TUCS

Alexander Okhotin

Expressive power of LL(k) Boolean grammars

Turku Centre *for* Computer Science

TUCS Technical Report
No 823, June 2007

Expressive power of LL(k) Boolean grammars

Alexander Okhotin

Academy of Finland, *and*
Department of Mathematics, University of Turku, Turku, Finland, *and*
Turku Centre for Computer Science
alexander.okhotin@utu.fi

TUCS Technical Report
No 823, June 2007

Abstract

The family of languages generated by Boolean grammars and usable with the recursive descent parsing is studied. It is demonstrated that Boolean LL languages over a unary alphabet are regular, while Boolean LL subsets of $\Sigma^* a^*$ obey a certain periodicity property, which, in particular, makes the language $\{a^n b^{2^n} \,|\, n \geqslant 0\}$ nonrepresentable. It is also shown that $\{a^n b^n cs \,|\, n \geqslant 0,\ s \in \{a, b\}\}$ is not generated by any linear conjunctive LL grammar, while linear Boolean LL grammars cannot generate $\{a^n b^n c^* \mid n \geqslant 0\}$.

Keywords: Boolean grammars, conjunctive grammars, language equations, parsing, recursive descent

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

Boolean grammars [6] are an extension of the context-free grammars, in which the rules may contain explicit Boolean operations. While context-free grammars can combine syntactical conditions using only disjunction (effectively specified by multiple rules for a single symbol), Boolean grammars additionally allow conjunction and negation. The extended expressive power of Boolean grammars and their intuitive clarity make them a much more powerful tool for specifying languages than the context-free grammars. Another important fact is that 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 [6, 8, 7].

Though the practical properties of Boolean grammars seem to be as good as in the case of the more restricted context-free grammars, theoretical questions for Boolean grammars present a greater challenge. Already a formal definition of Boolean grammars involves certain theoretical problems [3, 6]. A major gap in the knowledge on these grammars is the lack of methods of proving nonrepresentability of languages [6]. Even though the family generated by Boolean grammars has been proved to be contained in $\mathit{DTIME}(n^3) \cap \mathit{DSPACE}(n)$, there is still no proof that any context-sensitive language lies outside of this class.

Results of the latter kind are hard to obtain for many interesting classes of automata and grammars. Consider the family of *trellis automata*, also known as one-way real-time cellular automata, which were studied since 1970s, and which have recently been proved to be equal in power to a subclass of Boolean grammars [5]. No methods of establishing nonrepresentability of languages in this family were known for two decades, until the first such result by Yu [12], who established a pumping lemma for a special case. Only a decade later the first context-free language not recognized by these automata was found by Terrier [11]. Another example is given by the growing context-sensitive languages, for which a method of proving nonrepresentability was discovered by Jurdzinski and Loryś [2].

The purpose of this paper is to establish some limitations of the expressive power of the subcase of Boolean grammars to which the recursive descent parsing is applicable: the *LL(k) Boolean grammars* [7]. Already for this class, obtaining nonrepresentability proofs presents a challenge: consider that there exists an LL(1) linear conjunctive grammar for the language of computations of any Turing machine, which rules out a general pumping lemma. There also exists an LL(1) Boolean grammar for a P-complete language [10], which shows computational nontriviality of this class. This paper proposes several methods for proving nonrepresentability 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 de-

scent 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. Section 6 considers subsets of $\Sigma^* a^*$ representable by Boolean LL grammars and establishes a periodicity property of such languages, which, in particular, implies nonrepresentability of the language $\{a^n b^{2^n} \mid n \geqslant 0\}$. Stronger nonrepresentability 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.

2 Boolean grammars and their non-left-recursive subset

Definition 1. [6] *A Boolean grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols respectively; P is a finite set of rules of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \tag{1}$$

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

Let us further assume that $m \geqslant 1$ and $n \geqslant 0$ in every rule (1). Note that if $m = 1$ and $n = 0$ in every such rule, then a context-free grammar is obtained. An intermediate family of *conjunctive grammars* [4] has $m \geqslant 1$ and $n = 0$ in every rule. *Linear* subclasses of Boolean, conjunctive and 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 $A \to \alpha_i$ and $A \to \neg\beta_j$ (for all i, j) are called conjuncts, positive and negative respectively, and α_i and β_j are their bodies. Let $conjuncts(P)$ be the set of all conjuncts. The notation $A \to \pm\alpha_i$ and $A \to \pm\beta_j$ is used to refer to a positive or a negative conjunct with the specified body.

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 rather nontrivial 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

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right] \tag{2}$$

The vector $(\ldots, L_G(A), \ldots)$ of languages generated by the nonterminals of the grammar is defined by a solution of this system. Since this system, in

general, may have no solutions or multiple solutions, this definition requires more precise conditions, which have been a subject of research [3, 6].

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 the following strong requirement.

Definition 2. [7] *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar. The relation of context-free 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 $\alpha, \beta, \gamma, \delta \in (\Sigma \cup N)^*$, $A \in N$ and for all conjuncts $A \to \pm\eta\sigma\theta$. Denote its reflexive and transitive closure by $\rightsquigarrow^*$ and its transitive closure by $\rightsquigarrow^+$.

Definition 3. [7] *Let $G = (\Sigma, N, P, S)$ be Boolean grammar, define the corresponding conjunctive grammar $G_+ = (\Sigma, N, P_+, S)$ by removing all negative conjuncts from every rule in G, that is, for every rule (1) in P, P_+ contains the rule $A \to \alpha_1 \& \ldots \& \alpha_m$.*

Then the Boolean grammar G is said to be strongly non-left-recursive if and only if for all $A \in N$ and $\theta, \eta \in (\Sigma \cup N)^*$, such that $\varepsilon\langle A\rangle\varepsilon \rightsquigarrow^+ \theta\langle A\rangle\eta$, it holds that $\varepsilon \notin L_{G_+}(\theta)$,

In a strongly non-left-recursive grammar it is possible to define the height of a nonterminal A, denoted $h(A)$, as the greatest number of steps in a derivation $\varepsilon\langle A\rangle\varepsilon \rightsquigarrow^* \theta\langle B\rangle\eta$, where $\varepsilon \in L_{G_+}(\theta)$ and $B \in N$.

For every strongly non-left-recursive grammar, the corresponding system of equations (2) has a unique solution [7]. Then, for every $A \in N$, $L_G(A)$ is defined as the value of A in this solution. Let $L(G) = L_G(S)$.

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 $\{a^n b^n c^n \mid n \geqslant 0\}$.*

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
S & \to & A\&C & S & = & A \cap C & L_G(S) & = & \{a^n b^n c^n \mid n \geqslant 0\} \\
A & \to & aA \mid D & A & = & aA \cup D & L_G(A) & = & \{a^i b^j c^k \mid j = k\} \\
D & \to & bDc \mid \varepsilon & D & = & bDc \cup \varepsilon & L_G(D) & = & \{b^m c^m \mid m \geqslant 0\} \\
C & \to & aCc \mid B & C & = & aCc \cup B & L_G(C) & = & \{a^i b^j c^k \mid i = k\} \\
B & \to & bB \mid \varepsilon & B & = & bB \cup \varepsilon & L_G(B) & = & b^*
\end{array}$$

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 $\{a^n b^n c^n \mid n \geqslant 0\}$ as an intersection of two context-free languages:

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

Example 2. *[10] The following strongly non-left-recursive Boolean grammar generates a P-complete language:*

$$\begin{array}{lcl} S & \to & E\&\neg AbS\&\neg CS \\ A & \to & aA \mid \varepsilon \\ C & \to & aCAb \mid b \\ E & \to & aE \mid bE \mid \varepsilon \end{array}$$

The grammar specifies a variant of the circuit value problem, in which every gate x_i computes the function $x_i = \neg x_{i-1} \wedge \neg x_j$, for some $j = j(i) < i$. This function is well visible in the rule for S. Note that the entire family generated by Boolean grammars is contained in $\mathsf{DTIME}(n^3) \subset \mathsf{P}$ [6], and hence this language is among the hardest of its kind.

Example 3. [4] *The following strongly non-left-recursive conjunctive grammar generates the language* $\{wcw \mid w \in \{a,b\}^*\}$*:*

$$\begin{array}{lcl} S & \to & C\&D \\ C & \to & XCX \mid c \\ X & \to & a \mid b \\ D & \to & aA\&aD \mid bB\&bD \mid cE \\ A & \to & XAX \mid cEa \\ B & \to & XBX \mid cEb \\ E & \to & aE \mid bE \mid \varepsilon \end{array}$$

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

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

3 Boolean recursive descent parser

A recursive descent parser for a Boolean grammar uses a parsing table similar to the well-known context-free LL table. Let $k \geqslant 1$. For a string w, define

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

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

Definition 4. [7] *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 $\theta, \eta \in (\Sigma \cup N)^*$, such that $v \in L_G(\eta)$.*

Definition 5. [7] *Let $G = (\Sigma, N, P, S)$ be a strongly non-left-recursive Boolean grammar, let $k > 0$. An LL(k) table for G is a function $T_k : N \times \Sigma^{\leqslant k} \to P \cup \{-\}$, such that for every rule $A \to \varphi$ and $u, v \in \Sigma^*$, for which $u \in L_G(\varphi)$ and v follows A, it holds that $T_k(A, First_k(uv)) = A \to \varphi$.*

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

Both grammars in Examples 1–2 are LL(1). For the grammar in Example 1, the smallest LL(1) table satisfying the above definition is the following one:

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

For instance, $T_1(B, b) = B \to bB$ and $T_1(B, c) = B \to \varepsilon$ because $\varepsilon\langle S\rangle\varepsilon \rightsquigarrow^+ a\langle B\rangle c$ and $b, c \in First_k(L_G(Bc))$. On the other hand, since no string starting with a follows B or is generated by B, Definition 5 imposes no requirements on $T_1(B, a)$, so it can be anything in $\{-, B \to \varepsilon, B \to bB\}$. Note that the known algorithm for constructing LL(k) tables for Boolean grammars [7] will set $T_1(S, b) = T_1(S, c) = S \to A\&C$, because both $L_G(A)$ and $L_G(C)$ contain strings starting 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 $E \to \varepsilon$ and $E \to a$ (or $E \to b$). Suppose this grammar has an LL(k) table T_k for some k and consider $\varepsilon\langle S\rangle\varepsilon \rightsquigarrow^* aX^{k-1}c\langle E\rangle aX^{k-1}$. On one hand, for the rule $E \to \varepsilon$ and the strings $u = \varepsilon \in L_G(\varepsilon)$ and $v = a^k \in L_G(aX^{k-1})$, it follows that $E \to \varepsilon \in T_k(A, a^k)$. On the other hand, taking $E \to aE$, $u = a \in L_G(aE)$ and $v = a^k \in L_G(aX^{k-1})$, one obtains $E \to aE \in T_k(A, a^k)$. Since $E \to \varepsilon \neq E \to aE$, this yields a contraction. It remains unknown whether the language generated by this grammar, $\{wcw \,|\, w \in \{a, b\}^*\}$, is generated by any Boolean LL(k) grammar.

Let us now define the recursive descent parser for a given Boolean LL(k) grammar. 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 $w = w_1 w_2 \ldots w_{|w|}$ and a positive integer p pointing at a position in this string. Each procedure $s()$, where $s \in \Sigma \cup N$, starts with some initial value of this pointer, $p = i$, and eventually either returns, setting the pointer to $p = j$ (where $i \leqslant j \leqslant |w|$), or *raises an exception*, in the sense of an exception handling model of, e.g., C++.

The procedure corresponding to every terminal $a \in \Sigma$ [7] is defined, as in the context-free case, as

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

For every nonterminal $A \in \Sigma$, the corresponding procedure $A()$ [7] 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, First_k(w_p w_{p+1} ...)))
    {
    case A → α_1 & ... & α_m & ¬β_1 & ... & ¬β_n:
        (code for the conjunct A → α_1)
        ⋮
        (code for the conjunct A → α_m)
        (code for the conjunct A → ¬β_1)
        ⋮
        (code for the conjunct A → ¬β_n)
        return;
    case A → ...
      ⋮
    default:
        raise exception;
    }
}
```

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

```
let start = p;
s_1();
⋮
s_ℓ();
let end = p;

(for the first positive conjunct)
```

```
p = start;
s_1();
⋮
s_ℓ();
if i ≠ end, then raise exception;

(for every subsequent positive conjunct)
```

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

```
boolean failed = false;
try
{
      p = start;
      s_1();
      ⋮
      s_ℓ();
      if p ≠ end, then raise exception;
}
exception handler:
      failed = true;
if ¬failed, then raise exception;
p = end;              /* if this is the last conjunct in the rule */
```

The code for the first conjunct $A \to \alpha_1$ stores the initial value of the pointer in the variable *start*, and remembers the end of the substring recognized by α_1 in the variable *end*. Every subsequent positive conjunct $A \to \alpha_i$ is tried starting from the same position *start*, and the variable *end* 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.

The main procedure [7] 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. [7] *Let $k \geqslant 1$. Let $G = (\Sigma, N, P, S)$ be an LL(k) Boolean grammar. Let $T : N \times \Sigma^{\leqslant k} \to P \cup \{-\}$ be an LL(k) table for G, let the set of procedures be constructed with respect to G and T. Then, for every $y, z, \widetilde{z} \in \Sigma^*$ and $s_1, \ldots, s_\ell \in \Sigma \cup N$ ($\ell \geqslant 0$), such that z follows $s_1 \ldots s_\ell$ and $First_k(z) = First_k(\widetilde{z})$, the code $s_1(); \ldots; s_\ell()$, executed on the input $y\widetilde{z}$,*

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

4 Simple formal properties

A few technical results need to be established for use in the subsequent arguments.

Definition 6. *An LL(k) Boolean grammar $G = (\Sigma, N, P, S)$ is said to be well-behaved, if, for every $A \in N$, (i) $L(A) \neq \varnothing$ 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 $G = (\Sigma, \{S\}, \{S \to aS\}, S)$ generating $\varnothing$ is also considered well-behaved.*

Lemma 2. *For every LL(k) Boolean grammar $G = (\Sigma, N, P, S)$ there exists an equivalent well-behaved LL(k) Boolean grammar.*

The transformation is not effective, because it requires testing the emptiness of a language, which is undecidable already for linear conjunctive LL grammars.

Sketch of a proof. If $L(G) = \varnothing$, such a grammar exists; assume $L(G) \neq \varnothing$. First, construct a grammar $G' = (\Sigma, N', P', S)$, where $N' = \{A | L_G(A) \neq \varnothing\}$, while P' contains all rules

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_\ell,$$

such that $\alpha_i, \beta_j \in (\Sigma \cup N')^*$ and there exist $\beta_{\ell+1}, \ldots, \beta_n \in (\Sigma \cup N)^* \cdot (N \setminus N') \cdot (\Sigma \cup N)^*$, with $n \geqslant \ell$, where

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_\ell \& \neg\beta_{\ell+1} \& \ldots \& \neg\beta_n \in P$$

In other words, for every rule in P, such that its positive conjuncts contain no references to symbols in $N \setminus N'$, all negative conjuncts with such references are removed, while those rules in P that contain positive conjuncts referring to symbols in $N \setminus N'$ are not used in the construction of P'. 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 for every $A \in N'$, $L_{G''}(A) = L_{G'}(A)$. Hence, $L(G'') = L(G)$, and since the grammar satisfies Definition 6 by construction, the lemma is proved. □

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

Let us prove that G' is LL(k). Suppose it is not, that is, there exists a pair of distinct rules $A \to \varphi, A \to \varphi' \in P$ and strings $u, v, u', v' \in \Sigma^*$ and $\theta, \eta, \theta', \eta' \in (\Sigma \cup N)^*$, such that $\varepsilon\langle T\rangle\varepsilon \rightsquigarrow^* \theta\langle A\rangle\eta$, $\varepsilon\langle T\rangle\varepsilon \rightsquigarrow^* \theta'\langle A\rangle\eta'$ $u \in First_k(\varphi)$, $u' \in First_k(\varphi')$, $v \in L_G(\eta)$ $v' \in L_G(\eta')$ 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, we obtain $\varepsilon\langle S\rangle\varepsilon \rightsquigarrow^* \mu\theta\langle A\rangle\eta\nu$ 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 since 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 $A \to \varphi$ and $A \to \varphi'$, The contradiction obtained proves that G' is LL(k).

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

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 generating $L(G) \cap a^*$.*

Sketch of a proof. Let us remove all conjuncts referring to symbols in $\Sigma \setminus \{a\}$. Construct a grammar $G' = (\Sigma, N, P', S)$, where P' contains all rules

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_\ell,$$

where $\alpha_i, \beta_j \in (\{a\} \cup N)^*$ and there exist $\beta_{\ell+1}, \ldots, \beta_n \in (\Sigma \cup N)^* \cdot (\Sigma \setminus \{a\}) \cdot (\Sigma \cup N)^*$, with $n \geqslant \ell$, such that

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_\ell \& \neg\beta_{\ell+1} \& \ldots \& \neg\beta_n \in P'$$

The grammar G' is also LL(k), and $L_{G'}(A) = L_G(A)$ for every $A \in N'$. Using Lemma 2, an equivalent well-behaved grammar is obtained. □

Lemma 5. *For every well-behaved LL(k) Boolean grammar G there exists a well-behaved LL(k) Boolean grammar G', such that $L(G') = L(G)$ and for each nonterminal A in G' there is either one or more rules of the form $A \to B_1 \& \ldots \& B_m \& \neg C_1 \& \ldots \& \neg C_n$, or a single rule of the form $A \to BC$, $A \to a$ or $A \to \varepsilon$.*

The proof of Lemma 5 is by straightforward decomposition of complex rules by inserting references to auxiliary nonterminals with simpler rules. 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, Boolean grammars can generate the following nonregular unary language [6]:

Example 4. *The following Boolean grammar generates the language* $\{a^{2^n} \mid n \geqslant 0\}$*:*

$$\begin{array}{lcl} S & \to & A\&\neg aA \mid aB\&\neg B \mid aC\&\neg C \\ A & \to & aBB \\ B & \to & \neg CC \\ C & \to & \neg DD \\ D & \to & \neg A \end{array}$$

By Theorem 1, there is no equivalent LL(k) Boolean grammar.

We shall now demonstrate 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.

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

The following lemma is an essential component of the proof:

Lemma 6. *Let* $G = (\Sigma, N, P, S)$ *be a well-behaved LL(k) Boolean grammar, let* $B \in N$*,* $a \in \Sigma$ *and let some string in* $a^k\Sigma^*$ *follow* B*. Then there exists at most one number* $i \geqslant 0$*, such that* $a^i \in L_G(B)$*.*

Proof. Supposing the contrary, let $a^{i_1}, a^{i_2} \in L(B)$, where $0 \leqslant i_1 \leqslant i_2$, and let $a^k x$, where $x \in \Sigma^*$, be a string that follows B. Consider the string $a^{k+(j_2-j_1)}x$, for which we know that $k + (j_2 - j_1) \geqslant k$, and hence $First_k(a^{k+(j_2-j_1)}x) = First_k(a^k x) = a^k$. By Lemma 1, $a^{i_1} \in L(B)$ implies that $B()$ returns on the input $w_1 = a^{i_1}a^{k+(j_2-j_1)}x$, 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, proving the claim. □

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 $G = (\{a\}, N, P, S)$, such that for every $A \in N$ the set P contains either one or more rules of the form $A \to D_1\& \ldots \&D_q\&\neg E_1\& \ldots \&\neg E_r$, or a single rule of the form $A \to BC$, $A \to a$ or $A \to \varepsilon$.

Let us prove that in every rule $A \to BC$, if $L(C) \not\subseteq a^{\leqslant k-1}$, then $L(B)$ is a singleton. By assumption, there exists $a^j \in L(C)$, where $j \geqslant k$. Let a^ℓ

be a string that follows A, then $a^{j+\ell} \in a^k a^*$ follows B. Since we know that $L(B) \subseteq a^*$ and $L(B) \neq \varnothing$, Lemma 6 states that $|L(B)| = 1$.

Now let us reconstruct the grammar to show that $L(G)$ is regular. For every rule $A \to BC$, such that $L(B)$ is a singleton, replace this rule with the rule $A \to a^i C$, where $L(B) = \{a^i\}$. If $L(B)$ is not a singleton, then $L(C) \subseteq a^{\leqslant k}$ by the claim above, and the rule can be equivalently replaced with $\{A \to a^i B \mid a^i \in L(C)\}$.

The system of language equations corresponding to the transformed grammar has the same set of solutions as the original system, that is, it has a unique solution $(\ldots, L_G(A), \ldots)$. Since the system uses one-sided concatenation, all components of this solution are regular. □

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. □

In particular, the aforementioned language $\{a^{2^n} \mid w \in \{a, b\}^*\}$ is not Boolean LL(k).

6 Nonrepresentability results for subsets of $\Sigma^* a^*$

Let us establish a method of proving nonrepresentability of some languages over non-unary alphabets. This method exploits long blocks of identical letters in a certain similarity to Yu's [12] nonrepresentability argument for trellis automata,

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

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

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 $A \to D_1 \& \ldots \& D_q \& \neg E_1 \& \ldots \& \neg E_r$, or a unique rule of the form $A \to BC$, $A \to a$ or $A \to \varepsilon$.

According to Corollary 1, for every $A \in N$ and $a \in \Sigma$, the set $L_G(A) \cap a^*$ is regular. Let $d(A, a) \geqslant 0$ and $p(A, a) \geqslant 1$ be numbers, such that $L(A) \cap a^*$

is ultimately periodic starting from $d(A,a)$ and with the least period $p(A,a)$. Define $p = \operatorname{lcm}_{A,a} p(A,a)$, $d_0 = \max_{A,a} d(A,a)$ and $d = d_0 \cdot |N|$.

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

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

$$n < d \cdot |w| + d_0 \cdot |\mathcal{X}| + 1 \tag{3}$$

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

The essence of this claim is that if there are too many as, then the parser cannot keep count. The below argument reveals that the parser must be matching the symbols of w to the first symbols of a^n.

The proof of the claim is 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 $A \to \varepsilon$ or $A \to b$ with $b \in \Sigma$. In either case $n \leqslant 1$ and the inequality (3) holds.

Now suppose $A()$ 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 $A \to BC$, then the execution of $A()$ starts 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. The first of them is trivial:

Case I: $B()$ consumes nothing. If $B()$ consumes ε, then $C()$ is 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 any string that follows A therefore follows C. Then, by the induction hypothesis applied to this computation of $C()$, $n < d \cdot |w| + d_0 \cdot |\mathcal{Y}| + 1$, 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 course of this computation. Since, obviously, $\mathcal{Y} \subseteq \mathcal{X}$, (3) follows.

Consider the other two cases, which are illustrated in Figure 1:

Case II: $B()$ consumes a nonempty proper prefix of w. Suppose $w = uv$, 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.

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 course of this computation. By the induction hypothesis applied to this computation,

$$n < d \cdot |v| + d_0 \cdot |\mathcal{Y}| + 1 \leqslant d \cdot |v| + d + 1 \leqslant d \cdot |w| + 1,$$

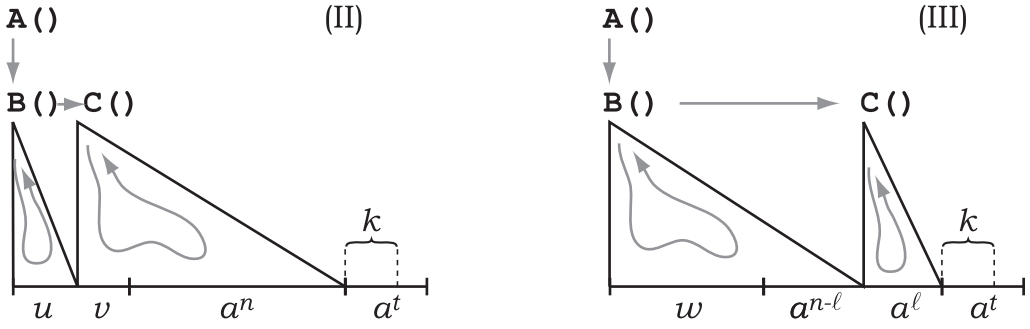

Figure 1: Proof of Theorem 2, Claim 2.1, cases II and III.

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

$$n < d \cdot |w| + d_0 \cdot |\mathcal{X}| + 1$$

Case III: $B()$ consumes the entire w and possibly some a's. Let $B()$ consume $wa^{n-\ell}$, for some $0 \leqslant \ell \leqslant n$.

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,

$$n - \ell < d \cdot |w| + d_0 \cdot |\mathcal{Y}| + 1 \tag{4}$$

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 $B \notin \mathcal{Y}$, since the computation would enter an infinite recursion otherwise. Therefore, $\mathcal{Y} \subseteq \mathcal{X} \setminus \{B\}$ and $|\mathcal{Y}| \leqslant |\mathcal{X}| - 1$, hence (4) can be transformed as follows:

$$n < d \cdot |w| + d_0 \cdot |\mathcal{Y}| + \ell + 1 \leqslant d \cdot |w| + d_0 \cdot (|\mathcal{X}| - 1) + \ell + 1 \tag{5}$$

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

$$n < d \cdot |w| + d_0 \cdot |\mathcal{X}| - (d_0 - \ell) + 1 \leqslant d \cdot |w| + d_0 \cdot |\mathcal{X}| + 1, \tag{6}$$

which completes the proof of this case.

It remains to consider the case of a rule $A \to D_1 \& \ldots \& D_q \& \neg E_1 \& \ldots \& \neg E_r$ ($q \geqslant 1$, $r \geqslant 0$) 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.

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

The proof of this claim analyzes the generation of wa^n, using Claim 2.1 to single out a nonterminal C generating a sufficiently long sequence of as. Then the periodicity of $L(C) \cap a^*$ is used to pump this sequence. The proof is done by an induction on the lexicographically ordered pairs $(|w|, h(A))$.

Let $|w| > 0$ and $wa^n \in L(A)$. Then there exists a rule in P, which generates wa^n. This cannot be a rule of the form $A \to \varepsilon$ or $A \to b$, since $|wa^n| \geqslant 2$.

Suppose there is a rule $A \to BC$, such that there exists a partition of wa^n into two parts, the first being from $L(B)$ and the second being from $L(C)$. Depending on the partition, there are three cases to consider:

Case I: $\varepsilon \in L(B)$, $wa^n \in L(C)$. Since $h(C) \leqslant h(A) - 1$, 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 $n \geqslant n_0(w,t) + p$, then similarly $wa^{n-p} \in L(A)$.

Case II: B generates a proper prefix of w. Let $w = uv$ and assume $u \in L(B)$ and $va^n \in L(C)$. Since a^t follows C (because a^t follows A) and $|v| < |w|$, we have $n \geqslant n_0(w,t) \geqslant n_0(v,t)$ (or, in the second case, $n \geqslant n_0(w,t)+p \geqslant n_0(v,t)+p$), and hence the induction hypothesis is applicable to va^n and C. In the first case we obtain $va^{n+p} \in L(C)$, 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 w and 0 or more a's, with at least k a's left. Suppose $wa^{n-\ell} \in L(B)$ and $a^\ell \in L(C)$, where $\ell + t \geqslant k$. Since $a^\ell a^t$ follows B, 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 $B()$, $n - \ell \leqslant d \cdot |w| + d_0 \cdot |N| + 1 \leqslant d\cdot(|w|+1)+1$, and therefore $n \leqslant d\cdot(|w|+1)+1+\ell$. On the other hand, $n \geqslant n_0(w,t) = d\cdot(|w|+1)+d_0+k-t+1$ by our assumption. Combining these inequalities, we obtain $d\cdot(|w|+1)+d_0+k-t+1 \leqslant d\cdot(|w|+1)+1+\ell$, that is, $\ell \geqslant d_0+k-t \geqslant d_0$.

Since $L_G(C) \cap a^*$ is ultimately periodic starting from d_0 with period p, $a^\ell \in L_G(C)$ implies $a^{\ell+p} \in L_G(C)$. 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 we have $n \geqslant n_0(w,t)+p = d\cdot(|w|+1)+d_0+k-t+1+p$, and therefore $d\cdot(|w|+1)+d_0+k-t+1+p \leqslant d\cdot(|w|+1)+1+\ell$, that is, $\ell \geqslant d_0+k-t+p \geqslant d_0+p$. Then, by the periodicity of $L_G(C) \cap a^*$ starting from d_0 with period p, $a^\ell \in L_G(C)$ implies $a^{\ell-p} \in L_G(C)$. Finally, by $wa^{n-\ell} \in L(B)$, $wa^{n-\ell}a^{\ell-p} = wa^{n-p} \in L(A)$ is obtained.

Case IV: B **generates** $wa^{n-(k-t-1)}$ **or more.** Let $wa^{n-t'} \in L(B)$ and $a^{t'} \in L(C)$, where $t' \leqslant k-t-1$. Since $n \geqslant n_0(w,t) = d \cdot (|w|+1)+d_0+k-t+1$ by assumption, $n-t' \geqslant d \cdot (|w|+1)+d_0+1+k-(t'+t)+1 = n_0(w,t'+t)$. Since, in addition, $h(B) \leqslant h(A)-1$ and $a^{t'+t}$ follows B, the induction hypothesis is applicable to $wa^{n-t'} \in L(B)$. We obtain $wa^{n-t'+p} \in L(B)$, and therefore $wa^{n+p} \in L(A)$.

In the second case, $n \geqslant n_0(w,t)+p$ by assumption, hence $n-t' \geqslant n_0(w,t'+t)+p$.

Consider the case of wa^n generated using a rule

$$A \to D_1 \& \ldots \& D_q \& \neg E_1 \& \ldots \& \neg E_r \tag{7}$$

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

Let us first show that $wa^{n+p} \in L(A)$. For every positive conjunct $A \to D_i$, 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 $n+p \geqslant n_0(w,t)+p$, 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 $n \geqslant n_0(w,t)+p$, let us 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 $n-p \geqslant n_0(w,t)$ 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.

Finally, define $d' = d+d_0+p+k-t+1$, and the statement of the theorem follows from Claim 2.2. □

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

Proof. By Theorem 2, there exist constants $d, d', p \geqslant 0$, such that for every string $a^n b^\ell \in L$ with $\ell \geqslant dn + d'$, it holds that $a^n b^{\ell+p} \in L$. 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 $a^n b^\ell \in L$ we must have $\ell < dn + d'$. In other words, $f(n) < dn + d' \leqslant (d+d')n$, and setting $C = d + d'$ proves the theorem. □

Example 5. *The linear conjunctive language* $\{a^n b^{2^n} \mid n \geqslant 0\}$ *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 $\{a^n b^n c^n \,|\, n \geqslant 0\}$ was given in Example 1 on page 3. 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 [9], and hence their expressive power is far from being trivial. However, it turns out that some very simple languages are beyond their scope:

Theorem 3. *Let Σ be an alphabet, let $a, b \in \Sigma$ ($a \neq b$). Then, for every language $L \subseteq \Sigma^*$, $L \cdot \{a, b\}$ 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.

Let us show 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 for L, let $T : N \times \Sigma^{\leqslant k} \to P$ 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 $|yz| > k$, which returns, consuming y, then $z = \varepsilon$ and ε follows B.*

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 $f : \Sigma^*\{a, b\} \to \Sigma^*\{a, b\}$ 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 $f(ws)$ starts 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. Note that the path to A in the tree of recursive calls is shorter than the path to B, and hence, by the induction hypothesis, ε follows A and

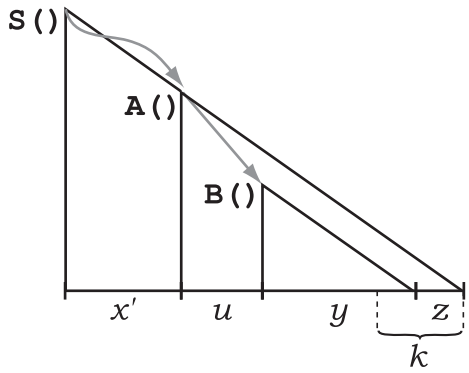

Figure 2: Proof of Theorem 3, Claim 3.1, induction step.

$z' = \varepsilon$, hence $v = z$. We thus obtain that $A()$ returns on uyz, consuming uyz. The form of the computation is illustrated in Figure 2.

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

$$T_k(A, First_k(uyz)) = A \to \ldots \& uBz \& \ldots, \tag{8}$$

where the conjunct $A \to uBz$ 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 $|uyz| \geqslant |yz| > k$, it is known that $First_k(f(uyz)) = First_k(uyz)$. Then this computation starts with choosing the same rule (8). By assumption, this computation is accepting as well, and this proves that it eventually reaches the conjunct $A \to uBz$ 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 $f(uyz)$.

We have thus 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 (8), ε follows B. This completes the proof of Claim 3.1.

Let us now define a new grammar $G' = (\Sigma, N, P', S)$ as follows. Let m be the greatest length of the right-hand side of a rule in P. Every rule of the form $A \to u$ ($u \in \Sigma^*$) or $A \to u_1 B_1 \& \ldots \& u_n B_n$ in P is in P' as well. In addition, for every $A \in N$ and for every $w \in L_G(A)$, such that $|w| \leqslant k + m$, the set P' contains a rule $A \to w$. Let us 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 $A \to y \in P'$ defined above, and $y \in L_G(A)$ for every such rule.

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 $A \to y$, 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 $A \to u_1 B_1 \& \ldots \& u_n B_n$.

Let us proceed with a 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: $|y| \leqslant k + m$. Then $A \to y \in P'$, which proves the claim.

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

$$T_k(A, First_k(y)) = A \to u_1 B_1 v_1 \& \ldots \& u_n B_n v_n, \tag{9}$$

such that $y \in L_G(u_i B_i v_i)$ for every i. Let $y = u_i x_i v_i$, where $x_i \in L_G(B_i)$. Then $A()$ eventually calls $B()$ on $x_i v_i z$, which returns, consuming x_i.

By the definition of m, $|u_i B_i v_i| \leqslant m$, and therefore $|x_i| > k$. Applying Claim 3.1 to the computation of $B_i()$, we obtain that $v_i z = \varepsilon$. Then the rule (9) is in fact of the form

$$A \to u_1 B_1 \& \ldots \& u_n B_n, \tag{9'}$$

and is therefore in P'.

On the other hand, applying the induction hypothesis to the computation of $B_i()$, we obtain $x_i \in L_{G'}(B_i)$, and hence $y \in L_{G'}(u_i B_i v_i)$. From this, using the rule (9′), we obtain $y \in L_{G'}(A)$, which completes the proof of Claim 3.3.

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

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 $ws \in L_{G'}(S) = L(G')$. Claim 3.4 is proved.

We have thus 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, which completes the proof of the theorem. □

Example 6. *The context-free LL language* $\{a^n b^n cs \,|\, n \geqslant 0, s \in \{a, b\}\}$*, which is at the same time linear context-free, is not linear conjunctive LL.*

Theorem 4. *The family of LL linear conjunctive languages is closed under intersection. It is not closed under union, concatenation and reversal.*

Proof. The closure under intersection is given by a direct application of conjunction.

Each of the three nonclosure results is proved by representing the language $L = \{a^n b^n cs \mid n \geqslant 0, s \in \{a, b\}\}$ from Example 6. In the case of

union, consider the languages $\{a^n b^n ca \mid n \geqslant 0\}$ and $\{a^n b^n cb \mid n \geqslant 0\}$: each of them is obviously LL(1) linear context-free, and their union equals L. For concatenation, it is sufficient to represent L as $\{a^n b^n c \mid n \geqslant 0\} \cdot \{a, b\}$, where each of these languages is LL(1) linear context-free. For the nonclosure under reversal, consider the language $\{scb^n a^n \mid n \geqslant 0, s \in \{a, b\}\}$, which is in $\mathit{LinCFLL} \subset \mathit{LinConjLL}$, while its reversal equals L. Hence, supposing the closure of LL linear conjunctive languages under any of these three operations, one obtains that $\{a^n b^n cs \mid n \geqslant 0, s \in \{a, b\}\}$ is in $\mathit{LinConjLL}$, thus violating Example 6 and yielding a contradiction. □

8 Linear Boolean LL grammars

Linear Boolean grammars are known to have the same expressive power as linear conjunctive grammars [5, 6]. In contrast, their LL subsets differ in power, as the language from Example 6, which is not representable by any LL(k) linear conjunctive grammar, has a simple LL(1) linear Boolean grammar, which relies on de Morgan's laws to specify a union of languages via conjunction and negation:

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

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

Note that there is no equivalent LL(k) linear conjunctive grammar, see Example 6.

This separate class of languages thus deserves a separate study. The following nonrepresentability result, proved by a counting argument, is useful in assessing their expressive power.

Lemma 7. *The language* $\{a^n b^n c^\ell \mid n, \ell \geqslant 0\}$ *is not LL linear Boolean.*

Proof. Suppose there exists a well-behaved LL(k) linear Boolean grammar $G = (\Sigma, N, P, S)$ for this language. Assume, without loss of generality, that every conjunct in this grammar is of the form $A \to \pm B$, $A \to \pm sB$, $A \to \pm Ct$, $A \to s$ or $A \to \varepsilon$, where $s, t \in \Sigma$ and $B, C \in N$.

First infer the following property from Theorem 2:

Claim 7.1. *There exist numbers* $d, p \geqslant 0$*, such that for every nonterminal* $B \in N$ *and for all* $n \geqslant 1$*,* $\ell \geqslant dn$ *and* $i \geqslant 0$*,*

$$b^n c^{\ell+ip} \in L(B) \quad \text{if and only if} \quad b^n c^\ell \in L(B).$$

It will now be proved that for fixed numbers $m, \ell \geqslant 0$ and for a nonterminal A, the membership of strings of the form $a^m b^n c^\ell$ (with various $n \geqslant 0$) in $L(A)$ depends upon the membership of strings $b^n c^\ell$ in the languages $L(B)$, for different $B \in N$, and the dependence function is unique for all values of n.

Claim 7.2. *For every $A \in N$, for every $m \geqslant 0$, and for every $\ell \geqslant m \cdot |N| + h(A)$, there exists a Boolean function $f_{A,m,\ell}(\ldots, x_{B,i}, \ldots)$, where $B \in N$ and $0 \leqslant i \leqslant \ell$, such that for every $n \leqslant m$, $a^m b^n c^\ell \in L(A)$ if and only if $f_{A,m,\ell}(\ldots, \sigma_{D,i}, \ldots)$, where $\sigma_{D,i} = 1$ if and only if $b^n c^i \in L(D)$.*

The proof is an induction on the lexicographically ordered pairs $(m, \ell, h(A))$.

Basis $m = 0$**.** Then $f_{A,m,\ell} = x_{A,\ell}$.

Induction step. Let $m \geqslant 1$ and let $\ell \geqslant m \cdot |N| + h(A)$. In order to determine the condition of the membership of $a^m b^n c^\ell$ in $L(A)$, consider all conjuncts of all rules in A.

A conjunct of the form $A \to w$ with $|w| \leqslant 1$ cannot generate this string.

Consider a conjunct of the form $A \to \pm B$. Then $h(B) < h(A)$, and the induction hypothesis is applicable to B, which gives a function $f_{B,m,\ell}$, such that, for all $n \geqslant 0$, $a^m b^n c^\ell \in L(B)$ if and only if $f_{B,m,\ell}(\ldots, \sigma_{D,i}, \ldots) = 1$.

For a conjunct of the form $A \to \pm aB$, $a^m b^n c^\ell \in L(aB)$ if and only if $a^{m-1} b^n c^\ell \in L(B)$. Since $\ell \geqslant m \cdot |N| + h(A)$, we have $\ell \geqslant m \cdot |N| = (m-1) \cdot |N| + |N| \geqslant (m-1) \cdot |N| + h(B)$, which makes the induction hypothesis applicable to B. Then there exists a function $f_{B,m-1,\ell}$, such that, for all $n \geqslant 0$, $a^{m-1} b^n c^\ell \in L(B)$ if and only if $f_{B,m-1,\ell}(\ldots, x_{D,i}, \ldots) = 1$.

For a conjunct of the form $A \to \pm Cc$, $a^m b^n c^\ell \in L(Cc)$ if and only if $a^m b^n c^{\ell-1} \in L(C)$. Since $h(B) < h(A)$, the inequality $\ell \geqslant m \cdot |N| + h(A)$ implies $\ell - 1 \geqslant m \cdot |N| + h(A) - 1 \geqslant m \cdot |N| + h(B)$. Then, by the induction hypothesis, there exists a function $f_{C,m,\ell-1}$, such that, for all $n \geqslant 0$, $a^m b^n c^{\ell-1} \in L(C)$ if and only if $f_{C,m,\ell-1}(\ldots, x_{D,i}, \ldots) = 1$.

Since the rules for A constitute a Boolean formula over their conjuncts, the function $f_{A,m,\ell}$ can be defined as a combination of the above functions implementing individual conjuncts.

Let us now improve the statement of Claim 7.2, so that every function $f_A(\ldots, x_{D,\ell}, \ldots)$ depends upon a bounded number of Boolean variables, which does not increase with m and ℓ.

Claim 7.3. *For every $A \in N$, for every $m \geqslant 0$, and for every $\ell \geqslant m \cdot |N| + h(A)$, there exists a Boolean function $f_{A,m,\ell}(\ldots, x_{B,i}, \ldots)$, where $B \in N$ and $0 \leqslant i < d + p$, such that for every $n \leqslant m$, $a^m b^n c^\ell \in L(A)$ if and only if $f_{A,m,\ell}(\ldots, \sigma_{D,i}, \ldots)$, where $\sigma_{D,i} = 1$ if and only if $b^n c^i \in L(D)$.*

By Claim 7.2, there exists a function $f_{A,m,\ell}(\ldots, x_{D,\ell}, \ldots)$, such that $a^m b^n c^\ell \in L_G(A)$ if and only if $f_{A,m}(\ldots, \sigma_{D,i}, \ldots)$ for all $n \geqslant 0$, with $\sigma_{D,i} = 1$ if and only if $b^n c^i \in L_G(D)$. For every number $i \geqslant d$, define a number $[i]$, such that $d \leqslant [i] < d + p$ and $[i] - i$ divides by p. According to Claim 7.1, $b^n c^i \in L_G(D)$ if and only if $b^n c^{[i]} \in L_G(D)$. This allows us to define $\widehat{f}_{A,m,\ell}(\ldots, x_{\sigma_{D,i}}, \ldots) = f_{A,m,\ell}(\ldots, x_{\sigma_{D,[i]}}, \ldots)$. This function satisfies Claim 7.3.

Now define $m_0 = 2^{2^{|N| \cdot (d+p)}}$ and $\ell_0 = |N| \cdot m_0 + \max_{A \in N} h(A)$. For every $m \in \{0, \ldots, m_0\}$, denote $\widehat{f}_m = f_{S,m,\ell_0}$. Each function $\widehat{f}_m$ depends on $|N| \cdot (d+p)$ variables. There exist $2^{2^{|N| \cdot (d+p)}} = m_0$ distinct Boolean functions over this number of variables, and hence our set of $m_0 + 1$ such functions must contain a pair of duplicates:

Claim 7.4. *There exist two numbers m and $\widetilde{m}$, with $0 \leqslant m < \widetilde{m} \leqslant m_0$, such that $\widehat{f}_m = \widehat{f}_{\widetilde{m}}$.*

Consider strings of the form $a^* b^m c^{\ell_0}$. For all $i \in \{\ell_0 - (d+p-1), \ldots, \ell_0\}$ and $D \in N$, define $\sigma_{D,i} = 1$ if $b^m c^i \in L_G(D)$ and $\sigma_{D,i} = 0$ otherwise. Take a true statement $a^m b^m c^{\ell_0} \in L(G)$. By Claim 7.3, $a^m b^m c^{\ell_0} \in L(G)$ if and only if $\widehat{f}_m(\ldots, \sigma_{D,i}, \ldots) = 1$. The latter, according to Claim 7.4, is equivalent to $\widehat{f}_{\widetilde{m}}(\ldots, \sigma_{D,i}, \ldots) = 1$, which, by Claim 7.3 again, holds if and only if $a^{\widetilde{m}} b^m c^{\ell_0} \in L(G)$, which is not true. A contradiction of the form *"true if and only if false"* has thus been obtained, which proves the lemma. □

Theorem 5. *The family of LL linear Boolean languages is closed under all Boolean operations. It is not closed under concatenation and reversal.*

Proof. Intersection and complementation can be specified directly, the closure under union follows by de Morgan's laws.

Suppose *LinBoolLL* is closed under concatenation. Then the language $\{a^n b^n \mid n \geqslant 0\} \cdot c^*$ is in *LinBoolLL*, which contradicts Lemma 7.

Similarly, if *LinBoolLL* were closed under reversal, then $(\{c^\ell b^n a^n \mid n, \ell \geqslant 0\})^R$, would be in *LinBoolLL*, again contradicting Lemma 7. □

9 Hierarchy

The results of this paper lead to a detailed comparison of different subfamilies of LL(k) Boolean grammars with each other and with different subfamilies of Boolean grammars. The following theorem summarizes these results.

Theorem 6.

I. *LinCFLL* $\subset$ *LinConjLL*, *with* $\{a^n b^n c^n \mid n \geqslant 0\} \in$ *LinConjLL* $\setminus$ *LinCFLL*.

II. *LinCFLL* $\subset$ *CFLL*, *with* $\{a^n b^n c \mid n \geqslant 0\} \cdot \{a, b\} \in$ *CFLL* $\setminus$ *LinCFLL*.

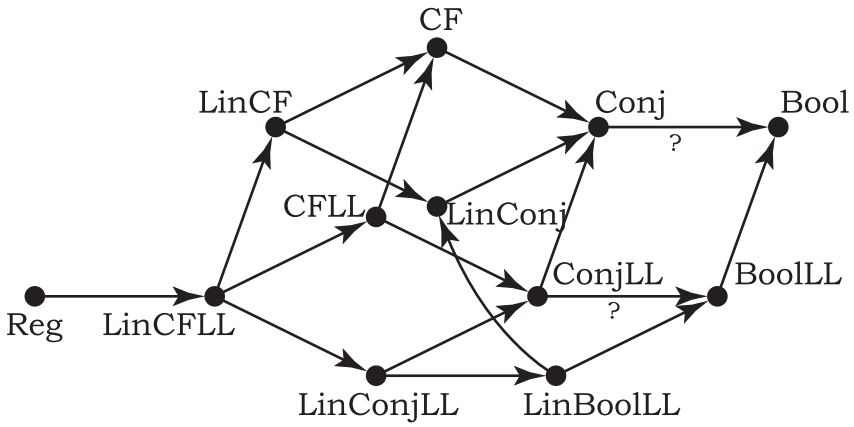

Figure 3: Expressive power of subfamilies of Boolean grammars.

III. $\mathsf{LinConjLL} \subset \mathsf{ConjLL}$, *with* $\{a^n b^n c \mid n \geqslant 0\} \cdot \{a, b\} \in \mathsf{ConjLL} \setminus \mathsf{LinConjLL}$.

IV. $\mathsf{CFLL} \subset \mathsf{ConjLL}$, *with* $\{a^n b^n c^n \mid n \geqslant 0\} \in \mathsf{ConjLL} \setminus \mathsf{CFLL}$.

V. $\mathsf{LinConjLL} \subset \mathsf{LinBoolLL}$, *with* $\{a^n b^n c \mid n \geqslant 0\} \cdot \{a, b\} \in \mathsf{LinBoolLL} \setminus \mathsf{LinConjLL}$.

VI. $\mathsf{LinBoolLL} \subset \mathsf{BoolLL}$, *with* $\{a^n b^n c^\ell \mid n, \ell \geqslant 0\} \in \mathsf{BoolLL} \setminus \mathsf{LinBoolLL}$.

VII. $\mathsf{LinCFLL} \subset \mathsf{LinCF}$, *with* $\{a^n b^n c \mid n \geqslant 0\} \cdot \{a, b\} \in \mathsf{LinCF} \setminus \mathsf{LinCFLL}$.

VIII. $\mathsf{LinBoolLL} \subset \mathsf{LinConj}$, *with* $\{a^n b^n c^\ell \mid n, \ell \geqslant 0\} \in \mathsf{LinConj} \setminus \mathsf{LinConjLL}$.

IX. $\mathsf{CFLL} \subset \mathsf{CF}$, *with* $\{a^n i b^{in} \mid n \geqslant 0, i \in \{1, 2\}\} \in \mathsf{CF} \setminus \mathsf{CFLL}$.

X. $\mathsf{ConjLL} \subset \mathsf{Conj}$, *with* $\{a^n b^{2^n} \mid n \geqslant 1\} \in \mathsf{Conj} \setminus \mathsf{ConjLL}$.

XI. $\mathsf{BoolLL} \subset \mathsf{Bool}$, *with* $\{a^n b^{2^n} \mid n \geqslant 1\} \in \mathsf{Bool} \setminus \mathsf{BoolLL}$.

Proof. All eleven inclusions are immediate, so it remains to argue that in each of these cases the given language separates the respective families.

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

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

$$\begin{array}{rcl} S & \to & AcX \\ A & \to & aAb \mid \varepsilon \\ X & \to & a \mid b \end{array}$$

On the other hand, this language is not in $\mathsf{LinConjLL}$ according to Example 6, and hence not in $\mathsf{LinCFLL}$.

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

IV. The non-context-free language $\{a^n b^n c^n \mid n \geqslant 0\}$ 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 $\{a^n b^n c^n \mid n \geqslant 0\}$ belongs to *LinBoolLL*. However, it is not in *LinConjLL* by Example 6,

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

$$\begin{array}{lcl} S & \to & AC \\ A & \to & aAb \mid \varepsilon \\ C & \to & cC \mid \varepsilon \end{array}$$

On the other hand, as stated in Lemma 7, it is not in *LinBoolLL*.

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

VIII. The language $\{a^n b^n c^\ell \mid n, \ell \geqslant 0\}$ is linear context-free as a concatenation of a linear context-free language $\{a^n b^n \mid n \geqslant 0\}$ and a regular language c^*, and therefore it belongs to the larger family *LinConj*. It is not in *LinBoolLL* by Lemma 7.

IX. The language $\{a^n i b^{in} \mid n \geqslant 0, i \in \{1, 2\}\}$ is in *CF*, since it is generated by the following context-free grammar:

$$\begin{array}{lcl} S & \to & A \mid B \\ A & \to & aAb \mid 1 \\ B & \to & aBbb \mid 2 \end{array}$$

This grammar is not LL(k) for any k, and it is well-known that no LL(k) context-free grammar generates this language [1].

X. The language $\{a^n b^{2^n} \mid n \geqslant 1\}$ is generated by a linear conjunctive grammar [5], 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, $\{a^n b^{2^n} \mid n \geqslant 1\}$ is in *Bool*, but, due to Example 5, is not in *BoolLL*. □

The resulting inclusion diagram is given in Figure 3, in which arrows with a question mark denote inclusions not known to be proper, the rest being proper.

Let us note some open problems. It remains to determine whether the families *ConjLL* and *BoolLL* are different. For some useful abstract languages generated by Boolean grammars, most notably for $\{wcw \mid w \in \{a, b\}^*\}$, it is important to know whether they are Boolean LL(k). Finally, while this paper establishes the first negative results for Boolean LL(k) grammars, it remains to invent a method of proving languages to be nonrepresentable by Boolean grammars of the general form. The lack of such a method is a significant gap in our knowledge on Boolean grammars.

Acknowledgement

Supported by the Academy of Finland under grant 118540.

References

[1] J. Autebert, J. Berstel, L. Boasson, "Context-free languages and push-down automata", in: Rozenberg, Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, Berlin, 1997, 111–174.

[2] T. Jurdzinski, K. Loryś, "Church-Rosser Languages vs. UCFL", *Automata, Languages and Programming* (ICALP 2002, Malaga, Spain, July 8–13, 2002), LNCS 2380, 147–158.

[3] V. Kountouriotis, Ch. Nomikos, P. Rondogiannis, "Well-founded semantics for Boolean grammars", *Developments in Language Theory* (DLT 2006, Santa Barbara, USA, June 26–29, 2006), LNCS 4036, 203–214.

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

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

[6] A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1 (2004), 19–48.

[7] A. Okhotin, "An extension of recursive descent parsing for Boolean grammars", Technical Report 2004–475, School of Computing, Queen's University, Kingston, Ontario, Canada; revised version submitted.

[8] A. Okhotin, "Generalized LR parsing algorithm for Boolean grammars", *International Journal of Foundations of Computer Science*, 17:3 (2006), 629–664.

[9] A. Okhotin, "Language equations with symmetric difference", *Computer Science in Russia* (CSR 2006, St. Petersburg, Russia, June 8–12, 2006), LNCS 3967, 292–303.

[10] A. Okhotin, "A simple P-complete problem and its representations by language equations", *Machines, Computations and Universality* (MCU 2007, Orléans, France, September 10–14, 2007), accepted.

[11] V. Terrier, "On real-time one-way cellular array", *Theoretical Computer Science*, 141 (1995), 331–335.

[12] S. Yu, "A property of real-time trellis automata", *Discrete Applied Mathematics*, 15:1 (1986), 117–119.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978–952–12–1912–2
ISSN 1239-1891