SIGACT News Complexity Theory Column 8

Lane A. Hemaspaandra
Dept. of Computer Science, University of Rochester
Rochester, NY 14627, USA lane@cs.rochester.edu

Introduction to Complexity Theory Column 8

The next issue will contain a guest column by Anne Condon. As to past columns, my sincere thanks to Bill Gasarch and Alan Selman for pointing out the following errors in SIGACT News Complexity Theory Column 6 (*Semi-Membership Algorithms: Some Recent Advances*, D. Denny-Brown, Y. Han, L. Hemaspaandra, and L. Torenvliet, Sept. 1994). The Jockusch citation on page 12 should be to his 1968 *Transactions of the AMS* paper, and the [Sel79] on page 17, line −12, should be [Sel82]. Our comments that the unique collapse result was the first result not about P-selective sets that followed from the theory of P-selective sets were perhaps more controversial than we thought. Alan Selman sent an eloquent discussion correctly pointing out that his work includes such results, e.g., "If E $\neq$ NE then there exist sets B and C such that (a) $B \in \mathrm{NP} - \mathrm{P}$ and $\overline{B} \not\leq_m^p B$, and (b) $B \leq_{ptt}^p C$ and $C \leq_{tt}^p B$ yet $C \not\leq_{ptt}^p B$." One could quibble that these results are essentially about P-selectivity, as the B rather crucially is P-selective. But in fact this indeed is part of the proof rather than of the theorem, and thus Professor Selman's examples indeed are excellent certificates of the broadness of the applicability of the study of the P-selective sets. Without further ado...

Guest Column: The Satanic Notations: Counting Classes Beyond #P and Other Definitional Adventures

Lane A. Hemaspaandra[1] *Heribert Vollmer*[2]

Abstract

We explore the potentially "off-by-one" nature of the definitions of counting (#P versus #NP), difference (DP versus DNP), and unambiguous (UP versus UNP; FewP versus FewNP) classes, and make suggestions as to logical approaches in each case. We discuss the strangely differing representations that oracle and predicate models give for counting classes, and we survey the properties of counting classes beyond #P. We ask whether subtracting a #P function from a P function it is no greater than necessarily yields a #P function.

[1]Department of Computer Science, University of Rochester, Rochester, NY 14627. Supported in part by grants NSF-CCR-8957604, NSF-INT-9116781/JSPS-ENGR-207, and NSF-CCR-9322513.

[2]Department of Mathematics, University of California, Santa Barbara, CA 93106. Email address: vollmer@math.ucsb.edu. Supported in part by grant NSF-CCR-9302057 and an Alexander von Humboldt fellowship. Permanent address: Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, D-97072 Würzburg, Germany.

1 Counting Classes Beyond $\#P$: $\#NP$ vs. $\#P^{NP}$ vs. $\#NP^{NP}$

Valiant [Val79] defined #P to be the class of functions counting the number of accepting paths of NP machines. What notation should one use to describe, for example, the number of accepting paths of the base level of an NP^{NP} machine? Even more to the point, what notation can one use to describe the number of accepting computations of a FewP [All86,AR88] machine—NP machines that are required to have a global polynomial bound on their number of accepting paths? For the latter, it would be tempting to use the notation #FewP, and in fact Wagner informally did so to describe his observation that the class Few [CH90] was exactly equal to $P^{\#FewP[1]}$. However, note that this has the quite perverse effect of making #P a potentially larger class than #FewP (the fact that $P \subseteq FewP$ notwithstanding!), since #P counts paths of NP machines and #FewP counts paths of a certain restriction of NP machines.

One natural solution would be to redefine things so as to use #NP to denote what is currently denoted #P. While this would seem quite intuitive, it is not a rigorous definition unless one were to rigorously define what it means to "compute paths of a machine from the class FOO." So rather than adopting this change, let us consider two rigorous approaches to defining the counting classes beyond #P.

The first approach we'll discuss is that presented in Valiant's seminal paper on #P. In this paper, he *does* define what he feels $\#\mathcal{C}$ should mean. His definition is (slightly simplified, but capturing the definition as it applies to all the classes we'll discuss):

Definition 1.1 [Val79] For any class $\mathcal{C}$, define $\#\mathcal{C} = \cup_{A \in \mathcal{C}} (\#P)^A$, where by $(\#P)^A$ we mean the functions counting the accepting paths of nondeterministic polynomial-time Turing machines having A as their oracle.

Note that this definition has some interesting properties. For example, when using an oracle (via Turing reductions) it does not matter whether the oracle or its complement is used. Thus, we have that, e.g., #NP = #coNP. Also, note that with this definition we have a crisp answer to the question of what notation to use to describe the number of accepting paths of the base NP machine of an NP^{NP} machine—the notation to use is #NP. Torán has studied the variant of Valiant's definition in which a certain "positive" requirement is added [Tor88, Chapter 2].

Valiant's above approach to counting is heavily machine-based, rather than predicate-based. In contrast, Toda ([Tod91a], see also the independent [Bur92]) proposed a predicate-based definition of a counting hierarchy, and this line has been followed in a number of recent papers [WT92, VW93,Vol94b,Vol94a]. One motivation for such a predicate-based approach is that counting is most natural in terms of predicates, and this definition allows one greater precision in specifying counting classes. Toda denoted application of his counting operator as "NUM·," thus avoiding any conflict with Valiant's notation. However, Toda's notation is (arguably) the "right" approach to counting, and thus later papers have used # (ambiguity alert: in some cases with no "·"!) in this context. To adopt the nice # notation but to ensure that this hierarchy and Valiant's won't be notationally confused, we'll follow the suggestion from [Vol94a] of using a "#·" to define the new classes, e.g., $\# \cdot \mathcal{C}$.

Definition 1.2 For any class $\mathcal{C}$, define $\# \cdot \mathcal{C}$ to be the class of (total, single-valued) functions f such that, for some $\mathcal{C}$-computable two-argument predicate R and some polynomial p, for every string x it holds that:

$$f(x) = ||\{y \mid p(|x|) = |y| \text{ and } R(x,y)\}||.$$

Informally, $\# \cdot \mathcal{C}$ is the class of functions counting the number of appropriate-length second arguments that cause some $\mathcal{C}$ predicate to be true for the given first argument. Note that it need not in general be the case that $\#\mathcal{C} = \# \cdot \mathcal{C}$. For example, it was noted above that $\#\mathrm{NP} = \#\mathrm{coNP}$, however we claim below that $\# \cdot \mathrm{NP} = \# \cdot \mathrm{coNP}$ if and only if $\mathrm{NP} = \mathrm{coNP}$. On the other hand, it is easy to note that $\#\mathrm{P} = \# \cdot \mathrm{P}$.

Also, it is not too hard to see that under this definition the class that counts accepting paths of $\mathrm{NP}^{\mathrm{NP}}$ machines is $\# \cdot \mathrm{P}^{\mathrm{NP}}$—and thus that $\#\mathrm{NP} = \# \cdot \mathrm{P}^{\mathrm{NP}}$. Though this might at first seem a bit strange, it in fact is quite natural. After all, this claim holds simply because the Stockmeyer/Wrathall [Sto77,Wra77] characterization of NP via P predicates itself relativizes, and thus it is hardly surprising that the paths of NP machines relativized by NP oracles should be described by the # operator operating on P relativized by NP oracles.

This generalization of # P allows convenient generalizations of other complexity classes. We say that:

1. $A \in \exists \cdot \mathcal{C}$ if there is some function $f \in \# \cdot \mathcal{C}$ such that $x \in A \iff f(x) > 0$.
2. $A \in \mathrm{C} \cdot \mathcal{C}$ if there is some function $f \in \# \cdot \mathcal{C}$ and some function g computable in polynomial time such that $x \in A \iff f(x) \geq g(x)$.
3. $A \in \mathrm{C_=} \cdot \mathcal{C}$ if there is some function $f \in \# \cdot \mathcal{C}$ and some function g computable in polynomial time such that $x \in A \iff f(x) = g(x)$.
4. $A \in \oplus \cdot \mathcal{C}$ if there is some function $f \in \# \cdot \mathcal{C}$ such that $x \in A \iff f(x) \equiv 1 \pmod 2$.
5. $A \in \mathrm{U} \cdot \mathcal{C}$ if the characteristic function of A is in $\# \cdot \mathcal{C}$
6. $A \in \mathrm{X} \cdot \mathcal{C}$ if the characteristic function of A can be written as the difference of two $\# \cdot \mathcal{C}$ functions.

It is well-known that the above operators capture, as special cases, the "standard" complexity classes NP, PP [Sim75], $\mathrm{C_=P}$ [Sim75,Wag86], $\oplus$P [PZ83,GP86], UP [Val76], and SPP [OH93, FFK94], respectively, via $\exists \cdot \mathrm{P} = \mathrm{NP}$, $\mathrm{C} \cdot \mathrm{P} = \mathrm{PP}$, $\mathrm{C_=} \cdot \mathrm{P} = \mathrm{C_=P}$, $\oplus \cdot \mathrm{P} = \oplus\mathrm{P}$, $\mathrm{U} \cdot \mathrm{P} = \mathrm{UP}$, and $\mathrm{X} \cdot \mathrm{P} = \mathrm{SPP}$. A careful look at the definitions and a little thought (or a look at [Vol94a]) reveal that for classes $\mathcal{C}$ and $\mathcal{C}'$ that fulfill some innocuous requirements (they have to include the class P and be closed downwards under many-one reductions) we have $\mathcal{C} \subseteq \mathrm{U} \cdot \mathcal{C} \subseteq \exists \cdot \mathcal{C}$ (thus, $\mathrm{U} \cdot \mathcal{C} = \mathcal{C}$ if $\mathcal{C}$ is closed under existential quantification), and

$$\# \cdot \mathcal{C} = \# \cdot \mathcal{C}' \iff \mathrm{U} \cdot \mathcal{C} = \mathrm{U} \cdot \mathcal{C}'.$$

This yields the following interesting consequences:

Proposition 1.3 1. $\#\mathrm{P} = \# \cdot \mathrm{NP} \iff \mathrm{UP} = \mathrm{NP}$.

2. $\# \cdot \mathrm{NP} = \# \cdot \mathrm{P}^{\mathrm{NP}} \iff \mathrm{NP} = \mathrm{UP}^{\mathrm{NP}} \iff \mathrm{NP} = \mathrm{coNP}$.[3]

Since $\# \cdot \mathrm{coNP} = \# \cdot \mathrm{P}^{\mathrm{NP}}$ (as we will show below), we have now (in contrast to Valiant's #-classes)

Proposition 1.4 $\# \cdot \mathrm{NP} = \# \cdot \mathrm{coNP} \iff \mathrm{NP} = \mathrm{coNP}$.

[3]Here we are tacitly using the fact that $\mathrm{U} \cdot \mathrm{P} = \mathrm{UP}$ itself relativizes, and thus $\mathrm{U} \cdot (\mathrm{P}^{\mathrm{NP}}) = \mathrm{UP}^{\mathrm{NP}}$.

Köbler, Schöning, and Torán [KST89] defined the class spanP of all functions counting the number of different outputs of nondeterministic polynomial-time transducers (i.e., Turing machines that output a value on every accepting path). Clearly, spanP $= \# \cdot \mathrm{NP}$, and thus, the second part of Proposition 1.3 is in fact their claim that spanP $= \# \mathrm{NP} \iff \mathrm{NP} = \mathrm{coNP}$.

The following result of Toda ([Tod91a, Theorem 4.1.6], see also [Bur92,VW93]) states that, perhaps surprisingly, the combination of a $\#$ operator with universal quantification is very powerful. Indeed, it is seemingly more powerful than a $\# \cdot \exists$ quantifier sequence. An informal explanation for this is that the $\#$ operator is somewhat existential-flavored and in some sense absorbs some of the power of a following $\exists$ quantifier. More concretely, the following result/proof reflects the power of a universal quantifier to limit focus to the (lexicographically) first of a certain object.

Theorem 1.5 $\# \cdot \mathrm{coNP} = \# \cdot \mathrm{P}^{\mathrm{NP}}$.

Proof sketch: The $\subseteq$ direction is clear, since the $\#$ operator is monotonic in its argument. For the $\supseteq$ inclusion, let $f \in \# \cdot \mathrm{P}^{\mathrm{NP}}$ be witnessed by a nondeterministic polynomial-time Turing machine M, asking questions to an NP oracle A. Let A be accepted by a machine M'.

Describe paths of the computation of M on some input x as follows. First, in the description, a sequence of zeroes and ones describes the nondeterministic choices of M ("left" or "right"); second, we have a list YES of pairs (q, p) of queries together with paths of M' (the path descriptions here are simply a sequence of zeroes and ones for the nondeterministic guesses of M'); third, we have a list NO of queries.

Define the following set B consisting of path descriptions as above, such that the following conditions hold:

1. In the simulation we are about to describe, the path is an accepting path and all queries that are asked appear in at least one of the lists, and no such queries appear in both lists. We simulate M on input x as follows. We use for M's nondeterministic moves the sequence of zeroes and ones in the first part of the path description. If M along this computation path poses a query to A, we look in our lists YES and NO, to see whether we find the query. If it appears (only) in list YES, then we continue the simulation as if the answer were "yes"; if it appears (only) in list NO, then we continue the simulation as if the answer were "no"; if it appears in both or neither list, then this path description does not meet this condition.

2. For every (q, p) from the list YES, if we simulate M' with input q deterministically according to path description p, then M' accepts.

3. Moreover, for every (q, p) from the list YES, there is no path description p' of another path of M' on input q such that p' is lexicographically smaller than p and p' encodes an accepting path of M' on input q as described in the previous item (that is: p is the lexicographically first path witnessing that $x \in A$).

4. For every query q from list NO, $q \notin A$.

Note that B is in coNP. This can be seen as follows. The simulations in items 1 and 2 are deterministic. The condition in item 3 is a typical polynomially-bounded "for-all" condition ("for all appropriate-length paths p', either p' is rejecting or p' is to the right of p"), and thus is in coNP. Item 4 is (since $A \in \mathrm{NP}$) clearly a coNP condition.

Finally, observe that every accepting path in M leads to exactly one path description in the set B. This shows that $f(x)$ is equal to the number of path descriptions in B associated with x. Thus, $f \in \# \cdot \mathrm{coNP}$. ∎

Though Proposition 1.4 above shows that $\# \cdot \mathrm{NP}$ and $\# \cdot \mathrm{coNP}$ almost certainly differ, they do not differ by much. In fact, polynomial-time computation is enough to smooth over the differences (as has been noted in [OH93,OTTW] for certain cases involving closure properties). In particular, if $f \in \# \cdot \mathcal{C}$ then for some polynomial q it holds that $f'(x) = 2^{q(|x|)} - f(x)$ is in $\# \cdot \mathrm{co}\mathcal{C}$. So, clearly, $\mathrm{P}^{\# \cdot \mathrm{NP}} = \mathrm{P}^{\# \cdot \mathrm{coNP}}$.

For function classes $\mathcal{F}$ and $\mathcal{F}'$, define

$$\mathcal{F} - \mathcal{F}' = \{\, h \mid \text{there exist functions } f \in \mathcal{F} \text{ and } f' \in \mathcal{F}' \text{ such that } h = f - f' \,\}.$$

By FP denote the class of polynomial-time computable functions. The just-mentioned fact can be stated as $\# \cdot \mathcal{C} \subseteq \mathrm{FP} - \# \cdot \mathrm{co}\mathcal{C}$, and we have

$$\# \cdot \mathrm{NP} - \mathrm{FP} = \# \cdot \mathrm{coNP} - \mathrm{FP} = \#\,\mathrm{P}^{\mathrm{NP}} - \mathrm{FP}.$$

Fenner, Fortnow, and Kurtz [FFK94] introduced the class GapP, which is the class of all functions computing the difference between the numbers of accepting and rejecting paths of nondeterministic polynomial-time Turing machines. It is easy to see from this definition that $\mathrm{GapP} = \#\,\mathrm{P} - \#\,\mathrm{P}$, and since by the above $\#\,\mathrm{P} \subseteq \mathrm{FP} - \#\,\mathrm{P}$ (and moreover, we know that $\#\,\mathrm{P}$ is closed under addition), we even have $\mathrm{GapP} = \#\,\mathrm{P} - \mathrm{FP}$. This holds in the presence of an arbitrary oracle. Thus, we can add to the above list of equalities the fact that $\# \cdot \mathrm{NP} - \mathrm{FP}$ is exactly GapP relativized by SAT:

$$\# \cdot \mathrm{NP} - \mathrm{FP} = \mathrm{GapP}^{\mathrm{NP}}.$$

Of course, it is well-known that $\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{NP} = \mathrm{coNP}$. Thus, it might be very tempting to conclude from Proposition 1.4 and Theorem 1.3 that $\mathrm{UP} = \mathrm{NP} \Rightarrow \mathrm{NP} = \mathrm{coNP}$, which, by the way, would resolve Part (b) of our forthcoming Open Question 3.2. However, such a conclusion relies on the reasoning that $\# \cdot \mathrm{P} = \# \cdot \mathrm{NP} \Rightarrow \# \cdot \mathrm{NP} = \# \cdot \mathrm{coNP}$, and there is no reasoning to think this implication is correct. Of course, the reasoning *is* correct, in light of the comments above on subtraction, if #P is closed under subtraction. However, as the closure of #P under subtraction has recently been shown [OH93] to imply that the polynomial hierarchy (and even $\mathrm{P}^{\#\mathrm{P}}$) collapses to UP, the whole issue is trivialized in this case anyway.[4]

[4]Lovers of closure properties of #P ([HO93] might be helpful if you don't belong to this odd group and yet would like to wade through this arcane footnote) will realize that something more subtle is going on here. In fact, the comments above on subtraction show that it would in fact suffice (though in any case it might not be necessary, and thus we're about to doubly digress, though this will lead us to an interesting research topic) for #P to have the property that if $f(x)$ is a polynomial-time computable function and $g(x)$ is a #P-computable function never greater than $f(x)$, then $f(x) - g(x)$ is a #P function. That is, are all non-negative functions in FP − #P themselves in #P? Though it is known that #P is exactly as unlikely to be closed under subtraction by P functions as it is to be closed under subtraction by #P functions [OH93], the issue here is not one of subtracting P functions from #P functions but rather is one of subtracting #P functions from P functions that they are no greater than, and as far as the authors know, this has not been studied. Of course, clearly if it were the case that subtracting a #P function from a P function it was no greater than would always yield a #P function, then UP = coUP. In fact, due to Simon's ([Sim75], see, e.g., [OH93] for a formal statement) implicit observation that $\mathrm{C_{=}P}$ languages can be realized via machines that on rejection alway have fewer than the (polynomial-time computable) number of paths they have for acceptance, it is clear that we can also in the case discussed conclude that $\mathrm{coNP} = \mathrm{C_{=}P}$ (and thus, by Sipser's and Lautemann's [Sip83,Lau83] result on simulating BPP with quantifiers and Toda and Ogihara's [TO92] result that the polynomial hierarchy is in $\mathrm{BP} \cdot \mathrm{C_{=}P}$, in this case we can conclude, for example, that the polynomial hierarchy collapses), and that UP = SPP (which is stronger than the UP = coUP conclusion mentioned above), where SPP is the exact counting class defined in [OH93,FFK94]. Ogihara (personal communication) noted previously that these two conditions are necessary conditions for all non-negative functions in #P − #P to be in #P. We commend to the reader the question of finding a statement about complexity classes that completely characterizes this issue. In particular, does $\mathrm{UP} = \mathrm{P}^{\#\mathrm{P}}$ completely characterize this issue (of course, from [OH93] we know that this is a sufficient

The literature contains theories both of those closure properties that #P is likely to lack [OH93] and those that it is likely to possess [Reg85,CGH+89,BGH90]. GapP is also known to have many closure properties [FFK94]. In particular, both #P and GapP have the following three closure properties:

> Take some function f, and then define h by (1) summing up exponentially many values of f, or (2) multiplying polynomially many values of f, or (3) computing binomial coefficients of f over polynomially bounded functions. If $f \in \#\mathrm{P}$, then $h \in \#\mathrm{P}$ (and if $f \in \mathrm{GapP}$, then $h \in \mathrm{GapP}$).

It was shown in [Vol94a] that if $\mathcal{C}$ is a so called *abstract polynomial-time complexity class* (apc for short, see [VW], i.e., $\mathcal{C}$ contains the class P and is closed under join and conjunctive and disjunctive reductions), then $\# \cdot \mathcal{C}$ has all three pleasant properties just described. Don't worry if you don't know the technical details of the definition of apc's; what is important here is only that P, NP, all the other classes from the polynomial hierarchy, PP, $\oplus$P (and its generalizations to MOD$_k$P for k prime, see [BGH90]) are all apc's; thus most counting classes beyond # P that we probably will ever deal with have the properties familiar to those who know about # P.

Having read the just-presented discussion of what is known about counting classes beyond #P, the reader may ask why this topic is studied at all—that is, why is it interesting to consider counting classes beyond # P? After all, don't we already have enough classes? One possible answer is the following: Studying complexity-theoretic operators (such as #, $\exists$, C, $\mathrm{C}_=$, U, X, and $\oplus$ above) and studying properties common to many complexity classes (see the apc notion) helps us to better understand what is really going on in a number of proofs, and thus often allows one to give unified proofs for results that stood separate before, and in some cases even allows one to improve what was known before. It is known for example, that Toda's famous result that the polynomial hierarchy is Turing reducible to # P [Tod91b] holds in a far more general setting [TO92]. (An extensive treatment of questions of this kind can be found in [VW].) Thus, we feel that introducing general concepts (such as operators and apc's) should not be viewed as increasing the already huge number of complexity classes, but rather should be viewed as useful in gaining a more unified view of the huge number of currently studied classes.

Moreover, a very abstract approach may sometimes allow one to explore the limitations of one's proof techniques. This point deserves a bit more explanation: We already mentioned Toda's result [Tod91b]. It can be shown [VW] that this result, as well as the closure of PP under intersection [BRS91], the inclusion of BPP in the polynomial hierarchy [Sip83,Lau83], and a large number of other results also about classes definable by some quantifiers applied to the class P do in fact not only hold for P but also for any apc $\mathcal{C}$ (that in some cases must also be closed under complementation). Though these results, which are of substantial importance to complexity theory, can be proved from the assumption of apc-ness, it is unfortunately obvious that P $\neq$ NP will never be proven in

condition)? We may state our question more formally as follows.

Open Question 1.6 Let F be the statement: "For every polynomial-time computable function f and every $g \in \#\mathrm{P}$ such that $(\forall x)[f(x) \geq g(x)]$, it holds that $h(x) \in \#\mathrm{P}$, where $h(x) = f(x) - g(x)$." Find a statement about complexity classes that *completely characterizes* whether F holds.

It is important to keep in mind here the difference between the value of a counting function, which has to do with the number of accepting paths of a machine (or the number of strings causing a predicate to be true), and the total number of paths (strings). That is, if we know that the g mentioned above can be implemented via some (nicely normalized) machine that has as most $f(x)$ paths, then the closure is easy to claim. The problem is the case in which $g(x)$ counts the accepting paths of a machine having more paths than $f(x)$ but that (by magic) satisfies the property that it never has more than $f(x)$ accepting paths.

this way. This is simply because there are apc's $\mathcal{C}$, e.g., $\mathcal{C}$ = PSPACE, for which $\mathcal{C} = \exists \cdot \mathcal{C}$. So, we see that the statement $\mathcal{C} \neq \exists \cdot \mathcal{C}$ does not follow *per force* from the apc axioms. To prove P $\neq$ NP, we will need other techniques. (We note in passing that there are also apc's $\mathcal{C}$ for which $\mathcal{C} \neq \exists \cdot \mathcal{C}$, for example, P^A for any oracle A such that $\mathrm{P}^A \neq \mathrm{NP}^A$.)

There remains one curious point regarding Valiant's seminal paper. Though Valiant's article [Val79] defines #NP as described in Definition 1.1, in his article he claims a certain set "can be easily shown to be" Turing-complete for #NP. The authors can clearly see that Valiant's set is many-one complete for # · NP, but that the set is Turing-complete for #NP seems to require Theorem 1.5. Thus, Valiant seems to have been aware of the possibility of an operator-like (or, predicate-like) definition of counting classes, as we discussed above, as well as the result presented above as Theorem 1.5.

2 Counting Inside #P: Fewness and Logspace

Fewness

The astute reader will notice that our preferred definitional scheme for "#" classes does not directly address the #FewP issue mentioned at the start of Section 1. Our suggestion is that the logical way to handle the case of "the count of a FewP machine" is not to change notations from #P to #NP, but rather is *to incorporate the fewness requirement into an appropriate operator*. In particular, we suggest the following definition. Note that this definition allows one to use "fewness" as a uniform concept, rather than as an ad hoc condition. Thus, it would be quite reasonable to use this definition to speak of, e.g., $\#_{few} \cdot$ BPP.

Definition 2.1 For any class $\mathcal{C}$, define $\#_{few} \cdot \mathcal{C}$ to be the class of (total, single-valued) functions f such that, for some $\mathcal{C}$-computable two-argument predicate R and some polynomials p and q, for every string x it holds that (1) $f(x) = ||\{y \mid p(|x|) = |y| \text{ and } R(x,y)\}||$, and, and (2) $f(x) \leq q(|x|)$.

Thus, Wagner's claim mentioned earlier becomes Few = $\mathrm{P}^{(\#_{few} \cdot \mathrm{P})[1]}$.

Counting in Log-space

Let us briefly touch upon the topic of how to define a # operator for logspace classes. Logspace counting classes were first considered by Álvarez and Jenner [AJ93]. In that paper, the authors justified (by presenting a number of natural complete problems, mostly from automata and formal language theory) the study of the classes # L and spanL, which they defined analogously to their polynomial-time counterparts as counting the number of accepting paths and different outputs, resp., of Turing machines operating in space logarithmic in the input length.

Álvarez and Jenner [AJ93] showed that both # L and spanL are contained in the class # P. However, one of the surprising results of their paper is that, while in fact # L is included in FP and can even be computed by polynomial-size circuits with only log-squared depth (# L $\subseteq$ NC^2), spanL turns out to be a very powerful class. It was shown that every # P function can be computed by a logspace-bounded Turing machine that is allowed to ask an oracle for the values of a spanL function.

In the meantime, it has been observed in a number of papers (see the excellent survey by Allender and Ogihara [AO94]), that counting accepting paths of nondeterministic logspace-bounded machines characterizes the complexity of the determinant. This is an additional motivation for studying this naturally arising complexity class.

The question we want to address here is: Can we find a natural operator such that, similar to the above result for polynomial time, spanL $= \# \cdot$ NL? It turns out that we can, using the following form of logspace-bounded machines.

A *2-1-Turing machine* (or, *one-way protocol machine* [Lan86]) is a Turing machine with two input tapes: first a (regular) input tape that can be read as often as necessary, and second, an additional (protocol) tape that can be read only once (from left to right).

Define 2-1-L to be the class of all two argument predicates R that can be computed by logspace-bounded 2-1-TMs such that in the initial configuration, the first argument of the relation is on the regular input tape, and the second argument is on the one-way input tape. Finally,

Definition 2.2 [Bur92,Bur94] $\# \cdot$ L is the class of (total, single-valued) functions f such that, for some two-argument predicate $R \in$ 2-1-L and some polynomial p, for every string x it holds that: $f(x) = ||\{y \mid p(|x|) = |y| \text{ and } R(x, y)\}||$.

Similarly, one can define $\# \cdot$ NL and $\# \cdot$ coNL, and so on. Observe that all classes $\mathcal{C}$ that we considered in Section 1 are supersets of P for which defining $\# \cdot \mathcal{C}$ as in Definition 1.2 or by using 2-1-$\mathcal{C}$ predicates analogous to the approach of Definition 2.2 makes no difference (as the machine could start by deterministically copying its protocol tape's contents onto its work tape). This is the reason why we will use the same notation "$\# \cdot$" without becoming inconsistent.

Now, it can be shown [Bur92] that $\# \cdot$ L $= \#$ L and indeed that

$$\# \cdot \text{NL} = \text{spanL}.$$

Of course, one can now again take other operators, apply them to $\# \cdot$ L (or spanL) and see what classes one obtains. Jung [Jun85] showed that the class of problems decidable in probabilistic logspace is the same as the class of problems decidable by probabilistic logspace machines that in addition run in polynomial time. This shows that probabilistic logspace sets can be defined by asking whether a certain $\#$ L function is greater than some threshold (this is similar to the characterization of PP as PP $=$ C $\cdot$ P, mentioned above). We won't go into detail here about this and other operators. There is however one point we want to stress: the remarkable Immerman-Szelepcsényi [Imm88,Sze88] result about closure of NL under complementation of course does not necessarily hold for 2-1-NL. This is simply because in their proofs it is essential that the whole input can be read several times. Thus, it is not known whether $\# \cdot$ NL $= \# \cdot$ coNL, and the following surprising result from [Bur92] shows that this is very unlikely.

Theorem 2.3 $\# \cdot$ coNL $= \#$ P.

Thus, $\# \cdot$ coNL $= \# \cdot$ NL implies $\# \cdot$ NL $= \#$ P, and this in turn yields (applying the operator U to both sides of the equation) NL $=$ UP, contradicting our beliefs.[5]

Sketch of the proof of Theorem 2.3: (This proof is essentially Lange's proof of a related result [Lan86, Theorem 3.1].) Take #3SAT (the number of satisfying assignments of a propositional formula in conjunctive normal form with at most 3 literals per clause). This problem is $\#$ P-complete. We will sketch a logspace 2-1-TM M that solves this problem. M has in its initial configuration the encoding of a 3CNF formula on its regular input tape, and an encoding of an assignment on its one-way input tape. M has to accept if and only if this assignment is satisfying. M uses its universal guesses to branch for all clauses. To verify one clause, M now has only to read the assignment once from left to right, and accept if the guessed clause is satisfied. ∎

[5]In [AJ93] it was shown that spanL $= \#$ P even implies that NL $=$ NP.

So we see again that the quantifier sequence $\# \cdot \forall$ is very powerful.

Denoting the set of all deterministic logspace-computable functions by FL, we can see that (analogously to the remarks for $\# \cdot \mathrm{NP}$ following Theorem 1.5) $\# \cdot \mathrm{coNL} \subseteq \mathrm{FL} - \# \cdot \mathrm{NL}$, which directly implies $\# \cdot \mathrm{coNL} - \mathrm{FL} = \# \cdot \mathrm{NL} - \mathrm{FL} = \# \cdot \mathrm{NL} - \# \cdot \mathrm{NL}$, that is,

$$\#\mathrm{P} - \#\mathrm{P} = \#\mathrm{P} - \mathrm{FP} = \# \cdot \mathrm{NL} - \mathrm{FL} = \# \cdot \mathrm{NL} - \# \cdot \mathrm{NL},$$

or, using the equivalences we've discussed,

$$\mathrm{GapP} = \mathrm{spanL} - \mathrm{FL}.$$

This is a very nice formalization of the fact that spanL is a very hard logspace counting class.

3 Further Adventures in Notation

Difference and Unambiguous Classes

The class DP is defined as $\{L \mid (\exists L_1, L_2)[L_1 \in \mathrm{NP} \text{ and } L_2 \in \mathrm{NP} \text{ and } L = L_1 - L_2]\}$. (Historical note: the original notation was D^{P}, but DP is now the common usage.) This class was defined by Papadimitriou and Yannakakis [PY84], and forms the second level of the boolean hierarchy over NP [CGH+88,CGH+89]. But what notation would one use to define the class of differences of FewP sets (FewP being the "polynomially bounded accepting path count" subset of NP)? After all, using DP leaves us little wiggle room in terms of shrinking the class. It seems to us the logical solution would be to define:

Definition 3.1 For any class $\mathcal{C}$, we denote by $\mathrm{D}(\mathcal{C})$ (alternatively, $\mathrm{D}\cdot\mathcal{C}$) the class $\{L \mid (\exists L_1, L_2)[L_1 \in \mathcal{C} \text{ and } L_2 \in \mathcal{C} \text{ and } L = L_1 - L_2]\}$.

Thus, for example, the notation we suggest as being the most logical for the class currently called DP would be either of D(NP) or $\mathrm{D} \cdot \mathrm{NP}$.

Regarding FewP (or is it FewNP?), there has also been some interesting confusion as to the best name. Consider the subset of NP consisting of those NP sets accepted by some nondeterministic polynomial-time Turing machine that on each input has a polynomially bounded number of accepting paths (the polynomial depending on the machine, but uniform over all inputs). Allender [All86] defined this class, and called it FewNP. But in time, probably driven by the example of Valiant's notation UP (note: not UNP) for the unambiguous version of NP, the standard notation for this "fewness version of NP" has become (both in papers by Allender and by others) FewP. To our taste, this shift seems correct. That is, the most elegant way of viewing this class is probably as a "polynomially-accepting-path-bounded polynomial-length nondeterminism" operator "few" applied to P, just as one can view NP as a polynomial-length nondeterminism operator N (or the more recent use of $\exists$ in this context) applied to P. It is interesting to note that the operator approach that is currently fashionable (and is suggested in this article as being useful in such contexts as difference classes and counting classes) was heavily foreshadowed by Valiant's own 1979 #P paper, as he did define an inductive hierarchy (though it was based on oracles rather than operators).

For lovers of open problems, we mention that extremely little is known about the relationship between unambiguous computation and the collapse of the polynomial hierarchy. In fact, all four of the issues below are open.

Open Question 3.2 (a) Does P=UP imply that the polynomial hierarchy collapses? (b) Does UP=NP imply that the polynomial hierarchy collapses? (c) Does P=FewP imply that the polynomial hierarchy collapses? (d) Does FewP=NP imply that the polynomial hierarchy collapses?

Bounded Queries

Regarding bounded queries to an oracle, what is the "right" notation? For example, is the θ_2^p [Wag90] level of the polynomial hierarchy—those sets that can be computed by P machines given $\mathcal{O}(\log n)$ queries to an NP oracle—best denoted by $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log n)]}$, or by $\mathrm{P}^{\mathrm{NP}}[\mathcal{O}(\log n)]$? Some authors prefer the latter, feeling that the restriction applies to the base machine, and not to the oracle. Other authors prefer the former, reasoning that (1) in many settings, the restriction does apply to the oracle as well as the base machine (e.g., in cases where one bounds the number of strings *in* the set that are queried [Lon85,Hem89]), and (2) the former notation allows one to put differing bounds on each of two components of an oracle. Both notations seem clear enough.

Reducibilities

As to the ultimate question of what notation to use for reductions (e.g., is polynomial-time many-one reducibility best denoted $\leq_m^p$, or $\leq_m^{\mathrm{P}}$, or $\leq_m^{P}$, or ...), this issue seems to be a matter of religion for many. A quick count, based on the pages of recent "Structure in Complexity Theory" conference proceedings suggests that $\leq_m^p$ has the lead in this race. The issue is a bit murky. For the case of many-one reductions, the superscript of $\leq_m$ refers more to a computational power/mode than to a language class, and using lower-case letters seems more appropriate for this. However, for the case of Turing reductions, the superscript of $\leq_T$ really does refer to a class (that is, it is natural and generally consistent with current usage to define, for any class $\mathcal{C}$ whose relativizations are well-defined, $A \leq_T^{\mathcal{C}} B \iff A \in \mathcal{C}^B$), and thus using upper-case letters seems more appropriate in this case.

Function Classes

Finally, for a given complexity class what is the best way to denote its functional version? For example, in the case of functions computable deterministically in polynomial time, which of FP or PF is preferable? Both can be found in the literature. We feel that FP is slightly more natural for two reasons. In the light of [VW], where an operator F is introduced that transforms set classes into function classes and has the pleasant property that $\mathrm{F}\cdot\mathrm{P} = \mathrm{FP}$, the first alternative seems attractive. Also, since classes typically have their baggage hanging out towards the right, it is more graceful to have the F on the left, e.g., $\mathrm{D^PF}$ looks a bit strange.

Acknowledgments. For helpful discussions and pointers to references, we are indebted to H.-J. Burtschick, E. Hemaspaandra, M. Ogihara, J. Rothe, and K. W. Wagner.

References

[AJ93] C. Álvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[All86] E. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 1–11. Springer-Verlag *Lecture Notes in Computer Science #223*, June 1986.

[AO94] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 267–278. IEEE Computer Society Press, 1994.

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.

[BGH90] R. Beigel, J. Gill, and U. Hertrampf. Counting classes: Thresholds, parity, mods, and fewness. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 49–57. Springer-Verlag *Lecture Notes in Computer Science #415*, February 1990.

[BRS91] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. In *Proceedings of the 23nd ACM Symposium on Theory of Computing*, pages 1–9. ACM Press, May 1991.

[Bur92] H.-J. Burtschick. Dokumentation der Ergebnisse des Arbeitsteffens Komplexitätstheorie in Georgenthal/Thür. vom 24.2.1991–1.3.1991, September 1992. Unpublished notes.

[Bur94] H.-J. Burtschick. Comparing counting-classes for logspace, one-way logspace, logtime, and first-order. Technical Report 94-39, Forschungsberichte des Fachbereichs Informatik, Technische Universität Berlin, Berlin, Germany, 1994.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.

[CGH+89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

[CH90] J. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[GP86] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[HO93] L. Hemachandra and M. Ogiwara. Is #P closed under subtraction? In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Essays and Tutorials*, pages 523–536. World Scientific Press, 1993.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[Jun85] H. Jung. On probabilistic time and space. In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming*, pages 281–291. Springer-Verlag *Lecture Notes in Computer Science #194*, 1985.

[KST89] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26:363–379, 1989.

[Lan86] K.-J. Lange. Two characterizations of the logarithmic alternation hierarchy. In *Proceedings of the 12th Symposium on Mathematical Foundations of Computer Science*, pages 518–526. Springer-Verlag *Lecture Notes in Computer Science #233*, August 1986.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 14:215–217, 1983.

[Lon85] T. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, 14(3):585–597, 1985. Erratum appears in the same journal, 17(3):628.

[OH93] M. Ogiwara and L. Hemachandra. A complexity theory for closure properties. *Journal of Computer and System Sciences*, 46:295–325, 1993.

[OTTW] M. Ogiwara, T. Thierauf, S. Toda, and O. Watanabe. On closure properties of #P in the context of PF∘#P. *Journal of Computer and System Sciences*. To appear.

[PY84] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

[PZ83] 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.

[Reg85] K. Regan. Enumeration problems. Manuscript, 1982; revised 1985.

[Sim75] 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.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335, 1983.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.

[Tod91a] S. Toda. *Computational Complexity of Counting Complexity Classes*. PhD thesis, Tokyo Institute of Technology, Department of Computer Science, Tokyo, Japan, 1991.

[Tod91b] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Tor88] J. Torán. *Structural Properties of the Counting Hierarchies*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1988.

[Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Vol94a] H. Vollmer. *Komplexitätsklassen von Funktionen*. PhD thesis, Universität Würzburg, Institut für Informatik, Würzburg, Germany, 1994.

[Vol94b] H. Vollmer. On different reducibility notions for function classes. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, pages 449–460. Springer-Verlag *Lecture Notes in Computer Science #775*, February 1994.

[VW] H. Vollmer and K. Wagner. Complexity classes of optimization functions *Information and Computation*. To appear.

[VW93] H. Vollmer and K. Wagner. The complexity of finding middle elements. *International Journal of Foundations of Computer Science*, 4:293–307, 1993.

[Wag86] K. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, 1986.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

[WT92] O. Watanabe and S. Toda. Polynomial time 1-Turing reductions from #PH to #P. *Theoretical Computer Science*, 100:205–221, 1992.