Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 320 (2004) 419–448

Theoretical Computer Science

www.elsevier.com/locate/tcs

# On the number of nonterminals in linear conjunctive grammars☆

Alexander Okhotin

*School of Computing, Queen's University, Kingston, Ont., Canada K7L3N6*

Received 7 September 2003; received in revised form 25 February 2004; accepted 3 March 2004
Communicated by D. Perrin

**Abstract**

The number of nonterminals in a linear conjunctive grammar is considered as a descriptional complexity measure of this family of languages. It is proved that a hierarchy collapses, and for every linear conjunctive grammar there exists and can be effectively constructed a linear conjunctive grammar that accepts the same language and contains exactly two nonterminals. This yields a partition of linear conjunctive languages into two nonempty disjoint classes of those with nonterminal complexity 1 and 2. The basic properties of the family of languages for which one nonterminal suffices are established. Nonterminal complexity of grammars in the linear normal form is also investigated.
© 2004 Elsevier B.V. All rights reserved.

*Keywords:* Formal languages; Conjunctive grammar; Trellis automaton; Cellular automaton; Language equation; Descriptional complexity; Minimal linear grammar

## 1. Introduction

Linear conjunctive grammars, which are linear context-free grammars augmented with an explicit intersection operation, were introduced several years ago [16,18], and soon thereafter were shown to generate a language family that has been studied long before [23]. It is now known that the following formalisms define the same family of languages: one-way real-time cellular automata [6,24], trellis automata [2,4,5], a certain very restricted type of Turing machines [13,14], linear conjunctive grammars

☆ A preliminary version of this paper was presented at the Descriptional Complexity of Formal Systems (DCFS 2003) workshop held in Budapest, Hungary, July 12–14, 2003, and its extended abstract appeared in the proceedings.

*E-mail address:* okhotin@cs.queensu.ca (A. Okhotin).

[16,23], language equations with union, intersection and linear concatenation [17] and the recently introduced linear Boolean grammars [21]. The abundancy of natural characterizations coming from different areas of theoretical computer science clearly justifies the importance of this language family and gives a reason for the continued study of its properties.

Let us summarize the most important known facts about this family. It contains some usual examples of non-context-free languages, such as $\{a^nb^nc^n \mid n \geqslant 0\}$ [4,6,18] and $\{wcw \mid w \in \{a,b\}^*\}$ [4,16], as well as the language of all computations of any Turing machine [15,16,18] and the von Dyck language [6,23], which is not linear context-free. It contains some **P**-complete languages [13,20]. Every language it contains can be recognized in time $O(n^2)$ and space $O(n)$ [4,16]. It is known to be closed under all set-theoretic operations [4,6,18], not closed under concatenation [25] and not closed under star [23]. It is incomparable with the context-free languages [25] and is properly contained in the conjunctive languages [23].

The descriptional complexity of one-way real-time cellular automata was first investigated in [15], where the tradeoffs between them and several devices of different recognition power were proved to be recursively unbounded using the method of [12]. The succinctness of description of these languages by linear conjunctive grammars was first studied in [22], where the transition from them to trellis automata was shown to cause superpolynomial blowup in the total length of description. It was also proved in [22] that for every $n$ there exist languages that can be denoted with an $n$-state but not an $(n-1)$-state trellis automaton, thus establishing a strict infinite hierarchy of $n$-state languages.

This paper considers another descriptional complexity measure for this family of languages—the number of nonterminals in a linear conjunctive grammar. Somewhat surprisingly, the more or less expected infinite hierarchy of $n$-nonterminal languages actually collapses, as it turns out that *two nonterminals always suffice*. This contrasts the known infinite hierarchy of $n$-nonterminal context-free and linear context-free languages [9] and can be compared with the results on the representability of all recursively enumerable languages in several different grammar formalisms using a fixed number of nonterminals [7].

The proof of this main result, given in Section 3, is constructive: given a trellis automaton, one can represent its entire operation in the form of a certain language and denote this language using a single nonterminal, employing the other one as a start symbol that "analyzes" the encoded computation by referring to the former nonterminal and generates exactly the strings accepted by the automaton.

This upper bound of 2 nonterminals is strict: as demonstrated in Section 4, two nonterminals are required even for some regular languages. At the same time, for every linear conjunctive language a certain closely related language can be denoted using a single nonterminal; among these are some **P**-complete languages. The basic properties of one-nonterminal linear conjunctive grammars are developed in Section 4. The class of languages they generate admits another noteworthy characterization, as the set of languages defined by one-variable language equations with union, intersection and linear concatenation, and thus the results on one-nonterminal languages have important implications on the theory of language equations.

In the final Section 5 the nonterminal complexity of linear conjunctive grammars in the linear normal form [16] is investigated and compared with the number of states in trellis automata.

## 2. Grammars, equations and automata

### 2.1. Linear conjunctive grammars

Conjunctive grammars, introduced in [16], are an extension of context-free grammars with an explicit intersection operation.

**Definition 1.** A conjunctive grammar is a quadruple $G=(\Sigma,N,P,S)$, in which $\Sigma$ and $N$ are disjoint finite nonempty sets of terminal and nonterminal symbols respectively; $P$ is a finite set of grammar rules, each of the form

$$A \to \alpha_1 \& \cdots \& \alpha_n \quad (A \in N,\ n \geqslant 1,\ \alpha_i \in (\Sigma \cup N)^* \text{ for all } i), \tag{1}$$

where the strings $\alpha_i$ are distinct and their order is considered insignificant; $S \in N$ is a nonterminal designated as the start symbol. For a rule (1), an object of the form $A \to \alpha_i$ is called a conjunct; denote the set of all conjuncts as *conjuncts*$(P)$.

A conjunctive grammar generates strings by deriving them from the start symbol, generally in the same way as context-free grammars do. Intermediate strings used in course of a derivation are defined as follows:

**Definition 2.** Let $G=(\Sigma,N,P,S)$ be a conjunctive grammar. The set of conjunctive formulae $\mathscr{F} \subset (\Sigma \cup N \cup \{\text{“(”}, \text{“\&”}, \text{“)”}\})^*$ is defined inductively:
- The empty string $\varepsilon$ is a conjunctive formula.
- Any symbol from $\Sigma \cup N$ is a formula.
- If $\mathscr{A}$ and $\mathscr{B}$ are nonempty formulae, then $\mathscr{A}\mathscr{B}$ is a formula.
- If $\mathscr{A}_1, \ldots, \mathscr{A}_n$ $(n \geqslant 1)$ are formulae, then $(\mathscr{A}_1 \& \cdots \& \mathscr{A}_n)$ is a formula.

Next, define the notion of derivability in one step as a binary relation $\overset{G}{\Longrightarrow}$ on the set $\mathscr{F}$. There are two types of derivation steps:

(1) A nonterminal can be rewritten with a body of a rule enclosed in parentheses:

$$s'As'' \overset{G}{\Longrightarrow} s'(\alpha_1 \& \cdots \& \alpha_n)s'' \tag{2}$$

if $A \to \alpha_1 \& \cdots \& \alpha_n \in P$ and $s'As'' \in \mathscr{F}$, where $s', s'' \in (\Sigma \cup N \cup \{\text{“(”}, \text{“\&”}, \text{“)”}\})^*$.

(2) A conjunction of one or more identical terminal strings enclosed in parentheses can be replaced with one such string without the parentheses:

$$s'(w \& \cdots \& w)s'' \overset{G}{\Longrightarrow} s'ws'' \tag{3}$$

if $w \in \Sigma^*$ and $s'ws'' \in \mathscr{F}$.

**Definition 3.** Let $G=(\Sigma,N,P,S)$ be a conjunctive grammar. The language of a formula is the set of all terminal strings derivable from the formula in zero or more steps: $L_G(\mathscr{A})=\{w\in\Sigma^*\mid \mathscr{A}\overset{G}{\Longrightarrow}{}^* w\}$. Define $L(G)=L_G(S)$.

Let us now restrict conjunctive grammars of the general form to obtain the subclass of *linear conjunctive grammars*:

**Definition 4.** A conjunctive grammar $G=(\Sigma,N,P,S)$ is said to be linear, if each rule in $P$ is of the form

$$A\to u_1B_1v_1\&\cdots\&u_mB_mv_m \quad (m\geqslant 1,\ u_i,v_i\in\Sigma^*,\ B_i\in N) \quad \text{or} \tag{4a}$$

$$A\to w \quad (w\in\Sigma^*) \tag{4b}$$

It is said to be in the linear normal form, if the rules of the type (4a) are restricted to be of the form $A\to bB_1\&\cdots\&bB_m\&C_1c\&\cdots\&C_nc$, where $b,c\in\Sigma$, $m+n\geqslant 1$ and $B_i\in N$.

## 2.2. Language equations with union, intersection and linear concatenation

Context-free grammars are known to have an algebraic representation by least solutions of systems of language equations with union and concatenation [1,3]. Conjunctive grammars possess a similar characterization by language equations with union, intersection and concatenation [17].

Let us retell the main results of [17], simplifying them for the subcase of linear conjunctive grammars studied in this paper.

**Definition 5** (System of equations). Let $\Sigma$ be an alphabet. Let $n\geqslant 1$. Let $X=(X_1,\ldots,X_n)$ be a set of language variables, which assume values of languages over $\Sigma$. Let $\varphi_1,\ldots,\varphi_n$ be expressions that depend upon the variables $X$ and may contain these variables, the constant languages $\{\varepsilon\}$ and $\{a\}$ (for all $a\in\Sigma$), set-theoretic union and intersection, as well as linear concatenation (i.e., left- and right-concatenation of $\{a\}$). Then

$$\begin{aligned} X_1 &= \varphi_1(X_1,\ldots,X_n)\\ &\ \vdots\\ X_n &= \varphi_n(X_1,\ldots,X_n)\end{aligned} \tag{5}$$

is called a resolved system of equations over $\Sigma$ in variables $X$. A vector of languages $L=(L_1,\ldots,L_n)$ is a solution of (5) if for every $i$ the value of $\varphi_i$ under the assignment $X_j=L_j$ (for all $j$) is $L_i$.

Note that language equations can be defined in a more rigorous way by first defining a formula as a syntactical concept and then supplying semantics for it by defining its value on a vector of languages [17,21].

Denote the right-hand side of a system (5) as a vector function:

$$\varphi(X_1,\dots,X_n)=(\varphi_1(X_1,\dots,X_n),\dots,\varphi_n(X_1,\dots,X_n)) \tag{6}$$

and inductively define its substitutions into itself as

$$\varphi^0(X_1,\dots,X_n)=(X_1,\dots,X_n) \tag{7a}$$

and

$$\varphi^{i+1}(X_1,\dots,X_n)=\varphi(\varphi^i(X_1,\dots,X_n)). \tag{7b}$$

Define a partial order "$\preccurlyeq$" on the set of language vectors of length $n$ as componentwise inclusion: $(L_1',\dots,L_n')\preccurlyeq(L_1'',\dots,L_n'')$ if and only if $L_i'\subseteq L_i''$ for all $i$ $(1\leqslant i\leqslant n)$. Then one can prove that the operator $\varphi$ on the set $(2^{\Sigma^*})^n$ is monotone and $\cup$-continuous with respect to this partial order [17], which, by the fixed point theory, yields the following result:

**Theorem 1** (Okhotin [17]). *Every system (5) over an alphabet $\Sigma$ and in variables $X_1,\dots,X_n$ has least solution (with respect* to *"$\preccurlyeq$") given by*

$$L=(L_1,\dots,L_n)=\sup_{i\geqslant 0}\varphi^i\left(\underbrace{\emptyset,\dots,\emptyset}_{n}\right). \tag{8}$$

**Theorem 2** (Okhotin [17]). *A language $L$ is generated by a linear conjunctive grammar if and only there exists a system of equations (5) with union, intersection and linear concatenation, such that $L$ is the first component of its least solution.*

Another important issue treated in [17] is the uniqueness of solution. For language equations with unrestricted concatenation it is convenient to use the notion of a *strict system*, developed in [17] by generalizing the corresponding notion for the context-free case [1], as a sufficient condition for uniqueness. When concatenation is linear, this condition degrades to quite a simple one, which will now be stated:

**Definition 6** (Okhotin [17]). An expression (in the sense of Definition 5) $\varphi$ is said to be admissible in strict systems if it is $\{\varepsilon\}$, $\{a\}$, union or intersection of two expressions admissible in strict systems, or a left- or right-concatenation of a terminal symbol to an arbitrary expression as in Definition 5.

A system (5) of language equations with union, intersection and linear concatenation is said to be strict if every expression $\varphi_i$ on its right-hand side is admissible in strict systems.

Basically, the only restriction imposed on strict systems is that variables can appear in expressions only when concatenated to terminal symbols. This dismisses pathological cases of language equations like $X=X$ and guarantees the following property:

**Theorem 3** (Okhotin [17]). *Every strict system has unique solution.*

Now let us strengthen the result of Theorem 2 on the equivalence of linear conjunctive grammars and language equations as in Definition 5. In this stronger formulation it will be very useful in the following:

**Theorem 4.** *Let $n \geqslant 1$. For any vector of languages $(L_1, \ldots, L_n)$ over $\Sigma$, the following two conditions are equivalent*:

- *There exists a linear conjunctive grammar $G = (\Sigma, \{A_1, \ldots, A_n\}, P, A_1)$, such that $L_G(A_i) = L_i$ for all $i$.*
- *There exists a system of language equations with union, intersection and linear concatenation over $\Sigma$ in variables $(X_1, \ldots, X_n)$, such that its unique solution is $(L_1, \ldots, L_n)$.*

**Proof.** The conversion of an $n$-nonterminal grammar to a system of $n$ equations can be done as follows: first, unit conjuncts are removed from the grammar using the method of [16], i.e., all rules of the form $A \to B \& \alpha_2 \& \cdots \& \alpha_n$ are eliminated; this does not affect the number of nonterminals and the languages they generate. Then every nonterminal $A_i$ is represented as a variable $X_i$. If there are some rules for $A_i$, then the equation for $X_i$ is

$$X_i = \bigcup_{A_i \to \alpha_1 \& \cdots \& \alpha_n \in P} \bigcap_{j=1}^{n} \alpha_j, \tag{9a}$$

where all instances of nonterminals in $\alpha_j$ are replaced with the corresponding variables. If there are no rules for $X_i$, then the equation may be taken as, say,

$$X_i = \varepsilon \& a \quad (\text{for some } a \in \Sigma). \tag{9b}$$

This system has least solution $(L_G(A_1), \ldots, L_G(A_n))$ by the results of [17]. On the other hand, it is a strict system, so the mentioned solution is unique by Theorem 3.

Going from language equations back to linear conjunctive grammars is somewhat complicated by the fact that the expressions in the right-hand sides of equations might be arbitrarily complex, while the rules in linear conjunctive grammars must be of the specific form (4). This can be trivially solved by introducing new variables and moving subexpressions to separate equations, as in Theorem 2, but this would contradict our goal of having as many nonterminals as there were variables.

In order to meet this goal, let us start from equivalent transformations of right-hand sides. Two expressions over union, intersection and linear concatenation that depend upon language variables $(X_1, \ldots, X_n)$ and contain constant languages $\{a\}$ $(a \in \Sigma)$ and $\{\varepsilon\}$ are said to be *equivalent*, denoted $\varphi \equiv \psi$, if they evaluate to the same language for every substitution of languages for its variables. The following equivalencies can easily be proved to hold for all expressions $\varphi$, $\psi$ and $\xi$:

$$\varphi \cap (\psi \cup \xi) \equiv (\varphi \cap \psi) \cup (\varphi \cap \xi), \tag{10a}$$

$$(\psi \cup \xi) \cap \varphi \equiv (\psi \cap \varphi) \cup (\xi \cap \varphi), \tag{10b}$$

$$a(\varphi \cup \psi) \equiv a\varphi \cup a\psi, \tag{10c}$$

$$a(\varphi \cap \psi) \equiv a\varphi \cap a\psi\,, \tag{10d}$$

$$(\varphi \cup \psi) \cdot a \equiv \varphi a \cup \psi\, a, \tag{10e}$$

$$(\varphi \cap \psi) \cdot a \equiv \varphi a \cap \psi\, a. \tag{10f}$$

Now, given any expression $\varphi$ and using these equivalencies as rewriting rules that operate from left to right, one can move linear concatenation to the bottom of the formula, raise union to the top level and let intersection remain in between.

This can be done with the right-hand sides of a given system $X = \varphi(X)$, resulting in a system $X = \varphi'(X)$ with the same unique solution, in which every equation is of the form

$$X_i = \bigcup_{j=1}^{m_i} \bigcap_{k=1}^{l_{ij}} u_{ijk} X_{ijk} v_{ijk}. \tag{11}$$

This system of equations can be directly simulated by a grammar

$$A_i \to u_{ij1} X_{ij1} v_{ij1} \& \cdots \& u_{ijl_{ij}} X_{ijl_{ij}} v_{ijl_{ij}} \quad (1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m_i) \tag{12}$$

which satisfies the first condition of the theorem. □

It should be noted that the claim of Theorem 4 is specific to linear conjunctive grammars, and likely does not hold for conjunctive grammars of the general form. Indeed, equalities (10d) and (10f) cannot be generalized for the general case (as concatenation is not distributive over intersection), and hence one would probably need to add extra nonterminals to represent concatenation of intersections. A statement similar to Theorem 2 [17] remains the strongest known statement on the equivalence of conjunctive grammars of the general form and language equations with intersection.

## 2.3. Trellis automata

(Systolic) trellis automata [4] were introduced in early 1980s as a model of a massively parallel system with simple identical processors connected in a uniform pattern. *Triangular* (*real-time*, *homogeneous*) *trellis automata* are a particular case of trellis automata, in which the connections between nodes form a figure of triangular shape, as shown in Fig. 1. These automata are used as acceptors of strings loaded from the bottom, and the acceptance is determined by the topmost element. In the following they will be referred to as just *trellis automata*.

Following the notation of [23], define a trellis automaton as a quintuple $M = (\Sigma, Q, I, \delta, F)$, where $\Sigma$ is the input alphabet, $Q$ is a finite nonempty set of states (of processing units), $I : \Sigma \to Q$ is a function that sets the initial states (loads values into the bottom processors), $\delta : Q \times Q \to Q$ is the transition function (the function computed by processors) and $F \subseteq Q$ is the set of final states (effective in the top processor). Given a string $a_1 \ldots a_n$ ($a_i \in \Sigma$, $n \geqslant 1$), every node corresponds to a certain substring $a_i \ldots a_j$ ($1 \leqslant i \leqslant j \leqslant n$) of symbols on which its value depends. The value of a bottom node corresponding to one symbol of the input is $I(a_i)$; the value of a successor of

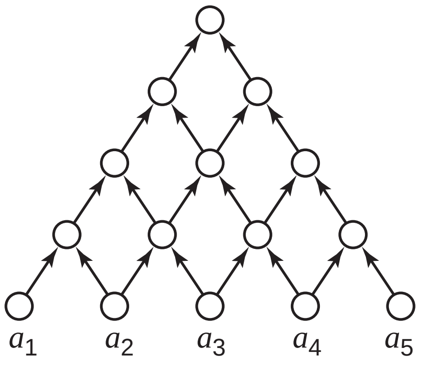


Fig. 1. Computation of a trellis automaton.

two nodes is $\delta$ of the values of these ancestors. Denote the value of a node corresponding to $a_i \dots a_j$ as $\Delta(I(a_i \dots a_j)) \in Q$: here $I(a_i \dots a_j)$ is a string of states (the bottom row of the trellis), while $\Delta$ denotes the result (a single state) of a triangular computation starting from a row of states. By definition, $\Delta(I(a_i)) = I(a_i)$ and $\Delta(I(a_i \dots a_j)) = \delta(\Delta(I(a_i \dots a_{j-1})), \Delta(I(a_{i+1} \dots a_j)))$. Now define $L(M) = \{w \mid \Delta(I(w)) \in F\}$.

The computational equivalence of trellis automata to linear conjunctive grammars was proved as follows:

**Theorem 5** (Okhotin [23]). *For every linear conjunctive grammar $G$ there exists and can be effectively constructed a trellis automaton $M$, such that $L(M) = L(G) \backslash \{\varepsilon\}$.*

Assuming without loss of generality that $G$ is in the linear normal form, the proof of Theorem 5 is based upon a subset construction and results in exponentially as many states as there are nonterminals in the normal form grammar $G$. To be more specific, if $G = (\Sigma, N, P, S)$, then the set of states of a constructed trellis automaton is $Q = \Sigma \times 2^N \times \Sigma$.

**Theorem 6** (Okhotin [23]). *For every trellis automaton $M$, there exists and can be effectively constructed a linear conjunctive grammar $G$, such that $L(G) = L(M)$.*

Theorem 6 is proved by a simple construction [23], where for each state $q$ of the automaton $(\Sigma, Q, I, \delta, F)$ one creates a nonterminal $A_q$, for each initial state $q = I(a)$ one adds a rule $A_q \to a$, for each transition $q = \delta(q_1, q_2)$ one adds $|\Sigma|^2$ rules

$$A_q \to bA_{q_2} \& A_{q_1}c \quad (\text{for all } b, c \in \Sigma) \tag{13}$$

and finally one introduces a start symbol $S$ with a rule $S \to A_q$ for every $q \in F$. The correctness of this construction is stated as follows:

**Lemma 1** (Okhotin [23]). *For every $w \in \Sigma^+$ and $q \in Q$, $A_q \overset{G}{\Longrightarrow}{}^* w$ iff $\Delta(I(w)) = q$.*

## 3. Two nonterminals suffice

In this section the main result of the paper is established, which is a construction of a linear conjunctive grammar with two nonterminals out of an arbitrary linear conjunctive grammar.

### 3.1. Encoding of a computation

The given grammar is first converted to an equivalent trellis automaton $M=(\Sigma, Q, I, \delta, F)$—for instance, using the known method [23],—and the rest of the construction is a particular kind of simulation of this automaton by another grammar.

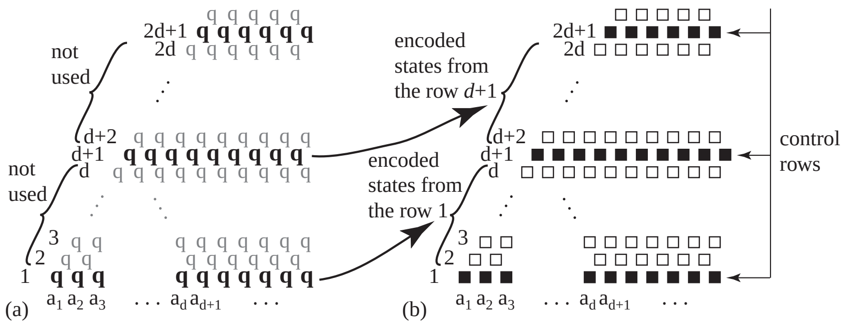
The straightforward method of simulating trellis automata by linear conjunctive grammars, explained in the previous section, represents every state with a nonterminal. As a result, for every string $w \in \Sigma^+$, the set $\{A_q \mid A_q \Longrightarrow^* w\}$ of nonterminals that derive $w$ directly corresponds to the state of one node $\Delta(I(w))$ in the trellis. If one aims to reflect the value of each and every element of the trellis in the form of a derivation of the corresponding string from some nonterminal, then it is easy to see that no less than $\lceil \log_2 |Q| \rceil$ nonterminals are required just to be able to store all this information.

However, storing every element is unnecessary, as one can simulate at a time larger portions of a computation than a single $\delta$ transition, and explicitly store only some selection of the computed states, such that the rest of the states could be inferred from those that are stored. The grammar that will be constructed records every $(6|Q|-1)$-th row of the computation and simulates the whole $\delta$ machinery of a trellis automaton using just *one* nonterminal instead of the set $\{A_q \mid q \in Q\}$ from [23]. This nonterminal can be viewed as a single bit corresponding to every substring of the input string, and the operation of the grammar, which involves deriving longer strings from the shorter ones, is essentially a computation of these bits using finite-range data access. The other nonterminal will be a start symbol that will decode the appropriate information from the first nonterminal and use it to accept the right strings.

Let $d=6|Q|-1$. Our one-bit simulation determines and uses the values in every $d$th row of the computation of the trellis automaton, starting from the first one; these rows are emphasized in Fig. 2(a). Every $d$th row of the simulation (i.e., the rows $1, 1+d, 1+2d, \ldots$) is called a *control row*, and its bits are always set to true (shown as black squares in Fig. 2(b)). The numbers of states forming each $(kd+1)$-th $(k \geqslant 0)$ row of the trellis are encoded as bits in the rows $kd+2, \ldots, kd+d$ of the one-bit simulation (white squares in Fig. 2(b), which can be true or false).

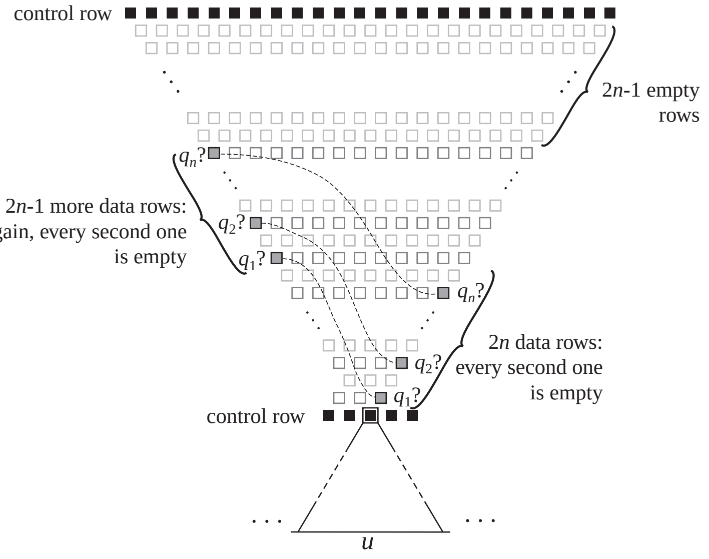
Let $u$ be a substring (of some longer input string), such that $|u| = 1 \pmod{d}$, and let us show how the state $q_i = \Delta(I(u))$ is encoded in the bit rows following the row number $|u|$ in the simulated computation. In Fig. 3, the central black square in the lower control row corresponds to the substring $u$. The number of the state $q_i$ is encoded in the bits shown in grey: both bits labeled $q_i$ in the figure are set to true, while the rest of the grey bits are set to false.

Reflecting the same state $q_i$ in two identical bits is a clear redundancy of the encoding. However, this redundancy is necessary, because the "data access" capabilities of a grammar simulating an automaton are fairly limited: the derivability of a substring



not used
not used
2d+1
2d
d+2
d+1
d
3
2
1
a1 a2 a3 . . . ad ad+1 . . .
(a)
encoded states from the row d+1
encoded states from the row 1
(b)
control rows

Fig. 2. (a) A computation and (b) its one-bit simulation.

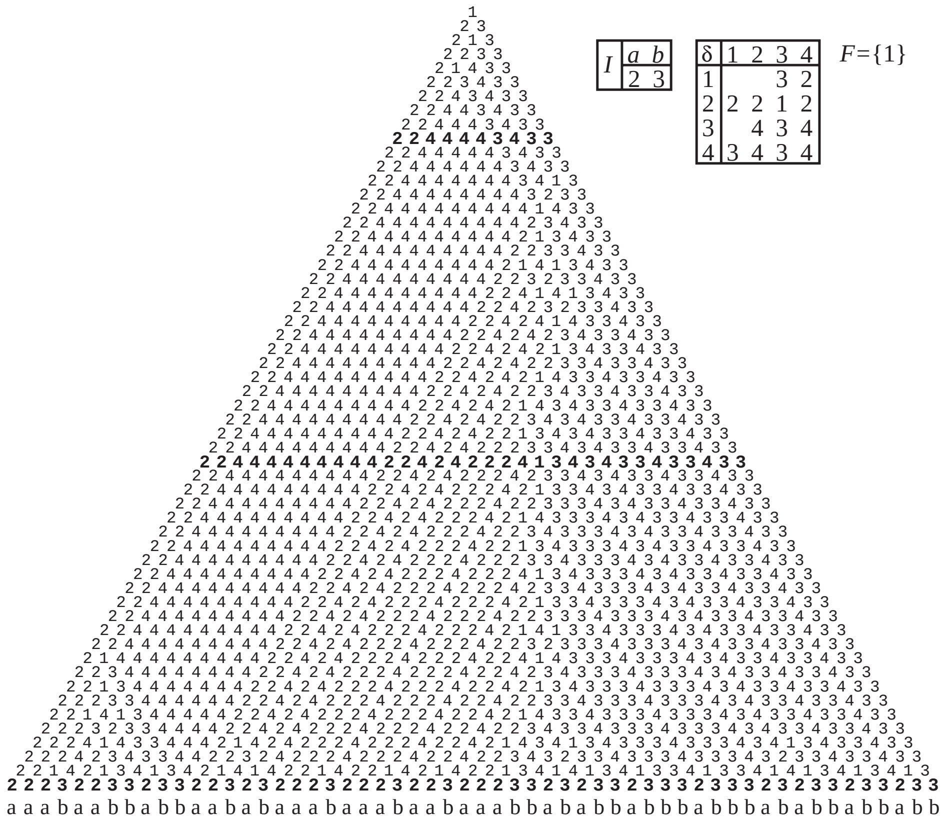


control row
2n-1 empty rows
2n-1 more data rows: again, every second one is empty
qn?
q2?
q1?
2n data rows: every second one is empty
control row
u

Fig. 3. Encoding of a state $q_i = \Delta(I(u))$, where $|u| = 1 \pmod{d}$.

from a nonterminal depends on this substring alone, and cannot be influenced by the context it appears. As will be shown below, most of the time only one of these two bits will be accessible.

This information is stored in $4n-1$ rows above the control row; every second row among these is intentionally left blank (i.e., set to false). The remaining $2n-1$ rows are also empty. As will be proved below, these empty rows ensure that one can always tell where the control rows are. Once the control rows of the simulated computation are located, it is possible to decode the numbers of those states from the original computation that are located in every $d$th row. Knowing the states in these selected rows, one can easily compute the states in any row located in between: indeed, in



| $I$ | $a$ | $b$ |
|---|---|---|
| | 2 | 3 |

| $\delta$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | 3 | 2 |
| 2 | 2 | 2 | 1 | 2 |
| 3 | | 4 | 3 | 4 |
| 4 | 3 | 4 | 3 | 4 |

$F = \{1\}$

Fig. 4. A computation of a trellis automaton for the von Dyck language.

order to compute one state in a row $kd + i$, it is sufficient to know $i$ adjacent states in the row $kd + 1$. This decoding process will be explained in more detail in a formal construction below, and for now let us illustrate the encoding on an example.

Consider a certain four-state trellis automaton for the Dyck language over $\Sigma = \{a, b\}$ [4]. This automaton and its computation on the 56-symbol-long string $w = aaabaabbabb$ $aababaaabaaabaabaaabbababbabbbabbbababbabbabb$ are given in Fig. 4. The number $d$ equals $6|Q| - 1 = 6 \cdot 4 - 1 = 23$, and thus every 23rd row of this computation will be stored in its one-bit simulation; in the figure these rows (1, 24, 47) are emphasized with a boldface font.

The next Fig. 5 contains the one-bit simulation of this computation. The top left corner of the figure reminds how every state is encoded, and indeed one can easily decipher the states emphasized in Fig. 4 by a careful examination of the bits in Fig. 5. For instance, the row number 24 of the original computation contains one instance of state “1”; it is easy to find the encoding of this state by looking right above the middle control row in the simulated computation.

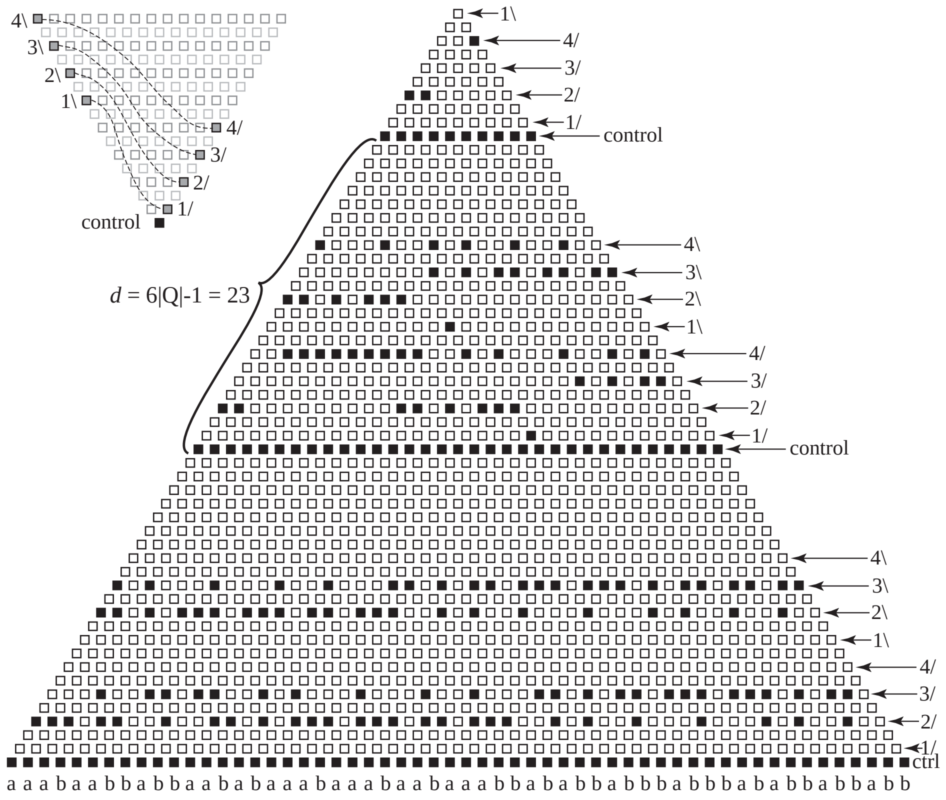

4\
3\
2\
1\
4/
3/
2/
1/
control
d = 6|Q|-1 = 23
ctrl
a a a b a a b b a b b a a b a b a a a b a a a b a a b a a a b b a b a b b a b b b a b b b a b a b b a b b a b b

Fig. 5. One-bit simulation of the computation from Fig. 4.

## 3.2. A language equation for the encoded computation

Now let us formalize this one-bit simulation as a *language* of the strings that have their corresponding bit set to true. Fix a trellis automaton $M$ with the set of states $Q=\{q_1,\dots,q_n\}$ and let $d=6n-1$. Consider the language

$$L'_M = L_{\text{control}} \cup L_{\text{left}} \cup L_{\text{right}}, \tag{14a}$$

where

$$L_{\text{control}} = \{w \mid |w| = 1\,(\text{mod}\,d)\}, \tag{14b}$$

$$L_{\text{left}} = \{xw \mid |w| = 1\,(\text{mod}\,d),\ |x| = 2n+2i-1,\ \text{where } \Delta(I(w)) = q_i\}, \tag{14c}$$

$$L_{\text{right}} = \{wy \mid |w| = 1\,(\text{mod}\,d),\ |y| = 2i-1,\ \text{where } \Delta(I(w)) = q_i\} \tag{14d}$$

defined with respect to $M$. Note that all strings from $L_{\text{left}}$ have length $2n+2i$ modulo $d$ $(1 \leqslant i \leqslant n)$, while the strings from $L_{\text{right}}$ have length $2i$ modulo $d$. Thus each of the three

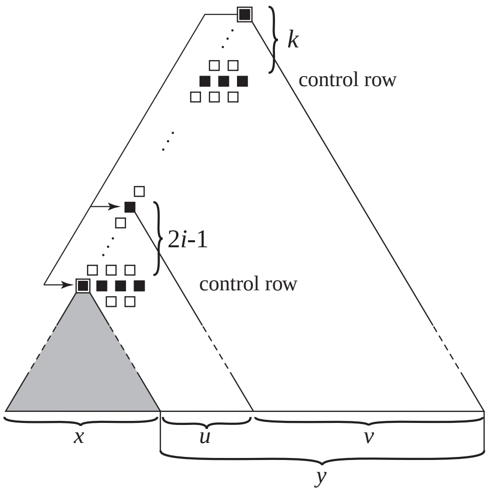


Fig. 6. Decoding of the states: $xuv \in locate[k](L'_M)$ $(\Delta(I(x)) = q_i$ for some $i)$.

components in (14) has its own set of admissible lengths modulo $d$, and these sets are disjoint.

**Lemma 2.** *For every trellis automaton $M = (\Sigma, Q, I, \delta, F)$ there exists and can be effectively constructed a language equation $X = \varphi(X)$, where $\varphi$ may contain the operations of union, intersection and concatenation of terminals, such that language* (14) *corresponding to $M$ is the unique solution of this equation.*

**Proof.** For every $k$, such that $1 \leqslant k \leqslant d$, define the function

$$locate[k](A) = \bigcup_{q_i \in Q} \bigcup_{\substack{u,v:\ |u|=2i-1,\\ |uv|=d+k-1}} Auv \cap Av, \tag{15}$$

which maps a language (denoted by the language variable $A$) to a language. Let us substitute $A = L'_M$; it is claimed that a string $w$ is in $locate[k](L'_M)$ if and only if $|w| \geqslant d+1$ and $|w| = k \pmod d$.

$\Leftarrow$ In order to prove that every such $w$ is in $locate[k](L'_M)$, factorize it as $w = xy$, where $|x| = 1 \pmod d$ and $d \leqslant |y| < 2d$. Since $|w| = k \pmod d$, $|y| = k-1 \pmod d$, and hence $|y| = d+k-1$. Let $q_i = \Delta(I(x))$ be the result of computation of $M$ on $x$. Further factorize $y$ as $uv$, where $|u| = 2i-1$. Then $xu \in L_{\text{right}}$. On the other hand, $x \in L_{\text{control}}$. Hence, $xu \in L'_M$ and $x \in L'_M$, and therefore, $xuv \in L'_M v \cap L'_M uv \subseteq locate[k](L'_M)$. This is illustrated in Fig. 6.

$\Rightarrow$ Let us prove the converse. If $w \in locate[k](L'_M)$, then $w$ can be factorized as $xuv$, where $x \in L'_M$, $xu \in L'_M$, $|u| = 2i-1$ and $|uv| = d-1+k$ for some $1 \leqslant i \leqslant n$. Since $x$ is in (14), consider the two potential possibilities:

- $x \in L_{\text{control}}$ (i.e., the control row has been correctly identified as intended). Then $|x| = 1 \pmod d$; therefore, $|w| = |xuv| = 1 + d - 1 + k = k \pmod d$. Also, $x$

must be at least 1 symbol long, and hence $|w|=|xuv| \geqslant 1+d-1+k=d+k \geqslant d+1$.

- $x \in L_{\text{right}}$ or $x \in L_{\text{left}}$ (supposing that the some data bit could be mistaken for a control row). Then $|x|=2j \pmod d$ for some $1 \leqslant j \leqslant 2n$, and thus $x$ can be factorized as $yz$, where $|y|=1 \pmod d$ and $|z|=2j-1$. Then $xu=y \cdot zu$ is a concatenation of $y$ (of length $1 \operatorname{modulo} d$) and a string $zu$, such that $|zu|=2(i+j)-2$, where $2 \leqslant i+j \leqslant 3n$, and therefore $|zu|$ is an even number between 2 and $6n-2$.

This upper bound $6n-2$ does not permit $yzu$ to reach the next control row and the rows above; the evenness of $|zu|$ shows that $yzu$ cannot be in $L_{\text{left}}$ or in $L_{\text{right}}$. Hence, $yzu=xu \notin L'_M$, which yields a contradiction, proving that this case is in fact impossible.

Note that the empty rows inserted between the data rows (see Fig. 3) were essentially used to rule out this possibility.

Now, define

$$\varphi_{\text{control}}(A) = locate[1](A) \cup \bigcup_{a \in \Sigma} a, \tag{16}$$

which, by the argument above, implies

$$\varphi_{\text{control}}(L'_M) = L_{\text{control}}. \tag{17}$$

Obtaining a similar representation of $L_{\text{right}}$ and $L_{\text{left}}$ requires a more complicated construction. First, it is needed to decode the information on the states contained in (14c) and (14d), which will be done by the following function:

$$is[q_i,j](A) = \begin{cases} \displaystyle\bigcup_{\substack{u,z:\ |u|=j-1,\\ |z|=d+1-j-(2i-1)}} uAz & \text{if } 1 \leqslant j \leqslant 4n, \\ \displaystyle\bigcup_{\substack{y,v:\ |y|=j-1-(2n+2i-1),\\ |v|=d+1-j}} yAv & \text{if } 4n+1 \leqslant j \leqslant d+1, \end{cases} \tag{18}$$

which ensures that the state in the $j$th relative position in the control row below is $q_i$ (see Fig. 7). Formally, the claim is that for every string $w$ ($|w|>d$) of length $1 \operatorname{modulo} d$, $w \in is[q_i,j](L'_M)$ if and only if for the partition $w=uxv$ ($|u|=j-1$, $|v|=d+1-j$) it holds that $\Delta(I(x))=q_i$.

$\ominus$ Let $w=uxv$, where $|u|=j-1$, $|v|=d+1-j$ and $\Delta(I(x))=q_i$. Note that, since $|uxv|=1 \pmod d$ and $|uv|=d$, $x$ also has length $1 \operatorname{modulo} d$. Consider two cases:

- If $j \leqslant 4n$, then $|v|=6n-j \geqslant 2n$, and hence $v$ can be factorized as $yz$, where $|y|=2i-1$ and $|z|=d+1-j-(2i-1)$. By (14d), $xy \in L_{\text{right}}$, and thus $w=uxyz \in uL'_M z \subseteq is[q_i,j](L'_M)$. This case is illustrated in Fig. 7(a).
- If $j \geqslant 4n+1$, $|u| \geqslant 4n$, and $u$ can be factorized as $yz$, where $|z|=2n+2i-1$ and $|y|=j-1-(2n+2i-1)$. By (14c), $zx \in L_{\text{left}}$, and $w=yzxv \in yL'_M v \subseteq is[q_i,j](L'_M)$. See Fig. 7(b).

Note that for low values of $j$ the relevant bits from $L_{\text{left}}$ would be inaccessible, and similarly for $j$ close to $d+1$ the bits from $L_{\text{right}}$ might be out of reach; this

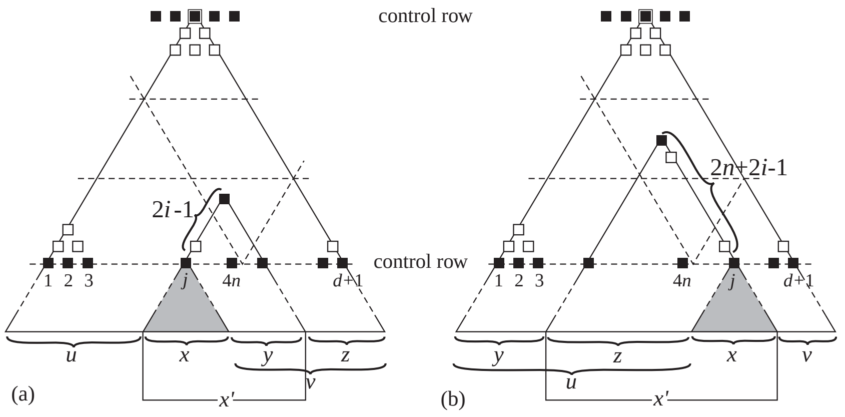


Fig. 7. Decoding of the states: $is[q_i, j](L'_M)$, the cases (a) $j \leqslant 4n$ and (b) $j > 4n$.

is the exact reason why the same state is redundantly encoded in two different places.

⊜ Let $w \in is[q_i, j](L'_M)$ ($|w| > d$, $|w| = 1 \pmod d$). Then, by (18):

- If $1 \leqslant j \leqslant 4n$, then $w = ux'z$, where $|u| = j - 1$, $x' \in L'_M$ and $|z| = d + 1 - j - (2i - 1)$. Factorize $x'$ as $xy$, where $|y| = 2i - 1$. Note that $|uyz| = j - 1 + 2i - 1 + d + 1 - j - (2i - 1) = d$ and $|x| = |w| - |uyz| = 1 - d = 1 \pmod d$. Since $|u| = j - 1$ and $|yz| = 2i - 1 + d + 1 - j - (2i - 1) = d + 1 - j$, the factorization $w = u \cdot x \cdot yz$ satisfies the requirement above. Since $|y| = 2i - 1$, the string $xy \in L'_M$ has to be in $L_{\text{right}}$, and therefore, by (14d), $\Delta(I(x)) = q_i$.
- If $4n + 1 \leqslant j \leqslant d + 1$, then $w = yx'v$, where $|y| = j - 1 - (2n + 2i - 1)$, $x' \in L'_M$ and $|v| = d + 1 - j$. Factorize $x' = zx$, where $|z| = 2n + 2i - 1$. As in the previous case, $|yzv| = j - 1 - (2n + 2i - 1) + (2n + 2i - 1) + d + 1 - j = d$ and $|x| = 1 \pmod d$. The factorization $w = yz \cdot x \cdot v$ is again as required, since $|yz| = j - 1$. According to (14c), $zx \in L_{\text{left}}$, and therefore $\Delta(I(x)) = q_i$.

This proves that (18) works as intended: for every $w$, such that $|w| > d$ and $|w| = 1 \pmod d$, $w \in is[q_i, j](L'_M)$ if and only if $M$ produces $q_i$ on certain substring of $w$ that is $d$ symbols *shorter*. Let us now devise a expression over $L'_M$ that contains $w$ ($|w| > d$, $|w| = 1 \pmod d$) if and only if $M$ produces $q_i$ on *the same* string $w$.

Indeed, decoding $d + 1$ adjacent states using $is[q_i, j]$ allows us to simulate $d$ rows of a computation of trellis automaton and thus determine $\Delta(I(w))$. Define

$$computes[q_i](A) = \bigcup_{\substack{q_{t_1}, \ldots, q_{t_{d+1}} \in Q: \\ \Delta(q_{t_1} \ldots q_{t_{d+1}}) = q_i}} \bigcap_{j=1}^{d+1} is[q_{t_j}, j](A) \tag{19}$$

and now, for every $w$ as above (i.e., $|w| = 1 \pmod d$ and $|w| > d$), $w \in computes[q_i](A)$ if and only if $\Delta(I(w)) = q_i$. This process is explained in Fig. 8.

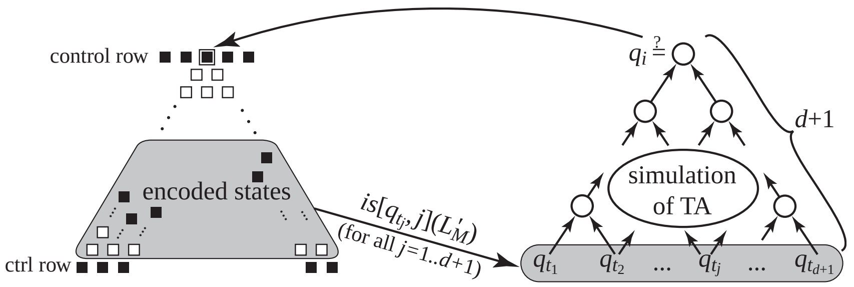


Fig. 8. Decoding of the states: $computes[q_i](L'_M)$.

In terms of $locate[k]$ and $computes[q_i]$ it is easy to define the functions:

$$\varphi_{\text{left}}(A) = \bigcup_{\substack{w \in L_{\text{left}},\\ |w| \leqslant d}} w \cup \bigcup_{q_i \in Q} \left( locate[2n+2i](A) \cap \bigcup_{u:\ |u|=2n+2i-1} u \cdot computes[q_i](A) \right), \tag{20a}$$

$$\varphi_{\text{right}}(A) = \bigcup_{\substack{w \in L_{\text{right}},\\ |w| \leqslant d}} w \cup \bigcup_{q_i \in Q} \left( locate[2i](A) \cap \bigcup_{v:\ |v|=2i-1} computes[q_i](A) \cdot v \right) \tag{20b}$$

and prove that

$$\varphi_{\text{left}}(L'_M) = L_{\text{left}}, \tag{21a}$$

$$\varphi_{\text{right}}(L'_M) = L_{\text{right}}. \tag{21b}$$

Let us give the proof for (21b). By (20b), $w \in \varphi_{\text{right}}(L'_M)$ if and only if either $|w| \leqslant d$ and $w \in L_{\text{right}}$, or there exists $i$ ($1 \leqslant i \leqslant n$), such that

$$w \in locate[2i](L'_M) \tag{22}$$

and a factorization $w = xv$ (where $|v| = 2i-1$) yields

$$x \in computes[q_i](L'_M). \tag{23}$$

As proved above, (22) is equivalent to $|w| > d$ and $|w| = 2i \pmod d$. This means that $|x| = |w| - |v| = 1 \pmod d$, and hence, by the properties of $computes[q_i]$, (23) is equivalent to $\Delta(I(x)) = q_i$.

Taking note that these conditions on $v$ and $x$ are as in (14d), it can be concluded that $w \in \varphi_{\text{right}}(L'_M)$ if and only if either $|w| \leqslant d$ and $w \in L_{\text{right}}$, or $|w| > d$ and $w \in L_{\text{right}}$—i.e., if and only if $w \in L_{\text{right}}$. This completes the proof of (21b); the case of $\varphi_{\text{left}}$ is handled in exactly the same way.

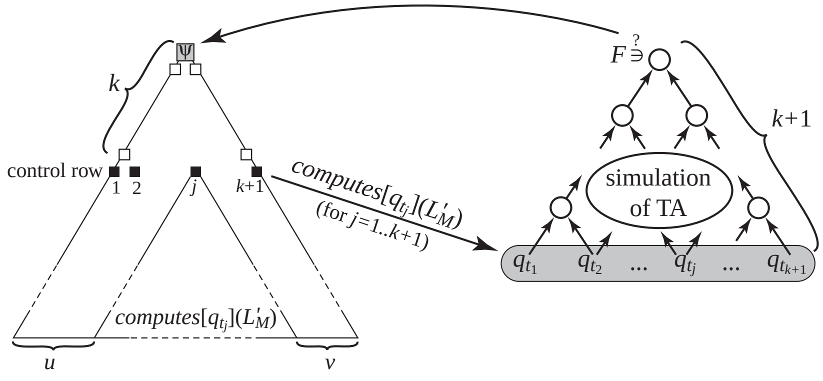


Fig. 9. Decoding $L(M)$ out of $L'_M$ using $\psi$.

Now consider the language equation:

$$X = \varphi_{\text{control}}(X) \cup \varphi_{\text{left}}(X) \cup \varphi_{\text{right}}(X). \tag{24}$$

By (17), (21a), (21b) and (14a), the language $L'_M$ is a solution of this equation. On the other hand, this equation is strict (see Definition 6), and therefore its solution $L'_M$ is unique, which completes the proof of the lemma. □

## 3.3. Decoding the original language

**Lemma 3.** *For every trellis automaton $M = (\Sigma, Q, I, \delta, F)$, there exists an expression $\psi(X)$ over union, intersection and linear concatenation, such that $\psi(L'_M) = L(M)$, where $L'_M$ is as in* (14).

**Proof** (Sketch of proof). Let *locate*$[k](X)$ and *computes*$[q](X)$ be as in the proof of Lemma 2. Define

$$\psi = \bigcup_{\substack{w \in L(M):\\ |w| \leqslant d}} w \cup \bigcup_{k=1}^{d} \left( locate[k](X) \cap \right.$$
$$\left. \bigcup_{\substack{q_{t_1},\ldots,q_{t_{k+1}} \in Q:\\ \Delta(q_{t_1}\ldots q_{t_{k+1}}) \in F}} \bigcap_{j=1}^{k} \bigcup_{\substack{u,v:\ |u|=j-1,\\ |v|=k+1-j}} u \cdot computes[q_{t_j}](X) \cdot v \right). \tag{25}$$

It is left to demonstrate that, indeed, $\psi(L'_M) = L(M)$.

For every string $w$ of length $ld+k$ ($l \geqslant 1$, $1 \leqslant k \leqslant d$), it suffices to decode the outcome of the computation of $M$ on all substrings of $w$ of length $ld$ from the language $L'_M$ (using the mapping *computes*), thus obtaining a vector of $k+1$ states, and then ensure that the subcomputation of $M$ on this vector leads to a final state. This method is illustrated in Fig. 9.

Formula (25) separately considers all possible lengths of $w$ modulo $d$ (the union $\bigcup_{k=1}^{d}$ and the reference to $locate[k](X)$), and for each $k$ it tries all possible vectors of $k+1$ states that lead to a final state. For every such vector $(q_{t_1},\dots,q_{t_{k+1}})$, each of its states $q_{t_j}$ is checked for being the outcome of the computation of $M$ on the substring $x$ of $w$, such that $w=uxv$, $|u|=j-1$ and $|v|=k+1-j$. This can be done by a single reference to $computes[q_{t_j}](L'_M)$. □

Now the main result of this paper can be stated:

**Theorem 7.** *For every trellis automaton $M=(\Sigma,Q,I,\delta,F)$, there exist and can be effectively constructed*:

- *a system of language equations with two variables that has unique solution $(L(M), L'_M)$.*
- *a linear conjunctive grammar $G$ with two nonterminals, such that $L(G)=L(M)$.*

**Proof.** Let $X=\varphi(X)$ be the language equation constructed in Lemma 2, which has unique solution $L'_M$. Let $\psi$ be the expression constructed in Lemma 3. Then the system

$$\begin{aligned} S &= \psi(X), \\ X &= \varphi(X) \end{aligned} \tag{26}$$

is easily seen to have unique solution $(L(M),L'_M)$, which proves the first part of the theorem.

A two-nonterminal linear conjunctive grammar for $L(M)$ can be constructed out of system (26) by Theorem 4. □

Another related result is that every linear conjunctive language can be "almost" represented by a single language equation:

**Theorem 8.** *Let $\Sigma$ be an alphabet, let $\# \notin \Sigma$. Then for every trellis automaton $M$ there exists and can be effectively constructed a single language equation that has unique solution $L'_M \cup \#L(M)$.*

**Proof.** Let $X=\varphi(X)$ be the language equation from Lemma 2 and let $\psi$ be the expression from Lemma 3. Define the expression

$$\mu(X) = X \cap \left( \bigcup_{a\in\Sigma} a \cup \bigcup_{\substack{u\in\Sigma^*:\\ 1\leqslant |u| \leqslant d+1}} uX \right) \tag{27}$$

and the equation

$$X = \varphi(\mu(X)) \cup \#\psi(\mu(X)). \tag{28}$$

Let us prove that Eq. (28) has unique solution $L'_M \cup \#L(M)$.

The first claim is that

$$\mu(L'_M \cup \#L(M)) = L'_M. \tag{29}$$

The left-hand side of (29) can be equivalently transformed as follows:

$$\begin{aligned}
&\mu(L'_M \cup \#L(M)) \\
&\quad = (L'_M \cup \#L(M)) \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u \cdot (L'_M \cup \#L(M)) \right) \\
&\quad = (L'_M \cup \#L(M)) \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} uL'_M \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u\#L(M) \right) \\
&\quad = L_1 \cup L_2,
\end{aligned}$$

where

$$L_1 = L'_M \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} uL'_M \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u\#L(M) \right), \tag{30a}$$

$$L_2 = \#L(M) \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} uL'_M \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u\#L(M) \right). \tag{30b}$$

Language (30a) equals

$$L_1 = \underbrace{L'_M \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} uL'_M \right)}_{L'_M} \cup \underbrace{L'_M \cap \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u\#L(M)}_{\emptyset} = L'_M,$$

where the first part follows from the containment of every string of length 1 modulo $d$ in $L'_M$, while the second component is $\emptyset$, because no string in $L'_M$ contains #. Turning to (30b):

$$L_2 = \#L(M) \cap \left( \bigcup_{a \in \Sigma} a \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} uL'_M \cup \bigcup_{\substack{u \in \Sigma^*: \\ 1 \leqslant |u| \leqslant d+1}} u\#L(M) \right) = \emptyset,$$

here the two languages are disjoint, because all strings in $\#L(M)$ start from #, while all strings in the other language start from some symbol in $\Sigma$.

Putting together the equalities obtained:

$$\mu(L'_M \cup \#L(M)) = L_1 \cup L_2 = L'_M \cup \emptyset = L'_M, \tag{31}$$

which proves claim (29).

Now let us substitute $L'_M \cup \#L(M)$ for $X$ in (28): $\varphi(\mu(L'_M \cup \#L(M))) \cup \#\psi(\mu(L'_M \cup \#L(M)))$ equals $\varphi(L'_M) \cup \#\psi(L'_M)$ by Eq. (29), which in turn equals $L'_M \cup \#\psi(L'_M)$ by

Lemma 2 and $L'_M \cup \#L(M)$ by Lemma 3, thus proving that $L'_M \cup \#L(M)$ is a solution of (28). This solution is unique, because the equation is strict. □

## 4. One-nonterminal grammars

Having proved that every linear conjunctive language can be generated using two nonterminal symbols, it is natural to investigate whether this estimation is precise, and could not a single nonterminal be sufficient. It turns out that it could not, and, as it will be shortly demonstrated, there exist linear conjunctive languages that require two nonterminals.

This section is devoted to the study of one-nonterminal linear conjunctive grammars and of the language family they generate. Due to the correspondence between linear conjunctive grammars and systems of language equations, these are the languages denoted by unique (or least) solutions of individual language equations with union, intersection and linear concatenation, resolved with respect to their single variable; this characterization makes the study of their properties worthwhile.

A similar object, linear context-free grammars with one nonterminal—*minimal linear context-free grammars*—have been studied in the early days of formal language theory [3,8,10] (note that Chomsky and Schützenberger [3] and Haines [10] use a more restrictive definition than just the singularity of a nonterminal; the present paper follows the definition of Greibach [8], also assumed by Harrison [11]), and some key results on linear context-free grammars were demonstrated to hold for this class: this is the existence of languages with non-context-free complement [10] and the undecidability of the ambiguity problem [8].

The conjunctive counterpart of this class, linear conjunctive languages of nonterminal complexity 1, as it was already demonstrated in Theorem 8, similarly share some essential qualities of linear conjunctive languages of the general form. Their basic formal properties are studied in this section.

### 4.1. The shrinking lemma

Let us first establish the following *shrinking lemma* for one-nonterminal linear conjunctive grammars, using which one can prove that some linear conjunctive languages actually require two nonterminals.

**Lemma 4** (Shrinking lemma). *Let $G = (\Sigma, \{S\}, P, S)$ be a linear conjunctive grammar comprised of a single nonterminal. Then there exists a constant $k \geqslant 0$, such that for every string $w \in L(G)$, where $|w| > k$, there exists a factorization $w = xuy$ $(0 < |x| + |y| \leqslant k)$, such that $u \in L(G)$.*

**Proof.** Let $G = (\Sigma, \{S\}, P, S)$ be an arbitrary linear conjunctive grammar that generates an infinite language. Assume, without loss of generality, that $G$ does not contain unit conjuncts of the form $S \to S$ (all rules containing this conjunct are of no use and can be removed from $P$ [16]). Let

$$k = \max_{S \to \alpha \in conjuncts(P)} |\alpha| \tag{32}$$

be the maximum length of a right-hand side of a conjunct. Let $w \in L(G)$ be any string, such that $|w| > k$.

Since $S \Longrightarrow^* w$, there exists a rule $S \to \alpha_1 \& \cdots \& \alpha_m$, such that $\alpha_i \Longrightarrow^* w$ for all $i$. Consider the first conjunct of the rule: $\alpha_1$ cannot be in $\Sigma^*$, since that would imply $\alpha_1 = w$ and $|\alpha_1| > k$, which is a contradiction. So, $\alpha_1 = xSy$ for some $x, y \in \Sigma^*$ $(0 < |x| + |y| \leqslant k)$. Since $\alpha_1 \Longrightarrow^* w$, this implies that $w = xuy$ for some $u \in \Sigma^*$, such that $S \Longrightarrow^* u$. This factorization satisfies the statement of the lemma. □

The shrinking lemma is a necessary but not a sufficient condition of being a one-nonterminal language. An example of language that satisfies shrinking lemma but is not one-nonterminal will be constructed in Theorem 9 (the case of union). However, in many cases the shrinking lemma does work:

**Proposition 1.** *The regular language $ba^*b$ cannot be generated by a one-nonterminal linear conjunctive grammar.*

**Proof.** Suppose $L = ba^*b$ is generated by such a grammar. Then, by Lemma 4, there exists a constant $k \geqslant 0$, such that for the string $ba^kb \in L$ there exists a factorization $ba^kb = xuy$ $(|x| + |y| > 0)$, where $u \in L$. Therefore, a string of the form $ba^i$, $a^ib$ or $a^i$ must be in $L$, which is a contradiction. □

Could there be a *pumping* lemma for one-nonterminal linear conjunctive grammars? The answer is negative. It is known that linear conjunctive languages of the general form do not have a pumping lemma due to the undecidability of the emptiness problem. If one supposes that there is a pumping lemma for the one-nonterminal case, then, given an arbitrary linear conjunctive grammar $G$, one can construct an equivalent trellis automaton $M$ and consider the language $L'_M \cup \#L(M)$ that is a one-nonterminal language by Theorem 8. Since the supposed one-nonterminal pumping lemma should apply to every sufficiently long string $\#w \in \#L(M)$ from this language, producing $\#w' \in \#L(M)$, its statement would at the same time be applicable to the corresponding string $w \in L(M)$, similarly producing $w' \in L(M)$. This gives a pumping lemma for linear conjunctive grammars of the general form, which is a contradiction.

Let us continue with some examples of languages that can be denoted with one-nonterminal grammars.

**Proposition 2.** *There exists a linear conjunctive grammar with one nonterminal for the regular language $a^*b \cup ba^*$.*

**Proof.** The grammar is $S \to b|ab|ba|aaS\&aS|Saa\&Sa$. While the first three rules denote a base set of strings, the last two specify that if a string begins with an $a$ (ends with an $a$, resp.), then one more $a$ can be appended to its beginning (to its end, resp.).

Formally, it can be proved that this grammar generates $a^*b \cup ba^*$ by checking that this language is a solution of the corresponding language equation. □

Note that, although the language $a^*b \cup ba^*$ is obviously linear context-free, it is easy to prove that no one-nonterminal linear context-free grammar generates it.

**Proposition 3.** *There exists a language generated by a one-nonterminal linear conjunctive grammar, which cannot be represented as a finite intersection of context-free languages.*

**Proof.** Consider the linear conjunctive language $L = \{wcw \mid w \in \{a, b\}^*\}$, and let $M = (\Sigma, Q, I, \delta, F)$ be an arbitrary trellis automaton for this language. Let $L'_M$ be the (one-nonterminal) language constructed using the method of Lemma 2, and let us prove that it is not in the intersection closure of the context-free languages.

Suppose it can be represented as an intersection of finitely many context-free languages. Then, by (25) in Lemma 3, the language $L$ can also be represented as such a finite intersection, which is known to be untrue [26]. □

A wealth of other one-nonterminal languages is provided by Theorem 8. Its result will be used in the later Section 4.4 to find the hardest language in the family.

## *4.2. Closure properties*

**Theorem 9.** *The family of one-nonterminal linear conjunctive languages is not closed under union, intersection, complement, concatenation and star.*

**Proof.** The case of *concatenation* is straightforward: it suffices to represent the language $ba^*b$, which is not a one-nonterminal language by Proposition 1, as $b \cdot a^*b$.

In order to prove nonclosure under *intersection*, consider the languages $L_1 = (a \cup b)^*b$ and $L_2 = b(a \cup b)^*$. Both can clearly be generated using one nonterminal, but their intersection $L_1 \cap L_2 = b(a \cup b)^*b \cup b$ cannot, which is proved in the same way as in Proposition 1.

Turning to the case of *complement*, let $L = a^*b \cup ba^* \cup a^*$. $L$ is generated by a grammar $S \to \varepsilon | a | b | ab | ba | aaS \& aS | Saa \& Sa$, based on the same idea as the grammar from Proposition 2. Let us prove that its complement, $\overline{L} = a^*ba^*b(a \cup b)^* \cup a^+ba^+$, cannot be generated with one nonterminal. Supposing that it can, let $k$ be the constant given by the shrinking lemma, and consider the string $w = ba^kb \in \overline{L}$. There should exist a factorization $w = xuy$, such that $0 < |x| + |y| \leqslant k$ and the string $u$ is in $\overline{L}$; however, every such factorization results in $u \in ba^*$, $u \in a^*b$ or $u \in a^*$, which implies $u \notin \overline{L}$ and yields a contradiction.

Proving that one-nonterminal languages are not closed under *union* is somewhat harder, because the shrinking lemma will not work: it is easy to see that for every $L_1, L_2$ that satisfy the shrinking lemma, the language $L_1 \cup L_2$ also does. Consider the languages:

$$L_1 = \{a^m b a^n \mid m < n\}, \tag{33a}$$

$$L_2 = \{a^m b a^n \mid m > n\}. \tag{33b}$$

$L_1$ is generated by the grammar $S \to aS|aSa|b$, and $L_2$ by a similar one. Let us give a direct proof that their union, $L = L_1 \cup L_2 = \{a^m b a^n \mid m \neq n\}$ is not a one-nonterminal language.

Suppose $L$ is generated by a linear conjunctive grammar $G = (\{a, b\}, \{S\}, P, S)$; as in Lemma 4, assume without loss of generality that $S \to S \notin conjuncts(P)$. As in (32), let $k$ be the maximum length of conjuncts' right-hand sides. Consider the derivation of the string $ba^k \in L$:

$$S \Longrightarrow (\alpha_1 \& \cdots \& \alpha_l) \Longrightarrow \cdots \Longrightarrow ba^k \tag{34}$$

It is easy to see that every $\alpha_i$ must be of the form $Sa^{j_i}$, where $j_i > 0$. Indeed, the case $\alpha_i = ba^k$ is impossible by the definition of $k$, because $|ba^k| > k$; $\alpha_i$ cannot be in $ba^*Sa^*$, because then $ba^*Sa^* \Longrightarrow^* ba^k$ would mean that some string from $a^*$ is derivable from $S$, which cannot be true; $\alpha_i \in Sa^+$ is the only remaining opportunity to derive $ba^k$. Therefore, derivation (34) is of the form

$$S \Longrightarrow (Sa^{j_1} \& \cdots \& Sa^{j_l}) \Longrightarrow \cdots \Longrightarrow ba^k \tag{35}$$

and the grammar contains the rule $S \to Sa^{j_1} \& \cdots \& Sa^{j_l}$ $(j_i > 0)$.

Using this rule and taking note that for every $i$ the string $a^k ba^{k-j_i}$ is in $L$, construct the following derivation:

$$\begin{aligned} S &\Longrightarrow (Sa^{j_1} \& \cdots \& Sa^{j_l}) \Longrightarrow \cdots \Longrightarrow (a^k ba^{k-j_1} a^{j_1} \& \cdots \& a^k ba^{k-j_l} a^{j_l}) \\ &\Longrightarrow a^k ba^k. \end{aligned} \tag{36}$$

Since $a^k ba^k \notin L$, this contradicts the assumption that $L(G) = L$, proving that $L = L_1 \cup L_2$ is not a one-nonterminal linear conjunctive language, and thus this family is not closed under union.

It remains to prove nonclosure under *star*. Let $L = \{a^n b^n \mid n \geqslant 0\}$, an obvious one-nonterminal language. Suppose that $L^*$ is a one-nonterminal language as well, and let $k$ be the constant given by the shrinking lemma. Consider $w = a^k b^k a^k b^k \in L^*$. Since every factorization $w = xuy$ $(0 < |x| + |y| \leqslant k)$ gives $u = a^{k-|x|} b^k a^k b^{k-|y|} \notin L^*$ (because $k - |x| \neq k$ or $k \neq k - |y|$), a contradiction is obtained. □

## 4.3. Decision problems

**Theorem 10.** *The emptiness and universality problems for one-nonterminal linear conjunctive grammars are decidable. The equivalence problem is undecidable.*

The decidability results are proved by establishing necessary and sufficient conditions for emptiness and universality. These conditions are given in Lemmata 5 and 6, which will be followed by the proof of Theorem 10.

**Lemma 5.** *Let $G=(\Sigma,\{S\},P,S)$ be a one-nonterminal linear conjunctive grammar over $\Sigma$. Then $L(G)\neq\emptyset$ if and only if $P$ contains a rule of the form $S\to w$ $(w\in\Sigma^*)$.*

**Proof.** If there is such a rule, then $w\in L(G)$ and thus $L(G)\neq\emptyset$. Conversely, for any conjunctive derivation to be successful, it has to contain at least one application of a rule of the form $A\to w$, and thus if there is no rule of this form, then $L(G)=\emptyset$. □

**Lemma 6.** *Let $G=(\Sigma,\{S\},P,S)$ be a linear conjunctive grammar, such that $S\to S\notin$ conjuncts$(P)$, and let $d$ be the maximum length of right-hand side of a conjunct. Then $L(G)=\Sigma^*$ if and only if*

- *Every string of length less or equal to $2d$ is in $L(G)$, and*
- *For every string $w$ of length $2d$ there exists a rule*

$$S\to u_1Sv_1\&\cdots\&u_lSv_l\in P, \tag{37}$$

*such that every $u_i$ is a prefix of $w$ and every $v_i$ is a suffix of $w$.*

**Proof.** $\textcircled{\Rightarrow}$ If $L(G)=\Sigma^*$, then the first condition is obviously satisfied. Supposing that the second is not—i.e., there exists a string $w$, such that no rule in $P$ has all conjuncts in the required form—one can easily see that $w$ cannot be derived at all and therefore $L(G)\neq\Sigma^*$, which is untrue.
$\textcircled{\Leftarrow}$ Let both conditions hold and let us show that every $w\in\Sigma^*$ is in $L(G)$. The proof is an induction on the length of $w$.

*Basis*: $|w|\leqslant 2d$. Every such $w$ is in $L(G)$ by the first condition.

*Induction step*: $|w|\geqslant 2d$. Factorize the string as $w=w'zw''$, where $|w'|=|w''|=d$. Now for the string $w'w''$ there should exist a rule of form (37), such that every $u_i$ is a prefix of $w'w''$ and every $v_i$ is a suffix of $w'w''$. Since $|u_i|,|v_i|\leqslant d$ and $|w'|=|w''|=d$, it follows that for every $i$, $w'=u_ix_i$ and $w''=y_iv_i$ for some $x_i,y_i\in\Sigma^*$.

Since every string $x_izy_i$ is shorter than $w=u_ix_izy_iv_i$, it is in $L(G)$ by the induction hypothesis. This allows to construct the derivation

$$S\Longrightarrow(u_1Sv_1\&\cdots\&u_lSv_l)\Longrightarrow(u_1x_1zy_1v_1\&\cdots\&u_lx_lzy_lv_l)\Longrightarrow w'zw'', \tag{38}$$

which proves that $w\in L(G)$. □

**Proof** (Proof of Theorem 10). *The emptiness problem*, stated as "given a linear conjunctive grammar $G=(\Sigma,\{S\},P,S)$, determine whether $L(G)=\emptyset$", can be algorithmically solved by checking the condition given in Lemma 5. Similarly, *the universality problem* ("given $G$, determine whether $L(G)=\Sigma^*$") can be decided according to Lemma 6.

Let us show that there is no algorithm to solve *the equivalence problem* for one-nonterminal linear conjunctive grammars (given $G_1$ and $G_2$, determine whether $L(G_1)=L(G_2)$). Suppose it is decidable and construct an algorithm for checking the emptiness of the language generated by a linear conjunctive grammar of the general form, which is easily seen to be undecidable by reduction from Turing machine halting problem [18].

Let $G$ be an arbitrary linear conjunctive grammar over $\Sigma$. Construct a trellis automaton $M=(\Sigma, Q, I, \delta, F)$ that accepts $L(G)$ and consider the equation $X=\varphi(X)$ from Lemma 2, which has unique solution $L'_M$ as in (14), and the equation $X=\varphi(\mu(X))\cup\#\psi(\mu(X))$ from Theorem 8, which has unique solution $L'_M\cup\#L(M)$.

Taking note that $L'_M=L'_M\cup\#L(M)$ if and only if $L(M)=\emptyset$, each of the two equations can be converted to a one-nonterminal grammar using the method of Theorem 4, and then by checking the equivalence of these grammars one can decide the emptiness of $L(G)$, which contradicts its known undecidability. □

## *4.4. The hardest language and the membership problem*

In [13] it was shown that some **P**-complete languages can be accepted by a certain restricted type of Turing machines and thus by trellis automata; this was rediscovered in [20] using trellis automata and linear conjunctive grammars. Using the results of this paper one can prove a stronger claim, stating that a one-nonterminal grammar (or a single resolved language equation) is actually enough to denote a **P**-complete language.

**Theorem 11.** *The family of one-nonterminal linear conjunctive languages contains a* **P**-*complete language.*

**Proof.** Let $L_0$ be some **P**-complete linear conjunctive language, let $M$ be a trellis automaton for $L_0$. Using the method of Theorem 8, construct a one-variable language equation that has unique solution $L=L'_M\cup\#L(M)$ Clearly, $L_0$ is reducible to $L$ under any sensible definition of reducibility (it suffices to add # to the beginning of a string), which proves the **P**-hardness of $L$.

On the other hand, $L$ is in **P** as a linear conjunctive language. This completes the proof of the statement that $L$ is a **P**-complete one-nonterminal linear conjunctive language. □

The complexity of the general membership problem for this family of grammars now follows as a simple corollary of Theorem 11.

**Theorem 12.** *The membership problem for one-nonterminal linear conjunctive grammars, stated as "Given a grammar $G=(\Sigma,\{S\},P,S)$ and a string $w\in\Sigma^*$, determine whether $w\in L(G)$", is* ***P****-complete.*

**Proof.** It is solvable in polynomial time, because so is the membership problem for conjunctive grammars of the general form [19]. Its **P**-hardness can be proved by reducing the fixed membership problem in some one-nonterminal **P**-complete language, which exists by Theorem 11, to this more general problem. □

The properties of one-nonterminal linear conjunctive grammars are summarized and compared to those of linear conjunctive grammars of the general form in Table 1.

Table 1
Properties of one-nonterminal linear conjunctive grammars

| | One-nonterminal | Of the general form |
|---|---|---|
| *Closure properties* | | |
| $\cup$ | $-$ | $+$ |
| $\cap$ | $-$ | $+$ |
| $\sim$ | $-$ | $+$ |
| $\cdot$ | $-$ | $-$ [25] |
| $*$ | $-$ | $-$ [23] |
| *Decision problems* | | |
| Emptiness | $+$ | $-$ |
| Universality | $+$ | $-$ |
| Equivalence | $-$ | $-$ |
| The hardest language | **P**-complete | **P**-complete |

## 5. Nonterminal complexity of grammars in the linear normal form

While a finite number of nonterminals turns out to be sufficient in linear conjunctive grammars of the general form, this is not the case for the grammars in the linear normal form, since every $n$-nonterminal grammar in this normal form can be converted to a trellis automaton of $2^{n+C}$ states, where $C$ depends upon the alphabet [23], and there exist finitely many automata of a fixed size.

Let us investigate succinctness tradeoffs between these two representations.

### 5.1. Simulating trellis automata with grammars

The original proof of simulation of trellis automata by linear conjunctive grammars [23] represented every state of an $n$-state automaton with a nonterminal symbol and added one extra start symbol, thus resulting in $n+1$ nonterminals (see Theorem 6). In this section a new method, much more efficient in terms of the number of nonterminals, is proposed: an $n$-state trellis automaton is converted to a grammar with $O(\log_2 n)$ nonterminals.

Given an automaton $M=(\Sigma,Q,I,\delta,F)$, where $|Q|=n$, let $k=\lceil \log_2 n\rceil$ and rename the states (without loss of generality), so that $Q\subseteq\{0,1\}^k$. Now construct the grammar with the nonterminals $\{S,A_0^{(1)},\ldots,A_0^{(k)},A_1^{(1)},\ldots,A_1^{(k)}\}$ and with the following rules:

$$A_{i_t}^{(t)} \to a \quad (\text{if } I(a)=(i_1,\ldots,i_k);\ \text{for all } t), \tag{39a}$$

$$A_{l_t}^{(t)} \to bA_{j_1}^{(1)}\&\cdots\&bA_{j_k}^{(k)}\&A_{i_1}^{(1)}c\&\cdots\&A_{i_k}^{(k)}c$$
$$(\text{if } \delta((i_1,\ldots,i_k),(j_1,\ldots,j_k))=(l_1,\ldots,l_k);\ \text{for all } b,c\in\Sigma \text{ and } t), \tag{39b}$$

$$S\to A_{i_1}^{(1)}\&\cdots\&A_{i_k}^{(k)} \quad (\text{for all } (i_1,\ldots,i_k)\in F). \tag{39c}$$

Let us prove the correctness of construction.

**Lemma 7.** *For every string* $w \in \Sigma^+$, *and for every* $t$ $(1 \leqslant t \leqslant k)$ *and* $x \in \{0, 1\}$, *if* $A_x^{(t)} \overset{G}{\Longrightarrow}^* w$ *then the tth component of* $\Delta(I(w))$ *is* $x$.

**Proof.** Induction on the length of $w$.

*Basis*: $w = a \in \Sigma$. If $A_x^{(t)} \Longrightarrow^* a$, then there should be a rule $A_x^{(t)} \to a$ in the grammar. By construction (39a), this implies that the $t$th component of $I(a)$ is $x$.

*Induction step*: Let $|w| \geqslant 2$ and let $A_x^{(t)} \Longrightarrow^* w$. Then there exists a derivation of the form

$$A_x^{(t)} \Longrightarrow (bA_{j_1}^{(1)} \& \cdots \& bA_{j_k}^{(k)} \& A_{i_1}^{(1)} c \& \cdots \& A_{i_k}^{(k)} c) \Longrightarrow \cdots \Longrightarrow w. \tag{40}$$

This implies that $w = buc$ for some $u \in \Sigma^*$, $A_{i_t}^{(t)} \Longrightarrow^* bu$ (for all $t$) and $A_{j_t}^{(t)} \Longrightarrow^* uc$ (for all $t$). Invoking the induction hypothesis $2k$ times, one obtains that the first component of $\Delta(I(bu))$ is $i_1$, the second component of $\Delta(I(bu))$ is $i_2$ and so on, and similarly every $t$th component of $\Delta(I(uc))$ is $j_t$. Putting these facts together, $\Delta(I(bu)) = (i_1, \ldots, i_k)$ and $\Delta(I(uc)) = (j_1, \ldots, j_k)$.

Let $(l_1, \ldots, l_k) = \delta((i_1, \ldots, i_k), (j_1, \ldots, j_k)) = \Delta(I(w))$. By (39b) in the construction of the grammar, the existence of the rule $A_x^{(t)} = bA_{j_1}^{(1)} \& \cdots \& bA_{j_k}^{(k)} \& A_{i_1}^{(1)} c \& \cdots \& A_{i_k}^{(k)}$ implies that $x = l_t$, which is exactly the $t$th component of $\Delta(I(w))$. $\square$

**Lemma 8.** *For every string* $w \in \Sigma^+$, *if* $\Delta(I(w)) = (i_1, \ldots, i_k)$, *then* $A_{i_t}^{(t)} \overset{G}{\Longrightarrow}^* w$ *for all* $t$.

**Proof.** Induction on $|w|$.

*Basis*: $w = a \in \Sigma$. By the construction of grammar (39a), for every $t$ there is a rule $A_{i_t}^{(t)} \to a$, and hence $A_{i_t}^{(t)} \Longrightarrow^* a$.

*Induction step*: Let $\Delta(I(w)) = (l_1, \ldots, l_k)$. Let $w = buc$, where $b, c \in \Sigma$, $u \in \Sigma^*$. Let $\Delta(I(bu)) = (i_1, \ldots, i_k)$ and $\Delta(I(uc)) = (j_1, \ldots, j_k)$. By the induction hypothesis:

$$A_{i_t}^{(t)} \Longrightarrow^* bu \quad \text{(for all } t) \tag{41a}$$

and

$$A_{j_t}^{(t)} \Longrightarrow^* uc \quad \text{(for all } t). \tag{41b}$$

Since $\delta((i_1, \ldots, i_k), (j_1, \ldots, j_k)) = \delta(\Delta(I(bu)), \Delta(I(uc))) = \Delta(I(w)) = (l_1, \ldots, l_k)$, by the construction of the grammar there should be a rule (39b) for every $t$, which, together with all $2k$ rules (41), is enough to construct a derivation of $w$ from $A_{l_t}^{(t)}$. $\square$

**Theorem 13.** *For every n-state trellis automaton there exists an can be effectively constructed an equivalent linear conjunctive grammar in the linear normal form with* $2\lceil \log_2 n \rceil + 1$ *nonterminals.*

**Proof.** Construct the grammar as (39), and let us prove that for every string $w \in \Sigma^+$, $S \Longrightarrow^* w$ if and only if $\Delta(I(w)) \in F$.

$\bigcirc\!\!\!\!\Rightarrow$ Let $S \Longrightarrow^* w$. Consider the derivation

$$S \Longrightarrow (A_{i_1}^{(1)} \& \cdots \& A_{i_k}^{(k)}) \Longrightarrow \cdots \Longrightarrow w, \tag{42}$$

which implies that $A_{i_t}^{(t)} \overset{G}{\Longrightarrow}^* w$ for all $t$. Using Lemma 7, it can be obtained that for every $t$ the $t$th component of $\Delta(I(w))$ is $i_t$. The conjunction of these $k$ statements implies $\Delta(I(w)) = (i_1, \ldots, i_k)$. Now, since $S \to A_{i_1}^{(1)} \& \cdots \& A_{i_k}^{(k)} \in P$, from construction (39c) it follows that $(i_1, \ldots, i_k) \in F$.

$\bigcirc\!\!\!\!\Leftarrow$ Let $\Delta(I(w)) = (i_1, \ldots, i_k) \in F$. Then, by Lemma 8, $A_{i_t}^{(t)} \Longrightarrow^* w$ for every $t$. By the construction of the grammar, there is a rule (39c). This allows to construct the derivation

$$S \Longrightarrow (A_{i_1}^{(1)} \& \cdots \& A_{i_k}^{(k)}) \Longrightarrow \cdots \Longrightarrow (w \& \cdots \& w) \Longrightarrow w, \tag{43}$$

proving that $w \in L(G)$. □

### 5.2. Simulating grammars with trellis automata

How many states does a minimal trellis automaton equivalent to an $n$-nonterminal linear conjunctive grammar in the linear normal form have? The grammar-to-automaton construction of [23] provides $|\Sigma|^2 \cdot 2^n = 2^{n+C}$ upper bound. Let us infer $2^{\lfloor n/2 \rfloor - 1}$ lower bound from the automaton-to-grammar construction given in Theorem 13.

Suppose the contrary, i.e., that there exists $n > 0$, such that for every grammar in the linear normal form with $n$ nonterminals there exists an equivalent automaton with $2^{\lfloor n/2 \rfloor - 1} - 1$ states. Choose $m$, such that $n = 2\lceil \log_2 m \rceil + 1$. Then for every $m$-state automaton the construction from Theorem 13 can be applied to obtain a grammar comprised of $n$ nonterminals, and then the supposed construction can be used to get another $(2^{\lfloor (2\lceil \log_2 m \rceil + 1)/2 \rfloor - 1} - 1)$-state automaton that accepts the same language as the original one. However,

$$2^{\lfloor (2\lceil \log_2 m \rceil + 1)/2 \rfloor - 1} - 1 = 2^{\lfloor \lceil \log_2 m \rceil + \frac{1}{2} \rfloor - 1} - 1 = 2^{\lceil \log_2 m \rceil - 1} - 1 < 2^{\log_2 m} - 1 = m - 1$$

and thus it is shown that an arbitrary $m$-state automaton can be reduced, which cannot be true in light of the results of [22].

Thus it has been proved that the number of states in the minimal trellis automata equivalent to an $n$-nonterminal linear conjunctive grammar in the linear normal form lies between $2^{\lfloor n/2 \rfloor - 1}$ and $2^{n+C}$, i.e., is $2^{O(n)}$.

## 6. Conclusion

It was demonstrated that two nonterminals in a linear conjunctive grammar are sufficient to denote any language from this quite noteworthy family, and a certain variation of any such language can even be generated with a single nonterminal. Besides characterizing the nonterminal complexity of this family of grammars, these results will surely have impact on the theory of language equations, showing how much can be expressed in a single resolved equation that uses quite a modest set of language-theoretic operations.

A classification of linear conjunctive languages into two classes of those that require one nonterminal and those that require two of them was developed. One-nonterminal linear conjunctive languages, akin to the known minimal linear context-free languages, were found to cover some paradigmatic examples of languages from this family. Additionally, they turned out to have different decidability and closure properties than linear conjunctive grammars with two nonterminals. For the class of linear conjunctive grammars in the linear normal form, an exponential tradeoff between the number of nonterminals and the number of states in trellis automata was proved.

Naturally, the given technique of converting a linear conjunctive grammar to an equivalent one with two nonterminals causes a significant blowup in the total length of description. In future research it might be interesting to investigate the tradeoff between these two measures of succinctness.

A more interesting question is whether a bounded number of nonterminals is enough for conjunctive grammars of the general form, or is there an infinite hierarchy of $n$-nonterminal conjunctive languages, similar to the context-free one? If a fixed number of nonterminals is enough, then the construction would probably be quite different from the one presented in this paper, as the present method does not seem to be extendable to general conjunctive grammars. On the other hand, proving the more likely hierarchy result appears to be a challenging task, since it would most probably require to prove some languages not to be conjunctive, for which no technique is known yet.

## Acknowledgements

I am grateful to an anonymous referee for noting a subtle technical error in the initial version of the construction in Section 3, and to Kai Salomaa for his comments on different revisions of the paper.

## References

[1] J. Autebert, J. Berstel, L. Boasson, Context-free languages and pushdown automata, Rozenberg, Salomaa (Eds.), Handbook of Formal Languages, Vol. 1, Springer, Berlin, 1997, pp. 111–174.

[2] C. Choffrut, K. Culik II, On real-time cellular automata and trellis automata, Acta Inform. 21 (1984) 393–407.

[3] N. Chomsky, M.P. Schützenberger, The algebraic theory of context-free languages, in: Braffort, Hirschberg (Eds.), Computer Programming and Formal Systems, 1963, pp. 118–161.

[4] K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata", I, International J. Comput. Math. 15 (1984) 195–212;
K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata", II, International J. Comput. Math. 16 (1984) 3–22.

[5] K. Culik II, J. Gruska, A. Salomaa, Systolic trellis automata: stability, decidability and complexity, Inform. and Control 71 (1986) 218–230.

[6] C. Dyer, One-way bounded cellular automata, Inform. and Control 44 (1980) 261–281.

[7] H. Fernau, Nonterminal complexity of programmed grammars, Theoret. Comput. Sci. 296 (2) (2003) 225–251.
[8] S.A. Greibach, The undecidability of the ambiguity problem for minimal linear grammars, Inform. and Control 6 (2) (1963) 119–125.
[9] J. Gruska, Descriptional complexity of context-free languages, Proc. MFCS, 1973, pp. 71–83.
[10] L.A. Haines, Note on the complement of a (minimal) linear language, Inform. and Control 7 (1964) 307–314.
[11] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.
[12] J. Hartmanis, On the succinctness of different representations of languages, SIAM J. Comput. 9 (1980) 114–120.
[13] O.H. Ibarra, S.M. Kim, Characterizations and computational complexity of systolic trellis automata, Theoret. Comput. Sci. 29 (1984) 123–153.
[14] O.H. Ibarra, S.M. Kim, S. Moran, Sequential machine characterizations of trellis and cellular automata and applications, SIAM J. Comput. 14 (2) (1985) 426–447.
[15] A. Malcher, Descriptional complexity of cellular automata and decidability questions, J. Automat. Languages Combin. 7 (4) (2002) 549–560.
[16] A. Okhotin, Conjunctive grammars, J. Automat. Languages Combin. 6 (4) (2001) 519–535.
[17] A. Okhotin, Conjunctive grammars and systems of language equations, Programming Comput. Software 28 (2002) 243–249.
[18] A. Okhotin, On the closure properties of linear conjunctive languages, Theoret. Comput. Sci. 299 (2003) 663–685.
[19] A. Okhotin, A recognition and parsing algorithm for arbitrary conjunctive grammars, Theoret. Comput. Sci. 302 (2003) 365–399.
[20] A. Okhotin, The hardest linear conjunctive language, Inform. Process. Lett. 86 (5) (2003) 247–253.
[21] A. Okhotin, Boolean grammars, Developments in Language Theory, Proceedings of DLT 2003, Szeged, Hungary, July 7–11, 2003, Lecture Notes in Computer Science, Vol. 2710, Springer, Berlin, pp. 398–410.
[22] A. Okhotin, State complexity of linear conjunctive languages, J. Automat. Languages Combin. 9 (2004) to appear.
[23] A. Okhotin, On the equivalence of linear conjunctive grammars to trellis automata, Inform. Théorique Appl. 38 (2004) 69–88.
[24] A.R. Smith III, Real-time language recognition by one-dimensional cellular automata, J. Comput. System Sci. 6 (1972) 233–252.
[25] V. Terrier, On real-time one-way cellular array, Theoret. Comput. Sci. 141 (1995) 331–335.
[26] D. Wotschke, The Boolean closures of deterministic and nondeterministic context-free languages, in: W. Brauer (Ed.), Gesellschaft für Informatik e., Vol. 3, Jahrestagung 1973, Lecture Notes in Computer Science, Vol. 1, Springer, Berlin, 1973, pp. 113–121.