

Self-Specifying Machines

Lane A. Hemaspaandra*
 Department of Computer Science
 University of Rochester
 Rochester, NY 14627, USA

Harald Hempel and Gerd Wechsung†
 Institut für Informatik
 Friedrich-Schiller-Universität Jena
 07740 Jena, Germany

October 22, 1998

Abstract

We study the computational power of machines that specify their own acceptance types, and show that they accept exactly the languages that $\leq_m^{\#P}$ -reduce to NP sets. A natural variant accepts exactly the languages that $\leq_m^{\#P}$ -reduce to P sets. We show that these two classes coincide if and only if $P^{\#P[1]} = P^{\#P[1]:NP^{O(1)}}$, where the latter class denotes the sets acceptable via at most one question to $\#P$ followed by at most a constant number of questions to NP.

1 Introduction

This paper studies the power of self-specifying acceptance types. As is standard, by acceptance type we mean the set of numbers of accepting paths that cause a machine to accept. Many complexity classes have a fixed acceptance type. For example, a set is in NP if and only if for some nondeterministic polynomial-time Turing machine (NPTM) M it holds that for each x , $x \in L$ if and only if $\#acc_M(x) \in \{1, 2, 3, \dots\}$, where $\#acc_M(x)$ represents the number of accepting paths of machine M on input x . Replacing $\{1, 2, 3, \dots\}$ with the set $\{1, 3, 5, \dots\}$ yields a perfectly acceptable definition of the complexity class $\oplus P$ [38, 12], and so on for many standard classes, such as coNP , US [4], etc. In fact, quite surprisingly, it turns out that there is a *single* polynomial-time computable set, $\text{MiddleBit} = \{i \mid \text{the } \lfloor (\log_2 i)/2 \rfloor \text{th bit of } i \text{ is a one}\}$, that is universal for PP^{PH} in the sense that this one set serves simultaneously as the fixed accepting type of all sets in PP^{PH} . This follows immediately from the fact that $\text{PP}^{\text{PH}} \subseteq \text{MP}$ (Green et al. [14], who also define the “middle bit” class MP). Thus, in standard notation (a full definition is included later), the fixed set MiddleBit has the property that $\text{PP}^{\text{PH}} \subseteq R_m^{\#P}(\{\text{MiddleBit}\})$.

*Supported in part by grants NSF-CCR-9322513 and NSF-INT-9513368/DAAD-315-PRO-fo-ab. Work done in part while visiting Friedrich-Schiller-Universität Jena. Email: lane@cs.rochester.edu.

†Supported in part by grant NSF-INT-9513368/DAAD-315-PRO-fo-ab. Work done in part while visiting Le Moyne College. Email: {hempel,wechsung}@informatik.uni-jena.de.

In contrast, some classes have been defined via an *external* function or set specifying their acceptance type. For example, the complexity class $C=P$ [40, 47] can be defined as the class of languages L such that for some NPTM M and some FP function g , for all x , $x \in L$ if and only if $\#acc_M(x) = g(x)$. In the even more abstract setting of so-called leaf languages ([8, 44], see also [26, 28]), separate predicates specify which numbers of accepting paths of a machine specify acceptance and which specify rejection. Such notions are sufficiently flexible to describe a broad range of classes, including even “promise” counting classes such as SPP [36, 11], etc.

In this paper, we introduce and study self-specifying acceptance types. A language is in SelfOutput if for some NPTM M and every input x , $x \in L$ if and only if $\#acc_M(x)$ is the output of some accepting path of $M(x)$. Similarly, a language is in SelfPath if for some NPTM M whose computation tree is always a complete tree (of polynomial depth specified by the input’s length) and every input x , $x \in L$ if and only if the lexicographically ($\#acc_M(x)$)th path of $M(x)$ is an accepting path. Note that self-specification allows the machine to dynamically set its own acceptance type, but restricts the machine by requiring that the machine’s paths not only specify its acceptance type but also execute it (in the sense that they themselves form the $\#acc_M$ set—i.e., the $\#P$ function in the sense defined below—being analyzed by the acceptance type).

Valiant [42, 43] introduced the class $\#P$, which is the class of functions f such that for some NPTM M and every x , $f(x) = \#acc_M(x)$. We prove that $\text{SelfOutput} = R_m^{\#P}(\text{NP})$ and that $\text{SelfPath} = R_m^{\#P}(\text{P})$. That is, SelfOutput and SelfPath consist of the sets that $\leq_m^{\#P}$ -reduce to NP and P sets, respectively. Put another way, SelfOutput and SelfPath capture the power of counting with respect to NP-computable and P-computable acceptance sets. Essentially equivalently, we establish that $\text{SelfOutput} = \text{NP} // \#P$ and $\text{SelfPath} = \text{P} // \#P$, where the $//$ is a certain recently introduced advice notation. For a definition of $//$ see Section 2. Also equivalently, we note that $\text{SelfOutput} = \text{P}^{\#P[1]:\text{NP}[1]_+}$, the class of languages accepted by P machines given at most one call to a $\#P$ oracle followed by at most one positive [33, 39] query to an NP oracle.

Note that it is not at all clear whether SelfPath equals SelfOutput (equivalently, in light of the characterizations of this paper, whether $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[1]_+}$). However, we show that these two classes are equal if and only if $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[\mathcal{O}(1)]}$. This is a so-called “downward separation” result (see, e.g., [21], for some background), and indeed what our proof actually establishes is that the following three conditions are equivalent:

1. $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[1]_+}$,
2. $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[1]}$, and
3. $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[\mathcal{O}(1)]}$.

Since, in contrast with the just-mentioned open issue of whether $\text{P}^{\#P[1]} = \text{P}^{\#P[1]:\text{NP}[1]_+}$, it is easy to see via standard techniques [10, 38] that $\text{P}^{\#P[1]}$ *does* equal $\text{P}^{\text{NP}[1]:\#P[1]}$ (indeed, even $\text{P}^{\#P[1]} = \text{P}^{\text{NP}[\mathcal{O}(\log n)]:\#P[1]}$), the comments of the previous paragraph give some weak evidence that order of access may be important in determining computational power, a

theme that has been raised and studied in other settings (see the survey [17]). Unfortunately, in the present setting, giving firm evidence for this seems hard. In fact, there is no known oracle separation of $\mathsf{P}^{\#\mathsf{P}^{[1]}}$ from PSPACE (much less of $\mathsf{P}^{\#\mathsf{P}^{[1]}}$ from $\mathsf{P}^{\#\mathsf{P}^{[1]}\cdot\mathsf{NP}^{[1]}}$), though much effort has been made in that direction.

2 Preliminaries

For standard notations and definitions that are not included here, we refer the reader to any complexity textbook, e.g., [7, 2, 37]. $\mathbf{N} = \{0, 1, 2, \dots\}$. For any $n \in \mathbf{N}$, define $string(n)$ to be n written in binary with no leading zeros. For any string $x \in \Sigma^*$, let $string(x) = x$. For any $x \in \Sigma^* - \epsilon$, define $int(x)$ to be x interpreted as the binary representation of a natural number. By convention, let $int(\epsilon) = 0$. (The function $int(\cdot)$ is not a one-to-one function, e.g., $int(0011) = int(11) = 3$. It merely is the natural direct reading of strings as representations of natural numbers.) For each NPTM M , for each $x \in \Sigma^*$, and for each accepting path y of $M(x)$, let $pathout_M(x, y)$ be the integer output on path y , which by definition we take to be $int(w)$, where w is the bits of the work tape between the left endmarker and the first tape cell that holds neither a 0 nor a 1, i.e., that is some other symbol or a blank (see [6, 5]). Define $acc_M(x) = \{y \in \Sigma^* \mid y \text{ is an accepting path of } M(x)\}$, $iacc_M(x) = \{int(z) \mid z \in acc_M(x)\}$, $iout_M(x) = \{n \in \mathbf{N} \mid (\exists y)[y \in acc_M(x) \text{ and } n = pathout_M(x, y)]\}$, $\#acc_M(x) = ||acc_M(x)||$, and $span_M(x) = ||iout_M(x)||$. Recall that $\#\mathsf{P} = \{f : \Sigma^* \rightarrow \mathbf{N} \mid (\exists \text{ NPTM } M)(\forall x)[f(x) = \#acc_M(x)]\}$ [42, 43]. $\#\mathsf{SAT}$ is the function such that $\#\mathsf{SAT}(f)$ is the number of satisfying assignments of f if f is a satisfiable boolean formula, and $\#\mathsf{SAT}(f)$ is 0 otherwise. $\#\mathsf{SAT}$ is known to be $\#\mathsf{P}$ -complete (see [42, 43, 50]).

A set A is many-one reducible to B via a $\#\mathsf{P}$ function, $A \leq_m^{\#\mathsf{P}} B$, if and only if there exists a function $f \in \#\mathsf{P}$ such that, for all x , $x \in A \iff string(f(x)) \in B$. For any a and b for which \leq_a^b is defined and any class \mathcal{C} , let $R_a^b(\mathcal{C}) = \{L \mid (\exists C \in \mathcal{C})[L \leq_a^b C]\}$.

Hemaspaandra, Hempel, and Wechsung [20] introduced the study of the power of ordered query access (other papers on or related to query order include [18, 1, 25, 3, 48, 34, 19] and the survey [17]). We adopt this notion, and extend it to the function class case and to the positive query case. For any function or language classes \mathcal{C}_1 and \mathcal{C}_2 , define $\mathsf{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1]}$ to be the class of languages accepted by polynomial-time machines making at most one query to a \mathcal{C}_1 oracle followed by at most one query to a \mathcal{C}_2 oracle. If \mathcal{C}_2 is a language class, then $\mathsf{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1]+}$ is the class of all sets in $\mathsf{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1]}$ witnessed by a polynomial-time oracle machine that accepts if and only if the \mathcal{C}_2 set is queried and the answer to that query is “yes.”¹

¹We mention that this requires that the querying machine is such that if (in light of of the \mathcal{C}_1 query’s actual answer if any—note that we do not require that a query to either \mathcal{C}_1 or \mathcal{C}_2 necessarily be made) there is a \mathcal{C}_2 query then the truth-table the querying machine has with respect to the answer from the \mathcal{C}_2 query, given the true answer to the \mathcal{C}_1 query if there is any such query, is that the machine will accept if and only if the answer is “yes.” However, for most natural classes, in particular $\mathcal{C}_2 = \mathsf{NP}$, without loss of generality we can assume that the querying machine always makes exactly two queries and that its truth table with respect to the two answers (even given a “lying” answer to the first query) is: accept if and only if the second answer is “yes.”

Two other formalisms can represent similar notions. Let $\langle \cdot, \cdot \rangle$ be a pairing function from $\Sigma^* \times \Sigma^*$ to Σ^* having the standard properties (easily computable, easily invertible, etc.). Recall that, as noted above for the case $\mathcal{F} = \#P$, for any class of functions \mathcal{F} and any sets A and B : $A \leq_m^{\mathcal{F}} B \iff (\exists f \in \mathcal{F})(\forall x)[x \in A \iff \text{string}(f(x)) \in B]$. Generalizing the seminal “advice classes” notion of Karp and Lipton [29], Köbler and Thierauf [31] have studied an interesting notion, which previously appeared in less general form in work of Krentel and others. Köbler and Thierauf note that the notion is related to many-one reductions via functions. The notion is as follows: For any function class \mathcal{F} and any complexity class \mathcal{C} define: $\mathcal{C} // \mathcal{F} = \{L \mid (\exists f \in \mathcal{F})(\exists C \in \mathcal{C})(\forall x)[x \in L \iff \langle x, \text{string}(f(x)) \rangle \in C]\}$ [31].

We note that it will not always be the case that $P^{\mathcal{F}[1]:\mathcal{C}[1]^+} = \mathcal{C} // \mathcal{F} = R_m^{\mathcal{F}}(\mathcal{C})$. The reason why this may not always hold is that it is possible that \mathcal{C} lacks the power to decode pairing functions or to form from a function value the appropriate \mathcal{C} query, or that (in terms of proving $\mathcal{C} // \mathcal{F} \subseteq R_m^{\mathcal{F}}(\mathcal{C})$) \mathcal{F} lacks the power to code the input into its output in a way that \mathcal{C} can decode. However, for flexible language classes such as P, NP, etc., and flexible function classes such as $\#P$, OptP [32], etc., this equality will hold, as has been noted by Hemaspaandra and Hoene [16] for the particular case of $R_m^{\text{OptP}}(\oplus P)$. In particular, in terms of relations of interest in the current paper, note that we clearly can claim

$$P^{\#P[1]:NP[1]^+} = NP // \#P = R_m^{\#P}(NP), \text{ and}$$

$$P^{\#P[1]} = P // \#P = R_m^{\#P}(P).$$

Note that expressions of this form are perhaps more natural than one might first guess. They capture the power of a class whose computation is aided by some advice provided by some complexity-bounded function class. Indeed, $R_m^{\text{OptP}}(P)$ turns out, as Krentel established [32], to be exactly P^{NP} . $R_m^{\text{OptP}}(\oplus P)$ turns out to be exactly the sets that \leq_m^p -reduce to languages having easy “implicit membership tests” [16], and also is exactly the class of languages accepted at the second level of Cai and Furst’s [9] safe storage hierarchy ([35], see also the discussion in [22]).

Generalizing the earlier definition to the case of more queries, for any function or language classes \mathcal{C}_1 and \mathcal{C}_2 and for any k , let $P^{\mathcal{C}_1[1]:\mathcal{C}_2[k]}$ denote the class of languages accepted by polynomial-time machines making at most one query to a \mathcal{C}_1 oracle followed by at most k queries to a \mathcal{C}_2 oracle. $P^{\mathcal{C}_1[1]:\mathcal{C}_2[\mathcal{O}(1)]}$ will denote $\bigcup_{k>0} P^{\mathcal{C}_1[1]:\mathcal{C}_2[k]}$.

We say that an NPTM is *normalized* if, for some polynomial p and for every input x , every computation path on input x has exactly $p(|x|)$ nondeterministic choices. (Note that it is not the case that every computation step must make a nondeterministic choice. However, under our definition, on each input x every possible binary string of exactly $p(|x|)$ bits corresponds uniquely to a computation path.) Though normalization is known not to be important in defining NP languages or even—as shown by Simon [40]—PP languages, there is evidence that the type of normalization one uses is critical in defining BPP languages [15]. Here, it will be important in our definition of SelfPath. However, we note that in our definition of SelfOutput, even if a normalization requirement were added the class defined would remain the same.

We now introduce the classes SelfOutput and SelfPath, which model self-specifying acceptance. A language L is in SelfOutput if for some NPTM M and all x , $x \in L$ if and only if $\#acc_M(x) \in iout_M(x)$. A language L is in SelfPath if for some normalized NPTM M and every x , $x \in L$ if and only if (a) $\#acc_M(x) > 0$ and (b) $\#acc_M(x) - 1 \in iacc_M(x)$, that is, the lexicographically $(\#acc_M(x))$ th path of $M(x)$ is an accepting path. In the definition of SelfPath, we view the leftmost path (i.e., the path on which all nondeterministic guesses are 0) of the machine as the lexicographically first path, and so on.²

3 Self-Specifying Acceptance Types

We characterize SelfPath and SelfOutput as the sets that $\leq_m^{\#P}$ -reduce to P and NP sets, respectively.

Theorem 3.1

1. SelfPath = $R_m^{\#P}(P)$.
2. SelfOutput = $R_m^{\#P}(NP)$.

Proof (1) Recall from Section 2 that $R_m^{\#P}(P) = P^{\#P[1]}$. Suppose $A \in \text{SelfPath}$, then by definition there is an NPTM M such that

$x \in A \iff$ the lexicographically $(\#acc_M(x))$ th path of $M(x)$ is an accepting path.

So with one call to a $\#P$ oracle we can compute the value of $\#acc_M(x)$ and check whether the $(\#acc_M(x))$ th path of $M(x)$ is an accepting path. Since $M(x)$ is a complete tree, we can do this latter check in deterministic polynomial time.

Now let $A \in P^{\#P[1]}$ via a deterministic oracle machine M and, without loss of generality, let the $\#P$ oracle be $\#SAT$. Without loss of generality we may assume that on each input x it holds that M asks exactly one question to $\#SAT$, getting the answer $f(x)$. Furthermore, let M' be an NPTM witnessing $f \in \#P$. Let M' be normalized, i.e., for some polynomial p , $M'(x)$ makes exactly $p(|x|)$ nondeterministic moves.

We will define an NPTM N , which will witness the fact that $A \in \text{SelfPath}$. $N(x)$ will be normalized to make exactly $p(|x|) + 3$ nondeterministic moves. The computation of $N(x)$ along path y , where $y \in \Sigma^{p(|x|)+3}$, goes as follows (see Figure 1). (Here we take strings y to represent a computation path, where y is simply the sequence of nondeterministic choices along the path.) Recall that $0 \leq f(x) \leq 2^{p(|x|)}$, so $f(x)$ can take on $2^{p(|x|)} + 1$ possible values. The cases will reflect this (see the conditions defining Cases 1.1 and 2.1, which lets $int(y')$ sweep through exactly those values).

Case 1 $y = 00y'$.

²It is not hard to see that one forms the same class of languages with the following alternate definition, which in some sense starts the path counting at 0: A language L is in SelfPath if for some normalized NPTM M and every x , (a) $M(x)$ has at least one rejecting path, and (b) $x \in L \iff$ the lexicographically $(1 + \#acc_M(x))$ th path of $M(x)$ is an accepting path.

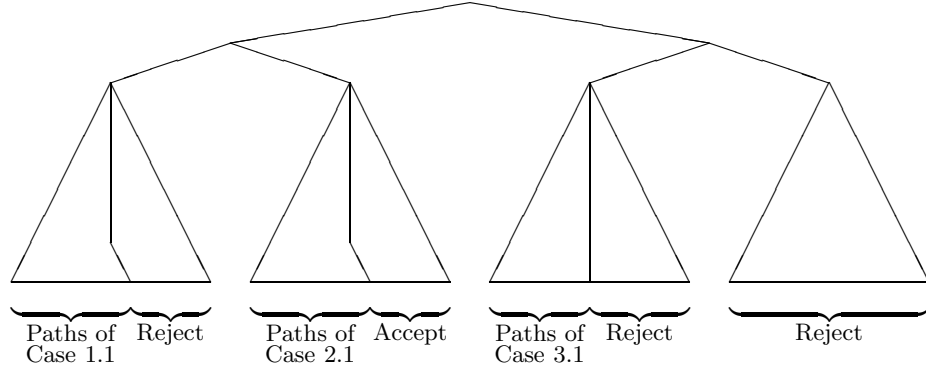


Figure 1: Computation tree of $N(x)$.

Case 1.1 $int(y') \leq 2^{p(|x|)}$.

Simulate $M(x)$, assuming the answer to the #SAT query is $int(y')$ and reject if and only if $M(x)$ accepts.

Case 1.2 $int(y') > 2^{p(|x|)}$.

Reject.

Case 2 $y = 01y'$.

Case 2.1 $int(y') \leq 2^{p(|x|)}$.

Simulate $M(x)$, assuming the answer to the #SAT query is $int(y')$ and accept if and only if $M(x)$ accepts.

Case 2.2 $int(y') > 2^{p(|x|)}$.

Accept.

Case 3 $y = 10y'$.

Case 3.1 $y' = 0z$, where $z \in \Sigma^{p(|x|)}$.

Simulate $M'(x)$ on path z , i.e., accept if and only if path z on $M'(x)$ accepts.

Case 3.2 $y' = 1z$, where $z \in \Sigma^{p(|x|)}$.

Reject.

Case 4 $y = 11y'$.

Reject.

Note that $||\{0y' \mid y' \in \Sigma^{p(|x|)+2} \text{ and } N(x) \text{ accepts along } 0y'\}\| = 2^{p(|x|)+1}$
and $||\{1y' \mid y' \in \Sigma^{p(|x|)+2} \text{ and } N(x) \text{ accepts along } 1y'\}\| = f(x)$.

Hence $N(x)$ has exactly $2^{p(|x|)+1} + f(x)$ accepting paths. However, note that the $(2^{p(|x|)+1} + f(x))$ th path of $N(x)$ is a path of the form of Case 2.1 and by construction it accepts if and only if $M(x)$, given the answer $f(x)$, accepts. So $A \in \text{SelfPath}$.

(2) Recall from Section 2 that $R_m^{\#P}(\text{NP}) = P^{\#P[1]:\text{NP}[1]+}$. Suppose $A \in \text{SelfOutput}$ then by definition there is an NPTM M such that

$x \in A \iff \#acc_M(x)$ is the integer reading of the output of some accepting path of $M(x)$.

So with one call to $\#SAT$ we can compute the value of $\#acc_M(x)$ and by querying “ $\langle x, \text{string}(\#acc_M(x)) \rangle \in B?$ ”, where $B =_{def} \{\langle x, \text{string}(m) \rangle \mid m \in \text{iout}_M(x)\}$, we can decide whether $x \in A$. Note that $B \in \text{NP}$ and that the second query is a positive one.

Now let $A \in P^{\#P[1]:\text{NP}[1]+}$ and thus $A \in P^{\#SAT[1]:\text{SAT}[1]+}$. Furthermore we may, without loss of generality (see Footnote 1), assume that some deterministic machine, M , that witnesses $A \in P^{\#SAT[1]:\text{SAT}[1]+}$ has the property that on every input it asks exactly one query to each oracle and the query to SAT is strictly positive in the sense that the machine accepts if and only if the query is answered “yes.” This strict positivity property is important to keep in mind throughout the proof.

Since $\#P$ is closed under polynomial-time input transformation let $f(x)$ be the answer to the query that $M(x)$ makes to $\#SAT$, and denote the question to SAT by $z(x, f(x))$. Note that, as SAT is a cylinder, we may without loss of generality assume that z is such that there exists a polynomial q for which $(\forall x \in \Sigma^*)(\forall n : 0 \leq n \leq 2^{p(|x|)})[|z(x, n)| = q(|x|)]$.

Let M' be an NPTM witnessing $f \in \#P$. Let M' be normalized to have branching depth exactly given by the polynomial p . Suppose M_{SAT} witnesses $\text{SAT} \in \text{NP}$ and M_{SAT} is normalized to have branching depth exactly given by the polynomial q' , so $M_{\text{SAT}}(z(x, f(x)))$ has exactly $2^{q'(q(|x|))}$ paths. Without loss of generality, let q and q' be such that

$$(\forall n)[q(n) \geq 1 \text{ and } q'(n) \geq 1].$$

We construct the NPTM T witnessing $A \in \text{SelfOutput}$. On input x , $T(x)$ does the following (see Figure 2):

- $T(x)$ guesses $i \in \{0, 1\}$.
 - If $i = 0$ was guessed, $T(x)$ runs $M'(x)$, but outputs the integer 0 on all accepting paths of $M'(x)$.
 - If $i = 1$ was guessed, $T(x)$ guesses an n , $0 \leq n \leq 2^{p(|x|)}$, and simulates what $M(x)$ would do were n the answer to the $\#SAT$ query. $M(x)$ under this assumption queries “ $z(x, n) \in \text{SAT}?$ ” so $T(x)$ guesses a path y of $M_{\text{SAT}}(z(x, n))$. Recall that M' 's behavior is such that it accepts if and only if the answer to its second query is “yes.” If $M_{\text{SAT}}(z(x, n))$ accepts on path y , then $T(x)$ also accepts on path y and outputs the integer $(2^{p(|x|)} + 1)2^{q'(q(|x|))} + n$. Otherwise $T(x)$ accepts on the present path and outputs the integer 0.

By construction $T(x)$ has exactly $(2^{p(|x|)} + 1)2^{q'(|x|)} + f(x)$ accepting paths. This value is an output of some accepting path, if and only if $x \in A$. So $A \in \text{SelfOutput}$. \blacksquare

Note that the proof technique in fact can be applied to characterize via self-specifying acceptance other classes, such as $R_m^{\#P}(\text{UP})$, $R_m^{\#P}(\oplus P)$, $R_m^{\#P}(\text{C=P})$, and $R_m^{\#P}(\text{PP})$. For example, if we change the definition of SelfOutput to add a requirement that no two accepting paths output the same value, this “unambiguous SelfOutput ” class equals $R_m^{\#P}(\text{UP})$. Similarly, if we change the definition of SelfOutput to accept exactly when the number of paths itself is the output on an odd number of paths, we obtain $R_m^{\#P}(\oplus P)$. Similar claims hold for $R_m^{\#P}(\text{C=P})$ and (with a bit of care) for $R_m^{\#P}(\text{PP})$.

Let us adopt Valiant’s [42] standard definition of $\#NP$ (informally, $\#NP = (\#P)^{NP}$), and the analogous definition of $\#PH$ (informally, $\#PH = (\#P)^{PH}$) [49].³ In this paper, we have discussed the sets $\leq_m^{\#P}$ -reducible to certain classes. One might naturally wonder whether $\leq_m^{\#NP}$ -reductions to the same classes yield even greater computational power. However, note that from Toda and Watanabe’s [49] result $\#PH \subseteq \text{FP}^{\#P[1]}$ we can easily prove the following proposition, which says that for most natural classes $\leq_m^{\#NP}$ -reductions to the class (or even $\leq_m^{\#PH}$ -reductions to the class) yield no greater computational power than $\leq_m^{\#P}$ -reductions to the class. (We say \mathcal{C} is closed downwards under \leq_m^p reductions if $R_m^p(\mathcal{C}) \subseteq \mathcal{C}$.)

Proposition 1 *For any complexity class \mathcal{C} closed downwards under \leq_m^p reductions, it holds that $R_m^{\#P}(\mathcal{C}) = R_m^{\#PH}(\mathcal{C})$.*

Proof: The inclusion $R_m^{\#P}(\mathcal{C}) \subseteq R_m^{\#PH}(\mathcal{C})$ is immediate. For the “ \supseteq ” inclusion, let $A \in R_m^{\#PH}(\mathcal{C})$ via $f \in \#PH$ and $C \in \mathcal{C}$. Thus, $x \in A \iff \text{string}(f(x)) \in C$. Note that due to the inclusion $\#PH \subseteq \text{FP}^{\#P[1]}$ [49], we also have $\text{string}(f(\cdot)) \in \text{FP}^{\#P[1]}$. Hence there exist a DPTM M and a function $g \in \#P$ such that, for each $x \in \Sigma^*$, $M^{g[1]}(x)$ computes $\text{string}(f(x))$. Without loss of generality we assume that $M(x)$ always asks exactly one question to its oracle.

Let $q(x)$ denote the query asked by $M(x)$. For any string $x \in \Sigma^*$, $(1x)_{\text{binary}}$ will denote the integer value of the string formed by prepending a 1 before x and interpreting the string as a binary integer. Let $\langle \cdot, \cdot \rangle_{\mathbf{N}}$ be an easily computable pairing function from $\mathbf{N} \times \mathbf{N}$ to \mathbf{N} such that (a) $h(x) = \langle (1x)_{\text{binary}}, g(q(x)) \rangle_{\mathbf{N}}$ is a $\#P$ function, and (b) there exist polynomial-time functions to extract from $\langle \cdot, \cdot \rangle_{\mathbf{N}}$ its first and second components. Such pairing functions are known to exist via standard techniques [38, 10]. For any $m \in \mathbf{N}$, let \widehat{m} denote the constant function that, on any input, returns the value m . Let

$$D = \{ \text{string}(\langle (1x)_{\text{binary}}, m \rangle_{\mathbf{N}}) \mid \text{the value output by } M^{\widehat{m}}(x) \text{ is in } C \},$$

³Vollmer [45] and Toda and Watanabe ([49], using the different notation “ $\text{NUM} \cdot \mathcal{C}$ ”) have proposed interesting and different “ $\#$ ”-type classes. Often, though not always, Vollmer’s classes are referred to using notation such as $\# \cdot \text{NP}$, in order to avoid ambiguity as to whether his classes or Valiant’s classes are being discussed (see [23]). Here, we uniformly use Valiant’s classes, though we mention in passing that for $\#PH$ the two notions are known to coincide [49, Proposition 3.1].

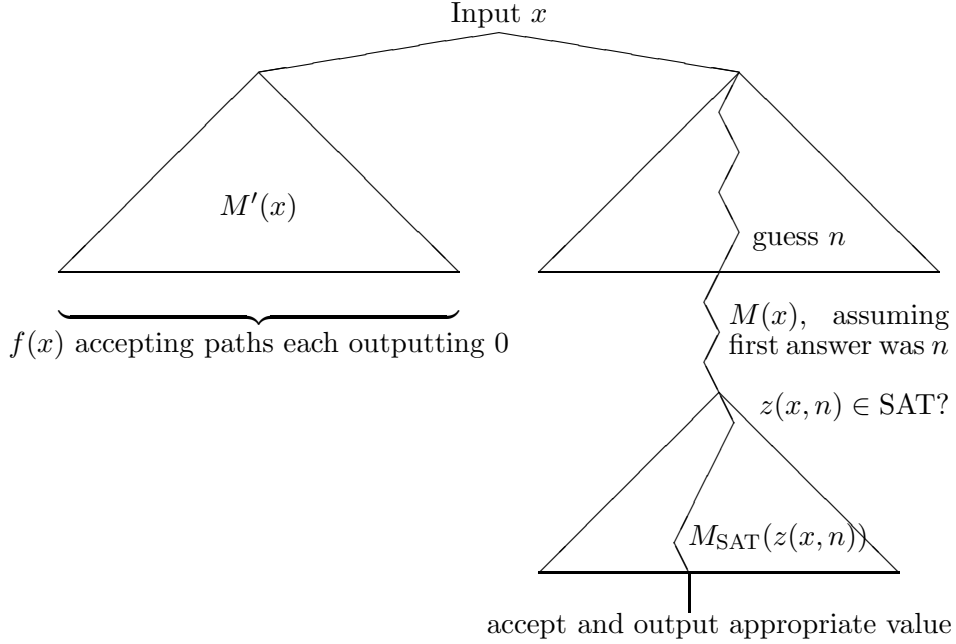


Figure 2: Computation tree of $T(x)$.

and note that $D \in \mathcal{C}$ since \mathcal{C} is closed downwards under \leq_m^p reductions. We have

$$\begin{aligned} x \in A &\iff \text{string}(f(x)) \in \mathcal{C} \\ &\iff \text{string}(h(x)) \in D. \end{aligned}$$

This shows $A \in \mathbf{R}_m^{\#\mathbf{P}}(\mathcal{C})$. ■

Proposition 1 implies that if in the definition of SelfOutput we replace “ $x \in L \iff \#acc_M(x)$ is the integer reading of the output of some accepting path” with “ $x \in L \iff span_M(x)$ is the integer reading of the output of some accepting path” (where $span_M(x)$ [30] as is standard denotes the number of distinct outputs of $M(x)$), then the class defined remains unchanged.

From Theorem 3.1 and the discussion of Section 2, it is clear that we have the following.

Corollary 3.2 $\text{SelfPath} = \text{SelfOutput} \iff \mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}^{\#\mathbf{P}[1]:\mathbf{NP}[1]_+}$.

Are SelfPath and SelfOutput in fact equal? In light of our Corollary 3.2, this is equivalently a question— $\mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}^{\#\mathbf{P}[1]:\mathbf{NP}[1]_+}$ —about somewhat more familiar-looking complexity classes, and in this form it allows us to see more clearly a relationship with an open issue from the literature. In particular, no oracle separation yet exists for these classes. In fact, separating even over a vastly larger gap that includes these classes is an open issue. That

is, $\text{PP}^{\oplus\text{P}} \subseteq \text{P}^{\#\text{P}[1]} \subseteq \text{P}^{\#\text{P}[1]:\text{NP}[1]_+} \subseteq \text{P}^{\#\text{P}[2]} \subseteq \text{PSPACE}$ (the first containment—which is nontrivial—is due to Toda [41]), yet no known oracle separates $\text{PP}^{\oplus\text{P}}$ from PSPACE (see Green [13] for background and for progress on a related line). On the other hand, as even $\text{P}^{\text{NP}[\mathcal{O}(\log n)]:\#\text{P}[1]}$ easily equals $\text{P}^{\#\text{P}[1]}$ (via the technique of [10, 38]), clearly if query order can be swapped in the characterization of SelfOutput then $\text{SelfPath} = \text{SelfOutput}$.

It would be nice to achieve some structural collapse from the assumption $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+}$. We in fact can prove such a collapse.

Theorem 3.3 $\text{SelfPath} = \text{SelfOutput} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}$.

Theorem 3.3 equivalently says

$$\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}.$$

Related work shows that the equality of SelfPath and SelfOutput would collapse the boolean hierarchy over SelfOutput [24].

Proof of Theorem 3.3: We will prove this in two steps. First we will prove that

$$\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]},$$

and then we will prove that

$$\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}.$$

From these two equivalences we are done, in light of Corollary 3.2.

The right to left implication of

$$\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]}$$

is immediate. Regarding the left to right implication, suppose that $B \in \text{P}^{\#\text{P}[1]:\text{NP}[1]}$ and assume that $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+}$. Note that

$$\text{P}^{\#\text{P}[1]:\text{NP}[1]} \subseteq \{L_1 \cup L_2 \mid L_1 \in \text{P}^{\#\text{P}[1]:\text{NP}[1]_+} \wedge \overline{L_2} \in \text{P}^{\#\text{P}[1]:\text{NP}[1]_+}\}.$$

From this, combined with our $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]_+}$ assumption and the fact that $\text{P}^{\#\text{P}[1]}$ is closed under complementation and union, we have $B \in \text{P}^{\#\text{P}[1]}$.

That $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]} \iff \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}$ holds can be seen as follows. The right to left implication is immediate. Regarding the left to right implication we have, as our assumption, that $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[1]}$. We will prove that, for each $k \geq 1$: $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[k]} \Rightarrow \text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[k+1]}$. This suffices, via induction.

So let k be some positive integer and assume that $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[k]}$. Let $A \in \text{P}^{\#\text{P}[1]:\text{NP}[k+1]}$. Let M be a deterministic polynomial-time machine that accepts A via at most one query to $\#\text{P}$ followed by at most $k+1$ queries to NP . Without loss of generality, assume that M always asks exactly one query to $\#\text{SAT}$ followed by exactly $k+1$ queries to SAT . Consider the machine M_{no} that (on each input) exactly simulates M , except rather than asking the first of M 's $k+1$ queries to SAT it assumes the answer to that query is

“no.” Consider also the machine M_{yes} that (on each input) exactly simulates M , except rather than asking the first of M 's $k+1$ queries to SAT it assumes the answer to that query is “yes.” Let

$$L_{no} = \{x \mid x \in L(M_{no}^{\#\text{SAT}[1]:\text{SAT}[k]})\}$$

and

$$L_{yes} = \{x \mid x \in L(M_{yes}^{\#\text{SAT}[1]:\text{SAT}[k]})\}.$$

Note that $L_{no} \in \text{P}^{\#\text{P}[1]:\text{NP}[k]}$ and $L_{yes} \in \text{P}^{\#\text{P}[1]:\text{NP}[k]}$. By our assumption, it follows that both L_{no} and L_{yes} are in $\text{P}^{\#\text{P}[1]}$. Let \widehat{M}_{no} (respectively, \widehat{M}_{yes}) be a machine that makes one query to $\#\text{SAT}$, and whose language is L_{no} (respectively, L_{yes}). We claim that $A \in \text{P}^{\#\text{P}[1]:\text{NP}[1]}$, via the following machine M_v . On input x , $M_v(x)$ asks its $\#\text{P}$ oracle for the answer to (a) the $\#\text{SAT}$ query asked by $M(x)$, (b) the $\#\text{SAT}$ query asked by $\widehat{M}_{no}(x)$, and (c) the $\#\text{SAT}$ query asked by $\widehat{M}_{yes}(x)$. This can all be done via *one* query to $\#\text{SAT}$, as it is well known that three parallel queries to $\#\text{SAT}$ can be encoded as one query to $\#\text{SAT}$ ([38], see also [10]). Using the answer to item (a), $M_v(x)$ then determines the first query to SAT that would be made by $M(x)$, namely, the SAT query $M(x)$ makes after $M(x)$ gets the reply from its $\#\text{SAT}$ query. $M_v(x)$ asks this question to its own SAT oracle. If the answer is “no,” then $M_v(x)$ accepts if and only if $x \in L_{no}$, and we can easily evaluate this with no additional queries, via simulating $\widehat{M}_{no}(x)$ using the $\#\text{SAT}$ answer obtained in item (b). If the answer is yes, then $M_v(x)$ accepts if and only if $x \in L_{yes}$, and we can easily evaluate this with no additional queries, via simulating $\widehat{M}_{yes}(x)$ using the $\#\text{SAT}$ answer obtained in item (c). So $A \in \text{P}^{\#\text{P}[1]:\text{NP}[1]}$, and thus by our assumption we have $A \in \text{P}^{\#\text{P}[1]}$. ■

4 Open Questions

We completely characterized the self-specifying classes SelfOutput and SelfPath as the sets $\leq_m^{\#\text{P}}$ -reducible to NP and P sets, respectively. Can one prove $\text{SelfOutput} = \text{SelfPath}$? That is, can one prove $\text{R}_m^{\#\text{P}}(\text{P}) = \text{R}_m^{\#\text{P}}(\text{NP})$, which would instantly imply by Theorem 3.3 that $\text{P}^{\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}$ (equivalently, $\text{P}^{\text{NP}[\mathcal{O}(1)]:\#\text{P}[1]} = \text{P}^{\#\text{P}[1]:\text{NP}[\mathcal{O}(1)]}$)?

Though Simon proved that the class PP remains the same regardless of whether or not the underlying machines are normalized,⁴ a number of recent papers have studied normalization in other contexts and it is now clear that in some contexts—e.g., the class BPP—classes may be different depending on whether or not there is a normalization requirement [15, 28, 27]. Recall that SelfOutput does remain the same whether defined with or without the requirement that the underlying machines be normalized. Does SelfPath remain the same class if its normalization requirement is removed? Clearly the resulting class

⁴Note that for probabilistic polynomial-time machines, normalization in terms of *number of choices per path in the model in which not every step is necessarily viewed as a choice node* and normalization in terms of *number of choices per path in the model in which every step is necessarily viewed as a choice node* are clearly equivalent. This holds true in the unbounded-error case. It is also true in the bounded-error case (i.e., BPP is the same class whether one defines it using the former normalization or the latter normalization). In particular, for PP it does not matter which normalization model one adopts; for PP, both are equivalent to the unnormalized model.

contains SelfPath, i.e., $P^{\#P[1]}$, and is contained in $P^{\#P}$ (which Vollmer and Wagner [46] showed equals $P^{\text{“NameofMiddlePath”}[1]}$; thus $P^{\text{“NameofMiddlePath”}[1]}$, though it is an upper bound for the unnormalized version of SelfPath, does not offer a tighter upper bound than $P^{\#P}$).

Acknowledgments

We are very grateful to Edith Hemaspaandra, Johannes Köbler, Jörg Rothe, and an anonymous referee for helpful suggestions.

References

- [1] M. Agrawal, R. Beigel, and T. Thierauf. Pinpointing computation with modular queries in the boolean hierarchy. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 322–334. Springer-Verlag *Lecture Notes in Computer Science #1180*, December 1996.
- [2] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2nd edition, 1995.
- [3] R. Beigel and R. Chang. Commutative queries. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 159–165. IEEE Computer Society Press, June 1997.
- [4] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55:80–88, 1982.
- [5] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13(3):461–487, 1984.
- [6] R. Book, T. Long, and A. Selman. Qualitative relativizations of complexity classes. *Journal of Computer and System Sciences*, 30:395–413, 1985.
- [7] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.
- [8] D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.
- [9] J. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2(1):67–76, 1991.
- [10] J. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990.
- [11] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

- [12] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.
- [13] F. Green. Lower bounds for depth-three circuits with equals and mod-gates. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, pages 71–82. Springer-Verlag *Lecture Notes in Computer Science #900*, March 1995.
- [14] F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán. The power of the middle bit of a #P function. *Journal of Computer and System Sciences*, 50(4):456–467, 1995.
- [15] Y. Han, L. Hemaspaandra, and T. Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.
- [16] L. Hemachandra and A. Hoene. On sets with efficient implicit membership tests. *SIAM Journal on Computing*, 20(6):1148–1156, 1991.
- [17] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. An introduction to query order. *Bulletin of the EATCS*, (63):93–107, 1997.
- [18] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Query order in the polynomial hierarchy. *Journal of Universal Computer Science*, 4(6):574–588, 1998.
- [19] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. $R_{1-tt}^{SN}(\text{NP})$ distinguishes robust many-one and Turing completeness. *Theory of Computing Systems*, 31(3):307–325, 1998.
- [20] L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, 1999.
- [21] L. Hemaspaandra and S. Jha. Defying upward and downward separation. *Information and Computation*, 121(1):1–13, 1995.
- [22] L. Hemaspaandra and M. Ogihara. Universally serializable computation. *Journal of Computer and System Sciences*, 55(3):547–560, 1997.
- [23] L. Hemaspaandra and H. Vollmer. The Satanic notations: Counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.
- [24] H. Hempel and G. Wechsung. On the power of #P reductions. Technical Report Math/Inf/97/2, Institut für Informatik, Friedrich-Schiller-Universität, Jena, Germany, January 1997.
- [25] U. Hertrampf. Acceptance by transformation monoids (with an application to local self-reductions). In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 213–224. IEEE Computer Society Press, June 1997.

- [26] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the 8th Structure in Complexity Theory Conference*, pages 200–207. IEEE Computer Society Press, May 1993.
- [27] U. Hertrampf, H. Vollmer, and K. Wagner. On balanced versus unbalanced computation trees. *Mathematical Systems Theory*, 29(4):411–421, July/August 1996.
- [28] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information and Computation*, 129(1):21–33, 1996.
- [29] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309, April 1980. An extended version has also appeared as: Turing machines that take advice, *L’Enseignement Mathématique*, 2nd series, 28, 1982, pages 191–209.
- [30] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26:363–379, 1989.
- [31] J. Köbler and T. Thierauf. Complexity-restricted advice functions. *SIAM Journal on Computing*, 23(2):261–275, 1994.
- [32] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.
- [33] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.
- [34] T. McNicholl. Some results on commutative oracles. Manuscript, May 1998.
- [35] M. Ogihara. On serializable languages. *International Journal of Foundations of Computer Science*, 5(3–4):303–318, 1994.
- [36] M. Ogiwara and L. Hemachandra. A complexity theory for closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.
- [37] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [38] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag Lecture Notes in Computer Science #145, 1983.
- [39] A. Selman. Reductions on NP and P-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.
- [40] J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, N.Y., January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.

- [41] S. Toda. *Computational Complexity of Counting Complexity Classes*. PhD thesis, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, 1991.
- [42] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
- [43] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
- [44] N. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Russian Academy of Sciences–Izvestiya–Mathematics*, 42(2):261–298, 1994.
- [45] H. Vollmer. *Komplexitätsklassen von Funktionen*. PhD thesis, Institut für Informatik, Universität Würzburg, Würzburg, Germany, 1994.
- [46] H. Vollmer and K. Wagner. The complexity of finding middle elements. *International Journal of Foundations of Computer Science*, 4:293–307, 1993.
- [47] K. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, 1986.
- [48] K. Wagner. A note on parallel queries and the symmetric-difference hierarchy. *Information Processing Letters*, 66:13–20, 1998.
- [49] O. Watanabe and S. Toda. Polynomial time 1-Turing reductions from #PH to #P. *Theoretical Computer Science*, 100:205–221, 1992.
- [50] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):76–82, 1991.