Discrete Mathematics and Theoretical Computer Science **4**, 2001, 179–194

Finite Automata with Generalized Acceptance Criteria[†]

Timo Peichl and Heribert Vollmer

Theoretische Informatik, Universität Würzburg, Am Hubland, D-97074 Würzburg, Germany.

received Nov 24, 1999, revised September, 2000, accepted October 2000.

We examine the power of nondeterministic finite automata with acceptance of an input word defined by a leaf language, i. e., a condition on the sequence of leaves in the automaton's computation tree. We study leaf languages either taken from one of the classes of the Chomsky hierarchy, or taken from a time- or space-bounded complexity class. We contrast the obtained results with those known for leaf languages for Turing machines and Boolean circuits.

Keywords: finite automata, generalized acceptance criteria, leaf language, formal languages, complexity classes

1 Introduction

Let M be a nondeterministic finite automaton and w be an input word. Usually w is said to be accepted by M if and only if there is at least one possible computation path of M which accepts w. In this paper we look at the tree $T_M(w)$ of all computations that automaton M on input w can possibly perform. A node v in this tree is labelled by a configuration C of M at a certain point during its computation on input w, where such a configuration is given by the state of M and the portion of the input which is still unscanned. The children of v in the computation tree are associated with the successor configurations of C, i. e., if the transition function of M has several entries for this particular C, then each of these will lead to a successor configuration and a child of v in the computation tree. The leaves in the tree are associated to those configurations that M reaches when all input symbols are consumed.

Now the acceptance criterion of nondeterministic automata can be rephrased as follows: An input word w is accepted by M if and only if in the computation tree of M on x there is at least one leaf labelled with an accepting state.

Using the concept of computation trees, we will study modified acceptance criteria in this paper. Consider for example the following question: If we say that a word is accepted by M if and only if the number of accepting leaves in the computation tree is divisible by a fixed prime number p, can non-regular languages be recognized in this way? The acceptance here is thus given by a more complicated condition on the cardinality of the set of accepting paths in the computation tree. (For the definition of the class REG

[†]A preliminary version appeared in the Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, Springer Lecture Notes in Computer Science Vol. 1644, pp. 605–614, 1999.

1365–8050 © 2001 Maison de l'Informatique et des Mathématiques Discrètes (MIMD), Paris, France

we just require that this cardinality is non-zero.) But we do not only consider such cardinality conditions in the present paper.

If we attach certain symbols to the leaves in $T_M(w)$, e. g., the symbol 1 to an accepting leaf and 0 to a non-accepting leaf, then the computation tree of M on input w defines a word, which we get by concatenating the symbols attached to the leaves, read from left to right (in a natural order of the paths of $T_M(w)$ we define below). We call this string the *leaf word* of M on w. Observe that the length of the leaf word can be exponential in the length of w. Generally, an acceptance criterion is nothing other than the set of those leaf words that make M accept its input; that is, such a criterion is defined by a so called *leaf language* L over the alphabet of the leaf symbols. By definition a word is accepted by M if and only if the leaf word of M on input w is in L. In the example above we used as leaf language the set L of all binary words with a number of 1's divisible by p.

We now ask what class of languages such automata can accept given a particular class of leaf languages. E. g., what if we allow all regular languages as acceptance criteria, can non-regular languages be recognized? The main result of this paper is a negative answer to this question. As another example, if the criterion is given by a context-free language, then we see that non-regular, even non context-free languages can be recognized. To mention a final example, if we allow leaf languages from the circuit class NC^1 (a class whose power is captured in a sense by the regular languages, since there are regular languages complete for NC^1 under uniform projections, a very strict reducibility [BIS90]), then we obtain that even PSPACE-complete languages can be accepted by such finite automata.

In this paper we study in a systematic way the power of acceptance criteria given by leaf languages which are (1) taken from a (complexity) class defined via space or time restrictions for Turing machines, or (2) taken from a (formal language) class of the Chomsky hierarchy.

The power of *nondeterministic Turing machines* whose acceptance is given by a leaf language is well-studied, see, e. g., [BCS92, Ver93, HLS$^+$93, JMT96]. More recently the model has also been applied to Boolean circuits, see [CMTV98]; formally, in this latter model so called *programs over automata* were used as leaf string generators – in the case of language decision these programs are known to yield exactly the power of the class NC^1 [Bar89]. Programs over automata consist of (uniform) projections whose outputs are fed into nondeterministic FAs. The power of finite automata per se, the probably most basic type of machine, with acceptance defined by a leaf language, has not been considered so far in the literature. The present paper closes this gap. In general, as had to be expected, our results differ quite a lot from those obtained in the above cited papers. However, in the context of leaf languages taken from a complexity class we will see that finite automata as underlying model are essentially as good as polynomial-time Turing machines.

2 Preliminaries

We assume the reader is familiar with the basic automata and machine models from formal language theory and complexity theory, see, e. g., [HU79, BDG95, BC94, Pap94]. For more background on the models we use, we refer the reader to the different chapters in [RS97].

Our Turing machines are standard multi-tape machines, see [HU79]. For the definition of sublinear time classes we use *indexing machines*, introduced in [Sip83]. These machines cannot directly access their input tape, but instead have to write down a number in binary on a so called index tape. When they enter a specified read state with $\mathrm{bin}(i)$ on the index tape, they are supplied with the ith input symbol (or a particular blank symbol, if i exceeds the input length) in unit time. We use the so called *standard* (or, *proviso $\mathcal{U}$) model* which does not delete its index tape after a read operation, see [CC95, RV97]. This means that, even with a logarithmic time-bound, such a machine may access logarithmically many bits of its input; this fact allows it to determine the length of the input using one-sided binary search.

In our main proof we make use of a generalized model of automata, known as alternating finite automata (AFA). They were introduced by Chandra, Kozen, and Stockmeyer in [CKS81] and work like the better known alternating Turing machines. Although the model at first sight seems to be more powerful than deterministic automata, it was shown that the class of languages they accept is REG [CKS81].

The following, somewhat intuitive, exposition is basically from [Yu97], for a more precise definition of the model refer to [CKS81].

Let $\mathcal{B} = \{0,1\}$ and Q be a finite set. Then $\mathcal{B}^Q$ is the set of all mappings from Q into $\mathcal{B}$. Note that $u \in \mathcal{B}^Q$ can be considered as a $|Q|$-dimensional vector with entries in $\mathcal{B}$.

An *alternating finite automaton (AFA)* is a quintuple $A = (Q, \Sigma, s, F, g)$ where Q is the finite set of states, Σ is the input alphabet, $s \in Q$ is the starting state, $F \subseteq Q$ is the set of final states and g is a function from Q into the set of all functions from $\Sigma \times \mathcal{B}^Q$ into $\mathcal{B}$.

Note that $g(q)$, for $q \in Q$, is a function from $\Sigma \times \mathcal{B}^Q$ into $\mathcal{B}$, denoted below by g_q.

How does an AFA work? Inductively, we define the language accepted by a state $q \in Q$ as follows: A state $q \in Q$ accepts the empty word λ, if and only if $q \in F$. Having a nontrivial input $x = ay$, $a \in \Sigma$, $y \in \Sigma^*$, q reads the first letter a and calls all states to work on the rest y of the input. The states working on y will accept or reject and those results can be described by a vector $u \in \mathcal{B}^Q$. Now the value $g_q(a,u) \in \mathcal{B}$ shows whether q accepts or rejects. An AFA A accepts an input when the initial state s accepts it.

One form to represent an alternating finite automaton $A = (Q, \Sigma, s, F, g)$ is to give a system of equations, for all $q \in Q$, of the form

$$X_q = \sum_{a \in \Sigma} a \cdot g_q(a, X) + \varepsilon_q, \quad q \in Q.$$

X_q represents the state $q \in Q$ and X is the vector of all variables X_q. The final ε_q is used to denote if q is accepting: If $\varepsilon_q = \lambda$ then q is an accepting state; otherwise we set $\varepsilon_q = 0$. The reader may think of the symbol 0 as used here in these equations by convention for "rejection"; we do not want to imply anything else from its use – in particular, 0 need not be an alphabet letter. (This does not say, of course, that this is an arbitrary convention; that there is good reason to use 0 here can be seen from the extensive treatment of the equation calculus in [Yu97].)

In the equation $X_q = a \cdot X_r + b \cdot (X_r \wedge \overline{X_s}) + c \cdot 0$, for example, q is not an accepting state. In this state there is a deterministic transition into r when reading an a. State q definitely rejects when reading a c. If a b is read then q will accept if and only if r accepts the rest of the input and s rejects it.

It is clear that one obtains a nondeterministic automaton with a system of equations in which only the $\vee$ function occurs. A more detailed elaboration of this topic and a proof of the following statement is given in [CKS81, Yu97].

Proposition 2.1 *The class of languages accepted by alternating finite automata is REG.*

3 Leaf Automata

In this section we will formally define *finite automata with generalized acceptance criterion*.

The basic model we use is that of nondeterministic finite automata. On an input word w such a device defines a tree of possible computations. We want to consider this tree, but with a natural order on the leaves. Therefore we make the following definition:

A *finite leaf automaton* (leaf automaton for short) is a tuple $M = (Q, \Sigma, \delta, s, \Gamma, v)$, where

- Σ is an alphabet, the *input alphabet*;
- Q is the finite set of *states*;
- $\delta\colon Q \times \Sigma \to Q^+$ is the *transition function*;
- $s \in Q$ is the *initial state*;
- Γ is an alphabet, the *leaf alphabet*;
- $v\colon Q \to \Gamma$ is a function that associates a state q with its *value* $v(q)$.

If we contrast this with the definition of nondeterministic finite automata, where we have that $\delta(q,a)$ is a *set* of states, we here additionally fix an ordering on the possible successor states by arranging them in a string from Q^+. We explicitly remark that in leaf automata we allow the same state to appear more than once as a successor in $\delta(q,a)$; an example is the automaton $\widehat{N}$ in the proof of Theorem 4.1.

Let M be as above. The computation tree $T_M(w)$ of M on input w is a labeled directed rooted tree defined as follows:

1. The root of $T_M(w)$ is labeled (s,w).
2. Let i be a node in $T_M(w)$ labeled by (q,x), where $x \neq \lambda$, $x = ay$ for $a \in \Sigma$, $y \in \Sigma^*$. Let $\delta(q,a) = q_1 q_2 \cdots q_k$. Then i has k children in $T_M(w)$, and these are labeled by $(q_1,y),(q_2,y),\ldots,(q_k,y)$ in this order.

We now consider an extension $\delta^*\colon Q \times \Sigma^* \to Q^+$ of the transition function δ as follows:

1. $\delta^*(q,\lambda) = q$ for all $q \in Q$.
2. $\delta^*(q,ay) = \delta^*(q_1,y)\delta^*(q_2,y)\cdots\delta^*(q_k,y)$, if $q \in Q$, $a \in \Sigma$, $y \in \Sigma^*$, and $\delta(q,a) = q_1 q_2 \cdots q_k$.

Let $\hat{v}\colon Q^+ \to \Gamma^+$ be the homomorphic extension of v. If now $w \in \Sigma^*$, then $\mathrm{leafstring}^M(w) =_{\mathrm{def}} \hat{v}(\delta^*(s,w))$ is the *leaf string* of M on input w.

If we look at the tree $T_M(w)$ and attach the symbol $v(q)$ to a leaf in this tree with label (q,ε), then $\mathrm{leafstring}^M(w)$ is the string of symbols attached to the leaves, read from left to right in the order induced by δ.

As an example, suppose $M = (\Sigma, Q, \delta, s, F)$ is a usual non-deterministic finite automaton, where $F \subseteq Q$ is the set of accepting states. Define a leaf automaton $M' = (Q, \Sigma, \delta', s, \Gamma, v)$, where $\Gamma = \{0,1\}$, $v(q) = 1$ if $q \in F$ and $v(q) = 0$ else, and $\delta'(q,a)$ is concatenation of the elements of the set $\delta(q,a)$ ordered arbitrarily. Then obviously, M accepts input w if and only if $\mathrm{leafstring}^{M'}(w)$ contains at least one letter 1, i. e., $\mathrm{leafstring}^{M'}(w) \in 0^*1(0+1)^*$. Conversely, every leaf automaton with $\Gamma = \{0,1\}$ may be thought of as a non-deterministic finite automaton.

In the above example we used the language $0^*1(0+1)^*$ as acceptance criterion. We want to use arbitrary languages below. Therefore we define: Let $M = (\Sigma, Q, \delta, s, \Gamma, v)$ be a leaf automaton, and let $A \subseteq \Gamma^*$. The language $\mathsf{Leaf}^M(A) =_{\mathrm{def}} \left\{ w \in \Sigma^* \;\middle|\; \mathrm{leafstring}^M(w) \in A \right\}$ is the language accepted by M with acceptance criterion A. The class $\mathsf{Leaf}^{\mathrm{FA}}(A)$ consists of all languages $B \subseteq \Sigma^*$, for which there is a leaf automaton

M with input alphabet Σ and leaf alphabet Γ such that $B = \mathsf{Leaf}^M(A)$. If $\mathcal{C}$ is a class of languages then $\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C}) =_{\mathrm{def}} \bigcup_{A \in \mathcal{C}} \mathsf{Leaf}^{\mathrm{FA}}(A)$.

Our example above shows that $\mathsf{Leaf}^{\mathrm{FA}}(0^*1(0+1)^*) = \mathrm{REG}$. It will be our aim in the upcoming section to identify $\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})$ for different classes $\mathcal{C}$.

We will also consider a more restricted form of leaf automata, defined as follows:

Let $M = (\Sigma, Q, \delta, s, \Gamma, v)$ be such that $|\delta(q,a)| \leq 2$ for all $q \in Q$ and $a \in \Sigma$; that is, in every step M has at most two possible successor states. In terms of the computation tree $T_M(x)$ this means that leaves trivially have no successors and inner nodes have either one or two successors. Observe that all paths have length exactly n. Thus a path is given by a word $p \in \{\mathrm{L},\mathrm{R}\}^n$, describing how one has to move from the root to the leaf (L stands for left, R for right). Since there may be inner nodes in $T_M(x)$ with only one successor (which, by definition, then is considered the *left* successor), there maybe words $q \in \{\mathrm{L},\mathrm{R}\}^n$ with no corresponding path. In this case we say that *the path q is missing*. We say that the computation tree $T_M(x)$ is *balanced*, if the following holds: There is a path $p \in \{\mathrm{L},\mathrm{R}\}^n$ in $T_M(x)$ such that to the left of p no path is missing, and to the right of p all paths are missing. Thus p is the rightmost path in $T_M(x)$, and $T_M(x)$ informally is a complete binary tree with a missing subpart in the right.

For $A \subseteq \Gamma^*$, the class $\mathsf{BLeaf}^{\mathrm{FA}}(A)$ consists of all languages $B \subseteq \Sigma^*$, for which there is a leaf automaton M with input alphabet Σ and leaf alphabet Γ such that

1. For all input words $w \in \Sigma^*$, the computation tree $T_M(w)$ is balanced; and
2. $B = \mathsf{Leaf}^M(A)$.

We will compare the classes $(\mathsf{B})\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})$ with $(\mathsf{B})\mathsf{Leaf}^{\mathrm{P}}(\mathcal{C})$, $(\mathsf{B})\mathsf{Leaf}^{\mathrm{L}}(\mathcal{C})$, $(\mathsf{B})\mathsf{Leaf}^{\mathrm{LOGT}}(\mathcal{C})$ (the classes of languages definable with leaf languages taken from $\mathcal{C}$ as acceptance criterion for (balanced) nondeterministic Turing machines operating respectively in polynomial time, logarithmic space, and logarithmic time), and $(\mathsf{B})\mathsf{Leaf}^{\mathrm{NC}^1}(\mathcal{C})$ (languages definable with leaf languages taken from $\mathcal{C}$ as acceptance criterion for so called programs over automata, a model which corresponds to the circuit class NC^1 [BIS90]; our $(\mathsf{B})\mathsf{Leaf}^{\mathrm{FA}}$-model can be obtained from this latter $\mathsf{Leaf}^{\mathrm{NC}^1}$-model by omitting the programs but taking only finite automata). For background on these models, we refer the reader to [HLS+93, JMT96, CMTV98].

4 Acceptance Criteria Given by a Complexity Class

We first turn to leaf languages defined by time- or space-bounded Turing machines.

Theorem 4.1 *Let $t(n) \geq \log n$. Then $\mathsf{BLeaf}^{FA}\big(ATIME(t(n))\big) = ATIME\big(t(2^n)\big)$.*

Proof. The proof uses standard padding arguments.

($\supseteq$): Let $A \in \mathrm{ATIME}\big(t(2^n)\big)$ via Turing machine M. Define the leaf automaton $\widehat{N} = (\Sigma, Q, \delta, s, \Sigma, v)$, where $Q = \{s\} \cup \{q_a \,|\, a \in \Sigma\}$, $v(q_a) = a$ for all $a \in \Sigma$, and δ is given as follows:

$$\begin{array}{rcll} \delta(s,a) & = & sq_a & \text{for all } a \in \Sigma, \quad \text{and} \\ \delta(q_b,a) & = & q_bq_b & \text{for all } a,b \in \Sigma. \end{array}$$

The reader may check that, given input $x = x_1 \cdots x_n$, $\widehat{N}$ produces the leaf word

$$v(s)x_n x_{n-1}^2 x_{n-2}^4 x_{n-3}^8 \cdots x_1^{2^{n-1}};$$

hence the ith symbol of x is equal to the 2^{n+1-i}th symbol in $\text{leafstring}^{\widehat{N}}(x)$. It is clear that the computation tree of $\widehat{N}$ is always balanced.

Now define the indexing machine M' operating essentially as M, but when M reads its ith input symbol then M' reads its 2^{n+1-i}th input symbol. To simulate M's read operations, M' on input of length 2^m (corresponding to input $x_1 \cdots x_m$ of machine M) first initializes its index tape with the string 10^m. Now head movements of M can easily be simulated by adjusting the index tape (movements to the right correspond to deleting a 0, movements to the left to adding a 0). M' thus accepts a leaf word $v(s)x_m x_{m-1}^2 x_{m-2}^4 x_{m-3}^8 \cdots x_1^{2^{m-1}}$ of $\widehat{N}$ if and only if $x_1 \cdots x_m \in A$. Machine M' operates in time $t(2^m)$ with respect to input length $n = 2^m$, hence $A \in \mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{ATIME}(t(n))\big)$.

($\subseteq$): Let $A \in \mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{ATIME}(t(n))\big)$; let N be the corresponding leaf automaton, and let M be the Turing machine accepting the leaf language in time t. Define M' as follows: M' works as M, but when M reads its ith input symbol, M' guesses the input bit and then branches universally. On one of these branches the simulation of M is continued, on the other branch N on its ith path is simulated deterministically. This is possible, since the computation tree is balanced, and hence the number i written down in binary immediately gives the nondeterministic choices of N on the ith path. The time requirements for this simulation are given by the time bound of machine M (i.e., $t(2^n)$ for an input of M' of length n) *plus* time $O(n)$ for a single simulation of N. ❑

At this point, two remarks are in order. First, observe that, for the left-to-right inclusion, to obtain the time bound $t(2^n)$ for machine M' we make essential use of its ability to branch existentially and universally; hence this works only for alternating machines. Second, for the above simulation it is necessary that the computation tree produced by N is balanced, because we have to find the ith path of N, given only number i. Next we want to examine what happens when these requirements no longer hold.

Let us first address the case of deterministic machines, i.e., we no longer have the power of alternation, used above to guess and verify. We will see that, nevertheless, a statement similar to the above can be proved if the resource bound class is closed under addition and multiplication.

Theorem 4.2 *Let $t(n) \geq \log n$. Then $\mathsf{BLeaf}^{FA}\big(DTIME\big((t(n))^{O(1)}\big)\big) = DTIME\big((t(2^n))^{O(1)}\big)$.*

Proof. The proof is similar to the above one. For the right to left inclusion, just replace ATIME by DTIME in the above argument.

For the left to right inclusion, let $A \in \mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{DTIME}\big((t(n))^{O(1)}\big)\big)$ via the leaf automaton N and Turing machine M accepting the leaf language. As in the proof of Theorem 4.1 we define M' to work as M, but this time, when M reads its ith input symbol, we interrupt the simulation of M, simulate N on its ith path to compute the input symbol of M, and then resume the simulation of M. These subprograms for simulations of N will lead to an extra factor of $t(n)$ w.r.t. the time requirements of M', which poses no problem here. Note that above, in Theorem 4.1, we did not have this extra time available, hence the other form of simulation there, making use of the power of alternation. ❑

Next we turn to non-balanced computation trees. Given the position of a symbol of the leaf string now no longer enables us to immediately follow the corresponding path in the leaf automaton, because the relation between the position and the nondeterministic choices on the corresponding path is not valid any more. However, as soon as t is at least linear, this is no longer needed as we observe next.

Let us also consider the case of unbalanced trees where the automata we consider have the property that $|\delta(q,a)| \leq 2$ for all $q \in Q$ and $a \in \Sigma$. Our notation for the obtained classes is $\mathsf{Leaf}_2^{\mathrm{FA}}(\cdot)$.

Theorem 4.3 *Let $t(n) \geq n$. Then we have:*

1. $\mathsf{BLeaf}^{FA}\big(DTIME(t(n))\big) = \mathsf{Leaf}_2^{FA}\big(DTIME(t(n))\big) = DTIME\big(t(2^n)\big)$.
2. $\mathsf{Leaf}^{FA}\big(DTIME(t(n))\big) = DTIME\big(t(2^{O(n)})\big)$.

Proof. Similar to the above.

To prove all of the inclusions $\mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big) \subseteq \mathrm{DTIME}\big(t(2^n)\big)$, $\mathsf{Leaf}_2^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big) \subseteq \mathrm{DTIME}\big(t(2^n)\big)$, and $\mathsf{Leaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big) \subseteq \mathrm{DTIME}\big(t(2^{O(n)})\big)$, it is sufficient to observe that we now have enough time to first compute the whole leaf word (for statement 1 in time 2^n, note that we have at most binary branches in the finite automaton; for statement 2 in time $2^{O(n)}$) and then simulate M (in time $t(2^n)$, resp., $t(2^{O(n)})$).

The converse inclusions in statement 1, $\mathrm{DTIME}\big(t(2^n)\big) \subseteq \mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big)$ and $\mathrm{DTIME}\big(t(2^n)\big) \subseteq \mathsf{Leaf}_2^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big)$, are proven exactly as before.

For the inclusion $\mathrm{DTIME}\big(t(2^{O(n)})\big) \subseteq \mathsf{Leaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big)$ we also proceed along the same line, but, if necessary, we replace automaton $\widehat{N}$ from Theorem 4.1 by an automaton with a higher branching degree to pad a length n input to a length 2^{cn} string for suitable $c \in \mathbb{N}$. ❑

In fact the above result can be generalized to many familiar complexity measure. In particular, let Φ be one of the measures $\mathrm{DTIME}, \mathrm{NTIME}, \mathrm{DSPACE}, \mathrm{NSPACE}, \Sigma_k\mathrm{TIME}, \bigoplus\mathrm{TIME}, \ldots$. Let $t(n) \geq n$ in case of a time-restriction, and $t(n) \geq \log n$ in case of a space-restriction. The proof given for Theorem 4.3 remains valid in the case of these measures and bounds; hence we conclude that

$$\begin{aligned} \mathsf{BLeaf}^{\mathrm{FA}}\big(\Phi(t(n))\big) &= \Phi\big(t(2^n)\big), \\ \mathsf{Leaf}^{\mathrm{FA}}\big(\Phi(t(n))\big) &= \Phi\big(t(2^{O(n)})\big). \end{aligned}$$

More generally, using Hertrampf's locally definable acceptance types [Her92, Her94], we conclude that

$$\begin{aligned} \mathsf{BLeaf}^{\mathrm{FA}}\big((\mathcal{F})\mathrm{TIME}(t(n))\big) &= (\mathcal{F})\mathrm{TIME}\big(t(2^n)\big), \\ \mathsf{Leaf}^{\mathrm{FA}}\big((\mathcal{F})\mathrm{TIME}(t(n))\big) &= (\mathcal{F})\mathrm{TIME}\big(t(2^{O(n)})\big), \end{aligned}$$

for any locally definable acceptance type $\mathcal{F}$.

Hence we obtain in particular:

Corollary 4.4

1. $\mathsf{BLeaf}^{FA}(POLYLOGTIME) = P$.
2. $\mathsf{BLeaf}^{FA}(NC^1) = ALINTIME$.
3. $\mathsf{BLeaf}^{FA}(L) = \mathsf{Leaf}^{FA}(L) = LIN$.
4. $\mathsf{BLeaf}^{FA}(NL) = \mathsf{Leaf}^{FA}(NL) = NLIN$.
5. $\mathsf{BLeaf}^{FA}(POLYLOGSPACE) = \mathsf{Leaf}^{FA}(POLYLOGSPACE) = PSPACE$.
6. $\mathsf{BLeaf}^{FA}(P) = \mathsf{Leaf}^{FA}(P) = E$.
7. $\mathsf{BLeaf}^{FA}(NP) = \mathsf{Leaf}^{FA}(NP) = NE$.

The above proofs make use of fairly standard padding techniques. The main point is the definition of an automaton which pads a given word of length n into a word of length 2^n (or $2^{O(n)}$ in the unbalanced case). Turing machines and Boolean circuits can pad up to length $2^{n^{O(1)}}$, therefore similar proofs show that, e. g., the classes $\mathsf{Leaf}^{\mathrm{P}}(\mathrm{NC}^1)$, $\mathsf{Leaf}^{\mathrm{L}}(\mathrm{NC}^1)$, $\mathsf{Leaf}^{\mathrm{NC}^1}(\mathrm{NC}^1)$, $\mathsf{Leaf}^{\mathrm{P}}(\mathrm{POLYLOGSPACE})$, $\mathsf{Leaf}^{\mathrm{L}}(\mathrm{POLYLOGSPACE})$, and $\mathsf{Leaf}^{\mathrm{NC}^1}(\mathrm{POLYLOGSPACE})$ coincide with $\mathrm{ATIME}(n^{O(1)}) = \mathrm{PSPACE}$, see [HLS+93, JMT96, CMTV98]. Hence we see that here in the context of complexity classes as leaf languages, the ability to pad is the central point, and Turing machines, Boolean circuits, and finite automata behave quite similarly.

5 Acceptance Criteria Given by a Formal Language Class

We now consider in turn the different classes that make up the Chomsky hierarchy of formal languages.

5.1 Regular Languages

One can easily see that REG is defined by the regular leaf language $0^*1(0+1)^*$, but already the language $\{1\}$ over $\mathcal{B} = \{0,1\}$ defines REG as the following proof shows. Furthermore, we show next in our main result that a regular leaf language cannot define a class containing nonregular languages.

Theorem 5.1 *$\mathsf{BLeaf}^{FA}(REG) = \mathsf{Leaf}^{FA}(REG) = REG$.*

Proof. The inclusion $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{REG}) \subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{REG})$ is trivial. To show $\mathrm{REG} \subseteq \mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{REG})$ we define the leaf language $B = \{1\} \in \mathrm{REG}$ over $\mathcal{B} = \{0,1\}$. Let $A \in \mathrm{REG}$ be given. Then there exists a DFA N which accepts A. We use N as the leaf automaton producing the leaf string 1 or 0 when accepting or rejecting. Thus we have: $x \in A \iff \mathrm{leafstring}^N(x) = 1 \iff \mathrm{leafstring}^N(x) \in B$. Of course, the computation tree of N is always balanced.

Finally we have to show $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{REG}) \subseteq \mathrm{REG}$. Let $A \in \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{REG})$ be a language over the alphabet Σ. Then there exist a DFA M and a leaf automaton N with the following property: $x \in A \iff M$ accepts $\mathrm{leafstring}^N(x)$. Let the automata N and M be given by $N = (\Sigma, Q_N, \delta_N, s_N, \Gamma, v)$ and $M = (\Gamma, Q_M, \delta_M, s_M, F_M)$. For $q \in Q_N$ and $a \in \Sigma$ we denote the branching degree by $r(q,a) = |\delta_N(q,a)|$ and write $\delta_N(q,a) = \delta_{N,1}(q,a)\ldots\delta_{N,r(q,a)}(q,a)$.

We construct an AFA $\widehat{M} = (\Sigma, Q_{\widehat{M}}, s_{\widehat{M}}, F_{\widehat{M}}, g)$ which accepts A. The set of states is defined by $Q_{\widehat{M}} = \{s_{\widehat{M}}\} \cup (Q_M \times Q_M \times Q_N)$. In the sequel we will denote a state $q_{\widehat{M}} \in Q_{\widehat{M}} \setminus \{s_{\widehat{M}}\}$ by a triple, e. g., $q_{\widehat{M}} = (q_0, q_e, q_N)$, with the following intuition: When starting in $q_{\widehat{M}}$, $\widehat{M}$ will accept if and only if the leaf string produced by the leaf automaton N starting in q_N leads M from q_0 to q_e. $\widehat{M}$ follows the computation of N, while it guesses an accepting sequence of states of M. At the end $\widehat{M}$ checks whether this sequence coincides with the sequence of states one gets when following M working on the leaf string. This will be done by using the principle of "divide and conquer."

We define the function g as well as the set of final states $F_{\widehat{M}}$ by systems of equations as described in Sect. 2 (note that '$+$' and '$\cdot$' are parts of the equation formalism, while '$\vee$' and '$\wedge$' are used to specify Boolean functions):

$$s_{\widehat{M}} = \sum_{x \in \Sigma} x \cdot g_{s_{\widehat{M}},x} + \varepsilon_{s_{\widehat{M}}} \quad \text{with} \quad \varepsilon_{s_{\widehat{M}}} = \begin{cases} \lambda & \text{if } \delta_M\Big(s_M, v(s_N)\Big) \in F_M, \\ 0 & \text{otherwise,} \end{cases}$$

$$g_{s_{\widehat{M}},x} = \bigvee_{q_1,\ldots,q_{r-1}\in Q_M,\, q_r\in F_M} \left[\bigwedge_{i=1}^{r} \Big(q_{i-1}, q_i, \delta_{N,i}(s_N,x) \Big) \right],$$
$$q_0 = s_M, \text{ and } r = r(s_N,x).$$

Note that the branching degree r depends on the state and the letter of the input, so the value of r might differ for different x in $g_{s_{\widehat{M}},x}$. Remember that $s_{\widehat{M}} \in F_{\widehat{M}} \iff \varepsilon_{s_{\widehat{M}}} = \lambda$. The "divide and conquer" approach is directly reflected by the syntactic shape of the Boolean functions $g_{s_{\widehat{M}},x}$ (and $g_{q_{\widehat{M}},x}$ below): Similar to, e.g., the proof of Savitch's Theorem [BDG95, Theorem 2.27] or the proof of the PSPACE-completeness of QBF [BDG95, Theorem 3.29], the disjunctive normal-form expresses that *there are* "intermediate states" $q_1,\ldots,q_{r-1}$ such that *for all* these states, the corresponding subcomputations are valid.

Next, for $q_{\widehat{M}} = (q_0,q_e,q_N) \in Q_{\widehat{M}}$ we define:

$$q_{\widehat{M}} = \sum_{x\in\Sigma} x \cdot g_{q_{\widehat{M}},x} + \varepsilon_{q_{\widehat{M}}} \quad \text{with} \quad \varepsilon_{q_{\widehat{M}}} = \begin{cases} \lambda & \text{if } \delta_M\Big(q_0, v(q_N)\Big) = q_e, \\ 0 & \text{otherwise,} \end{cases}$$

$$g_{q_{\widehat{M}},x} = \bigvee_{q_1,\ldots,q_{r-1}\in Q_M} \left[\bigwedge_{i=1}^{r-1} \Big(q_{i-1}, q_i, \delta_{N,i}(q_N,x) \Big) \wedge \Big(q_{r-1}, q_e, \delta_{N,r}(q_N,x) \Big) \right],$$
$$\text{and } r = r(q_N,x).$$

Again, r depends on q_N and x, and we have $q_{\widehat{M}} \in F_{\widehat{M}} \iff \varepsilon_{q_{\widehat{M}}} = \lambda$.

Now we must show that the alternating automaton $\widehat{M}$ accepts the language $\mathrm{L}(\widehat{M}) = A$. The state $q_{\widehat{M}} = (q_0,q_e,q_N) \in Q_{\widehat{M}}$ has the following intuitive meaning: Starting NFA N in state q_N on the input y we obtain a leaf string w. Starting $\widehat{M}$ in $q_{\widehat{M}}$, the input y will be accepted if and only if this leaf string leads M from state q_0 to q_e, i. e., if $\hat{\delta}(q_0,w) = q_e$. We prove this by induction on y:

$|y| = 0$: For $y = \lambda$ the leaf string is the letter $v(q_N)$. Starting in $q_{\widehat{M}} = (q_0,q_e,q_N)$, $y = \lambda$ will be accepted if and only if $\varepsilon_{q_{\widehat{M}}} = \lambda$. This is true for $\delta_M\Big(q_0, v(q_N)\Big) = q_e$, i. e., the leaf string $v(q_N)$ leads M from q_0 to q_e.

Assuming this to be correct for all $y \in \Sigma^*$, $|y| < n$, we now consider the case $|y| = n$: Let $q_{\widehat{M}} = (q_0,q_e,q_N)$ be the current state of $\widehat{M}$ and $y = y_1 \cdots y_n$. In state q_N, N branches into $r = r(q_N,y_1) = |\delta_N(q_N,y_1)|$ subtrees when it reads y_1. According to the equation for $g_{q_{\widehat{M}},y_1}$, $\widehat{M}$ in state $q_{\widehat{M}}$ accepts y if and only if there exists a sequence of states $q_1,\ldots,q_{r-1} \in Q_M$ with the following property: In each subtree i (r resp.), $i = 1,\ldots,r-1$, the word $y_2 \cdots y_n$ will be accepted when starting respectively in state $\Big(q_{i-1}, q_i, \delta_{N,i}(q_N,y_1)\Big)$ or $\Big(q_{r-1}, q_e, \delta_{N,r}(q_N,y_1)\Big)$. Following our induction assumption this is true if and only if in each subtree M is transduced from q_{i-1} to q_i (from q_{r-1} to q_e resp.) by the corresponding leaf string. Thus $\widehat{M}$ accepts y starting in $q_{\widehat{M}}$ if and only if M is lead from q_0 to q_e by the whole leaf string.

Analogously, $\widehat{M}$ accepts the input y, $|y| > 0$, starting from $s_{\widehat{M}}$ if there is additionally to the states $q_i \in Q_M$ an accepting state $q_r \in F_M$ such that $\delta^*_M(s_M, \text{leafstring}^N(y)) = q_r$. If $y = \lambda$ then N produces the single letter leaf string $v(s_N)$, and we have:

$$\lambda \in A \iff M \text{ accepts } v(s_N) \iff \delta_M\Big(s_M, v(s_N)\Big) \in F_M \iff \varepsilon_{s_{\widehat{M}}} = \lambda \iff \widehat{M} accepts y = \lambda. \quad (1)$$

Thus we have $\mathrm{L}(\widehat{M}) = A$. ❑

The result just given is dramatically different from corresponding results for other models: It is known that $\mathsf{Leaf}^{\mathrm{P}}(\mathrm{REG}) = \mathrm{PSPACE}$ and $\mathsf{Leaf}^{\mathrm{L}}(\mathrm{REG}) = \mathrm{P}$.

5.2 Contextfree Languages

We found REG to be closed under the $\mathsf{Leaf}^{\mathrm{FA}}$-operator, but it is well known that REG is closed under many operations. What about the other classes in Chomsky's hierarchy, e.g., CFL? First we show in Lemma 5.3 that every class defined via leaf languages is closed under intersection if the class of leaf languages is closed under a certain type of concatenation. Then it will be easy to see that CFL is not closed under the $\mathsf{Leaf}^{\mathrm{FA}}$-operator. Furthermore we give some arguments for an upper bound of $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$.

First, however, we observe the following:

Lemma 5.2 *$CFL \subseteq \mathsf{BLeaf}^{FA}(CFL)$.*

Proof. It is known that for every $L \in \mathrm{CFL}$ over some alphabet Σ and every $\$ \notin \Sigma$, the language $L^{\$} = \{\, a_1\$a_2\$a_3\$\cdots\$a_n \mid a_1,\ldots,a_n \in \Sigma \cup \{\lambda\}, a_1\cdots a_n \in L\}$, the so called *padded version* of L, is in CFL.

By an easy modification of automaton $\widehat{N}$ from the proof of Theorem 4.1 we obtain a leaf automaton M that, given an input $a_1\cdots a_n$, produces a full binary computation tree whose leaf string is of the form $\$^*a_1\$^*\cdots\$^*a_n\$^*$. Hence, M with leaf language $L^{\$}$ accepts L. ❑

Lemma 5.3 *Let $\mathcal{C}$ be a class of languages with the following properties:*

1. *$L_1, L_2 \in \mathcal{C} \implies L_1\#L_2 \in \mathcal{C}$, where # is a new symbol and*
2. *$L \in \mathcal{C} \implies L \cup \{\lambda\} \in \mathcal{C}$.*

Then $\mathsf{Leaf}^{FA}(\mathcal{C})$ is closed under intersection.

Proof. Let $A = \mathsf{Leaf}^{M_A}(L_A)$ and $B = \mathsf{Leaf}^{M_B}(L_B)$ with the leaf automata M_A, M_B (where, w.l.o.g., we assume that the state sets of M_A and M_B are disjoint) and the leaf languages $L_A, L_B \in \mathcal{C}$ over the alphabets Σ_A, Σ_B. Construct a leaf automaton M' with $\text{leafstring}^{M'}(x) = \text{leafstring}^{M_A}(x)\#\text{leafstring}^{M_B}(x)$ for all $x \neq \lambda$ and the new symbol $\# \notin \Sigma_A \cup \Sigma_B$ in the following way: The set of states of M' consists of the states of M_A and M_B, a new initial state s with the value $v(s) = \#$, and a new state m producing the leaf string # on every input. In state s there is a nondeterministic transition into m and the successors of the initial states of M_A and M_B, where m is in the middle of these three states according to the order of the computation tree. In all other states M' works just like M_A or M_B, respectively. For all $x \neq \lambda$ we obtain

$$\begin{aligned} x \in A \cap B \quad &\iff \quad \text{leafstring}^{M_A}(x) \in L_A \text{ and } \text{leafstring}^{M_B}(x) \in L_B \\ &\iff \quad \text{leafstring}^{M_A}(x)\#\text{leafstring}^{M_B}(x) \in L_A\#L_B \\ &\iff \quad \text{leafstring}^{M'}(x) \in L_A\#L_B. \end{aligned}$$

For the special case of $x = \lambda$ we now define:

$$L'_A = \begin{cases} L_A \cup \{\lambda\} & \text{if } \lambda \in A \cap B \text{ and} \\ L_A & \text{else.} \end{cases}$$

Analogously we define L'_B and we get $\lambda \in A \cap B \iff \text{leafstring}^{M'}(\lambda) = v(s) = \# \in L'_A\#L'_B$. Now we have $A \cap B = \mathsf{Leaf}^{M'}(L'_A\#L'_B) \in \mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})$. ❑

The second assumption in Lemma 5.3 is just for some technical reasons concerning the empty word and could be easily replaced e.g. by $L \in \mathcal{C} \implies L \setminus \{\lambda\} \in \mathcal{C}$.

It is well known that CFL is not closed under intersection, but it fulfills the prerequisites of the previous lemma. Thus we know that $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ contains all intersections of contextfree languages and $\mathrm{CFL} \subsetneq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$. We also want to give some upper bounds for $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$: We know that $\mathrm{CFL} \subseteq \mathrm{P}$ and $\mathrm{CFL} \subseteq \mathrm{DSPACE}(\log^2(n))$. By monotonicity and our results in Sect. 4 we obtain

Theorem 5.4 *$CFL \subsetneq \mathsf{Leaf}^{FA}(CFL) \subseteq DSPACE(n^2) \cap E$.*

The above proof makes convenient use of the well-known fact that CFL is not closed under intersection; however it only works for unbalanced computation trees. But also in the balanced case it is easy to show that non context-free languages can be accepted by leaf automata with context-free leaf language:

Say that a language class $\mathcal{C}$ is closed under *weak intersection* if, whenever $L_1, L_2 \in \mathcal{C}$, then $\#\#(L_1 \cap L_2) \in \mathcal{C}$, where # is a new symbol not occurring in the alphabets of L_1, L_2.

Lemma 5.5 *Let $\mathcal{C}$ be a class of languages with the following properties:*

1. *If $L \in \mathcal{C}$, $L \subseteq \Sigma^*$, $\$ \notin \Sigma$, then the language $L^\$$ is in $\mathcal{C}$.*
2. *$L_1, L_2 \in \mathcal{C} \implies L_1\#^*L_2 \in \mathcal{C}$, where # is a new symbol and*
3. *$L \in \mathcal{C} \implies L \cup \{\lambda\} \in \mathcal{C}$.*

Then $\mathsf{BLeaf}^{FA}(\mathcal{C})$ is closed under weak intersection.

Proof. We argue in a way similar to the proof of Lemma 5.3. Let A, B, M_A, M_B be as defined there. Let M'_A be as M_A but producing a full binary computation tree for every input by inserting paths that output the neutral letter \$. We construct a leaf automaton M' that on input $\#\#x$ produces the leafstring $\mathrm{leafstring}^{M'}(x) = \mathrm{leafstring}^{M'_A}(x)\#^{2^{n+1}}\mathrm{leafstring}^{M_B}(x)$ for all $x \neq \lambda$. This can easily be achieved as follows: While reading the first two symbols ##, automaton M' produces a full binary tree with four leaves v_1, v_2, v_3, v_4. In v_1 automaton M'_A is started on the rest of the input, i.e., on x; in v_4 automaton M_B is started. Below v_2 and v_3 full binary trees that output the symbol # on every path are produced. Let L'_A be the padded version of L_A, and obtain L''_A and L'_B from L'_A and L_B as in Lemma 5.3. Arguing as above we then obtain that $\#\#(A \cap B) = \mathsf{Leaf}^{M'}(L''_A\#^*L'_B) \in \mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})$. ❑

Theorem 5.6 *$CFL \subsetneq \mathsf{BLeaf}^{FA}(CFL)$.*

Proof. Certainly CFL fulfills the assumptions of Lemma 5.5, hence $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{CFL})$ is closed under weak intersection, but this property is not shared by CFL. ❑

All in all we thus obtain the inclusion chain

$$\mathrm{CFL} \subsetneq \mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathrm{DSPACE}(n^2) \cap \mathrm{E}.$$

5.3 Context-sensitive and Recursively Enumerable Languages

The class of context-sensitive languages CSL has already been treated in Sect. 4, because it can be characterized by linear bounded automata. Thus we have $\mathrm{CSL} = \mathrm{NLIN}$ and together with the results from Sect. 4 we obtain:

Theorem 5.7 $\mathsf{BLeaf}^{FA}(CSL) = NSPACE(2^n)$ *and* $\mathsf{Leaf}^{FA}(CSL) = NSPACE(2^{O(n)})$.

Our last result, that the leaf language class RE defines RE in the balanced as well as in the unbalanced case, is not surprising:

Theorem 5.8 $\mathsf{BLeaf}^{FA}(RE) = \mathsf{Leaf}^{FA}(RE) = RE$.

Proof. $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{RE}) \subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{RE})$ is trivial.

Next, we show $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{RE}) \subseteq \mathrm{RE}$: Let $A = \mathsf{Leaf}^{M}(B)$, where $B \in \mathrm{RE}$ is given by the recursive and onto function $f\colon \mathbb{N} \to B$. The set of all inputs x for which M produces a given leaf string w is also enumerable: Simulate M on every $x \in \Sigma^*$ and if $\mathrm{leafstring}^{M}(x) = w$ then output x. Let $g\colon \mathbb{N} \times \Sigma^* \to \Sigma^*$ be the corresponding recursive enumeration function. Now we use Cantor's dovetailing method to enumerate A, e.g., calculate in this order

$$g\Big(1,f(1)\Big),\quad g\Big(2,f(1)\Big),g\Big(1,f(2)\Big),\quad g\Big(3,f(1)\Big),g\Big(2,f(2)\Big),g\Big(1,f(3)\Big)\ldots$$

Finally, since RE is closed under padding with a neutral letter \$, the proof of $\mathrm{RE} \subseteq \mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{RE})$ is the same as for context-free languages. ❑

6 Conclusion

We examined the acceptance power of nondeterministic finite automata with different kinds of leaf languages. Comparing our results with those known for nondeterministic Turing machines with leaf language acceptance, we saw that if the leaf language class is a formal language class then we obtain a huge difference in computational power, but in the case of a resource-bounded leaf language class the difference between finite automata, Boolean circuits, and Turing machines (almost) disappears. This is due to the fact that in all three cases only the power of the devices to pad out their given input to a long leaf string is the central point.

It is known that the operator $\mathsf{Leaf}^{\mathrm{LOGT}}(\cdot)$, i. e., leaf languages for nondeterministic logarithmic time-bounded machines, is a *closure operator*: $\mathsf{Leaf}^{\mathrm{LOGT}}(\mathcal{C})$ coincides with the closure of the class $\mathcal{C}$ under DLOGTIME reductions [JMT96]. In the beginning the authors had the hope to be able to show that the operator $\mathsf{Leaf}^{\mathrm{FA}}(\cdot)$ is also some form of closure operator. However, the results from Sect. 4 prove that this is not the case. If $\mathcal{C}$ is a reasonably large enough complexity class, then $\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C}) \subsetneq \mathsf{Leaf}^{\mathrm{FA}}\big(\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})\big)$, hence the operator $\mathsf{Leaf}^{\mathrm{FA}}(\cdot)$ lacks the property of being closed. In this sense, the $\mathsf{Leaf}^{\mathrm{FA}}$-model is even more complicated than the $\mathsf{Leaf}^{\mathrm{LOGT}}$-model.

The main remaining open question of course is if the upper and lower bounds obtained in this paper for $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ can be strengthened. Our results here leave a lot of room for improvement, and certainly one would expect to be able to give stronger bounds. Nevertheless, we have been unable so far to do so. An idea would be to follow the proof of Theorem 5.1. For each language $A \in \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ one can construct an alternating pushdown automaton which accepts A. But unfortunately this yields not more than $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathrm{E}$, because in [CKS81] Chandra, Kozen, and Stockmeyer showed that the set ALT-PDA of all languages accepted by such automata equals E. One might hope that the lower bound $\mathrm{PSPACE} = \mathsf{Leaf}^{\mathrm{NC}^1}(\mathrm{CFL})$ could be transferred to our context – after all, there is a very strong connection between the class NC^1 and finite automata, since there are regular languages complete for NC^1 under very strict reductions such as uniform projections, see [BIS90]. However our Theorem 5.4 shows that this hope is not justified; we have $\mathrm{PSPACE} \not\subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$.

Acknowledgment. We are grateful to Klaus W. Wagner (Würzburg) and Ulrich Hertrampf (Stuttgart) for helpful discussions. We also acknowledge helpful comments by the anonymous referees.

References

[Bar89] D. A. Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[BC94] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. International Series in Computer Science. Prentice Hall, London, 1994.

[BCS92] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[BDG95] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 2nd edition, 1995.

[BIS90] D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[CC95] L. Cai and J. Chen. On input read-modes of alternating Turing machines. *Theoretical Computer Science*, 148:33–55, 1995.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28:114–133, 1981.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[Her92] U. Hertrampf. Locally definable acceptance types for polynomial time machines. In *Proceedings 9th Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 199–207. Springer Verlag, 1992.

[Her94] U. Hertrampf. Complexity classes defined via k-valued functions. In *Proceedings 9th Structure in Complexity Theory*, pages 224–234. IEEE Computer Society Press, 1994.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA, 1979.

[JMT96] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information and Computation*, 129:21–33, 1996.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[RS97] R. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume I. Springer Verlag, 1997.

[RV97] K. Regan and H. Vollmer. Gap-languages and log-time complexity classes. *Theoretical Computer Science*, 188:101–116, 1997.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[Ver93] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[Yu97] S. Yu. Regular languages. In R. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume I, chapter 2, pages 41–110. Springer Verlag, Berlin Heidelberg, 1997.