On Balanced vs. Unbalanced Computation Trees*

Ulrich Hertrampf† *Heribert Vollmer*‡ *Klaus W. Wagner*†

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

email: {hertramp,vollmer,wagner}@informatik.uni-wuerzburg.de

Abstract

A great number of complexity classes between P and PSPACE can be defined via leaf languages for computation trees of nondeterministic polynomial time machines. Jenner, McKenzie, and Thérien (Proceedings of the 9th Conference on Structure in Complexity Theory, 1994) raised the issue of whether considering balanced or unbalanced trees makes any difference. For a number of leaf language classes, coincidence of both models was shown, but for the very prominent example of leaf language classes from the alternating logarithmic time hierarchy the question was left open. It was only proved that in the balanced case these classes exactly characterize the classes from the polynomial time hierarchy. Here, we show that balanced trees apparently make a difference: In the unbalanced case, a class from the logarithmic time hierarchy characterizes the corresponding class from the polynomial time hierarchy with a PP-oracle.

Along the way, we get an interesting normal form for PP computations.

*To appear in *Mathematical Systems Theory.*

†Supported by Deutsche Forschungsgemeinschaft, Grant no. Wa 847/1-1, "k-wertige Schaltkreise"

‡Supported in part by an Alexander von Humboldt fellowship

1. Introduction

Almost all natural complexity classes in the range between P and PSPACE can be defined by putting certain requirements on the computation trees produced by nondeterministic machines. Take for example the class NP, where we require that for a certain machine M defining an NP-language, a word is in the language under consideration if and only if the computation tree possesses at least one accepting path. For $\oplus$P, the number of accepting paths is required to be even. For PP, the number of accepting paths has to be greater than the number of rejecting paths.

It can already be seen from these examples that the condition put on the computation tree can often be formulated as a condition only on the leafstring of the tree, neglecting the structure of the tree itself, where the leafstring of a computation tree is the concatenation of the symbols printed at the leaves of the tree. Thus, if we have a set A of leafstrings (the so called leaf language), this set defines in a natural way a complexity class $\mathcal{C}$, namely $B \in \mathcal{C}$ if there exists a machine M such that an input x belongs to B if and only if the leafstring produced by M on input x belongs to A. (In symbols: $\mathcal{C} = \mathrm{Leaf}^{\mathrm{P}}(A)$.) In the same way, a class of leaf languages defines a complexity class. Concerning the classes from above, we see that NP is defined via the leaf language of all binary words which have at least one occurrence of the letter 1, $\oplus$P via the language of all binary words where the number of occurrences of the symbol 1 is divisible by 2, and PP via the language of all binary words which have more 1's than 0's.

It has been observed that in many cases it is of no importance whether we restrict ourselves to balanced computation trees or whether we allow arbitrary (unbalanced) trees. A balanced computation tree is a binary tree, where all paths have the same length and moreover, if we identify every path with the string over $\{0,1\}$ describing the sequence of nondeterministic choices on this path, then there is some string z such that all paths y with $|y| = |z|$ and $y \leq z$ (in lexicographic ordering) exist, but no such path with $y > z$ exists. Analogously to the above, we use the notation $\mathrm{BalancedLeaf}^{\mathrm{P}}(A)$ to denote the class that is defined via leaf language A and balanced computation trees. It is well known (see e.g. [15]) that in the cases of the classes NP, $\oplus$P, and PP, using the leaf languages given above, the same classes can be defined using arbitrary computation trees as well as using balanced computation trees.

Jenner, McKenzie, and Thérien already considered the issue of balanced vs. unbalanced trees in [14] and observed that both cases coincide if the leaf language is closed under a certain form of padding (where a special symbol is allowed to be inserted at arbitrary positions in a word without changing its language membership). There is also another (sufficient, but probably not necessary) condition for the coincidence of the balanced and unbalanced classes: If L is a language that can be represented as a word problem of a semigroup, then $\mathrm{Leaf}^{\mathrm{P}}(L) = \mathrm{BalancedLeaf}^{\mathrm{P}}(L)$ [11]. (Note that such languages L are regular.) Moreover, in [11] it is proven that for every regular language L there exists a regular language L' of the kind described above, such that $\mathrm{Leaf}^{\mathrm{P}}(L) = \mathrm{Leaf}^{\mathrm{P}}(L') = \mathrm{BalancedLeaf}^{\mathrm{P}}(L')$.

Clearly, if we want to define classes using the above approach we generally think of the leaf language to be of low complexity. We want the computation power of the class to be reflected by the computation of the nondeterministic machine. The evaluation of the tree then should be easy to carry out. But unfortunately, if we consider such leaf languages of very low complexity, it is often not clear whether they are closed under padding, and whether the balanced case differs from the unbalanced one. For instance, the context-sensitive languages, the regular languages, the languages with solvable monoids, the aperiodic languages, are all closed under padding, and thus they characterize EXPSPACE, PSPACE, MOD-PH, PH, both in the balanced and unbalanced case [12, 14]. But the classes of the alternating logarithmic time hierarchy are only known to characterize in the balanced case the classes of the polynomial time hierarchy.

The unbalanced case was left as an open question in [14]. We answer this question and show that unbalanced computation trees with leaf languages from some class of the logarithmic time hierarchy characterize exactly the corresponding class of the polynomial time hierarchy equipped with a PP-oracle.

We want to give some intuition why this is the case:

If we consider the balanced computation tree of a machine M on some input x, and we number the paths of that tree, then the set L_M defined as

$$\{ (x, i, a) \mid \text{the } i\text{-th path of } M \text{ on input } x \text{ yields result } a \}$$

can be decided deterministically in polynomial time. This holds since, given the number of a path, we can easily compute from that number the nondeterministic guesses of M on the corresponding path and then simulate deterministically. If the tree is unbalanced, then it is much harder to compute the nondeterministic guesses on the i-th path of M. However, it can be done as follows: Given some number i, to compute the first nondeterministic guess of M, we simply ask a function from $\#\mathrm{P}$ for the number of paths in the left subtree of the root. If this number is less than i, then the first nondeterministic guess of M is "right", else it is "left". This procedure can be continued until we arrive at a leaf. This proves that computing the guesses of machine M can be done in $\mathrm{P}^{\#\mathrm{P}}$.

Defining the class Path to consist of all languages L_M as above (varying over all nondeterministic polynomial time machines M), we can show that as an oracle, Path is as powerful as $\#\mathrm{P}$, i.e.

$$\mathrm{P}^{\mathrm{Path}} = \mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}}.$$

As a simple corollary, we get the following interesting normal form: We can normalize every PP-computation in such a way that all accepting paths appear to the left of all rejecting paths.

If we now have an unbalanced computation tree with a leaf language from the Σ_k-level of the logarithmic time hierarchy, then we can prove a correspondence between this model and a $\Sigma_k^{p\,\mathrm{Path}}$ computation: The computation of the machine accepting the leaf language directly reflects the Σ_k^p computation; and the unbalanced computation tree will turn out to be as good as a Path oracle. A logarithmic time machine asking for a certain leaf symbol is nothing other than a Σ_k^p machine asking a Path or PP oracle.

It turns out that the result sketched above can be proved in a far more general context: It does not only hold for Σ_k-computations, but for any (so-called) locally definable acceptance scheme [10] that we might put on nondeterministic logarithmic time or polynomial time machines. That is, whenever we have nondeterministic machines where in the nodes of the computation tree a certain function f from k-valued logic is applied, an analogue to the above holds, i.e.

1. $(f)\mathrm{P} = \mathrm{BalancedLeaf}^{\mathrm{P}}((f)\mathrm{LOGTIME})$.
2. $(f)\mathrm{P}^{\mathrm{PP}} = \mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME})$.

The result of Jenner, McKenzie, and Thérien on balanced trees with logtime leaf languages and our result on unbalanced trees are then special cases of those two theorems, since for every k, there is such a function f characterizing Σ_k-computations.

After presenting the formal details of our computation models, we will briefly show that the logtime result of Jenner, McKenzie, and Thérien holds also in the more general setting, and then make the above argumentation precise in Section 4.

2. Preliminaries

We assume the reader to be familiar with standard complexity theory notions. Our notation here is standard; see e.g. [1, 2, 13]. We presuppose familiarity with the classes of the alternating polynomial time hierarchy [17, 5], the alternating logarithmic time hierarchy [5, 16], and the probabilistic classes BPP and PP [6].

We use the following general computation models:

An *m-valued locally defined acceptance type* is a set F of functions from $\{0, \ldots, m-1\}^r$ into $\{0, \ldots, m-1\}$ for some $r \in \{2, 3, 4, \ldots\}$. Every locally definable acceptance scheme F corresponds to a complexity class, denoted by (F)P, in the following way: A language L belongs to (F)P if there exists a polynomial time nondeterministic machine M such that for every input x:

- (forming an *evaluation tree*) M attaches during its computation on input x to every non-leaf node of the computation tree with r successors an r-ary function from $F \cup \{\mathrm{id}\}$ $(r \geq 1)$ and to every leaf a value from $\{0, 1, \ldots, m-1\}$.
- (evaluating the tree) The values of the leaves are given. The value of a non-leaf node with attached r-ary function f is $f(a_1, a_2, \ldots, a_r)$, when $a_1, a_2, \ldots, a_r$ are the values of the successors.
- (acceptance) $x \in L$ if and only if the value of the root is 1.

A nondeterministic LOGTIME machine can be defined like a nondeterministic polynomial time machine with the difference that the LOGTIME machine may only use logarithmic time. However, to obtain a reasonable model one has to allow random access to the input bits. Thus we assume an additional index tape, where the machine may write an address. Then, if the machine enters some specific read state while the address on the index tape is i, it is given the input bit x_i. The index tape is not erased after a read operation. We define the class (F)LOGTIME analogously to the class (F)P, but with a logarithmic time machine instead of a polynomial time machine.

If F consists of one single function f, we simply write (f)P and (f)LOGTIME. It was shown in [8, 9] that considering acceptance types of only one binary function is no restriction; i. e. for every scheme F there exists one binary function f such that $(F)\mathrm{P} = (f)\mathrm{P}$. The proof of this fact carries over to the logarithmic time case, thus $(F)\mathrm{LOGTIME} = (f)\mathrm{LOGTIME}$ for a suitable binary function f. Moreover, binary functions were presented characterizing all classes from the polynomial time hierarchy and the class PSPACE (and similarly the classes from the alternating logtime hierarchy and the class NC^1).

In the leaf language approach to the characterization of complexity classes, the acceptance of a word input to a nondeterministic machine depends only on the values printed at the leaves of the computation tree. To be more precise, let M be a nondeterministic Turing machine, halting on every path, with some order on the nondeterministic choices. Then, $\mathrm{leafstring}^M(x)$ is the concatenation of the symbols printed at the leaves of the computation tree of M on input x. Call a computation tree of a machine M *balanced*, if all of its computation paths have the same length, and moreover, if we identify every path with the string over $\{0, 1\}$ describing the sequence of nondeterministic choices on this path, then there is some string z such that all paths y with $|y| = |z|$ and $y \leq z$ (in lexicographic ordering) exist, but no such path with $y > z$ exists. (Strictly speaking, the definition of *balanced* given in [14] is different from ours and, at first sight, slightly more general. However, it is easy to see that both definitions are equivalent.) Now given a language A, this language defines the class $\mathrm{BalancedLeaf}^{\mathrm{P}}(A)$ of all languages B for which there exists a nondeterministic polynomial time machine M whose computation tree is always balanced, such that $x \in B \iff \text{leafstring}^M(x) \in A$. Let $\mathcal{C}$ be a class of languages. The class $\text{BalancedLeaf}^{\text{P}}(\mathcal{C})$ consists of the union over all $A \in \mathcal{C}$ of the classes $\text{BalancedLeaf}^{\text{P}}(A)$.

This computation model was introduced by Bovet, Crescenzi, and Silvestri [3, 4] and later examined by Hertrampf, Lautemann, Schwentick, Vollmer, and Wagner [12], and Jenner, McKenzie, and Thérien [14]. Jenner, McKenzie, and Thérien also considered the case where the computation trees are not required to be balanced. For that case, let A be any language. Then, the class $\text{Leaf}^{\text{P}}(A)$ consists of those languages B for which there exists a nondeterministic polynomial time machine M without further restriction, such that $x \in B \iff \text{leafstring}^M(x) \in A$. Let $\mathcal{C}$ be a class of languages. The class $\text{Leaf}^{\text{P}}(\mathcal{C})$ consists of the union over all $A \in \mathcal{C}$ of the classes $\text{Leaf}^{\text{P}}(A)$.

The following relationship between the computation models just defined is known to hold (see [12]): For a class $\mathcal{C}$ there exists a binary associative function f such that $\mathcal{C} = (\{f\})\text{P}$ if and only if there exists a regular language L such that $\mathcal{C} = \text{Leaf}^{\text{P}}(L)$.

Relativizations

Strictly speaking, the approach of Bovet, Crescenzi, and Silvestri from [4] was more general than the one described above. They allowed not only to specify a leaf language A for the words to be accepted but also a leaf language R for words to be rejected. Thus, for $A \cap R = \emptyset$, $B \in \text{BalancedLeaf}^{\text{P}}(A, R)$ if and only if there exists a nondeterministic machine M with balanced computation trees, such that $x \in B \Rightarrow \text{leafstring}^M(x) \in A$ and $x \notin B \Rightarrow \text{leafstring}^M(x) \in R$. Then, they proved that

$$\text{BalancedLeaf}^{\text{P}}(A, R) \subseteq \text{BalancedLeaf}^{\text{P}}(A', R')$$

holds under all relativizations, if and only if the pair (A, B) is reducible to (A', B') by a polylogarithmic time bit-reduction. (This means that every bit of the result of the reduction function can be computed by a polylogarithmic time deterministic machine (with random access input tape), given the argument of the reduction function. For an exact definition, see [3, 12].) It is relatively easy to see that their result also holds in the more restricted model defined above, i. e.

$$\text{BalancedLeaf}^{\text{P}}(A) \subseteq \text{BalancedLeaf}^{\text{P}}(A')$$

under all relativizations if and only if A reduces to A' as just sketched.

What we want to point out here is that this result does not hold in the unbalanced case. Define BPP_{path} [7] to consist of all languages A such that there exists a nondeterministic polynomial time Turing machine M and an $\varepsilon > 0$ such that for all inputs x, we have: if $x \in A$, then the number of accepting paths of M is greater than $1/2 + \varepsilon$ times the total number of paths of M, and if $x \notin A$, then the number of rejecting paths of M is greater than $1/2 + \varepsilon$ times the total number of paths of M. Note that the class BPP is defined very similar not counting paths but considering acceptance probabilities. Define A' to consist of all words $x \in \{0,1\}^*$ such that at least two thirds of all symbols in x are 1's, and R' to consist of all words x over the same alphabet such that at least two thirds of all symbols in x are 0's. Then obviously, $\text{BalancedLeaf}^{\text{P}}(A', R') = \text{BPP}$, and $\text{Leaf}^{\text{P}}(A', R') = \text{BPP}_{\text{path}}$. Defining $A =_{\text{def}} \{0,1\}^*1\{0,1\}^*$ and $R = \{0\}^*$, obviously $\text{BalancedLeaf}^{\text{P}}(A, R) = \text{Leaf}^{\text{P}}(A, R) = \text{NP}$. Since there exists an oracle separating NP and BPP, we know that (A, R) does not reduce to (A', R'), but on the other hand $\text{NP} \subseteq \text{BPP}_{\text{path}}$ for all oracles (see [7]). So we see that the main result from [4] does not hold in the unbalanced case.

3. Logtime Leaf Languages for Balanced Trees

Jenner, McKenzie, and Thérien showed in [14] that balanced computation trees with leaf languages from one class of the logarithmic time hierarchy characterize the corresponding class from the polynomial time hierarchy. Here, we want to generalize their result to arbitrary locally definable (not necessarily Σ_k-) computations. We will make convenient use of the framework of locally definable acceptance types. Recall from the previous section, that considering acceptance types consisting of only one binary function is no restriction. Our proof is essentially the one from [14], therefore, we give only a very brief sketch.

3.1 Theorem. *Let f be a binary function. Then*

$$\mathrm{BalancedLeaf}^{\mathrm{P}}((f)\mathrm{LOGTIME}) = (f)\mathrm{P}.$$

Proof. Let $B \in (f)\mathrm{P}$ via machine M, running in time p. It is easy now to define a polynomial time nondeterministic machine, that on input $x = x_1 \cdots x_n$ produces a balanced computation tree with the leaf string

$$\#x_1\#x_2\#\#\#x_3\#^7x_4 \cdots \#^{2^{n-1}-1}x_n\#^{2^{p(n)}-2^n},$$

i.e. a leaf string of length $2^{p(n)}$, where bit x_i occurs at position 2^i. Now, define an f-machine M' with random access input tape, operating like M, but instead of reading bit number i from the input tape, M' writes 2^i on its index tape.

For the other direction, let $B \in \mathrm{BalancedLeaf}^{\mathrm{P}}((f)\mathrm{LOGTIME})$ via a polynomial time machine M and $A \in (f)\mathrm{LOGTIME}$. We design a polynomial time machine M' working on input x like the logarithmic time machine for A, but instead of querying the index tape for the i-th input bit, M' simulates the i-th path of M on input x, thus computing the i-th bit of $\mathrm{leafstring}^M(x)$. □

3.2 Corollary [14].

a) $\mathrm{BalancedLeaf}^{\mathrm{P}}(\Sigma_k\text{-LOGTIME}) = \Sigma_k^p$

b) $\mathrm{BalancedLeaf}^{\mathrm{P}}(\Pi_k\text{-LOGTIME}) = \Pi_k^p$

c) $\mathrm{BalancedLeaf}^{\mathrm{P}}(\mathrm{AC}^0) = \mathrm{PH}$

Proof. As pointed out in the preliminaries, there exist acceptance types characterizing Σ_k and Π_k computations, both in the polynomial time and in the logarithmic time case. This directly yields the first two statements. For the last statement we remark that no single function characterizing AC^0 and PH is known, however, it is evident that the proof of Theorem 3.1 goes through for the case of generalized acceptance types consisting of infinitely many functions. □

4. Logtime Leaf Languages for Unbalanced Trees

We now proceed to our main result. For the rest of this section, let f denote a binary k-valued function for some $k \geq 2$.

4.1 Lemma. $\mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME}) \subseteq (f)\mathrm{P}^{\mathrm{PP}}$

Proof. We show that $\mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME}) \subseteq (f)\mathrm{P}^{\#\mathrm{P}}$. It is clear that $(f)\mathrm{P}^{\#\mathrm{P}} \subseteq (f)\mathrm{P}^{\mathrm{PP}}$, since a $\#\mathrm{P}$ function can be computed deterministically in polynomial time by binary search with queries to a PP oracle. The proof is identical to the argument outlined in Theorem 3.1 that

$$\mathrm{BalancedLeaf}^{\mathrm{P}}((f)\mathrm{LOGTIME}) \subseteq (f)\mathrm{P},$$

except for the way in which M' computes the i-th symbol of $\mathrm{leafstring}^{M}(x)$. Here, M' accomplishes this by making calls to the oracle

$$A =_{\mathrm{def}} \{ (x, m, a) \mid \text{the } m\text{-th path of } M \text{ on input } x \text{ produces symbol } a \}.$$

Obviously, this oracle is in $\mathrm{P}^{\#\mathrm{P}}$, because we can trace the m-th path of M by asking an oracle how many paths of the tree of M on x are in the left subtree and how many paths are in the right subtree, and then moving on to the appropriate subtree. Clearly these questions are in $\#\mathrm{P}$.

$\square$

Before we state and prove our next lemma, we have to define the class Path, which is similar to the oracle A of the previous proof – it contains every language L that can possibly occur as A (depending on the machine M).

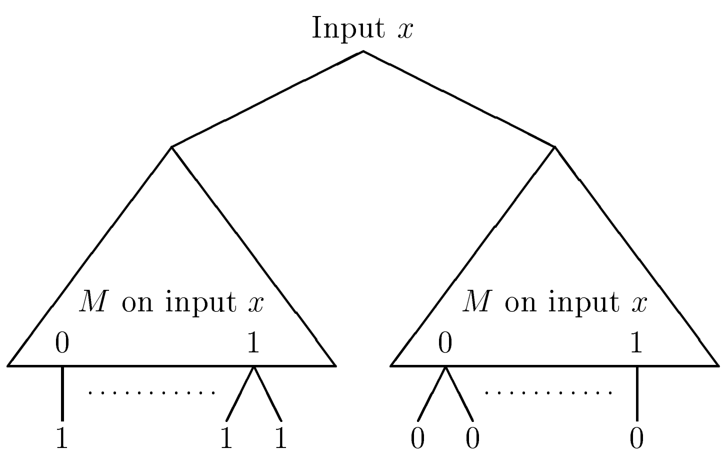
$$\begin{aligned}\mathrm{Path} =_{\mathrm{def}} \{ L \mid \exists M\colon M &\text{ is a polynomial time NTM and} \\ &(x, m, a) \in L \iff \text{the } m\text{-th path of } M \text{ on } x \text{ outputs } a \}\end{aligned}$$

When Path is used as an oracle, we distinguish two access mechanisms: the unrestricted oracle access allows arbitrary triples (x, m, a) to be queried. For the restricted oracle access, we allow only queries of the form (x, m, a), where x is the original input to the base machine. Using this type of oracle access, we define for a given acceptance type f:

$$\begin{aligned}(f)\mathrm{P}^{\mathrm{Path}}[\mathrm{res}] =_{\mathrm{def}} \{ L \mid L &\text{ is accepted by an } f\text{-machine} \\ &\text{with restricted access to an oracle from Path} \}\end{aligned}$$

4.2 Lemma. $(f)\mathrm{P}^{\mathrm{PP}} \subseteq (f)\mathrm{P}^{\mathrm{Path}}[\mathrm{res}]$

Proof. Let M be a PP-machine, which is normalized in the sense that the computation tree is a full binary tree of depth $p(|x|) > 0$ on input x, where p is a polynomial. Design M' as follows:



Thus, in the left half of the computation tree of M', we simulate the output behaviour of M with the following slight modification: if M outputs 1, then we have two leaves, both output 1, but if M outputs 0, then we have just one leaf of output 1. Similarly in the right half, if M outputs 0, then M' outputs 0 on two leaves, but if M outputs 1, then M' simply outputs 0.

Clearly, the computation tree of M' on input x has $c_x = 3 \cdot 2^{p(|x|)}$ leaves, and the leafstring is of the form $1^{a(x)}0^{c_x - a(x)}$, and M accepts if and only if $a(x) > \frac{1}{2} \cdot c_x$, so M accepts x if and only if the $(3 \cdot 2^{p(|x|)-1} + 1)$-th path of M' on input x accepts.

We should emphasize here that the total number of paths, as well as the significant path number, where the acceptance or rejection will be decided, depend only on $|x|$ and the fixed polynomial p.

We proved the fact that $(f)\mathrm{P}^{\mathrm{PP}} \subseteq (f)\mathrm{P}^{\mathrm{Path}}$. But we are not allowed to give different inputs to our Path-machine M'. We solve this problem by nondeterministically simulating M' on every input that could be asked to the oracle during the $(f)\mathrm{P}^{\mathrm{PP}}$ computation. So let the polynomial q be the time bound of the base machine (the (f)P-machine). Then the largest oracle queries to the PP-machine cannot exceed the length of $q(|x|)$. Now we can design a Path-machine as follows:

> `On input` x`:`
>
> `Guess` $i \in \{0, 1, \ldots, q(|x|)\}$`.`
>
> (In the computation tree, the corresponding paths should be ordered according to the natural order of the numbers i.)
>
> `Guess a string` y `of length` i`.`
>
> (As above, the corresponding paths should be ordered corresponding to the lexicographic order of the guessed strings y.)
>
> `Simulate` M' `on input` y`.`

An (f)P-machine with the language of this Path-machine as an oracle can find the answer to every oracle query of the $(f)\mathrm{P}^{\mathrm{PP}}$-machine by computing how many paths are produced by the computations of M' on shorter strings and how many paths are produced by lexicographically earlier strings of the same length, and then just asking for the value of the one path which is significant for the acceptance behaviour of M' on the oracle query in question. That is, to compute the oracle answer for query string z (where z is the k-th string of length $|z|$ in lexicographical order), ask for the path with number

$$s_z =_{\mathrm{def}} \sum_{i=0}^{|z|-1} 3 \cdot 2^{p(i)+i} + (k-1) \cdot 3 \cdot 2^{p(|z|)} + 3 \cdot 2^{p(|z|)-1} + 1.$$

□

4.3 Remark. The Path-machine M' given in the above proof in fact is also a PP-machine. Thus, we can normalize every PP-computation in such a way that all accepting paths appear to the left of all rejecting paths.

If we have two classes of languages $\mathcal{K}_1$ and $\mathcal{K}_2$, and for a reducibility notion $\leq$ we want to express the fact that for every $A \in \mathcal{K}_1$ there is a $B \in \mathcal{K}_2$ to which A reduces, then we write $\mathcal{K}_1 \leq \mathcal{K}_2$.

4.4 Corollary.

a) PP $\leq_m^{\log}$ Path.

b) Path $\subseteq \mathrm{P}^{\mathrm{PP}}$.

c) $\mathrm{P}^{\mathrm{Path}} = \mathrm{P}^{\mathrm{PP}}$.

Proof. Statement a) follows from the first part of the proof of Lemma 4.2 (construction of machine M'). The proof of Statement b) has been outlined in the second part of the proof of Lemma 4.1. Statement c) follows immediately from a) and b). $\square$

For a counterpart of Lemma 4.1 we now want to show that an (f)P computation with a PP oracle (or, equivalently, a Path oracle) can be simulated by a $\mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME})$ computation. However, in the unrestricted oracle access model, we cannot prevent the base machine to ask for paths of different computations of some machine, that is, for computations with different inputs; and this seems difficult to simulate. That is the reason why we have to consider the restricted oracle access model, and as we see from Lemma 4.2, this is sufficient to obtain a converse to Lemma 4.1.

4.5 Lemma. $(f)\mathrm{P}^{\mathrm{Path}}[\mathrm{res}] \subseteq \mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME})$

Proof. Let $L \in (f)\mathrm{P}^{\mathrm{Path}}[\mathrm{res}]$ via M_1, a polynomial time (f)P-machine with restricted access to the oracle $A \in$ Path. Let M_2 be a machine witnessing $A \in$ Path. Let p_1 be the polynomial bounding the runtime of M_1, and p_2 the polynomial bounding the runtime of M_2. Obviously, there exists a polynomial time machine M, such that if the machine M_2 on input x produces the leafstring w_x, then M produces the leafstring $x\#w_x\#^{2^{p_1(|x|)}}$.

Now consider the following nondeterministic machine M':

> `On input` w`:`
>
> `Check the initial bits of` w `until a # appears.`
>
> `Store these bits on an extra tape, call them` x`.`
>
> (So $w = x\#w'$.)
>
> `Simulate` M_1 `on input` x`.`
>
> `If` M_1 `asks the oracle for` (x, m, a)`, then`
>
> `get input symbol` $m + |x| + 1$ (which is the m-th symbol of w').
>
> `If` M_1 `uses the input, then use the extra tape.`

If M' works on $x\#w_x\#^{2^{p_1(|x|)}}$ then obviously M' simulates exactly the computation of M_1 on input x. So we only have to check whether M' works in logarithmic time. But the computation time is bounded by $p_1(|x|)$, and the input to M' has length $|x| + 1 + |w_x| + 2^{p_1(|x|)}$, which is greater than $2^{p_1(|x|)}$. This proves the lemma. $\square$

Now we obtain our main result immediately from Lemmas 4.1, 4.2, and 4.5:

4.6 Theorem. *Let f be a binary function. Then*

$$\mathrm{Leaf}^{\mathrm{P}}((f)\mathrm{LOGTIME}) = (f)\mathrm{P}^{\mathrm{PP}}.$$

4.7 Corollary.

a) $\mathrm{Leaf}^{\mathrm{P}}(\mathrm{DLOGTIME}) = \mathrm{P}^{\mathrm{PP}}$

b) $\mathrm{Leaf}^{\mathrm{P}}(\mathrm{NLOGTIME}) = \mathrm{NP}^{\mathrm{PP}}$

c) $\mathrm{Leaf}^{\mathrm{P}}(\Sigma_k\text{-LOGTIME}) = \Sigma_k^{p\,\mathrm{PP}}$

d) $\mathrm{Leaf}^{\mathrm{P}}(\Pi_k\text{-LOGTIME}) = \Pi_k^{p\,\mathrm{PP}}$

e) $\mathrm{Leaf}^{\mathrm{P}}(\mathrm{AC}^0) = \mathrm{PH}^{\mathrm{PP}}$

Proof. Completely analogous to the proof of Corollary 3.2. □

Acknowledgement. The ideas of this paper developed during a seminar on Complexity Theory on Burg Rothenfels in March 1994. We would like to thank the other participants of the seminar, i.e. Herbert Baier, Gerhard Buntrock, and Diana Rooß, for stimulating and helpful discussions. We are also grateful to an anonymous referee for a lot of valuable hints and suggestions.

References

[1] J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity I*; Springer Verlag (Berlin – Heidelberg – New York, 1988).

[2] J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity II*; Springer Verlag (Berlin – Heidelberg – New York, 1990).

[3] D. P. Bovet, P. Crescenzi, R. Silvestri, Complexity classes and sparse oracles; *Proceedings of the 6th Structure in Complexity Theory Conference* (1991), pp. 102–108.

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

[5] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, Alternation; *Journal of the ACM* **28** (1981), pp. 114–133.

[6] J. Gill, Computational complexity of probabilistic complexity classes; *SIAM Journal on Computing* **6** (1977), pp. 675–695.

[7] Y. Han, L. Hemachandra, T. Thierauf, Threshold computation and cryptographic security; *Proceedings of the 4th International Symposium on Algorithms and Computation* (1993), LNCS 762, pp. 230–239.

[8] U. Hertrampf, Locally definable acceptance types—the three valued case; *1st Latin American Symp. on Theoretical Informatics* (1992), LNCS 583, pp. 262–271.

[9] U. Hertrampf, Locally definable acceptance types for polynomial time machines; *Proceedings of the 9th Symp. on Theoretical Aspects of Computer Science* (1992), LNCS 577, pp. 199–207.

[10] U. Hertrampf, Complexity classes defined via k-valued functions; *Proceedings of the 9th Structure in Complexity Theory Conference* (1994), pp. 224–234.

[11] U. Hertrampf, Über Komplexitätsklassen, die mit Hilfe k-wertiger Funktionen definiert werden; Habilitationsschrift, Universität Würzburg, 1994.

[12] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, K.W. Wagner, On the power of polynomial time bit-reductions; *Proceedings of the 8th Structure in Complexity Theory Conference* (1993), pp. 200–207.

[13] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*; Addison-Wesley Publishing Company (Reading, Mass., 1979).

[14] B. Jenner, P. McKenzie, D. Thérien, Logspace and logtime leaf languages; *Proceedings of the 9th Structure in Complexity Theory Conference* (1994), pp. 242–254.

[15] J. Simon, On some central problems in computational complexity; Ph. D. Thesis, Cornell University, Ithaca, New York, 1975.

[16] M. Sipser, Borel sets and circuit complexity; *Proceedings of the 15th Symp. on the Theory of Computing* (1983), pp. 61–69.

[17] C. Wrathall, Complete sets and the polynomial-time hierarchy; *Theoretical Computer Science* **3** (1977), pp. 23–33.