Theoretical Informatics and Applications
RAIRO: Theoret. Informatics Appl. **38** (2004) 69–88
DOI: 10.1051/ita:2004004

ON THE EQUIVALENCE OF LINEAR CONJUNCTIVE GRAMMARS AND TRELLIS AUTOMATA *

Alexander Okhotin[1]

Abstract. This paper establishes computational equivalence of two seemingly unrelated concepts: linear conjunctive grammars and trellis automata. Trellis automata, also studied under the name of one-way real-time cellular automata, have been known since early 1980s as a purely abstract model of parallel computers, while linear conjunctive grammars, introduced a few years ago, are linear context-free grammars extended with an explicit intersection operation. Their equivalence implies the equivalence of several other formal systems, including a certain restricted class of Turing machines and a certain type of language equations, thus giving further evidence for the importance of the language family they all generate.

Mathematics Subject Classification. 68Q01, 68Q42, 68Q70, 68Q80.

1. Introduction

The family of languages studied in this paper has been known for a couple of decades. It is located strictly between linear context-free and deterministic context-sensitive languages. It is closed under all set-theoretic operations, but not under concatenation. It is incomparable with the context-free languages. All its languages are square-time, but among them there are some important non-context-free ones, such as $\{a^n b^n c^n \,|\, n \geqslant 0\}$, $\{a^m b^n c^m d^n \,|\, m, n \geqslant 0\}$, $\{wcw \,|\, w \in \{a, b\}^*\}$, the

Keywords and phrases. Conjunctive grammar, trellis automaton, cellular automaton, language equation, Turing machine.

* *This paper is an almost entirely rewritten version of the paper "Automaton representation of linear conjunctive languages" presented at the conference DLT 2002 (Developments in Language Theory) held in Kyoto, Japan, September 18–21, 2002.*

[1] School of Computing, Queen's University, Kingston, Ontario, Canada;
e-mail: okhotin@cs.queensu.ca

language of accepting computations of any Turing machine and some **P**-complete languages.

The first mention of this family in the literature was in the paper [6], which, following [14, 15], studied restricted types of one-dimensional cellular automata as language acceptors, among them *one-way real-time cellular automata*.

At approximately the same time, but with different motivation behind, a very similar concept was born. These were systolic trellis automata, introduced in [4] as a model of a massively parallel computer with simple identical processors connected in a uniform pattern. They are used as acceptors of strings loaded into a designated row of processors, and the output of some other designated processor determines the acceptance. Their formal properties were further studied in [5]. Their simplest type, *homogeneous trellis automata* (nicknamed triangular [8] for the structure of the trellis) were proved to be equivalent to one-way real-time cellular automata in [2].

Both trellis automata and cellular automata of all kinds are essentially parallel computing devices. Their sequential characterization was given in [7], where *Turing machines with a certain heavy restriction imposed on them* were considered, and this restriction made them computationally equivalent to trellis automata. This was followed by a further study of the properties of the family they generate [7]; among the problems left open was its closure under concatenation, which was eventually solved negatively [16].

Conjunctive grammars, the second subject of this paper, were introduced in [9] as an extension of context-free grammars with an explicit intersection operation, motivated by the need of a more expressive, but still computationally efficient formalism for specifying languages. These goals were generally achieved, as conjunctive grammars inherit all the descriptive means offered by context-free grammars and add a new operation that has intuitively understandable semantics, while, on the other hand, they possess almost the same recognition and parsing algorithms as the context-free grammars do [12]. *Linear conjunctive grammars* [9] are a subclass of conjunctive grammars obtained by restricting the use of concatenation in the same way as used in the definition of linear context-free grammars; alternatively, they can be viewed as an extension of linear context-free grammars with an intersection operation. A study of their theoretical properties was undertaken in [11].

Just like context-free grammars are known to have an algebraic characterization by language equations with union and concatenation, so are the conjunctive grammars. It has been proved in [10] that if the operation of intersection is allowed in the classical language equations of [1, 3] (which contain the semiring operations of union and concatenation), then the languages defined by components of their least solutions will be exactly the class of languages generated by conjunctive grammars. If at the same time the use of concatenation in the systems is restricted to left- and right-concatenation of terminal symbols – *i.e.*, if one considers *systems of language equations with union, intersection and linear concatenation* – then the class of linear conjunctive grammars is similarly characterized.

In this paper it will be proved that linear conjunctive grammars and trellis automata define the same class of languages, which will imply the computational equivalence of all five formal systems emphasized with italic above. An overview of these concepts is given in Section 2, followed by the main equivalence result in Section 3 and some of its corollaries in Section 4.

2. The formal systems to be proved equivalent

This section gives an overview of all the formalisms that will be proved equivalent as the result of this paper.

2.1. Trellis automata and one-way real time cellular automata

Although trellis automata and cellular automata have different motivation and background, and are intended to model processes of a different kind in a different way, the correspondence between them goes beyond computational equivalence: it turns out [2] that internally they are the same.

In order to emphasize this strong similarity, uniform notation for both types of automata, derived from the notation used for finite automata, will be used in this paper.

Definition 1. One-way real-time cellular automata and trellis automata are defined as quintuples $M = (\Sigma, Q, I, \delta, F)$, where Σ is the input alphabet, Q is a finite nonempty set of states, $I : \Sigma \to Q$ is the initial function, $\delta : Q \times Q \to Q$ is the transition function, and $F \subseteq Q$ is the set of final states.

While syntactically two types of automata have been defined to be identical, the definitions of the way they operate differ.

For a one-way real-time cellular automaton, computation is parallel transformation of nonempty strings over the alphabet Q, where positions in these strings represent cells. The initial function I is extended to a homomorphism $I : \Sigma^* \to Q^*$ that transforms an input $w \in \Sigma^+$ to the initial configuration $I(w)$. The transition function $\delta(q', q'')$ determines the new value of every cell that has current value q' and right neighbour with current value q''. δ is extended to $\hat{\delta}$ that applies δ to every pair of consecutive states, resulting in a new contents of the tape. Given a string w, the state in the leftmost cell after $|w| - 1$ such steps determines the acceptance (whence the title real-time).

Successive configurations of a one-way real-time CA and the dependencies between the contents of the cells at different steps of the computation are shown in Figure 1a.

Definition 2. Let $M = (\Sigma, Q, I, \delta, F)$ be a one-way real-time cellular automaton. Define the initial function $I(a_1 \dots a_n) = I(a_1) \dots I(a_n)$. Define the transition function between configurations $\hat{\delta} : Q^+ \to Q^+$ as follows:

$$\hat{\delta}(q_1 q_2 q_3 \dots q_{n-1} q_n) = (\delta(q_1, q_2), \delta(q_2, q_3), \dots, \delta(q_{n-1}, q_n)). \tag{1}$$

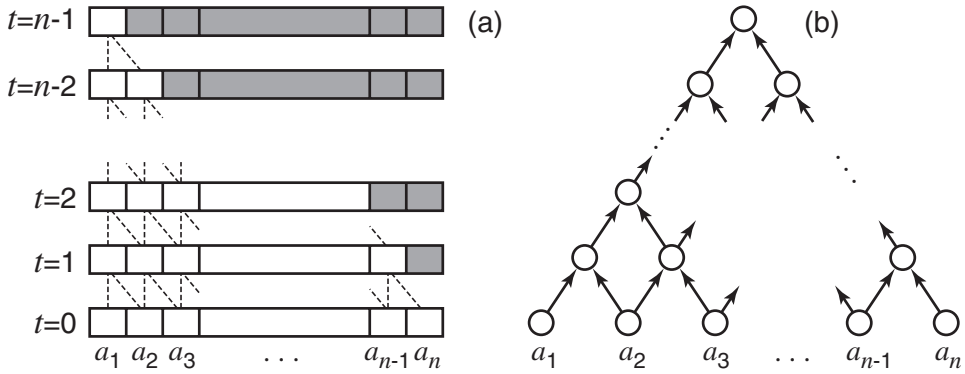

FIGURE 1. (a) one-way real-time CA; (b) homogeneous (triangular) TA.

The computation of M on a string w is a succession of configurations: $I(w)$, $\hat{\delta}(I(w))$, $\hat{\delta}^2(I(w))$, ..., $\hat{\delta}^{|w|-1}(I(w))$. The language accepted by M is now defined as:

$$L(M) = \{w \in \Sigma^+ \mid \hat{\delta}^{|w|-1}(I(w)) \in F\}. \tag{2}$$

For convenience, denote $\Delta(q_1 \dots q_n) = \hat{\delta}^{n-1}(q_1 \dots q_n) \in Q$ for every string of states $q_1 \dots q_n \in Q^+$. Now (2) can be reformulated as

$$L(M) = \{w \in \Sigma^+ \mid \Delta(I(w)) \in F\}. \tag{3}$$

Semantics of trellis automata [4] is defined in a different way. Input strings of length n are loaded into a trellis of $n(n+1)/2$ elementary processing units illustrated in Figure 1b: n designated processors at the bottom receive their symbols and each of them computes its value using I. Each of the rest of the processors has two predecessors in the trellis, and it computes its value by applying δ to what it receives from its predecessors. The value computed by the pinnacle of the trellis determines the acceptance of the input.

Definition 3. Let $M = (\Sigma, Q, I, \delta, F)$ be a trellis automaton. For every string $w = a_1 \dots a_n \in \Sigma^+$, the trellis consists of processors $\{(i,j) \mid 1 \leqslant i \leqslant j \leqslant n\}$. Denote the value in a processor (i,j) as $\Delta_{i..j}[w]$, and define these values as

$$\Delta_{i..i}[a_1 \dots a_n] = I(a_i), \tag{4a}$$

$$\Delta_{i..j}[a_1 \dots a_n] = \delta(\Delta_{i..j-1}[a_1 \dots a_n], \Delta_{i+1..j}[a_1 \dots a_n]) \quad \text{for } i < j. \tag{4b}$$

The language accepted by a trellis automaton $M = (\Sigma, Q, I, \delta, F)$ is

$$L(M) = \{w \in \Sigma^+ \mid \Delta_{1..|w|}[w] \in F\}. \tag{5}$$

Comparing Figures 1a and 1b, it is easy to note that, despite the different interpretation of the values, these are the same $n(n+1)/2$ elements of Q computed

using the same formulae. Formally, the ith cell of the instantaneous description $\hat{\delta}^k(I(a_1 \dots a_n))$ is the same as the value in the processor $\Delta_{i..i+k}[a_1 \dots a_n]$, and both equal $\Delta(I(a_i \dots a_{i+k}))$.

In our notation, the result on the equivalence of trellis automata to one-way real-time cellular automata can be stated as follows:

Theorem 1 (Choffrut and Culik [2])**.** *Definitions 2 and 3 are equivalent.*

If one looks upon Figure 1 and uses the present notation, this result of [2] might seem obvious. But taking into account that even the symmetry of one-way real-time cellular automata was not initially understood [6], and the original motivation and notation for cellular and trellis automata were quite different from each other, it should be admitted that showing them identical was a nontrivial observation.

An evident limitation of both types of automata is their inability to accept or reject the empty string. Although this is only a technical limitation that does not affect their generative power on longer strings, it is essential for defining the language family properly. Since the other devices considered in this paper do not have problems with handling ϵ, it is convenient to assume that trellis automata are also equipped with the means to handle it. Formally this can be achieved by defining them as $M = (\Sigma, Q, I, \delta, F, e)$, where $e \in \{0, 1\}$, and $\epsilon \in L(M)$ if and only if $e = 1$.

2.2. A restricted type of Turing machines

A certain type of sequential computational devices was introduced in [7] in order to characterize trellis automata. These are Turing machines with a one-way read-only input tape and a two-way read-write work tape. The work tape alphabet Γ contains start marker $\#$ and blank symbol Λ. Initially, the one-way infinite work tape contains $\#$ in the leftmost cell and an infinite number of Λs to the right. The movement of the machine's heads is restricted so that it makes alternate left-to-right and right-to-left sweeps over the work tape.

On its way to the right, such a machine is required to remain in state q_0 until it reaches the leftmost Λ symbol; then it reads an input symbol, enters some state other than q_0, rewrites Λ with some symbol from $\Gamma \setminus \{\#, \Lambda\}$, and proceeds to the left. On its way to the left it can do arbitrary rewriting. Once it reaches the start marker, the machine proceeds to the next right sweep, unless the input string has ended; in this case it accepts or rejects depending on the current state.

Formally, these restrictions upon the standard Turing machine can be stated as follows:

Definition 4. [7]. A simple Turing machine is a sextuple $M = (\Sigma, \Gamma, Q, \delta, q_0, F)$, where $\#, \Lambda \in \Gamma$, $\delta : Q \times \Gamma \times (\Sigma \cup \{\epsilon\}) \to Q \times (\Gamma \setminus \{\Lambda\}) \setminus \{-1, +1\}$, $q_0 \in Q$, $F \subseteq Q$.

The following restriction is imposed on δ: for all $q, q' \in Q$, $x, x' \in \Gamma$, $a \in \Sigma \cup \{\epsilon\}$ and $d \in \{-1, +1\}$, if $\delta(q, x, a) = (q', x', d)$, then:

(1) if $q = q_0$ and $x \neq \Lambda$, then $a = \epsilon$, $q' = q_0$, $x' = x$ and $d = +1$: left-to-right sweep without changing the state and the tape;

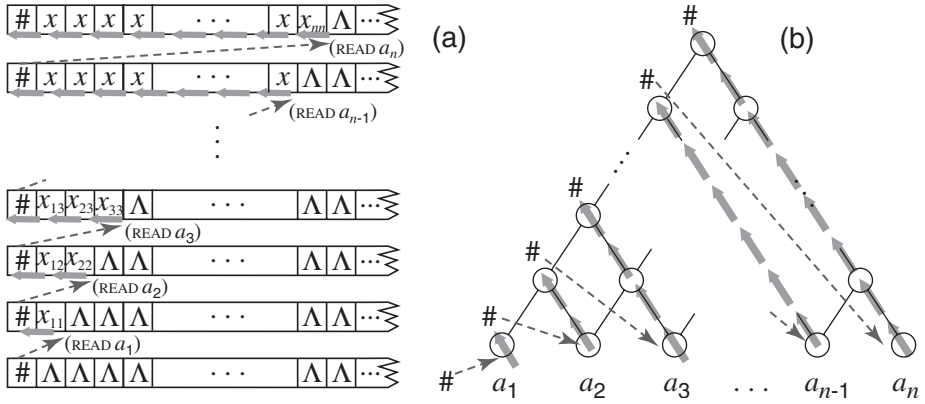

FIGURE 2. (a) a computation of a simple Turing machine; (b) its trellis automaton interpretation.

(2) if $q = q_0$ and $x = \Lambda$, then $a \neq \epsilon$, $q' \neq q_0$, $x' \neq \#$ and $d = -1$: an input symbol is read at the beginning of the right-to-left sweep;
(3) if $q \neq q_0$ and $x \neq \#$, then $a = \epsilon$, $q' \neq q_0$, $x' \neq \#$ and $d = -1$: right-to-left sweep, the tape is being changed;
(4) if $q \neq q_0$ and $x = \#$, then $a = \epsilon$, $q' = q_0$, $x' = \#$ and $d = +1$: end of a right-to-left sweep, starting a left-to-right sweep.

If the machine's input is exhausted and it is reading the first # square, it accepts the input if its current state is in F, rejects otherwise.

The equivalence of this restricted type of Turing machines to trellis automata was proved in [7]; the relationship between the two devices, despite their different origin, again amounts to isomorphism: in Figure 2 it is shown how a computation of such a Turing machine actually computes the same states that comprise the triangle of a computation of a trellis automaton. For the technical details of this simulation the reader is referred to the original paper [7].

2.3. Linear conjunctive grammars

Let us start with a definition of conjunctive grammars of the general form [9]:

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

$$A \to \alpha_1 \& \ldots \& \alpha_n \quad (A \in N;\ n \geqslant 1;\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*), \tag{6}$$

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

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 defined as follows:

Definition 6. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. The set of conjunctive formulae $\mathcal{F} \subset (\Sigma \cup N \cup \{\text{“(”}, \text{“\&”}, \text{“)”}\})^*$ 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 the relation of one-step derivability of the set of formulae:

- a nonterminal can be rewritten with a body of a rule enclosed in parentheses:
$$s_1 A s_2 \underset{G}{\Longrightarrow} s_1(\alpha_1 \& \ldots \& \alpha_n) s_2; \tag{7}$$
- a conjunction of one or more identical terminal strings enclosed in parentheses can be replaced with one such string without the parentheses:
$$s_1(w \& \ldots \& w) s_2 \underset{G}{\Longrightarrow} s_1 w s_2. \tag{8}$$

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

The language of a formula is the set of all terminal strings derivable from the formula: $L_G(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \underset{G}{\Longrightarrow}{}^* w\}$. The language generated by the grammar is the language generated by its start symbol: $L(G) = L_G(S)$.

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

Definition 8. A conjunctive grammar $G = (\Sigma, N, P, S)$ is said to be linear, if every rule in P is of the form

$$A \to u_1 B_1 v_1 \& \ldots \& u_m B_m v_m \quad (m \geqslant 1,\ u_i, v_i \in \Sigma^*,\ B_i \in N) \tag{9a}$$
$$A \to w \quad (w \in \Sigma^*). \tag{9b}$$

A number of usual examples of non-context-free languages, including $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{a^m b^n c^m d^n \mid m, n \geqslant 0\}$, $\{wcw \mid w \in \{a, b\}^*\}$ and the language of all derivations in any string-rewriting system were proved to be linear conjunctive in [9,11]. The question of whether linear conjunctive grammars and conjunctive grammars of the general form are equal in generative power was left open in [9,11]; it will be solved negatively as one of the results of this paper.

It has been proved in [9] that every linear conjunctive grammar can be effectively transformed to an equivalent grammar of the following form:

Definition 9. 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

$$A \to bB_1\& \dots \&bB_m\&C_1c\& \dots \&C_nc \quad (m+n \geqslant 1;\ B_i, C_j \in N;\ b, c \in \Sigma), \tag{10a}$$

$$A \to a \quad (A \in N, a \in \Sigma), \tag{10b}$$

$$S \to \epsilon, \quad \text{only if } S \text{ does not appear in right hand sides of rules.} \tag{10c}$$

Despite the increased generative power in comparison with linear context-free grammars, any language generated by a linear conjunctive grammar is still a quadratic-time language. Let us briefly describe the $O(n^2)$ time recognition algorithm of [9], which will be used in the simulation of linear conjunctive grammars by trellis automata.

Given a string $a_1 \dots a_n \in \Sigma^+$ $(n \geqslant 1)$, the algorithm computes the sets

$$T_{ij} = \{A \mid A \in N,\ A \overset{G}{\Longrightarrow}{}^* a_i \dots a_j\} \tag{11}$$

for all i and j $(1 \leqslant i \leqslant j \leqslant n)$. The sets T_{ii} can be computed immediately as

$$T_{ii} = \{A \mid A \in N,\ A \to a_i \in P\}. \tag{12}$$

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

$$\begin{aligned} T_{ij} = \{A \mid &\text{there is a rule } A \to bB_1\& \dots \&bB_m\&C_1c\& \dots \&C_nc \in P, \\ &\text{such that } b = a_i,\ c = a_j,\ \text{for all } p\,(1 \leqslant p \leqslant m)\ B_p \in T_{i+1,j}, \\ &\text{and for all } q\,(1 \leqslant q \leqslant n)\ C_q \in T_{i,j-1}\}, \end{aligned} \tag{13}$$

i.e., T_{ij} depends on the sets $T_{i+1,j}$ and $T_{i,j+1}$, and the symbols a_i and a_j. The recognition algorithm [9], Algorithm 2, does n assignments (12) and $n(n-1)/2$ assignments (13), and each of these assignments can be done in constant time.

2.4. 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]. Conjunctive grammars possess a similar characterization by language equations with union, intersection and concatenation [10].

Definition 10 (system of equations)**.** Let Σ be an alphabet. Let $n \geqslant 1$. Let $X = (X_1, \dots, X_n)$ be a vector of language variables, which assume values of languages over Σ. Let $\varphi_1, \dots, \varphi_n$ be expressions that depend upon the variables X and may contain these variables, the constant languages $\{\epsilon\}$ and $\{a\}$ (for all $a \in \Sigma$),

set-theoretic union and intersection, as well as concatenation. Then

$$\begin{cases} X_1 &= \varphi_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{14}$$

is called a resolved system of equations over Σ in variables X.

A vector of languages $L = (L_1, \ldots, L_n)$ is a solution of (14) if for every i the value of φ_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 [10, 13].

Denote the right hand side of a system (14) as a vector function

$$\varphi(X_1, \ldots, X_n) = (\varphi_1(X_1, \ldots, X_n), \ldots, \varphi_n(X_1, \ldots, X_n)), \tag{15}$$

and inductively define its substitutions into itself as

$$\varphi^0(X_1, \ldots, X_n) = (X_1, \ldots, X_n) \quad \text{and} \tag{16a}$$

$$\varphi^{i+1}(X_1, \ldots, X_n) = \varphi(\varphi^i(X_1, \ldots, X_n)). \tag{16b}$$

Define the partial order "$\preccurlyeq$" on the set of language vectors of length n as componentwise inclusion: $(L'_1, \ldots, L'_n) \preccurlyeq (L''_1, \ldots, L''_n)$ if and only if $L'_i \subseteq L''_i$ for all i $(1 \leqslant i \leqslant n)$.

Then one can prove that the operator φ on the set $(2^{\Sigma^*})^n$ is monotone and $\cup$-continuous with respect to this partial order [10], which, by the fixed point theory, yields the following result:

Theorem 2. *[10]. Every system (14) over an alphabet Σ and in variables $X_1, \ldots, X_n$ has least solution (with respect to "$\preccurlyeq$") given by*

$$L = (L_1, \ldots, L_n) = \sup_{i \geqslant 0} \varphi^i(\underbrace{\emptyset, \ldots, \emptyset}_{n}). \tag{17}$$

Theorem 3. *[10]. A language L is generated by a conjunctive grammar if and only there exists a system of equations (14), such that L is the first component of its least solution.*

The same result for the linear case is not directly stated in [10], but can be proved using the same method:

Theorem 4. *A language L is generated by a linear conjunctive grammar if and only there exists a system of equations (14), in which the use of concatenation is restricted to left- and right-concatenation of terminals to expressions, such that L is the first component of its least solution.*

3. The equivalence result

3.1. Linear conjunctive grammars to trellis automata

Let us show that trellis automata can accept any language generated by a linear conjunctive grammar by simulating the recognition algorithm described in Section 2.3.

Each state of the automaton will be associated with some subset of N, so that the state $\Delta(I(a_i \dots a_j))$ "knows" the set T_{ij}; thus, the essence of conversion is subset construction. In order to eliminate the direct dependence of every step of computation on the input string (as in (13)), the characters of the input string will also be encoded in the states, so that the processing unit responsible for computing the state $\Delta(I(a_i \dots a_j))$ will additionally remember the symbols a_i and a_j. The computation of each T_{ii} out of a_i will be done by the function I following (12). The assignment (13) to every T_{ij} $(i < j)$ will be modeled by the function δ.

Formally, let $G = (\Sigma, N, P, S)$ be a linear conjunctive grammar in the linear normal form. Construct the trellis automaton $M = M(G) = (\Sigma, Q, I, \delta, F)$, where $Q = \Sigma \times 2^N \times \Sigma$ and

$$I(a) = (a, \{A \mid A \to a \in P\}, a), \tag{18a}$$

$$\delta((b, X, b'), (c', Y, c)) = (b, \{A \mid \exists A \to bB_1 \& \dots bB_m \& C_1 c \& \dots \& C_n c : X, Y \subseteq N,\ B_i \in Y,\ C_j \in X\}, c), \tag{18b}$$

$$F = \{(a, X, b) \mid X \subseteq N,\ S \in X,\ a, b \in \Sigma\}. \tag{18c}$$

The correctness of the construction is stated in the following lemma:

Lemma 1. *Let $w \in \Sigma^+$ and let $\Delta(I(w)) = (b, X, c)$. Then the first symbol of w is b, the last symbol of w is c, and for each nonterminal $A \in N$, $A \overset{G}{\Longrightarrow}{}^* w$ if and only if $A \in X$.*

Proof. Induction on $|w|$.

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

Induction step. Let $w = buc$ $(b, c \in \Sigma,\ u \in \Sigma^*)$. By the definition of Δ, $\Delta(I(buc)) = \delta(\Delta(I(bu), \Delta(I(uc))$. By the induction hypothesis, $\Delta(I(bu)) = (b, Y, b')$, where b' is the last symbol of bu and $Y \subseteq N$, and $\Delta(I(uc)) = (c', Z, c)$, where c' is the first symbol of uc and $Z \subseteq N$. Then, by (18b), $\Delta(I(buc)) = (b, X, c)$ for some $X \subseteq N$. Fix $A \in N$.

$\Longrightarrow$**:** If the nonterminal A generates buc, then there exists a rule of the form

$$A \to bB_1 \& \dots bB_m \& C_1 c \& \dots \& C_n c \in P, \tag{19}$$

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

By the induction hypothesis, this means that $B_i \in Z$ and $C_j \in Y$ for all i, j, which, by the rule (19) and by the construction of δ, implies that $A \in X$.

TABLE 1. The states of the automaton.

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	$(b,\ \{B\},\ a)$

$\Leftarrow$**:** If $A \in X$, then, by (18b), there exists some rule of the form (19), where B_i are all in Z and all C_j are in Y. Then, by the induction hypothesis, $B_i \Longrightarrow^* uc$ and $C_i \Longrightarrow^* bu$, which allows to construct a derivation of the string w from the nonterminal A. □

Theorem 5. *For every linear conjunctive grammar $G = (\Sigma, N, P, S)$ there exists and can be effectively constructed a trellis automaton $M = (\Sigma, Q, I, \delta, F)$, such that $L(M) = L(G) \setminus \{\epsilon\}$.*

Let us give an example of transformation (18) of a grammar in the linear normal form into an equivalent automaton.

Example 1. The following linear conjunctive grammar in the linear normal form generates 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 cardinality of the set $\Sigma \times 2^N \times \Sigma$ is $2{\cdot}128{\cdot}2 = 512$, but in fact only fourteen of these states will be potentially reachable from the initial state. These reachable states are enumerated in Table 1. The initial function I of the constructed automaton maps a to 0 and b to 10; the transition function $\delta : Q \times Q \to Q$ is given in Table 2; the set of accepting states is $F = \{2\}$.

Consider the string $aba \in L$ and the computation of the recognition algorithm [9] Algorithm 2, for the grammar G given the input aba. The sets $T_{ij} \subseteq N = \{S, K, P, A, R, Q, B\}$ $(1 \leqslant i \leqslant j \leqslant 3)$, as in (11), constructed by this algorithm are given in Figure 3a. The next Figure 3b shows the computation of the constructed automaton on the same input string. This computation is explained in Figure 3c, where the numbers of the states of the automaton are interpreted according to Table 1.

TABLE 2. The function δ.

	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

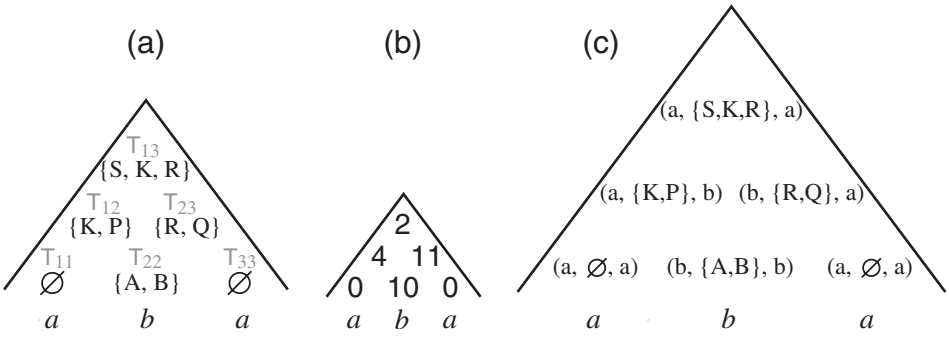

FIGURE 3. The string *aba*: (a) recognition algorithm for the grammar; (b) computation of the automaton; (c) explanation of the computation.

3.2. TRELLIS AUTOMATA TO LINEAR CONJUNCTIVE GRAMMARS

Having proved that trellis automata are at least as powerful as linear conjunctive grammars, it remains to demonstrate how the grammars can in turn simulate the automata.

Let $M = (\Sigma, Q, I, \delta, F)$ be a trellis automaton. 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:

$$S \to A_q \quad (\text{for all } q \in F) \tag{20a}$$
$$A_{I(a)} \to a \quad (\text{for all } a \in \Sigma) \tag{20b}$$
$$A_{\delta(q_1,q_2)} \to A_{q_1} c \& b A_{q_2} \quad (\text{for all } q_1, q_2 \in Q \text{ and } b, c \in \Sigma). \tag{20c}$$

The following lemma states the correctness of the construction.

Lemma 2. *For every string $w \in \Sigma^+$ and for every 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 $w = buc$ $(b, c \in \Sigma,\ u \in \Sigma^*)$.

$\Rightarrow$**:** If $A_q \overset{G}{\Longrightarrow}{}^* buc$, then there exists a rule

$$A_q \to A_{q_1} c \& b A_{q_2} \in P, \quad \text{such that} \tag{21a}$$
$$A_{q_1} \Longrightarrow^* bu, \tag{21b}$$
$$A_{q_2} \Longrightarrow^* uc, \tag{21c}$$

where, by the construction of P, the rule (21a) must have $q = \delta(q_1, q_2)$; by induction hypothesis, (21c) holds if and only if $q_2 = \Delta(I(uc))$ and (21b) holds if and only if $q_1 = \Delta(I(bu))$. Therefore, $q = \delta(\Delta(I(bu)), \Delta(I(uc))) = \Delta(I(buc))$.

$\Leftarrow$**:** Let $q_1 = \Delta(I(bu))$, $q_2 = \Delta(I(uc))$. Then $\delta(q_1, q_2) = \Delta(I(buc)) = q$. By induction hypothesis, this implies $A_{q_1} \Longrightarrow^* bu$, $A_{q_2} \Longrightarrow^* uc$, and, using the rule (20c), $A_{\delta(q_1,q_2)} \Longrightarrow^* buc$, which means that $A_q \Longrightarrow^* w$. □

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

Proof. $L(M) = L(G(M)) \setminus \{\epsilon\}$ 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 6. *For every trellis automaton M there exists and can be effectively constructed a linear conjunctive grammar G, such that $L(G) = L(M)$.*

Together with the earlier Theorem 5, this allows to make the following conclusion, which is the main result of this paper:

Theorem 7. *A language $L \subseteq \Sigma^+$ is accepted by some trellis automaton if and only if it is generated by some linear conjunctive grammar.*

Let us give an example of this transformation. Consider a trellis automaton for the *Dyck language*, a common example of a language that is not linear context-free:

Example 2. Define $M = (\Sigma, Q, I, \delta, F)$, where $\Sigma = \{a, b\}$, the set of states is $Q = \{\nearrow, \nwarrow, Acc, \sqcup\}$; the initial function is $I(a) = \nearrow$ and $I(b) = \nwarrow$; the transition function is defined as $\delta(\nearrow, \nwarrow) = Acc$, $\delta(\nearrow, q) = \nearrow$ (for all $q \neq Acc$), $\delta(q, \nwarrow) = \nwarrow$ (for all $q \neq Acc$), $\delta(q, \nwarrow) = \nwarrow$, $\delta(Acc, \sqcup) = \nearrow$, $\delta(\sqcup, Acc) = \nwarrow$ and the rest of transitions lead to $\sqcup$. Let the set of accepting states be $F = \{Acc\}$.

The general idea of constructing a one-way real-time cellular automaton for this language was given in [6], while an actual construction of a trellis automaton was independently done in [4]; the construction above mostly follows [4]. The transition table of this automaton and a sample computation on the string $w = aabaababbbaabb$ (borrowed from [6], Fig. 5) are depicted in Figure 4.

Direct application of the automaton-to-grammar construction yields the following linear conjunctive grammar:

$$
\begin{aligned}
&S \to A_{Acc}\\
&A_{Acc} \to A_{\nearrow} b \& a A_{\nwarrow}\\
&A_{\nearrow} \to A_{Acc} a \& a A_{\sqcup} \mid A_{Acc} a \& b A_{\sqcup} \mid A_{Acc} b \& a A_{\sqcup} \mid A_{Acc} b \& b A_{\sqcup}\\
&A_{\nearrow} \to A_{\nearrow} a \& a A_{Acc} \mid A_{\nearrow} a \& b A_{Acc} \mid A_{\nearrow} b \& a A_{Acc} \mid A_{\nearrow} b \& b A_{Acc}\\
&A_{\nearrow} \to A_{\nearrow} a \& a A_{\nearrow} \mid A_{\nearrow} a \& b A_{\nearrow} \mid A_{\nearrow} b \& a A_{\nearrow} \mid A_{\nearrow} b \& b A_{\nearrow}\\
&A_{\nearrow} \to A_{\nearrow} a \& a A_{\sqcup} \mid A_{\nearrow} a \& b A_{\sqcup} \mid A_{\nearrow} b \& a A_{\sqcup} \mid A_{\nearrow} b \& b A_{\sqcup}\\
&A_{\nearrow} \to a\\
&A_{\nwarrow} \to A_{Acc} a \& a A_{\nwarrow} \mid A_{Acc} a \& b A_{\nwarrow} \mid A_{Acc} b \& a A_{\nwarrow} \mid A_{Acc} b \& b A_{\nwarrow}\\
&A_{\nwarrow} \to A_{\nwarrow} a \& a A_{\nwarrow} \mid A_{\nwarrow} a \& b A_{\nwarrow} \mid A_{\nwarrow} b \& a A_{\nwarrow} \mid A_{\nwarrow} b \& b A_{\nwarrow}\\
&A_{\nwarrow} \to A_{\sqcup} a \& a A_{\nwarrow} \mid A_{\sqcup} a \& b A_{\nwarrow} \mid A_{\sqcup} b \& a A_{\nwarrow} \mid A_{\sqcup} b \& b A_{\nwarrow}\\
&A_{\nwarrow} \to A_{\sqcup} a \& a A_{Acc} \mid A_{\sqcup} a \& b A_{Acc} \mid A_{\sqcup} b \& a A_{Acc} \mid A_{\sqcup} b \& b A_{Acc}\\
&A_{\nwarrow} \to b\\
&A_{\sqcup} \to A_{\nwarrow} a \& a A_{\nearrow} \mid A_{\nwarrow} a \& b A_{\nearrow} \mid A_{\nwarrow} b \& a A_{\nearrow} \mid A_{\nwarrow} b \& b A_{\nearrow}\\
&A_{\sqcup} \to A_{\nwarrow} a \& a A_{\sqcup} \mid A_{\nwarrow} a \& b A_{\sqcup} \mid A_{\nwarrow} b \& a A_{\sqcup} \mid A_{\nwarrow} b \& b A_{\sqcup}\\
&A_{\sqcup} \to A_{\sqcup} a \& a A_{\nearrow} \mid A_{\sqcup} a \& b A_{\nearrow} \mid A_{\sqcup} b \& a A_{\nearrow} \mid A_{\sqcup} b \& b A_{\nearrow}\\
&A_{\sqcup} \to A_{\sqcup} a \& a A_{\sqcup} \mid A_{\sqcup} a \& b A_{\sqcup} \mid A_{\sqcup} b \& a A_{\sqcup} \mid A_{\sqcup} b \& b A_{\sqcup}.
\end{aligned}
$$

Some rules of this grammar are clearly superfluous: for instance, since S generates the Dyck language, then no string derived by A_{Acc} can start with b, thus no string can be derived from the formula $(A_{Acc} a \& b A_{\sqcup})$, and therefore the rule $A_{\nearrow} \to A_{Acc} a \& b A_{\sqcup}$ can be safely removed. Once all the unused rules are eliminated, nonterminals are renamed to the more common $\{S', S, A, B, X\}$ instead of $\{S, A_{Acc}, A_{\nearrow}, A_{\nwarrow}, A_{\sqcup}\}$, and the empty string is reinstated, the grammar takes the following form:

$$
\begin{aligned}
&S' \to S \mid \epsilon\\
&S \to Ab \& aB\\
&A \to Sa \& aX \mid Sb \& aX \mid Ab \& aS \mid Aa \& aA \mid Ab \& aA \mid Aa \& aX \mid Ab \& aX \mid a\\
&B \to Sb \& aB \mid Bb \& aB \mid Bb \& bB \mid Xb \& aB \mid Xb \& bB \mid Xb \& aS \mid Xb \& bS \mid b\\
&X \to Ba \& aA \mid Ba \& bA \mid Bb \& aA \mid Bb \& bA \mid Ba \& aX \mid Ba \& bX \mid\\
&\quad \mid Bb \& aX \mid Bb \& bX \mid Xa \& aA \mid Xa \& bA \mid Xb \& aA \mid Xb \& bA \mid\\
&\quad \mid Xa \& aX \mid Xa \& bX \mid Xb \& aX \mid Xb \& bX.
\end{aligned}
$$

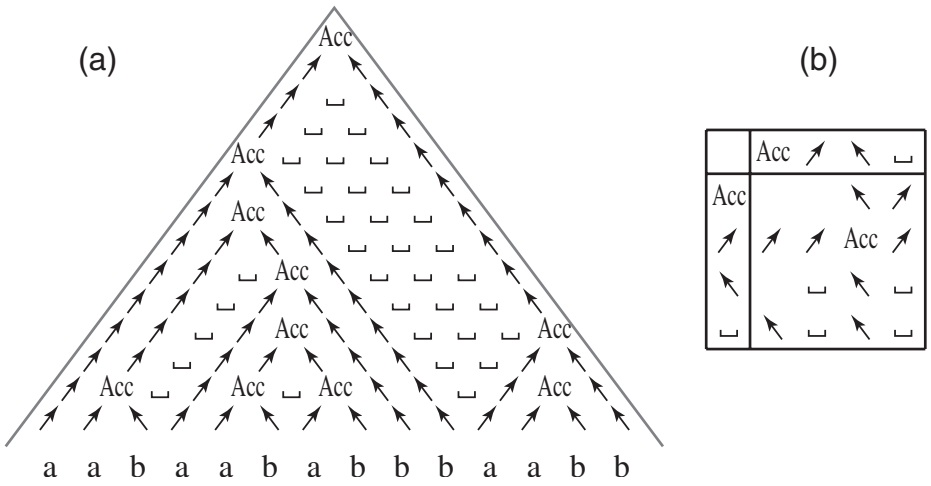

FIGURE 4. Trellis automaton for the Dyck language:
(a) computation on *aabaababbbaabb*; (b) transition table.

However, this grammar is anything but human-readable. Constructing an intuitively meaningful linear conjunctive grammar for the Dyck language remains a problem to solve.

4. General properties of the family

It is easy to prove that the languages generated by trellis automata are closed under all set-theoretic operations: like in the case of DFAs, the closure under union and intersection is obtained using a direct product construction, while complement can be implemented by simply inverting the set of accepting states [4,6]. The same was proved for linear conjunctive grammars using very different techniques: the intersection operation is explicit; union, as in the context-free case, can be obtained by writing multiple rules for one nonterminal; the closure under complement was shown by an elaborate construction of a grammar [11].

Another question is the closure under reversal. It has once been proposed as an open problem for one-way real-time cellular automata [6]; it was positively solved only when their equivalence to trellis automata was noted [2]. For linear conjunctive grammars it is straightforward.

The closure under quotient with finite languages has been shown only for linear conjunctive grammars [11]; it can now be extended to trellis automata (augmented with the capability of handling the empty string). The nonclosure under quotient with regular languages is straightforward [11].

The problem of whether the languages accepted by trellis automata are closed under concatenation was raised in [6] and was open for almost twenty years; finally, the nonclosure was proved in [16] by giving a linear context-free language and

showing by a counting argument that its square cannot be accepted by a one-way real-time cellular automaton. In [11] the problem which we can now recognize as equivalent was raised: whether the languages generated by linear conjunctive grammars are closed under concatenation. A negative answer can now be given.

Although this family of languages is not closed under concatenation, it is closed under some particular cases of concatenation: left-concatenation of Σ^* [6], left- and right-concatenation with arbitrary regular languages [7, 11], marked concatenation [7, 11] and concatenation over disjoint alphabets [7, 11].

The closure or nonclosure of this family under Kleene star was determined neither for trellis automata nor for linear conjunctive grammars [11]. Now, combining the known results on both automata and grammars, this problem can be solved negatively in a couple of lines.

Theorem 8. *The family of linear conjunctive languages is not closed under Kleene star.*

Proof. Consider the following representation of concatenation:

$$L_1 \cdot L_2 = \{c\}^{-1} \cdot \big((c \cdot L_1 \cup L_2 \cdot d)^* \cap c \cdot \Sigma^* \cdot d\big) \cdot \{d\}^{-1}, \tag{22}$$

where $L_1, L_2 \subseteq \Sigma^*$ and c, d are distinct symbols not in Σ. Since linear conjunctive languages are closed under left- and right-concatenation with $\{a\}$ [7,11], under left- and right-quotient with $\{a\}$ [11], under union and under intersection, if we suppose their closure under star, then (22) would imply closure under concatenation, which is not the case [16]. □

Every language accepted by a trellis automaton is easily seen to be deterministic context-sensitive. In fact, a longer chain of inclusions has been investigated in relation to conjunctive and Boolean grammars [13]. Let $\mathcal{L}(LinCF)$, $\mathcal{L}(CF)$, $\mathcal{L}(Lin\&)$, $\mathcal{L}(\&)$ and $\mathcal{L}(Bool)$ denote the families generated by linear context-free, context-free, linear conjunctive, conjunctive and Boolean grammars, respectively. Let $\mathcal{L}(DetCS)$ and $\mathcal{L}(CS)$ be (deterministic) context-sensitive languages. Then

$$\mathcal{L}(Lin\&) \subset \mathcal{L}(\&) \subseteq \mathcal{L}(Bool) \subseteq \mathcal{L}(DetCS) \subseteq \mathcal{L}(CS). \tag{23}$$

Only the first of these inclusions can now be proved to be proper (see Th. 9 below), and it is not even known whether there are context-sensitive languages not generated by conjunctive grammars [9] (although the equality is highly improbable, as it would imply **P**=**PSPACE**).

By the results of the present paper, the inclusions between the languages generated by linear context-free, context-free, linear conjunctive and conjunctive grammars can be shown to be strict, thus solving a problem raised in [11].

Theorem 9. *The following relations hold:*

$$\mathcal{L}(CF) \setminus \mathcal{L}(Lin\&) \neq \emptyset, \tag{24a}$$

$$\mathcal{L}(LinCF) \subset \mathcal{L}(CF) \cap \mathcal{L}(Lin\&), \tag{24b}$$

$$\mathcal{L}(Lin\&) \setminus \mathcal{L}(CF) \neq \emptyset, \tag{24c}$$

$$\mathcal{L}(CF) \cup \mathcal{L}(Lin\&) \subset \mathcal{L}(\&). \tag{24d}$$

Proof. (24c) can be proved, for instance, by giving the language $L_{wcw} = \{wcw | w \in \{a,b\}^*\}$, which is linear conjunctive [9], but not context-free.

The inclusion (24b) holds by definitions of the grammars involved, as context-free and linear conjunctive grammars can both be viewed as generalizations of linear context-free grammars. In order to prove that it is proper, it suffices to consider the Dyck language, which is context-free by its definition, linear conjunctive by [4, 6], but is known not to be linear context-free.

(24a) is known from [16]: the square of the Terrier language

$$L_{\text{Terrier}} = \{a^n b^n \mid n \geqslant 1\} \cup \{a^n b \{a,b\}^* a b^n \mid n \geqslant 1\} \tag{25}$$

is context-free, but not linear conjunctive.

Turning to (24d), the inclusion holds since context-free and linear conjunctive grammars are both syntactical subcases of conjunctive grammars. Let us prove that the language $L = L_{wcw} \cup L^2_{\text{Terrier}}$ is neither context-free nor linear conjunctive. If it were context-free, then, by the closure of context-free languages under intersection with regular languages, the language $L \cap \{a,b\}^* \cdot c \cdot \{a,b\}^* = L_{wcw}$ would also be context-free, which is known to be untrue. If L were linear conjunctive, then, by the closure of linear conjunctive languages under intersection, $L \cap \{a,b\}^* = L^2_{\text{Terrier}}$ would be linear conjunctive as well, which is also false. □

The relations between these four families, as given by Theorem 9, are summarized in Figure 5.

For the case of a unary alphabet, the relationship is harder to determine. It is known that linear context-free, context-free and linear conjunctive languages over a one-letter alphabet are exactly the regular languages. It is an open problem whether it is so for conjunctive grammars [9], while for Boolean grammars it is actually false [13].

The results of this paper have one simple implication on conjunctive grammars that is worth being mentioned. It turns out that the linear normal form of Definition 9, originally introduced in [9], is not optimal in the sense that it can be further restricted, at the same time preserving the generative power.

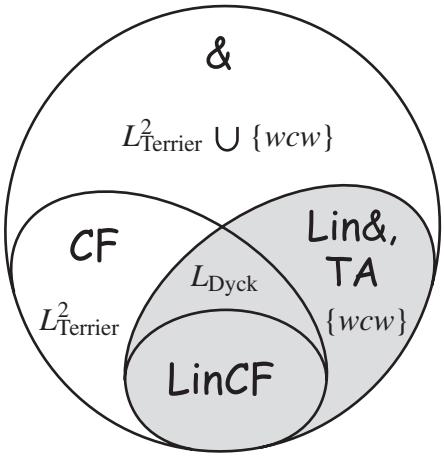

FIGURE 5. Relationship between families of languages.

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

$$A \to bB\&Cc \quad (A, B, C \in N,\ b, c \in \Sigma), \tag{26a}$$

$$A \to a \quad (A \in N, a \in \Sigma), \tag{26b}$$

$$S \to \epsilon, \quad \text{only if } S \text{ does not appear in right hand sides of rules.} \tag{26c}$$

Every grammar can be transformed to this normal form by first converting it into a trellis automaton according to Theorem 5 and then converting it back to a grammar by Theorem 6; the construction (20) used in the latter step produces grammars of the form (26).

Theorem 10. *For every linear conjunctive grammar there exists and can be effectively constructed an equivalent linear conjunctive grammar in the shortened linear normal form.*

5. Conclusion

It was proved that the class of languages generated by linear conjunctive grammars and the class of languages accepted by trellis automata are the same. The following formalisms are now known to be equivalent (see Fig. 6 for the known equalities; the central one is established in this paper):

- one-way real-time cellular automata (1980) [6];
- trellis automata (1981) [2, 4];
- a restricted type of Turing machines (1985) [7];
- linear conjunctive grammars (2000) [9, 11];

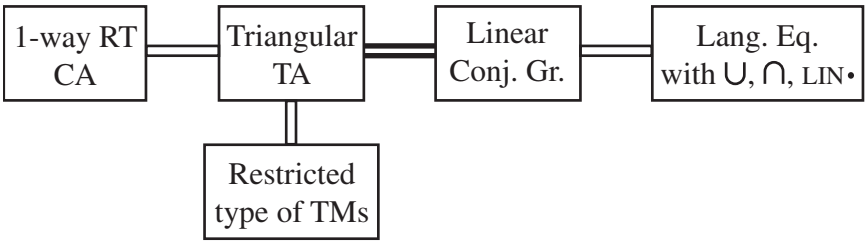

FIGURE 6. Equalities between classes.

- language equations with union, intersection and linear concatenation (2002) [10], both by unique solutions and by least solutions.

There is one more item to be added to this list: linear Boolean grammars [13], which are a further extension of linear conjunctive grammars, where the use of negation is allowed under a certain semantical condition that prevents the concept from being computationally universal.

So, it can be concluded that this single family of languages has characterizations in terms of parallel computing devices, in terms of sequential machines, in terms of transformational grammars, in terms of language equations and in terms of one more class of formal grammars. All these characterizations are simple and natural, and each of them associates this single family to a different area of theoretical computer science. The abundancy of equivalent definitions coming from widely different areas justifies the significance of the family and, hopefully, will lead to further study of its properties.

Acknowledgements. I am grateful to Sheng Yu for pointing me to the concept of trellis automaton and to Kai Salomaa for inspiring discussions on the subject of this paper, as well as for his detailed comments on its numerous revisions.

REFERENCES

[1] J. Autebert, J. Berstel and L. Boasson, Context-Free Languages and Pushdown Automata. *Handbook of Formal Languages* **1** (1997) 111-174.
[2] C. Choffrut and K. Culik II, On real-time cellular automata and trellis automata. *Acta Inform.* **21** (1984) 393-407.
[3] N. Chomsky and M.P. Schützenberger, The algebraic theory of context-free languages. *Computer programming and formal systems* (1963) 118-161.
[4] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata (I, II). *Int. J. Comput. Math.* **15** (1984) 195-212; **16** (1984) 3-22; preliminary version in: Research Rep. CS–81–34, Dept. of Computer Sci., U. of Waterloo, Canada (1981).
[5] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata: stability, decidability and complexity. *Inform. Control* **71** (1986) 218–230.
[6] C. Dyer, One-way bounded cellular automata. *Inform. Control* **44** (1980) 261-281.
[7] O.H. Ibarra and S.M. Kim, Characterizations and computational complexity of systolic trellis automata. *Theoret. Comput. Sci.* **29** (1984) 123-153.

[8] O.H. Ibarra, S.M. Kim and S. Moran, Sequential machine characterizations of trellis and cellular automata and applications. *SIAM J. Comput.* **14** (1985) 426-447.
[9] A. Okhotin, Conjunctive grammars. *J. Autom. Lang. Comb.* **6** (2001) 519-535; preliminary version in: Pre-proceedings of DCAGRS 2000, London, Ontario, Canada, July 27-29 (2000).
[10] A. Okhotin, Conjunctive grammars. and systems of language equations. *Programming and Computer Software* **28** (2002) 243-249.
[11] A. Okhotin, On the closure properties of linear conjunctive languages. *Theoret. Comput. Sci.* **299** (2003) 663-685.
[12] A. Okhotin, Whale Calf, a parser generator for conjunctive grammars, in *Implementation and Application of Automata*, Proc. CIAA 2002, Tours, France, July 3–5, 2002. *Lect. Notes Comput. Sci.* (2002). **2608** 213-220.
[13] A. Okhotin, Boolean grammars, in *Developments in Language Theory*, Proc. DLT 2003, Szeged, Hungary, July 7–11, 2003. *Lect. Notes Comput. Sci.* **2710** (2003) 398-410.
[14] A.R. Smith III, Cellular automata and formal languages, in *Proc. 11th IEEE Annual Sympo. Switching and Automata Theory* (1970) 216-224.
[15] A.R. Smith III, Real-time language recognition by one-dimensional cellular automata. *J. Comput. Syst. Sci.* **6** (1972) 233-252.
[16] V. Terrier, On real-time one-way cellular array. *Theoret. Comput. Sci.* **141** (1995) 331-335.

Communicated by J. Gruska.
Accepted January 15, 2004.