RAIRO-Theor. Inf. Appl. 49 (2015) 153–178

Available online at:
DOI: 10.1051/ita/2015004

www.rairo-ita.org

LINEAR GRAMMARS WITH ONE-SIDED CONTEXTS AND THEIR AUTOMATON REPRESENTATION*

MIKHAIL BARASH¹ AND ALEXANDER OKHOTIN²

Abstract. The paper considers a family of formal grammars that extends linear context-free grammars with an operator for referring to the left context of a substring being defined, as well as with a conjunction operation (as in linear conjunctive grammars). These grammars are proved to be computationally equivalent to an extension of one-way real-time cellular automata with an extra data channel. The main result is the undecidability of the emptiness problem for grammars restricted to a one-symbol alphabet, which is proved by simulating a Turing machine by a cellular automaton with feedback. The same construction proves the Σ_2^0 -completeness of the finiteness problem for these grammars and automata.

Mathematics Subject Classification. 68Q42, 68Q80.

1. Introduction

The idea of defining context-free rules applicable only in certain contexts dates back to the early work of Chomsky. However, the mathematical model improvised by Chomsky, which he named a "context-sensitive grammar", turned out to be too powerful for its intended application, as it could simulate a space-bounded Turing machine. Recently, the authors [4] made a fresh attempt on implementing the same idea. Instead of the string-rewriting approach from the late 1950s, which never quite worked out for this task, the authors relied upon the modern understanding

 $Keywords\ and\ phrases.$ Context-free grammars, conjunctive grammars, contexts, cellular automata, trellis automata, undecidability..

^{*} This paper has been presented at the 11th Latin American Theoretical Informatics Symposium (LATIN 2014) held in Montevideo, Uruguay on March 31–April 4, 2014.

¹ Turku Centre for Computer Science, Turku 20014, Finland. mikbar@utu.fi

² Department of Mathematics and Statistics, University of Turku, Turku 20014, Finland.

of formal grammars as a first-order logic over positions in a string, discovered by Rounds [22]. This led to a family of grammars that allows such rules as $A \to BC \& \lhd D$, which asserts that all strings representable as a concatenation BC and preceded by a left context of the form D have the property A. The semantics of such grammars are defined through logical deduction of items of the form "a substring v written in left context u has a property A" [4], and the resulting formal model inherits some of the key properties of formal grammars, including parse trees, an extension of the Chomsky normal form [4], a form of recursive descent parsing [2] and a variant of the Cocke–Kasami–Younger parsing algorithm that works in time $O\left(\frac{n^3}{\log n}\right)$ [20].

This paper aims to investigate the *linear subclass* of grammars with one-sided contexts, where linearity is understood in the usual sense of formal grammars, that is, as a restriction to concatenate nonterminal symbols only to terminal strings. An intermediate family of *linear conjunctive grammars*, which allows using the conjunction operation, but no context specifications, was earlier studied by the second author [15,17]. Those grammars were found to be computationally equivalent to *one-way real-time cellular automata* [9,23], also known under a proper name of *trellis automata* [7,8,10,11,25].

This paper sets off by developing an analogous automaton representation for linear grammars with one-sided contexts. The proposed *trellis automata with feedback*, defined in Section 4, augment the original cellular automaton model by an extra communication channel. Its motivation comes from the understanding those automata as circuits with uniform connections [7], to which one can add a new type of connections. As the model is proved to be equivalent to linear grammars with one-sided contexts, it follows that this new type of connections has exactly the same power as context specifications do in grammars.

In the next Section 5, the intuition on grammars with contexts developed through their automaton representation is used to construct grammars for two usual examples of non-regular languages: for powers of k, $L_k = \{a^{k^n} \mid n \geq 0\}$, and for squares, $\{a^{n^2} \mid n \geq 1\}$. To compare, standard trellis automata over a one-symbol alphabet recognize only regular languages [8], whereas conjunctive grammars are known to describe powers of k [12,13], but using an entirely different method.

The next Section 6 presents a simulation of a Turing machine by a trellis automaton with feedback. Under some technical assumptions on the Turing machine, a simulating automaton, given an input a^n , performs $O(n^2)$ first steps of the machine's computation on an empty input. Accordingly, it can accept or reject the input a^n depending on the current state of the Turing machine.

That construction is used in Section 7 to prove the undecidability of the emptiness problem for linear grammars with one-sided contexts over a one-symbol alphabet. The finiteness problem for these grammars is proved to be complete for the second level of the arithmetical hierarchy.

Finally, the last Section 8 establishes some closure properties of the linear grammars with contexts. Here the automaton representation becomes particularly

useful, as it gives an immediate proof of the closure of this language family under complementation, which, using grammars alone, would require a complicated construction.

2. Grammars with one-sided contexts

Grammars with contexts were introduced by the authors [4] as a model capable of defining context-free rules applicable only in contexts of a certain form.

Definition 2.1 [4]. A grammar with left contexts is a quadruple $G = (\Sigma, N, R, S)$, where

- Σ is the alphabet of the language being defined;
- N is a finite set of auxiliary symbols ("nonterminal symbols" in Chomsky's terminology), disjoint with Σ , which denote the properties of strings defined in the grammar;
- R is a finite set of grammar rules, each of the form

$$A \to \alpha_1 \& \dots \& \alpha_k \& \triangleleft \beta_1 \& \dots \& \triangleleft \beta_m \& \triangleleft \gamma_1 \& \dots \& \triangleleft \gamma_n,$$
 (2.1)

with $A \in N$, $k \ge 1$, $m, n \ge 0$ and $\alpha_i, \beta_i, \gamma_i \in (\Sigma \cup N)^*$;

• $S \in N$ represents syntactically well-formed sentences of the language.

Every rule (2.1) is comprised of conjuncts of three kinds. Each conjunct α_i specifies the form of the substring being defined, a conjunct $\lhd \beta_i$ describes the form of its left context, while a conjunct $\unlhd \gamma_i$ refers to the form of the left context concatenated with the current substring. To be precise, let $w \in \Sigma^*$ be the whole string being defined, and consider defining its substring v by a rule (2.1), where v = uvx for $v, v, v \in \Sigma^*$. Then, each conjunct α_i describes the form of v, each left context operator $v \in \mathcal{C}$ describes the form of v, and each extended left context operator $v \in \mathcal{C}$ describes the form of v. The conjunction means that all these conditions must hold at the same time for this rule to be applicable.

If no context specifications are used in the grammar, that is, if m=n=0 in each rule (2.1), then this is a *conjunctive grammar* [15, 19]. If, furthermore, only one conjunct is allowed in each rule (k=1), this is an ordinary context-free grammar. A grammar is called *linear*, if every conjunct refers to at most one nonterminal symbol, that is, $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_m, \gamma_1, \ldots, \gamma_n \in \Sigma^* N \Sigma^* \cup \Sigma^*$.

To see how grammars with contexts are formally defined, one could begin with reconsidering the definition of ordinary grammars. Their usual definition, given by Chomsky, employs string rewriting to define a parse tree top-down: if there is a root S with descendants A and B, then one can represent that by rewriting S with AB, and then eventually rewrite A to a substring u and u to another substring u. Alternatively, one can regard the same process bottom-up, as a construction of larger and larger subtrees, each indicating that some substring has a certain property. One can first infer that u has the property u and that u has the property u and from these elementary propositions one can then deduce that u has

the property S. This outlook on ordinary grammars was presented, for instance, in a monograph by Kowalski ([14], Chap. 3), and it stands at the foundation of the logical approach to grammars, as defined by Rounds [22].

The definition of grammars with left contexts extend the latter logical interpretation of grammars. This time, it uses deduction of elementary statements of the form "a substring $v \in \Sigma^*$ in the left context $u \in \Sigma^*$ has the property $X \in \Sigma \cup N$ ", denoted by $X(u\langle v \rangle)$. A full definition applicable to every grammar with left contexts is presented in the authors's previous paper [4]; this paper gives a definition specialized for linear grammars.

Definition 2.2. Let $G = (\Sigma, N, R, S)$ be a linear grammar with left contexts, and consider deduction of items of the form $X(u\langle v\rangle)$, with $u, v \in \Sigma^*$ and $X \in N$. Each rule $A \to w$, with $w \in \Sigma^*$, defines an axiom scheme

$$\vdash_G A(x\langle w \rangle),$$

for all $x \in \Sigma^*$. Each rule of the form

defines the following scheme for deduction rules for all $u, v \in \Sigma^*$:

$$\left\{B_i(ux_i\langle v_i\rangle)\right\}_{i\in\{1,\dots,k\}}, \left\{D_i(x_i'\langle u_i\rangle)\right\}_{i\in\{1,\dots,m\}}, \left\{E_i(x_i''\langle w_i\rangle)\right\}_{i\in\{1,\dots,n\}} \vdash_G A(u\langle v\rangle),$$

where $x_i v_i y_i = v$, $x_i' u_i y_i' = u$ and $x_i'' w_i y_i'' = uv$. Then the language defined by a nonterminal symbol A is

$$L_G(A) = \{ u\langle v \rangle \mid u, v \in \Sigma^*, \vdash_G A(u\langle v \rangle) \}.$$

The language defined by the grammar G is the set of all strings with an empty left context defined by S.

$$L(G) = \{ w \mid w \in \Sigma^*, \vdash_G S(\varepsilon \langle w \rangle) \}$$

This definition is illustrated in the grammar below. The grammar can be regarded as trivial, but it is useful for demonstrating how derivations work.

Example 1. The following grammar defines the singleton language $\{abac\}$.

$$S \to aBc$$

$$B \to bA \& \lhd A$$

$$A \to a$$

The derivation of the string abac begins by deriving the inner substring ba from B. First, the rule $A \to a$ is defines the symbol a in the empty context and in the context ab.

$$\vdash A(\varepsilon \langle a \rangle) \qquad (A \to a)$$

$$\vdash A(ab\langle a \rangle) \qquad (A \to a)$$

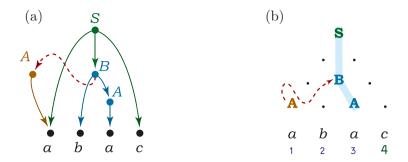


FIGURE 1. Derivation of the string *abac* in Example 1: (a) as a usual parse tree; (b) as an informal bottom-up illustration.

Then the rule $B \to bA \& \lhd A$ defines the substring $a\langle ba \rangle$ by concatenating b to a one-symbol string $ab\langle a \rangle$ derived from A. Furthermore, this rule requires the left context of $a\langle ba \rangle$ to be described by A, and this has also been derived above.

$$A(ab\langle a \rangle), A(\varepsilon\langle a \rangle) \vdash B(a\langle ba \rangle)$$
 $(B \to bA \& \lhd A)$

Finally, the rule $S \to aBc$ concatenates the substrings a, ba and c to obtain the desired string abac.

$$B(a\langle ba\rangle) \vdash S(\varepsilon\langle abac\rangle)$$
 $(S \to aBc)$

This derivation of the string w = abac is illustrated in Figure 1, where elements of the triangle correspond to non-empty substrings of w, and each element contains all nonterminal symbols generating the corresponding substring. The dotted line shows the effect of a context operator $(\triangleleft A)$, whereas the band rising up from the second A through B to S demonstrates the rules appending symbols to substrings.

Several non-trivial examples of grammars with one-sided contexts were given in the original paper [4]. One of them described an abstract language representing declaration before use.

Example 2 ([4], Ex. 2). Consider the language

$$\{u_1 \dots u_n \mid \text{for every } u_i, \text{ either } u_i = a^*c,$$

or $u_i = b^k c$ and there exists j < i, for which $u_j = a^k c$.

It is generated by the following grammar.

$$\begin{split} S &\rightarrow AS \mid CS \mid \varepsilon \\ A &\rightarrow aA \mid c \\ B &\rightarrow bB \mid c \\ C &\rightarrow B \& \leqslant EFc \\ E &\rightarrow AE \mid BE \mid \varepsilon \\ F &\rightarrow aFb \mid cE \end{split}$$

Substrings of the form a^kc represent declarations, and every substring b^kc is a reference to a declaration a^kc . Nonterminal S defines strings of the form $u_1 \dots u_\ell \langle u_{\ell+1} \dots u_n \rangle$ (with $u_i \in a^*c \cup b^*c$), such that every reference u_i in the suffix $u_{\ell+1} \dots u_n$ has a corresponding earlier declaration in the prefix $u_1 \dots u_{i-1}$. The rule $S \to AS$ appends a declaration, and the rules $S \to CS$ and $C \to B \& \leqslant EFc$ append a reference with a matching earlier declaration. The context $\leqslant EFc$ checks the declaration, with E representing the prefix of the string up to the declaration, and with F matching the symbols a in the declaration to the symbols b in the reference.

Though the given grammar is not linear as it is, it can be converted to linear as follows. The first non-linear rule $S \to AS$ expresses a concatenation of a^*c with S; in a linear grammar, this concatenation can be defined by a new nonterminal A_S with the rules $A_S \to aA_S$ and $A_S \to cS$. In a similar way, one can express concatenations AE and BE in the rules for E.

The rules $S \to CS$ and $C \to B \& \leq EFc$ concatenate a string from b^*c to S, and also apply a context operator to this string. In a linear grammar, this is done by a new nonterminal C_S , which has two rules, $C_S \to bC_S$ and $C_S \to cS \& \lhd E_F$. These rules simulate concatenation of a string from b^*c to S, and the latter rule applies a context operator, where a new symbol E_F simulates the concatenation EF.

The ideas from Example 2 can be used to describe a full grammar for the set of well-formed programs in a model programming language [3].

The language described in the next example is known to have no linear conjunctive grammar [25].

Example 3 (Törmä [24]). The following linear grammar with contexts defines the language $\{a^nb^{in} \mid i,n \ge 1\}$.

$$S \to ab \mid aSb \mid aB$$
$$B \to bB \mid b \& \lhd S$$

The languages defined by nonterminals B and S are as follows.

$$L(B) = \{ a^n b^{in-j} \langle b^j \rangle \mid n \geqslant 1, \ 0 \leqslant j \leqslant in+1 \}$$

$$L(S) = \{ a^j \langle a^{n-j} b^{in-j} \rangle \mid n \geqslant 1, \ 0 \leqslant j \leqslant n \}$$

For each $n \ge 1$, the strings $a^n b^{in}$, with $i \ge 1$, are defined inductively on i. The first string $a^n b^n$ is defined by the first two rules for nonterminal S. In Figure 2, this is illustrated by the leftmost vertical S-column.

Every next string $w' = a^n b^{(i+1)n}$ is obtained from the previous string $w = a^n b^{in}$ by appending as many symbols b as there are symbols a in w. First, a context operator in the rule $B \to b \& \lhd S$ produces a one-symbol string $a^n b^{in} \langle b \rangle$. Then, B generates the string $a^n \langle b^{in+1} \rangle$ by the rule $B \to bB$ applied $i \cdot n$ times, as shown in a B-diagonal in the figure.

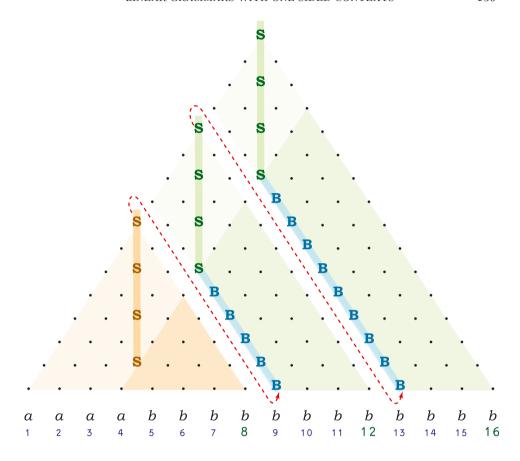


FIGURE 2. The grammar in Example 3 defining the language $\{a^nb^{in} \mid i,n \geqslant 1\}$.

Next, the rule $S \to aB$ generates the string $a^{n-1}\langle ab^{in+1}\rangle$, thus setting up the matching of the new symbols b to the existing symbols a. Then, each application of the rule $S \to aSb$ appends one a and one b, and, overall, it appends as many symbols b as there as as in the beginning of the string, as illustrated in an S-column emerging from a B-diagonal. The resulting string with an empty context is $\varepsilon\langle a^nb^{(i+1)n}\rangle$, as desired.

3. Linear grammars and normal form

It is known [4,20], that every grammar with contexts can be transformed to a certain normal form, which extends the Chomsky normal form for ordinary context-free grammars. While the original Chomsky normal form has all rules of

the form $A \to BC$ and $A \to a$, its extension for grammars with contexts allows using multiple conjuncts BC and context specifications $\triangleleft D$.

A similar normal form shall now be established for the linear subclass of grammars.

Theorem 3.1. For every linear grammar with left contexts $G = (\Sigma, N, R, S)$, there exists another linear grammar with left contexts $G' = (\Sigma, N', R', S)$ that defines the same language and has all rules of the form

$$A \rightarrow bB_1 \& \dots \& bB_\ell \& C_1 c \& \dots \& C_k c$$
 (3.1a)

$$A \to a \& \triangleleft D_1 \& \dots \& \triangleleft D_m,$$
 (3.1b)

where $A, B_i, C_i, D_i \in \mathbb{N}$, $a, b, c \in \Sigma$, $\ell + k \geqslant 1$ and $m \geqslant 0$.

The size of G' is at most triple exponential in the size of G, where the size is measured by the total number of symbols used in the description of the grammar.

Sketch of a proof. The transformation is carried out along the same lines as in the general case [4]. Given an arbitrary linear grammar with contexts $G = (\Sigma, N, R, S)$, its transformation to the normal form starts with a preprocessing phase: long conjuncts are cut until all of them are of the form bB, Cc or a, and every context specification $\triangleleft \gamma$ or $\triangleleft \gamma$ with $\gamma \in \Sigma$ or $|\gamma| > 1$ is restated as $\triangleleft X_{\gamma}$ or $\triangleleft X_{\gamma}$, respectively, where X_{γ} is a new nonterminal with a unique rule $X_{\gamma} \rightarrow \gamma$.

This results in a grammar $G_1 = (\Sigma, N_1, R_1, S)$ with the rules of the following form:

$$A \to bB$$
 (3.2a)

$$A \to Cc$$
 (3.2b)

$$A \to a$$
 (3.2c)

$$A \to B_1 \& \dots \& B_k \& \triangleleft D_1 \& \dots \& \triangleleft D_m \& \triangleleft E_1 \& \dots \& \triangleleft E_n$$
 (3.2d)

$$A \to \varepsilon$$
, (3.2e)

where $a, b, c \in \Sigma$ and $A, B_i, D_i, E_i \in N$.

The transformation continues with the elimination of *null conjuncts* ([4], Sect. 4.1), that is, any rules of the form $A \to \dots \& \varepsilon$. First, one has to determine, which nonterminals generate the empty string, and in which contexts they generate it. This information is represented in a set Nullable(G) $\subseteq 2^N \times N$. Given a nonterminal $A \in N$ and a string $u \in \Sigma^*$, the string $u\langle \varepsilon \rangle$ is defined by A if and only if the set Nullable(G) contains a pair ($\{K_1, \dots, K_t\}, A$), with $A, K_1, \dots, K_t \in N$, such that $\varepsilon\langle u \rangle \in L_G(K_1), \dots, \varepsilon\langle u \rangle \in L_G(K_t)$.

Using the set NULLABLE(G), a new grammar $G_2 = (\Sigma, N_1, R_2, S)$ without null conjuncts can be constructed as follows.

- (1) The rules of the form (3.2a)–(3.2d) are copied to the new grammar.
- (2) For every rule of the form (3.2a) and for every pair $(\{K_1, \ldots, K_t\}, B) \in \text{Nullable}(G)$, a rule $A \to b \& \triangleleft K_1 \& \ldots \& \triangleleft K_t$ is added to the new grammar.

- (3) For every rule of the form (3.2b) and for every pair $(\{K_1, \ldots, K_t\}, C) \in \text{NULLABLE}(G)$, the new grammar has the rule $A \to c \& \lhd K_1 \& \ldots \& \lhd K_t$. Moreover, if $\varepsilon \langle \varepsilon \rangle \in L_G(K_i)$ for all $i \in \{1, \ldots, t\}$, then a rule $A \to c \& \lhd \varepsilon$ should be added.
- (4) For every rule of the form (3.2d), a rule $A \to B_1 \& ... \& B_k \& E_1 \& ... \& E_n \& \triangleleft \varepsilon$ shall be added to the new grammar, if $\varepsilon \langle \varepsilon \rangle \in L_G(D_i)$ for all $i \in \{1, ..., m\}$.

After this step, the rules of the grammar can be of the following (linear) form:

$$A \to a$$
 (3.3a)

$$A \to bB$$
 (3.3b)

$$A \to Cc$$
 (3.3c)

$$A \to B_1 \& \dots \& B_k \& \triangleleft D_1 \& \dots \& \triangleleft D_m \& \triangleleft E_1 \& \dots \& \triangleleft E_n$$
 (3.3d)

$$A \to B_1 \& \dots \& B_k \& \triangleleft \varepsilon$$
 (3.3e)

$$A \to b \& \triangleleft K_1 \& \dots \& \triangleleft K_t$$
 (3.3f)

$$A \to c \& \triangleleft K_1 \& \dots \& \triangleleft K_t$$
 (3.3g)

$$A \to c \& \triangleleft \varepsilon$$
 (3.3h)

Having constructed this grammar, one can apply verbatim the rest of the steps of the general transformation to the normal form. This transformation constitues in elimination of *null contexts* $\lhd \varepsilon$ ([4], Sect. 4.2) and of *unit conjuncts* ([4], Sect. 4.3), as in the rules $A \to \ldots \& B$. The final step is elimination of *extended left contexts* $\lhd E$, which are all expressed through proper left contexts $\lhd D$ [20]. Each step preserves the linearity of a grammar.

4. Automaton representation

Linear conjunctive grammars are known to be computationally equivalent to one of the simplest types of cellular automata: the *one-way real-time cellular automata*, also known under the proper name of *trellis automata*. This section presents a generalization of trellis automata, which similarly corresponds to linear grammars with one-sided contexts.

A standard trellis automaton processes an input string of length $n \ge 1$ using a uniform array of $\frac{n(n+1)}{2}$ nodes, as presented in Figure 3a. Each node computes a value from a fixed finite set Q. The nodes in the bottom row obtain their values directly from the input symbols using a function $I \colon \Sigma \to Q$. The rest of the nodes compute the function $\delta \colon Q \times Q \to Q$ of the values in their predecessors. The string is accepted if and only if the value computed by the topmost node belongs to the set of accepting states $F \subseteq Q$.

Theorem 4.1 Okhotin [17]. A language $L \subseteq \Sigma^+$ is defined by a linear conjunctive grammar if and only if L is recognized by a trellis automaton.

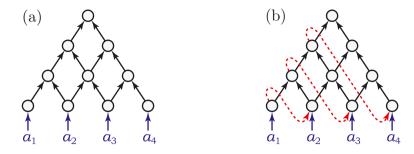


FIGURE 3. (a) Trellis automata; (b) trellis automata with feedback.

In terms of cellular automata, every horizontal row of states in Figure 3a represents an automaton's configuration at a certain moment of time. An alternative motivation developed in the literature on trellis automata [7–9] is to consider the entire grid as a digital circuit with uniform structure of connections. In order to obtain a similar representation of linear grammars with left contexts, the trellis automaton model is extended with another type of connections, illustrated in Figure 3b.

Definition 4.2. A trellis automaton with feedback is a sextuple $M = (\Sigma, Q, I, J, \delta, F)$, in which:

- Σ is the input alphabet,
- Q is a finite non-empty set of states,
- $I: \Sigma \to Q$ is a function that sets the initial state for the first symbol,
- $J: Q \times \Sigma \to Q$ sets the initial state for every subsequent symbol, using the state computed on the preceding substring as a feedback;
- $\delta \colon Q \times Q \to Q$ is the transition function, and
- $F \subseteq Q$ is the set of accepting states.

The behaviour of the automaton is described by a function $\Delta \colon \Sigma^* \times \Sigma^+ \to Q$, which defines the state $\Delta(u\langle v\rangle)$ computed on each string $u\langle v\rangle$. The value of this function is defined by the following formulae formalizing the connections in Figure 3b.

$$\Delta(\varepsilon\langle a \rangle) = I(a)$$

$$\Delta(w\langle a \rangle) = J(\Delta(\varepsilon\langle w \rangle), a)$$

$$\Delta(u\langle bvc \rangle) = \delta(\Delta(u\langle bv \rangle), \Delta(ub\langle vc \rangle))$$

The language recognized by the automaton is $L(M) = \{ w \in \Sigma^+ \mid \Delta(\varepsilon \langle w \rangle) \in F \}.$

Theorem 4.3. A language $L \subseteq \Sigma^+$ is defined by a linear grammar with left contexts if and only if L is recognized by a trellis automaton with feedback.

The proof is by effective constructions in both directions.

Lemma 4.4. Let $G = (\Sigma, N, R, S)$ be a linear grammar with left contexts, in which every rule is of the form

$$A \to bB_1 \& \dots \& bB_\ell \& C_1 c \& \dots \& C_k c \qquad (b, c \in \Sigma, B_i, C_i \in N),$$

$$A \to a \& \triangleleft D_1 \& \dots \& \triangleleft D_m \qquad (a \in \Sigma, m \geqslant 0, D_i \in N),$$

$$(4.1a)$$

and define a trellis automaton with feedback $M = (\Sigma, Q, I, J, \delta, F)$ by setting $Q = \Sigma \times 2^N \times \Sigma$,

$$I(a) = (a, \{ A \mid A \to a \in R \}, a)$$

$$J((b, X, c), a) = (a, \{ A \mid \exists rule \ (4.1b) \ with \ D_1, \dots, D_m \in X \}, a)$$

$$\delta((b, X, c'), (b', Y, c)) = (b, \{ A \mid \exists rule \ (4.1a) \ with \ B_i \in X \ and \ C_i \in Y \}, c)$$

$$F = \{ (b, X, c) \mid S \in X \}.$$

For every string with context $u\langle v \rangle$, let b be the first symbol of v, let c be the last symbol of v, and let $Z = \{ A \mid u\langle v \rangle \in L_G(A) \}$. Then $\Delta(u\langle v \rangle) = (b, Z, c)$. In particular, $L(M) = \{ w \mid \varepsilon\langle w \rangle \in L_G(S) \} = L(G)$.

Proof. Induction on pairs (|uv|, |v|), ordered lexicographically.

Basis: $\varepsilon\langle a\rangle$ with $a\in \Sigma$. The state computed on this string is $\Delta(\varepsilon\langle a\rangle)=I(a)=(a,Z,a)$ with $Z=\{A\mid A\to a\in R\}$. The latter set Z is the set of all symbols $A\in N$ with $\varepsilon\langle a\rangle\in L_G(A)$.

Induction Step I: $u\langle a\rangle$ with $u\in \Sigma^*$ and $a\in \Sigma$. The state computed by the automaton on the string $u\langle a\rangle$ is defined as $\Delta(u\langle a\rangle))=J(\Delta(\varepsilon\langle u\rangle),a)$. By the induction hypothesis, the state reached on the string $\varepsilon\langle u\rangle$ is $\Delta(\varepsilon\langle u\rangle)=(a,X,a)$, where a is the first symbol of u and $X\subseteq N$ is the set of symbols that generate $\varepsilon\langle u\rangle$. Substituting this value into the expression for the state reached on $u\langle a\rangle$ yields $\Delta(u\langle a\rangle)=J((a,X,a),a)=(a,Z,a)$, where

$$Z = \{ A \mid \text{there exists a rule (4.1b) with } D_1, ..., D_m \in X \}$$

= $\{ A \mid \text{there exists a rule (4.1b) with } \varepsilon \langle u \rangle \in L_G(D_i) \text{ for all } i \}.$

The latter condition means that Z is the set of all symbols $A \in N$ that generate the string $u\langle a \rangle$ using a rule of the form (4.1b). Since this string can only be generated by rules of that form, this is equivalent to $Z = \{A \mid u\langle a \rangle \in L_G(A)\}$, as claimed.

Induction Step II: $u\langle bvc \rangle$ with $u,v \in \Sigma^*$ and $b,c \in \Sigma$. The state computed on such a string is $\Delta(u\langle bvc \rangle) = \delta(\Delta(u\langle bv \rangle), \Delta(ub\langle vc \rangle))$. By the induction hypothesis, the states reached by the automaton on the strings $u\langle bv \rangle$ and $ub\langle vc \rangle$ are respectively $\Delta(u\langle bv \rangle) = (b,X,b')$ and $\Delta(ub\langle vc \rangle) = (c',Y,c)$, where b is the last symbol of bv,c' is the first symbol of $vc,X\subseteq N$ is the set of nonterminal symbols generating $u\langle bv \rangle$ and $Y\subseteq N$ contains all such nonterminals that generate the string $ub\langle vc \rangle$.

Substituting the states reached on these shorter strings into the expression for the state computed on $u\langle bvc\rangle$ gives $\Delta(u\langle bvc\rangle) = \delta((b,X,b'),(c',Y,c)) = (b,Z,c)$, where

$$Z = \{ A \mid \text{there exists a rule (4.1a) with } B_i \in X \text{ and } C_i \in Y \}$$

$$= \{ A \mid \text{there exists a rule (4.1a) with } u\langle bv \rangle \in L_G(B_i) \text{ and }$$

$$ub\langle vc \rangle \in L_G(C_i), \text{ for all } i, j \}.$$

That is, Z is exactly the set of nonterminals that generate the string $ub\langle vc\rangle$ by a rule of the form (4.1a). The string $u\langle bvc\rangle$ can only be generated by a rule of such a form, and, thus, $Z = \{A \mid u\langle bvc\rangle \in L_G(A)\}$, as desired.

Lemma 4.5. Let $M = (\Sigma, Q, I, J, \delta, F)$ be a trellis automaton with feedback and define the grammar with left contexts $G = (\Sigma, N, R, S)$, where $N = \{A_q \mid q \in Q\} \cup \{S\}$, and the set R contains the following rules.

$$A_{I(a)} \to a \& \lhd \varepsilon$$
 $(a \in \Sigma)$ (4.2a)

$$A_{J(q,a)} \to a \& \triangleleft A_q$$
 $(q \in Q, a \in \Sigma)$ (4.2b)

$$A_{\delta(p,q)} \to bA_q \& A_p c$$
 $(p, q \in Q, b, c \in \Sigma)$ (4.2c)

$$S \to A_q$$
 $(q \in F)$ $(4.2d)$

Then, for every string with context $u\langle v \rangle$, $\Delta(u\langle v \rangle) = r$ if and only if $u\langle v \rangle \in L_G(A_r)$. In particular, $L(G) = \{ w \mid \Delta(\varepsilon \langle w \rangle) \in F \} = L(M)$.

Proof. Induction on lexicographically ordered pairs (|uv|, |v|).

Basis: $\varepsilon\langle a\rangle$ with $a\in \Sigma$. Then $\Delta(\varepsilon\langle a\rangle)=I(a)$. At the same time, $\varepsilon\langle a\rangle$ may only be generated by the rule of the form (4.2a), and such a rule for A_r exists if and only if I(a)=r.

Induction Step I: $u\langle a\rangle$ with $u\in \Sigma^+$ and $a\in \Sigma$.

 \bigoplus Let $\Delta(u\langle a\rangle) = r$. Then, $r = J(\Delta(\varepsilon\langle u\rangle), a)$. Let $q = \Delta(\varepsilon\langle u\rangle)$. By the induction hypothesis, $\varepsilon\langle u\rangle \in L_G(A_q)$. Since J(q, a) = r, the grammar contains a corresponding rule of the form (4.2b), which can be used to deduce the membership of $u\langle a\rangle$ in $L_G(A_r)$ as follows.

$$A_q(\varepsilon\langle u\rangle) \vdash_G A_r(u\langle a\rangle)$$
 $(A_r \to a \& \lhd A_q)$ (4.3)

 \bigoplus Conversely, assume that $u\langle a\rangle \in L_G(A_r)$. Then its deduction must end with an application of a rule of the form (4.2b), as in (4.3). By construction, the existence of such a rule implies r=J(q,a). Applying the induction hypothesis to $A_q(\varepsilon\langle u\rangle)$ yields $\Delta(\varepsilon\langle u\rangle)=q$. Then the automaton calculates as follows: $\Delta(u\langle a\rangle)=J(\Delta(\varepsilon\langle u\rangle),a)=J(q,a)=r$, as desired.

Induction Step II: $u\langle bvc \rangle$ with $u, v \in \Sigma^*$ and $b, c \in \Sigma$.

 \bigoplus Assume first that $\Delta(u\langle bvc\rangle) = r$. Then $r = \delta(p,q)$, where $p = \Delta(u\langle bv\rangle)$ and $q = \Delta(ub\langle vc\rangle)$. By the induction hypothesis, $u\langle bv\rangle \in L_G(A_p)$ and $ub\langle vc\rangle \in L_G(A_q)$. From this, using a rule of the form (4.2c), one can deduce

$$A_p(u\langle bv\rangle), A_q(ub\langle vc\rangle) \vdash_G A_r(u\langle bvc\rangle)$$
 $(A_r \to bA_q \& A_pc),$ (4.4)

that is, $u\langle bvc\rangle \in L_G(A_r)$, as claimed.

 \bigoplus Conversely, if $u\langle bvc \rangle \in L_G(A_r)$, then the deduction establishing $A_r(u\langle bvc \rangle)$ must end as (4.4), using a rule of the form (4.2c). Then, by the construction, $r = \delta(p,q)$. Since the items $A_p(u\langle bv \rangle)$ and $A_q(ub\langle vc \rangle)$ are deduced in the grammar, by the induction hypothesis, $\Delta(u\langle bv \rangle) = p$ and $\Delta(ub\langle vc \rangle) = q$. Then $\Delta(u\langle bvc \rangle) = \delta(p,q) = r$.

5. Defining non-regular unary languages

Ordinary context-free grammars over a unary alphabet $\Sigma = \{a\}$ define only regular languages. Unary linear conjunctive languages are also regular, because a trellis automaton operates on an input a^n as a deterministic finite automaton [8]. The non-triviality of unary conjunctive grammars was discovered by Jeż [12], who constructed a grammar for the language $\{a^{4^k} \mid k \geqslant 0\}$ using iterated conjunction and concatenation of languages.

It turns out that one can also describe some non-regular languages over a unary alphabet using a left context operator, and without relying upon non-linear concatenation. The simplest example of that is the following grammar.

Example 4. For each base $k \ge 2$, the language $L_k = \{a^{k^n} \mid n \ge 0\}$ is generated by the following grammar.

$$S \to a \,\&\, \lhd \varepsilon \mid \varepsilon \,\&\, \lhd S \mid aSa^{k-1}$$

The grammar in Example 4 can be transformed to an automaton by Lemma 4.4. Actually, a succinct automaton can be constructed directly as follows.

Example 5. Consider a trellis automaton with feedback $M = (\Sigma, Q, I, J, \delta, F)$ over the alphabet $\Sigma = \{a\}$ and with the set of states $Q = \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$, where $I(a) = \mathbf{p}$ is the initial state, the feedback function gives states $J(\mathbf{p}, a) = \mathbf{q}$ and $J(\mathbf{r}, a) = \mathbf{p}$, and the transition function is defined by $\delta(t, \mathbf{p}) = \mathbf{p}$ for all $t \in Q$, $\delta(\mathbf{q}, \mathbf{q}) = \delta(\mathbf{r}, \mathbf{q}) = \mathbf{q}$, $\delta(\mathbf{p}, \mathbf{q}) = \mathbf{r}$ and $\delta(\mathbf{p}, \mathbf{r}) = \mathbf{p}$. The only accepting state is \mathbf{r} . Then M recognizes the language $\{a^{2^k-2} \mid k \geq 2\}$.

The computation of this automaton is illustrated in Figure 4. The state computed on each one-symbol substring $a^{\ell}\langle a\rangle$ is determined by the state computed on $\varepsilon\langle a^{\ell}\rangle$ according to the function J. Most of the time, $\Delta(\varepsilon\langle a^{\ell}\rangle) = \mathbf{p}$ and hence $\Delta(a^{\ell}\langle a\rangle) = \mathbf{q}$, and the latter continues into a triangle of states \mathbf{q} . Once for every power of two, the automaton computes the state \mathbf{r} on $\varepsilon\langle a^{2^k-2}\rangle$, which sends a signal through the feedback channel, so that J sets $\Delta(a^{2^k-2}\langle a\rangle) = \mathbf{p}$. This in turn produces the triangle of states \mathbf{p} and the next column of states \mathbf{r} .

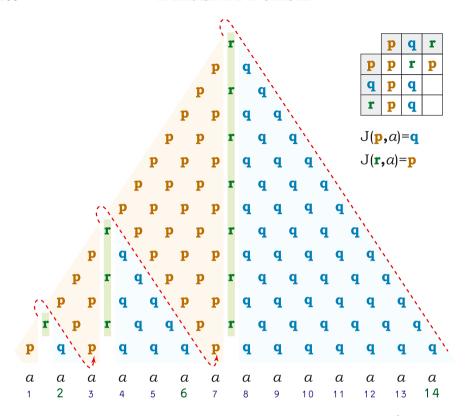


FIGURE 4. How the automaton in Example 5 recognizes $\{a^{2^k-2} \mid k \ge 2\}$.

Another example of a unary linear grammar with contexts defines the set of squares over a unary alphabet. This example is based upon an idea of Birman and Ullman [6], who used it to recognize the set of squares by a special recursive descent parser with backtracking.

Example 6 (adapted from Birman and Ullman [6], p. 21). The following linear grammar with contexts describes the language $\{a^{n^2} \mid n \geqslant 1\}$.

$$\begin{split} S &\rightarrow a \& \lhd \varepsilon \mid aaaa \& \lhd \varepsilon \mid aBaa \& \lhd S \mid aS \\ B &\rightarrow a \& \lhd S \mid aBa \& \lhd C \\ C &\rightarrow a \& \lhd S \mid aC \mid Ba \end{split}$$

The grammar is centered around the nonterminal S, which should define, among others, strings of the form $\varepsilon\langle a^m\rangle$, with m being a perfect square. The two shortest strings $\varepsilon\langle a\rangle$ and $\varepsilon\langle a^4\rangle$ are explicitly defined in the first two rules for S. Other prefixes of the string, that is, $\varepsilon\langle a^m\rangle$, where m is not a perfect square, are defined by the symbol C, which acts more or less as the complement of S.

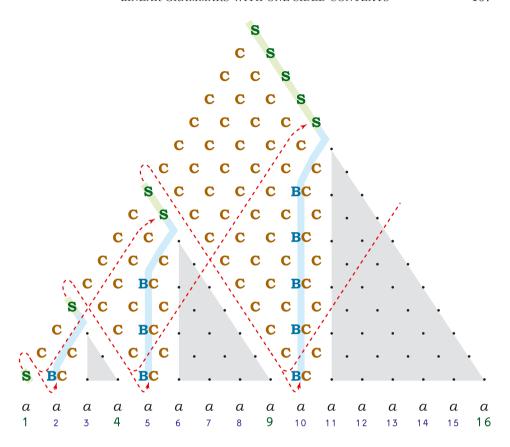


FIGURE 5. The grammar in Example 6 describing the language $\{a^{n^2} \mid n \geqslant 1\}$.

Nonterminal B generates vertical B-columns, each beginning in a one-symbol string $a^{n^2}\langle a\rangle$, with $n\geqslant 1$, and then rising up through $a^{n^2-i}\langle a^{2i+1}\rangle$, for all $i=1,\ldots,2n-2$. The rule $B\to a\& \lhd S$ defines the bottom element of every such column, and the companion rule $C\to a\& \lhd S$ simultaneously defines symbol C in the same position. Every next element of a vertical B-column is defined from the previous element by the rule $B\to aBa\& \lhd C$. The context C in this rule is satisfied unless the left context is a perfect square. Therefore, the height of this column of Bs is equal to the number of symbols C between the two previous occurrences of a perfect square in the main diagonal. Thus, the height of every consecutive B-column is by 2 greater than the height of the previous one.

The symbols C are spawned from B-columns. The rule $C \to Ba$ defines an initial C-column to the right of a B-column, whereas the rule $C \to aC$ propagates C towards the main diagonal.

The nonterminal S generates S-diagonals, starting with a string $a^{(n-2)^2}\langle a^{n^2}\rangle$ and propagating up to desired strings of the form $\varepsilon\langle a^{n^2}\rangle$. The rule $S\to aBaa\ \& \lhd S$ defines the bottom element of an S-diagonal as a combination of two conditions. First, aBaa makes a jump from the topmost symbol B in a B-column in a fixed direction and distance (this can be done from any B in this column). At the same time, the context S in this rule refers to the next to last perfect square. These two conditions together define the bottom of an S-diagonal; the rule $S\to aS$ propagates symbols S towards the main diagonal.

The languages generated by the nonterminals S, B and C are as follows.

$$\begin{split} L(S) &= \{ \, a^i \langle a^{n^2 - i} \rangle \mid n \geqslant 1, \, 0 \leqslant i \leqslant (n - 2)^2 \, \} \\ L(B) &= \{ \, a^{n^2 - i} \langle a^{2i + 1} \rangle \mid n \geqslant 1, \, 0 \leqslant i \leqslant 2n - 2 \, \} \\ L(C) &= \{ \, a^i \langle a^{n^2 + j} \rangle \mid n \geqslant 1, \, 1 \leqslant j \leqslant 2n, \, 0 \leqslant i \leqslant \min\{n^2 - j + 2, n^2\} \, \} \end{split}$$

In the empty context, S generates all strings of the form $\varepsilon \langle a^{n^2} \rangle$, with $n \ge 1$. This is the language generated by the grammar.

6. Simulating a Turing machine

It is now known that linear grammars with contexts over a one-symbol alphabet are non-trivial. How far does their expressive power go? For conjunctive grammars (which allow non-linear concatenation, but no context specifications), Jeż and Okhotin [13] developed a method for manipulating base-k notation of the length of a string in a grammar, which allowed the following language to be represented: for every trellis automaton M over an alphabet $\{0,1,\ldots,k-1\}$, there is a conjunctive grammar generating $L_M = \{a^{\ell} \mid \text{the base-}k \text{ notation of } \ell \text{ is in } L(M) \}$ [13]. This led to the following undecidability method: given a Turing machine T, one first constructs a trellis automaton M for the language VALC $(T) \subseteq \Sigma^*$ of computation histories of T; then, assuming that the symbols in Σ are digits in some base-k notation, one can define the unary version of VALC(T) by a conjunctive grammar.

Linear grammars with contexts are an entirely different model, and the grammar in Example 4 has nothing in common with the basic unary conjunctive grammar discovered by Jeż [12], in spite of defining almost the same language. Nevertheless, the general undecidability method also works for linear grammars with contexts, using different base unary languages.

The idea is that given a Turing machine, one shall construct a trellis automaton with feedback that simulates $\mathcal{O}(n)$ steps of the machine and accepts the input a^n , which is a string over a unary alphabet, depending on the state of the computation of the Turing machine at a certain time. Each individual cell, computed by a trellis automaton with feedback, should hold information about the computation of the Turing machine, such as the contents of a certain tape square at a certain time.

In order to simulate a Turing machine in such a way, it is useful to assume a machine of the following special kind. This machine operates on an initially blank

tape, which is infinite to the left, and proceeds by making sweeps from left to right over the tape. When the machine arrives at the end of the tape, it appends a blank tape square to the left and begins a new sweep from that blank square.

Definition 6.1. A rotating Turing machine is a quintuple $T = (\Gamma, \mathcal{S}, s_0, \nabla, \mathcal{F})$, where

- Γ is a finite tape alphabet containing a blank symbol $\square \in \Gamma$;
- S is a finite set of *states*;
- $s_0 \in \mathcal{S}$ is the *initial state*;
- the function $\nabla \colon \mathcal{S} \times \Gamma \to \mathcal{S} \times \Gamma$ determines the next move of the Turing machine and is called a *move function*, to distinguish it from transition functions of trellis automata;
- \mathcal{F} is a finite set of *flickering states*.

A configuration of T is a string of the form $[\![k]\!]usav$, where $k \ge 1$ is the number of the sweep, and usav with $u, v \in \Gamma^*$, $a \in \Gamma$ and $s \in \mathcal{S}$ represents the tape contents uav with the head scanning the symbol a in the state s.

The initial configuration of the machine is $[1]s_0\square$. Each kth sweep of the machine deals with a tape with k symbols, and consists of k steps of the following form.

$$[\![k]\!]uscdv \vdash_T [\![k]\!]uc's'dv \qquad (\nabla(s,c) = (s',c'))$$

After the last move of a sweep, the machine appends a blank symbol to the tape and then begins another sweep.

$$[\![k]\!]wsc \vdash_T [\![k+1]\!]s' \square wc' \qquad \qquad (\nabla(s,c) = (s',c'))$$

A rotating Turing machine never halts; at the end of each sweep, it may *flicker* by entering a state from \mathcal{F} . Define the set of numbers accepted by T as $S(T) = \{k \mid [\![1]\!] s_0 \square \vdash_T^* [\![k]\!] s_f cw \text{ for } s_f \in \mathcal{F} \}.$

A similar class of Turing machines was studied by Ibarra and Kim [10, 11]. Unlike the model defined in this section, the state of their machine is reset to the initial state in the beginning of each new sweep. Because of this, these devices are trivial over a unary alphabet.

The aforementioned Turing machines by Ibarra and Kim [10] can be understood as a model equivalent to trellis automata. A similar result holds for rotating Turing machines: they can be proved equivalent to trellis automata with feedback.

Let $T = (\Gamma, \mathcal{S}, s_0, \nabla, \mathcal{F})$ be a rotating Turing machine. Construct a trellis automaton with feedback $M_T = (\{a\}, Q, I, J, \delta, F)$ for simulating the machine as follows. Its set of states is $Q = \{\mathbf{q}^{sc} \mid s \in \mathcal{S}, c \in \Gamma\}$, with each state \mathbf{q}^{sc} representing the Turing machine's being in a state $s \in \mathcal{S}$, with its head scanning a symbol $c \in \Gamma$. Let the initial state of the automaton be $I(a) = \mathbf{q}^{s_0 \square}$; this corresponds to the tape square $s_0 \square$ in the beginning of the simulation.

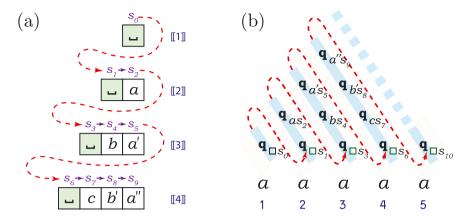


FIGURE 6. (a) Computation of a rotating Turing machine; (b) a trellis automaton with feedback simulating such a machine.

Each diagonal of the automaton corresponds to the kth sweep of the Turing machine, and the tape symbols written there represent the tape contents at the end of that sweep, as illustrated in Figure 6b. In this kind of a computation, each transition of the trellis automaton has to simulate two moves of the Turing machine T. For any two neighbouring states in the M_T 's computation, (s,c) and (t,d), the former state represents T in some kth sweep at some position i+1, whereas the latter state refers to T in the sweep number k+1 at the position i, as illustrated in Figure 7b. Then the trellis automaton has to determine the configuration in the (k+1)th sweep at position i+1. The Turing machine arrives to that configuration in the state produced at the last step by a move $\nabla(t,d) = (t',d')$. Here, it sees a symbol produced at the previous sweep by another move $\nabla(s,c) = (s',c')$. These data are combined in the following transition.

$$\delta\left(\mathbf{q}^{sc},\ \mathbf{q}^{td}\right) = \mathbf{q}^{t'c'}$$

After the last move of the machine in the current sweep, the feedback data channel J is used to append a new blank symbol to the bottom element of the next diagonal.

$$J(\mathbf{q}^{sc}) = \mathbf{q}^{s'\square} \qquad \qquad (\nabla(s,c) = (s',c'))$$

The set of accepting states of the automaton is $F = \{ \mathbf{q}^{cs_f} \mid c \in \Gamma, s_f \in \mathcal{F} \}$.

The correctness of the construction is established in the following lemma.

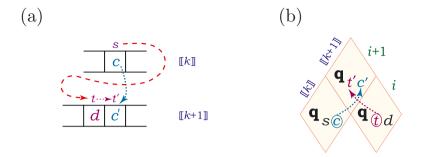


FIGURE 7. (a) Computation of a rotating Turing machine during kth and (k+1)th sweeps; (b) trellis automaton's cells in the kth and (k+1)th diagonals corresponding to those sweeps.

Lemma 6.2. Let $T = (\Gamma, S, s_0, \nabla, \mathcal{F})$ be a rotating Turing machine, and let $M_T = (\{a\}, Q, I, J, \delta, F)$ be the trellis automaton with feedback constructed from T as described above. Then $L(M_T) = \{a^k \mid k \in S(T)\}.$

The claim can be proved by a straightforward induction, inferring the state in each cell from the previously determined values of the neighbouring cells.

7. Undecidable problems

The simulation of Turing machines by trellis automata with feedback over a one-symbol alphabet is useful for proving undecidability of basic decision problems for these automata, such as emptiness or equivalence. Due to Theorem 4.3, the same results equally hold for linear grammars with contexts. Besides establishing the undecidability, the results state the exact level of undecidability of these problems in the arithmetical hierarchy.

The arithmetical hierarchy is defined as follows. A set is in Σ_k^0 , if it can be expressed as $\{y \mid \exists x_1 \forall x_2 \dots Q_k x_k \ P(x_1, \dots, x_k, y)\}$, for some recursive predicate P, where $Q_k = \exists$ if k is odd, and $Q_k = \forall$ if k is even. Similarly, a set is in Π_k^0 , if its complement is in Σ_k^0 , that is, if it admits a representation $\{y \mid \forall x_1 \exists x_2 \dots Q_k x_k \ P(x_1, \dots, x_k, y)\}$. In particular, the class Σ_1^0 at the first level of the arithmetical hierarchy is the class of recursively enumerable (r.e.) sets. Each class Σ_i^0 and Π_i^0 , with $i \geqslant 1$, has complete sets with respect to many-one reductions by Turing machines. For example, the Turing machine halting problem is Σ_1^0 -complete, whereas non-halting is Π_1^0 -complete.

The first decision problem for linear grammars with contexts is testing whether the language defined by a given grammar is empty. The undecidability of the *emptiness problem* follows from Lemma 6.2. To be precise, the problem is complete for the complements of the r.e. sets.

Theorem 7.1. The emptiness problem for linear grammars with left contexts over a one-symbol alphabet is Π_1^0 -complete. It remains in Π_1^0 for any alphabets.

Proof. The non-emptiness problem is clearly recursively enumerable, because one can simulate a trellis automaton with feedback on all inputs, accepting if it ever accepts. If the automaton accepts no strings, the algorithm does not halt.

The Π_1^0 -hardness is proved by reduction from the Turing machine halting problem (non-halting, to be precise). Given a machine T and an input w, construct a rotating Turing machine T_w , which first prints w on the tape (over $1 + \log |w|$ sweeps, using around |w| states), and then proceeds by simulating T, using one sweep for each step of T. If the simulated machine T ever halts, then T_w changes into a special state s_f and continues moving its head until the end of the current sweep.

Construct a trellis automaton with feedback M simulating the machine T_w according to Lemma 6.2, and define its set of accepting states as $F = \{ \mathbf{p}_{cs_f}^{\square} \mid c \in \Sigma \}$. Then, by the theorem, M accepts some string a^{ℓ} if and only if T_w ever enters the state s_f , which is in turn equivalent to T's halting on w.

For conjunctive grammars, there is a stronger result than just the undecidability of the emptiness problem. Namely, for every fixed conjunctive language L_0 over any alphabet, the problem of testing whether a given conjunctive grammar describes L_0 is Π_1^0 -complete ([19], Thms. 25 and 26). The same property holds for linear grammars with left contexts, and has a much simpler proof than for conjunctive grammars.

Corollary 7.2. For every fixed language L_0 defined by a linear grammar with left contexts, the problem of testing whether a given linear grammar with left contexts defines the language L_0 is Π_1^0 -complete.

Proof. The problem is in Π_1^0 , because its complement, the *inequivalence to* L_0 , is recursively enumerable, solved by simulating a trellis automaton with feedback on all inputs, looking for any mismatches to L_0 .

The Π_1^0 -hardness is proved by reduction from the emptiness problem. Given a grammar G, one can construct a new grammar G_1 that describes the *symmetric difference* of L(G) and L_0 ; the corresponding construction shall be explained later in Lemma 8.3. Then, $L(G_1) = L_0$ if and only if $L(G) = \emptyset$, which completes the reduction.

The more general problem of checking the equivalence of two given grammars is also Π_1^0 -complete (its complement is still recursively enumerable, through the same method of simulating automata on all inputs).

Corollary 7.3. The equivalence problem for linear grammars with left contexts is Π_1^0 -complete.

The second slightly more difficult undecidability result asserts that testing the finiteness of a language generated by a given grammar is complete for the second level of the arithmetical hierarchy.

Theorem 7.4. The finiteness problem for linear grammars with left contexts over a one-symbol alphabet is Σ_2^0 -complete. It remains Σ_2^0 -complete for any alphabet.

Proof (a sketch). Reduction from the finiteness problem for a Turing machine, which is Σ_2^0 -complete (see Rogers [21], Sect. 14.8). Given a Turing machine T, construct a rotating Turing machine T', which simulates T running on all inputs, with each simulation using a segment of the tape. Initially, T' sets up to simulate T running on ε , and then it regularly begins new simulations. Every time one of the simulated instances of T accepts, the constructed machine "flickers" by entering an accepting state in the end of one of its sweeps. Construct a trellis automaton with feedback M corresponding to this machine. Then L(M) is finite if and only if L(T) is finite.

8. Closure properties

A few additional results on the expressive power of linear grammars with contexts concern their closure under several operations.

The first result is the closure under concatenating a linear conjunctive language from the right, which is interesting because the family of linear conjunctive languages is itself not closed under concatenation [23].

Lemma 8.1. Let $K \subseteq \Sigma^*$ be defined by a linear grammar with contexts, and let $L \subseteq \Sigma^*$ be a linear conjunctive language. Then the language $K \cdot L$ can be defined by a linear grammar with contexts.

Proof. Let $G_1 = (\Sigma, N_1, R_1, S_1)$ and $G_2 = (\Sigma, N_2, R_2, S_2)$ be the grammars generating the languages K and L, respectively. Construct a linear conjunctive grammar with contexts $G = (\Sigma, N_1 \cup N_2 \cup \{S\}, R_1 \cup R_2 \cup R, S)$, where R contains the following rules.

$$S \to aS$$
 (for all $a \in \Sigma$)
 $S \to S_2 \& \triangleleft S_1$

The latter rule represents the concatenation of a string u from K with a string v from L by expressing v written in the context u, that is, $u\langle v\rangle$. The goal is to describe $\varepsilon\langle uv\rangle$, which is done by applying the rules of the form $S\to aS$ for every symbol of u.

This, in particular, implies that the language

$$L = \{ a^{i_1} b^{j_1} \dots a^{i_m} b^{j_m} \mid m \geqslant 2, \ i_t, j_t \geqslant 1, \ \exists \ell : i_1 = j_\ell \land i_{\ell+1} = j_m \},\$$

used by Terrier [23] to show that linear conjunctive languages are not closed under concatenation, can be defined by a linear grammar with contexts.

By the same method as in Lemma 8.1, one can show that the Kleene star of any linear conjunctive language can be represented by a linear grammar with contexts.

Lemma 8.2. Let L be a linear conjunctive language. Then the language L^* can be defined by a linear grammar with contexts.

Proof. Let $G = (\Sigma, N, R, S)$ be a linear conjunctive grammar that defines L. Construct a linear grammar with contexts $G' = (\Sigma, N \cup \{S', \widetilde{S}\}, R \cup R', S')$, with the following rules in R'.

$$S' \to \widetilde{S} \& \lhd \varepsilon \mid \varepsilon \& \lhd \varepsilon$$
$$\widetilde{S} \to S \& \lhd S'$$
$$\widetilde{S} \to a\widetilde{S} \qquad \text{(for all } a \in \Sigma)$$

Then $L(G') = L(G)^*$.

In this grammar, the nonterminal S' defines all strings of the form $\varepsilon\langle u_1 \dots u_k \rangle$, with $k \geq 0$ and $u_1, \dots, u_k \in L(G)$. For k = 0, the empty string is generated by the rule $S \to \varepsilon \& \lhd \varepsilon$. Every next string $\varepsilon\langle u_1 \dots u_k u_{k+1} \rangle$ is represented by concatenating the previous string $\varepsilon\langle u_1 \dots u_k \rangle$ from S' to a string u_{k+1} given by S' in the way similar to the previous lemma.

Similarly to the case of linear conjunctive languages ([16], Th. 7), the languages defined by linear grammars with contexts are closed under concatenation and star over disjoint alphabets, through a center marker, etc.

Turning to Boolean operations on languages, the closure under union and under intersection is obvious, as these operations can be expressed in grammars. In spite of having no negation operator, the language family defined by linear grammars with contexts is closed under all Boolean operations, just like the linear conjunctive languages.

Lemma 8.3. If the languages K, L are defined by linear grammars with left contexts, then so are the languages $K \cup L$, $K \cap L$ and \overline{L} .

For complementation, one can define a direct grammar-to-grammar construction, as for linear conjunctive grammars [16]. However, an easier approach is to construct a trellis automaton with feedback recognizing the given language, and then invert its set of accepting states.

Another standard operation on formal languages is the *quotient*: for two languages $K, L \subseteq \Sigma^*$, their left quotient is $K^{-1} \cdot L = \{v \mid \exists u \in K : uv \in L\}$ and their right quotient is $L \cdot K^{-1} = \{u \mid \exists v \in K : uv \in L\}$. Already for linear conjunctive languages, it is known that every recursively enumerable set is representable as a quotient of a linear conjunctive language and a regular language ([16], Th. 11). Therefore, the languages defined by linear grammars with contexts are also not closed under this operation. However, quotient with a finite language preserves this family. The construction is slightly different for quotient on the left and on the right, because context operators act only on the left.

Lemma 8.4. Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $K \subset \Sigma^*$ be a finite language. Then the language $L(G) \cdot K^{-1}$ can be defined by a linear grammar with contexts.

Proof (a sketch). Whenever the language K consists of multiple strings, the desired quotient can be represented as a union of quotients with singletons.

$$LK^{-1} = \bigcup_{u \in K} L\{u\}^{-1}$$

For a finite K, this is finite union. Furthermore, the quotient with a string $u = a_1 \dots a_\ell$ is representable as a sequence of ℓ quotients with one-symbol strings. Then, since the family of languages defined by linear grammars with contexts is closed under union, it is sufficient to prove the closure in the case of K containing a single one-symbol string: $K = \{d\}$, with $d \in \Sigma$.

Assume, without loss of generality, that G is in the linear normal form, provided by Theorem 3.1. Construct a grammar $G' = (\Sigma, N \cup N', R', S')$, where $N' = \{A' \mid A \in N\}$. The intention is to have $L_{G'}(A) = L_{G}(A)$ and $L_{G'}(A') = \{u\langle v \rangle \mid u\langle vd \rangle \in L_{G}(A)\}$ for all $A \in N$. The new rules in R' are defined as follows.

• For each rule of the form $A \to bB_1 \& \dots \& bB_\ell \& C_1c \& \dots \& C_kc$, the set R' contains the same rule for A.

$$A \rightarrow bB_1 \& \dots \& bB_\ell \& C_1 c \& \dots \& C_k c$$

If all symbols concatenated from the right are equal to d (that is, if c = d or if k = 0), then there is also a truncated rule for A'.

$$A' \to bB'_1 \& \dots \& bB'_\ell \& C_1 \& \dots \& C_k$$
 (if $c = d$)

• Every rule $A \to a \& \lhd D_1 \& \ldots \& \lhd D_m$ is included in R' as it is, and if a = d, a truncated rule for A' is added.

$$A \to a \& \lhd D_1 \& \dots \& \lhd D_m$$

 $A' \to \varepsilon \& \lhd D_1 \& \dots \& \lhd D_m$ (if $a = d$)

The facts that $\vdash_{G'} A(u\langle v \rangle)$ if and only if $\vdash_G A(u\langle v \rangle)$ and $\vdash_{G'} A'(u\langle v \rangle)$ if and only if $\vdash_G A(u\langle v d \rangle)$ can be proved by an easy induction. Hence, $L(G') = L(G) \cdot \{d\}^{-1}$. \square

The construction for the closure with a symbol on the left is similar but not symmetric.

Lemma 8.5. Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $K \subset \Sigma$ be a finite language. Then the language $K^{-1} \cdot L(G)$ can be defined by a linear grammar with contexts.

Proof (a sketch). Similarly to the previous lemma, the quotient from the left can be represented as a union of quotients with singletons.

Let G be in the linear normal form. Construct a grammar $G' = (\Sigma, N \cup N', R', S')$, where $N' = \{A' \mid A \in N\}$, such that $L_{G'}(A) = L_G(A)$ and $L_{G'}(A') = \{u\langle v \rangle \mid du\langle v \rangle \in L_G(A)\}$ for all $A \in N$, as follows.

• For each rule of the form $A \to bB_1 \& \dots \& bB_\ell \& C_1 c \& \dots \& C_k c$, the set R' contains this rule, and, if b = d or $\ell = 0$, then a truncated rule for A' is also added to R'.

$$A \to bB_1 \& \dots \& bB_\ell \& C_1 c \& \dots \& C_k c$$

$$A' \to B_1 \& \dots \& B \& C'_1 c \& \dots \& C'_k c$$
 (if $b = d$)

• For a rule $A \to a \& \triangleleft D_1 \& \dots \& \triangleleft D_m$, the set R' contains the following rule.

$$A' \to a \& \triangleleft D'_1 \& \dots \& \triangleleft D'_m$$

If additionally a = d and m = 0, then an extra rule

$$A' \to \varepsilon$$

is added to R'.

Similarly to the previous lemma, one can prove that $\vdash_{G'} A(u\langle v \rangle)$ if and only if $\vdash_G A(u\langle v \rangle)$ and $\vdash_{G'} A'(u\langle v \rangle)$ if and only if $\vdash_G A(u\langle v d \rangle)$.

The last class of operations to be considered are homomorphisms. Given two alphabets Σ and Ω , a homomorphism $h \colon \Sigma^* \to \Omega^*$ is a mapping that satisfies $h(\varepsilon) = \varepsilon$ and h(uv) = h(u)h(v) for all $u, v \in \Sigma$; it is completely defined by the images of one-symbol strings. A homomorphism is a code, if h(u) = h(v) implies u = v. It is known that the family of linear conjunctive languages is closed under a homomorphism h if and only if either h is a code, or h trivially maps everything to the empty string [18]. The closure under codes also holds for linear grammars with contexts.

Lemma 8.6. Let $G = (\Sigma, N, R, S)$ be a linear grammar with contexts, and let $h \colon \Sigma^* \to \Omega^*$ be a code. Then the language h(L(G)) is defined by a linear grammar with contexts.

Proof. Construct a new linear grammar with contexts $G' = (\Omega, N, R', S)$ with the following set of rules. Consider any rule in R.

Then the new grammar contains a rule with all strings encoded by h.

$$A \to h(x_1)B_1h(y_1) \& \dots \& h(x_k)B_kh(y_k) \&$$

 $\lhd h(x_1')D_1h(y_1') \& \dots \& \lhd h(x_m')D_mh(y_m') \&$
 $\unlhd h(x_1'')E_1h(y_1'') \& \dots \& \unlhd h(x_n'')E_nh(y_n'')$

It is claimed that every symbol $A \in N$ generates the language $L_{G'}(A) = h(L_G(A))$. In one direction, it has to be shown that for every string $u\langle v \rangle$ generated by A in G, the encoding $h(u)\langle h(v)\rangle$ is in $L_{G'}(A)$. This is actually true for every homomorphism h. In the other direction, if an item $A(x\langle y\rangle)$ can be deduced in the grammar G', then x and y are images of uniquely determined strings $u, v \in \Sigma^*$, with h(u) = x and h(v) = y, and one can prove that the string $u\langle v\rangle$ is in $L_G(A)$. Both proofs follow by a straightforward induction on the number of steps in the derivations.

As compared to the well-researched closure properties of other families of formal grammars ([19], Sect. 8.2), the results for grammars with contexts, whether linear or with general concatenation, are still quite fragmentary. It is conjectured that they are not closed under most of the basic operations, such as concatenation and star, but this cannot yet be proved due to the lack of negative proof methods for these grammars.

9. Future work

A suggested topic for future research is to investigate the main ideas in the literature on trellis automata [7–10, 23, 25] and see whether they can be extended to trellis automata with feedback, and hence to linear grammars with contexts. In particular, it is essential to learn how to prove that some languages cannot be recognized by any automaton of this kind.

It would also be interesting to see how the parsing algorithms for grammars with contexts, conjunctive grammars and related models [1, 2, 19] could be adapted to linear grammars with contexts.

Acknowledgements. The authors are grateful to an anonymous reviewer for a good advise to replace an elaborate simulation of a Turing machine given in the preliminary version of this paper [5] with a straightforward construction now given in Section 6.

References

- T. Aizikowitz and M. Kaminski, LR(0) conjunctive grammars and deterministic synchronized alternating pushdown automata. Comput. Sci. Russia (CSR 2011, St. Petersburg, Russia, 14–18 June 2011). Vol. 6651 of Lect. Notes Comput. Sci. (2011) 345–358.
- [2] M. Barash, Recursive descent parsing for grammars with contexts. SOFSEM 2013 student research forum Špindlerův Mlýn, Czech Republic, 26-31 January, 2013. Local Proceedings II, 10-21, Institute of Computer Science AS CR (2013).
- [3] M. Barash, Programming language specification by a grammar with contexts. In Fifth Workshop on Non-Classical Models of Automata and Applications, NCMA 2013, Umeå, Sweden, 13–14 August, 2013, edited by S. Bensch, F. Drewes, R. Freund, F. Otto. books@ocg.at 294, Österreichische Computer Gesellschaft (2013) 51–67. http://users.utu.fi/mikbar/kieli/
- [4] M. Barash and A. Okhotin, An extension of context-free grammars with one-sided context specifications. *Inform. Comput.* 237 (2014) 268–293.
- [5] M. Barash and A. Okhotin, Linear grammars with one-sided contexts and their automaton representation. LATIN 2014: Theoretical Informatics, Montevideo, Uruguay, 31 March-4 April 2014. In vol. 8392, Lect. Notes Comput. Sci. (2014) 190-201.

- [6] A. Birman and J.D. Ullman, Parsing algorithms with backtrack. Inform. Control 23 (1973) 1–34.
- [7] K. Čulík II, J. Gruska and A. Salomaa, Systolic trellis automata I. Int. J. Comput. Math. 15 (1984) 195–212.
- [8] K. Čulík II, J. Gruska and A. Salomaa, Systolic trellis automata II. Int. J. Comput. Math. 16 (1984) 3–22.
- [9] C. Dyer, One-way bounded cellular automata. Inform. Control 44 (1980) 261–281.
- [10] O.H. Ibarra and S.M. Kim, Characterizations and computational complexity of systolic trellis automata. Theoret. Comput. Sci. 29 (1984) 123–153.
- [11] O.H. Ibarra, S.M. Kim and S. Moran, Sequential machine characterizations of trellis and cellular automata and applications. SIAM J. Comput. 14 (1985) 426–447.
- [12] A. Jeż, Conjunctive grammars can generate non-regular unary languages. Int. J. Found. Comput. Sci. 19 (2008) 597–615.
- [13] A. Jeż and A. Okhotin, Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. Theory Comput. Syst. 46 (2010) 27–58.
- [14] R. Kowalski, Logic for Problem Solving. North-Holland, Amsterdam (1979).
- [15] A. Okhotin, Conjunctive grammars. J. Autom. Lang. Comb. 6 (2001) 519-535.
- [16] A. Okhotin, On the closure properties of linear conjunctive languages. Theoret. Comput. Sci. 299 (2003) 663–685.
- [17] A. Okhotin, On the equivalence of linear conjunctive grammars to trellis automata. RAIRO: ITA 38 (2004) 69–88.
- [18] A. Okhotin, Homomorphisms preserving linear conjunctive languages. J. Autom. Lang. Comb. 13 (2008) 299–305.
- [19] A. Okhotin, Conjunctive and Boolean grammars: the true general case of the context-free grammars. Comput. Sci. Rev. 9 (2013) 27–59.
- [20] A. Okhotin, Improved normal form for grammars with one-sided contexts. Theoret. Comput. Sci. 588 (2015) 52–72.
- [21] H. Rogers, Jr., Theory of Recursive Functions and Effective Computability. McGraw-Hill (1967).
- [22] W.C. Rounds, LFP: A logic for linguistic descriptions and an analysis of its complexity. Comput. Linguistics 14 (1988) 1–9.
- [23] V. Terrier, On real-time one-way cellular array. Theoret. Comput. Sci. 141 (1995) 331–335.
- [24] I. Törmä. Personal communication. February 2013.
- [25] S. Yu, A property of real-time trellis automata. Discrete Appl. Math. 15 (1986) 117–119.

Communicated by Ch. Choffrut.

Received December 5, 2014. Accepted June 11, 2015.