Journal of Computer and System Sciences **65,** 393–414 (2002)
doi:10.1006/jcss.2002.1855

Unary Context-Free Grammars and Pushdown Automata, Descriptional Complexity and Auxiliary Space Lower Bounds

Giovanni Pighizzini[1,2]

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39-20135, Milan, Italy
E-mail: pighizzi@dsi.unimi.it

and

Jeffrey Shallit[3] and Ming-wei Wang

Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
E-mail: shallit@beta.math.uwaterloo.ca; m2wang@hopper.math.uwaterloo.ca

Received November 12, 2001; revised March 27, 2002

It is well known that a context-free language defined over a one-letter alphabet is regular. This implies that unary context-free grammars and unary pushdown automata can be transformed into equivalent finite automata. In this paper, we study these transformations from a descriptional complexity point of view. In particular, we give optimal upper bounds for the number of states of nondeterministic and deterministic finite automata equivalent to unary context-free grammars in Chomsky normal form. These bounds are functions of the number of variables of the given grammars. We also give upper bounds for the number of states of finite automata simulating unary pushdown automata. As a main consequence, we are able to prove a $\log \log n$ lower bound for the workspace used by one-way auxiliary pushdown automata in order to accept nonregular unary languages. The notion of space we consider is the so-called *weak space* concept. © 2002 Elsevier Science (USA)

Key Words: formal languages; context-free grammars; pushdown automata; descriptional complexity; space complexity.

[1] Partially supported by Ministero dell'Istruzione dell'Università e della Ricerca.
[2] To whom correspondence should be addressed.
[3] Partially supported by a grant from NSERC.

1. INTRODUCTION

In the theory of formal languages, *unary* or *tally* languages, i.e., languages defined over a one-letter alphabet, have been the subject of much interesting and fruitful research (e.g., [6, 8, 16]). In many cases, this research exhibits important differences between the universe of all languages and the world of unary languages. Probably, the first result of this kind is the collapse of the classes of unary context-free and regular languages, proved by Ginsburg and Rice [11]. An immediate consequence is that every context-free grammar over a one-letter alphabet can be converted into an equivalent finite state automaton.

In this paper, we consider formalisms such as context-free grammars, finite automata and pushdown automata, and we compare them not only for their expressive powers, but also taking into account also their sizes.

Roughly speaking, the size or, better, the *descriptional complexity* of a formal structure (as, for instance, a grammar or an automaton) is the number of symbols needed to write down its description.

In 1971, Meyer and Fischer [19] proved that for any given recursive function f and for arbitrarily large integers n, there exists a context-free grammar G whose description uses n symbols, such that G generates a regular language L, and any deterministic finite automaton accepting L must have at least $f(n)$ states. This implies that it is not possible to get a recursive bound between the size of context-free grammars generating regular languages and the number of states of equivalent deterministic finite automata. However, Meyer and Fischer's construction used an alphabet of two symbols, and up to now the study of the unary case was left open.

The main result of this paper closes this question by showing that the situation in the unary case is totally different. In fact, we prove that for any unary context-free grammar in Chomsky normal form, with $h \geqslant 2$ variables, there exists an equivalent deterministic finite automaton with less than 2^{h^2} states. This easily implies that any unary context-free grammar of size n can be transformed into an equivalent deterministic automaton with $2^{O(n^2)}$ states. The basic ideas used to get this result are similar to those of the proof given by Parikh [20], that every context-free language is semilinear. However, here the arguments are refined in order to get a sharp upper bound on the number of states of the deterministic automaton equivalent to the grammar. In fact, we are also able to show that our simulation cannot be asymptotically improved.

We further deepen this investigation by considering pushdown automata. It is well known that these devices are equivalent to context-free grammars. Hence, by the above-mentioned result of Ginsburg and Rice, unary pushdown automata (i.e., pushdown automata with a unary input alphabet) characterize the class of unary regular languages, i.e., they are equivalent to finite automata. By carrying on the analysis of the costs, in terms of states, of the simulations between different kinds of unary automata (i.e., automata accepting unary languages) [6, 18], as a consequence of our main result, we state upper bounds on the number of states of nondeterministic and deterministic finite automata simulating a given unary pushdown automaton.

The interest in these results extends beyond the study of automaton simulations. In fact, they are a fundamental step for the solution of a problem related to space lower bounds.

We recall that the analysis of the minimal amount of space used by Turing machines to accept nonregular languages began with the fundamental work of Stearns et al. [23], and has been extensively considered in the literature (e.g., [1, 2, 14]). This investigation is also related to several questions in structural complexity, in particular to sublogarithmic space-bounded computations (see, e.g., [9, 10, 24]).

Some different space notions have been approached during these studies. Among them, we now recall *strong* and *weak spaces*. A machine M works in *strong* space $s(n)$ if and only if *any computation on each input* of length n uses no more than $s(n)$ worktape cells [14, 23]; M works in *weak* space $s(n)$ if and only if, on each accepted input of length n, there exists *at least one accepting computation* using no more than $s(n)$ worktape cells [1]. Of course, if M accepts a language L in strong space $s(n)$, then it also accepts the same language in weak space $s(n)$.

The same space notions can be introduced also for *one-way auxiliary pushdown automata*, i.e., pushdown automata extended with a worktape. Considering these devices, the question arises of finding the minimal amount of worktape space, if any, needed to recognize noncontext-free languages. For strong space, this question was solved by Brandenburg [4] by proving a $\log\log n$ lower bound, whose optimality is witnessed by a unary language.[4]

The situation in the weak case is very different. In fact, as proved by Chytil [7], for any integer k, there exists a noncontext-free language accepted in space $O(\log^{(k)} n)$.[5] This result was improved by Wagner [25], who showed that for each arbitrarily slowly increasing but unbounded recursive function s and for each function $s' \in o(s)$ there is a language accepted by an auxiliary pushdown automaton in weak $O(s(n))$ space, but not in weak $O(s'(n))$ space. A crucial point in these results is that the languages used to get these separations are defined over alphabets of at least two symbols. Up to now, the study of the unary case was left open.

In the paper, we get a solution to this problem. In particular, we are able to show that the $\log\log n$ lower bound for the strong case holds also in the weak case for unary languages, and it is optimal. This result is a nontrivial consequence of our simulation of unary pushdown automata by deterministic finite automata. Furthermore, this result shows another main difference between unary and general languages and between strong and weak space: when we consider the unary case, or strong space, the lower bound is $\log\log n$; instead, in the case of weak space, there are nonunary noncontext-free languages accepted within very slowly growing space bounds.

The paper is organized as follows. In Section 2, we recall some notions and facts used in the paper. In Section 3, we study the cost of the simulation of unary context-free grammars by nondeterministic finite automata. The construction used to get this

[4] Actually, the space definition presented in [4] corresponds to the weak notion. However, the argument used to prove the lower bound works only for strong space.

[5] $\log^{(k)}$ denotes the *iterated logarithm*, namely, $\log^{(1)} z = \log z$ and $\log^{(k)} z = \log^{(k-1)} \log z$ for $k > 1$.

simulation is refined in Section 4 in order to prove our main result, namely the optimal conversion of unary context-free grammars into deterministic finite automata. Finally, Section 5 is devoted to unary pushdown automata. First, an upper bound to the cost of the simulation of unary pushdown automata by equivalent finite automata is stated; subsequently, this result is used to prove the optimal $\log\log n$ space lower bound for auxiliary pushdown automata accepting unary noncontext-free languages.

2. PRELIMINARIES

In this section, we recall basic notions, notation and facts used in the paper. (For more details see, e.g., [15].)

Given a set S, $\#S$ denotes its cardinality, and 2^S the family of all its subsets. For any $z > 0$, $\ln z$ is the natural logarithm of z, while $\log z$ is the logarithm of z taken to the base 2. By $\log^{(k)}$, we denote the *iterated logarithm*, namely, $\log^{(1)} z = \log z$ and $\log^{(k)} z = \log^{(k-1)}(\log z)$ for $k > 1$. The *greatest common divisor* of integers $a_1, \ldots, a_s$ is denoted by $\gcd(a_1, \ldots, a_s)$, and their *least common multiple* by $\operatorname{lcm}(a_1, \ldots, a_s)$. The following result will be useful (see [5]):

Theorem 1. *Let $a_1, \ldots, a_s$, be positive integers $\leqslant n$, and let X be the set of the numbers of the form*

$$a_1x_1 + \cdots + a_sx_s,$$

where $x_1, \ldots, x_s \geqslant 0$ are integers. Then the set of numbers in X greater than n^2 coincides with the set of multiples of $\gcd(a_1, \ldots, a_s)$ *greater than n^2.*

When $s = 2$ and the given numbers are relatively prime, the result of Theorem 1 can be strengthened as follows:

Theorem 2. *If a, b are integers $\geqslant 0$ with* $\gcd(a, b) = 1$, *then the greatest number that cannot be expressed as a linear combination $ax + by$, with integers $x, y \geqslant 0$ is $ab - a - b$.*

Given an alphabet Σ, Σ^* denotes the set of strings over the alphabet Σ, with the empty string denoted by ε, and Σ^+ denoting the set $\Sigma^* - \{\varepsilon\}$. Given a string $x \in \Sigma^*$, $|x|$ denotes its length. Given an integer n, by $L^{<n}$ ($L^{\leqslant n}$, resp.), we denote the set of strings of length less than n (no greater than n, resp.) belonging to the language L. A language L is said to be *unary* (or *tally*) whenever it can be built over a *single-letter* alphabet. In this case, we let $L \subseteq a^*$.

In the paper, we consider nondeterministic finite automata (nfa), finite automata with ε-moves, and deterministic finite automata (dfa), over the unary alphabet $\{a\}$. An nfa is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of states, Σ is the finite nonempty input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The transition function δ can be extended to strings in the usual way. The language accepted by M is the set $L(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \cap F \neq \emptyset\}$.

Finite automata with ε-moves are defined as nfa's, with the only difference that the domain of δ is $Q \times (\Sigma \cup \{\varepsilon\})$. Any automaton with ε-moves can be simulated by an nfa with the same number of states.

An nfa M is said to be *deterministic* (dfa) if and only if $\#\delta(q, \sigma) = 1$, for any $q \in Q$, and $\sigma \in \Sigma$ (this implies that deterministic automata are assumed to be *complete*).

Note that the transition graph of a unary dfa M, i.e., of a dfa over the alphabet $\Sigma = \{a\}$, consists of a path, which starts from the initial state, followed by a cycle of one or more states. As in [6], the *size* of M is the pair (λ, μ), where $\lambda \geqslant 1$ and $\mu \geqslant 0$ denote the number of states which belong to the cycle and to the path, respectively. Observing the form of unary dfa's, it is not difficult to conclude that unary regular languages correspond to ultimately periodic sets of integers:

THEOREM 3. *Given a unary regular language L and two integers* $\lambda \geqslant 1$, $\mu \geqslant 0$, *the following statements are equivalent*:

(i) *L is accepted by a dfa of size* (λ, μ);

(ii) *for any integer* $m \geqslant \mu$, $a^m \in L$ *if and only if* $a^{m+\lambda} \in L$.

A *context-free grammar* (cfg, for short), is a 4-tuple $G = (V, \Sigma, P, S)$, where V is the set of variables, Σ is the set of terminals, $S \in V$ is the initial symbol and $P \subseteq V \times (V \cup \Sigma)^*$ is the finite set of productions. A production $(A, \alpha) \in P$ is denoted by $A \to \alpha$. The relations $\Rightarrow, \overset{*}{\Rightarrow}$, and $\overset{+}{\Rightarrow}$ are defined in the usual way. Given $\alpha, \beta \in (V \cup \Sigma)^*$, if θ is a derivation of β from α, then we write $\theta : \alpha \overset{*}{\Rightarrow} \beta$. A useful representation of derivations of context-free grammars can be obtained using *parse trees*.

A parse tree (or *tree*, for short) for a context-free grammar G is a labeled tree satisfying the following conditions:

1. Each internal node is labeled by a variable in V.

2. Each leaf is labeled by either a variable, a terminal, or ε. However, if the leaf is labeled ε, then it must be the only child of its parent.

3. If an internal node is labeled with a variable A, and its children, from left to right, are labeled with $X_1, X_2, \ldots, X_k \in V \cup \Sigma$, then $A \to X_1 X_2 \ldots X_k$ is a production of G.

If T is a parse tree whose root is labeled with a variable $A \in V$ and such that the labels of the leaves, from left to right, form a string $\alpha \in (V \cup \Sigma)^*$, then we write $T : A \overset{*}{\Rightarrow} \alpha$. Furthermore, we indicate as $v(T)$ the set of variables which appear as labels of some nodes in T.

The language generated by the grammar G, i.e., the set $\{x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x\}$, is denoted by $L(G)$. The class of languages generated by cfg's is called the class of *context-free languages*.

The class of context-free languages properly contains the class of regular languages (i.e., the languages accepted by finite automata), but in the unary case, these two classes collapse [11].

A grammar $G = (V, \Sigma, P, S)$ is said to be in *Chomsky normal form* if and only if its productions have the form $A \to BC$ or the form $A \to a$, with $A, B, C \in V$ and $a \in \Sigma$. It is well known that each context-free language not containing the empty word can be generated by a context-free grammar in Chomsky normal form.

The following property of parse trees of grammars in Chomsky normal form can be easily proved (see, e.g., [15, Theorem 7.17]):

LEMMA 1. *Let $T : A \overset{*}{\Rightarrow} \alpha$ be a parse tree according to a context-free grammar G in Chomsky normal form. If the longest path from the root to a leaf of T has length k (measured by the number of edges), then $|\alpha| \leqslant 2^{k-1}$.*

For the sake of simplicity, we will consider languages without the empty word ε. However, our results can be easily extended to languages containing ε.

Context-free languages can also be characterized by *pushdown automata*. As usual, a *pushdown automaton* (pda, for short) is denoted by a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where Q is the finite *set of states*, Σ is the *input alphabet*, Γ is the *pushdown alphabet*, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the *start symbol*, $F \subseteq Q$ is the set of *final states*. Without loss of generality, we make the following assumptions about pda's:

(i) at the start of the computation the pushdown store contains only the start symbol Z_0; this symbol is never pushed or popped on the stack;

(ii) the input is accepted if and only if the automaton reaches a final state, the pushdown store contains only Z_0 and all the input has been scanned;

(iii) if the automaton moves the input head, then no operations are performed on the stack;

(iv) every push adds exactly one symbol on the stack.

Note that the transition function δ of a pda M can be written as

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times (\{-, \mathrm{pop}\} \cup \{\mathrm{push}(A) \mid A \in \Gamma\})}.$$

In particular, for $q, p \in Q$, $A, B \in \Gamma$, $\sigma \in \Sigma$, $(p, -) \in \delta(q, \sigma, A)$ means that the pda M, in the state q, with A at the top of the stack, by consuming the input σ, can reach the state p without changing the stack contents; $(p, \mathrm{pop}) \in \delta(q, \varepsilon, A)$ $((p, \mathrm{push}(B)) \in \delta(q, \varepsilon, A), (p, -) \in \delta(q, \varepsilon, A)$, respectively) means that M, in the state q, with A at the top of the stack, without reading any input symbol, can reach the state p by popping off the stack the symbol A on the top (by pushing the symbol B on the top of the stack, without changing the stack, respectively).

In order to evaluate the complexities of grammars and finite automata equivalent to a given pda M, we will consider two parameters: the number n of states of M and the number m of pushdown symbols, i.e., the cardinality of the alphabet Γ. In fact, for a fixed input alphabet Σ, each pda satisfying the above condition (iv) has a description whose length is polynomial in n and m. Without this condition, other parameters, such as the maximum number of symbols that can be pushed on the stack in one move, must be considered. For instance, note that for each $n \geqslant 1$, the regular language $L_n = (a^n)^*$, which requires n states to be recognized by an nfa or by a dfa, can be accepted by a pda with two states and two pushdown symbols that, in one move, is able to push n symbols on the stack. Similar considerations can be formulated for grammars. The language L_n is generated by the grammar containing only one variable S and the productions $S \to a^n$ and $S \to a^n S$. However, it is not difficult to see that for grammars in Chomsky normal form, the number of variables is a "reasonable" measure of complexity [13].

We recall that a *one-way nondeterministic auxiliary pushdown automaton* (auxpda, for short) [4] is a pushdown automaton, extended with a read/write worktape. At the start of the computation the worktape is empty. Moves are defined as for pda's, with the following differences: each transition also depends on the contents of the currently scanned worktape cell; a transition also modifies the worktape, by writing a new symbol on the currently scanned cell and by moving the worktape head one position left or right.

Also for auxpda's, without loss of generality, we make assumptions (i)–(iv). Space complexity is defined considering only the auxiliary worktape.

Finally, we recall the notions of space we are interested in (see, e.g., [24]).

DEFINITION 1. An auxiliary pushdown automaton M works in *weak* space $s(n)$ if and only if, on each accepted input of length n, there exists *at least one accepting computation* using no more than $s(n)$ worktape cells.

M works in *strong* space $s(n)$ if and only if *any computation on each input* of length n uses no more than $s(n)$ worktape cells.

Clearly, if a language L is accepted by an auxpda M in strong $s(n)$ space, then it is accepted by the same auxpda M is weak $s(n)$ space. In the literature, some other space notions are considered [2, 24], but they are beyond the scope of this paper.

3. FROM UNARY GRAMMARS TO NONDETERMINISTIC AUTOMATA

In this section, we study the simulation of unary cfg's by nfa's. In particular, we show that for any given unary cfg $G = (V, \{a\}, P, S)$ in Chomsky normal form with h variables, there exists an equivalent nfa with $2^{O(h)}$ states. Furthermore, we prove that this result is optimal.

Let us start with the following preliminary result:

LEMMA 2. *Let $G = (V, \{a\}, P, S)$ be a unary context-free grammar in Chomsky normal form with h variables, and let $T : S \overset{*}{\Rightarrow} a^l$ be a tree in G. Then*:

(i) *for each variable $A \in v(T)$ and for each tree $T' : A \overset{+}{\Rightarrow} a^i A a^j$, there exists a tree $T'' : S \overset{*}{\Rightarrow} a^{l+i+j}$, with $v(T'') = v(T) \cup v(T')$;*

(ii) *if $l > 2^{h-1}$ then there exist three integers s, i, j, with $l = s + i + j$, $s > 0$, and $0 < i + j < 2^h$, a tree $T_1 : S \overset{*}{\Rightarrow} a^s$, a variable $A \in v(T_1)$, and a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, such that $v(T) = v(T_1) \cup v(T_2)$.*

Proof. The arguments used in the proof are similar to those of the "pumping lemma" for context-free languages (see, e.g., [15]).

(i) In the tree $T : S \overset{*}{\Rightarrow} a^l$ we choose a node v_1 labeled with the variable A. Let T_1 be the subtree of T rooted at v_1.

We can build a new tree T'' by inserting the tree T' at the node v_1 in T: first, we replace the subtree T_1 rooted at v_1 with the tree T'; subsequently, we append T_1 to the leaf labeled A of the tree so obtained. It is not difficult to observe that T'' derives the string a^{l+i+j} from S, i.e., $T'' : S \overset{*}{\Rightarrow} a^{l+i+j}$. Moreover, $v(T'') = v(T) \cup v(T')$.

(ii) By Lemma 1, if $l > 2^{h-1}$, then the tree $T : S \overset{*}{\Rightarrow} a^l$ must contain a longest path with at least $h+1$ edges. Hence, there are two nodes v_1 and v_2 on this path which are labeled with the same variable A, where v_1 is closer than v_2 to the root of T. By replacing in T the subtree rooted at v_1 with the subtree rooted at v_2, we get a new tree $T_1 : S \overset{*}{\Rightarrow} a^s$ with $s < l$. Let T_2 be the subtree obtained from T by taking as root v_1 and by deleting the subtree rooted at v_2. Then $T_2 : A \overset{+}{\Rightarrow} a^iAa^j$, for some integers i,j, with $i+j>0$. Furthermore, $v(T) = v(T_1) \cup v(T_2)$.

Finally, we observe that the node v_1 can be chosen sufficiently close to the bottom of the tree T. In particular, we can choose v_1 in such a way that the longest path from v_1 to a leaf has length no more than $h+1$. By Lemma 1, this implies that the length of the terminal string generated by the subtree of T rooted at v_1 is bounded by 2^h. Since the subtree rooted at v_2 must generate at least one terminal symbol, we conclude that $i+j<2^h$. ∎

As a consequence of statement (ii) of Lemma 2, any parse tree $T : S \overset{*}{\Rightarrow} a^l$ of a "long" string (i.e., with $l > 2^{h-1}$) in $L(G)$ can be obtained by "padding" a parse tree $T_1 : S \overset{*}{\Rightarrow} a^s$ of a shortest string ($s<l$), with a tree $T_2 : A \overset{+}{\Rightarrow} a^iAa^j$, where $A \in v(T_1)$. Of course, if $s > 2^{h-1}$, the parse tree T_1 can also be obtained in a similar way. Furthermore, by statement (i), each tree so constructed generates a string belonging to $L(G)$. Using these ideas, we write the following nondeterministic procedure in order to generate all the strings of $L(G)$:

```
nondeterministically select a tree $T_1 : S \overset{*}{\Rightarrow} a^l$, with $l \leqslant 2^{h-1}$
*enabled* ← $v(T_1)$
*iterate* ← nondeterministically choose *true* or *false*
**while** *iterate* **do**
    nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^iAa^j$,
        with $0<i+j<2^h$ and $A \in$ *enabled*
    *enabled* ← *enabled* $\cup\ v(T_2)$
    $l \leftarrow l+i+j$
    *iterate* ← nondeterministically choose *true* or *false*
**endwhile**
output $a^l$
```

Note that, at the beginning of each iteration, the variable *enabled* contains the set of nonterminals which occur in the tree simulated so far. One of them is used to pump the output string. We now describe an automaton $M = (Q, \{a\}, \delta, q_0, F)$, which implements a similar strategy. The automaton uses ordinary transitions to count input factors corresponding to derivation trees of the form $T_1 : S \overset{*}{\Rightarrow} a^l$, with $l \leqslant 2^{h-1}$, or $T_2 : A \overset{+}{\Rightarrow} a^iAa^j$, with $0 < i + j < 2^h$. When the end of a factor is reached, the automaton can perform an ε-move in order to update the set *enabled* (which is kept in its finite control) and to reach a state where the input can be accepted or a new iteration of the while loop of the algorithm can be simulated. To this end, each state of M is defined by two components: a set of enabled variables and an integer. Formally, M is defined as follows:

- $Q = 2^V \times \{0, \dots, 2^h - 1\}$;
- $q_0 = (\emptyset, 0)$;
- for $0 \leqslant l \leqslant 2^h - 1$, $\alpha \subseteq V : \delta((\alpha, l), a) = \{(\alpha, l+1)\}$ if $l < 2^h - 1$, and $\emptyset$ otherwise;
- for $0 < l \leqslant 2^{h-1}$:

$$\delta((\emptyset, l), \varepsilon) = \{(\beta, 0) \mid \exists T : S \overset{*}{\Rightarrow} a^l \text{ and } \beta = v(T)\};$$

- for $0 < l < 2^h$, $\alpha \subseteq V$, $\alpha \neq \emptyset$:

$$\delta((\alpha, l), \varepsilon) = \{(\beta, 0) \mid \exists A \in \alpha\ \exists T : A \overset{+}{\Rightarrow} a^iAa^j, \text{ s.t. } i + j = l \text{ and } \beta = \alpha \cup v(T)\};$$

- $\delta((\alpha, l), \varepsilon) = \emptyset$ in the other cases;
- $F = \{(\alpha, 0) \mid \alpha \neq \emptyset\}$.

We now illustrate the above construction of the automaton M with a simple example. Let $G = (\{S, A\}, \{a\}, P, S)$ be the grammar with the following productions:

$$\begin{aligned} S &\to SA \mid a, \\ A &\to SS. \end{aligned}$$

It is not difficult to verify that the language L generated by G is $a(aa)^*$. In Fig. 1, the automaton M obtained from G with the previous construction is depicted (the states of the form $(\{A\}, l)$, which are unreachable, are omitted).

To define the ε-transitions of M, four trees T_1, T_2, T_3, T_4 have been considered:

- $T_1 : S \overset{*}{\Rightarrow} a$ with $v(T_1) = \{S\}$, defining the transition from $(\emptyset, 1)$ to $(\{S\}, 0)$;
- $T_2 : S \overset{+}{\Rightarrow} Saa$ with $v(T_2) = \{S, A\}$, defining the transitions from $(\{S\}, 2)$ and $(\{S, A\}, 2)$ to $(\{S, A\}, 0)$;
- $T_3 : A \overset{*}{\Rightarrow} aAa$ and $T_4 : A \overset{*}{\Rightarrow} aaA$ with $v(T_3) = v(T_4) = \{S, A\}$, defining the transition from $(\{S, A\}, 2)$ to $(\{S, A\}, 0)$.

Note that

$$\delta((\emptyset, 0), a^3) = \{(\{S\}, 2), (\{S, A\}, 0), (\emptyset, 3)\}.$$

The accepting path from $(\emptyset, 0)$ to $(\{S, A\}, 0)$ on a^3 corresponds to the tree obtained by padding T_1 with T_2. This is a consequence of a property of M, proved in the next Lemma 3.

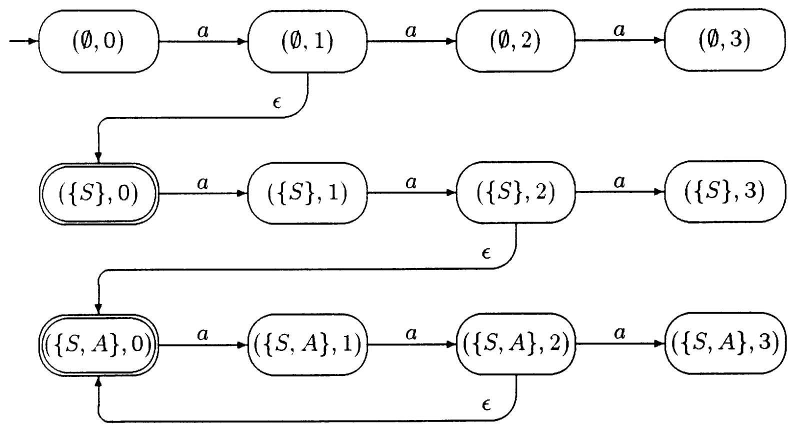

FIG. 1. The automaton M.

Finally, we point out that, after removing ε-moves and unreachable states, M reduces to the automaton represented in Fig. 2.

The main property of the automaton M, which is useful to prove that the language accepted by it coincides with $L(G)$, is the following:

LEMMA 3. *Given an integer* $x > 0$ *and a set* $\alpha \subseteq V$, $(\alpha, 0) \in \delta((\emptyset, 0), a^x)$ *if and only if there exists a parse tree* $T : S \overset{*}{\Rightarrow} a^x$ *such that* $v(T) = \alpha$.

Proof. To prove the *only if* part, we consider a computation $\mathscr{C}$ from $(\emptyset, 0)$ to $(\alpha, 0)$ on input a^x and we proceed by induction on the number of ε-moves in $\mathscr{C}$. We observe that only ε-moves can enter $(\alpha, 0)$. Hence, the last transition of $\mathscr{C}$ must be on the empty word.

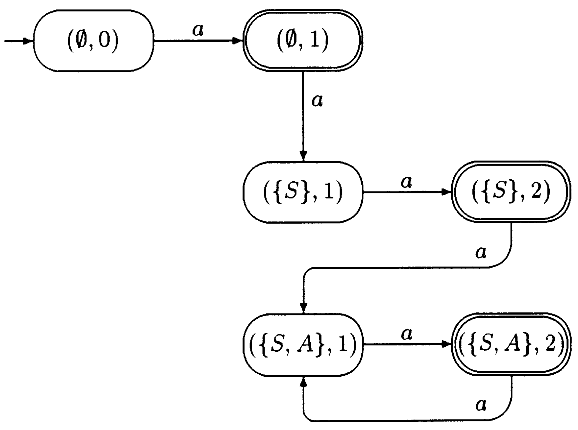

FIG. 2. The automaton M after removing ε-transitions and unreachable states.

If there is only one ε-move, then $(\emptyset, x) \in \delta((\emptyset, 0), a^x)$ and $(\alpha, 0) \in \delta((\emptyset, x), \varepsilon)$. By the definition of δ, this implies the existence of a tree $T : S \overset{*}{\Rightarrow} a^x$ such that $v(T) = \alpha$.

Now, suppose that $\mathscr{C}$ contains $k > 1$ ε-moves. Let (γ, l) be the state reached immediately before the last transition; this means that $(\gamma, l) \in \delta((\emptyset, 0), a^x)$ and $(\alpha, 0) \in \delta((\gamma, l), \varepsilon)$. Hence, by the definition of the ε-moves of M, there exist $A \in \gamma$ and $T' : A \overset{+}{\Rightarrow} a^i A a^j$, such that $i + j = l$ and $\alpha = \gamma \cup v(T')$. Furthermore, since the only transition entering a state (γ, i) with $i > 0$, is from the state $(\gamma, i-1)$ on input a, the state reached by M immediately before consuming the first symbol of the suffix a^l of a^x must be $(\gamma, 0)$. In other words, there exists an integer $y \geqslant 0$ such that $\alpha^x = a^y a^l, \delta((\emptyset, 0), a^y) = (\gamma, 0)$ and $\delta((\gamma, 0), a^l) = (\gamma, l)$. By the induction hypothesis, this implies the existence of a tree $T : S \overset{*}{\Rightarrow} a^y$, with $v(T) = \gamma$. By Lemma 2(i) with $A \in v(T)$, we can pad T with T' to get a tree $T'' : S \overset{*}{\Rightarrow} a^x = a^y a^l$ such that $v(T'') = v(T) \cup v(T') = \gamma \cup v(T') = \alpha$.

Now, we prove the *if* part by induction on x.

If $x \leqslant 2^{h-1}$, then $(\emptyset, x) \in \delta((\emptyset, 0), a^x)$. Given a tree $T : S \overset{*}{\Rightarrow} a^x$, from the state $(\emptyset, x)$ it is possible to perform an ε-move to the state $(v(T), 0)$. Hence, $(v(T), 0) \in \delta((\emptyset, 0), a^x)$.

If $x > 2^{h-1}$ then, by Lemma 2(ii), there are three integers, s, i, j such that $x = s + i + j$ and $0 < i + j < 2^h$, a tree $T_1 : S \overset{*}{\Rightarrow} a^s$, a variable $A \in v(T_1)$, and a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, such that $v(T) = v(T_1) \cup v(T_2)$. Since $s < x$, by the induction hypothesis, it follows that $(v(T_1), 0) \in \delta((\emptyset, 0), a^s)$. Furthermore, by the definition of $M, (v(T_1), i + j) \in \delta((v(T_1), 0), a^{i+j})$, and, being $A \in v(T_1)$, $(v(T_1) \cup v(T_2), 0) \in \delta((v(T_1), i + j), \varepsilon)$. Hence, $(v(T_1) \cup v(T_2), 0) = (v(T), 0) \in \delta((\emptyset, 0), a^x)$. ∎

Now, we are able to prove the main result of this section:

THEOREM 4. *For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most* $2^{2h-1} + 1$ *states.*

Proof. As an immediate consequence of Lemma 3, the automaton M defined above recognizes the language $L(G)$. The automaton M uses ε-moves, but it is well known that they can be eliminated without increasing the number of states. So, an upper bound 2^{2h} for the number of states of a nfa equivalent to a given cfg in Chomsky normal form easily follows. However, we can do better by deleting from M some states which are not useful. In particular, from the definition of M, we can observe that from each state of the form $(\emptyset, l)$ with $l > 2^{h-1}$ we cannot reach an accepting state, while each state of the form (α, l) with $\alpha \neq \emptyset$ and $S \notin \alpha$ cannot be reached from the initial state. Furthermore, by removing ε-moves, we can also eliminate all the states of the form $(\alpha, 0)$, with $\alpha \neq \emptyset$.

In summary, after the elimination of these states, we get another equivalent nfa M', whose states are the pairs (α, l), such that

- either $\alpha = \emptyset$ and $0 \leqslant l \leqslant 2^{h-1}$,
- or $S \in \alpha \subseteq V$ and $1 \leqslant l \leqslant 2^h - 1$.

At this point, we can easily conclude that the number of states of M' is $2^{2h-1} + 1$. ∎

We point out that, as observed in Section 2, the result of Theorem 4 does not hold if we consider cfg's over larger alphabets.

Now we complete this section by proving that the upper bound given in Theorem 4 is close to optimal. More precisely:

THEOREM 5. *For any integer $h \geqslant 1$, there exists a unary cfg in Chomsky normal form, with h variables, such that any equivalent nfa must have at least* $2^{h-1}+1$ *states.*

Proof. For $h=1$, we consider the grammar $(\{S\},\{a\},\{S\to a\},S)$, generating the language $\{a\}$, which cannot be accepted by any 1-state nfa.

For $h>1$, we consider the unary grammar G with variables $A_0, A_1, \ldots, A_{h-1}$, productions

$$\begin{aligned} A_0 &\to a, \\ A_j &\to A_{j-1}A_{j-1} \qquad \text{for } j=1,\ldots,h-2, \\ A_{h-1} &\to A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1} \end{aligned}$$

and start symbol A_{h-1}.

It is easy to show that, for $j=0,\ldots,h-2$, $A_j \overset{*}{\Rightarrow} a^x$ if and only if $x=2^j$, and that

$$L(G)=(a^{2^{h-1}})^+.$$

Finally, observing that the shortest string belonging to $L(G)$ is $a^{2^{h-1}}$, we can easily conclude that any nfa accepting $L(G)$ must have at least $2^{h-1}+1$ states. ∎

4. FROM UNARY GRAMMARS TO DETERMINISTIC AUTOMATA

By Theorem 4, given a unary cfg $G=(V,\{a\},P,S)$ in Chomsky normal form with h variables, there exists an equivalent nfa with $2^{O(h)}$ states. This automaton can be transformed into a dfa applying the subset construction or the determinization procedure for unary automata, presented in [6]. In both cases, the number of states of the resulting dfa is bounded by a function which grows at least as a double exponential in h.

In this section, we prove that this cost can be dramatically reduced. In fact, we show that there exists a dfa, equivalent to G, with less than 2^{h^2} states.

Let L denote the language generated by the given grammar G. For any $A \in V$, let L_A be the set of strings in L having a derivation tree which uses the variable A, i.e., $L_A=\{x \in \Sigma^* \mid \exists T : S \overset{*}{\Rightarrow} x \text{ with } A \in \nu(T)\}$. Of course, $L=L_S$.

We say that a variable $A \in V$ is *cyclic* whenever there are two integers i and j such that $A \overset{+}{\Rightarrow} a^iAa^j$. We now consider the set V_p containing each cyclic variable A for which it is possible to fix an integer λ_A satisfying the following conditions:

- $0<\lambda_A<2^h$,
- $\lambda_A=i+j$ for two integers i,j such that there exists a parse tree $T_A : A \overset{+}{\Rightarrow} a^i\,Aa^j$.

In other words:

$$V_p = \{A \in V \mid \exists i,j, 0 < i + j < 2^h \text{ and } \exists T_A : A \overset{+}{\Rightarrow} a^i A a^j\}.$$

Intuitively, the variables belonging to V_p are useful to "pump" derivation trees of "short" strings belonging to L, in order to get derivation trees of "long" strings. More precisely, in the light of Lemma 2(ii), it is immediate to get the following equality:

$$L = L^{\leqslant 2^{h-1}} \cup \bigcup_{A \in V_p} L_A. \tag{1}$$

At this point, we are able to evaluate the size of a dfa accepting the language L_A, for any $A \in V_p$:

LEMMA 4. *Given a variable $A \in V_p$, the language L_A can be accepted by a dfa of size (λ_A, μ_A), where $\lambda_A < 2^h$ can be chosen as indicated above, and $\mu_A = 2^{2h} + (2h-3)2^{h-1} + 2 - h$.*

Furthermore, if $h = 2$ then μ_A can be reduced to 10.

Proof. First, we observe that, by Lemma 3, the regularity of L_A can be proved by building the automaton $M_A = (Q, \Sigma, \delta, q_0, F_A)$, where Q, Σ, δ, q_0 are defined as for the automaton M presented in Section 3, while $F_A = \{(\alpha, 0) \mid A \in \alpha\}$.

Now, according to Theorem 3, in order to prove that L_A is accepted by a dfa of size (λ_A, μ_A), we show that for any $x \geqslant \mu_A$, $a^x \in L_A$ if and only if $a^{x+\lambda_A} \in L_A$.

By definition of L_A and by Lemma 2(i), it is easy to see that for any $x \geqslant 0$, $a^x \in L_A$ implies that $a^{x+\lambda_A} \in L_A$.

Conversely, let $x \geqslant \mu_A$ be an integer such that $a^{x+\lambda_A} \in L_A$, and let $T : S \overset{*}{\Rightarrow} a^{x+\lambda_A}$ be a derivation tree such that $A \in \nu(T)$. By Lemma 3, there is a path $\mathscr{C}$ in the automaton M_A on the input $a^{x+\lambda_A}$ from the initial state $(\emptyset, 0)$ to the final state $(\nu(T), 0)$.

Let $\{l_1, \ldots, l_s\}$ be the set of numbers $u + v < 2^h$ such that there exists a derivation of the form $B \overset{+}{\Rightarrow} a^u B a^v$, with $B \in V$, involving only variables in $\nu(T)$, i.e.,

$$\{l_1, \ldots, l_s\} = \{u + v < 2^h \mid \exists B \in \nu(T)\, \exists T' : B \overset{+}{\Rightarrow} a^u B a^v \text{ s.t. } \nu(T') \subseteq \nu(T)\}.$$

Note that, by definition of M_A, the length of any input factor consumed in a simple cycle on the path $\mathscr{C}$ must belong to $\{l_1, \ldots, l_s\}$. On the other hand, the number λ_A could not belong to $\{l_1, \ldots, l_s\}$.

Let $\mathscr{C}_0$ be the path obtained from $\mathscr{C}$ by removing all cycles, and x_0 be the number of input symbols consumed by it. Since $\mathscr{C}_0$ and $\mathscr{C}$ ends in the same state $(\nu(T), 0)$, by Lemma 3, there exists a parse tree $T_0 : S \overset{*}{\Rightarrow} a^{x_0}$ such that $\nu(T) = \nu(T_0)$. Furthermore,

$$x_0 \leqslant 2^{h-1} + (2^h - 1)h.$$

In fact, for $\alpha, \beta \subseteq V$ and $0 \leqslant i \leqslant 2^h - 1$, $(\beta, 0) \in \delta((\alpha, i), \varepsilon)$ implies $\alpha \subseteq \beta$. Hence, the number of different first components of the states in $\mathscr{C}_0$ is at most $h + 1$. Moreover, in the states with the empty set as a first component, M_A can consume no more than

2^{h-1} input symbols, while in the states with a nonempty first component, M_A can consume no more than $2^h - 1$ input symbols. Also,

$$x + \lambda_A = x_0 + l_1x_1 + l_2x_2 + \cdots + l_sx_s,$$

where, for $k = 1, \ldots, s$, $x_k \geqslant 0$ is the number of times some cycle of length l_k is repeated in the path $\mathscr{C}$. We observe that the number

$$x - x_0 = l_1x_1 + l_2x_2 + \cdots + l_sx_s - \lambda_A$$

must be a multiple of the gcd of $l_1, l_2, \ldots, l_s$ and λ_A. Furthermore,

$$\begin{aligned} x - x_0 &\geqslant \mu_A - 2^{h-1} - (2^h - 1)h \\ &= 2^{2h} + (2h-3)2^{h-1} + 2 - h - 2^{h-1} - h2^h + h \\ &= 2^{2h} - 2 \cdot 2^h + 2 \\ &> (2^h - 1)^2. \end{aligned}$$

Since $l_1, \ldots, l_s$, $\lambda_A \leqslant 2^h - 1$, by Theorem 1 there exist integers $y_1, \ldots, y_s$, $y \geqslant 0$ such that

$$x - x_0 = l_1y_1 + l_2y_2 + \cdots + l_sy_s + \lambda_Ay,$$

i.e.,

$$x = x_0 + l_1y_1 + l_2y_2 + \cdots + l_sy_s + \lambda_Ay.$$

The last equality suggests the construction of an accepting path on a^x in the following way:

- we start with the path $\mathscr{C}_0$, from $(\emptyset, 0)$ to $(v(T), 0)$, where the input a^{x_0} is consumed;
- for $k = 1, \ldots, s$, we append to the path obtained so far y_k cycles of length l_k (the existence of these cycles, starting and ending in the state $(v(T), 0)$, follows from the definition of $l_1, \ldots, l_s$);
- we complete the new path by appending a path from $(v(T), 0)$ to $(v(T) \cup v(T_A), 0)$, which consumes λ_A input symbols, where $T_A : A \overset{+}{\Rightarrow} a^iAa^j$ and $i + j = \lambda_A$.

It is easy to verify that this path accepts the input a^x.

Thus, we can conclude that for $x \geqslant \mu_A$, $a^{x+\lambda_A} \in L_A$ implies $a^x \in L_A$.

Now, we consider the particular case of a grammar with two variables, i.e., $h = 2$. By considering an integer $x \geqslant 10$ such that $a^{x+\lambda_A} \in L_A$, we can repeat the previous proof, up to the point where we determine that the number

$$m = x - x_0 = l_1x_1 + l_2x_2 + \cdots + l_sx_s - \lambda_A$$

is a multiple of $l = \gcd(l_1, \ldots, l_s, \lambda_A)$.

Since $x \geqslant 10$ and $x_0 \leqslant 2^{h-1} + (2^h - 1)h = 8$, it turns out that $m \geqslant 2$. Furthermore, $\{l_1, \ldots, l_s, \lambda_A\} \subseteq \{1, 2, 3\}$ and $l \leqslant 3$. By enumerating all possible cases, we show that m

can be expressed as

$$m = l_1 y_1 + l_2 y_2 + \cdots + l_s y_s + \lambda_A y \tag{2}$$

for some $y_1, y_2, \ldots, y_s, y \geqslant 0$. From this result, it will be possible to complete the proof as in the general case.

When $1 \in \{l_1, \ldots, l_s, \lambda_A\}$ or $l_1 = l_2 = \cdots = l_s = \lambda_A = 2$ or $l_1 = l_2 = \cdots = l_s = \lambda_A = 3$, the result is trivial (in fact, we are able to express all multiples of l in form (2)). The remaining case is $\{l_1, \ldots, l_s, \lambda_A\} = \{2, 3\}$. In this case, equality (2) reduces to

$$m = 2z + 3w$$

for some $z,\ w \geqslant 0$. By Theorem 2, each number $\geqslant 2$, and then m, can be expressed in this form. ∎

At this point, we are able to prove the main result of this section:

THEOREM 6. *For any unary cfg $G = (V, \{a\}, P, S)$ in Chomsky normal form with $h \geqslant 2$ variables there exists an equivalent dfa with less than 2^{h^2} states.*

Proof. Let $L = L(G)$ be the language generated by the given grammar G. First, we suppose that the start symbol S of G belongs to V_p. Since $L = L_S$, we can get an upper bound on the number of the states of a dfa accepting L, by using Lemma 4. In particular,

$$\begin{aligned}\lambda_S + \mu_S &< 2^h + 2^{2h} + (2h-3)2^{h-1} + 2 - h \\ &= 2^{2h} + h2^h - 2^{h-1} + 2 - h \\ &\leqslant 2^{2h+1}.\end{aligned}$$

Hence, if $h \geqslant 3$, this amount is bounded above by 2^{h^2}. For $h = 2$, Lemma 4 gives the better upper bound 10 for μ_S. Thus,

$$\lambda_S + \mu_S < 2^h + 10 = 14 < 16 = 2^{h^2}.$$

We now suppose that $S \notin V_p$. We can express the language $L = L(G)$, according to equality (1):

$$L = L^{\leqslant 2^{h-1}} \cup \bigcup_{A \in V_p} L_A.$$

It is easy to show that the finite language $L^{\leqslant 2^{h-1}}$ can be accepted by a dfa of size $(1, 2^{h-1} + 1)$. Furthermore, by Lemma 4, for any variable $A \in V_p$, the language L_A can be accepted by a dfa of size (λ_A, μ_A) with $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h-3)2^{h-1} + 2 - h$.

To get the size of a dfa accepting L, we use a result proved in [21] stating that if a unary regular language $\tilde{L}$ is the union of k languages, which are accepted by dfa's of size $(\lambda_1, \mu_1), \ldots, (\lambda_k, \mu_k)$, respectively, then there is a dfa of size

$$(\mathrm{lcm}(\lambda_1, \ldots, \lambda_k), \max(\mu_1, \ldots, \mu_k))$$

accepting it.

In the case under consideration, we get that L is accepted by a dfa of size (λ, μ), where $\mu = 2^{2h} + (2h-3)2^{h-1} + 2 - h$ and $\lambda = \mathrm{lcm}(\{\lambda_A \mid A \in V_p\})$. Since $\lambda_A < 2^h$ and the cardinality of V_p is at most $h-1$ (in fact $S \notin V_p$), we get that $\lambda \leqslant (2^h - 1)^{h-1}$. Hence, the total number of states is

$$\begin{aligned}\lambda + \mu &\leqslant (2^h - 1)^{h-1} + 2^{2h} + (2h-3)2^{h-1} + 2 - h \\ &< 2^{h^2-h} + 2^{2h} + h2^h \\ &= 2^{h^2}(2^{-h} + 2^{2h-h^2} + h2^{h-h^2}).\end{aligned}$$

We now show that, for $h \geqslant 3$, the following inequality holds:

$$2^{-h} + 2^{2h-h^2} + h2^{h-h^2} < 1.$$

In fact,

$$\begin{aligned}2^{-h} + 2^{2h-h^2} + h2^{h-h^2} &= 2^{-h} + 2^{2h-h^2} + 2^{\log_2 h + h - h^2} \\ &\leqslant 2^{-h} + 2^{2h-h^2+1} \\ &\leqslant 2^{-3} + 2^{-2} \\ &< 1.\end{aligned}$$

Thus, for $h \geqslant 3$, we are able to conclude that the language is accepted by a dfa with less than 2^{h^2} states.

Now, we complete the proof by considering the case $h = 2$, with $S \notin V_p$. As a consequence of Lemma 4, we can reduce μ to 10. Thus,

$$\begin{aligned}\lambda + \mu &\leqslant (2^h - 1)^{h-1} + 10 \\ &= 13 < 16 = 2^{h^2}. \quad \blacksquare\end{aligned}$$

We point out that, in the particular case $h = 1$, the upper bound given in Theorem 6 does not hold. More precisely, it is not difficult to show that the only nonempty languages generated by unary cfg's in Chomsky normal form with one variable are $L_1 = \{a\}$ and $L_2 = \{a^k \mid k \geqslant 1\}$. The minimal complete dfa's accepting L_1 and L_2 have 3 and 2 states, respectively.

We complete this section by showing that the upper bound stated in Theorem 6 is tight. In particular, we will prove that for any integer n there exists a context-free grammar G_n in Chomsky normal form with $O(n)$ variables, such that any equivalent dfa requires 2^{cn^2} states, for a fixed constant $c > 0$. Actually, in order to simplify the exposition, the grammar G_n, we give contains not only productions of the form $A \to BC$ or $A \to a$, but also unit productions, i.e., productions of the form $A \to B$, where A and B are variables. Using standard techniques, as described, for instance, in [15], from a grammar of this kind it is possible to get an equivalent grammar in Chomsky normal form, *without increasing* the number of variables.

To define the grammar G_n, first of all we introduce, for each integer $i \geqslant 0$, variables A_i, B_i, C_i, D_i, and S_i, with the following productions:

$$A_0 \to a,$$

$$A_{i+1} \to A_i A_i,$$

$$B_i \to A_0 A_i,$$

$$C_0 \to a,$$

$$C_{i+1} \to a \mid C_i C_i,$$

$$D_i \to D_i B_i \mid C_i,$$

$$S_i \to D_0 \mid D_1 \mid \cdots \mid D_i.$$

It is not difficult to verify the following properties, for all integers $i, x \geqslant 0$:

- $A_i \overset{*}{\Rightarrow} a^x$ if and only if $x = 2^i$;
- $B_i \overset{*}{\Rightarrow} a^x$ if and only if $x = 2^i + 1$;
- $C_i \overset{*}{\Rightarrow} a^x$ if and only if $1 \leqslant x \leqslant 2^i$;
- $D_i \overset{*}{\Rightarrow} a^x$ if and only if $x \not\equiv 0 \pmod{2^i + 1}$;

(In fact, note that, using leftmost derivations, from D_i we can derive all the sentences of the form $C_i B_i^*$.)

- $S_i \overset{*}{\Rightarrow} a^x$ if and only if $x \not\equiv 0 \,(\mathrm{mod}\, \mathrm{lcm}(2^0 + 1, 2^1 + 1, \ldots, 2^i + 1))$.

Now, let $G_n = (V_n, \{a\}, P_n, S_n)$, where

$$V_n = \{S_n\} \cup \{A_i, B_i, C_i, D_i \mid 0 \leqslant i \leqslant n\}$$

and P_n be the set of productions given above containing these variables on the left-hand side. Hence,

$$L(G_n) = \{a^x \mid x \not\equiv 0 \,(\mathrm{mod}\, \mathrm{lcm}(2^0 + 1, 2^1 + 1, \ldots, 2^n + 1))\}.$$

For the rest of this section, we denote by l_n the number $\mathrm{lcm}(2^0 + 1, 2^1 + 1, \ldots, 2^n + 1)$.

Lemma 5. *Each dfa accepting $L(G_n)$ has at least l_n states.*

Proof. We observe that $a^{l_n} \notin L(G_n)$, while, for $0 < k < l_n$, $a^k \in L(G_n)$. Hence, given $0 < i < j \leqslant l_n$, the string $a^{l_n - j}$ distinguishes a^i and a^j. This implies that any dfa accepting $L(G_n)$ must have at least l_n states. ∎

Now, we estimate l_n. To this end the following result of Bézivin [3] is useful:

THEOREM 7. *Let a, b be integers with $b \neq 0$ and* $\gcd(a, b) = 1$. Let α, β *be the zeroes of the polynomial $x^2 - ax - b$. For $m \geqslant 2$ define*

$$u_m(n) = \frac{\alpha^{mn} - \beta^{mn}}{\alpha^n - \beta^n}.$$

Then,

$$\lim_{n \to \infty} \frac{\ln(u_m(1)u_m(2)\dots u_m(n))}{\ln \operatorname{lcm}(u_m(1)u_m(2)\dots u_m(n))} = \frac{(m-1)L(m)\pi^2}{6H(m)},$$

where $L(m) = \prod_{p|m}(1 - \frac{1}{p^2})$ and $H(m) = \sum_{d|m, d>1} \frac{\varphi(d)\varphi(m/d)d}{m}$.

(*The product in the definition of $L(m)$ is extended to all prime numbers p dividing m, while the sum in the definition of $H(m)$ is extended to all divisors of m, greater than* 1. *$\varphi(n)$ denotes Euler's function which associates with each integer $n > 0$ the number of integers $m \leqslant n$ which are relatively prime to n, i.e., $\varphi(n) = \#\{0 < m \leqslant n \mid \gcd(n, m) = 1\}$.*)

For $a = 3$, $b = -2$, $m = 2$, we get that $u_2(n) = 2^n + 1$, $\varphi(1) = \varphi(2) = 1$, $L(2) = 3/4$, and $H(2) = 1$. Hence,

$$\lim_{n \to \infty} \frac{\ln((2^0 + 1)(2^1 + 1) \cdots (2^n + 1))}{\ln l_n} = \frac{\pi^2}{8}. \tag{3}$$

On the other hand, from a standard result about infinite products, the limit

$$\lim_{n \to \infty} \frac{(2^0 + 1)(2^1 + 1) \cdots (2^n + 1)}{2^0 2^1 \cdots 2^n}$$

exists, because

$$\sum_{i \geqslant 0} \ln\left(\frac{2^i + 1}{2^i}\right) = \sum_{i \geqslant 0} \ln\left(1 + \frac{1}{2^i}\right) < \sum_{i \geqslant 0} \frac{1}{2^i} = 2.$$

(Actually, letting

$$c_1 = \lim_{n \to \infty} \frac{(2^0 + 1)(2^1 + 1) \cdots (2^n + 1)}{2^0 2^1 \cdots 2^n}$$

it is not hard through numerical methods to estimate $c_1 = 4.768$.) Hence,

$$\ln((2^0 + 1)(2^1 + 1) \cdots (2^n + 1)) \sim \ln c_1 + \frac{n(n+1)}{2} \ln 2.$$

Putting this together with (3), we get

$$\ln l_n \sim \frac{4 \ln 2}{\pi^2} n^2.$$

Combining this estimate with Lemma 5, we get:

COROLLARY 1. *There is a constant $c > 0$ such that, for infinitely many integers $h > 0$, there exists a unary context-free grammar in Chomsky normal form with h variables, such that each equivalent dfa must have at least 2^{ch^2} states.*

5. AN OPTIMAL LOWER BOUND FOR UNARY AUXILIARY PUSHDOWN AUTOMATA

In this section, we apply the results of Sections 3 and 4 to the study of unary auxiliary pushdown automata working in *weak* space. In particular, we are able to prove that $\log\log n$ is the minimal amount of space needed by these devices in order to recognize unary noncontext-free languages. Furthermore, this lower bound is optimal. We point out that, when the input alphabet contains at least two symbols, the situation is very different: in fact, in [7] it was shown that for any integer k there exists a noncontext-free language accepted in weak $O(\log^{(k)} n)$ space; this result was improved in [25] by proving that for each arbitrarily slowly increasing but unbounded recursive function s and for each function $s' \in o(s)$ there is a language accepted by an auxpda in weak $O(s(n))$ space, but not in weak $O(s'(n))$ space.

Let us start by studying the cost of converting a unary pushdown automaton into an equivalent dfa. In particular, first we convert the given pda into an equivalent grammar in Chomsky normal form and then, applying the results of Sections 3 and 4, we convert it into an equivalent finite automaton.

For the first part, we start by considering a pda $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, with n states and m pushdown symbols. (For this part, it is not necessary to assume that $\#\Sigma = 1$.) We define the grammar $G_1 = (V, \Sigma, P_1, S)$, where elements of V are triples $[q, A, p]$, with $q, p \in Q$, $A \in \Gamma$, plus the start symbol S, and P_1 contains the following productions:

1. $[q, A, p] \to [q, A, r][r, A, p]$, for $q, p, r \in Q$, $A \in \Gamma$;
2. $[q, A, p] \to [q', B, p']$, for $q, q', p, p' \in Q$, $A, B \in \Gamma$ such that $(q', \text{push}(B)) \in \delta(q, \varepsilon, A)$ and $(p, \text{pop}) \in \delta(p', \varepsilon, B)$;
3. $[q, A, p] \to \sigma$, for $q, p \in Q, \sigma \in \Sigma \cup \{\varepsilon\}, A \in \Gamma$ such that $(p, -) \in \delta(q, \sigma, A)$;
4. $[q, A, q] \to \varepsilon$, for $q \in Q$, $A \in \Gamma$;
5. $S \to [q_0, Z_0, q]$, for $q \in F$.

The techniques used to show that G_1 generates the language accepted by M are very similar to those presented in textbooks (see, e.g., [15]) to prove the correctness of the standard transformation of pda's into cfg's. In particular, the following result can be proved:

Lemma 6. *For any $x \in \Sigma^*$, $q, p \in Q$, $A \in \Gamma$, $[q, A, p] \overset{*}{\Rightarrow} x$ if and only if there exists a computation $\mathscr{C}$ of M verifying the following conditions*:

(i) *$\mathscr{C}$ starts in the state q and ends in the state p; in both these moments the symbol at the top of the stack is A and the height of the stack is the same*;

(ii) *during $\mathscr{C}$ the stack is never popped under its level at the beginning of $\mathscr{C}$*;

(iii) *the input factor consumed during $\mathscr{C}$ is x.*

As a consequence of Lemma 6, it is easy to show that the language generated by G_1 coincides with the language accepted by the given pda M. Furthermore, by applying standard techniques (as described, for instance, in [15]), it is possible to eliminate from G_1 all unit and ε-productions, in order to obtain an equivalent grammar G in Chomsky normal form, with the *same* set of variables. Hence,

THEOREM 8. *For any pda, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with* n^2m+1 *variables.*

Further results comparing the descriptional complexities of pushdown automata and of context-free grammars can be found in [12].

If the given pda M has a unary input alphabet $\Sigma = \{a\}$, then we can now use Theorems 4 and 6, in order to evaluate the number of states of finite automata equivalent to it:

COROLLARY 2. *For any unary pda with n states and m pushdown symbols, there exists an equivalent nfa with at most* $2^{2n^2m+1}+1$ *states and an equivalent dfa with less than* $2^{n^4m^2+2n^2m+1}$ *states.*

Now, we study space lower bounds for unary auxpda's. To this end, it is useful to recall the notion of *automaticity* [22], which is a measure of the complexity of the description of a language by deterministic automata. In particular, the automaticity of a regular language is a constant, while the automaticity of a nonregular language grows at least linearly. More precisely:

DEFINITION 2. Given a language $L \subseteq \Sigma^*$, the *automaticity* of L is the function $A_L : \mathbf{N} \to \mathbf{N}$, which associates with every integer n the minimum number of states of a dfa accepting a language L' such that $L^{\leqslant n} = L'^{\leqslant n}$.

THEOREM 9 (Karp [17]). *Let* $L \subseteq \Sigma^*$ *be a nonregular language. Then* $A_L(n) \geqslant (n+3)/2$, *for infinitely many n.*

The following result is useful in order to evaluate the automaticity of unary languages accepted by auxpda's:

THEOREM 10. *Given an auxpda M accepting a unary language L in weak* $s(n)$ *space, for any integer* $n \geqslant 0$ *there exists a dfa* M_n *with the following properties*:

(i) *the language* L_n *accepted by* M_n *coincides with L on strings of length at most n, i.e.,* $L_n^{\leqslant n} = L^{\leqslant n}$;

(ii) *the number of states of* M_n *is bounded by* $2^{2^{O(s(n))}}$.

Proof. Given the auxpda M and an integer n, we consider the pda M'_n which is obtained from M by encoding, in each state, a state of M and the contents of the first $s(n)$ worktape cells. M'_n performs a simulation step by step of M. When the simulated computation of M tries to exceed the first $s(n)$ worktape cells, M'_n stops and rejects.

Since we are considering *weak* space, an input of length n is accepted by M if and only if there is an accepting computation which does not use more than $s(n)$ worktape cells. By construction, the pda M'_n is able to mimic the same accepting computation. Furthermore, to each rejecting computation of M corresponds a rejecting computation of M'_n. Thus, we can easily conclude that $L_n^{\leqslant n} = L^{\leqslant n}$.

The number of states of M'_n is $2^{O(s(n))}$. By Corollary 2, there exists a dfa M_n with $2^{2^{O(s(n))}}$ states equivalent to M'_n. ∎

Now, we are able to prove the main result of this section:

THEOREM 11. *Let M be a unary auxpda accepting a noncontext-free language L in weak $s(n)$ space. Then $s(n) \notin o(\log \log n)$.*

Proof. By Theorem 10, given M, there exists a constant k such that, for any integer n, from M it is possible to get a dfa M_n, with at most $2^{2^{ks(n)}}$ states, such that the language L_n accepted by M_n verifies the equality $L^{\leqslant n} = L_n^{\leqslant n}$. Hence, the automaticity $A_L(n)$ is bounded by $2^{2^{ks(n)}}$.

Suppose that $s(n) \in o(\log \log n)$. Then $2^{2^{ks(n)}} < (n+3)/2$, for any sufficiently large n. Since L is nonregular, this is a contradiction to Theorem 9. ∎

We point out that the lower bound stated in Theorem 11 is optimal, since the language $L = \{a^{2^i} | i \geqslant 0\}$ is accepted by a deterministic auxpda in strong (and weak) $O(\log \log n)$ space [4, 7].

ACKNOWLEDGMENT

The authors thank anonymous referees for helpful comments and remarks.

REFERENCES

1. M. Alberts, Space complexity of alternating Turing machines, *in* “Proceedings of the FCT ’85,” Lecture Notes in Computer Science, Vol. 199, pp. 1–7, Springer-Verlag, Berlin, 1985.
2. A. Bertoni, C. Mereghetti, and G. Pighizzini, Strong optimal lower bounds for Turing machines that accept nonregular languages, *in* “Proceedings of the MFCS ’95,” Lecture Notes in Computer Science, Vol. 969, pp. 309–318. Springer-Verlag, Berlin, 1995.
3. J.-P. Bézivin, Plus petit commun multiple des termes consécutifs d’une suite réccurente linéaire, *Collect. Math.* **40** (1989), 1–11.
4. F. Brandenburg, On one-way auxiliary pushdown automata, *in* “Proceedings of the 3rd GI Conference,” Lecture Notes in Computer Science, Vol. 48, pp. 133–144, Springer, Berlin, 1977.
5. A. Brauer, On a problem of partitions, *Amer. J. Math.* **64** (1942), 299–312.
6. M. Chrobak, Finite automata and unary languages, *Theoret. Comput. Sci.* **47** (1986), 149–158.
7. M. Chytil, Almost context-free languages, *Fund. Inform.* **IX** (1986), 283–322.
8. V. Geffert, Tally version of the Savitch and Immerman–Szelepcsényi theorems for sublogarithmic space, *SIAM J. Comput.* **22** (1993), 102–113.
9. V. Geffert, Bridging across the log(n) space frontier, *Inform. Comput.* **142** (1998), 127–148.
10. V. Geffert, C. Mereghetti, and G. Pighizzini, Sublogarithmic bounds on space and reversals, *SIAM J. Comput.* **28**(1) (1999), 325–340.
11. S. Ginsburg and H. Rice, Two families of languages related to ALGOL, *J. ACM* **9**(3) (1962), 350–371.
12. J. Goldstine, J. Price, and D. Wotschke, A pushdown automaton or a context-free grammar—Which is more economical? *Theoret. Comput. Sci.* **18** (1982), 33–40.
13. J. Gruska, Descriptional complexity of context-free languages, *in* “Proceedings of the MFCS ’73,” pp. 71–83, Mathematical Institute of the Slovak Academy of Sciences, Bratislava, 1973.

14. J. Hopcroft and J. Ullman, Some results on tape-bounded Turing machines, *J. ACM* **16** (1969), 168–177.

15. J. Hopcroft, R. Motwani, and J. Ullman, "Introduction to Automata Theory, Languages, and Computation," 2nd ed., Addison–Wesley, Reading, MA, 2001.

16. J. Kaneps, Regularity of one-letter languages acceptable by 2-way finite probabilistic automata, *in* "Proceedings of the FCT '91," Lecture Notes in Computer Science, Vol. 529, pp. 287–296, Springer-Verlag, Berlin, 1991.

17. R.M. Karp, Some bounds on the storage requirements of sequential machines and Turing machines, *J. ACM* **14**(3) (1967), 478–489.

18. C. Mereghetti and G. Pighizzini, Optimal simulations between unary automata, *SIAM J. Comput.* **30** (2001), 1976–1992.

19. A. Meyer and M. Fischer, Economy of description by automata, grammars, and formal systems, *in* "Proceedings of the 12th IEEE Symposium on Switching and Automata Theory," pp. 188–191, 1971.

20. R. Parikh, On context-free languages, *J. ACM* **13** (1966), 570–581.

21. G. Pighizzini and J. Shallit, Unary language operations, state complexity and Jacobsthal's function, *Internat. J. Found. Comput. Sci.* **13** (2002), 145–159.

22. J. Shallit and Y. Breitbart, Automaticity I: Properties of a measure of descriptional complexity, *J. Comput. System Sci.* **53**(1) (1996), 10–25.

23. R. Stearns, J. Hartmanis, and P. Lewis, Hierarchies of memory limited computations, *in* "IEEE Conference Record on Switching Circuit Theory and Logical Design," pp. 179–190, 1965.

24. A. Szepietowski, "Turing Machines with Sublogarithmic Space," Lecture Notes in Computer Science, Vol. 843, Springer-Verlag, Berlin, l994.

25. K. Wagner, Do there exist languages with an arbitrary small amount of context-sensitivity? *in* "Computational Theory and Logic," Lecture Notes in Computer Science, Vol. 270, pp. 427–432, Springer-Verlag, Berlin, 1987.