Finite Automata with Generalized Acceptance Criteria

Timo Peichl and Heribert Vollmer

Theoretische Informatik, Universität Würzburg,
Am Exerzierplatz 3, D-97072 Würzburg, GERMANY

Abstract. We examine the power of nondeterministic finite automata with acceptance of an input word defined by a so called *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.

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. An node v in this tree corresponds to 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 all 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 correspond 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 corresponding to 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 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 else 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 the set of all binary words with a number of 1's divisible by p as leaf language.

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 [2]), 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., [4, 14, 10, 12]; recently the model has also been applied to Boolean circuits [6]. However, in the context of the probably most basic type of computation device, the finite automaton, leaf languages have not been considered so far. The present paper closes this gap. 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 interestingly finite automata as underlying model are essentially as powerful as polynomial-time Turing machines.

Due to lack of space, most proofs in this abstract have to remain sketchy or are even omitted.

2 Preliminaries

We assume the reader is familiar with basic automata and machine models from formal language theory and complexity theory, see, e. g., [11, 1, 3].

Our Turing machines are standard multi-tape machines, see [11]. For the definition of sublinear time classes we use *indexing machines*. 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. We use

the so called *standard* (or, *proviso $\mathcal{U}$*) *model* which does not delete its index tape after a read operation, see [5, 13].

In our main proof we make use of a generalized model of automata, known as alternating finite automata (AFA). These were introduced by Chandra, Kozen, and Stockmeyer in [7] and work similar to the better known alternating Turing machines. Although the model at first sight seems to be more powerful than deterministic automata, it was shown the class of languages they accept is REG [7]. The following exposition is basically from [15].

Let $\mathcal{B} = \{0, 1\}$ and Q be a set. Then $\mathcal{B}^Q$ is the set of all mappings of 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 $A = (Q, \Sigma, g, s, F)$ 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 $\mathcal{B}$ in the following denoted 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 in parallel. 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 $A = (Q, \Sigma, g, s, F)$ is to give a system of equations, for all $q \in Q$:

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

X_q represents the state $q \in Q$ and X is the vector of all variables X_q. If $\epsilon_q = \lambda$ then q is an accepting state, i. e., for all $q \in Q$:

$$\epsilon_q = \begin{cases} \lambda & \text{if } q \in F \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The equation $X_q = a \cdot X_r + b \cdot (X_r \wedge \overline{X_s}) + c \cdot 0$, for example, shows that q is not an accepting state. In this state there is a deterministic transition into r when reading an a and it 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 [7, 15].

Proposition 1. *The class of languages accepted by alternating finite automata is* REG.

3 Leaf Automata

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 with an order on the leaves. Therefore we make the following definition:

A *finite leaf automaton* 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*; and $v\colon Q \to \Gamma$ is a function that associates with a state q 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^+.

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 v 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 v 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) $\delta^*(q, x) = q$ for all $q \in Q$, if x is the empty word.
2) $\delta^*(q, x) = \delta^*(q_1, y)\delta^*(q_2, y) \cdots \delta^*(q_k, y)$, if $q \in Q$, $x = ay$ for $a \in \Sigma$, $y \in \Sigma^*$, 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 $M = (Q, \Sigma, \delta, s, F)$ 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 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 $M = (Q, \Sigma, \delta, s, \Gamma, v)$ be a leaf automaton, and let $A \subseteq \Gamma^*$. The language $\mathsf{Leaf}^M(A) = \{\, w \in \Sigma^* \mid \mathrm{leafstring}^M(w) \in A \,\}$ 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 $M = (Q, \Sigma, \delta, s, \Gamma, v)$ be such that $|\delta(q,a)| \leq 2$ for all $q \in Q$ and $a \in \Sigma$; that is, at 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, there may be words $q \in \{\mathrm{L},\mathrm{R}\}^n$ with no corresponding path. In this case we say that *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 for all input words $w \in \Sigma^*$, the computation tree $T_M(w)$ is balanced, and $B = \mathsf{Leaf}^M(A)$.

We will compare the classes $\mathsf{Leaf}^{\mathrm{FA}}(\mathcal{C})$ with $\mathsf{Leaf}^{\mathrm{P}}(\mathcal{C})$, $\mathsf{Leaf}^{\mathrm{L}}(\mathcal{C})$, $\mathsf{Leaf}^{\mathrm{LOGT}}(\mathcal{C})$ (the class of languages definable with leaf languages taken from $\mathcal{C}$ as acceptance criterion for nondeterministic Turing machines operating respectively in polynomial time, logarithmic space, and logarithmic time), and $\mathsf{Leaf}^{\mathrm{NC}^1}(\mathcal{C})$ (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 [2]; our $\mathsf{Leaf}^{\mathrm{FA}}$-model can be obtained from this latter $\mathsf{Leaf}^{\mathrm{NC}^1}$-model by omitting the programs but taking only finite automata). Due to space restrictions, we cannot give precise definitions here, but refer the reader to [10, 12, 6].

4 Acceptance Criteria Given by a Complexity Class

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

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

Proof (sketch). ($\supseteq$): Let $A \subseteq \Sigma^*$, $A \in \mathrm{ATIME}\big(t(2^n)\big)$ via Turing machine M. Define the leaf automaton $N = (\Sigma, Q, \delta, s, \Sigma, v)$, where $Q = \{s\} \cup \{q_a \mid a \in \Sigma\}$, $v(q_a) = a$ for all $a \in \Sigma$, and δ is given as follows (cf. also [6]):

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

The reader may check that given input $x = x_1 \cdots x_n$, the automaton 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 leafstring$^N(x)$. It is clear that the computation tree of 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 the language $\mathrm{pad}^N(A) = \{\text{leafstring}^N(x) \mid x \in A\}$ 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 symbol and then branches universally. On one of these branches the simulation of M is continued, on the other branch M' simulates N on its ith path and halts accepting iff the output symbol in this simulation coincides with the previously guessed symbol. This is possible, since the computation tree of N is balanced. □

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. Both requirements are no longer needed as soon as t is at least linear, as we state next.

Theorem 3. *Let* $t(n) \geq n$. *Then* $\mathsf{BLeaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big) = \mathrm{DTIME}\big(t(2^n)\big)$ *and* $\mathsf{Leaf}^{\mathrm{FA}}\big(\mathrm{DTIME}(t(n))\big) = \mathrm{DTIME}\big(t(2^{O(n)})\big)$.

The above result can be generalized to any reasonable complexity measure. Let Φ be one of DTIME, NTIME, DSPACE, NSPACE, Σ_kTIME, $\bigoplus$TIME, ... Let $t(n) \geq n$ in case of a time-restriction, and $t(n) \geq \log n$ in case of a space-restriction. Then $\mathsf{Leaf}^{\mathrm{FA}}\big(\Phi(t(n))\big) = \Phi\big(t(2^{O(n)})\big)$. In fact, using Hertrampf's locally definable acceptance types [8, 9], we obtain $\mathsf{Leaf}^{\mathrm{FA}}\big((\mathcal{F})\mathrm{TIME}(t)\big) = (\mathcal{F})\mathrm{TIME}\big(t(2^{O(n)})\big)$ for any locally definable acceptance type $\mathcal{F}$.

This yields in particular:

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

The above proof makes use of fairly standard padding techniques. The main point is the definition of automaton N which pads a 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 $\mathsf{Leaf}^{\mathrm{NC}^1}(\mathrm{POLYLOGSPACE})$ all coincide with $\mathrm{ATIME}(n^{O(1)}) = \mathrm{PSPACE}$ [10, 12, 6]. 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 similar.

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

Here we state and prove our main result.

Theorem 5. $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{REG}) = \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{REG}) = \mathrm{REG}$.

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: $x \in A \Leftrightarrow \mathrm{leafstring}^N(x) = 1 \Leftrightarrow \mathrm{leafstring}^N(x) \in B$. 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: $x \in A \Leftrightarrow M$ accepts $\mathrm{leafstring}^N(x)$. Let the automata N and M be given by $N = (Q_N, \Sigma, \delta_N, s_N, \Gamma, v)$ and $M = (Q_M, \Gamma, \delta_M, s_M, F_M)$. 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 $L = (Q_L, \Sigma, g, s_L, F_L)$ which accepts A. The set of states is defined by $Q_L = \{s_L\} \cup (Q_M \times Q_M \times Q_N)$. In the sequel we will denote a state $q_L \in Q_L \setminus \{s_L\}$ by a triple, e. g., $q_L = (q_0, q_e, q_N)$, with the following intuition: When starting in q_L, L 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. L follows the computation of N, while it guesses an accepting sequence of states of M. At the end L 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_L by systems of equations as described in Sect. 2:

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

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

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 $g_{s_L,x}$. Remember that $s_L \in F_L \Leftrightarrow \epsilon_{s_L} = \lambda$. For $q_L = (q_0, q_e, q_N) \in Q_L$ we define:

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

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

Again, r depends on q_N and x, and we have $q_L \in F_L \Leftrightarrow \epsilon_{q_L} = \lambda$.

Now we must show that the alternating automaton L accepts the language A. The state $q_L = (q_0, q_e, q_N) \in Q_L$ has the following intuitive meaning: Starting NFA N in state q_N on the input y we obtain a leaf string w. Starting L in q_L, the input y will be accepted if and only if this leaf string leads M from state q_0 to q_e, i. e., in the notation of [11], if $\hat{\delta}_M(q_0, w) = q_e$. We prove this by induction on y:

$|y| = 0$: For $y = \lambda$ the leaf string is the letter $v(q_N)$. Starting in $q_L = (q_0, q_e, q_N)$, $y = \lambda$ will be accepted if and only if $\epsilon_{q_L} = \lambda$. 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 $q_L = (q_0, q_e, q_N)$ be the current state of L 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_L,y_1}, L in state q_L 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 L accepts y starting in q_L if and only if M is lead from q_0 to q_e by the whole leaf string.

Analogously, L accepts the input y, $|y| > 0$, starting from s_L 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: $\lambda \in A \Leftrightarrow M$ accepts $v(s_N) \Leftrightarrow \delta_M\Big(s_M, v(s_N)\Big) \in F_M \Leftrightarrow \epsilon_{s_L} = \lambda \Leftrightarrow L$ accepts $y = \lambda$.

Thus we have $\mathrm{L}(L) = A$. □

The just given result is much 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

Theorem 6. $\mathrm{CFL} \subsetneq \mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathrm{DSPACE}(n^2) \cap \mathrm{E}$.

Proof (sketch). The $\subsetneq$ follows from the fact that we can show $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{CFL})$ is closed under "weak intersection," a property not shared by CFL. The first $\subseteq$ is immediate; the second $\subseteq$ follows from $\mathrm{CFL} \subseteq \mathrm{DSPACE}(\log^2(n)) \cap \mathrm{P}$ and the results given in Sect. 4. □

5.3 Contextsensitive and Recursively Enumerable Languages

Finally we obtain the following (statement 2 of course follows directly from the equality CSL = NLIN and the results given in Sect. 4):

Theorem 7. *1)* $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{RE}) = \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{RE}) = \mathrm{RE}$.
2) $\mathsf{BLeaf}^{\mathrm{FA}}(\mathrm{CSL}) = \mathrm{NSPACE}(2^n)$ *and* $\mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CSL}) = \mathrm{DSPACE}(2^{O(n)})$.

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 $\mathsf{Leaf}^{\mathrm{LOGT}}(\cdot)$ is a *closure operator*: $\mathsf{Leaf}^{\mathrm{LOGT}}(\mathcal{C})$ coincides with the closure of $\mathcal{C}$ under DLOGTIME reductions [12]. In the beginning the authors had the hope to be able to show that also the operator $\mathsf{Leaf}^{\mathrm{FA}}(\cdot)$ is some form of closure operator. However, the results from Sect. 4 prove that this is not the case. If $\mathcal{C}$ is a reasonable, 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 Theorem 6 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. 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 [7] 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 transfered 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 [2]. However our Theorem 6 shows that this hope is not justified; we have PSPACE $\not\subseteq \mathsf{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$.

Acknowledgment. We are grateful to K. W. Wagner–Würzburg for helpful hints and discussions.

References

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

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

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

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

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

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

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

[15] 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.