, then $p = p + 1;$ else raise exception; $\}$
--

For every nonterminal $A \in N$, the corresponding procedure $A()$ [19] chooses a rule using the parsing table, and then proceeds checking the conjuncts one by one. The code for this procedure is defined as follows:

```

A()
{
  switch(T(A, Firstk(wpwp+1...)))
  {
    case A → α1&...&αm&¬β1&...&¬βn:
      (code for the conjunct α1)
      ⋮
      (code for the conjunct αm)
      (code for the conjunct ¬β1)
      ⋮
      (code for the conjunct ¬βn)
      return;
    case A → ...
      ⋮
    default:
      raise exception;
  }
}

```

where the code for every positive conjunct $s_1 \dots s_\ell$ is

```

let first = p;
s1();
⋮
sℓ();
let last = p;

```

(for the first positive conjunct)

```

p = first;
s1();
⋮
sℓ();
if p ≠ last, then raise exception;

```

(for every subsequent positive conjunct)

while the code for every negative conjunct $\neg s_1 \dots s_\ell$ is

```

boolean failed = false;
try
{
  p = first;
  s1();
  ⋮
  sℓ();
  if p ≠ last, then raise exception;
}
exception handler:
  failed = true;
if ¬failed, then raise exception;
p = last; /* if this is the last conjunct in the rule */

```

The code for the first conjunct α_1 stores the initial value of the pointer in the variable *first*, and remembers the end of the substring recognized by α_1 in the variable *last*. Every subsequent positive conjunct α_i is tried beginning from the same position *first*, and the variable *last* is used to check that it consumes exactly the same substring. The code for every negative conjunct tries to recognize a substring in the same way, but reports a successful parse if and only if the recognition is unsuccessful, thus implementing negation.

Now it is easy to explain why the recursive descent parsing requires every rule in the grammar to have at least one positive conjunct. Indeed, the procedure for A needs to return a certain value of the pointer, which represents the end of a substring generated by A . This value can only be computed for positive conjuncts, while the code for negative conjuncts relies on the previously computed value *last*.

Finally, the main procedure [19] of the parser is

```

try
{
    int p = 1;
    S();
    if p ≠ |w| + 1, then raise exception;
}
exception handler:
    Reject;
Accept;

```

The correctness of Boolean recursive descent has been established as follows:

Lemma 1. [19] *Let $k \geq 1$. Let $G = (\Sigma, N, P, S)$ be an $LL(k)$ Boolean grammar. Let $T : N \times \Sigma^{\leq k} \rightarrow P \cup \{-\}$ be an $LL(k)$ table for G , and let the set of procedures be constructed with respect to G and T . Then, for all $y, z, \tilde{z} \in \Sigma^*$ and $s_1, \dots, s_\ell \in \Sigma \cup N$ ($\ell \geq 0$), for which z follows $s_1 \dots s_\ell$ and $First_k(z) = First_k(\tilde{z})$, the code $s_1(); \dots; s_\ell()$, executed on the input $y\tilde{z}$,*

- *returns, consuming y , if $y \in L_G(s_1 \dots s_\ell)$;*
- *raises an exception, if $y \notin L_G(s_1 \dots s_\ell)$.*

For an input string w , applying the lemma to $y = w$, $z = \tilde{z} = \varepsilon$, $\ell = 1$ and $s_1 = S$ asserts that its membership in $L(G)$ is recognized correctly.

4. Simple formal properties

A few *ad hoc* normal form results need to be established for use in the subsequent arguments. The first of these results shows that there is no loss of generality in the assumption that every nonterminal in an $LL(k)$ Boolean grammar is reachable from the initial symbol and generates a non-empty language.

Definition 7. *An $LL(k)$ Boolean grammar $G = (\Sigma, N, P, S)$ is said to be well-behaved, if, for every $A \in N$, (i) $L(A) \neq \emptyset$ and (ii) there exist $\theta, \eta \in (\Sigma \cup N)^*$, such that $\varepsilon \langle S \rangle \varepsilon \rightsquigarrow^* \theta \langle A \rangle \eta$. The grammar $G = (\Sigma, \{S\}, \{S \rightarrow aS\}, S)$ generating \emptyset is also considered well-behaved.*

Lemma 2. *For every $LL(k)$ Boolean grammar there exists an equivalent well-behaved $LL(k)$ Boolean grammar.*

The transformation confirming the correctness of Lemma 2 is not effective, because it requires testing the emptiness of a language, which is undecidable already for linear conjunctive LL grammars [18].

Sketch of a proof. Denote the given grammar by $G = (\Sigma, N \cup N_0, P, S)$, where every $A \in N$ generates a non-empty language, and every $A \in N_0$ satisfies $L_G(A) = \emptyset$. Assume that $L(G) \neq \emptyset$, and hence $S \in N$.

First, construct a grammar $G' = (\Sigma, N, P', S)$ as follows. For every rule

$$A \rightarrow \alpha_1 \& \dots \& \alpha_m \& \neg \beta_1 \& \dots \& \neg \beta_\ell \& \neg \beta_{\ell+1} \& \dots \& \neg \beta_n$$

$$(\alpha_i \in (\Sigma \cup N)^*, \beta_1, \dots, \beta_\ell \in (\Sigma \cup N)^*, \beta_{\ell+1}, \dots, \beta_n \in (\Sigma \cup N \cup N_0)^* N_0 (\Sigma \cup N \cup N_0)^*)$$

in P , whose positive conjuncts contain no references to the symbols in N_0 , the new set P' contains a corresponding rule with all negative conjuncts containing such references omitted:

$$A \rightarrow \alpha_1 \& \dots \& \alpha_m \& \neg \beta_1 \& \dots \& \neg \beta_\ell$$

The grammar G' is also $LL(k)$, and $L_{G'}(A) = L_G(A)$ for every $A \in N$.

At the second step, construct the grammar $G'' = (\Sigma, N'', P'', S)$, in which $N'' = \{A \mid \exists \theta, \eta : \varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* \theta\langle A \rangle \eta\}$ and $P'' \subseteq P'$ contains all rules for nonterminals from N'' . In other words, N'' is the smallest subset of N containing S , such that all rules for nonterminals in this subset do not refer to nonterminals outside of this subset. This grammar remains $LL(k)$, and $L_{G''}(A) = L_{G'}(A)$ for every $A \in N''$. Hence, $L(G'') = L(G)$, and since the grammar satisfies Definition 7 by construction, the lemma is proved. \square

The next statement is that a well-behaved $LL(k)$ Boolean grammar remains $LL(k)$ if another nonterminal is chosen as its initial symbol.

Lemma 3. *Let $G = (\Sigma, N, P, S)$ be a well-behaved $LL(k)$ Boolean grammar. Then, for every nonterminal $T \in N$ taken as a new initial symbol, the grammar $G' = (\Sigma, N, P, T)$ is an $LL(k)$ Boolean grammar with $L_{G'}(A) = L_G(A)$ for all $A \in N$, and there exists a well-behaved $LL(k)$ Boolean grammar generating the same language.*

Proof. Strong non-left-recursivity of G immediately implies strong non-left-recursivity of G' , because this condition is independent of the choice of the initial symbol. Since the systems of language equations corresponding to G and G' are identical, the unique solution $(\dots, L_G(A), \dots)$ of the former system is at the same time the unique solution of the latter system, which proves $L_{G'}(A) = L_G(A)$.

To prove that G' is $LL(k)$, suppose it is not, that is, there exists a pair of distinct rules $A \rightarrow \varphi, A \rightarrow \varphi' \in P$ and strings $u, v, u', v' \in \Sigma^*$ and $\theta, \eta, \theta', \eta' \in (\Sigma \cup N)^*$, such that $\varepsilon\langle T \rangle \varepsilon \rightsquigarrow^* \theta\langle A \rangle \eta$, $\varepsilon\langle T \rangle \varepsilon \rightsquigarrow^* \theta'\langle A \rangle \eta'$, $u \in L_G(\varphi)$, $u' \in L_G(\varphi')$, $v \in L_G(\eta)$, $v' \in L_G(\eta')$ and $First_k(uv) = First_k(u'v')$. Since G is well-behaved, the symbol T is accessible from S as $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* \mu\langle T \rangle \nu$. Combining this with the above, one obtains $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* \mu\theta\langle A \rangle \eta\nu$ and $\varepsilon\langle S \rangle \varepsilon \rightsquigarrow^* \mu\theta'\langle A \rangle \eta'\nu$. Let x be any string in $L_G(\nu)$, which exists, because G is well-behaved. Then the entry $T_k(A, First_k(uvx)) = T_k(A, First_k(u'v'x))$ of the $LL(k)$ table of G should contain both $A \rightarrow \varphi$ and $A \rightarrow \varphi'$. The contradiction obtained proves that G' is $LL(k)$.

Finally, an equivalent well-behaved $LL(k)$ Boolean grammar exists by Lemma 2. \square

Lemma 4. *Let $G = (\Sigma, N, P, S)$ be a well-behaved $LL(k)$ Boolean grammar, let $a \in \Sigma$. Then there exists a well-behaved $LL(k)$ Boolean grammar over the alphabet $\{a\}$ generating $L(G) \cap a^*$.*

Sketch of a proof. The construction is similar to the one in the first part of the proof of Lemma 2. Consider a grammar $G' = (\{a\}, N, P', S)$, in which, for every rule in P with positive conjuncts containing no symbols from $\Sigma \setminus \{a\}$, P' contains the same rule without negative conjuncts containing symbols from $\Sigma \setminus \{a\}$. This grammar is also $LL(k)$, and $L_{G'}(A) = L_G(A) \cap a^*$ for every $A \in N'$. Using Lemma 2, an equivalent well-behaved grammar is obtained. \square

Lemma 5. *For every well-behaved $LL(k)$ Boolean grammar G there exists a well-behaved $LL(k)$ Boolean grammar G' , with $L(G') = L(G)$, in which, for every nonterminal A , there is either one or more rules of the form $A \rightarrow B_1 \& \dots \& B_m \& \neg C_1 \& \dots \& \neg C_n$, or a single rule of the form $A \rightarrow BC$ with $B, C \in N$, $A \rightarrow a$ with $a \in \Sigma$, or $A \rightarrow \varepsilon$.*

The proof of Lemma 5 proceeds by a straightforward decomposition of complex rules with simpler rules: parts of conjunct bodies are moved into auxiliary nonterminals and replaced with references to these nonterminals. It is easy to do this decomposition without losing the $LL(k)$ property.

5. Boolean $LL(k)$ grammars over a unary alphabet

Context-free grammars over a one-letter alphabet are known to generate only regular languages, and linear conjunctive grammars have the same property. In contrast, already conjunctive grammars can generate non-regular unary languages:

Example 4 (Jež [6]). *The following conjunctive grammar, in which the initial symbol is A_1 , generates the language $\{a^{4^n} \mid n \geq 0\}$:*

$$\begin{aligned} A_1 &\rightarrow A_1 A_3 \& A_2 A_2 \mid a \\ A_2 &\rightarrow A_1 A_1 \& A_2 A_6 \mid aa \\ A_3 &\rightarrow A_1 A_2 \& A_6 A_6 \mid aaa \\ A_6 &\rightarrow A_1 A_2 \& A_3 A_3 \end{aligned}$$

Each A_i with $i \in \{1, 2, 3, 6\}$ generates the language $\{i \cdot 4^n \mid n \geq 0\}$.

The above grammar is left-recursive, yet it is possible to eliminate left recursion in it using the method of Okhotin and Reitwießner [25, Cor. 1]. However, even when the issue of left recursion is resolved, it still looks impossible to have an $LL(k)$ grammar for this language, because a hypothetical parser would receive essentially no information in its lookahead string (which remains a^k until the very end of the input), and hence would not be able to choose a rule.

In this section, this intuitive impossibility shall be formally confirmed. It will be proved that Boolean $LL(k)$ grammars over the unary alphabet generate only regular languages, and hence are weaker in power than Boolean grammars of the general form. In particular, the aforementioned language $\{a^{4^n} \mid n \geq 0\}$ is not Boolean $LL(k)$ for any k .

Theorem 1. *Every Boolean $LL(k)$ language over a unary alphabet is regular.*

The proof of Theorem 1 is based upon the following lemma, which states that if a long sequence of a s (that is, longer than the length of the lookahead) follows $B \in N$, then $L(B)$ must be a singleton. For later use, the lemma is proved in a slightly more general case of an alphabet possibly containing other letters besides a , so the statement takes the following form:

Lemma 6. *Let $G = (\Sigma, N, P, S)$ be an $LL(k)$ Boolean grammar, let $B \in N$, $a \in \Sigma$ and let some string in $a^k \Sigma^*$ follow B . Then $L_G(B) \cap a^*$ is a singleton or an empty set.*

The condition that some string beginning with a^k follows B is necessary to apply Lemma 1.

Proof. It is sufficient to prove that any two elements of $L(B) \cap a^*$ coincide.

Let $a^{i_1}, a^{i_2} \in L(B)$, where $0 \leq i_1 \leq i_2$, and let $a^k x$, with $x \in \Sigma^*$, be a string that follows B . Consider the string $a^{k+(i_2-i_1)}x$, for which it is known that $k + (i_2 - i_1) \geq k$, and hence $\text{First}_k(a^{k+(i_2-i_1)}x) = \text{First}_k(a^k x) = a^k$. By Lemma 1, $a^{i_1} \in L(B)$ implies that $B()$ returns on the input $w_1 = a^{i_1} a^{k+(i_2-i_1)}x$, consuming a^{i_1} . On the other hand, $a^{i_2} \in L(B)$ implies that $B()$ should return on $w_2 = a^{i_2} a^k x$, consuming a^{i_2} . Since $w_1 = w_2$, the computations of $B()$ on w_1 and w_2 are actually the same computation, and hence i_1 and i_2 must coincide, which proves the claim. \square

Proof of Theorem 1. According to Lemmata 2 and 5, there is no loss of generality in the assumption that the given language is generated by a well-behaved $LL(k)$ Boolean grammar $G = (\{a\}, N, P, S)$, in which every nonterminal $A \in N$ has either a single rule of the form $A \rightarrow BC$, $A \rightarrow a$, $A \rightarrow \varepsilon$, or one or more rules of the form $A \rightarrow D_1 \& \dots \& D_q \& \neg E_1 \& \dots \& \neg E_r$,

Now it is claimed that for every rule $A \rightarrow BC$, if C generates any string of length k or greater, then $L(B)$ is a singleton. Let $a^j \in L(C)$, where $j \geq k$, and let a^ℓ be a string that follows A . Then $a^{j+\ell} \in a^k a^*$ follows B , and since it is known that $L(B) \subseteq a^*$ and $L(B) \neq \emptyset$, Lemma 6 states that $|L(B)| = 1$.

The next step is to reconstruct the grammar to show that $L(G)$ is regular. For every rule $A \rightarrow BC$, such that $L(B)$ is a singleton, replace this rule with the rule $A \rightarrow a^i C$, where $L(B) = \{a^i\}$. If $L(B)$ is not a singleton, then $L(C) \subseteq a^{\leq k}$ by the claim above, and the rule $A \rightarrow BC$ can be equivalently replaced with the set of rules $\{A \rightarrow a^i B \mid a^i \in L(C)\}$, using the commutativity of concatenation over a unary alphabet.

Consider systems of language equations corresponding to the original and the transformed grammars. Most of the equations are identical, except the equations for all variables A with a unique rule $A \rightarrow BC$. Here the original system has an equation $A = BC$, while the new system has an equation of the form $A = L_B C$ or $A = L_C B$, where $L_B = L_G(B)$ or $L_C = L_G(C)$ are finite constants. So the new system is obtained from the original system by substituting components of its unique solution for some instances of variables,

and hence it must have the same unique solution. Since the new system uses one-sided concatenation, all components of this solution are regular, and the corresponding finite automata can be constructed by the method of Baader and Okhotin [2]. \square

Corollary 1. *Let $G = (\Sigma, N, P, S)$ be a well-behaved Boolean LL(k) grammar. Then, for every $a \in \Sigma$ and for every $A \in N$, the language $L_G(A) \cap a^*$ is regular.*

Proof. Consider the grammar $G' = (\Sigma, N, P, A)$. According to Lemma 3, G' is a well-behaved Boolean LL(k) grammar generating $L_G(A)$. Then, by Lemma 4, there exists a well-behaved Boolean LL(k) grammar G'' , such that $L(G'') = L_G(A) \cap a^*$. This language is regular by Theorem 1. \square

6. Non-representability results for subsets of Σ^*a^*

The triviality of Boolean LL(k) grammars over a unary alphabet, established in the previous section, shall now be applied to demonstrate their further limitations in dealing with long sequences of identical symbols.

Consider languages of the form $L_f = \{a^n b^{f(n)} \mid n \geq 1\}$, where $f: \mathbb{N} \rightarrow \mathbb{N}$. Standard context-free grammars can generate such languages only for linearly growing functions f , due to the pumping lemma. In contrast, linear conjunctive grammars can generate the language $\{a^n b^{2^n} \mid n \geq 1\}$, as proved by Ibarra and Kim [5, Ex. 2.1] using the automaton representation of these grammars. However, the known grammar for this language, obtained by simulating a cellular automaton, is left-recursive, and hence not LL(k).

It will now be shown that no language L_f with superlinearly growing $f(n)$ can be generated by an LL(k) Boolean grammar. This follows from the next theorem, which establishes a periodicity property for languages generated by LL(k) Boolean grammars. This result exploits long blocks of identical symbols in the ends of the strings, in a way relatively similar to Yu's [31] non-representability argument for trellis automata.

Theorem 2. *For every Boolean LL(k) language $L \subseteq \Sigma^*$ there exist such constants $d, d' \geq 0$ and $p \geq 1$, that for all $w \in \Sigma^*$, $a \in \Sigma$, $n \geq d \cdot |w| + d'$ and $i \geq 0$,*

$$wa^n \in L \quad \text{if and only if} \quad wa^{n+ip} \in L$$

Proof. By Lemmata 2 and 5, assume that L is generated by a well-behaved LL(k) Boolean grammar $G = (\Sigma, N, P, S)$, in which, for every $A \in N$, there is either one or more rules of the form $A \rightarrow D_1 \& \dots \& D_q \& \neg E_1 \& \dots \& \neg E_r$, or a unique rule of the form $A \rightarrow BC$, $A \rightarrow a$ or $A \rightarrow \varepsilon$.

According to Corollary 1, for every $A \in N$ and $a \in \Sigma$, the set $L(A) \cap a^*$ is regular. Let $d(A, a) \geq 0$ and $p(A, a) \geq 1$ be numbers, such that $L(A) \cap a^*$ is periodic beginning from $d(A, a)$ and with the least period $p(A, a)$. Define $p = \text{lcm}_{A,a} p(A, a)$, $d_0 = \max_{A,a} d(A, a)$ and $d = \max(d_0 \cdot |N|, 1)$. Then each $L(A) \cap a^*$ is periodic beginning from d_0 with a period p .

The first claim is that if $A()$ returns on an input string of the form wa^* without seeing the end of this string, then the number of a s in its tail cannot exceed $d \cdot (|w| + 1)$. To prove this inductively, a more elaborate formulation is needed:

Claim 2.1. *If $A()$ returns on $wa^n a^t$ (with $w \in \Sigma^+$, $n \geq 0$ and $t \geq k$) consuming wa^n , and any string in $a^k a^*$ follows A , then*

$$n \leq d \cdot |w| + d_0 \cdot |\mathcal{X}|, \tag{3}$$

where $\mathcal{X} \subseteq N$ is the set of all nonterminals X , such that in the course of the computation of $A()$ on $wa^n a^t$ the procedure $X()$ is ever called on $wa^n a^t$.

The below argument shows that the parser must be matching the symbols of w to the first symbols of a^n . Consequently, if there are too many a s, then the parser will not keep count, and the procedure $A()$ would consume as many a s as there are available, until the end of the string becomes visible.

Proof. The proof is carried out by an induction on the height of the tree of recursive calls made in this computation. The base case is when $A()$ makes no recursive calls. Then the rule for A is either $A \rightarrow \varepsilon$, or $A \rightarrow b$ with $b \in \Sigma$. In either case, $n \leq 1$ and the inequality (3) holds.

Assume that $A()$ makes recursive calls and consider the rule chosen in the beginning of its computation on $wa^n a^t$. If this is a rule of the form $A \rightarrow BC$, then the execution of $A()$ begins with calling $B()$ on $wa^n a^t$, which returns, consuming a certain prefix of this string. Depending on how much it consumes, there are three cases to consider, which are illustrated in Figure 1. The first of these cases is trivial:

Case I: $B()$ consumes nothing. If $B()$ consumes ε , then $C()$ is subsequently executed on $wa^n a^t$ and consumes wa^n . This computation of C has a tree of recursive calls of a lesser height, and also any string that follows A therefore follows C . Then the induction hypothesis is applicable to this computation of $C()$, which gives $n \leq d \cdot |w| + d_0 \cdot |\mathcal{Y}|$, where the set $\mathcal{Y} \subseteq N$ contains all nonterminals Y , such that the procedure $Y()$ is ever called on $wa^n a^t$ in the course of this computation. Since, obviously, $\mathcal{Y} \subseteq \mathcal{X}$, the inequality (3) follows.

Case II: $B()$ consumes a non-empty proper prefix of w . Let $w = uv$, where $u, v \in \Sigma^+$, and $B()$ is executed on $uva^n a^t$, consuming u . Then $C()$ is executed on $va^n a^t$ and consumes va^n , as shown in Figure 1(II).

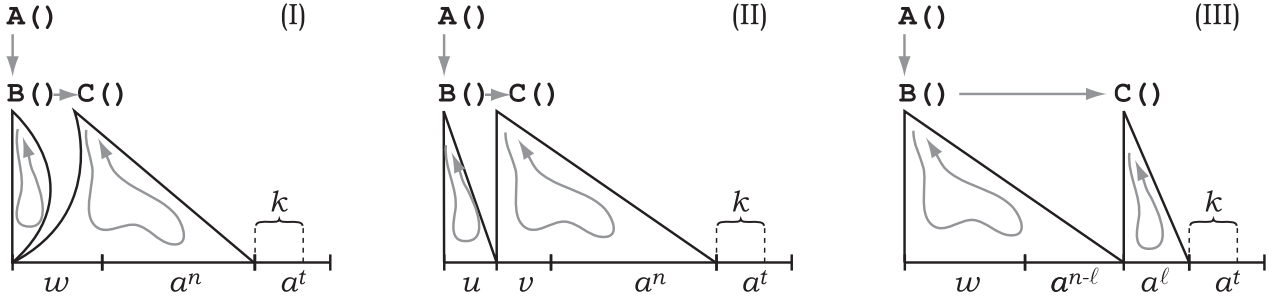


Figure 1: Proof of Theorem 2, Claim 2.1.

Consider the computation of $C()$, and let $\mathcal{Y} \subseteq N$ be the set of all nonterminals Y , such that $Y()$ is ever called on $va^n a^t$ in the course of this computation. By the induction hypothesis applied to this computation,

$$n \leq d \cdot |v| + d_0 \cdot |\mathcal{Y}| \leq d \cdot |v| + d \leq d \cdot |w|,$$

where the last inequality follows from $|v| \leq |w| - 1$. This proves

$$n \leq d \cdot |w| + d_0 \cdot |\mathcal{X}|.$$

Case III: $B()$ consumes the entire w and possibly some symbols a . Let $B()$ consume $wa^{n-\ell}$, for some ℓ with $0 \leq \ell \leq n$, as illustrated in Figure 1(III).

Let \mathcal{Y} be the set of nonterminals Y , such that $Y()$ is ever called on $wa^n a^t$ in the computation of $B()$. By the induction hypothesis applied to this computation,

$$n - \ell \leq d \cdot |w| + d_0 \cdot |\mathcal{Y}|. \quad (4)$$

Since the computation of $B()$ is a part of the computation of $A()$, $\mathcal{Y} \subseteq \mathcal{X}$ and $B \in \mathcal{X}$. On the other hand, note that $B \notin \mathcal{Y}$, because the computation would enter an infinite recursion otherwise. Therefore, $\mathcal{Y} \subseteq \mathcal{X} \setminus \{B\}$ and $|\mathcal{Y}| \leq |\mathcal{X}| - 1$, and hence (4) can be transformed as follows:

$$n \leq d \cdot |w| + d_0 \cdot |\mathcal{Y}| + \ell \leq d \cdot |w| + d_0 \cdot (|\mathcal{X}| - 1) + \ell. \quad (5)$$

Consider the computation carried out by $A()$ after $B()$ returns. Next, $C()$ is invoked on $a^\ell a^t$, and it returns, consuming a^ℓ . Since there exists a string in $a^k a^*$ that follows C , by Lemma 6, $L(C) \cap a^* = \{a^\ell\}$, that is, the regular set $L(C) \cap a^*$ is periodic beginning from $\ell + 1$. Then, by definition, $\ell < d_0$, and (5) can be further transformed to

$$n \leq d \cdot |w| + d_0 \cdot |\mathcal{X}| - (d_0 - \ell) \leq d \cdot |w| + d_0 \cdot |\mathcal{X}|, \quad (6)$$

which completes the proof of this case.

It remains to consider the case of a rule $A \rightarrow D_1 \& \dots \& D_q \& \neg E_1 \& \dots \& \neg E_r$ ($q \geq 1, r \geq 0$) being chosen by $A()$. Then the procedure $D_1()$ is called on $wa^n a^t$, and it returns, consuming wa^n . As in Case I above, the induction hypothesis is applicable to this computation, which proves (3) and establishes Claim 2.1. \square

The next statement refers to computations of $A()$ on strings wa^n , in which the end of the string is approached. It is claimed that for n large enough, such a computation may be “pumped”, that is, a string wa^{n+p} is accepted as well.

Claim 2.2. *Let $w \in \Sigma^+$ and $0 \leq t < k$ and assume that a^t follows A . Define $n_0(w, t) = d \cdot (|w| + 1) + d_0 + k - t$. For every $n \geq n_0(w, t)$, if $wa^n \in L(A)$, then $wa^{n+p} \in L(A)$; if furthermore $n \geq n_0(w, t) + p$, then $wa^{n-p} \in L(A)$.*

Note that the statement of the claim could be easily reformulated as an “if and only if” statement, but it is stated as it is, because it is proved exactly in this form. The proof analyzes the generation of wa^n , using Claim 2.1 to single out a nonterminal B producing a sufficiently long sequence of as . Then the periodicity of $L(B) \cap a^*$ is used to pump this sequence.

Proof. The proof is done by an induction on the lexicographically ordered pairs $(|w|, h(A))$, where $h(A) \geq 0$ is the “height” of A according to Definition 4. The induction is organized without an explicit basis: one of the cases in the proof—namely, case IV for a rule $A \rightarrow BC$ —does not use the induction hypothesis, and hence may be regarded as the base case. Such an argument is valid, because the induction is well-founded.

Let $|w| > 0$ and $wa^n \in L(A)$. Then there exists a rule for A that generates wa^n . This cannot be a rule of the form $A \rightarrow \varepsilon$ or $A \rightarrow b$, because $|wa^n| \geq 2$.

Assume the string wa^n is generated by a rule $A \rightarrow BC$, that is, there exists a partition of wa^n into two parts, the first one being from $L(B)$ and the second from $L(C)$. Depending on the partition, there are four cases to consider, which are illustrated in Figure 2.

Case I: B generates an empty prefix. Let $\varepsilon \in L(B)$ and $wa^n \in L(C)$. Since $h(C) \leq h(A) - 1$, the induction hypothesis can be applied to wa^n and C , which gives $wa^{n+p} \in L(C) \subseteq L(A)$ (using $\varepsilon \in L(B)$), and if $n \geq n_0(w, t) + p$, then similarly $wa^{n-p} \in L(A)$.

Case II: B generates a non-empty proper prefix of w . Let $w = uv$ with $u, v \in \Sigma^+$ and assume $u \in L(B)$ and $va^n \in L(C)$. Then $n \geq n_0(w, t) \geq n_0(v, t)$ (in the second case, $n \geq n_0(w, t) + p \geq n_0(v, t) + p$) and a^t follows C (because a^t follows A), so the induction hypothesis is applicable to va^n and C . In the first case this gives $va^{n+p} \in L(C)$, which implies $wa^{n+p} \in L(A)$; in the second case, $va^{n-p} \in L(C)$ and therefore $wa^{n-p} \in L(A)$.

Case III: B generates $wa^{n-(k-t-1)}$ or more (that is, with fewer than k symbols a ahead). Let $wa^{n-\ell} \in L(B)$ and $a^\ell \in L(C)$, where $\ell \leq k - t - 1$. Since $n \geq n_0(w, t) = d \cdot (|w| + 1) + d_0 + k - t$ by assumption, $n - \ell \geq d \cdot (|w| + 1) + d_0 + k - (\ell + t) = n_0(w, \ell + t)$. In addition, $h(B) \leq h(A) - 1$ and $a^{\ell+t}$ follows B , and hence the induction hypothesis is applicable to $wa^{n-\ell} \in L(B)$. This gives $wa^{n-\ell+p} \in L(B)$, and therefore $wa^{n+p} \in L(A)$.

In the second case, $n \geq n_0(w, t) + p$ by assumption, hence $n - \ell \geq n_0(w, \ell + t) + p$ and again $wa^{n-p} \in L(A)$.

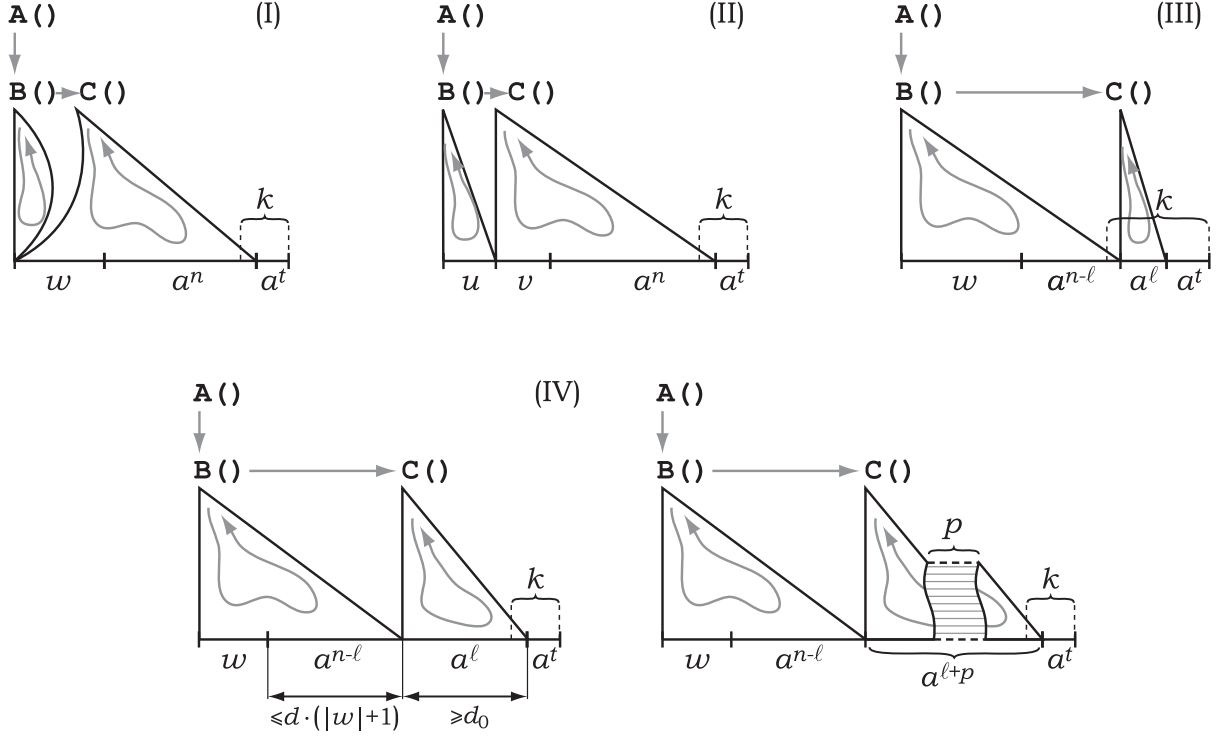


Figure 2: Proof of Theorem 2, Claim 2.2.

Case IV: B generates the whole w and at most $n - (k - t)$ symbols a (at least k of them ahead).

Assume that $wa^{n-\ell} \in L(B)$ and $a^\ell \in L(C)$, with $\ell + t \geq k$. Since a^t follows A , $a^\ell a^t$ consequently follows B , and then, by Lemma 1, $B()$ returns on $wa^{n-\ell} a^\ell a^t$, consuming $wa^{n-\ell}$.

By Claim 2.1 applied to the computation of $B()$, $n - \ell \leq d \cdot |w| + d_0 \cdot |N| \leq d \cdot (|w| + 1)$, and therefore $n \leq d \cdot (|w| + 1) + \ell$. On the other hand, $n \geq n_0(w, t) = d \cdot (|w| + 1) + d_0 + k - t$ by the assumption. Combining these inequalities gives a lower bound on ℓ as follows: $d \cdot (|w| + 1) + d_0 + k - t \leq d \cdot (|w| + 1) + \ell$, and hence $\ell \geq d_0 + k - t \geq d_0$.

Since $L(C) \cap a^*$ is periodic beginning from d_0 with period p , $a^\ell \in L(C)$ implies $a^{\ell+p} \in L(C)$. Concatenating $wa^{n-\ell} \in L(B)$ to this, one obtains $wa^{n-\ell} a^{\ell+p} = wa^{n+p} \in L(A)$.

In the second case it holds that $n \geq n_0(w, t) + p = d \cdot (|w| + 1) + d_0 + k - t + p$, and therefore $d \cdot (|w| + 1) + d_0 + k - t + p \leq d \cdot (|w| + 1) + \ell$, that is, $\ell \geq d_0 + k - t + p \geq d_0 + p$. Then, by the periodicity of $L(C) \cap a^*$ beginning from d_0 with period p , $a^\ell \in L(C)$ implies $a^{\ell-p} \in L(C)$. Finally, since $wa^{n-\ell} \in L(B)$, the string $wa^{n-\ell} a^{\ell-p} = wa^{n-p}$ is in $L(A)$ by the rule $A \rightarrow BC$.

Consider the remaining case of wa^n generated using a rule

$$A \rightarrow D_1 \& \dots \& D_q \& \neg E_1 \& \dots \& \neg E_r. \quad (7)$$

Then $wa^n \in L(D_i)$ for all i and $wa^n \notin L(E_j)$ for all j . For every D_i , $h(D_i) \leq h(A) - 1$ and a^t follows D_i ; and for every E_i , $h(E_i) \leq h(A) - 1$ and a^t follows E_i .

The first claim is that $wa^{n+p} \in L(A)$. For every positive conjunct D_i in (7), the induction hypothesis is applicable to D_i and wa^n , which gives $wa^{n+p} \in L(D_i)$. For every E_j , suppose that wa^{n+p} is in $L(E_j)$. Since $n + p \geq n_0(w, t) + p$, by the induction hypothesis (the second case), $wa^{n+p-p} = wa^n$ would be in $L(E_j)$, which would yield a contradiction. Therefore, $wa^{n+p} \notin L(E_i)$. All conjuncts of the rule (7) have thus been satisfied, and it follows that $wa^{n+p} \in L(A)$.

Consider the second case: assuming that $n \geq n_0(w, t) + p$, one has to prove that $wa^{n-p} \in L(A)$. By the induction hypothesis for D_i and wa^n (the second case), $wa^{n-p} \in L(D_i)$. Consider every E_j and suppose $wa^{n-p} \in L(E_j)$. Since $n - p \geq n_0(w, t)$ by the assumption, by the induction hypothesis (the first case), $wa^{n-p+p} = wa^n$ would be in $L(E_j)$, yielding a contradiction: hence, $wa^{n-p} \notin L(E_i)$. Again, in this last case, $wa^{n-p} \in L(A)$, and the proof of Claim 2.2 is complete. \square

Returning to the proof of Theorem 2, define $d' = d + d_0 + k$. Then Claim 2.2 is applicable to the nonterminal S , which is followed by ε , and it states that for every $n \geq d \cdot |w| + d'$, $wa^n \in L_G(S)$ holds if and only if $wa^{n+p} \in L_G(S)$, Q. E. D.

Theorem 2 implies that languages of the form $\{a^n b^{f(n)} \mid n \geq 1\}$ generated by $LL(k)$ Boolean grammars must have a linearly bounded f .

Corollary 2. *If a language of the form $\{a^n b^{f(n)} \mid n \geq 1\}$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ is an integer function, is Boolean LL, then the function $f(n)$ is bounded by $C \cdot n$ for some constant $C \geq 1$.*

Proof. By Theorem 2, there exist such constants $d, d' \geq 0$ and $p \geq 1$, that for every string $a^n b^\ell \in L$ with $\ell \geq dn + d'$, the string $a^n b^{\ell+p}$ is in L as well. Since, for every a^n , the language L contains only one string of the form $a^n b^*$, this condition should never hold, that is, for every $a^n b^\ell \in L$, the number ℓ should be less than $dn + d'$. In other words, $f(n) < dn + d' \leq (d + d')n$ for all $n \geq 1$, and setting $C = d + d'$ establishes the claim. \square

This, in particular, proves that Boolean recursive descent parsers cannot handle the linear conjunctive language mentioned in the beginning of this section.

Example 5. *The linear conjunctive language $\{a^n b^{2^n} \mid n \geq 0\}$ is not Boolean LL(k) for any k .*

7. Linear conjunctive LL grammars

Consider the family of languages generated by linear conjunctive grammars satisfying the definition of an $LL(k)$ grammar. A grammar of this kind for the non-context-free language $\{a^n b^n c^n \mid n \geq 0\}$ was given in Example 1. In addition to this simple example, it is worth note that these grammars can specify such an important language as the language of computations of a Turing machine [18], and hence their expressive power is far from being trivial. However, it turns out that some very simple languages are beyond their scope.

The following theorem shows that there is no $LL(k)$ linear conjunctive grammar for any non-regular language followed by one unspecified symbol, which may be a or b . The reason is that a parser for a language of this form cannot know the last symbol of the input string beforehand, and it cannot infer it from any preceding symbols, because there is no connection between them. This prevents the parser from applying any rules that would mean committing to one particular final symbol, and under such restrictions, the grammar may generate only a regular language.

Theorem 3. *Let Σ be an alphabet, let $a, b \in \Sigma$ ($a \neq b$). Then, for every $L \subseteq \Sigma^*$, the language $L \cdot \{a, b\}$ is linear conjunctive LL if and only if L is regular.*

Proof. The proof in one direction is trivial: if L is regular, then so is $L \cdot \{a, b\}$, and a finite automaton for the latter language can be transcribed as an $LL(1)$ linear context-free grammar.

The goal is to prove that an $LL(k)$ linear conjunctive grammar for $L \cdot \{a, b\}$ can be effectively transformed to a finite automaton for L . Let $G = (\Sigma, N, P, S)$ be an $LL(k)$ linear conjunctive grammar with $L(G) = L \cdot \{a, b\}$, let $T : N \times \Sigma^{\leq k} \rightarrow P$ be a parsing table.

The main idea of the argument is that as long as a procedure $B()$ cannot see the end of the input in the beginning of the computation (that is, it is outside of the range of the lookahead), it must read the entire input. Otherwise it would have to decide in the beginning whether the last symbol is a or b , which cannot be done before seeing this last symbol.

Claim 3.1. *Let $w \in L$ and $s \in \{a, b\}$. If the successful computation of the parser on ws contains a call to $B()$ on a suffix yz , with $ws = xyz$ and $|yz| \geq k + 1$, which returns, consuming y , then $z = \varepsilon$ and ε follows B .*

Proof. The proof is an induction on the length of the path in the tree of recursive calls connecting the root to the call to $B()$. The induction hypothesis consists of the statement of Claim 3.1 along with one more statement. Define a function $f : \Sigma^*\{a, b\} \rightarrow \Sigma^*\{a, b\}$ as $f(ua) = ub$ and $f(ub) = ua$ for every $u \in \Sigma^*$; now it is additionally claimed that the successful computation of the parser on $f(ws) \in L$ contains a call to $B()$ on the suffix $f(yz)$.

Basis: path of length 0. Here $B = S$ and $S()$ returns on ws , consuming ws , that is, $x = z = \varepsilon$. Then ε follows S by definition. Obviously, the computation of the parser on $f(ws)$ begins with a call to $S()$ on $f(ws)$.

Induction step. Suppose $B()$ returns on yz , consuming y . Consider the procedure $A()$, from which this call to $B()$ is made. There are factorizations $x = x'u$ and $z = vz'$, such that $A()$ is called on $uyvz'$, and it returns, consuming uyv . Since the path to A in the tree of recursive calls is shorter than the path to B , by the induction hypothesis, ε follows A and $z' = \varepsilon$, hence $v = z$. Then $A()$ returns on uyz , consuming uyz . The form of the computation is illustrated in Figure 3.

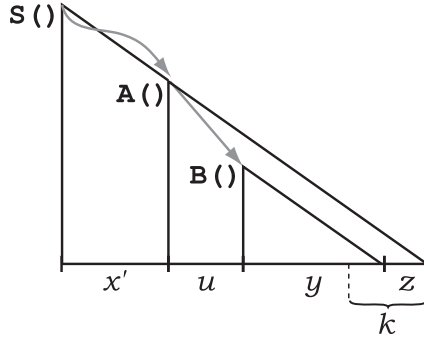


Figure 3: Proof of Theorem 3, Claim 3.1, induction step.

Consider the rule chosen by $A()$ in its computation on uyz , which is

$$T_k(A, First_k(uyz)) = A \rightarrow \dots \&uBz\& \dots, \quad (8)$$

where the conjunct uBz is the conjunct corresponding to the invocation of $B()$. By Lemma 1, $uyz \in L(uBz)$, that is, $y \in L(B)$.

On the other hand, the induction hypothesis asserts that the computation of the parser on $f(ws)$ contains a call to $A()$ on the suffix $f(uyz)$. Since $|uyz| \geq |yz| \geq k + 1$, it is known that $First_k(f(uyz)) = First_k(uyz)$. Then this computation begins with choosing the same rule (8). By the assumption, this computation eventually reaches the conjunct uBz and calls $B()$. After returning from this procedure, $A()$ reads the last $|z|$ characters of $f(uyz)$, which must match z . Then z must be a suffix of $f(uyz)$.

Thus it has been obtained that z is a common suffix of a string ending with a and a string ending with b . Therefore, $z = \varepsilon$. Then, by the conjunct uBz in (8), ε follows B . This completes the proof of Claim 3.1. \square

Now define a new grammar $G' = (\Sigma, N, P', S)$ as follows. Let m be the greatest number of terminal symbols in a conjunct in P . Every rule of the form $A \rightarrow u_1B_1\& \dots \&u_nB_n$ in P , with $u_i \in \Sigma^*$ and $B_i \in N$, is included in P' as well. In addition, for every $A \in N$ and for every $w \in L_G(A)$ with $|w| \leq k + m$, the set P' contains a rule $A \rightarrow w$. The task is now to prove that $L(G') = L(G)$.

Claim 3.2. *For every $A \in N$, $L_{G'}(A) \subseteq L_G(A)$.*

This claim easily follows from the fact that every rule in $P' \setminus P$ is of the form $A \rightarrow y \in P'$ defined above, and $y \in L_G(A)$ for every such rule. The converse inclusion is established in the following, by analyzing computations of a parser.

Claim 3.3. *Let $w \in L$ and $s \in \{a, b\}$ and consider the recursive descent parser for G . If its successful computation on ws contains a call to $A()$ on a suffix yz (with $ws = xyz$), which returns, consuming y , and z follows A , then $y \in L_{G'}(A)$.*

The general reasoning for this claim can be summarized as follows: if y is sufficiently short, it is generated by a rule $A \rightarrow y$, and if y is long enough, then Claim 3.1 is applicable, and it implies that y is derived using a rule of the form $A \rightarrow u_1B_1\&\dots\&u_nB_n$.

Proof. Let h be the height of the tree of recursive calls in the computation of $A()$. The proof is an induction on the lexicographically ordered pairs $(|y|, h)$.

Basis: $|y| \leq k + m$. Then $A \rightarrow y \in P'$, which proves the claim.

Induction step. Let $|y| > k + m$ and consider the call to $A()$ on yz , which returns, consuming y ; it should be proved that $y \in L_{G'}(A)$. The computation of $A()$ begins with choosing a rule

$$T_k(A, \text{First}_k(y)) = A \rightarrow u_1B_1v_1\&\dots\&u_nB_nv_n, \quad (9)$$

such that $y \in L_G(u_iB_iv_i)$ for every i . Let $y = u_ix_iv_i$, where $x_i \in L_G(B_i)$ and v_iz follows B_i . Then $A()$ eventually calls $B_i()$ on x_iv_iz , and it returns, consuming x_i .

By the definition of m , $|u_iv_i| \leq m$, and therefore $|x_i| > k$. Applying Claim 3.1 to the computation of $B_i()$ gives $v_iz = \varepsilon$. Then the rule (9) is in fact of the form

$$A \rightarrow u_1B_1\&\dots\&u_nB_n, \quad (9')$$

and is therefore in P' .

On the other hand, applying the induction hypothesis to the computation of $B_i()$ gives $x_i \in L_{G'}(B_i)$, and hence $y \in L_{G'}(u_iB_iv_i)$. Therefore, $y \in L_{G'}(A)$ by the rule (9'), which completes the proof of Claim 3.3. \square

Claim 3.4. $L(G') = L(G)$.

Proof. By Claim 3.2, $L_{G'}(S) \subseteq L_G(S)$. To establish the converse inclusion, consider any string $ws \in L(G)$. The successful computation of the parser on ws contains a call to $S()$ on the suffix ws , which returns, consuming ws , and this, according to Claim 3.3, implies $ws \in L_{G'}(S) = L(G')$. \square

It has thus been shown that the language $L \cdot \{a, b\}$ is generated by a conjunctive grammar G' with one-sided concatenation, and therefore it is regular. Hence, L is regular as well, Q. E. D.

The above theorem implies non-representability of some very simple languages, such as the following:

Example 6. *The linear context-free languages $\{a^n b^n cs \mid n \geq 0, s \in \{a, b\}\}$ and $\{a^n b^n s \mid n \geq 0, s \in \{a, b\}\}$ are not linear conjunctive LL.*

8. Linear Boolean LL grammars

Linear Boolean grammars are known to have the same expressive power as linear conjunctive grammars [15, 16]. In contrast, their LL subsets differ in power, as the first language from Example 6, which is not representable by any $\text{LL}(k)$ linear conjunctive grammar due to Theorem 3, has a simple $\text{LL}(1)$ linear Boolean grammar given below. This grammar relies on de Morgan's laws to represent a union of languages via conjunction and negation.

Example 7. The following LL(1) linear Boolean grammar generates the language $\{a^n b^n c s \mid n \geq 0, s \in \{a, b\}\}$:

$$\begin{aligned} S &\rightarrow X \& \neg T \\ T &\rightarrow X \& \neg Aca \& \neg Acb \\ A &\rightarrow aAb \mid \varepsilon \\ X &\rightarrow aX \mid bX \mid cX \mid \varepsilon \end{aligned}$$

One can actually adapt this construction to show that every language of the form $L \cdot \{a, b\}$, where L has an LL(k) linear Boolean grammar, is generated by an LL($k + 1$) linear Boolean grammar.

Still, the general idea behind Theorem 3—that of appending a fixed regular language R to a given language L , so that an LL parser, not knowing which element of R is at the end of the input string, would not be able to recognize any non-regular structure in L —this idea can be adapted for linear Boolean grammars. The next theorem demonstrates that a non-regular language followed by a tail c^* of unspecified length cannot be defined by such a grammar.

Theorem 4. Let Σ be an alphabet, let $c \in \Sigma$. Then, for every language $L \subseteq \Sigma^*$, the language Lc^* has an LL linear Boolean grammar if and only if L is regular.

The proof of the theorem begins with simplifying the grammar down to a linear conjunctive grammar of a certain special form.

Lemma 7. Every language generated by a strongly non-left-recursive linear Boolean grammar is generated by a strongly non-left-recursive linear conjunctive grammar, which can be effectively constructed.

Proof. The proof is by a direct transformation of the given grammar $G = (\Sigma, N, P, S)$, which is similar to the known construction for complementing a linear conjunctive grammar [14].

Construct a linear conjunctive grammar $G' = (\Sigma, N \cup N' \cup \widehat{N} \cup \widetilde{N}, P', S)$, where $N' = \{A' \mid A \in N\}$, $\widehat{N} = \{Z_{uAv} \mid G \text{ has a conjunct } \pm uAv\} \cup \{Y_\varphi \mid \text{there is a rule } A \rightarrow \varphi \text{ in } P\}$ and \widetilde{N} contains a few nonterminals generating regular languages. The goal of the construction is to have

$$L_{G'}(A) = L_G(A), \quad (10a)$$

$$L_{G'}(A') = \overline{L_G(A)}, \quad (10b)$$

$$L_{G'}(Z_{uAv}) = \overline{L_G(uAv)}, \quad (10c)$$

$$L_{G'}(Y_\varphi) = \overline{L_G(\varphi)}. \quad (10d)$$

For every rule

$$A \rightarrow u_1 B_1 v_1 \& \dots \& u_m B_m v_m \& \neg x_1 C_1 y_1 \& \dots \& \neg x_n C_n y_n \quad (11)$$

in the original grammar, the new grammar contains the rules

$$\begin{aligned} &A \rightarrow u_1 B_1 v_1 \& \dots \& u_m B_m v_m \& Z_{x_1 C_1 y_1} \& \dots \& Z_{x_n C_n y_n}, \\ Y_{u_1 B_1 v_1 \& \dots \& u_m B_m v_m \& \neg x_1 C_1 y_1 \& \dots \& \neg x_n C_n y_n} &\rightarrow Z_{u_1 B_1 v_1} \mid \dots \mid Z_{u_m B_m v_m} \mid x_1 C_1 y_1 \mid \dots \mid x_n C_n y_n, \end{aligned}$$

while every rule $A \rightarrow w$ in G is retained in G' , and also accompanied by a nonterminal Y_w generating $\Sigma^* \setminus \{w\}$.

For every nonterminal $A \in N$, let $A \rightarrow \varphi_1 \mid \dots \mid \varphi_\ell$ be all its rules in the original grammar. Then the new grammar contains the following rule for A' :

$$A' \rightarrow Y_{\varphi_1} \& \dots \& Y_{\varphi_\ell}.$$

For every conjunct uBv in the original grammar, the new grammar contains the rules

$$Z_{uBv} \rightarrow uB'v \mid X_{u,v},$$

where $X_{u,v} \in \widetilde{N}$ is an extra nonterminal generating the regular language $\overline{u\Sigma^*v}$.

The correctness of the construction is established by showing that (I) the grammar G' is strongly non-left-recursive, and hence the corresponding system of language equations has a unique solution, and that (II) every solution of the system of language equations corresponding to G' induces a solution of the system corresponding to G , so that these solutions coincide on the variables from N .

Consider that for linear Boolean grammars, the definition of left recursivity gets much simpler: a left recursion is just a sequence of rules

$$\begin{aligned} A_1 &\rightarrow \dots \& \pm A_2 w_1 \& \dots \\ A_2 &\rightarrow \dots \& \pm A_3 w_2 \& \dots \\ &\vdots \\ A_n &\rightarrow \dots \& \pm A_1 w_n \& \dots \end{aligned} \tag{12}$$

Suppose, for the sake of contradiction, that the first claim does not hold, that is, there is a left recursion of the form (12) in G' . It is claimed that there is a corresponding cycle of rules in G , witnessing its left recursivity and thus leading to a contradiction.

The supposed left-recursive cycle (12) in G' cannot entirely avoid the nonterminals from $N \cup N'$, so consider any rules for these nonterminals occurring in the cycle, and the conjuncts, through which the cycle proceeds. Every time a nonterminal $A \in N$ is used in the cycle in G' , the left recursion either directly proceeds to a nonterminal $B \in N$ using a rule $A \in \dots \& Bv \& \dots$ (which corresponds to a rule $A \in \dots \& Bv \& \dots$ in G), or first goes to a nonterminal Z_{Cy} by a rule $A \in \dots \& Z_{Bv} \& \dots$ (corresponding to a rule $A \in \dots \& \neg Bv \& \dots$ in G'), from whence it can only proceed to B' by the rule $Z_{Bv} \rightarrow B'v$. In other words, left recursion in G' proceeds from $A \in N$ to $B \in N$ or to $B' \in N'$, and in both cases, left recursion in G may proceed from A to B . Similarly, one can observe that from $A' \in N'$, left recursion in G' must proceed to Y_φ and then either to xBy , or to Z_{uBv} and then to B' , and in either case, left recursion may go from A to B in G . Thus a cycle (12) in G' induces a cycle in G of the same general form.

Since G' is non-left-recursive, the corresponding system of language equations has a unique solution, and it is left to show that it is related to the unique solution of G . This is checked by substituting the intended solution (10) into the expressions formed by the rules of G' , and verifying that the equations turn to equalities. For example, the right-hand side of the equation for each $Z_{uAv} \in \widehat{N}$ in G' evaluates to

$$uL_{G'}(A')v \cup \overline{u\Sigma^*v} = \overline{uL_G(A)v} \cup \overline{u\Sigma^*v} = \overline{uL_G(A)v} = L_{G'}(Z_{uAv}),$$

and the rest of the equations are checked similarly. \square

Next, a non-left-recursive linear conjunctive grammar is transformed into a normal form akin to the Greibach normal form for the context-free grammars.

Lemma 8. *Every non-left-recursive linear conjunctive grammar can be effectively transformed to a grammar with all rules of the form*

$$\begin{aligned} A &\rightarrow aB_1v_1 \& \dots \& aB_mv_m && (a \in \Sigma, m \geq 1, B_i \in N, v_i \in \Sigma^*) \\ A &\rightarrow w && (w \in \Sigma^*) \end{aligned}$$

Sketch of a proof. Let $G = (\Sigma, N, P, S)$ be a non-left-recursive linear conjunctive grammar, and let $h \geq 0$ be its height, that is, the length of the longest sequence of rules

$$\begin{aligned} A_1 &\rightarrow \dots \& A_2 w_1 \& \dots \\ A_2 &\rightarrow \dots \& A_3 w_2 \& \dots \\ &\vdots \\ A_h &\rightarrow \dots \& A_{h+1} w_h \& \dots \end{aligned} \tag{13}$$

If $h = 0$, then every non-empty conjunct in the grammar begins with a terminal symbol, and transforming the grammar to the desired form is a straightforward exercise. It remains to show that if the height is positive, then there is an equivalent grammar of a smaller height.

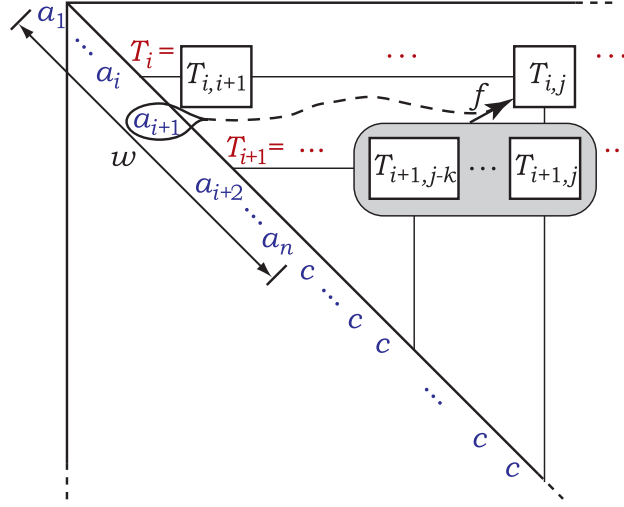


Figure 4: The parsing table T , and the dependence of $T_{i,j}$ on $T_{i+1,j-k}, \dots, T_{i+1,j}$ and a_{i+1} .

Consider the system of language equations corresponding to G , and denote it by $X = \varphi(X)$, where X is a vector of unknown languages, and φ is a vector of the right-hand sides of the equations. Since the grammar is linear, one can construct another linear conjunctive grammar with the same set of nonterminals and with the corresponding system of language equations $X = \varphi(\varphi(X))$, which is done generally by substituting all rules into all rules in all possible combinations. The resulting grammar $G^2 = (\Sigma, N, P', S)$ generates the same language as G , because concatenation of singletons is distributive over intersection. Furthermore, its height is at most $\lfloor \frac{h}{2} \rfloor$, because every sequence of rules (13) in G^2 is obtained from a sequence in G by doing every two steps as a single step. Applying this transformation $\lfloor \log_2 h \rfloor + 1$ times leads to a desired grammar of height 0. \square

Finally, once LL(k) Boolean grammars are reduced to grammars in this normal form, there is a convenient proof of their limitations in generating languages with long tails of identical letters.

Lemma 9. *Let $L \subseteq \Sigma^*$ and let $c \notin \Sigma$. Then, if the language Lc^* is generated by a linear conjunctive grammar with all rules of the form*

$$\begin{aligned} A &\rightarrow aB_1v_1 \& \dots \& aB_mv_m & (a \in \Sigma, m \geq 1, B_i \in N, v_i \in \Sigma^*), \\ A &\rightarrow w & (w \in \Sigma^*), \end{aligned}$$

then L is regular.

Proof. The proof proceeds by constructing a DFA for the reversal of L . For every string $w \in \Sigma^*$, this DFA shall recognize w^R by simulating a parse of a string in wc^* with a sufficiently long tail of c 's.

Let $w = a_1 \dots a_n$ with $a_i \in \Sigma$ be any string. For every string $w' = wc^\ell$, with $\ell \geq 1$, consider its standard parsing table according to the grammar G , that is, a $|w'| \times |w'|$ table of subsets of N , in which every element $T_{i,j}$ contains the set of nonterminals of G that generate the substring of wc^ℓ beginning with its $(i+1)$ -th symbol and ending with the j -th. Taking the c -tail to be unbounded, one can define an infinite parsing table as follows. Let $a_i = c$ for every $i \geq n+1$. For all $0 \leq i < j$, let $w_{i,j} = a_{i+1} \dots a_j$ be the substring of the infinite string $wc^\omega = wccc \dots$ ranging from position $i+1$ to position j . The corresponding entry of the parsing table is

$$T_{i,j} = \{A \in N \mid w_{i,j} \in L(A)\}.$$

Let k be the greatest length of the strings v across all conjuncts of the form aBv in the grammar, and consider any substring $w_{i,j}$, with $j \geq n+k$, and with $j-i$ greater than k and greater than the longest

right-hand side of a rule $A \rightarrow w$ in the grammar. Then a nonterminal A is in $T_{i,j}$, if there is such a rule $A \rightarrow aB_1v_1\&\dots\&aB_mv_m$, with $a = a_{i+1}$ and $v_1, \dots, v_m \in c^*$, that each nonterminal B_t belongs to the corresponding element $T_{i+1,j-|v_t|}$. Accordingly, $T_{i,j}$ functionally depends on the sets $T_{i+1,j-k}, \dots, T_{i+1,j}$ and on the symbol a_{i+1} by the following formula, which is illustrated in Figure 4:

$$T_{i,j} = f(a_{i+1}, T_{i+1,j-k}, \dots, T_{i+1,j}) = \{A \mid \exists A \rightarrow aB_1v_1\&\dots\&aB_mv_m \in P : a = a_{i+1}, v_t \in c^* \text{ and } B_t \in T_{i+1,j-|v_t|} \text{ for all } t\}.$$

Consider each i -th line of the parsing table, with $i \in \{0, \dots, n-1, n\}$, as an infinite sequence of subsets of N :

$$T_i = T_{i,i+1}, T_{i,i+2}, T_{i,i+3}, \dots$$

In particular, the n -th line consists of the sets of nonterminals generating the strings c, c^2, c^3, \dots . Since the grammar is linear, the unary languages $L(A) \cap c^*$ are known to be regular for all $A \in N$. Let $p \geq 1$ be their least common period. Then $c^n \in L(A)$ if and only if $c^{n+p} \in L(A)$ for all n large enough and for all $A \in N$, and thus the sequence of sets T_n is ultimately periodic with period p .

It is claimed that the lines T_{n-1}, \dots, T_1, T_0 are also ultimately periodic with period p . This is proved inductively on the number of the line, with the line T_n as the base case. For the induction step, assume that T_{i+1} has period p . Every $T_{i,j}$, with j large enough, functionally depends on $T_{i+1,j-k}, \dots, T_{i+1,j}$ and a_{i+1} , and the set $T_{i,j+p}$ is obtained from $T_{i+1,j-k+p}, \dots, T_{i+1,j+p}$ and a_{i+1} by the same formula. By the periodicity of T_{i+1} , the arguments of this formula are equal, as long as j is large enough, and therefore

$$T_{i,j+p} = f(a_{i+1}, T_{i+1,j-k+p}, \dots, T_{i+1,j+p}) = f(a_{i+1}, T_{i+1,j-k}, \dots, T_{i+1,j}) = T_{i,j}.$$

A finite automaton recognizing L^R remembers the periodic part of such a sequence of sets in its internal state, and calculates the functional dependence of the periodic part of T_i on the periodic part of T_{i+1} in its transitions. Let $Q = (2^N)^p$ be its set of states, and let the initial state q_0 represent the periodic part of the line of the parsing table corresponding to the strings c, c^2, c^3, \dots , which is

$$q_0 = (\{A \mid c^d \in L(A)\}, \{A \mid c^{d+1} \in L(A)\}, \dots, \{A \mid c^{d+p-1} \in L(A)\}),$$

for some d large enough. Then the transition from a state $(s_0, s_1, \dots, s_{p-1})$ by a symbol a is defined by

$$\delta((s_0, s_1, \dots, s_{p-1}), a) = (s'_0, s'_1, \dots, s'_{p-1}), \quad \text{where} \\ s'_t = f(a, s_{t-k \pmod p}, \dots, s_{t-1 \pmod p}, s_t).$$

A state $(s_0, s_1, \dots, s_{p-1})$ is accepting if $S \in s_0$.

Given a string w^R of length n as an input, after reading each prefix $a_n a_{n-1} \dots a_i$ of this string, the automaton will compute the periodic part of the line T_i in the parsing table for the infinite string wc^ω . Once w^R is processed entirely, the automaton knows the set of nonterminals that generate a string wc^ℓ , for some value of ℓ . Regardless of its actual value, $wc^\ell \in L_G(S)$ if and only if $w \in L$, and hence the input w^R is accepted if and only if w is in L , as desired. \square

The theorem is now established as a consequence of the above lemmata.

Proof of Theorem 4. Let Lc^* have a linear Boolean LL(k) grammar G . Then G is strongly non-left-recursive, and hence, by Lemma 7, there is a non-left-recursive conjunctive grammar G' generating the same language. The latter grammar is transformed, according to Lemma 8, to a grammar G'' in a normal form, and finally Lemma 9 asserts that the language L is regular.

Conversely, if L is regular, then so is Lc^* , and hence it has a linear Boolean LL(1) grammar. \square

Using Theorem 4, one can show non-representability of such languages as the following one.

Example 8. *The language $\{a^n b^n c^\ell \mid n, \ell \geq 0\}$ is not LL linear Boolean.*

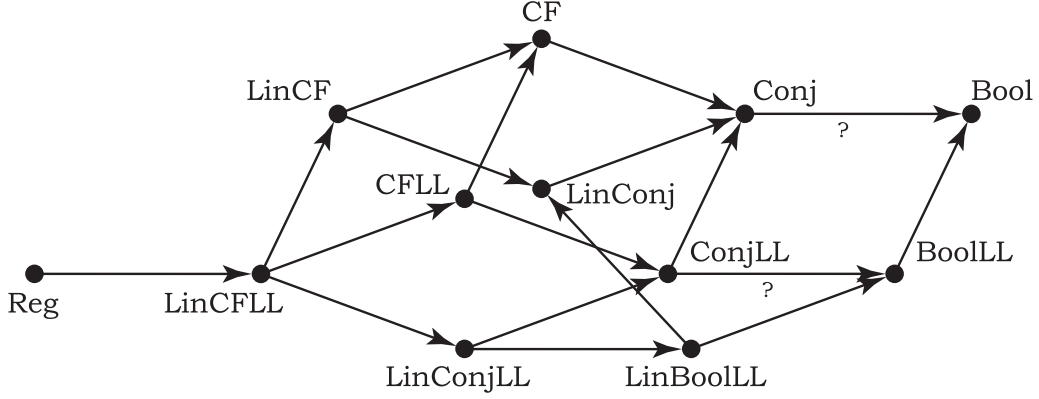


Figure 5: Expressive power of subfamilies of Boolean grammars.

It must be noted that the earlier *ad hoc* proof of Example 8, presented in the preliminary version of this paper [20, Lemma 7], was wrong.

To conclude this section, consider another example of the expressive power of LL linear Boolean grammars, which shows one more technique for representing languages by such grammars.

Example 9. *The LL(1) linear Boolean grammar*

$$\begin{aligned} S &\rightarrow aS\&\neg A \mid bB \\ A &\rightarrow aAb \mid \varepsilon \\ B &\rightarrow bB \mid \varepsilon \end{aligned}$$

generates the language $\{a^m b^n \mid 0 \leq m < n\}$.

All generated strings are in $a^* b^*$. For a string $a^m b^n$, the rule $S \rightarrow aS\&\neg A$ ensures that no proper suffix of this string is in $L(A) = \{a^\ell b^\ell \mid \ell \geq 0\}$, which holds if and only if $m < n$.

9. Hierarchy

The results of Sections 5–8 allow drawing a detailed comparison between different subfamilies of $LL(k)$ Boolean grammars and the basic subfamilies of Boolean grammars. These families are denoted as follows: *Bool*, *Conj* and *CF* are the families of languages generated by Boolean, conjunctive and standard context-free grammars, respectively; *BoolLL*, *ConjLL* and *CFL* refer to the families of languages generated by the $LL(k)$ subsets of these grammars. The languages generated by linear conjunctive grammars are denoted by *LinConj*, while linear Boolean grammars are known to generate the same family [15, 16]; their $LL(k)$ subfamilies are different and are denoted by *LinConjLL* and *LinBoolLL*, respectively. Finally, *LinCF* stands for the set of linear context-free languages, *LinCFL* denotes their $LL(k)$ subset, and *Reg* is the family of regular languages.

It is known that *LinConj* and *CF* are incomparable supersets of *LinCF* [28, 15], and that both are proper subsets of *Conj*. Furthermore, $Conj \subseteq Bool$, but it is not known whether this inclusion is proper [16]. The following theorem establishes the inclusions involving the LL subfamilies.

Theorem 5.

- I. $LinCFL \subset LinConjLL$, with $\{a^n b^n c^n \mid n \geq 0\} \in LinConjLL \setminus LinCFL$.
- II. $LinCFL \subset CFL$, with $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\} \in CFL \setminus LinCFL$.
- III. $LinConjLL \subset ConjLL$, with $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\} \in ConjLL \setminus LinConjLL$.
- IV. $CFL \subset ConjLL$, with $\{a^n b^n c^n \mid n \geq 0\} \in ConjLL \setminus CFL$.
- V. $LinConjLL \subset LinBoolLL$, with $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\} \in LinBoolLL \setminus LinConjLL$.

- VI. $\text{LinBoolLL} \subset \text{BoolLL}$, with $\{a^n b^n c^\ell \mid n, \ell \geq 0\} \in \text{BoolLL} \setminus \text{LinBoolLL}$.
- VII. $\text{LinCFLL} \subset \text{LinCF}$, with $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\} \in \text{LinCF} \setminus \text{LinCFLL}$.
- VIII. $\text{LinBoolLL} \subset \text{LinConj}$, with $\{a^n b^n c^\ell \mid n, \ell \geq 0\} \in \text{LinConj} \setminus \text{LinBoolLL}$.
- IX. $\text{CFLL} \subset \text{CF}$, with $a^+ \cup \{a^n b^n \mid n \geq 0\} \in \text{CF} \setminus \text{CFLL}$.
- X. $\text{ConjLL} \subset \text{Conj}$, with $\{a^n b^{2^n} \mid n \geq 1\} \in \text{Conj} \setminus \text{ConjLL}$.
- XI. $\text{BoolLL} \subset \text{Bool}$, with $\{a^n b^{2^n} \mid n \geq 1\} \in \text{Bool} \setminus \text{BoolLL}$.

Proof. The inclusion (VIII) holds, because every linear Boolean grammar has an equivalent linear conjunctive grammar [16], and hence $\text{LinBoolLL} \subseteq \text{LinBool} = \text{LinConj}$; see also Lemma 7 for a direct proof of this inclusion. The rest of the inclusions are immediate. It remains to argue that in each of these cases, the given language separates the respective families.

I. The language $\{a^n b^n c^n \mid n \geq 0\}$ is in LinConjLL by Example 1, and it is not in LinCFLL , because it is not context-free.

II. The language $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\}$ is CFLL , because it is generated by the following LL(1) context-free grammar:

$$\begin{aligned} S &\rightarrow AcX \\ A &\rightarrow aAb \mid \varepsilon \\ X &\rightarrow a \mid b \end{aligned}$$

On the other hand, this language is not in LinConjLL according to Example 6, and hence not in LinCFLL .

III. As shown above, $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\}$ is in CFLL and hence in ConjLL . Example 6 states that it is not in LinConjLL .

IV. The non-context-free language $\{a^n b^n c^n \mid n \geq 0\}$ cannot be in CFLL , but, on the other hand, it is in ConjLL due to Example 1.

V. As shown in Example 7, the language $\{a^n b^n cs \mid n \geq 0, s \in \{a, b\}\}$ belongs to LinBoolLL . However, it is not in LinConjLL , according to Example 6.

VI. The language $\{a^n b^n c^\ell \mid n, \ell \geq 0\}$ is in CFLL , and hence in BoolLL , because of the following LL(1) context-free grammar:

$$\begin{aligned} S &\rightarrow AC \\ A &\rightarrow aAb \mid \varepsilon \\ C &\rightarrow cC \mid \varepsilon \end{aligned}$$

On the other hand, as stated in Example 8, it is not in LinBoolLL .

VII. The language $\{a^n b^n c \mid n \geq 0\} \cdot \{a, b\}$ is in LinCF as a concatenation of a linear context-free language $\{a^n b^n \mid n \geq 0\}$ and a regular language $\{ca, cb\}$. It lies outside of LinCFLL due to Example 6.

VIII. The language $\{a^n b^n c^\ell \mid n, \ell \geq 0\}$ is linear context-free as a concatenation of a linear context-free language $\{a^n b^n \mid n \geq 0\}$ and a regular language c^* , and therefore it belongs to the larger family LinConj . It is not in LinBoolLL by Example 8.

IX. The language $a^+ \cup \{a^n b^n \mid n \geq 0\}$ is obviously in CF . However, no LL(k) context-free grammar generates this language [27].

X. The language $\{a^n b^{2^n} \mid n \geq 1\}$ is generated by a linear conjunctive grammar [5, 15], hence it is in Conj . On the other hand, as stated in Example 5, it is not in ConjLL .

XI. As in the previous case, $\{a^n b^{2^n} \mid n \geq 1\}$ is in Bool , but, due to Example 5, is not in BoolLL . \square

The resulting inclusion diagram is given in Figure 5, in which arrows with a question mark denote inclusions not known to be proper, the rest being proper. It remains to compare all (seventeen) pairs of families in the diagram that are not connected by a chain of inclusions, which is done in the next theorem.

Theorem 6.

- I. Both LinConjLL and LinBoolLL are incomparable with the following families: CFLL , LinCF , CF .
- II. LinCF is incomparable with CFLL .
- III. LinConj is incomparable with CF , CFLL , ConjLL and BoolLL .
- IV. Both ConjLL and BoolLL are not contained in LinCF and CF .

V. *ConjLL* is not contained in *LinBoolLL*.

VI. *Conj* is not contained in *BoolLL*.

Proof. I. The families *LinConjLL* and *LinBoolLL* each contain the non-context-free language $\{a^n b^n c^n \mid n \geq 0\}$, see Example 1, which is not in *CFL*, *LinCF* and *CF*. On the other hand, the latter three families contain the language $\{a^n b^n c^\ell \mid n, \ell \geq 0\}$, see the proof of Theorem 5(VI,VIII). This language does not belong to *LinBoolLL* by Example 8.

II. The language $a^+ \cup \{a^n b^n \mid n \geq 1\}$ is in *LinCF* as a union of two linear context-free languages; on the other hand, Rosenkrantz and Stearns [27] proved that this language is not context-free $LL(k)$ for any k . Another language $\{a^m b^m a^n b^n \mid m, n \geq 0\}$ is generated by the following context-free $LL(1)$ grammar:

$$\begin{aligned} S &\rightarrow AA \\ A &\rightarrow aAb \mid \varepsilon \end{aligned}$$

However, it is well-known to be not in *LinCF*, see, e.g., Berstel [3, Ex. 6.3].

III. The incomparability of *LinConj* and *CF* is a known result by Terrier [28]; while the families *LinConj* and *CFL* were recently proved incomparable by the author [24] by a similar method. The other two cases are handled using the same argument, which is as follows. On the one hand, the family *LinConj* contains the language $\{a^n b^{2^n} \mid n \geq 0\}$, which cannot be in *CFL*, *ConjLL* or *BoolLL* according to Example 5, and which is well-known to be not in *CF*. Conversely, as proved by the author [24], there exists a language

$$L = \{c^m a^{\ell_0} b \dots a^{\ell_{m-1}} b a^{\ell_m} b \dots a^{\ell_k} b d^n \mid m, n, \ell_i \geq 0, k \geq 1, \ell_m = n\}$$

generated by an $LL(1)$ context-free grammar

$$\begin{aligned} S &\rightarrow CD \\ C &\rightarrow cCAb \mid \varepsilon \\ A &\rightarrow aA \mid \varepsilon \\ D &\rightarrow aDd \mid bB \\ B &\rightarrow aB \mid bB \mid \varepsilon \end{aligned}$$

and this language is not linear conjunctive.

IV. The language $\{a^n b^n c^n \mid n \geq 0\}$ is in *ConjLL* and *BoolLL* by Example 1, but it is not in *LinCF* and not in *CF* because it is non-context-free.

V. The language $\{a^n b^n c^i \mid n, i \geq 0\}$ is in *ConjLL*, as demonstrated in the proof of Theorem 5(VI). However, it is not in *LinBoolLL* by Example 8.

VI. This part directly follows from part III, which states that already *LinConj* is not a subset of *BoolLL*. The witness language is $\{a^n b^{2^n} \mid n \geq 0\}$. \square

The results of Theorem 6(I–III) completely settle the relation between the corresponding families. On the other hand, parts IV–VI of Theorem 6 claim weaker results, which imply the distinctness of the given families, but do not rule out proper inclusions in the other direction.

Consider each case. The statement of part IV leaves open the following possible inclusions:

$$LinCF \stackrel{?}{\subseteq} ConjLL, \quad LinCF \stackrel{?}{\subseteq} BoolLL, \quad CF \stackrel{?}{\subseteq} ConjLL, \quad CF \stackrel{?}{\subseteq} BoolLL.$$

The weakest of them, stating the containment of *LinCF* in *BoolLL*, would imply linear-time parsing of every linear context-free language, which is a fairly unlikely result. The stronger ones would mean that every context-free language can be parsed in linear time, which, according to the current knowledge, is hardly possible. The problem of separating these families is accordingly proposed for future research.

Another uncertainty is about the inclusion

$$LinBoolLL \stackrel{?}{\subseteq} ConjLL$$

from Theorem 6(V). Can every linear Boolean LL grammar be rewritten without using negation, using unrestricted concatenation instead, and at the same time maintaining the LL property? The poor state of knowledge on the family *ConjLL* does not allow making any conjectures at this time.

Finally, consider Theorem 6(VI), which leaves open whether

$$\text{BoolLL} \stackrel{?}{\subseteq} \text{Conj}$$

holds true. Here the question is, whether negation in $\text{LL}(k)$ grammars is weak enough to be expressed using grammars without negation, possibly losing the LL property. This question might be related to the problems $\text{ConjLL} \stackrel{?}{=} \text{BoolLL}$ and $\text{Conj} \stackrel{?}{=} \text{Bool}$ left open in Theorem 5.

10. Closure properties

It is known from Rosenkrantz and Stearns [27] and Wood [30] that context-free LL languages (*CFL*) are not closed under union, intersection, complementation, concatenation (already with regular languages), star and reversal. That is, for each of these operations one can construct context-free LL arguments, so that the resulting language is not in *CFL*.

Consider the closure properties of the four new families: *BoolLL*, *ConjLL*, *LinBoolLL* and *LinConjLL*. Each of them is closed under intersection, while *BoolLL* and *LinBoolLL* are closed under all Boolean operations. This is established by the following straightforward construction:

Lemma 10. *Let \mathcal{G} be any of the following four families of grammars: Boolean $\text{LL}(k)$, conjunctive $\text{LL}(k)$, linear Boolean $\text{LL}(k)$ and linear conjunctive $\text{LL}(k)$. Then, for every two grammars $G_1, G_2 \in \mathcal{G}$, there exists and can be effectively constructed a grammar $G \in \mathcal{G}$ generating the language $L(G_1) \cap L(G_2)$.*

If \mathcal{G} is the family of Boolean $\text{LL}(k)$ or linear Boolean $\text{LL}(k)$ grammars, then, furthermore, there exist and can be effectively constructed grammars from \mathcal{G} for the languages $L(G_1) \cup L(G_2)$ and $\overline{L(G_1)}$.

Sketch of a proof. Let $G_i = (\Sigma, N_i, P_i, S_i)$ with $i = 1, 2$ be the two given grammars, assume $N_1 \cap N_2 = \emptyset$. Construct the grammar $G = (\Sigma, N_1 \cup N_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 \& S_2\}, S)$, where $S \notin N_1, N_2$. Clearly, G is strongly non-left-recursive and $\text{LL}(k)$, and $L(G) = L(G_1) \cap L(G_2)$.

Now let G_1 be a Boolean $\text{LL}(k)$ or linear Boolean $\text{LL}(k)$ grammar. Then the grammar $G = (\Sigma, N_1 \cup \{S, A\}, P_1 \cup \{S \rightarrow A \& \neg S_1, A \rightarrow \varepsilon\} \cup \{A \rightarrow aA \mid a \in \Sigma\}, S)$ is $\text{LL}(k)$ and $L(G) = \Sigma^* \setminus L(G_1)$. The closure under union follows by de Morgan's laws. \square

The following lemmata present witness languages for the non-closure of some families under Kleene star, reversal and union.

Lemma 11. *The language $L = \{a^n b^n \mid n \geq 0\} \cup c^*$ is linear context-free $\text{LL}(1)$, while L^* is not linear Boolean $\text{LL}(k)$ for any k .*

Proof. The $\text{LL}(1)$ linear context-free grammar for L contains the rules $S \rightarrow A$, $S \rightarrow cC$, $A \rightarrow aAb$, $A \rightarrow \varepsilon$, $C \rightarrow cC$ and $C \rightarrow \varepsilon$. Suppose L^* is linear Boolean $\text{LL}(k)$. Then, by Lemma 10, $L^* \cap a^* b^* c^* = \{a^n b^n c^i \mid n, i \geq 0\}$ is linear Boolean $\text{LL}(k)$ as well, which contradicts Example 8. \square

Lemma 12. *The language $L = c^* \{b^n a^n \mid n \geq 0\}$ is linear context-free $\text{LL}(1)$, while L^R is not linear Boolean $\text{LL}(k)$ for any k .*

Proof. The language L is generated by an $\text{LL}(1)$ linear context-free grammar with the following rules: $S \rightarrow cS$, $S \rightarrow A$, $A \rightarrow bAa$ and $A \rightarrow \varepsilon$. However, $L^R = \{a^n b^n \mid n \geq 0\} c^*$ is not linear Boolean $\text{LL}(k)$, as stated in Example 8. \square

Lemma 13. *The languages $L_1 = \{a^n b^n c a \mid n \geq 0\}$ and $L_2 = \{a^n b^n c b \mid n \geq 0\}$ are both $\text{LL}(1)$ linear context-free, while their union is not linear conjunctive $\text{LL}(k)$ for any k .*

Proof. The language L_1 (L_2 , respectively) is generated by an LL(1) linear context-free grammar with the rules $S \rightarrow Aca$ ($S \rightarrow Acb$, respectively), $A \rightarrow aAb$ and $A \rightarrow \varepsilon$. Their union $L_1 \cup L_2$ is the language from Example 6, which is not linear conjunctive LL. \square

The non-closure of some families under concatenation will be proved in a stronger form: that a concatenation with a regular language on the right or on the left leads out of these families.

Lemma 14. *The concatenation of an LL(1) linear context-free language $\{a^n b^n \mid n \geq 0\}$ and a regular language c^* is not linear Boolean LL(k) for any k .*

Proof. This concatenation is the language $\{a^n b^n c^i \mid n, i \geq 0\}$, which is not linear Boolean LL due to Example 8. \square

Lemma 15. *The concatenation of a finite language $\{\varepsilon, b\}$ with an LL(1) linear context-free language $L = \{a^n b^n a \mid n \geq 1\} \cup \{ba^n b^{n+1} \mid n \geq 1\}$ is not linear conjunctive LL(k) for any k .*

Proof. The language L is generated by the following LL(1) linear context-free grammar:

$$\begin{aligned} S &\rightarrow aAba \mid baAbb \\ A &\rightarrow aAb \mid \varepsilon \end{aligned}$$

Consider the concatenation

$$\{\varepsilon, b\}L = \{a^n b^n a \mid n \geq 1\} \cup \{ba^n b^{n+1} \mid n \geq 1\} \cup \{ba^n b^n a \mid n \geq 1\} \cup \{bba^n b^{n+1} \mid n \geq 1\}.$$

If it were linear conjunctive LL(k), then, by Lemma 10, the intersection $\{\varepsilon, b\}L \cap ba^+b^+\{a, b\}$ would be linear conjunctive LL(k) as well. However, since

$$\{\varepsilon, b\}L \cap ba^+b^+\{a, b\} = \{ba^n b^n a \mid n \geq 1\} \cup \{ba^n b^n b \mid n \geq 1\} = \{ba^n b^n \mid n \geq 1\} \cdot \{a, b\},$$

by Theorem 3 this would imply that $\{ba^n b^n \mid n \geq 1\}$ is a regular language. The contradiction obtained proves the lemma. \square

Yet another interesting operation is *cyclic shift*, $\text{SHIFT}(L) = \{vu \mid uv \in L\}$, which is notable for preserving context-free languages [12, 26]. However, it preserves none of the linear LL families.

Lemma 16. *The language $L = c^*\{a^n b^n \mid n \geq 0\}$ is LL(1) linear context-free, while its cyclic shift is not linear Boolean LL(k) for any k .*

Proof. The language L is generated by an LL(1) linear context-free grammar similar to the one in Lemma 12. Suppose its cyclic shift is linear Boolean LL(k). Then so is its intersection with $a^*b^*c^*$, which is the language from Example 8. \square

Besides the standard language-theoretic operations, consider a less common operation. This is the logical dual of concatenation [21], defined as follows:

$$K \odot L = \{w \mid \text{for every factorization } w = uv \text{ it holds that } u \in K \text{ or } v \in L\} = \overline{\overline{K} \cdot \overline{L}}.$$

This operation naturally occurs whenever concatenation and complementation are used together, such as in Boolean grammars. However, note that the operation itself is defined without using negation, and that it is *monotone*, in the sense that $K \subseteq K'$ and $L \subseteq L'$ imply $K \odot L \subseteq K' \odot L'$.

A noteworthy property of this operation is the closure of conjunctive grammars under dual concatenation with regular constants [21, Th. 11]. An analogue of this result for LL subfamilies of Boolean grammars will now be established.

Lemma 17. *Let \mathcal{G} be any of the following four families of grammars: Boolean $LL(k)$, conjunctive $LL(k)$, linear Boolean $LL(k)$ and linear conjunctive $LL(k)$. Then, for every grammar $G \in \mathcal{G}$ and for every finite automaton M , there exists and can be effectively constructed a grammar $G' \in \mathcal{G}$ generating the language $L(M) \odot L(G)$.*

If \mathcal{G} is the family of Boolean $LL(k)$ or linear Boolean $LL(k)$ grammars, then, furthermore, there exist and can be effectively constructed grammars from \mathcal{G} for the language $L(M) \cdot L(G)$.

Sketch of a proof. Let $G = (\Sigma, N, P, S)$ be a grammar from \mathcal{G} . Let $M = (\Sigma, Q, q_0, \delta, F)$ be a deterministic finite automaton, in which Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \rightarrow Q$ is a total transition function and $F \subseteq Q$ is the set of accepting states. As in the proof for conjunctive grammars of the general form [21, Th. 11], construct a new grammar $G' = (\Sigma, N \cup \{T_q \mid q \in Q\}, P \cup P', T_{q_0})$, where the rules in P' are:

$$T_q \rightarrow aT_{\delta(q,a)} \quad (\text{for all } q \in F \text{ and } a \in \Sigma) \quad (14a)$$

$$T_q \rightarrow aT_{\delta(q,a)} \& S \quad (\text{for all } q \notin F \text{ and } a \in \Sigma) \quad (14b)$$

$$T_q \rightarrow \varepsilon \quad (\text{for all } q \in F) \quad (14c)$$

$$T_q \rightarrow \varepsilon \& S \quad (\text{for all } q \notin F) \quad (14d)$$

Then it is known from the cited paper [21] that for every string $w \in \Sigma^*$ and for every state $q \in Q$, the string w is in $L_{G'}(T_q)$ if and only if for every its factorization $w = uv$, it holds that $\delta(q, u) \in F$ or $v \in L(G)$. In particular, $L(G') = L(M) \odot L(G)$.

The construction preserves the $LL(k)$ property. Indeed, the form of the new rules (14a,14b), which begin with terminal symbols, ensures that no left recursion can occur, and since the rules for each T_q have distinct first symbols, the $LL(k)$ condition is maintained.

Turning to the second statement of the lemma, a grammar for $L(M) \cdot L(G)$ can be obtained according to the representation $L(M) \cdot L(G) = \overline{L(M)} \odot \overline{L(G)}$, where the language $\overline{L(M)}$ is regular, the language $\overline{L(G)}$ has an LL linear Boolean grammar by Theorem 10, their dual concatenation is regular by the above arguments, and its complement is LL linear Boolean by another application of Theorem 10. \square

Lemma 18. *Let $\Sigma = \{a, b\}$. The dual concatenation of an $LL(1)$ linear context-free language $\{a^n b^n \mid n \geq 1\}$ with a co-finite language $\{a, b\}$ is not linear conjunctive $LL(k)$ for any k .*

Proof. By the definition of dual concatenation, $\{a^n b^n \mid n \geq 1\} \odot \overline{\{a, b\}}$ is the set of all strings $w \in \{a, b\}^*$, such that for every factorization $w = uv$ it holds that $u = a^n b^n$ for some $n \geq 1$ or $v \notin \{a, b\}$. The latter condition is true as long as $|v| \neq 1$. If $|v| = 1$, the condition is equivalent to $u \in \{a^n b^n \mid n \geq 1\}$.

Therefore,

$$\{a^n b^n \mid n \geq 1\} \odot \overline{\{a, b\}} = \{\varepsilon\} \cup \{a^n b^n \mid n \geq 1\} \cdot \{a, b\}.$$

If this language were linear conjunctive $LL(k)$, then so would be $\{a^n b^n \mid n \geq 1\} \cdot \{a, b\}$, by the closure under intersection with Σ^+ . As this is the language from Example 6, for which no linear conjunctive $LL(k)$ grammar exists, this proves the lemma. \square

All basic closure properties of *LinConjLL* and *LinBoolLL* follow from Lemmata 10–18 and are summarized in the following two theorems.

Theorem 7. *The family of LL linear conjunctive languages is closed under intersection and right-dual-concatenation of regular languages. It is not closed under union, complementation, concatenation with regular languages, left-dual-concatenation of regular languages, Kleene star, reversal and cyclic shift.*

Note that the non-closure of *LinConjLL* under complementation is implied by its closure under intersection and its non-closure under union.

Theorem 8. *The family of LL linear Boolean languages is closed under all Boolean operations, left-concatenation and left-dual-concatenation of regular languages. It is not closed under right-concatenation or right-dual-concatenation of regular languages, under Kleene star, reversal and cyclic shift.*

Here the non-closure under dual concatenation of regular languages from the right follows from the closure under complementation and the non-closure under concatenation of regular languages from the right, by the same argument as in the last claim of Lemma 17.

The closure properties of LL context-free languages are known from Rosenkrantz and Stearns [27] and Wood [30]. It remains to investigate cyclic shift and dual concatenation with regular languages.

Lemma 19. *The language $L = \{a^n b^n \mid n \geq 0\} \cup \{ba^n b^{2n-1} \mid n \geq 1\}$ is LL(1) linear context-free, while its cyclic shift is not deterministic context-free, and hence not LL(k) context-free for any k .*

Proof. The language L is generated by a grammar with the rules $S \rightarrow A$, $S \rightarrow baBb$, $A \rightarrow aAb$, $A \rightarrow \varepsilon$, $B \rightarrow aBbb$, $B \rightarrow \varepsilon$. If its cyclic shift is deterministic context-free, then so is the language $\text{SHIFT}(L) \cap a^*b^* = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$, which is a known non-deterministic context-free language [1, Sect. 5.2]. \square

Lemma 20. *Let $\Sigma = \{a, b\}$. The dual concatenation of an LL(1) context-free language $L = \{a^n b^{n-1} \mid n \geq 1\}$ with a regular language $R = a^*b^+b \cup a^* \cup \{ab\}$ is not context-free LL(k) for any k .*

Proof. Consider the language $(L \odot R) \cap \Sigma^*a$. If a string wa with $w \in \Sigma^*$ is in $L \odot R$, then, for the factorization $wa = \varepsilon \cdot wa$, either $\varepsilon \in L$ (which is false) or $wa \in R$, which implies $wa \in a^+$. Conversely, every string in a^+ is in $L \odot R$, because all of its suffixes are in R . Therefore, $(L \odot R) \cap \Sigma^*a = a^+$.

Next, consider $(L \odot R) \cap \Sigma^*b$, and let $wb \in L \odot R$ with $w \in \Sigma^*$ be any string in this language. As above, the factorization $wb = \varepsilon \cdot wb$ in the definition of the dual concatenation implies that wb must be in R ; in this case, in a^*b^+b or in $\{ab\}$. So let $wb = a^i b^j b$. Since $b \notin R$, for the factorization $wb = a^i b^j \cdot b$, the string $a^i b^j$ must be in L , that is, the equality $i = j + 1$ must hold. On the other hand, every string $a^n b^{n-1} b$ is in $L \odot R$, because all of its suffixes, except for b , are in R , while for the factorization $a^n b^{n-1} \cdot b$, the prefix $a^n b^{n-1}$ is in L . This proves that $(L \odot R) \cap \Sigma^*b = \{a^n b^n \mid n \geq 1\}$.

Finally, $\varepsilon \in R$ implies $\varepsilon \in L \odot R$, and it has thus been proved that $L \odot R = \{a^n b^n \mid n \geq 1\} \cup a^*$. This language is known not to be context-free LL [27]. \square

Lemma 21. *Let $\Sigma = \{a, b, c\}$. The dual concatenation of a co-finite language $\overline{\{\varepsilon, b\}}$ with an LL(1) linear context-free language $L = \{ba^n b^i c^n \mid i, n \geq 1\} \cup \{a^i b^n c^n \mid i, n \geq 1\}$ is not context-free.*

Proof. The language L is generated by an LL(1) linear context-free grammar with the rules $S \rightarrow baAc$, $S \rightarrow aT$, $T \rightarrow aT$, $T \rightarrow bBc$, $A \rightarrow aAc$, $A \rightarrow bC$, $B \rightarrow bBc$, $B \rightarrow \varepsilon$, $C \rightarrow bC$ and $C \rightarrow \varepsilon$. It is known [21, Lemma 4] that such a dual concatenation equals

$$\overline{\{\varepsilon, b\}} \odot L = L \cap (bL \cup \overline{b\Sigma^*}).$$

The latter could be rewritten as follows:

$$\begin{aligned} (\{ba^n b^i c^n \mid i, n \geq 1\} \cup \{a^i b^n c^n \mid i, n \geq 1\}) \cap (\{bba^n b^i c^n \mid i, n \geq 1\} \cup \{ba^i b^n c^n \mid i, n \geq 1\} \cup \overline{b\Sigma^*}) = \\ = \{ba^n b^n c^n \mid n \geq 1\} \cup \{a^i b^n c^n \mid i, n \geq 1\}. \end{aligned}$$

This is clearly a non-context-free language. \square

The last operation to be considered is intersection with a regular language. The non-closure of context-free LL(k) languages under this operation is known from Wood [30], who presented the following example: the language $L = \{a^n s a^n t \mid n \geq 0, s, t \in \{b, c\}\}$ is LL(1) context-free, while $L \cap (a^* b a^* b \cup a^* c a^* c) = \{a^n b a^n b \mid n \geq 0\} \cup \{a^n c a^n c \mid n \geq 0\}$ is a known non-LL context-free language. However, that language L is not in *LinCFL* by Theorem 3, and so this example does not settle the case of linear context-free LL languages. In fact, it is rather easy to prove the contrary, that is, that linear context-free LL(k) languages are closed under intersection with regular languages.

Lemma 22. *Let $G = (\Sigma, N, P, S)$ be an LL(k) linear context-free grammar and let $M = (\Sigma, Q, q_0, \delta, F)$ be a deterministic finite automaton. Consider the linear context-free grammar $G' = (\Sigma, N', P', S')$, in which*

	\cap Reg	\cup	\cap	\sim	\cdot Reg/Reg \cdot	\odot Reg/Reg \odot	\cdot	$*$	R	SHIFT
Reg	+	+	+	+	+	+	+	+	+	+ [12]
LinCFLL	+ L22	- L13	- E1	- [27]	L14 -/- L15	L18 -/- L21	- L14	- L11	- L12	- L16
CFLL	- [30]	- [27]	- [27]	- [27]	[30] -/- [30]	L20 -/- L21	- [27]	- [30]	- [27]	- L19
LinConjLL	+	- L13	+ L10	- T7	L14 -/- L15	L18 -/+ L17	- L14	- L11	- L12	- L16
LinBoolLL	+	+ L10	+ L10	+ L10	L14 -/+ L17	T8 -/+ L17	- L14	- L11	- L12	- L16
ConjLL	+	?	+ L10	?	?/?	?/+ L17	?	?	?	?
BoolLL	+	+ L10	+ L10	+ L10	?/+ L17	?/+ L17	?	?	?	?
LinCF	+	+	-	-	+	- [21]	-	-	+	-
CF	+	+	-	-	+	- [21]	+	+	+	+ [12, 26]
LinConj	+	+	+	+ [15]	+	+	- [28]	- [15]	+	- [29]
Conj	+	+	+	?	+	+ [21]	+	+	+	?
Bool	+	+	+	+	+	+	+	+	+	?

Table 1: Closure properties.

$N' = N \times Q \times 2^Q$, and a nonterminal $(A, q, R) \in N'$ is denoted by $A_{q,R}$. Its start symbol is $S' = S_{q_0,F}$, and the set P' contains the following rules:

$$\begin{aligned} A_{q,R} &\rightarrow u B_{\delta(q,u),\delta^{-1}(R,v)} v & (A \rightarrow uBv \in P, q \in Q, R \subseteq Q) \\ A_{q,R} &\rightarrow w & (A \rightarrow w \in P, q \in Q, \delta(q,w) \in R \subseteq Q), \end{aligned}$$

where $\delta^{-1}(R, v) = \{q \mid \delta(q, v) \in R\}$. Then G' is $LL(k)$ and

$$L_{G'}(A_{q,R}) = L_G(A) \cap \{w \mid \delta(q, w) \in R\}$$

for all $A \in N$, $q \in Q$ and $R \subseteq Q$. In particular, $L(G') = L(G) \cap L(M)$.

Sketch of a proof. Each nonterminal A in G is represented by multiple nonterminals of the form $A_{q,R}$ in G' , and the rules of G' simulate the corresponding rules of G , while keeping track of the behaviour of M in the subscripts. The proof that the languages generated by nonterminals $A_{q,R}$ are as stated is carried out by a standard induction on the number of steps in a context-free rewriting, which is omitted. Then one can directly show that if a string $z \in \Sigma^*$ follows a nonterminal $A_{q,R}$ by a certain sequence of rules in P' , then z follows A in G , by the sequence of the corresponding rules in P . By similar arguments, the grammar G' is non-left-recursive, because so is G (as a left recursion in G' would imply a left recursion in G going through the corresponding nonterminals).

It remains to verify the $LL(k)$ condition for G' . Let $T_k : N \times \Sigma^{\leq k} \rightarrow P$ be any $LL(k)$ table for G , and construct an $LL(k)$ table $T'_k : N' \times \Sigma^{\leq k} \rightarrow P'$ for G' by defining

$$T'_k(A_{q,R}, x) = A_{q,R} \rightarrow u B_{\delta(q,u),\delta^{-1}(R,v)} v, \quad \text{where } T_k(A, x) = A \rightarrow uBv.$$

To see that this is a correct table for G' , consider any nonterminal $A_{q,R}$ in G' and any rule $A_{q,R} \rightarrow u B_{q',R'} v \in P'$, with $q' = \delta(q, u)$ and $R' = \delta^{-1}(R, v)$, which originates from a rule $A \rightarrow uBv \in P$. Let w be any string in $L_{G'}(u B_{q',R'} v)$ and let z be any string that follows $A_{q,R}$ in G' . The goal is to prove that $T'_k(A_{q,R}, First_k(wz)) = A_{q,R} \rightarrow u B_{q',R'} v$.

The conditions on w and z translate to the grammar G as follows. Since $L_{G'}(B_{q',R'}) \subseteq L_G(B)$, the membership of w in $L_{G'}(u B_{q',R'} v)$ implies that $w \in L_G(uBv)$. The string z follows A in G . Therefore, $T_k(A, First_k(wz))$ contains the rule $A \rightarrow uBv$. Then, by the construction of T'_k , the entry $T'_k(A_{q,R}, First_k(wz))$ must contain the rule $A_{q,R} \rightarrow u B_{q',R'} v$. \square

Another property of linear context-free LL languages is their non-closure under complementation. It follows from a general result of Rosenkrantz and Stearns [27, Cor. 4], which states that the complement of a non-regular context-free LL language is never context-free LL.

All known closure properties of Boolean grammars and their main subclasses are put together in Table 1.

11. Conclusion

The new method of proving non-representability by $LL(k)$ Boolean grammars showed some limitations of their expressive power, as compared to conjunctive and Boolean grammars of the general form. However, it was not sufficient to establish their conjectured non-closure under concatenation, star and reversal.

For the subfamilies of $LL(k)$ linear conjunctive grammars and $LL(k)$ linear Boolean grammars, even stronger limitations were established. These results were sufficient not only to separate these families from each related family of formal languages, but also to prove all basic closure properties.

No non-representability methods specific for $LL(k)$ conjunctive languages were developed, and therefore this family could not have been separated from $LL(k)$ Boolean grammars. This is one of the open problems raised by this study. Another question that could not be answered is whether such an important abstract language generated by Boolean grammars as $\{wcv \mid w \in \{a, b\}^*\}$ is Boolean $LL(k)$.

Finally, while this paper establishes the first negative results for the LL subclass of Boolean grammars, it remains to invent a method of proving languages to be non-representable by Boolean grammars of the general form. The lack of such a method is the most significant gap in the present knowledge on Boolean grammars.

Acknowledgement

The author is grateful to the anonymous reviewers for careful reading and numerous valuable comments, and particularly for noticing a serious error in the earlier *ad hoc* proof of Example 8.

References

- [1] J. Autebert, J. Berstel, L. Boasson, "Context-free languages and pushdown automata", in: Rozenberg, Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, Berlin, 1997, 111–174.
- [2] F. Baader, A. Okhotin, "Complexity of language equations with one-sided concatenation and all Boolean operations", *20th International Workshop on Unification (UNIF 2006, Seattle, USA, August 11, 2006)*, 59–73.
- [3] J. Berstel, *Transductions and Context-Free Languages*, BG Teubner, Stuttgart, 1979.
- [4] K. Culik II, "Variations of the firing squad problem and applications", *Information Processing Letters*, 30 (1989), 153–157.
- [5] O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.
- [6] A. Jež, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.
- [7] A. Jež, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Theory of Computing Systems*, 46:1 (2010), 27–58.
- [8] A. Jež, A. Okhotin, "Complexity of equations over sets of natural numbers", *Theory of Computing Systems*, 48:2 (2011), 319–342.
- [9] A. Jež, A. Okhotin, "One-nonterminal conjunctive grammars over a unary alphabet", *Theory of Computing Systems*, to appear.
- [10] T. Jurdzinski, K. Loryś, "Lower bound technique for length-reducing automata", *Information and Computation*, 205:9 (2007), 1387–1412.
- [11] V. Kountouriotis, Ch. Nomikos, P. Rondogiannis, "Well-founded semantics for Boolean grammars", *Information and Computation*, 207:9 (2009), 945–967.
- [12] A. N. Maslov, "Cyclic shift operation for languages", *Problems of Information Transmission*, 9 (1973), 333–338.
- [13] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
- [14] A. Okhotin, "On the closure properties of linear conjunctive languages", *Theoretical Computer Science*, 299:1–3 (2003), 663–685.
- [15] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *Informatique Théorique et Applications*, 38:1 (2004), 69–88.
- [16] A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1 (2004), 19–48.
- [17] A. Okhotin, "Generalized LR parsing algorithm for Boolean grammars", *International Journal of Foundations of Computer Science*, 17:3 (2006), 629–664.
- [18] A. Okhotin, "Language equations with symmetric difference", *Computer Science in Russia (CSR 2006, St. Petersburg, Russia, June 8–12, 2006)*, LNCS 3967, 292–303.
- [19] A. Okhotin, "Recursive descent parsing for Boolean grammars", *Acta Informatica*, 44:3–4 (2007), 167–189.
- [20] A. Okhotin, "Expressive power of $LL(k)$ Boolean grammars", *Fundamentals of Computation Theory (FCT 2007, Budapest, Hungary, 27–30 August 2007)*, LNCS 4639, 446–457.

- [21] A. Okhotin, “Notes on dual concatenation”, *International Journal of Foundations of Computer Science*, 18:6 (2007), 1361–1370.
- [22] A. Okhotin, “Fast parsing for Boolean grammars: a generalization of Valiant’s algorithm”, *Developments in Language Theory (DLT 2010, London, Ontario, Canada, August 17–20, 2010)*, LNCS 6224, 340–351.
- [23] A. Okhotin, “A simple P-complete problem and its language-theoretic representations”, *Theoretical Computer Science*, 412:1–2 (2011), 68–82.
- [24] A. Okhotin, “Comparing linear conjunctive languages to subfamilies of the context-free languages”, *SOFSEM 2011: Theory and Practice of Computer Science (Nový Smokovec, Slovakia, 22–28 January 2011)*, LNCS 6543, 431–443.
- [25] A. Okhotin, C. Reitwießner, “Conjunctive grammars with restricted disjunction”, *Theoretical Computer Science*, 411:26–28 (2010), 2559–2571.
- [26] T. Oshiba, “Closure property of the family of context-free languages under the cyclic shift operation”, *Transactions of IECE*, 55D (1972), 119–122.
- [27] D. J. Rosenkrantz, R. E. Stearns, “Properties of deterministic top-down grammars”, *Information and Control*, 17 (1970), 226–256.
- [28] V. Terrier, “On real-time one-way cellular array”, *Theoretical Computer Science*, 141 (1995), 331–335.
- [29] V. Terrier, “Closure properties of cellular automata”, *Theoretical Computer Science*, 352:1–3 (2006), 97–107.
- [30] D. Wood, “A further note on top-down deterministic languages”, *Computer Journal*, 14:4 (1971), 396–403.
- [31] S. Yu, “A property of real-time trellis automata”, *Discrete Applied Mathematics*, 15:1 (1986), 117–119.