On a new family of automata

Alexander Okhotin
okhotin@cs.queensu.ca

Technical report 2002–456
Department of Computing and Information Science,
Queen's University, Kingston, Ontario, Canada K7L 3N6

March 2002

Abstract

This paper introduces a new family of automata that turns out to be computationally equivalent to linear conjunctive grammars.

Although, if viewed as a practical tool for describing practical languages, these automata are not nearly as convenient as linear conjunctive grammars are, they can be reasonably expected to be highly suitable for use in applications as a low-level computational model, due to low computational complexity and the extreme simplicity of their implementation.

1 Introduction

The main properties of linear context-free grammars and the languages they generate have been uncovered already in the early days of formal language theory [1]. One of the most attractive properties of these generative devices is their low computational complexity: the general membership problem is known to be **NLOGSPACE**-complete [6], and for every linear context-free language there exists a square-time and linear-space Turing machine that accepts it.

The latter recognition algorithm uses dynamic programming method to compute the collection of sets $\{T_{ij}\}_{1 \leqslant i \leqslant j \leqslant n}$ (where n is the length of the input string), such that each T_{ij} is the set of nonterminals that generate the substring of the input string from the i-th symbol to the j-th. Since every T_{ij} completely depends on $T_{i,j-1}$, $T_{i+1,j}$, the i-th and the j-th symbols of the input string, this leads to $O(n^2)$ time complexity. On the other hand, it not hard to note that if we arrange the sets T_{ij} in lines of the form $T_{1,1+k} \dots T_{n-k,n}$ $(0 \leqslant k < n)$, then only the $(k-1)$-th line is needed to compute each k-th line, and thus the algorithm can store only two lines at a time, making the space requirements linear.

This algorithm is known to have a simple generalization for the case of linear conjunctive grammars [2], in which all rules are of the form $A \to$

$u_1B_1v_1\&\ldots\&u_mB_mv_m$ (where $m \geqslant 1$, B_i are nonterminals and u_i, v_i are terminal strings) or $A \to w$ (where w is a terminal string). These grammars are known to be effectively transformable to a normal form similar to the context-free linear normal form, for which the mentioned sets T_{ij} can be computed with the same simplicity as in the context-free case, because each of them is fully dependent on the same four entities.

Aiming to generalize and refine this kind of computation, the present paper introduces a new family of automata, which operate by first converting a given string into a string of states using a definite mapping from the input alphabet Σ to the set of states Q, and then transforming these strings to shorter strings using another function $\delta : Q \times Q \to Q$, so that at each step of computation a string of states $q_1 \ldots q_m$ gets converted to the string $\delta(q_1, q_2)\delta(q_2, q_3)\ldots\delta(q_{m-1}, q_m)$, which is one symbol shorter. This is being done until the string shrinks to a single state, which will determine whether it is accepted.

The main motivation for the study of these automata is given by their computational equivalence to linear conjunctive grammars: in this paper we show that while these automata can simulate both linear context-free and linear conjunctive grammars, they can in turn be simulated by linear conjunctive grammars as well.

2 Definition

Let us introduce the family of automata we are going to study in this paper.

Definition 1. *An automaton is a quintuple $M = (\Sigma, Q, I, \delta, F)$, where*

- *Σ is the input alphabet.*
- *Q is a finite nonempty set of states.*
- *$I : \Sigma \to Q$ is a function that sets the initial states.*
- *$\delta : Q \times Q \to Q$ is a binary operator on the set Q that works as the transition function.*
- *$F \subseteq Q$ is the set of final states.*

An automaton $M = (\Sigma, Q, I, \delta, F)$ takes a nonempty string $w = a_1 \ldots a_n$ ($a_i \in \Sigma$, $n \geqslant 1$) as an input and then carries out the following computation: first, w is converted to the string of states $I(a_1)\ldots I(a_n)$, as shown in Figure 1(a); then, new strings of states are constructed out of existing strings of states by replacing a string of the form $q_1 \ldots q_m$ with the string $\delta(q_1, q_2), \delta(q_2, q_3), \ldots, \delta(q_{m-1}, q_m)$ (see Figure 1(b)). This is being done until the string of states shrinks to a single state; then the string is accepted if and only if this single state belongs to the set F.

Now let us formally define the computation of an automaton.

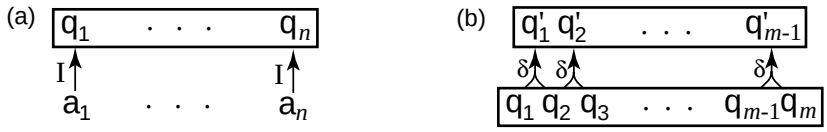

Figure 1: One step of computation.

Definition 2. *An instantaneous description of an automaton $(\Sigma, Q, I, \delta, F)$ is an arbitrary nonempty string over Q.*

The successor of an instantaneous description $q_1 q_2 \dots q_n$ (where $n \geqslant 2$), denoted as $\overline{\delta}(q_1 q_2 \dots q_n)$, is the instantaneous description $q'_1 q'_2 \dots q'_{n-1}$, such that $q'_i = \delta(q_i, q_{i+1})$ for all i.

An instantaneous description is said to be accepting if it consists of a single state from F. An instantaneous description is said to be rejecting if it consists of a single state not in F.

A sequence of instantaneous descriptions $\alpha_1, \dots, \alpha_n$ ($n \geqslant 1$) is called a computation of the automaton if $\alpha_{i+1} = \overline{\delta}(\alpha_i)$ for all i ($1 \leqslant i < n$) and α_n is either accepting or rejecting (i.e. $|\alpha_n| = 1$ and in general $|\alpha_i| = n - i + 1$). The computation is said to be either accepting or rejecting depending on its final instantaneous description.

Note that the computation is deterministic, i.e. every next instantaneous description is completely determined its predecessor. Therefore, the last instantaneous description is completely determined by the first instantaneous description, and thus a computation starting from any given nonempty string of states has a definite outcome, which we denote as a mapping $\Delta : Q^+ \to Q$:

Definition 3. *For each instantaneous description $q_1 q_2 \dots q_n$, denote the final instantaneous description of the computation starting from $q_1 q_2 \dots q_n$ as $\Delta(q_1 q_2 \dots q_n)$.*

Observation 1. *The following holds:*

- $\Delta(q) = q$.
- $\Delta(q_1 \dots q_n) = \overline{\delta}^{n-1}(q_1 \dots q_n)$.
- *For all* $0 \leqslant m < n$,

$$\overline{\delta}^m(q_1 q_2 \dots q_n) = \Delta(q_1 \dots q_{1+m})\Delta(q_2 \dots q_{2+m}) \dots \Delta(q_{n-m} \dots q_n) \quad (1)$$

- $\Delta(q_1 \dots q_n) = \delta(\Delta(q_1 \dots q_{n-1}), \Delta(q_2 \dots q_n))$.

It is left to define the initial instantaneous description of the automaton on the given input string:

Definition 4. *Let $A = (\Sigma, Q, I, \delta, F)$ be an automaton. For each string $w \in \Sigma^+$, define $I(w) = I(a_1)I(a_2) \dots I(a_n)$.*

The computation of the automaton A on the string w is the computation starting from the instantaneous description $I(w)$. The automaton is said to accept the string w if the computation starting from $I(w)$ is an accepting computation, i.e. if $\Delta(I(w)) \in F$; otherwise, the automaton is said to reject the string. The set of strings accepted by the automaton is called the language generated by the automaton:

$$L(A) = \{w \mid \Delta(I(w)) \in F\} \tag{2}$$

One evident limitation of the new automata is their inability to accept or reject the empty string; however, this is only a technical limitation which does not affect their generative power on longer strings.

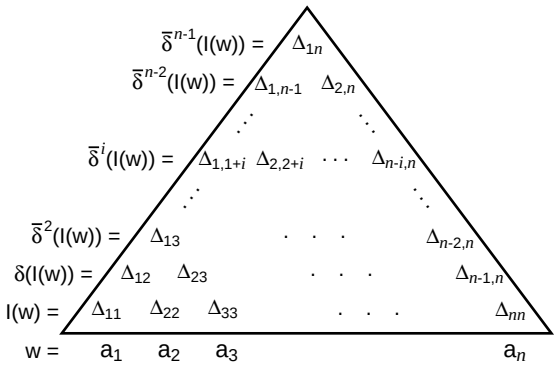

Figure 2: The computation of an automaton on a given input string.

The general form of a computation of an automaton is given in Figure 2, where a short notation Δ_{ij} for $\Delta(I(a_i \dots a_j))$ is adopted.

3 Linear conjunctive grammars and automata

In this section we provide a short overview of linear conjunctive grammars, consequently proving them to be computationally equivalent to the newly introduced automata.

3.1 Preliminaries

Let us start with a formal definition of conjunctive grammars [2]:

Definition 5. *A conjunctive grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols; P is a finite set of grammar rules of the form*

$$A \to \alpha_1 \& \dots \& \alpha_n \quad (A \in N,\ n \geqslant 1,\ \text{for all } i\ \alpha_i \in (\Sigma \cup N)^*), \tag{3}$$

where the strings α_i are distinct, and their order is considered insignificant; $S \in N$ is a nonterminal designated as the start symbol.

Three additional special symbols will be used: '(', '&' and ')'; it is assumed that none of them is in $\Sigma \cup V$.

For each rule of the form (3) and for each i ($1 \leqslant i \leqslant n$), $A \to \alpha_i$ is called a conjunct. Let $conjuncts(P)$ denote the sets of all conjuncts.

A conjunctive grammar generates strings by deriving them from the start symbol, generally in the same way as the context-free grammars do. Intermediate strings used in course of a derivation are actually formulae under the basis of concatenation and conjunction:

Definition 6. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. The set of conjunctive formulae $\mathcal{F}$ is defined inductively:*

- *The empty string ϵ is a conjunctive formula.*
- *Any symbol from $\Sigma \cup N$ is a formula.*
- *If $\mathcal{A}$ and $\mathcal{B}$ are nonempty formulae, then $\mathcal{AB}$ is a formula.*
- *If $\mathcal{A}_1, \ldots, \mathcal{A}_n$ ($n \geqslant 1$) are formulae, then $(\mathcal{A}_1 \& \ldots \& \mathcal{A}_n)$ is a formula.*

Definition 7. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Define $\overset{G}{\Longrightarrow}$, the relation of one-step derivability on the set of conjunctive formulae.*

1. *For any $s', s'' \in \Sigma \cup N \cup \{\text{'('}, \text{'\&'}, \text{')'}\}^*$ and $A \in N$, such that $s'As''$ is a formula, and for all $A \to \alpha_1 \& \ldots \& \alpha_n \in P$,*

$$s'As'' \overset{G}{\Longrightarrow} s'(\alpha_1 \& \ldots \& \alpha_n)s'' \tag{4}$$

2. *(the gluing rule) For any $s', s'' \in \Sigma \cup N \cup \{\text{'('}, \text{'\&'}, \text{')'}\}^*$, $n \geqslant 1$ and $w \in \Sigma^*$, such that $s'(\underbrace{w \& \ldots \& w}_{n \geqslant 1})s''$ is a formula,*

$$s'(\underbrace{w \& \ldots \& w}_{n})s'' \overset{G}{\Longrightarrow} s'ws'' \tag{5}$$

Let $\overset{G}{\Longrightarrow}{}^*$ denote the reflexive and transitive closure of $\overset{G}{\Longrightarrow}$.

Definition 8. *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: $L_G(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \overset{G}{\Longrightarrow}{}^* w\}$. The language generated by the grammar is the language generated by its start symbol: $L(G) = L_G(S)$.*

The semantics of conjunctive grammars is well characterized by the following equalities [2]:

Theorem 1. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ be formulae, let $A \in N$, let $a \in \Sigma$. Then,*

$$L_G(\epsilon) = \{\epsilon\} \tag{6a}$$

$$L_G(a) = \{a\} \tag{6b}$$

$$L_G(A) = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in P} L_G((\alpha_1 \& \ldots \& \alpha_m)) \tag{6c}$$

$$L_G(\mathcal{A}\mathcal{B}) = L_G(\mathcal{A}) \cdot L_G(\mathcal{B}) \tag{6d}$$

$$L_G((\mathcal{A}_1 \& \ldots \& \mathcal{A}_n)) = \bigcap_{i=1}^{n} L_G(\mathcal{A}_i) \tag{6e}$$

Let us now restrict general conjunctive grammars to obtain the subclass of linear conjunctive grammars, which will play an important role throughout this paper:

Definition 9. *A conjunctive grammar $G = (\Sigma, N, P, S)$ is said to be linear, if each rule in P is of the form*

$$A \to u_1 B_1 v_1 \& \ldots \& u_m B_m v_m \quad (u_i, v_i \in \Sigma^*,\ B_i \in N) \tag{7a}$$

$$A \to w \quad (w \in \Sigma^*) \tag{7b}$$

It has been proved in [2] that every linear conjunctive grammar can be effectively transformed to an equivalent grammar in the so-called linear normal form:

Definition 10. *A linear conjunctive grammar $G = (\Sigma, N, P, S)$ is said to be in the linear normal form, if each rule in P is of the form*

$$A \to bB_1 \& \ldots \& bB_m \& C_1 c \& \ldots \& C_n c \quad (m + n \geqslant 1;\ A, B_i, C_j \in N;\ b, c \in \Sigma), \tag{8a}$$

$$A \to a \quad (A \in N, a \in \Sigma), \tag{8b}$$

$$S \to \epsilon, \quad \textit{only if } S \textit{ does not appear in right parts of rules} \tag{8c}$$

Linear conjunctive grammars are known to be able to generate the classical non-context-free languages $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{a^m b^n c^m d^n \mid m, n \geqslant 0\}$ and $\{wcw \,|\, w \in \{a, b\}^*\}$ [2], as well as more complex languages, such as the language $\{ba^2ba^4b \ldots ba^{2n-2}ba^{2n}b \,|\, n \geqslant 0\}$ with square growth property and the language of all derivations within a finite string-rewriting system [5].

Despite the increased generative power in comparison with linear context-free grammars, any language generated by a linear conjunctive grammar is, as we shall now demonstrate, still a square-time language.

3.2 Recognition algorithm for linear conjunctive grammars

Efficient recognition and parsing was originally one of the main reasons for the study of conjunctive grammars [2], and the subclass of linear conjunctive

grammars is characterized by even better upper bound for complexity than that of general conjunctive grammars. The original paper [2] presents a quadratic-time and linear-space algorithm for the grammars in the linear normal form; the paper [3] defines a more practical parsing algorithm for arbitrary conjunctive grammars, which is generally cubic-time, but is known to use no more than $O(n^2)$ time and $O(n)$ space on any linear conjunctive grammar.

Let us review the simpler algorithm of [2]. For a given input string $w = a_1 \dots a_n \in \Sigma^+$ $(n \geqslant 1)$, define $T_{ij} = \{A \mid A \in N,\ A \stackrel{G}{\Longrightarrow}{}^* a_i \dots a_j\}$ for all i and j $(1 \leqslant i \leqslant j \leqslant n)$. The sets T_{ii} can be computed immediately:

$$T_{ii} = \{A \mid A \in N,\ A \to a_i \in P\} \tag{9a}$$

In order to compute the sets T_{ij} for $i < j$, it suffices to note [2] that

$$\begin{aligned} T_{ij} = \{A \mid A \in N,\ &\text{there is a rule } A \to bB_1\& \dots \&bB_m\&C_1c\& \dots \&C_nc \in P, \\ &\text{such that } b = a_i,\ c = a_j,\ \text{for all } p\,(1 \leqslant p \leqslant m)\ B_p \in T_{i+1,j} \text{ and} \\ &\text{for all } q\,(1 \leqslant q \leqslant n)\ C_q \in T_{i,j-1}\}, \end{aligned} \tag{9b}$$

and that the computation of (9b) takes constant time. This yields the following square-time algorithm [2]:

Algorithm 1. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar in the linear normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ $(n \geqslant 1)$ be the string being recognized.*

for $i = 1$ *to* n
    $T_{ii} = \{A \mid A \in N,\ A \to a_i \in P\}$
for $k = 1$ *to* $n - 1$
    for $i = 1$ *to* $n - k$
    {
        let $j = k + i$
        $T_{ij} =$ (as in (9b))
    }
if $S \in T_{ij}$*, then return "yes", else return "no".*

It is easily seen that each iteration of the outer loop depends only on the input string and on the products of the previous iteration. That allows us to discard earlier portions of the matrix in course of the computation and thus use only $O(n)$ space.

This implies that every linear conjunctive language is deterministic context-sensitive; this inclusion was also shown to be proper by investigating the unary case, where the generative capacity of linear conjunctive grammars does not exceed that of finite automata [5].

3.3 Simulation of linear conjunctive grammars by automata

Let us show that our automata can accept any language generated by a linear conjunctive grammar by just simulating the computation of Algorithm 1.

Each state of the automaton will be associated with some subset of N, so that the state $\Delta(I(a_i \dots a_j))$ "know" the set T_{ij}. In order to eliminate the direct dependence of every step of computation on the input string, the characters of the input string will also be encoded in the states, so that each state $\Delta(I(a_i \dots a_j))$ will remember the symbols a_i and a_j.

The first loop of the algorithm will be done by the function I on the basis of (9a). The assignment (9b) to every T_{ij} $(i < j)$ will be modeled by the function δ; thus, each iteration of the second loop (the outer) of the algorithm will be simulated by a single application of $\overline{\delta}$. The acceptance condition in the last statement of the algorithm will be checked with the help of the set F.

Formally, let $G = (\Sigma, N, P, S)$ be a linear conjunctive grammar in the linear normal form. We construct the automaton $M = M(G) = (\Sigma, Q, I, \delta, F)$, where

$$Q = \Sigma \times 2^N \times \Sigma, \tag{10a}$$

$$I(a) = (a, \{A \mid A \to a \in P\}, a), \tag{10b}$$

$$\begin{aligned} &\delta((b, Q, b'), (c', R, c)) = \\ &(b, \{A \mid \exists A \to bB_1 \& \dots bB_m \& C_1 c \& \dots \& C_n c : \ B_i \in R, C_j \in Q\}, c), \end{aligned} \tag{10c}$$

$$F = \{(a, R, b) \mid R \subseteq N, \ S \in X, \ a, b \in \Sigma\} \tag{10d}$$

The following statement can be established by a trivial induction on the length of the string:

Observation 2. *For each string $w \in \Sigma^+$, if $\Delta(I(w)) = (b, R, c)$, then the first symbol of w is b and the last symbol of w is c.*

The correctness of our construction is proved in the following lemma:

Lemma 1. *For each string $w \in \Sigma^+$, let $\Delta(I(w)) = (b, R, c)$.*

Then, for each nonterminal $A \in Q$, $A \overset{G}{\Longrightarrow}{}^* w$ if and only if $A \in R$.

Proof. Induction on $|w|$.

Basis $|w| = 1$**.** Let $w = a \in \Sigma$. Then, $\Delta(I(w)) = (a, R = \{A \mid A \to a \in P\}, a)$, and $A \in R$ if and only if $A \to a \in P$ if and only if $a \in L(A)$.

Induction step. Let $w = bxc$ $(b, c \in \Sigma, \ x \in \Sigma^*)$. Then, by Observation 2, $\Delta(I(bxc)) = (b, R, c)$ for some $R \subseteq N$; by the same reasoning, $\Delta(I(bx)) = (b, Q_1, b')$, where b' is the last symbol of bx and $Q_1 \subseteq N$, and $\Delta(I(xc)) = (c', Q_2, c)$, where c' is the first symbol of xc and $Q_2 \subseteq N$. By Observation 1,

$$\delta((b, Q_1, b'), (c', Q_2, c)) = (b, R, c) \tag{11}$$

$\Rightarrow$ Let $A \in N$ be some nonterminal. The nonterminal A generates bxc if and only if there exists a rule of the form

$$A \to bB_1 \& \dots bB_m \& C_1 c \& \dots \& C_n c \in P, \tag{12}$$

such that $xc \in L(B_i)$ and $bx \in L(C_j)$ for all i, j.

By induction hypothesis, this means that $B_i \in Q_2$ and $C_j \in Q_1$ for all i, j, which, by the rule (12) and by the construction of δ, implies that $A \in R$.

$\Leftarrow$ If $A \in R$, then there exists some rule of the form (12), where all B_i are all in Q_2 and all C_j are in Q_1. By induction hypothesis, $B_i \Longrightarrow^* xc$ and $C_i \Longrightarrow^* bx$, which allows to construct a derivation of the string w from the nonterminal A. □

Theorem 2. *For every linear conjunctive grammar $G = (\Sigma, N, P, S)$ there exists and can be effectively constructed an automaton $M = (\Sigma, Q, I, \delta, F)$, such that $L(M) = L(G) \pmod{\Sigma^+}$.*

3.4 Simulation of automata by linear conjunctive grammars

It was proved in Section 3.3 that our automata are at least as powerful as linear conjunctive grammars. In this section we show that, on the other hand, their generative power does not exceed that of linear conjunctive grammars, because linear conjunctive grammars can simulate the computation of automata. Together, these two results will show that the automata are computationally equivalent to linear conjunctive grammars.

Let $M = (\Sigma, Q, I, \delta, F)$ be an automaton. We construct the grammar $G = G(M) = (\Sigma, N_Q \cup \{S\}, P, S)$, where $N_Q = \{A_q \mid q \in Q\}$ and P contains the following rules:

$$S \to A_q \quad (\text{for all } q \in F) \tag{13a}$$

$$A_{I(a)} \to a \quad (\text{for all } a \in \Sigma) \tag{13b}$$

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

Lemma 2. *For each string $w \in \Sigma^+$ and for each state $q \in Q$, $A_q \overset{G}{\Longrightarrow}{}^* w$ if and only if $\Delta(I(w)) = q$.*

Proof. Induction on $|w|$.

Basis $|w| = 1$**.** Let $w = a \in \Sigma$. $A_q \overset{G}{\Longrightarrow}{}^* a$ iff $A_q \to a \in P$ iff $q = I(a)$ iff $q = \Delta(I(a))$.

Induction step. Let $w = bxc$ ($b, c \in \Sigma$, $x \in \Sigma^*$).

$\Leftarrow$ If $A_q \overset{G}{\Longrightarrow}{}^* bxc$, then there exists a rule

$$A_q \to bA_{q_2}\&A_{q_1}c \in P \tag{14a}$$

such that

$$A_{q_2} \Longrightarrow^* xc, \tag{14b}$$

$$A_{q_1} \Longrightarrow^* bx, \tag{14c}$$

where, by the construction of P (13c), the rule (14a) must have $q = \delta(q_1, q_2)$; by induction hypothesis, (14b) holds if and only if $q_2 = \Delta(I(xc))$ and (14c) holds if and only if $q_1 = \Delta(I(bx))$. Therefore, $q = \Delta(I(bx), I(xc)) = \Delta(I(bxc))$.

$\Leftarrow$ Let $q_1 = \Delta(I(bx))$, $q_2 = \Delta(I(xc))$. By Observation 1, $\delta(q_1, q_2) = \Delta(I(bxc)) = q$. By induction hypothesis, this implies $A_{q_1} \Longrightarrow^* bx$, $A_{q_2} \Longrightarrow^* xc$, and, by the rule (13c), $A_q = A_{\delta(q_1,q_2)} \Longrightarrow^* bxc$. □

Corollary 1. *For each automaton M, $L(M) = L(G(M))$.*

Proof. $L(M) = L(G(M))$ (mod Σ^+) follows from Lemma 2. Since $\epsilon \notin L(M)$ and $\epsilon \notin L(G(M))$, the languages of the automaton and the grammar coincide completely. □

Theorem 3. *For every automaton $M = (\Sigma, Q, I, \delta, F)$, there exists and can be effectively constructed a linear conjunctive grammar $G = (\Sigma, N, P, S)$, such that $L(G) = L(M)$.*

Together with the earlier Theorem 2, this allows to make the following conclusion:

Theorem 4. *A language $L \subseteq \Sigma^+$ is accepted by some automaton if and and only if it is generated by some linear conjunctive grammar.*

3.5 Shortened linear normal form

Besides giving the computational equivalence of linear conjunctive grammars and newly introduced automata, the combination of Theorems 2 and 3 leads us to one more noteworthy result related to linear conjunctive grammars alone.

It turns out that the linear normal form of Definition 10, originally introduced in [2], is not optimal in the sense that it can be further restricted, at the same time preserving the generative power.

Definition 11. *A linear conjunctive grammar $G = (\Sigma, N, P, S)$ is said to be in the **shortened linear normal form**, if each rule in P is of the form*

$$A \to bB\&Cc \quad (A, B, C \in N,\ b, c \in \Sigma), \tag{15a}$$

$$A \to a \quad (A \in N, a \in \Sigma), \tag{15b}$$

$$S \to \epsilon, \quad \text{only if } S \text{ does not appear in right parts of rules} \tag{15c}$$

Theorem 5. *For every linear conjunctive grammar there exists and can be effectively constructed an equivalent linear conjunctive grammar in the shortened linear normal form.*

Proof. Let $G_1 = (\Sigma, N_1, P_1, S_1)$ be an arbitrary linear conjunctive grammar. We use the transformations of Section 3.3 to obtain the automaton $M(G_1)$, and then apply the transformation given in Section 3.4 to get the grammar $G_2 = G(M(G_1)) = (\Sigma, N_2, P_2, S_2)$.

It follows from Theorems 2 and 3 that the languages $L(G_1)$ and $L(G_2)$ are equivalent modulo Σ^+. Additionally, by the construction from Theorem 3, the grammar G_2 is "almost" in shortened linear normal form, with the only exception of the rules of the form (13a) for the dedicated start symbol S_2.

In order to bring G_2 to conformity with Definition 11, we apply the transformation called *substitution of unit conjuncts* [2], replacing every rule $S_2 \to A$ of type (13a) with a collection of rules

$$S_2 \to \alpha_1 \& \ldots \& \alpha_m \quad (m = 1, 2;\ A \to \alpha_1 \& \ldots \& \alpha_m \in P_2) \tag{16}$$

Finally, if $\epsilon \in L(G_1)$, then the rule $S_2 \to \epsilon$ should be added to the resulting grammar, making it completely equivalent to G_2. □

4 Example

Let us consider the following linear conjunctive grammar for the language $\{a^n b^n a^n \mid n \geqslant 0\}$:

$$\begin{aligned}
&S \to Ka \& aR \mid \epsilon \\
&K \to aA \mid Ka \\
&P \to aA \\
&A \to Pb \mid b \\
&R \to Ba \mid aR \\
&Q \to Ba \\
&B \to bQ \mid b
\end{aligned}$$

The grammar is in the linear normal form. The cardinality of the set $\Sigma \times 2^N \times \Sigma$ is $2 \cdot 128 \cdot 2 = 512$, but in fact only fourteen out of these will be potentially reachable from the initial state.

These reachable states are enumerated in Table 1.

0	$(a,\ \emptyset,\ a)$
1	$(a,\ \emptyset,\ b)$
2	$(a,\ \{S, K, R\},\ a)$
3	$(a,\ \{K\},\ a)$
4	$(a,\ \{K, P\},\ b)$
5	$(a,\ \{K, P, A\},\ b)$
6	$(a,\ \{A\},\ b)$
7	$(a,\ \{R\},\ a)$
8	$(b,\ \emptyset,\ a)$
9	$(b,\ \emptyset,\ b)$
10	$(b,\ \{A, B\},\ b)$
11	$(b,\ \{R, Q\},\ a)$
12	$(b,\ \{R, Q, B\},\ a)$
13	$(b,\ \{B\},\ a)$

Table 1: The states of the automaton.

The initial function I and the set of final states F are shown in Table 2.

a	b
0	10

0	1	2	3	4	5	6	7	8	9	10	11	12	13
−	−	+	−	−	−	−	−	−	−	−	−	−	−

Table 2: The function I and the membership of states in F.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	1	7	0	1	4	4	7	0	1	4	7	7	0
1	0	1	7	0	1	4	4	7	0	1	4	7	7	0
2	3	1	2	3	1	4	4	2	3	1	4	2	2	3
3	3	1	2	3	1	4	4	2	3	1	4	2	2	3
4	3	6	2	3	6	5	5	2	3	6	5	2	2	3
5	3	6	2	3	6	5	5	2	3	6	5	2	2	3
6	0	1	7	0	1	4	4	7	0	1	4	7	7	0
7	0	1	7	0	1	4	4	7	0	1	4	7	7	0
8	8	9	8	8	9	9	9	8	8	9	9	13	13	8
9	8	9	8	8	9	9	9	8	8	9	9	13	13	8
10	11	9	11	11	9	9	9	11	11	9	9	12	12	11
11	8	9	8	8	9	9	9	8	8	9	9	13	13	8
12	11	9	11	11	9	9	9	11	11	9	9	12	12	11
13	11	9	11	11	9	9	9	11	11	9	9	12	12	11

Table 3: The function δ.

The transition function $\delta : Q \times Q \to Q$ is given in Table 3.

Finally, a sample computation of this automaton on the input string $w =$ *aaaaabbbbbaaaaa* is presented in Table 4.

5 Concluding remarks

We have introduced a new family of relatively simple computing devices that turned out to be equivalent to linear conjunctive grammars.

The relationship between linear conjunctive grammars and the automata introduced in this paper resembles that between regular expressions and finite automata: while grammars and regular expressions are usually more convenient for humans, the automata are much better suitable for machine implementation. Thus it is reasonable to expect that the newly introduced automata will prove useful as a low-level model in practical language-processing applications.

The techniques studied in this paper have been implemented in the parser generator [7]; it could also be mentioned that all the tables given in this paper were constructed automatically (in TEX format) and simply included in this document using `\input` directive.

							2								
						3		7							
					3		0		7						
				3		0		0		7					
			3		0		0		0		7				
		4		0		0		0		0		11			
	1		6		0		0		0		13		8		
		1		4		1		0		0		8		11		8	
			1		1		6		1		0		8		13		8		8	
				1		1		4		1		1		8		8		11		8		8	
			0		1		1		6		1		9		8		13		8		8		0	
		0		0		1		4		1		9		9		8		11		8		0		0	
	0		0		0		1		6		9		9		9		13		8		0		0		0	
0		0		0		0		4		9		9		9		9		11		0		0		0		0
0	0		0		0		0		10		10		10		10		10		0		0		0		0	
a	a	a	a	a	b	b	b	b	b	a	a	a	a	a

Table 4: Computation of the automaton on the string $w = aaaaabbbbbaaaaa$.

References

[1] M. A. Harrison, *Introduction to formal language theory*, Addison-Wesley, Reading, Mass., 1978.

[2] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), to appear.

[3] A. Okhotin, "A recognition and parsing algorithm for arbitrary conjunctive grammars", submitted to *Theoretical Computer Science*.

[4] A. Okhotin, "Linear conjunctive languages are closed under complement", Technical Report 2002–455, Department of Computing and Information Science of Queen's University, Kingston, Ontario, Canada.

[5] A. Okhotin, "On the closure properties of linear conjunctive languages", submitted to *Theoretical Computer Science*.

[6] I. H. Sudborough, "A note on tape-bounded complexity classes and linear context-free languages", *Journal of the ACM*, 22:4 (1975), 499–500.

[7] Whale Calf, a parser generator for conjunctive grammars, available at `http://www.cs.queensu.ca/home/okhotin/whalecalf/`