Order Number 9521313

Complexity classes: Tree models, operators and oracles

Sharma, Kiron, Ph.D.

City University of New York, 1995

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

4

COMPLEXITY CLASSES:

TREE MODELS, OPERATORS AND ORACLES

by

KIRON SHARMA

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1995

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

1/27/95
Date

Professor Stathis Zachos (Chair of Examining Committee)

1/27/95
Date

Professor Stanley Habib (Executive Officer)

Supervisory Committee

Professor Stathis Zachos
(Adviser, Brooklyn College)

Professor Stanley Habib
(Executive Officer, CCNY/GSUC)

Professor Kenneth McAloon
(Brooklyn College)

Dr. Stuart Haber
(Surety Technologies)

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

COMPLEXITY CLASSES:
TREE MODELS, OPERATORS AND ORACLES.

by
Kiron Sharma

Adviser: Professor E. Zachos

Our fundamental machine model is a NDTM. We consider complexity classes involving polynomial time.

Tree Models :

All the possible computation paths of a NDTM can be represented in the form of a tree structure. In general we have a different length for each path and a different fan-out at each internal node.

We present simplified padding and extension techniques used in order to obtain a uniform model (complete binary computational trees) for: (a) PP Computations, (b) #P Computations and (c) Computations belonging to sub-classes of the class #P. Similar techniques can be extended to most complexity classes within PH.

Operators :

We consider complexity class operators corresponding to nondeterministic, probabilistic and counting classes. The characterization of classes in terms of operators is closely related to the oracle characterization. We study the calculus of operators and in the process we identify some interesting properties and the behavior of certain widely used operators applied on complexity classes. We identify operators which yield unexpected collapse results, when these operators are assumed to satisfy certain properties. The goal is to argue about complexity class inclusions using algebraic techniques. We introduce two new *sensible* operators corresponding to the classes (NP $\cap$ co-NP) and ZPP respectively. The operator "ZP$\bullet$" provides us with a medium to study the Las-Vegas version of complexity classes. It turns out that the well known Graph Isomorphism problem is in the Las-Vegas version of NP i.e., the class ZP$\bullet$NP, exactly as the primality problem is in the Las-Vegas version of P i.e., ZPP.

Oracles :

We show that a BPP oracle is not very powerful. Amoung other things, we prove: NP using a BPP oracle is (lower than) contained in ZPP using a D^P oracle. We use quantifiers and combinatorial techniques to prove this result.

ACKNOWLEDGMENTS

It has been a long and seemingly endless journey through Graduate School. There have been many hurdles too. I could not have covered the difficult terrain and achieved my goals without the support, assistance and guidance of many people.

I would like to express my deep gratitude to Professor Stathis Zachos for his motivation, inspiration, unstinting advice and criticism. His perception of Theoretical Computer Science helped me obtain deep insight into the field and his perseverance fostered my transition from a student to a researcher. He has been my mentor in the real sense. When not in town he spent endless hours on the telephone with me, probing the results and polishing the final manuscript. I also thank Issidora, a close and dear friend, for her constant concern and support.

I am indebted to the members of the examining committee. Professor Stanley Habib took a great deal of personal interest and made many valuable suggestions. His masterful tutelage was critical for the refinement of my manuscript. Dr. Stuart Haber provided many significant ideas and pointers to new research paths during the preliminary phase of this work. Stuart also spent a great deal of time in order to provide critical suggestions for the final manuscript. He was always willing to squeeze out time from his busy schedule for discussion and for coming to the proposal and dissertation defense presentations. I can't thank him enough. Professor Ken McAloon provided many meaningful ideas and fresh

perspectives on my results. I thank him also for agreeing to serve on my committee in spite of his myriad other commitments.

Special thanks go to Joseph Driscoll and Rita Wertman who have helped me avoid the bureaucratic maze at every step.

My mother and my father have provided tremendous encouragement and total support. They never let me lose faith in myself. Their blessings and good wishes enabled me to complete this dissertation. I thank my sister Jyoti, her family, my brother Deepak and his family for their concern and support through the duration of this study. I also thank Deepak for painstakingly proofreading my dissertation and providing many valuable suggestions.

But I am most of all indebted to my husband Sanjiv, for love, total support, friendship, ideas and long hours he spent helping me prepare this manuscript. Without Sanjiv's total commitment to my completing the degree, I most certainly could not have done it.

There remain two little people to mention in my list of acknowledgements, and these are my daughters Shivani and Ambika. They have shown remarkable patience during the time I could not devote to them in the hope that I will finish SOON. They have been loving children and a source of solace during an otherwise strenuous period.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

I. COMPLEXITY CLASSES -- AN OVERVIEW

1. INTRODUCTION

The theory of Computational Complexity considers in general the basic resources of time and space as measures of complexity for computations. The concept of resources used by a computation has led to more efficient algorithms and to the concept of computationally feasible problems.

Complexity classes are specified by several parameters:

a) Underlying model of computation (*TM, High level programming, Boolean circuits* etc.).

b) Mode of computation (i.e., *Deterministic, Nondeterministic,* etc.).

c) Bounds on resources (*time and space* in general). A bound is defined as a function of the size of the input i.e., f: N-->N.

In general, the main goals of Computational Complexity theory are:

- to introduce classes of problems which have similar complexities with respect to a specific computational model, mode and measure of complexity; and,
- to study intrinsic properties and interrelations of such classes of problems.

In order to achieve these two goals, there are two approaches:

✧ First, Algorithmic :

The Algorithmic view involves solving "natural" problems efficiently. In addition, it also involves classification of these algorithms based on the computational model, resources used, etc..

✧ Second, Structural :

With the structural point of view we are concerned with intrinsic properties of complexity classes, including

- relationships between classes,
- implications of several hypotheses about complexity classes, and
- identification of structural properties of sets that affect their computational complexity.

Throughout the discussion, in this text we are dealing with Structural Computational Complexity.

It has been traditionally assumed that an algorithm is practical, efficient and tractable iff it can be rendered as a polynomial time-bounded Deterministic Turing Machine (DTM). Any algorithm whose time complexity function cannot be bounded by a polynomial, e.g., whose time complexity is exponential or even worse may be regarded as an inefficient algorithm. Problems for which there are no efficient algorithma are called intractable.

A Nondeterministic Turing Machine (NDTM) is informally viewed as a Turing Machine (TM), which at every step has more than one possible choice of behavior (e.g., two choices, three choices ...., m choices). We consider NDTMs only as ***acceptors*** and we restrict or bound the number of computation steps. Another view of a nondeterministic algorithm is a two stage algorithm; the first stage being a guessing stage (i.e., *guess a structure or a solution*), and the second stage being a checking, substantiation stage which proceeds in a normal deterministic manner (see [11]). Nondeterministic machines are more powerful than their

deterministic counterparts and this is because of a very weak "input-output behavior": we regard a nondeterministic computation as accepting its input if there is an accepting path. A nondeterministic algorithm is basically a definitional device to capture the notion of *verifiability*, rather than a realistic method for solving decision problems. Instead of having just one possible computation on a given input, it offers, one for each possible guess.

The computational activity corresponding to a nondeterministic computation lends itself naturally to a representation in form of a tree structure. The root node represents the initial state and the starting computation. Each derivative of a node represents one of the possible nondeterministic choices available at the relevant computational step and finally each leaf node provides a possible final output. It is required that for each computational path the number of steps is bounded (i.e., the depth of computation is bounded for example, by a polynomial in the size of the input). On the other hand, the total computational activity pertaining to the whole tree structure can be exponential in the size of the input.

Since a NDTM is considered as an ***acceptor***, it is said that a NDTM N accepts an input x, if there is one or more computation paths that result in acceptance (i.e., one or more leaves, providing the final output, result in acceptance or "Yes", while the other leaves, or the other computational paths may result in a nondecisive answer or a "NO"). It is said that N rejects an input x if no path results in acceptance. The language that N recognizes (decides correctly) is denoted by L(N).

The following figure represents a nondeterministic computation as a tree structure. We assume that the depth is bounded by a polynomial in the size of the input.

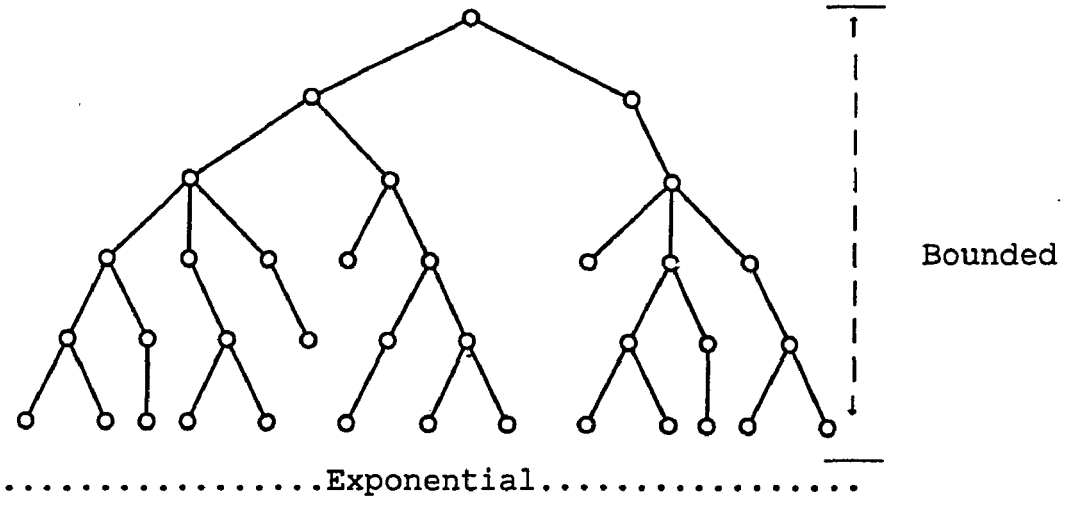

FIG 1. A NONDETERMINISTIC COMPUTATION

Without loss of generality, we can reduce the degree of nondeterminism i.e., we can assume a nondeterministic computation to be represented by a binary tree. [22].

As noted above, a complexity class is defined based on the following parameters,

a) Model of computation

b) Mode of computation and

c) Bounds on resources.

Here we will be using the model of computation (a) a Turing Machine, M. We will now discuss some useful modes (b) and resource bounds (c).

On an input say x, a bound is set, such that M uses no more than f(|x|) units of a specified resource, where (*f:N --> N, depends on the size of the input x*). If n is the length (size) of the input, then the amount of resource under consideration is bounded by f(n). TIME(f) is a set of languages decided by a DTM operating within time-bound f(n), i.e., M has no computational paths which are longer than f(n) on inputs of length n. NTIME(*f*) is a set of languages decided by a NDTM, operating within time-bound f(n). Similarly, SPACE(f) and NSPACE(f) are sets of languages,

when working "Space" is considered as a resource (input/output space is not counted).

The class P (*Deterministic Polynomial time computable*) is:

$$P = \bigcup_{j>0} \text{TIME}(n^j).$$

The Union of all these classes provides us with the set of all languages that are decidable by Turing Machines in some **polynomial time** bound (i.e., if L is a polynomially decidable language then it is in TIME(n^k) for some integer k>0). It can be said that an algorithm requires polynomial time if it runs in $O(n^k)$ steps for k>0. The importance of P resides in the opinion that P coincides with the class of problems that can be realistically solved by computers (computationally tractable). In practice, it can be said that all problems solvable by algorithms requiring a superpolynomial number of steps are computationally intractable (a function is superpolynomial if it grows faster than n^k for any fixed k). For many problems not known to have deterministic polynomial time (P) algorithms, there exist fairly simple nondeterministic (NP) algorithms.

The nondetermistic counterpart of P is NP which is:

$$NP = \bigcup_{j>0} NTIME(n^j).$$

Similarly, we can define PSPACE and NPSPACE, which are equal by a theorm of Savitch, see [22].

Besides Deterministic and Nondeterminstic Turing Machines, we will also discuss Probabilistic Turing Machines. In the last 15 years probabilistic algorithms have been proposed that determine the exact solution of problems faster than existing deterministic algorithms. Probabilistic algorithms may, when given an input, produce a correct answer, a wrong answer or no answer at all. The probabilistic complexity classes arise from the fact that there are many computational problems for which the most natural algorithmic approach is based on ***randomization***. Such an algorithm uses a full instruction set of deterministic algorithms and additionally has the ability to "flip unbiased coins" or guess randomly. **The outcome of such algorithms can be erroneous, but we have to deal with such errors. This is part of the trade-off between running time and accuracy of the computation.** Intuitively, probabilistic algorithms are deterministic ones, that make random choices in the course

of their execution. For a fixed input different runs of a probabilistic algorithm may thus give different results. Probabilistic algorithms with a bounded probability of error can be used to obtain a correct answer with a probability as high as desired by simply iterating the algorithm a limited number of times. Our probabilistic model of computation is essentially a tree similar to that corresponding to a NDTM with a different interpretation of its branchings, and a different interpretation of the set of leaf outcomes.

Another concept is a Turing Machine or an algorithm equipped with an oracle. Such an algorithm which is equipped with an oracle may repeatedly stop to query the oracle about membership of a specific string in the set represented by the oracle, obtain the answer from the oracle (this is considered as one step) and continue its normal computation (section 4.).

The theory of polynomial time complexity classes classifies the inclusion/ separation structure of deterministic/ nondeterministic/ probabilistic polynomial complexity classes (with or without oracles).

2. THE COMPUTATIONAL MODEL AND DEFINITIONS

The computational model throughout the discussion is a polynomial-time bounded Nondeterministic Turing Machine (NDTM) *M* viewed as an ***"acceptor"***. We assume that the tape alphabet (without loss of generality) is $\Sigma = \{0,1\}$. We restrict each computation path to a length of $p(|x|)$, p being a polynomial, and $|x|$, the length of input x. *Each algorithm or TM can be equipped with a counter that counts the number of computation steps performed and if the counter exceeds $p(|x|)$ then M rejects the input x. Thus, the final answer is either accept or reject.* The time for a computation is simply the number of steps made before the machine halts. Acceptance of a word $x \in \Sigma^*$ is defined in terms of probabilities $Pr[Acc_M(x)]$ and $Pr[Rej_M(x)]$ that a path may lead to acceptance or rejection.

For any class *C*, we can define a class co-*C* (complement-*C*):

co-*C* = $\{\Sigma^*-L$:L is a language over the alphabet Σ and $L \in C\}$ i.e., the class co-C is the set of all languages whose complements are in the class C.

We now define specific complexity classes.

Definition 1.

The class P :

A language $L \subseteq \Sigma^*$ is in P iff for some NDTM, M and all $x \in \Sigma^*$,

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] = 1 \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] = 1$$

This is equivalent to the existence of a Deterministic Turing Machine (DTM), that recognizes L i.e.,

$P = \{L \subseteq \Sigma^* : \exists$ poly-time DTM accepting $L\}$.

Definition 2.

The class NP :

A language $L \subseteq \Sigma^*$ is in NP iff for some NDTM M and all $x \in \Sigma^*$,

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] > 0 \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] = 1$$

$NP = \{L \subseteq \Sigma^* : \exists$ poly-time NDTM accepting $L\}$.

Definition 3.

The class co-NP :

A language $L \subseteq \Sigma^*$ is in co-NP iff $L' = \Sigma^* - L$ and $L' \in$ NP.

In other words, a language $L \subseteq \Sigma^*$ is in co-NP iff for some NDTM M and all $x \in \Sigma^*$,

$$x \in L \dashrightarrow \Pr[Acc_M(x)] = 1 \quad \text{and}$$

$$x \notin L \dashrightarrow \Pr[Rej_M(x)] > 0.$$

Definition 4.

The class ΔP :

$$\Delta P = NP \cap \text{co-NP}.$$

By imposing certain conditions on the acceptability criteria of a NDTM, we can define probabilistic complexity classes and counting classes. Following the form given above, that for some NDTM M and all $x \in \Sigma^*$ the conditions given in [12] for L to belong to Probabilistic Complexity classes RP, BPP and PP are given below.

Definition 5.

The class RP *(Random Polynomial time)*:

A language $L \subseteq \Sigma^*$ is in RP iff for some NDTM, M and all $x \in \Sigma^*$, there exist $\delta>0$:

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] > \tfrac{1}{2}+\delta \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] = 1$$

(Note: The restriction imposed on the NDTM accepting a language in RP, is that either at least a fixed fraction more than half the computations are accepting computations, or all computations are rejecting computations.)

Definition 6.

The class co-RP :

A language $L \subseteq \Sigma^*$ is in co-RP iff for some NDTM, M and all $x \in \Sigma^*$, there exist $\delta>0$:

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] = 1 \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] > \tfrac{1}{2}+\delta$$

(Note: The restriction imposed on the NDTM accepting a language in co-RP, is that either all computations are accepting computations or, at least a fixed fraction more than half the computations are rejecting computations.)

Definition 7.

The class ZPP *(Zero-error Probabilistic Polynomial)*:

$$ZPP = RP \cap co\text{-}RP.$$

Definition 8.

The class BPP *(Bounded error Probabilistic Polynomial)*:

A language $L \subseteq \sum^*$ is in BPP iff for some NDTM, M and all $x \in \sum^*$, there exist $\delta > 0$:

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] > \tfrac{1}{2}+\delta \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] > \tfrac{1}{2}+\delta$$

(Note: The restriction imposed on the NDTM accepting a language in BPP is that, either at least a fixed fraction more than half the computations are accepting computations or, at least a fixed fraction more than

half the computations are rejecting computations.)

Definition 9.

The class PP *(Probabilistic Polynomial Time):*

A language $L \subseteq \Sigma^*$ is in PP iff for some NDTM, M and all $x \in \Sigma^*$,

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] > \tfrac{1}{2} \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] > \tfrac{1}{2}$$

(Note: The restriction imposed on the NDTM accepting a language in PP, is that either more than half the computations are accepting computations or more than half the computations are rejecting computations. This restriction is however insignificant as e.g., the " $>\frac{1}{2}$" of the last line can equivalently be replaced by "$\geq\frac{1}{2}$")

It is known that RP $\subseteq$ NP $\subseteq$ PP and RP $\subseteq$ BPP $\subseteq$ PP. We do not know of any inclusion relationship between NP and BPP.

For the classes RP, co-RP and BPP we can arbitrarily increase the probability of correctness by repeating the algorithm. This property is known as "*Robustness*" and has

been discussed in detail in [38]. The robustness property does not extend to the class PP, since the probability of correctness **cannot** be substantially improved by repeating the algorithm.

Many papers in the literature [13,17,19] have considered definitions of complexity classes using quantifiers in place of machines. We shall use the fixed length quantifiers $(\exists y \cdot |y|=k)$, and $(\forall y \cdot |y|= k)$ together with $(\exists^{+} y \cdot |y|=k)$, and $(\text{Я} y \cdot |y|= k)$ which are informally interpreted to mean "*For most strings y,* such that $|y|=k$..." and "*For over half of the strings* y such that $|y|=k$ ...". We provide formal definitions of quantifiers $\exists^{+}$, Я.

Definition 10.

The quantifier $\exists^{+}$ (*the overwhelming majority quantifier*):

For a predicate $P(x,y)$,

$\exists^{+} y P(x,y)$ denotes that there is a $\delta > 0$ so that for all(inputs) x:

$\Pr(\{y|P(x,y)\}) > \frac{1}{2}+\delta$.

Definition 11.

The quantifier Я (*the majority quantifier*):

For a predicate P(x,y),

ЯyP(x,y) denotes that for all(inputs) x:

$Pr(\{y|P(x,y)\}) > \frac{1}{2}$.

Using these quantifiers alternative definitions of the above classes [39] can be given as follows:

For all languages $L \subseteq \Sigma^*$, L is respectively in (i) NP, (ii) P, (iii) RP, (iv) BPP, and (v) PP, iff there exists some polynomial-time predicate P(.,.) and a polynomial p(.) such that for all $x \in \Sigma^*$,

(i) The class NP:

$x \in L \rightarrow (\exists y \cdot |y| = p(|x|))\ P(x,y)$ and

$x \notin L \rightarrow (\forall y \cdot |y| = p(|x|))\ \neg P(x,y)$

(ii) The class P:

$x \in L \rightarrow (\forall y \cdot |y| = p(|x|))\ P(x,y)$ and

$x \notin L \rightarrow (\forall y \cdot |y| = p(|x|))\ \neg P(x,y)$

(iii) The class RP:

$x \in L \rightarrow (\exists^{+}y \cdot |y| = p(|x|))\ P(x,y)$ and

$x \notin L \rightarrow (\forall y \cdot |y| = p(|x|))\ \neg P(x,y)$

(iv) The class BPP:

$x \in L \rightarrow (\exists^{+}y \cdot |y| = p(|x|))\ P(x,y)$ and

$x \notin L \rightarrow (\exists^{+}y \cdot |y| = p(|x|))\ \neg P(x,y)$

(v) The class PP:

$x \in L \rightarrow (\text{Я}y \cdot |y| = p(|x|))\ P(x,y)$ and

$x \notin L \rightarrow (\text{Я}y \cdot |y| = p(|x|))\ \neg P(x,y)$

We suppose that all quantifiers are length bounded by a suitable polynomial $p(|x|)$ and suppress this bound in the notations, writing $\forall y\ P(x,y)$ instead of $(\forall y \cdot |y| = p(|x|))\ P(x,y)$ etc..

We abbreviate the definition above to follow simple notations as in [39], so that the criteria for an input x to be accepted as being in a language of the class are expressed as simple combinations of the known quantifiers.

(i') NP = $(\exists/\forall)$

(ii') P = $(\forall/\forall)$

(iii') RP = $(\exists^+/\forall)$

(iv') BPP = $(\exists^+/\exists^+)$

(v') PP = $(\acute{\exists}/\acute{\exists})$

Using quantifiers it can be verified that if the class C is represented by quantifiers as in the above notation: $C=(Q_1/Q_2)$, then the class co-C = (Q_2/Q_1). Therefore,

co-NP = $(\forall/\exists)$, co-P = $(\forall/\forall)$ = P, co-RP = $(\forall/\exists^+)$, co-BPP = $(\exists^+/\exists^+)$ = BPP and co-PP = $(\acute{\exists}/\acute{\exists})$ = PP.

The same idea can be extended for longer strings and predicates P(.,.,.) having appropriately many arguments, with the first place still referring to the input. Some basic examples are the classes in the Polynomial Hierarchy as we see in the following section.

Definition 12. For all quantifier strings Q_1, Q_2 of length n each over the alphabet consisting of $\{\exists,\forall,\exists^+\}$, (Q_1/Q_2) stands for the class of languages satisfying for all $x \in \sum^*$,

$x \in L \rightarrow Q_1 \underline{y}\ P(x,\underline{y})$ and

$x \notin L \rightarrow Q_2 \underline{y}\ \neg P(x,\underline{y})$

for some (n+1)-ary polynomial time predicate P, where $\underline{y}$ is a sequence of variables which has length n.

It is to be noted that not all pairs (Q_1/Q_2) make sense in the formula above; for example the pair $(\exists/\exists)$ contains all subsets of Σ^*. The two possibilities i.e., an input $x \in L$ or $x \notin L$ should be mutually exclusive.

Definition 13. A pair of quantifiers (Q_1/Q_2) is called *sensible* [17], if for any (n+1)-ary polynomial time predicate P and all $x \in \Sigma^*$, $Q_2\ y\ \neg\ P(x,\underline{y}) \rightarrow \neg\ Q_1\ y\ P(x,\underline{y})$.

3. PROBABILISTIC ALGORITHMS - AN EXPLANATION

In general, probabilistic algorithms, when given an input, may produce, a correct answer, a wrong answer or no answer at all. Conceptually one can distinguish between the following three types of probabilistic algorithms [38]:

1. Those that never lie but may give no answer with (say) Probability (no answer) < 0.001.
2. Those that always answer but may lie with (say) : Probability (wrong answer) < 0.001.
3. Those that always answer with : Probability (wrong answer) < 1/2

Algorithms of type (2.) and (3.) fall in the category of ***Monte-Carlo*** Algorithm, where as algorithm of type (1.) are called ***Las Vegas*** algorithms.

A polynomial (one sided) ***Monte Carlo*** TM for a language L is a Nondeterministic TM with length of the computations bounded by polynomial in the size of the input x, such that,

* If $x \in L$ then at least a fixed fraction more than half of the $2^{P(|x|)}$ computation of N on x halt with "Yes" or "Accept".

* If $x \notin L$ then all computations halt with a "No" or "Reject".

The class of all languages accepted by a polynomial (one sided) *Monte Carlo* TM is the class RP (called Randomized Polynomial time : definition 5.). We notice here that if a computation of the *Monte Carlo* TM above accepts an input x then the *decision is reliable and final.* There are no false positive answers, since for $x \notin L$, rejection is unanimous. This property is also shared by a Nondeterministic Turing Machine i.e., the class NP. On the other hand the probability of false negative is at most 1/2. From the complexity theory point of view, the *Monte Carlo* algorithms are efficient or at least feasible but may give incorrect answers. However, the probability of such incorrect answers can be greatly reduced by iterating the algorithm.

For the class RP, the probability of acceptance need not be (bounded away from) 1/2 but (bounded away from) any number strictly between zero and one, [38]. If we have a RP algorithm with a probability of false negative at most $1-\delta$: $\delta < 1/2$ then we could transform it into one with probability at most 1/2 by repeating the algorithm k times with k chosen as follows.

In the case of a TM repeating an algorithm means that from each leaf of the nondeterministic computation tree, we attach or hang on another tree identical to the original tree. We would do this k times for k repetitions. At the final leaf we would report a "Yes" if and only if at least one computation leading to the leaf has reported a "Yes". The probability of the false negative answer would now be $(1-\delta)^k$. *If we take* $k = [-1/\log(1-\delta)]$ *this probability is at most 1/2. The total time required is k times the original, which is still polynomial.*

In a similar manner a symmetric (two sided errors) *Monte Carlo* TM gives rise to the class BPP. By repeating a *Monte Carlo* algorithm (BPP algorithm) we can make the probability of false answers arbitrarily small, up to an inverse exponential.

More informally, in order to recognize a language in RP, there must be an algorithm for a NDTM with the property that on all inputs it either rejects "unanimously" or it accepts by majority. For a BPP algorithm, we require that either it accepts an input by a great majority or rejects an input by a great majority.

The *Monte Carlo* property is not exhibited by all polynomial bounded NDTMs. Given a machine there is no easy way to tell whether it qualifies by exhibiting the property: either it accepts an input by a great majority or rejects an input by a great majority. More over, there is no easy way to standardized NDTMs so that the *Monte Carlo* property is always exhibited and all *Monte-Carlo* algorithms are included i.e., there is no way to tell whether a machine always exhibits the required output behavior. **One of the shortcomings of these classes are that there are no complete problems for such classes.**

A ***Las Vegas*** algorithm may give no definite answer. Such an algorithm can be viewed as a result of combining two Monte Carlo algorithms; one with no false positive answers and one with no false negative answers.

The class (RP ∩ co-RP) is the class of languages with *Las Vegas* algorithms and is called ZPP (Zero-error Probabilistic Polynomial Time : definition 7). The *Las Vegas* algorithms are characterized by *feasibility, absolute reliability of*

correctness and a very low probability of nonconclusive termination.

Figure 2. depicts the relationship between the classes defined above The broken lines show classes which have no complete problems.

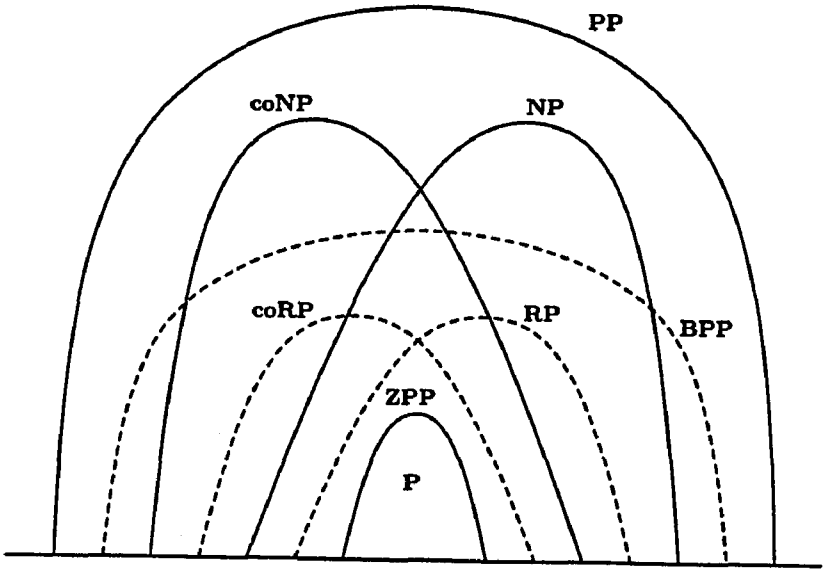

FIG 2. PROBABILISTIC CLASSES

4. THE POLYNOMIAL HIERARCHY

If *C* is any Deterministic or Nondeterministic time complexity class, we can define C^A to be the class of all languages decided (*or accepted*) by machines of the same sort and time bound as is *C*, except that the machines may now query an **oracle** A. An oracle is a **black box**, which answers all membership questions about the language A. The algorithm may repeatedly stop to query the oracle, obtain an answer from the oracle in one step and continue its computation. A hierarchy of classes is defined based on the concept of oracles, thereby yielding classes of **greater apparent difficulty**. While considering classes with languages decided in polynomial time, the number of queries to the oracle is bounded by a polynomial in the size of the input.

The Polynomial Hierarchy introduced in [29] as a computational analog to the Kleene arithmetic hierarchy of recursion theory consists of classes Δ_k^P, $\sum_k^P$, $\prod_k^P$, $k>0$, defined as follows:

Notations: $\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$

$\Delta_{k+1}^P = P^{\Sigma_k^P}$ (P with oracle Σ_k^P).

$\Sigma_{k+1}^P = NP^{\Sigma_k^P}$

$\Pi_{k+1}^P = \text{co-}NP^{\Sigma_k^P} = \text{co-}\Sigma_{k+1}^P$

In other words, Δ_{k+1}^P (Σ_{k+1}^P) is a set of languages that can be recognized in polynomial time (nondeterministic polynomial time) with an oracle in Σ_k^P. Π_{k+1}^P consists of all languages whose complement is in Σ_{k+1}^P. In particular, $\Delta_1^P = P$, $\Sigma_1^P = NP$ and Π_1^P = co-NP, while as stated earlier $\Delta_2^P = P^{NP}$. Note that $\Delta_k^P \subseteq \Sigma_k^P \cap \Pi_k^P$. Similarly $\Sigma_{k-1}^P \cup \Pi_{k-1}^P \subseteq \Delta_k^P$ and we do not know whether these containments are proper for any $k \geq 1$. Given these relationships, we can capture the set of all languages in the Polynomial Hierarchy (PH) as follows:

Definition 14.

The class PH is equal to the union :

$$\cup_{k=0}^{\infty} \Sigma_k^P.$$

Based on the number of queries made to the oracle, there are classes of the form $P^{NP[1]}$ (one query to the NP oracle), $P^{NP[2]}$,... etc.. It turns out that $NP^{NP} = NP^{NP[1]}$ (i.e., one query to the oracle is sufficient). The containment relationships within the polynomial hierarchy can be depicted as shown in figure 3.

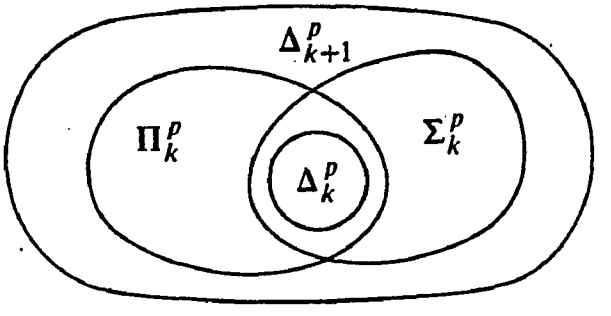

FIG 3. CONTAINMENT STRUCTURE WITHIN PH

The Polynomial Hierarchy can also be defined by alternating quantifiers instead of iterated use of oracles as above. Using alternating quantifiers for the classes in the Polynomial Hierarchy :

$\sum^{P}_{2} = (\exists\forall/\forall\exists)$, $\sum^{P}_{3} = (\exists\forall\exists/\forall\exists\forall)$ and so on.

The Polynomial Hierarchy can therefore be considered as built with quantifiers $\exists$ and $\forall$ [29,37] :

$$(\exists/\forall) \subseteq (\exists\forall/\forall\exists) \subseteq (\exists\forall\exists/\forall\exists\forall) \subseteq \ldots\ldots \subseteq \text{PSPACE}.$$

The following inclusions are known [12] :

$$P \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP, \text{ also } ZPP \subseteq \Delta P.$$

5. GAMES AND INTERACTIVE PROOFS

Another form of studying complexity problems is to describe their solution by *games* between two players. The terminology used in such games involves adversaries, e.g., games between two players of equal or unequal computational power such as "**Arthur-Merlin**" games [3]. **Arthur-Merlin** games are in terms of conversations between players of supposedly unequal computational power. **Arthur** (**A**) is an indifferent player with a polynomially bounded computational power and the ability to flip coins. **Merlin** (**M**) is a powerful player capable of optimizing the winning chances at every step. Given an input string x and a language L, **Merlin's** goal is to convince **Arthur** "beyond reasonable doubt" about membership of x in L. The two players alternate moves; the history of the game is *always known to both*. After $p(|x|)$

steps, a Deterministic Turing Machine reads the history and decides (in polynomial time) who wins. The class AM(1) consists of decision problems solvable by protocols in which only Arthur speaks and is equal to BPP [3]. MA(1) is the set of decision problems solvable by protocols which begin and end with Merlin's first transmission and this is equal to NP [3]. For each k, AM(k) (MA(k)) consists of all those decision problems solvable by Arthur-Merlin protocols in which Arthur (Merlin) goes first and there are exactly k messages sent. For an input $x \in L$ Merlin can convince Arthur with probability 3/4 of this fact and for an input $x \notin L$, Merlin can fool Arthur with probability no more than 1/4 (that $x \in L$), no matter what Merlin does.

Definition 15.

The class AM(k):

$L \in AM(k)$ iff there exist a k-move Arthur-Merlin game such that, Arthur moves first and for every $x \in \Sigma^*$,

$x \in L \rightarrow$ Pr (Merlin wins) $> 3/4$

$x \notin L \rightarrow$ Pr (Merlin loses) $< 1/4$.

The connection of alternating quantifiers to the **Arthur-Merlin** games has been presented in [39]. It is noted that

AM = $(\exists^+\exists/\exists^+\forall)$, MA = $(\exists\exists^+/\forall\exists^+)$, AMA = $(\exists^+\exists\exists^+/\exists^+\forall\exists^+)$.... etc.. It has been shown in [4] and [44] independently that MA $\subseteq$ AM and also that AM(k) = AM = MA(k+1) i.e., the game hierarchy collapses for all $k \geq 2$.

Let AM(poly) = MA(poly) = $\bigcup\{ AM(n^k) : k > 0 \}$ [3].

It is not expected that polynomially many alternations (in an AM game) collapse, i.e., if AM(poly) = AM. So we have,

MA $\subseteq$ AM $\subseteq$ AM(poly) $\subseteq$ PSPACE.

Similarly, a hierarchy of Interactive Proofs was defined in [14]. This generalized notion of an interactive proof is based on two players, one all powerful and the other with a source of random bits (or an equivalent ability to flip coins) and polynomial time computational power. The distinction with Arthur-Merlin games is that while Merlin (here called "the prover") operates under the same constraints, Arthur (here called "the verifier") always sends messages first and in addition is more clever. Arthur may send the result of an arbitrary polynomial computation based on input x and some random bits, which means that the verifier can keep "secrets" from the prover. In AM-games the verifier keeps no secrets.

Definition 16.

The class IP consists of all those decision problems solvable by interactive proof systems in which the total computation time of the verifier is polynomially bounded, but otherwise there is no limit on the amount of interaction.

By definition, AM(Poly) $\subseteq$ IP. Surprisingly all the extra power that the verifier has in the Interactive Proof System does not take us far from AM-games. We know from [15] that IP = AM(Poly) and IP(k) $\subseteq$ AM for $k \geq 1$.

Shamir has shown that IP = PSPACE [26A].

6. INCLUSION STRUCTURE

Figure 4. on the following page depicts the inclusion structure of some presently known important polynomial time complexity classes.

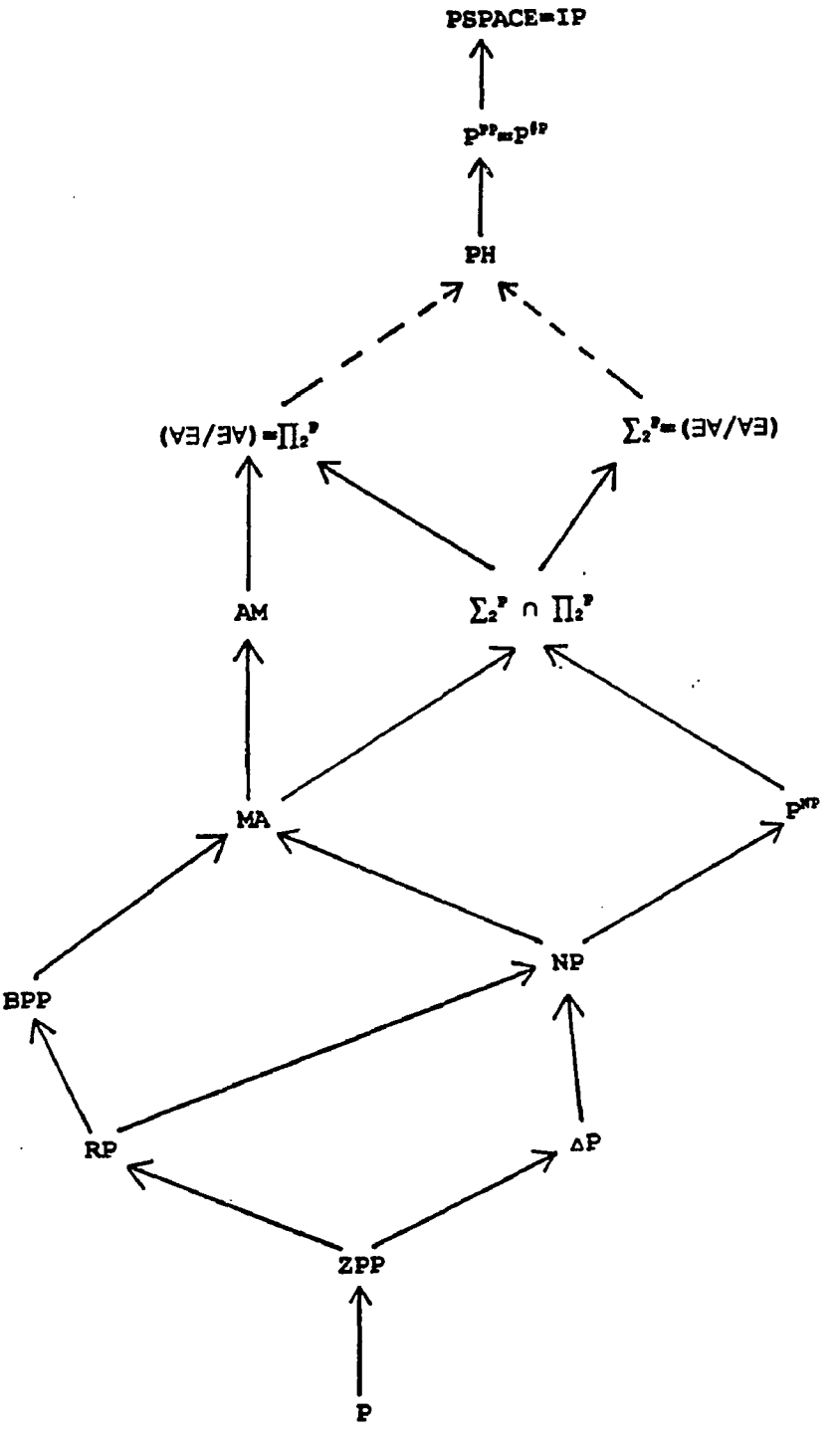

FIG 4. INCLUSION STRUCTURE OF SOME INTERESTING CLASSES

II. TREE MODELS OF POLYNOMIAL TIME COMPUTATIONS

1. TREE MODELS

Complexity classes are defined by referring to computational models and by imposing restrictions on the models. The most widely used computational models are DTM, NDTM, with or without oracles. The restrictions involve obvious limitations on time and space, as well as conditions on the number of accepting computations. In certain probabilistic models (e.g., for the class ZPP) the assumption for the computational model is that the algorithm may "stop" without an answer (i.e., an accept or a reject answer)- in a "don't know" state.

In this Chapter, we discuss different tree models of computations and their equivalence. Although these models encompass most of the classes of interest, we will review the counting class #P and the closely related probabilistic class PP in detail. It is not difficult to show that $P^{\#P}=P^{PP}$, i.e., a PP oracle is as good as a #P oracle. Trees as models

of computations for the counting class $\oplus P$ [24] and for probabilistic classes viz., BPP, RP, etc., have been used by Zachos ([38], [39], [43]). These models prove to be extremely useful while considering complexity classes in terms of quantifiers. Later, a similar model was presented in [8]. The tree models are useful in showing results which relate the PH to counting classes. It is known that all sets in PH (the Polynomial Hierarchy) are polynomial time (Cook) reducible to sets in PP [30]. In order to prove this Toda used the class $\oplus P$ as well as techniques that were used in [24] where $\oplus P$ was defined.

The computational model throughout the discussion is a polynomial-time bounded Nondeterministic Turing Machine (NDTM) *M* viewed as an ***"acceptor"***. We assume that the tape alphabet (without loss generality) is $\sum = \{0,1\}$. We restrict each computation path to a length of $p(|x|)$, p being a polynomial, and $|x|$, the length of input x. Each algorithm or TM can be equipped with a counter that counts the number of computation steps performed and if the counter exceeds $p(|x|)$ then M rejects the input x. The final answer is either accept or reject. The time for a computation is simply the number of steps made before the machine halts and is obviously bounded by a polynomial. Acceptance of a word

$x \in \Sigma^*$ is defined in terms of probabilities $Pr[Acc_M(x)]$ and $Pr[Rej_M(x)]$ such that a path may lead to acceptance or rejection.

On a particular input, computation paths of a Nondeterministic Turing Machine form a tree. In general, the branches of this tree (which represent the computational paths) may be of different lengths. In addition the fan-out at each node (the number of possible choices at each step), is not the same. Each path is of course bounded, i.e., each computation terminates in at most $p(|x|)$ steps, with the leaf nodes carrying the answer. It is important to point out that all the possible computations related to the tree are at most an exponential many.

The probabilistic model of computation is similar to the nondeterministic model and we consider the probability of obtaining an accepting computation as discussed in Chapter I. With each computation, we associate the probability of choosing a particular computation path. The probability of accepting an input is the sum of probabilities associated with the accepting computations of the machine on the given input. Acceptance of a word $x \in \Sigma^*$ by M is defined in terms of $Pr[Acc_M(x)]$ which is the probability that a path leads to

acceptance or $Pr[Rej_M(x)]$, the probability that a path leads to rejection of the input x.

Definition 1.

The class PP [12] *(Probabilistic Polynomial Time):*

A language $L \subseteq \Sigma^*$ is in PP iff for some NDTM, M and all $x \in \Sigma^*$,

$$x \in L \quad \text{--->} \quad Pr[Acc_M(x)] > \tfrac{1}{2} \quad \text{and}$$

$$x \notin L \quad \text{--->} \quad Pr[Rej_M(x)] > \tfrac{1}{2}$$

(Note: The restriction imposed on the NDTM accepting a language in PP, is that either more than half the computations are accepting computations or more than half the computations are rejecting computations. This restriction is however insignificant as e.g., the " > ½" of the last line can equivalently be replaced by " ≥ ½")

The class PP does not have the positive advantages (Robustness Property) associated with the "Las Vegas" class ZPP and the "Monte Carlo" classes BPP and RP. Polynomial repetition of a PP algorithm does not increase the probability of the correct answer.

The Counting Turing Machine and the class #P of counting problems that can be solved by such machines in polynomial time, were first introduced and studied in [34].

Definition 2.

The Counting Turing Machine:

The Counting Turing Machine is a NDTM whose "output" for a given input x, is the exact number of accepting computations for that input, i.e., the output of a Counting Machine is $|Acc_N(x)|$.

Definition 3.

The class #P:

The class #P is a set of all functions that are computable by polynomial time Counting Turing Machines.

The class #P consists of functions that *count exactly* the accepting path of some NP machine. $P^{\#P}$ is a class of languages computable by polynomial time Deterministic Turing Machines that use an oracle from #P. In this chapter we present simplified padding and extension techniques in order to obtain complete binary trees corresponding to PP computations, #P computations and computations corresponding

to sub-classes of the class #P [5] (viz., $\oplus$P and MOD_kP). Similar padding and extension techniques can be used to obtain complete binary trees corresponding to computational models of other complexity classes e.g., MA and AM.

2. TREES : VARIOUS STRUCTURES

Key Words

✧ *Fan out* : The number of nodes directly descending (children) from a parent node.

✧ *Binary tree* : A tree with maximum fan out equal to two.

✧ *Complete binary tree* : A binary tree, such that each node except the root node has one other sibling and all leaf nodes i.e., terminal nodes have the same depth.

✧ *Length of a path* : The length of the corresponding computation.

✧ *Degenerate node* : A node of a tree which has only one child.

While discussing trees corresponding to the computations of Nondeterministic, Probabilistic and Counting Machines, we keep in mind that only the leaf nodes are assigned values.

Therefore, a path rejects or accepts an input depending on the value of the corresponding terminal or leaf node.

Consider the following models of **Counting Turing Machines.** Let $p(|x|)$ denote a polynomial in size of the input $x \in \sum^*$: $p(|x|)$ is a bound on the height of the computation trees.

- C1 : The possible computation paths of a Counting Turing Machine M1 form a *complete binary tree* of height $p(|x|)$. Thus the number of all paths is $2^{p(|x|)}$. The result is a count of the number of accepting paths.

- C2 : The computational tree is a binary tree which is not necessarily complete.

- C3 : The computational tree is of *bounded fan-out* e.g., a ternary (fan-out=3) tree.

- C4 : The computational tree is of *polynomial fan-out.*

- C5 : The computational tree is of *exponential fan-out.*

Let $M =^P M'$ denote that M can be simulated by M' in

Let $M \equiv^P M'$ denote that M can be simulated by M' in polynomial time and vice versa. $C \equiv^P C'$, means that if there is a polynomial time bounded NDTM M of type C, then there exists NDTM M' of type C', that accepts exactly the same language accepted by M and vice versa.

3. TREE EXTENSION AND PADDING TECHNIQUES

PROPOSITION 1.

$$C1 \equiv^P C2 \equiv^P C3 \equiv^P C4 \equiv^P C5$$

Proof:

In each of the extension and padding steps, the idea is to preserve the exact number of accepting computations and we do not worry about the number of rejecting computations i.e., the leaf nodes carrying a reject answer.

The extension technique:

We can extend any binary tree into a *complete binary* tree with the same number of accepting paths in polynomial time.

Let the longest path of the binary tree be of length $p(|x|)$. Extend any path of length less than $p(|x|)$ further:

At every level the left child takes the value of the parent node, whereas the right child takes a rejecting (non-accepting) value. Extend all the way till each path length (i.e., a path from a root to a leaf) equals $p(|x|)$. This can surely be achieved in polynomial time.

Conversely, every complete binary tree (C1) is a binary tree (C2). Thus, C1 $=^{P}$ C2. This extension technique can be modified for a simulation of a tree with any fan-out.

The fan-out problem:

We need to prove that a counting machine that counts the number of accepting computations in a tree with arbitrary fan-out can be simulated by a complete binary tree. This can be achieved by the technique used

in [38] for the class RP. As an example consider a complete ternary tree. There are $3^{p(|x|)}$ computational paths. Let the least power of 2 exceeding $3^{p(|x|)}$ be "z". We now construct a binary tree of height "z" with the same outcome of the leaf nodes as the original tree.

As a result: $C1 =^P C2 =^P C3 =^P C4 =^P C5$.

□

PROPOSITION 2.

An analogous result can be shown for PP computational tree models M1, M2, M3, M4, M5 and the corresponding classes C1, C2, C3, C4, C5 defined analogously.

Proof:

Padding binary trees for PP algorithms :

Step 1. *Eliminate any degenerate nodes* in the following manner,

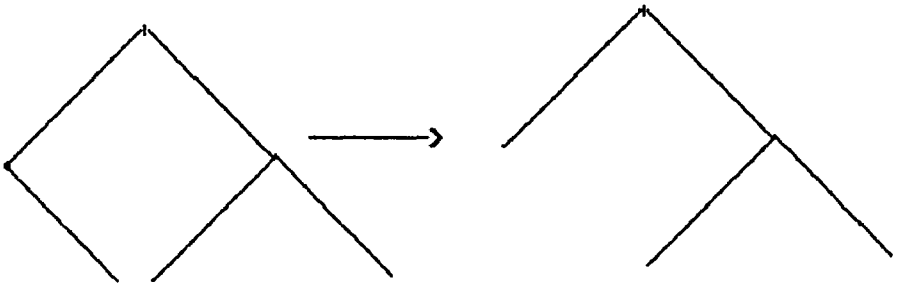

FIG. 5. ELIMINATION OF A DEGENERATE NODE

Step 2. *Duplicate each leaf node,* i.e., extend the tree to the next level in depth such that both the children which will be the new leaf nodes carry the value identical to the parent node.

Duplication in this manner will double the difference between the number of correct answers and the number of wrong answers for the PP algorithm.

Extension technique to form a complete binary tree:

After padding has been achieved, let the length of the longest path of the binary tree be $p(|x|)$. Extend any path of length less than $p(|x|)$ in the following manner:

At every level the left child takes the value of the parent node while the right child is assigned a rejecting value 50% of the time and an accepting value 50% of the time (in order to maintain the difference).

The fan-out problem:

Consider the fact that all nodes of the computation tree may not have the same fan-out. We use a technique in order to modify the existing tree such that all the nodes of the resulting tree have the same fan-out (i.e., each of the nodes except the root node has the same number of siblings).

As a result of the technique used here, all the nodes of the modified tree have a big fan-out. This does not create a problem, since the eventual, complete binary tree that we obtain after padding and extension is

still a PP computation tree simulating the original computation.

The first step is to create a clone for all the leaf nodes of the computation tree. This is in order to maintain a difference of at least two between the correct answers and the wrong answers. The duplication is achieved by adding a twin sibling for each leaf node.

Let us say that at this point the maximum fan-out of the tree is "m". Now consider the least power of two exceeding "m". Modify the tree so that all the nodes of the resulting tree have this fan-out. The new leaf nodes of the completed tree are assigned values such that, half of these take an accepting value and half of these take a rejecting value.

The resultant computation tree simulates the original computation. A node of fan-out 2^k can be easily simulated by a complete binary tree of height k.

Now a complete binary tree can be simulated in polynomial time from the resulting computational tree.

This can be achieved by the technique discussed in the previous proposition.

□

Since the padding and tree extension techniques discussed for the #P model, preserve the *exact* number of accepted computations, we can use the same technique to show similar results for the counting subclasses [5] of #P, viz., $\oplus$P, MOD_kP. These classes are also discussed in chapter III.

III. OPERATORS ON COMPLEXITY CLASSES

1. ROLE OF OPERATORS

We make observations about complexity classes when used as operators over other complexity classes. We identify properties of operators, when these are expressible in terms of ordinary quantifiers $\exists$ and $\forall$ together with $\exists^!$ ([38], [39]). Many times oracle classes can be expressed in terms of operators over complexity classes. Some oracle hierarchies collapse e.g., $BPP = BPP^{BPP} = \mathbf{BP}\cdot BPP$, but even such complexity classes yield several interesting classes when used as operators on other complexity classes e.g., $AM = \mathbf{BP}\cdot NP = \mathbf{co\text{-}R}\cdot NP$ (i.e., the Monte Carlo version of NP). The operator "**BP•**" has yielded very interesting results ([19], [26]). The class (RP ∩ co-RP) of languages with Las Vegas algorithms denoted by ZPP, offers realistic proposals for practical algorithms and in this sense is closely related to P. In order to study the Las Vegas versions of classes in the Polynomial Hierarchy, we introduce new operators corresponding to the Las Vegas class ZPP and the

class ΔP = NP $\cap$ co-NP. It is interesting that the only two, still remaining open problems from the original list of problems considered by Karp (whether NP-complete or in P) namely, **Primality** and **Graph Isomorphism** are in the Las Vegas version of P (ZPP) [1] and NP (**ZP**•*NP*) respectively, as shown later.

2. INTRODUCTION

The Monte-Carlo (Probabilistic) version of complexity classes (P, NP, PSPACE, $\sum^P_k$, $\prod^P_k$ etc.,) have been studied in depth [19], [26]. The class BPP has been extensively studied (see [15],[39],[44]) and it is known that the Monte-Carlo version of NP i.e., **BP**•*NP* is exactly the same class as AM(2). AM(2) was defined in [3], IP(2) i.e., two round interactive proof system was defined in [14] and in [15] it was shown that AM(2) = IP(2). In [19] several properties of general Probabilistic classes, with "**BP**•" as an operator have been identified. Our main interest in this chapter is to give a uniform picture of general operators and the properties that certain operators may satisfy. We also introduce the Las-Vegas versions (Zero-error probabilistic versions) of various complexity classes by defining the

operators corresponding to the class ZPP and the class ΔP. We present a model of computation for ΔP (the Las-Vegas computation ZPP follows the same model).

Several papers in the literature (e.g., [38], [39], [41]) have shown that many results can be proved by considering definitions of complexity classes using quantifiers. We use quantifiers to express operators over general complexity classes. We use the fixed length bounded quantifiers $(\exists y\ |y| = k)$, $(\forall y\ |y| = k)$ and $(\exists^{+} y\ |y| = k)$ (which is interpreted to mean : "for most strings such that $|y|=k$,..." and is also called the overwhelming majority quantifier defined in chapter I). We have used these quantifiers in the definition of complexity classes.

Our fundamental machine model is a polynomial-time bounded Nondeterministic Turing Machine M. Without loss of generality we assume the tape alphabet is $\Sigma=\{0,1\}$. For simplicity, we may view the set of possible computation paths of M on input x of length $|x|$ as a complete binary tree of length $p(|x|)$ for some polynomial p.

3. DEFINITIONS

Let C be a complexity class where we consider a polynomial time bound on the computational model. We define the following operators, keeping in mind that all quantifiers are length bounded by a suitable polynomial $p(|x|)$ i.e., polynomial in size of the input x.

Definition 1.

The operator "N• ", corresponding to the class NP :

Let $L \in \mathbf{N}\bullet C$; there is a $L_1 \in C$ such that

$$x \in L \text{ ----> } \exists\, y\ L_1(x,y)$$

and $x \notin L$ ----> $\forall\, y\ \neg L_1(x,y)$

The class NP can be expressed in terms of quantifiers as ($\exists/\forall$); therefore the operator "**N•**" can also be written as "($\exists/\forall$)•" (Remark: in [19], "$\exists$•" is used).

Similarly, the "**co-N•**" operator on the class C is defined as follows:

Let $L \in \mathbf{co\text{-}N}\bullet C$; there is $L_1 \in C$ such that

$$x \in L \text{ ----> } \forall\, y\ L_1(x,y)$$

and $x \notin L$ ----> $\exists\, y\ \neg\ L_1(x,y)$.

The operator "**co-N•**" can also be written as "(∀/∃)•". (Remark: in [19] "**∀•**" is used).

Notice that, $L' \in$ co-(**N•***C*) iff there is $L_1 \in C$ such that

$x \in L' \text{ ----> } \forall\, y \neg L_1(x,y)$

and $x \notin L' \text{ ----> } \exists\, y\ L_1(x,y)$.

Since $L_1 \in C \iff \neg L_1 \in co\text{-}C$, it can be seen that co-(**N•***C*) = **co-N•***co-C*.

Definition 2.

The operator "BP•" corresponding to the class BPP :

Let $L \in$ **BP•***C* ; there is $L_1 \in C$ such that

$x \in L \text{ ----> } \exists^+ y\ L_1(x,y)$

and $x \notin L \text{ ----> } \exists^+ y \neg L_1(x,y)$

Since BPP is closed under complement, the operator "**co-BP•**" is same as above and these can be written as "$(\exists^+/\exists^+)$•".

Notice that since $L_1 \in C \iff \neg L_1 \in co\text{-}C$, it can be verified that co-(**BP•***C*) = **BP•***co-C*.

Definition 3.

The operator "R•" corresponding to the class RP :

Let $L \in \mathbf{R}\bullet C$; there is $L_1 \in C$ such that

$$x \in L \text{ ----> } \exists^+ y \; L_1(x,y)$$

and $x \notin L$ ----> $\forall\, y \neg L_1(x,y)$

"R•" can also be written as **"$(\exists^+/\forall)$•"**. The operator **"co-R• "** can be defined similarly.

Here, again co-(**R**•C) = **co-R**•co-C.

Definition 4.

The operator "P•" corresponding to the class P :

Let $L \in \mathbf{P}\bullet C$; there is $L_1 \in C$ such that

$$x \in L \text{ ----> } \forall\, y \;\; L_1(x,y)$$

and $x \notin L$ ----> $\forall\, y \neg L_1(x,y)$

From here it is clear that **P**•$C = C$.

The class P is closed under complement and thus

co-(**P**•C) = **P**•co-C = co-C.

Similarly we could define other operators (not directly involving $\forall, \exists, \exists^+$) .

Definition 5.

The operator "⊕•" corresponding to the class ⊕P ([24], see also [13]) :

Let $L \in \oplus\bullet C$; there is a $L_1 \in C$ such that

$x \in L$ ----> $\oplus\, y\ L_1(x,y) = \|y: L_1(x,y)\|$ is odd &

$x \notin L$ ----> $\neg\oplus\, y\ L_1(x,y) = \|y: L_1(x,y)\|$ is even.

The operator "**co-⊕•**" is defined as follows:

$L \in$ **co-⊕•**C ; there is a $L_1 \in C$ such that

$x \in L$ ----> $\neg\oplus\, y\ L_1(x,y) = \|y: L_1(x,y)\|$ is even &

$x \notin L$ ----> $\oplus\, y\ L_1(x,y) = \|y: L_1(x,y)\|$ is odd.

The following fact is easy to prove considering that we have an even number of total paths, because we consider complete binary trees :

$\oplus\bullet C = \oplus\bullet co\text{-}C$.

We can similarly define any operator "**O•**" corresponding to a complexity class **OP** which is expressible as a sequence of quantifiers (Q/Q'). We use two notations i.e., **(Q/Q')•***P* is the same as (Q/Q'). For example, the operator "**N•co-N•**" (or "(∃∀/∀∃)•") corresponding to the class, $\sum^P_2$ i.e., (∃∀/∀∃) or **N•***co-NP*. The operator corresponding to the class MA (Merlin-Arthur), can be written as "**MA•**" or "(∃∀/∀∃⁺)•"

(Remark: It has been shown [41] as the completeness property, that $(\exists\forall/\forall\exists^{+}) = (\exists\exists^{+}/\forall\exists^{+})$).

It is to be noted here that MA ($= (\exists\forall/\forall\exists^{+})$) is not the same as **N•***co-RP* in terms of quantifiers. The class MA is expressed as follows:

Let $L \in$ MA then, there exists a polynomial time predicate $P(x,y,z)$ such that:

$x \in L \text{ ----> } \exists y \forall z\ P(x,y,z)$

$x \notin L \text{ ----> } \forall y \exists^{+} z\ \neg P(x,y,z)$

We express the class **N•***co-RP* as follows:

Let $L \in$ **N•***co-RP*, then there is a $L_1 \in$ co-RP such that,

$x \in L \text{ ----> } \exists\ y\ L_1(x,y)$. (a)

$x \notin L \text{ ----> } \forall\ y\ \neg L_1(x,y)$ (b)

Since $L_1 \in$ co-RP, the above conditions (a) and (b) are in fact equivalent to the following conditions in terms of quantifiers alone:

There exists a polynomial time predicate $P(x,y,z)$ such that,

$$x \in L \text{ ----> } \exists y \forall z\ P(x,y,z) \qquad (a')$$

$$x \notin L \text{ ----> } \forall y \exists^{+} z\ \neg P(x,y,z) \qquad (b')$$

$$\forall x \forall y (\forall z P(x,y,z) \vee \exists^{+} z \neg P(x,y,z)) \qquad (*)$$

The class MA is equivalent to the conditions (a') and (b') in terms of quantifiers as seen above. The (*) condition is not required for the class MA. Thus **N•co-RP** $\subseteq$ MA.

LEMMA 1. Let **O•P** and **O'•P** be complexity classes expressible in terms of quantifiers as (Q_1/Q_2) and (Q'_1/Q'_2). **O•O'•P** $\subseteq$ $(Q_1Q'_1/Q_2Q'_2)$ but it is in general, ***not true*** that, **O•O'•P** = $(Q_1Q'_1/Q_2Q'_2)$.

□

This result especially holds true when the second operator i.e., "**O'•** " corresponds to a probabilistic class, in which case we need an added condition i.e., the (*) condition to express the special behavior of the machine model (see chapter I). The (*) condition in fact imposes an added restriction on the machine model e.g., there must be an algorithm for a NDTM with the property that on all inputs it either accepts "unanimously" or it rejects by a great

majority. We shall therefore call it the ***confining condition***. The ***confining condition*** depends on the class *C, in the definition of O•C.*

For the class **N•*BPP***, let L ϵ **N•*BPP*** then there is a $L_1 \epsilon$ BPP such that,

$$x \epsilon L \text{ ----> } \exists y\ L(x,y)$$

$$x \notin L \text{ ----> } \forall y\ \neg L(x,y).$$

LEMMA 2. (i) **N•co-*RP*** $\subseteq$ **N•*BPP*** $\subseteq$ **MA**

(ii) **co-N•*RP*** $\subseteq$ **co-N•*BPP*** $\subseteq$ **co-MA**

Proof : (i) Let L ϵ **N•*co-RP*** then there exists a polynomial time predicate P(x,y,z) such that:

$$x \epsilon L \text{ ----> } \exists y \forall z\ P(x,y,z) \qquad (a')$$

$$x \notin L \text{ ----> } \forall y \exists^{+} z \neg P(x,y,z) \qquad (b')$$

$$\forall x \forall y (\forall z P(x,y,z) \vee \exists^{+} z \neg P(x,y,z)) \qquad (*)$$

The conditions (a') and (b') can be rewritten as,

$$x \epsilon L \text{ ----> } \exists y \exists^{+} z\ P(x,y,z) \qquad (a')$$

$$x \notin L \text{ ----> } \forall y \exists^{+} z\ \neg P(x,y,z) \qquad (b')$$

The ***confining condition*** (*) based on the class *co-RP* above is stiffer than the following ***confining condition*** that is based on the class *BPP*:

$$\forall x \forall y (\exists^{+} z P(x,y,z) \vee \exists^{+} z \neg P(x,y,z)) \qquad (*')$$

The class **N•***BPP* is expressible by (a'),(b') together with (*'). Therefore L ϵ **N•***BPP*. The class MA is expressible in terms of quantifiers as conditions (a') and (b') alone. No ***confining condition*** i.e., (*) condition is required in order to express the class MA in terms of quantifiers as seen above. Therefore L ϵ MA.

The proof for (ii) proceeds similarly.

□

In [9] it has been shown that there is an oracle c relative to which MA $\not\subseteq$ NP^{BPP}. Thus **N•***BPP* is a proper subclass of MA relative to the oracle c.

On the other hand, the class AM is the same as **co-R•***NP*, since there is **no *confining condition*** corresponding to the class **co-R•***NP*. The operator corresponding to the class AM

(Arthur- Merlin), i.e., **co-R•*NP*** can be written as "**AM•**" or "$(\forall\exists/\exists^{+}\forall)$•" which is equivalent to "$(\exists^{+}\exists/\exists^{+}\forall)$•".

In the next section, we consider complexity classes involving polynomial time, expressible in terms of quantifiers and those that are expressible as intersections of these classes.

4. GENERAL PROPERTIES OF OPERATORS

In this section we first list the properties that may be satisfied by operators corresponding to complexity classes and involving polynomial time. We then discuss various operators and their properties. We argue over formulae using combinatorial techniques.

We call an operator represented in terms of quantifiers as "**(R/R')•**" *symmetric* if (R/R') = (R'/R) [viz., "**P•** " and "**BP•**" from the definitions in the previous section].
Remark: This does not necessarily imply R = R' e.g., "$(\forall\exists^{+}/\exists^{+}\forall)$•" is *symmetric*, [41].

* *Property* 1 : co-(O•C) = co-O•co-C, ∀ C

(The Co-Property).

* *Property* 2 : $C_1 \subseteq C_2 \Rightarrow O \bullet C_1 \subseteq O \bullet C_2$, ∀ C_1 & ∀ C_2

(Monotonicity).

* *Property* 3 : $C \subseteq O \bullet C$, ∀ C

(Vacuous extension).

* *Property* 4 : $O \bullet (C_1 \cap C_2) \subseteq O \bullet C_1 \cap O \bullet C_2$, ∀ C_1 & ∀ C_2

(Distributive Property).

* *Property* 5 : $O \bullet (C \cap \text{co-}C) \subseteq O \bullet C \cap O \bullet \text{co-}C$, ∀ C

(Special case of Property 4.).

* *Property* 6 : $O_1 \bullet P \subseteq O_2 \bullet P \Rightarrow O_1 \bullet C \subseteq O_2 \bullet C$, ∀ C

(Monotonicity w.r.t. Operators).

* *Property* 7 : $P \subseteq C \Rightarrow O \bullet P \subseteq O \bullet C$, ∀ C

(Special case of Property 2.).

* *Property* 8 : O•O•C = O•C, ∀ C

(Idempotent)

THEOREM 3.

Let "O•", "O'•" be any operators corresponding to complexity classes OP, O'P respectively, expressible in terms of

quantifiers as (R_1/R_2) and (R'_1/R'_2). Let C, C_1 and C_2 be complexity classes.

1. $co\text{-}(O\bullet C) = co\text{-}O\bullet co\text{-}C.$ {1.1}
2. $C_1 \subseteq C_2 \Rightarrow O\bullet C_1 \subseteq O\bullet C_2.$ {1.2}
3. $C \subseteq O\bullet C.$ {1.3}
4. $O\bullet(C_1 \cap C_2) \subseteq O\bullet C_1 \cap O\bullet C_2$ {1.4}
5. $O\bullet(C \cap co\text{-}C) \subseteq O\bullet C \cap O\bullet co\text{-}C$ {1.5}

Proof :

1. $co\text{-}(O\bullet C) = co\text{-}O\bullet co\text{-}C.$

   Since "$O\bullet$" is expressible in terms of quantifiers as "$(R_1/R_2)\bullet$", the operator "$co\text{-}O\bullet$" is expressible as "$(R_2/R_1)\bullet$",

   $L \in co\text{-}O\bullet co\text{-}C \Leftrightarrow$ there is a $L_1 \in co\text{-}C$:

   $$x \in L \rightarrow R_2\ y\ L_1(x,y)$$
   $$\text{and}\quad x \notin L \rightarrow R_1\ y\ \neg L_1(x,y).$$

   Let $L_2 = \neg L_1$, then $L_2 \in C$.

   So, $L \in co\text{-}O\bullet co\text{-}C \Leftrightarrow \exists L_2 \in C$:

$x \notin \overline{L} \rightarrow R_2\ y\ \neg L_2(x,y)$

and $x \in \overline{L} \rightarrow R_1\ y\ L_2(x,y)$.

$\Leftrightarrow \overline{L} \in \mathbf{O}\bullet C$

$\Leftrightarrow L \in$ co-$(\mathbf{O}\bullet C)$, where $\overline{L}$ is the complement of L.

2. $C_1 \subseteq C_2 \Rightarrow \mathbf{O}\bullet C_1 \subseteq \mathbf{O}\bullet C_2$.

"$\mathbf{O}\bullet$" is an operator expressible as "$(\mathbf{R_1}/\mathbf{R_2})\bullet$", so in order to express any language $L \in \mathbf{O}\bullet C_1$, we have :

There exists $L_1 \in C_1$ such that

$x \in L \rightarrow R_1\ y\ L_1(x,y)$

and $x \notin L \rightarrow R_2\ y\ \neg L_1(x,y)$.

Since $C_1 \subseteq C_2$, it follows that $L_1 \in C_2$.

From the above expression we observe that $L \in \mathbf{O}\bullet C_2$ therefore, $\mathbf{O}\bullet C_1 \subseteq \mathbf{O}\bullet C_2$.

3. $C \subseteq \mathbf{O}\bullet C$.

"$\mathbf{O}\bullet$" is an operator expressible as "$(\mathbf{R_1}/\mathbf{R_2})\bullet$".

If $L \in C$ then,

$x \in L \Rightarrow R_1\ y\ L(x)$ and

$x \notin L \Rightarrow R_2\ y\ \neg L(x)$ by vacuous quantification. Thus $L \in O{\bullet}C$.

4. $O{\bullet}(C_1 \cap C_2) \subseteq O{\bullet}C_1 \cap O{\bullet}C_2$.

Since $(C_1 \cap C_2) \subseteq C_1$ & $(C_1 \cap C_2) \subseteq C_2$, using Property 2. we directly obtain:

$O{\bullet}(C_1 \cap C_2) \subseteq O{\bullet}C_1 \cap O{\bullet}C_2$.

5. $O{\bullet}(C \cap co\text{-}C) \subseteq O{\bullet}C \cap O{\bullet}co\text{-}C$.

Proof is a consequence of Property 4.

■

COROLLARY 4. $O \bullet NP \subseteq O \bullet P^{NP[1]}$.

Proof :

The proof follows from property 2 above, since $NP \subseteq P^{NP[1]}$.

■

We shall call an operator "O•" ***sensible*** if it satisfies properties 1, 2 and thus 4 and 5. Based on this notion of *sensible operators* we define two new operators in the next section.

Although it is obvious that $P \bullet NP = NP$ and $P \subseteq ZPP$, we will show later that it is highly unlikely that, $NP \subseteq ZP \bullet NP$, (if so then the polynomial hierarchy collapses at the second level).

This leads us to believe that properties 3. and 6., probably do not hold for all operators. It is probably not even true, that for any

"$O_1\bullet$ ", "$O_2\bullet$" and "$O\bullet$" if,

$O_1 \bullet P \subseteq O_2 \bullet P$ then $O_1 \bullet O \bullet P \subseteq O_2 \bullet O \bullet P$.

5. THE OPERATOR "$\Delta\bullet$" AND THE LAS-VEGAS OPERATOR "$ZP\bullet$"

The Arthur-Merlin system has been defined in [3]. Arthur is a randomized verifier & Merlin is a nondeterministic prover. Merlin tries to convince Arthur that a certain string belongs to a language L. AM(k) [MA(k)] denotes a k move Arthur-Merlin game where Arthur [Merlin] moves first. It is known [3,4] that,

$$AM(k) = AM = MA(k+1).$$

In terms of operators and quantifiers, (see [17], [19])

$AM = (\forall\exists/\exists^{+}\forall) =$ **co-R•***NP* $= (\exists^{+}\exists\ /\exists^{+}\forall) =$ **BP•***NP*

co-AM = **R•***co-NP* $= (\exists^{+}\forall/\exists^{+}\exists) =$ co-(**BP•***NP*) = **BP•***co-NP*

$MA = (\exists\exists^{+}/\forall\exists^{+}) = (\exists\forall/\forall\exists^{+})$.

$co\text{-}MA = (\forall\exists^{+}/\exists\exists^{+}) = (\forall\exists^{+}/\exists\forall)$.

The following results are known,

NP $\subseteq$ MA $\subseteq$ AM and co-NP $\subseteq$ co-MA $\subseteq$ co-AM

BPP $\subseteq$ MA $\subseteq$ AM also BPP $\subseteq$ co-MA $\subseteq$ co-AM

However, it has also been shown that if co-NP $\subseteq$ AM, then the PH collapses to the second level (see [7]). In this section we define the operators "**ZP•**" (corresponding to the class ZPP) and "Δ•" (corresponding to the class NP $\cap$ co-NP). We will first elaborate upon the model of computation for the class ΔP. The model for the Las Vegas class ZPP would be similar. A problem which has a Las Vegas algorithm can be thought of, as a combination of two algorithms, one that has no false positive answers and another that has no false negative answers.

MODEL OF COMPUTATION FOR ΔP: We consider binary trees as models of computation. Each path from the root to a leaf is a possible computation on input $x \in \Sigma^{*}$ by a NDTM.

(a) Let M_1 be a NDTM accepting $L_1 \in NP$. For $x \in L_1$, the computation tree is such that some computation paths result in a "YES" leaf i.e., an accepting computation; while the other paths may result in a nondecisive answer or a "?" leaf. In case $x \notin L_1$ then all computation paths result in "?".

(b) Now consider a NDTM M_2, accepting $L_2 \in$ co-NP. Let $x \in L_2$. In this case all computations result in a "?" leaf. For $x \in L_2$ there is at least one computation that ends in a leaf "NO" while others result in a "?" leaf.

(c) By definition $\Delta P = NP \cap$ co-NP. Let M_3 be a NDTM, accepting $x \in L_3$: $L_3 \in \Delta P$. Then the computation tree, in the model considered here is such that, there is at least one computation path which results in the correct answer and many others may result in a "?" leaf i.e., giving no answers at all; the correct answer being either "NO" or "YES."

Using the above model of computation, we show the following result.

THEOREM 5.

$$\Delta P^c = NP^c \cap \text{co-}NP^c, \text{ for all oracles c.}$$

Proof :

In order to show $\Delta P^c = NP^c \cap \text{co-}NP^c$ consider a language $L \in NP^c$, accepted by NDTM M_1 querying an oracle "c" . The tree of all computation paths is similar to one defined in (a) above, except that at each computation path an oracle query is permitted. Similarly, if $L \in \text{co-}NP^c$ then a NDTM M_2 querying an oracle "c" accepts L, [M_2 as defined in (b)] also if $L \in \Delta P^c$ then the NDTM M_3 with an oracle "c" accepts L.

We first show $L \in \Delta P^c \Longrightarrow L \in NP^c \cap \text{co-}NP^c$, i.e., given a NDTM, described as M_3 above that accepts L in polynomial time, we can construct NDTMs M_1 and M_2, such that both M_1 and M_2 accept L in polynomial time iff M_3 accepts L.

Construction of NP^c model: The NP^c tree is the same as the ΔP^c tree except that all the "NO" leaves are converted to "?".

Construction of co-NP^c model: The co-NP^c tree is the same as the ΔP^c tree except that all the "YES" leaves are converted to "?".

For the **converse,** join the NP^c and the co-NP^c trees to form a ΔP^c tree, by adding a new root as shown in figure 6.

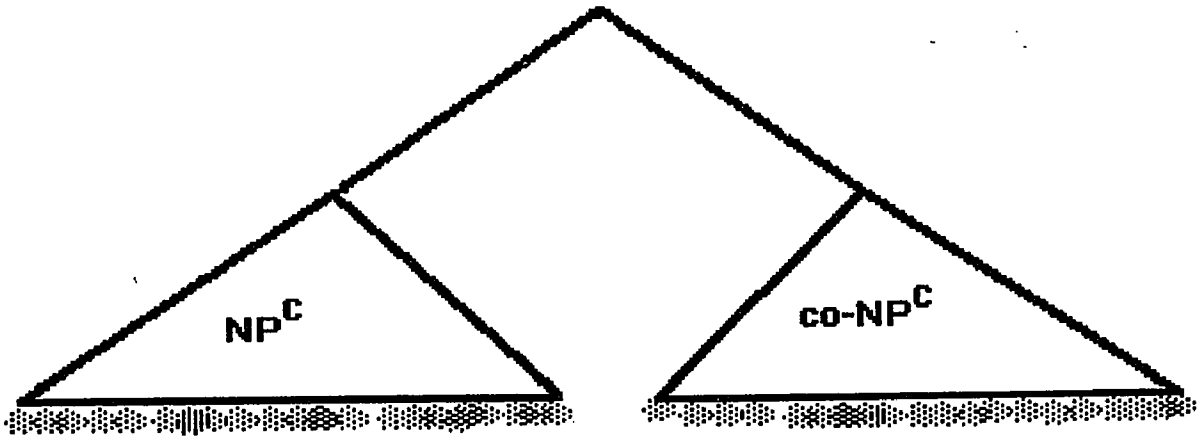

FIG 6. ΔP^c COMPUTATION

Here is a general fact which will be useful in what follows:
co-(C1 ∩ C2) = co-C1 ∩ co-C2 ∀ classes C1 and C2.

In case of operators corresponding to complexity classes, defined as intersection classes, viz., ΔP and ZPP, the definitions are formulated as follows:

Definition 5.

The operator "$\Delta\bullet$" corresponding to the class ΔP (= NP ∩ co-NP):

$$\Delta\bullet C = \mathbf{N}\bullet co\text{-}C \cap \mathbf{co\text{-}N}\bullet C.$$

The operator "**co-$\Delta\bullet$**" is defined as follows :

$$\mathbf{co\text{-}}\Delta\bullet C = \mathbf{N}\bullet C \cap \mathbf{co\text{-}N}\bullet co\text{-}C = \Delta\bullet co\text{-}C$$

Note that the operator "**co-$\Delta\bullet$**" is different from "$\Delta\bullet$".

Definition 6.

The Las Vegas operator "ZP•" corresponding to the class ZPP (= RP ∩ co-RP):

$$\mathbf{ZP}\bullet C = \mathbf{R}\bullet co\text{-}C \cap \mathbf{co\text{-}R}\bullet C$$

The operator "**co•ZP•**" is defined as follows :

$$\mathbf{co\text{-}ZP}\bullet C = \mathbf{R}\bullet C \cap \mathbf{co\text{-}R}\bullet C = \mathbf{ZP}\bullet co\text{-}C$$

Note that the operator "**co-ZP•**" is different from "**ZP•**".

Operators "**ZP•**" and "**Δ•**" are *sensible* since these operators satisfy properties 1 and 2 from the last section (see Theorem 6. below).

Notice that "**ZP•**" is not symmetric in the sense that **ZP•***C* is not the same as **ZP•***co-C*. On the other hand **ZP•***C* is closed under complement (see Lemma A below). The operator "**ZP•**" provides us with a medium to study the "Las Vegas" versions of various complexity classes.

LEMMA A

The classes **Δ•***C* and **ZP•***C* are closed under complement for any class *C*.

Proof :

co-(Δ•*C*) = co-(**N•***co-C* $\cap$ **co-N•**C) by definition

= co-(**N•**co-*C*) $\cap$ co-(**co-N•***C*) by the above fact

= **co-N•***C* $\cap$ **N•***co-C* by the Co-Property 1.

= **Δ•***C* by definition.

Similarly co-(**ZP•***C*) = **ZP•***C*.

□

THEOREM 6.

The operators "**ZP•**" and "$_{\Delta}$•" are *sensible*.

Proof :

We first show that both operators "**ZP•**" and "$_{\Delta}$•" satisfy property 1.

$$\textbf{co-ZP•co-}C = \textbf{ZP•}C \text{ (by definition)}$$

$$= \text{co-}(\textbf{ZP•}C) \text{ (by Lemma A)}$$

Similarly for the operator "$_{\Delta}$•".

We now show that "**ZP•**" satisfies property 2.

Let $C_1 \subseteq C_2$, it follows that co-$C_1 \subseteq$ co-C_2.

$$\textbf{ZP•}C_1 = \textbf{R•}co\text{-}C_1 \cap \textbf{co-R•}C_1$$

$$\textbf{ZP•}C_2 = \textbf{R•}co\text{-}C_2 \cap \textbf{co-R•}C_2$$

From Theorem 1, part 2, it follows,

$\textbf{R•}co\text{-}C_1 \subseteq \textbf{R•}co\text{-}C_2$, and $\textbf{co-R•}C_1 \subseteq \textbf{co-R•}C_2$.

Therefore, $\textbf{ZP•}C_1 \subseteq \textbf{ZP•}C_2$.

The same argument follows for the operator "$_{\Delta}$•".

■

The operator "**BP•**" enables us to investigate classes which have "Monte Carlo" algorithms (Monte Carlo algorithms have the property of solving a problem with a low probability of error). The following observations follow :

PROPOSITION B

$$\mathbf{BP}\bullet\Delta \subseteq \mathbf{ZP}\bullet NP$$

Proof :

If $L \in \mathbf{BP}\bullet\Delta$ then, there is a L_1 in ΔP such that:

$$x \in L \text{ ---> } \exists^{+}y \;\; L_1(x,y)$$

$$x \notin L \text{ ---> } \exists^{+}y \; \neg L_1(x,y).$$

Now $\mathbf{BP}\bullet\Delta = \mathbf{BP}\bullet(NP \cap co\text{-}NP)$

$$\subseteq \mathbf{BP}\bullet NP \cap \mathbf{BP}\bullet co\text{-}NP \text{ (using \{1.4\})}$$

$$= (\mathbf{co\text{-}R}\bullet NP \cap \mathbf{R}\bullet co\text{-}NP) = AM \cap co\text{-}AM$$

i.e., $\mathbf{BP}\bullet\Delta \subseteq AM \cap co\text{-}AM$.

Also note that $\mathbf{ZP}\bullet NP = \mathbf{R}\bullet co\text{-}NP \cap \mathbf{co\text{-}R}\bullet NP$

$$= co\text{-}AM \cap AM.$$

Therefore, $\mathbf{BP}\bullet\Delta \subseteq \mathbf{ZP}\bullet NP$.

□

Because of Lemma A note that, $\mathbf{ZP}\bullet NP$ is closed under complement, i.e., $\mathbf{ZP}\bullet NP = \mathbf{co\text{-}ZP}\bullet co\text{-}NP = co\text{-}(\mathbf{ZP}\bullet NP)$

It was shown in [7] that if co-NP $\subseteq$ AM, the Polynomial Hierarchy collapses to the second level. Babai introduced the term "Las vegas" in an unpublished manuscript which has been referenced in [18]. The term was coined in conjunction with his research in randomized algorithms for variations of the Graph Isomorphism (GI) problem (i.e., determining whether two graphs are isomorphic). Babai then considered a variant of a simpler GI problem in which the vertices of the two graphs are colored and the problem involves looking for a color preserving isomorphism. Shortly after that the colored GI problem was shown to be in P [10]. It is known from [7] that if the GI problem is NP-complete then the Polynomial Hierarchy collapses to $\sum^P_2 = \prod_2^P$ = AM. This means that the GI problem is not likely to be NP complete (because we believe that the Polynomial Hierarchy does not collapse). We know that the GI problem is in AM $\cap$ co-AM (see [15] [25]). While characterizing the class **ZP•NP**, we have noted above that it is exactly the class AM $\cap$ co-AM (closed under complementation). It follows that if we can show for any NP-complete problem to be in **ZP•*NP*** the Polynomial Hierarchy collapses to the second level. Equivalently, if NP $\subseteq$ **ZP•*NP*** or co-NP $\subseteq$ co-(**ZP•*NP***) then the Polynomial Hierarchy collapses at the second level. Since the GI problem lies in

the class AM ∩ co-AM which is **ZP•*NP***, so does the complement of GI (because the class **ZP•*NP*** is closed under complementation). This provides a new insight into this interesting problem i.e., the Graph Isomorphism problem lies in the "Las-Vegas" version of NP.

Thus, although P ⊆ ZPP it is unlikely that **N•P ⊆ ZP•N•P**. As a matter of fact it seems unlikely that the following holds for every "**O_1•**", "**O_2•**" & "**O•**":

If **O_1•P ⊆ O_2•*P*** then **O_1•O•*P* ⊆ O_2•O•*P***.
(This is a special case of property 6.)

It seems unlikely that the operator "**ZP•**" when applied to the class NP yields a complexity class higher than NP.

In [19], GI has been shown as a low set with respect to various operators. The idea of lowness stems from the fact that the set does not help in the computation i.e., if *C* is low for the operator "**O•**" then **O•*C*** = **O•*P*** (looking at oracles, a set c is low for the class OP if OP^c = OP).

To investigate the "Las-Vegas" versions of some other complexity classes, we apply the operators "**ZP•**" and "**co-ZP•**" on an other complexity class :

★ **ZP•***co-NP* = **R•***NP* ∩ **co-R•***co-NP*.

In terms of quantifiers,

$$\mathbf{R}\bullet NP = (\exists+\exists/\forall\forall) = (\exists/\forall) = \text{NP}$$

also **co-R•***co-NP* = $(\forall\forall/\exists+\exists)$ = $(\forall/\exists)$ = co-NP.

therefore **ZP•***co-NP* = NP ∩ co-NP = ΔP.

★ On the other hand co-(**ZP•**co-NP) = **co-ZP•***NP*

= **ZP•**co-*NP* = ΔP.

★ **ZP•***RP* = **co-R•***RP* ∩ **R•***co-RP*

$\subseteq$ BPP ∩ BPP

= BPP.

We do not know if this containment is proper.

★ **ZP•***co-RP* = co-RP ∩ RP = ZPP = co-(**ZP•***co-RP*) = **co-ZP•***RP*

and co-(**ZP•***co-RP*) = **co-ZP•***RP* = **ZP•***co-RP* = ZPP.

PROPOSITION 7.

$$\mathbf{ZP}\bullet\Delta P = \Delta\mathrm{P}$$

Proof:

$$\mathbf{ZP}\bullet\Delta P = \mathbf{co\text{-}R}\bullet\Delta P \cap \mathbf{R}\bullet co\text{-}\Delta P = \mathbf{co\text{-}R}\bullet\Delta P \cap \mathbf{R}\bullet\Delta P$$

$$= \mathbf{co\text{-}R}\bullet(NP \cap co\text{-}NP) \cap \mathbf{R}\bullet(NP \cap co\text{-}NP)$$

$$\subseteq \mathbf{co\text{-}R}\bullet NP \cap \mathbf{co\text{-}R}\bullet co\text{-}NP \cap \mathbf{R}\bullet NP \cap \mathbf{R}\bullet co\text{-}NP$$

$$= \mathrm{AM} \cap \mathrm{co\text{-}NP} \cap \mathrm{NP} \cap \mathrm{co\text{-}AM}$$

$$= \mathrm{co\text{-}AM} \cap \mathrm{AM} \cap \Delta\mathrm{P} = \Delta\mathrm{P}$$

□

On the other hand $\Delta\mathrm{P} \subseteq \mathbf{ZP}\bullet\Delta P$, because

$\Delta\mathrm{P} \subseteq \mathbf{R}\bullet\Delta P$ and $\Delta\mathrm{P} = \mathbf{co\text{-}R}\bullet\Delta\mathrm{P}$ (by {1.3}).

Therefore also $\mathrm{co\text{-}}(\mathbf{ZP}\bullet\Delta P) = \mathbf{co\text{-}ZP}\bullet co\text{-}\Delta = \Delta\mathrm{P}$

It is easily seen that , $\mathbf{ZP}\bullet ZPP = \mathrm{ZPP} = \mathbf{ZP}\bullet P$ i.e., the operator "**ZP•**" is idempotent when used on P.

THEOREM 8.

$$\sum^{\mathrm{P}}_{2}\bullet(\mathbf{ZP}\bullet NP) = \sum^{\mathrm{P}}_{2}$$

Proof :

$$\sum^{\mathrm{P}}_{2}\bullet(\mathbf{ZP}\bullet NP) = \sum^{\mathrm{P}}_{2}\bullet(\mathbf{co\text{-}R}\bullet NP \cap \mathbf{R}\bullet co\text{-}NP)$$

$$= \sum^{\mathrm{P}}_{2}\bullet(\mathbf{BP}\bullet NP \cap \mathbf{BP}\bullet co\text{-}NP)$$

(using the distributive property {1.4})

$$\subseteq \Sigma^P_2 \cap \Sigma^P_2 \bullet \mathbf{BP} \bullet \Pi^P_1 \qquad \{\text{See [19]}\}$$

$$\subseteq \Sigma^P_2 \cap \Sigma^P_3 = \Sigma^P_2$$

$\Sigma^P_2 \subseteq \Sigma^P_2 \bullet (\mathbf{ZP} \bullet NP)$ follows from Property 7.

The class $\mathbf{ZP} \bullet NP$ (the "Las-Vegas" version of NP) is low for Σ^P_2.

□

COROLLARY 9.

$$\Sigma^P_k \bullet (\mathbf{ZP} \bullet NP) = \Sigma^P_k \quad \text{for } k > 2.$$

□

We now look at the operator "$\Delta\bullet$",

$\Delta \bullet C = \mathbf{N} \bullet co\text{-}C \cap \mathbf{co\text{-}N} \bullet C$ (definition 5, page 69)

$co\text{-}(\Delta \bullet C) = \mathbf{co\text{-}}\Delta \bullet co\text{-}C$ ("$\Delta\bullet$" is *sensible*)

$co\text{-}\Delta \bullet C = \mathbf{N} \bullet co\text{-}C \cap \mathbf{co\text{-}N} \bullet C$ (definition 5 "$\mathbf{co\text{-}}\Delta\bullet$")

As noted above (Lemma A), although "$\Delta\bullet$" is different from "$\mathbf{co\text{-}}\Delta\bullet$" by definition, the class $\Delta \bullet C$ (for any complexity class C) is closed under complement.

Applying "$\Delta\bullet$" on specific complexity classes we have :

$$\Delta \bullet NP = \mathbf{N} \bullet co\text{-}NP \cap \mathbf{co\text{-}N} \bullet NP = \Sigma^P_2 \cap \Pi^P_2$$

As mentioned above,

$$\text{co-}(\Delta\bullet NP) = \textbf{co-}\Delta\bullet co\text{-}NP$$

$$= \textbf{N}\bullet co\text{-}NP \cap \textbf{co-N}\bullet NP = \Sigma^P_2 \cap \Pi^P_2 = \Delta\bullet NP.$$

$\Delta\bullet co\text{-}NP$ is also closed under complement,

$$\Delta\bullet co\text{-}NP = \textbf{N}\bullet NP \cap \textbf{co-N}\bullet co\text{-}NP = NP \cap \text{co-NP} = {}_\Delta P$$

For the classes in the Polynomial Hierarchy in general,

$$\Delta\bullet\Sigma^P_k = \Sigma^P_{k+1} \cap \Pi^P_{k+1} = \textbf{co-}\Delta\bullet\Pi^P_k$$

On the complexity class RP, the operator "$\Delta\bullet$" has the following outcome:

$\Delta\bullet RP = \textbf{N}\bullet co\text{-}RP \cap \textbf{co-N}\bullet RP$. Since $\textbf{N}\bullet co\text{-}RP \subseteq$ MA and $\textbf{co-N}\bullet RP \subseteq$ co-MA, it follows that $\Delta\bullet\text{RP} \subseteq \text{MA} \cap \text{co-MA}$.

$$\text{co-}\Delta\bullet RP = \textbf{N}\bullet RP \cap \textbf{co-N}\bullet co\text{-}RP = NP \cap \text{co-NP} = {}_\Delta P$$

We now investigate the class $\textbf{ZP}\bullet D^P$. The class D^P was first introduced in [23] and consists of languages expressed in the form $(X \cap Y)$, where $X \in$ NP and $Y \in$ co-NP. As noted in [23] the class D^P is different from (NP $\cap$ co-NP). It contains all of (NP $\cup$ co-NP) and is not equal to the union unless NP = co-NP. The class D^P when used as an oracle, is equivalent to two calls to an NP oracle.

$$\textbf{ZP}\bullet D^P = \textbf{co-R}\bullet D^P \cap \textbf{R}\bullet co\text{-}D^P.$$

In order to formalize the above expression, let $P_1(.,.,.)$ and $P_2(.,.,.)$ be polynomial time predicates. For any language $L \in$ **co-R•**D^P there is a $D \in D^P$ *such that:*

$$x \in L \text{ ---> } \forall\, y\ D(x,y)$$

$$x \notin L \text{ ---> } \exists^+ y\ \neg D(x,y)$$

where there exist, $N_1 \in$ NP and $N_2 \in$ co-NP such that:

$$D(x,y) \text{ <--> } N_1(x,y) \wedge N_2(x,y) \text{ and}$$

$$\neg D(x,y) \text{ <--> } \neg N_1(x,y) \vee \neg N_2(x,y)$$

that is, there exist P_1 and P_2 in P such that:

$$(x,y) \in N_1 \text{ <----> } \exists\ z\ P_1(x,y,z) \text{ and}$$

$$(x,y) \in N_2 \text{ <----> } \forall\ z\ P_2(x,y,z).$$

It follows that,

$$x \in L \text{ <----> } \forall\ y\ [\exists z P_1(x,y,z) \wedge \forall z P_2(x,y,z)]$$

$$x \notin L \text{ <----> } \exists^+ y[\forall z\ \neg P_1(x,y,z) \vee \exists z\ \neg P_2(x,y,z)]$$

In Chapter IV, we will show that $NP^{BPP} \subseteq$ **ZP•**D^P. While it seems doubtful that NP $\subseteq$ **ZP•***NP* (NP and **ZP•***NP* are not directly comparable), we conclude that NP $\subseteq$ **ZP•**D^P

It is interesting to study operators and complexity classes in terms of quantifiers, since the quantifier characterizations are closely connected to restrictions of the number of oracle queries a machine makes along any computation path. We present

in Table 1., at the end of Chapter V, a detailed picture of complexity classes when used as operators. The following figure depicts the inclusion structure of some complexity classes in terms of operators.

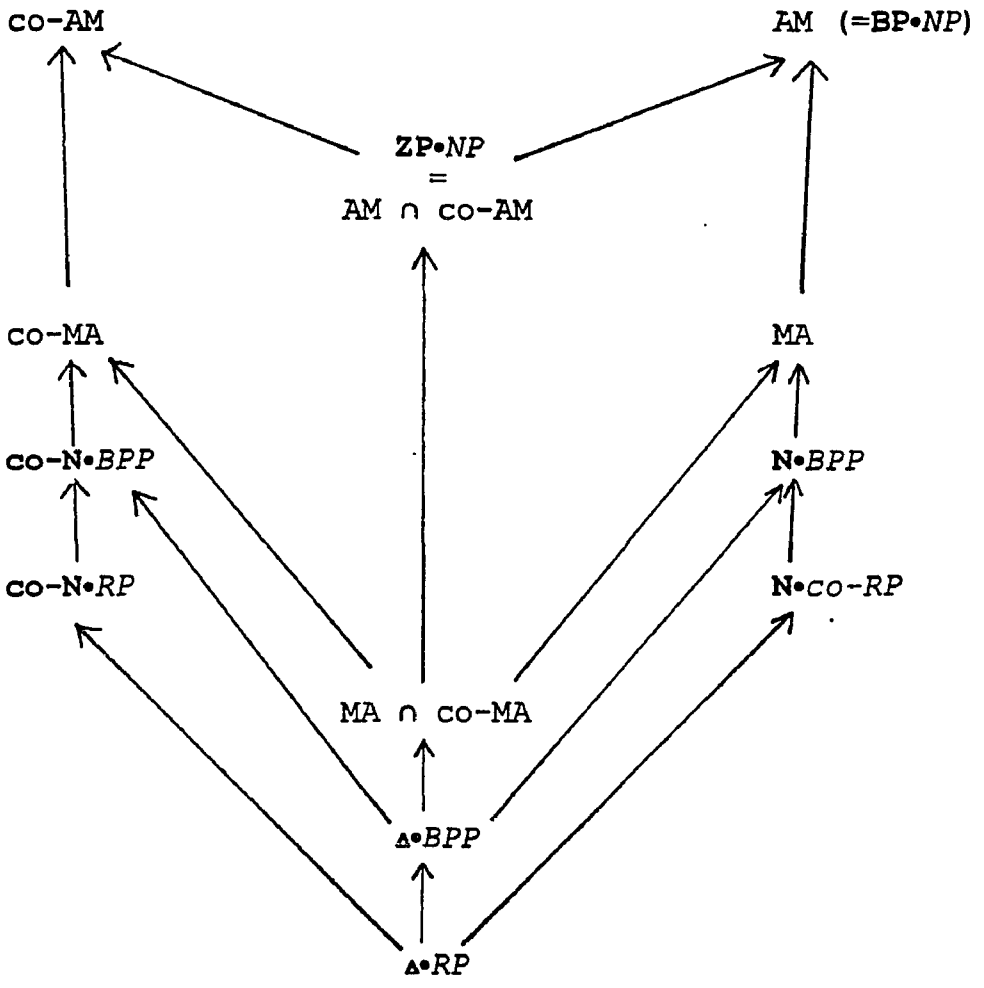

FIG 7. OPERATORS ON COMPLEXITY CLASSES

In the following section we investigate the properties of the Counting Quantifier [32] in the above light.

6. THE COUNTING QUANTIFIER

Many well-known complexity classes can be defined in terms of Nondeterministic Turing Machines (operating in polynomial time), by appropriate interpretations of the results of all possible computation paths. NP, co-NP, PP and members of the Polynomial Hierarchy [29] are examples of such complexity classes. Counting classes consist of languages in which acceptance is determined by the number of accepting computations.

The Polynomial time Counting Hierarchy (CH), was introduced by Wagner in [36], analogous to the Polynomial Time Hierarchy (PH). Many natural computational problems whose complexity cannot be modelled in terms of Existential ($\exists$), Universal ($\forall$) or Majority ($\exists^+$) quantifiers are more adapted to the idea of counting. Wagner introduced the Counting quantifiers $C^{P^{(n)}}_{\geq f(x)}$. Informally, the quantifier $C^{P^{(n)}}_{\geq f(x)} y\ P(y)$ means that there are at least f(x) strings y, of length p(x), satisfying the

predicate P(y). $C^{p(n)}_{=f(x)}$ y P(y) means that there are exactly f(x) strings of length p(x) satisfying the predicate P(y)).

The Counting Hierarchy (CH) arises in a natural way, by combining the Counting quantifiers with the Existential & the Universal quantifiers. The Counting Hierarchy (CH) contains PH & is contained in PSPACE. Wagner has shown [36] that every level of CH has complete problems.

For a NDTM N, let $|Acc_N(x)|$ be the number of paths of N, accepting the input x, $|Path_N(x)|$ be the total number of paths of N on input x. Before discussing the counting quantifier, we preview the following notations and definitions in order to relate to the counting classes.

$CP_{R(x,\alpha(x),\pi(x))}$ be a class of languages L defined as follows:
$(\exists N)(\forall x)$ $[x \in L \iff R(x, |Acc_N(x)|, |Path_N(x)|)]$.
It can be easily observed that if we let $\alpha(x) > r(x)\pi(x)$, then,

* PP : $0 < r(x) < 1$
* co-NP : $r(x) = 1$
* Σ^* : $r(x) = 0$

* If we let $R(x,\alpha(x),\pi(x)) \equiv (\alpha(x) \equiv 1 \pmod 2)$ then the class defined is $\oplus P$ [24].
* If we let $R(x,\alpha(x),\pi(x)) \equiv (\alpha(x) \not\equiv 0 \pmod k)$ then the class defined is MOD_kP.
* If we let $R(x,\alpha(x),\pi(x)) \equiv (\alpha(x) = 1)$ then the class defined is UP (Unique P).

Definition 8. The class MOD_kP :

For every positive integer k, $MOD_kP = CP_{\alpha(x)\not\equiv 0 (\mathrm{mod} k)}$. that is, a language L belongs to MOD_kP iff there exists a NDTM N, such that

$x \in L \leftrightarrow |Acc_N(x)| \not\equiv 0 \bmod(k)$.

The class $\oplus P$ defined in [24] is the class MOD_kP such that k=2.

Definition 9. The class UP (Unique P) [33] :

A language is in UP if there is a polynomial time NDTM N such that for all input x,

$|Acc_N(x)| = 1$ if $x \in L$ and $|Acc_N(x)| = 0$ if $x \notin L$.

The class UP was the first complexity class defined using the idea of bounding the number of accepting paths of a language accepted by a Nondeterministic Turing machine. This class

plays an important role in areas of one-way functions and Cryptography [16].

Definition 10. The polynomial Counting quantifier $C^{p(|x|)}_{\geq f(x)}$ y, (introduced in [36]):

Let FP denote the class of polynomial time computable functions. For a function f: $\Sigma^* \rightarrow \mathbb{N}$, $f \in FP$, a polynomial p and a predicate P(.,.),

$C^{p(|x|)}_{\geq f(x)}$ y P(x,y) <=> $||\{y: |y| \leq p(|x|) \ \& \ P(x,y)\}|| \geq f(x)$,

where ||S|| is the cardinality of the set S.

We can similarly define $C^{p(|x|)}_{=f(x)}$ y as follows :

$C^{p(|x|)}_{=f(x)}$ y P(x,y) <=> $||\{y: |y| \leq p(|x|) \ \& \ P(x,y)\}|| = f(x)$.

Since we only consider quantifiers ranging over strings of polynomial length, we drop the superscript p.

Definition 11. * The quantifier $C_{\geq}$ y:

$$C_{\geq}\ y\ P(x,y) = \bigcup_{f \in FP} C_{\geq f(x)}\ y\ P(x,y).$$

* The quantifier $C_{=}$ y:

$$C_{=}\ y\ P(x,y) = \bigcup_{f \in FP} C_{=f(x)}\ y\ P(x,y).$$

Definition 12. The operator "$C_{\epsilon}\bullet$" (corresponding to the quantifier C_{ϵ} y P(x,y)) :

If C is a complexity class involving polynomial time, then $C_{\epsilon}\bullet C$ is defined as follows:

$L \in C_{\epsilon}\bullet C$, iff there is a function f in FP, such that for every x, f(x) > 0, a polynomial p and a language $L_1 \in C$,

$$x \in L \iff C^{p(x)}_{\epsilon f(x)}\ y\colon L_1(x,y).$$

In [37], a characterization of the polynomial hierarchy in terms of polynomially bounded existential and universal quantifiers is given. We now alternate the polynomial Counting quantifier $C_{\epsilon}\bullet$ with the Existential and the Universal quantifiers in order to define the Counting Hierarchy (CH).

Definition 13. The polynomial Counting Hierarchy (CH) [32] is the smallest family of language classes satisfying :

(i) P $\in$ CH

(ii) If K $\in$ CH then $\forall K$, $\exists K$ and $C_{\epsilon}K \in$ CH, where all quantifiers range over strings of polynomial length.

The threshold function f(x) of the operator "$\mathbf{C}_{\pm}\bullet$" can be changed to one half of the possible strings. This fact has been observed in [27], [32] and [36]. $\mathbf{C}_{\pm}\bullet P$ is therefore the same class as PP. It has also been shown that $PP^{C} = \mathbf{C}_{\pm}\bullet C$ for any class C in CH [32].

LEMMA 10.

"$\mathbf{C}_{\pm}\bullet$" and "$\oplus\bullet$" are sensible operators.

Proof :

In order for "$\mathbf{C}_{\pm}\bullet$" to be a sensible operator it must satisfy properties 1 and 2 (Section 3).

Property 1. (Co-property) :

$co\text{-}(\mathbf{C}_{\pm}\bullet C) = co\text{-}\mathbf{C}_{\pm}\bullet co\text{-}C = \mathbf{C}_{\pm}\bullet co\text{-}C$ (see [32]).

Property 2. (Monotonicity) :

$C \subseteq C' \Rightarrow \mathbf{C}_{\pm}\bullet C \subseteq \mathbf{C}_{\pm}\bullet C'$

The result follows, since $PP^{C} \subseteq PP^{C'}$.

For the operator "$\oplus\bullet$" we verify the co-property :

$co\text{-}(\oplus\bullet C) = \mathbf{co\text{-}}\oplus\bullet co\text{-}C.$

Let $L \in co\text{-}(\oplus\bullet C)$ then there is a $L_1 \in C$:

$x \in L \dashrightarrow \neg \oplus y\ L_1(x,y)$ and

$x \notin L \dashrightarrow \oplus y\ L_1(x,y)$.

We consider a complete binary tree (then there are an even number of paths) :

$\neg \oplus y\ L_1(x,y) \Longleftrightarrow \neg \oplus y\ \neg L_2(x,y)$, where L_2 is in co-C.

$x \in L \dashrightarrow \neg \oplus y\ \neg L_2(x,y)$ and

$x \notin L \dashrightarrow \oplus y\ \neg L_2(x,y)$.

By definition of "**co-⊕•**", $L \in$ co-($\oplus \cdot C$) iff

$L \in$ **co-⊕•***co-C*.

It can be verified that Monotonicity and Distributive properties also hold true for the "**⊕•**" operator.

■

We use the notation "**C≥•**" interchangeably with "**PP•**". It was shown in [30] by Toda that PH ⊆ P^{PP} and PH ⊆ **BP⊕•***P*. This result was generalized to other counting classes in [31] where it was shown that : PH ⊆ **BP•***K* such that $K \in \{PP, MOD_k, C_=\}$. We provide a new structured skeleton of a proof of the main result of [30].

The following Lemma can be proved by combinatorial swapping arguments similar to [39].

LEMMA 11. (Parity Swapping):

$\oplus \cdot BPP \subseteq \mathbf{BP} \cdot \oplus P$.

□

COROLLARY 12.

- $\oplus \cdot \mathbf{BP} \cdot C \subseteq \mathbf{BP} \cdot \oplus \cdot C$, where C is a polynomial time complexity class.
- $\oplus \cdot \mathbf{BP} \cdot \oplus P \subseteq \mathbf{BP} \cdot \oplus P$

□

THEOREM 13.

$PH \subseteq P^{PP}$.

Proof (sketch):

The proof utilizes the properties of the class $\oplus P$ [24].

1.) $\oplus \cdot \oplus P = \oplus P$ [24].

2.) $\oplus \cdot BPP \subseteq BP \cdot \oplus P$ and $\oplus \cdot BP \cdot \oplus P \subseteq BP \cdot \oplus P$ [LEMMA 11].

3.) $NP \subseteq \mathbf{BP} \cdot \oplus P$ [35].

4.) $\sum_k^P \subseteq \mathbf{BP} \cdot \oplus \cdot \prod_k^P$, using the same argument as in [35].

5.) PH $\subseteq$ $\mathbf{BP}\bullet\circledast P$, by induction on levels of PH, using 2. and 4. above.

6.) $\mathbf{C}_{\Sigma}\bullet\circledast P \subseteq P^{PP}$ [30], i.e., $PP^{\circledast P} \subseteq P^{PP}$

7.) $\mathbf{BP}\bullet\circledast P \subseteq \mathbf{C}_{\Sigma}\bullet\circledast P$.

8.) PH $\subseteq P^{PP}$, using 5., 6. and 7. above.

□

COROLLARY 14.

(i) $\mathbf{BP}\bullet\oplus\bullet PH \subseteq \mathbf{BP}\bullet\circledast P$.

(ii) $\mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus\bullet PH \subseteq P^{PP}$

Proof :

(i) "$\mathbf{BP}\bullet\oplus\bullet$" is a sensible operator. Since PH $\subseteq$ $\mathbf{BP}\bullet\oplus P$, we apply the operator "$\mathbf{BP}\bullet\oplus\bullet$" on both the classes and obtain,

$\mathbf{BP}\bullet\oplus\bullet PH \subseteq \mathbf{BP}\bullet\oplus\mathbf{BP}\bullet\oplus P \subseteq \mathbf{BP}\bullet\oplus P$ (using corollary 12.)

(ii) From corollary 12. above, we know that $\mathbf{C}_{\Sigma}\bullet\oplus\bullet P \subseteq P^{PP}$ and also $\mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus\bullet P \subseteq \mathbf{C}_{\Sigma}\bullet\oplus\bullet P$. Therefore, $\mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus P \subseteq \mathbf{C}_{\Sigma}\bullet\oplus P \subseteq P^{PP}$. Moreover, since " $\mathbf{C}_{\Sigma}\bullet$" is a sensible operator and PH $\subseteq$ $\mathbf{BP}\bullet\oplus P$,

$$\mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus\bullet PH \subseteq \mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus\bullet P \subseteq \mathbf{C}_{\Sigma}\bullet\oplus\bullet P.$$

Hence, $\mathbf{C}_{\Sigma}\bullet\mathbf{BP}\bullet\oplus\bullet PH \subseteq P^{PP}$

□

LEMMA 15.

If the operator "$\mathbf{C}_{\Sigma}\bullet$" is idempotent (Property 8.[section 3]) then PH $\subseteq$ PP.

Proof :

From theorem 13. we know, PH $\subseteq$ P^{PP}.

If "$\mathbf{C}_{\Sigma}\bullet$" is idempotent then $\mathbf{C}_{\Sigma}\bullet\mathbf{C}_{\Sigma}\bullet P = \mathbf{C}_{\Sigma}\bullet P$ i.e., $PP^{PP} \subseteq PP$.

Hence the result follows.

□

Depending on the threshold function f(x) (which defines the acceptability criteria for the counting machine) the class $\mathbf{C}_{\Sigma}\bullet P$ could represent various classes. For example,

(a) If f(x) = 1 then $\mathbf{C}_{\Sigma f(x)}\bullet P$ represents the class NP.

(b If f(x) = {all possible paths} then $\mathbf{C}_{\Sigma f(x)}\bullet P$ represents the class co-NP.

(c) If f(x) = 1 then $\mathbf{C}_{=f(x)}\bullet P$ represents the class UP (At most one accepting computation).

Note: If f(x) = 1 then "$\mathbf{C}_{\Sigma f(x)}\bullet$" is idempotent.

IV BPP ORACLES : LOW POWER

1. INTRODUCTION

BPP seems to be a much more tractable complexity class than NP, but currently we know nothing definite about the inclusion relations, if any, between NP and BPP. It is known that if NP $\subseteq$ BPP, then RP = NP and in addition, the Polynomial Hierarchy collapses to BPP [43]. Sipser showed that BPP $\subseteq$ $\sum_4^P$ [28] and Lautemann [21] improved this to $\sum_2^P \cap \prod_2^P$. A much simplified proof of this result is presented in [41], where it is also shown that BPP $\subseteq$ ZPP^{NP} [41]. In [40] Zachos and Heller showed that BPP $\subseteq$ AM $\cap$ co-AM and also that BPP $\subseteq$ MA $\cap$ co-MA. The two classes NP and BPP at this time seem to be incomparable. Comparing the augmenting power with the use of oracles of NP and BPP complexity provides a deeper insight into the comparison of the two classes.

It is known that P^{BPP} = BPP, where as, P^{NP} probably contains NP in a proper way. BPP is "low" when used as an oracle. E.g.,

$\Sigma_2^{P,BPP} = \Sigma_2^P$ [41] where as $\Sigma_2^{P,NP} = \Sigma_3^P$, which probably contains Σ_2 in a proper way.

We will show that $NP^{BPP} \subseteq ZPP^{NP}$ and this provides further evidence of the "lowness" of BPP. As a matter of fact the stronger version of the result is $NP^{BPP} \subseteq \mathbf{ZP}\cdot D^P$. This latter class is a Las Vegas version of D^P (i.e., two questions to an NP oracle are enough). The class NP^{BPP} is interesting in itself. It contains NP, BPP and RP and on the other hand is contained in MA and consequently in AM. We do not know if MA is contained in $\mathbf{ZP}\cdot D^P$. We also do not know if MA is even contained in ZPP^{NP}. We do know that (MA $\cap$ co-MA) $\subseteq$ (AM $\cap$ co-AM) $\subseteq \mathbf{ZP}\cdot NP$.

2. CHARACTERIZATION OF BPP.

The class BPP as we know, is the set of all decision problems solvable by probabilistic polynomial time Turing Machines in which the answer always has a probability of at least $\frac{1}{2}+\delta$ of being correct, for some fixed $\delta > 0$. We have seen in Chapter I that BPP = $(\exists^+/\exists^+)$. The "B" in BPP stands for "bounded error". Since the probability of correctness can be rapidly

increased by iteration, the precise choice of the parameter δ in the definition of the quantifier "$\exists^+$" does not affect the language class it defines. The quantity $\frac{1}{2}+\delta$ i.e., the threshold may be replaced by $1-2^{-q(|x|)}$ in the definition. This phenomenon is known as robustness and has been studied in detail in [39] and [41].

The classes defined by means of the quantifiers $\exists^+$ exhibit the robustness property. The robustness property for the class BPP yields the fact that BPP problems can be solved in practice with small probability of error . This is not known to be the case for all problems in NP. It has been shown in [39] that $BPP = (\exists^+\forall/\forall\exists^+)$ and since BPP is closed under complementation:

$$BPP = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall).$$

This characterization of BPP is called decisive [39] in the following sense:

For $L \in BPP$ i.e., $x \in L \leftrightarrow \exists^+u\forall v\ P(x,u,v)$

and $x \notin L \leftrightarrow \forall u\exists^+v\ \neg P(x,u,v)$, for a polynomial predicate $P(.,.,.)$.

Thus for this P :

$x \in L$ iff $\exists u\forall v\ P(x,u,v)$ iff $\exists^+u\forall v\ P(x,u,v)$

and $x \notin L$ iff $\forall u\exists v\ \neg P(x,u,v)$ iff $\forall u\exists^+v\ \neg P(x,u,v)$.

We have:

$$\exists u \forall v \; P(x,u,v) \; <--> \; x \in L$$

$$\text{and} \quad \forall u \exists v \; \neg P(x,u,v) \; <--> \; x \notin L$$

This means even if "+" is dropped, it can be decided whether $x \in L$ or $x \notin L$. This is not true for the $(\exists^+/\exists^+)$ characterization of BPP.

The hierarchy defined by alternations of $\exists^+$ and $\forall$ collapses at the second level i.e.,

$(\exists^+\forall\exists^+ \ldots Q_k)/(\forall\exists^+\forall \ldots Q'_k) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$, where $Q_k = \exists^+$ and $Q'_k = \forall$ if k, the number of alternations is odd and $Q_k = \forall$ and $Q'_k = \exists^+$ otherwise.

On the other hand, we do not know if the hierarchy defined by operators, i.e.,

$$\mathbf{R}\bullet co\text{-}RP \subseteq \mathbf{R}\bullet\mathbf{co}\text{-}\mathbf{R}\bullet RP \subseteq \ldots\ldots \subseteq \text{BPP collapses.}$$

3. USING BOOLEAN MATRICES TO VISUALIZE QUANTIFIED FORMULAE

LEMMA 1. (Swapping Lemma) [39]

$\forall y\ \exists^{+}z\ P(x,y,z) \dashrightarrow \exists^{+}C\ \forall y\ \exists z \epsilon C\ P(x,y,z)$

where $P(x,y,z)$ is a polynomial time predicate and $\exists z \epsilon C\ P(x,y,z)$ is a polynomial size disjunction, hence also a polynomial time predicate .

□

In order to have a practical and direct view of quantifiers, we shall consider a matrix $M[y,v] = P(x,y,v)$ in terms of 0's and 1's for an input x. We now make the following interpretations:

* $\forall y \exists^{+}v\ M[y,v] = 1$ => for all rows most entries are 1's.

* $\exists^{+}C \forall y\ \exists v \epsilon C\ M[y,v] = 1$ => for most collections (combs) C of columns, every row has a 1 in at least one column (tooth of the comb) of C.

* The statement corresponding to Lemma 1. above can be interpreted to mean: for all rows most entries are 1's

implies that for most collections C of columns, all rows have a 1 in at least one column of C.

* $\forall u\ (\exists^{+}vP(x,u,v) \vee \exists^{+}v\ \neg P(x,u,v))$ in terms of the matrix M[u,v] according to the definitions of $\exists^{+}$ is:

There exists a $\delta>0$, such that,

$\forall u\ \{Pr\{v|\ M[u,v] = 1\} > \frac{1}{2}+\delta \wedge Pr\{v|\ M[u,v] = 0\} > \frac{1}{2}+\delta\}$.

Putting this in simple words, we have, for all rows either most entries are 0's or most entries are 1's.

Some other swapping properties of quantifiers are:

Let $Q \in \{\exists, \forall, \exists^{+}, \not\exists\}$ then

(i) $Qy\ \forall z\ P(x,y,z) \dashrightarrow \forall z\ Qy\ P(x,y,z)$

(ii) $\exists y\ Qz\ P(x,y,z) \dashrightarrow Qz\ \exists y\ P(x,y,z)$

We refer to a collection C (or a set) of columns as a comb such that $|C| \leq g(|x|)$ for some polynomial g(.). Each member column of this comb is a tooth. A comb C is called faithful (*faithful*(C)) iff for all rows, most teeth of this comb have 0's or most teeth of the comb have 1's. In terms of quantifiers we have,

faithful(C) $\Leftrightarrow \forall u\ (\exists^{+}v \in C\ P(x,u,v) \vee \exists^{+}v \in C\ \neg P(x,u,v))$.

faithful(C) is a co-NP predicate.

4. BPP AS AN ORACLE TO NP, MA AND AM

The BPP oracle as we know does not enhance the power of Σ_2^P. We will show that augmenting the NP-machine with a BPP oracle yields no more than the Las Vegas version of the class D^P.

THEOREM 2.

$$NP^{BPP} \subseteq ZP \cdot D^P .$$

Proof:

Let $L \in NP^{BPP}$.

In terms of quantifier notations,

$$x \in L \dashrightarrow \exists u \; P^{BPP}(x,u) \qquad (1)$$

$$x \notin L \dashrightarrow \forall u \; \neg P^{BPP}(x,u) \qquad (2)$$

where $P^{BPP}(x,u)$ is a polynomial-time predicate with a BPP oracle. Considering the behavior of the BPP oracle $L \in NP^{BPP}$ iff there exists a polynomial time predicate $P(.,.,.)$ such that,

$$x \in L \dashrightarrow \exists u \; \exists^{+} v \; P(x,u,v) \qquad (1')$$

$$\text{and} \quad x \notin L \dashrightarrow \forall u \; \exists^{+} v \; \neg P(x,u,v) \qquad (2')$$

$$\text{and} \quad \forall x \forall u (\exists^{+} v \; P(x,u,v) \vee \exists^{+} v \; \neg P(x,u,v)) \qquad (3')$$

Now $\forall x \forall u \; (\exists^{+} v \; P(x,u,v) \vee \exists^{+} v \; \neg P(x,u,v))$ implies:

$\forall x \forall u\ \exists^{+}C\ (\exists^{+}v \in C\ P(x,u,v)\ \vee\ \exists^{+}v \in C\ \neg P(x,u,v))$, where C is a polynomial size collection of columns (a comb).

Using the proof technique of Lemma 1., from (3') get,

$\forall x\ \exists^{+}C\ \forall u\ (\exists^{+}v \in C\ P(x,u,v)\ \vee\ \exists^{+}v \in C\ \neg P(x,u,v))$ (3'')

Using the abbreviation of *faithful*(C) from the previous section, condition (3'') precisely means that for all inputs, most combs are *faithful* i.e.,

$\forall x\ \exists^{+}C$ *faithful*(C).

From conditions (1'), (2') & (3'') we obtain,

$x \in L \rightarrow \exists u \exists^{+}v\ P(x,u,v)\ \wedge\ \exists^{+}C$ *faithful*(C)

$\rightarrow \exists^{+}C\ (\exists u \exists^{+}v \in C\ P(x,u,v)\ \wedge$ *faithful*(C))

$x \notin L \rightarrow \forall u \exists^{+}v\ \neg P(x,u,v)$

$\rightarrow \forall C\ ($*faithful*(C) $\rightarrow\ \forall u \exists^{+}v \in C\ \neg P(x,u,v))$

This shows that $L \in R^{NP}$.

In the characterization of L, only two queries to the NP oracle are required:

$x \in L \rightarrow \exists^{+}C\ Q(x,C)\ \&\ x \notin L \rightarrow \forall C\ \neg Q(x,C)$

where $Q \in D^{P}$.

We also know that $NP^{BPP} \subseteq MA \subseteq co\text{-}R^{NP[1]} \subseteq \mathbf{co\text{-}R{\bullet}D^P}$, hence we conclude that $NP^{BPP} \subseteq \mathbf{ZP{\bullet}D^P}$.

□

Of course $NP^{BPP} \subseteq MA^{BPP}$ because $NP \subseteq MA$ and also $AM = \mathbf{BP{\bullet}NP} \subseteq BPP^{NP}$.

THEOREM 3.

(i) $MA^{BPP} = MA$

(ii) $AM^{BPP} = AM$

Proof:

(i) Let $L \in MA^{BPP}$. This means,

$$x \in L \dashrightarrow \exists u \exists^{+} v \; P^{BPP}(x,u,v) \quad \text{and}$$

$$x \notin L \dashrightarrow \forall u \exists^{+} v \; \neg P^{BPP}(x,u,v).$$

Then by the identity $P^{BPP} = BPP$ and quantifier contraction, for some polynomial-time predicate P':

$$x \in L \dashrightarrow \exists u \exists^{+} y \; P'(x,u,y) \quad \text{and}$$

$$x \notin L \dashrightarrow \forall u \exists^{+} y \; \neg P'(x,u,y).$$

Hence $L \in MA$.

(ii) Analogous to the above.

□

V. CONCLUSIONS AND INTERESTING OPEN PROBLEMS

We have investigated trees as representational models for the computation of a Nondeterministic Turing Machine. In Chapter II we have shown the invariance of the classes PP, #P and counting subclasses of #P viz., $\oplus$P, MOD_kP with respect to computational trees of various structures, viz., varying fan-out and different length of computational paths. We have proved, taking into account all the properties of the computational models, that we can represent any computation of a PP machine and counting machines (the class #P and its subclasses) with complete binary trees. We can extend the techniques used here to other classes e.g., the classes MA and AM, of games between two players.

In Chapter III we have created a calculus of operators on complexity classes and have investigated properties that lead us to define sensible operators. We observe that by forcing certain properties on operators, we may imply unexpected collapse results. We have introduced operators "$\Delta\bullet$" and "ZP$\bullet$" corresponding to the classes ΔP and ZPP. The operator "ZP$\bullet$"

provides us with a medium to study the "Las-Vegas" version of various complexity classes. The term "Las-Vegas" algorithms refers to randomized algorithms which either give a correct answer or no answer at all (where the probability of the second option can be made exponentially small by repeating the algorithm). The "Las-Vegas" version of any complexity class is closed under complementation. A "Las-Vegas" algorithm (ZPP) for primality testing appeared in [1], whereas no P algorithm is known for primality yet. The class ZPP offers realistic proposals for practical algorithms. We have found that another interesting problem i.e., the Graph Isomorphism problem, is contained in the "Las-Vegas" of NP (**ZP•NP**). The class **ZP•NP** is closed under complementation and, therefore, the Graph Nonisomorphism problem is also in **ZP•NP**. The Graph Isomorphism problem is known to be in NP and not known to be in co-NP. It is known from [7] that the Polynomial Hierarchy will collapse to the second level if the Graph Isomorphism problem turns out to be NP-complete. In fact we conclude that if there is an NP complete problem that can be shown to be in the Las-Vegas version of NP (i.e., **ZP•NP**) then the Polynomial Hierarchy collapses to the second level. We also conclude that the class **ZP•NP** when used as an oracle does not add any computational power to classes as low as $\sum_2^P$ in the Polynomial Hierarchy

(PH). We have also investigated the consequences of specifying certain properties for the Counting Operator (the Counting Quantifier). In the same chapter III we also provide a new structured skeleton of Toda's celebrated result : $PH \subseteq P^{PP}$.

It has been widely discussed and it is known, that the classes NP and BPP at the present time are not directly comparable and that BPP is "low" when used as an oracle, [17], [19], [26], [41]. In Chapter IV we have presented further evidence on the "lowness" of BPP as an oracle. We have shown that the class BPP when used as an oracle does not substantially enhance the computational power of the classes NP, MA and AM. The class NP^{BPP} is contained in the Las-Vegas version of D^P (i.e., $\mathbf{ZP}\cdot D^P$). On the other hand, for a slightly higher class, namely MA, we do not know if it is contained in $\mathbf{ZP}\cdot D^P$ or even in ZPP^{NP}. We would, therefore, be interested in investigating further the classes $\mathbf{ZP}\cdot NP$ and ZPP^{NP}. We have also shown that $MA^{BPP} = MA$ and $AM^{BPP} = AM$, i.e., the BPP oracle does not enhance the computational power of the classes MA and AM. At the end of this chapter, we present in a tabular form (table 1 and table 2) some interesting oracle and operator classes.

OPEN PROBLEMS :

* $MA \in \mathbf{ZP} \bullet D^P$ or $MA \in ZPP^{NP}$?

* $\mathbf{R} \bullet co\text{-}RP = \mathbf{R} \bullet BPP$?

* $\mathbf{R} \bullet co\text{-}RP = BPP$?

* $\mathbf{N} \bullet co\text{-}RP = \mathbf{N} \bullet BPP = MA$?

* $\Delta \bullet BPP = MA \cap co\text{-}MA$?

COMPLEXITY OPERATOR \ COMPLEXITY CLASS	P	ZPP	RP	co-RP	ΔP	BPP	NP	co-NP	Σ_2^P	$\oplus P$
P•	P	ZPP	RP	co-RP	ΔP	BPP	NP	co-NP	Σ_2^P	$\oplus P$
ZP•	ZPP	ZPP	ZP•RP	ZPP	ΔP	BPP	ZP•NP	ΔP	$ZP \bullet \Sigma_2^P$	ZP•⊕P
co-ZP•	ZPP	ZPP	ZPP	ZP•RP	ΔP	BPP	ΔP	ZP•NP	Δ_2^P	ZP•⊕P
R•	RP	RP	RP	R•co-RP	$R \bullet \Delta P$	BPP	NP	co-AM	Σ_2^P	R•⊕P
co-R•	co-RP	co-RP	co-R•RP	co-RP	$co\text{-}R \bullet \Delta P$	BPP	AM	co-NP	$co\text{-}R \bullet \Sigma_2$	co-R•⊕P
Δ•	ΔP	ΔP	Δ•RP	ΔP	ΔP	Δ•BPP	Δ_2^P	ΔP	Δ_3	Δ•⊕P
co-Δ•	ΔP	ΔP	ΔP	Δ•RP	ΔP	Δ•BPP	ΔP	Δ_2^P	Δ_2	Δ•⊕P
BP•	BPP	BPP	BPP	BPP	$BP \bullet \Delta P$	BPP	AM	co-AM	$BP \bullet \Sigma_2$	BP•⊕P
N•	NP	NP	NP	N•co-RP	NP	N•BPP	NP	Σ_2^P	Σ_2^P	N•⊕P
co-N•	co-NP	co-NP	co-N•RP	co-NP	co-NP	co-N•BPP	Π_2^P	co-NP	Π^P_3	co-N•⊕P
⊕P	⊕P	⊕•ZPP	⊕•RP	⊕•co-RP	⊕•ΔP	⊕•BPP	⊕•NP	⊕•co-NP	$\oplus \bullet \Sigma_2^P$	⊕P

TABLE 1. OPERATORS ON COMPLEXITY CLASSES

COMPLEXITY CLASS \ ORACLE SET	P	ZPP	RP	co-RP	ΔP	BPP	NP	co-NP	Σ_2^P	$\oplus P$
P	P	ZPP	P^{RP}	P^{RP}	ΔP	BPP	P^{NP}	P^{NP}	P^{Σ}	$\oplus P$
ZPP	ZPP	ZPP	ZPP^{RP}	ZPP^{RP}	ΔP	BPP	ZPP^{NP}	ZPP^{NP}	ZPP^{Σ}	$ZPP^{\oplus P}$
RP	RP	RP	$RP^{RP[1]}$	$RP^{RP[1]}$	$RP^{\Delta P}$	BPP	RP^{NP}	RP^{NP}	RP^{Σ}	$RP^{\oplus P}$
co-RP	co-RP	co-RP	co-$RP^{RP[1]}$	co-$RP^{RP[1]}$	co-$RP^{\Delta P}$	BPP	co-RP^{NP}	co-RP^{NP}	co-RP^{Σ}	co-$RP^{\oplus P}$
ΔP	ΔP	ΔP	ΔP^{RP}	ΔP^{RP}	ΔP	ΔP^{BPP}	$\Delta P^{NP[1]}$	$\Delta P^{NP[1]}$	ΔP^{Σ}	$\Delta P^{\oplus P}$
BPP	BPP	BPP	BPP	BPP	$BPP^{\Delta P}$	BPP	BPP^{NP}	BPP^{NP}	BPP^{Σ}	$BPP^{\oplus P}$
NP	NP	NP	$NP^{RP[1]}$	$NP^{RP[1]}$	NP	$NP^{BPP[1]}$	Σ_2^P	Σ_2^P	Σ_3^P	$NP^{\oplus P}$
co-NP	co-NP	co-NP	co-$NP^{RP[1]}$	co-$NP^{RP[1]}$	co-NP	co-$NP^{BPP[1]}$	Π_2^P	Π_2^P	Π_3^P	co-$NP^{\oplus P}$
MA	MA	MA	MA	MA	MA	MA	MA^{NP}	MA^{NP}	MA^{Σ}	$BPP^{\oplus P}$
AM	AM	AM	AM	AM	AM	AM	AM^{NP}	AM^{NP}	AM^{Σ}	$BPP^{\oplus P}$
Σ_2^P	Σ_2^P	Σ_2^P	Σ_2^P	Σ_2^P	Σ_2^P	Σ_2^P	Σ_3^P	Σ_3^P	Σ_4^P	$BPP^{\oplus P}$
$\oplus P$	$\oplus P$	$\oplus P^{ZPP}$	$\oplus P^{RP}$	$\oplus P^{RP}$	$\oplus P^{\Delta P}$	$\oplus P^{BPP}$	$\oplus P^{NP}$	$\oplus P^{NP}$	$\oplus P^{\Sigma}$	$\oplus P$

TABLE 2. COMPLEXITY CLASSES WITH ORACLES

VI. REFERENCES

[1] L. Adleman and M. Huang. Recognizing primes in random polynomial time Proc. 19th ACM Symposium on Theory of Computing (1987), pp 462-470.

[2] E. W. Allender. Invertible functions. Ph.D. dissertation Georgia Institute of Technology, (1985).

[3] L. Babai. Trading group theory for randomness. Proceedings 17th ACM Symposium on Theory of Computing (1985), pp 421-429.

[4] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. Journal of Computer and System Sciences, 36 (1988), pp 254-276.

[5] R. Beigel and J. Gill. Counting classes. Theoretical Computer Sci.. (1992), pp 3.

[6] A. Blass and Y. Guvervich. On the unique satisfiability problem. Information and Control 55(1982) pp 80-88.

[7] R. B. Boppana, J. Hastad and S. Zachos. Does co-NP have short interactive proofs? Information Processing letters, 25 (1987), pp 27-32.

[8] D. P. Bovet, P. Crescenzi and R. Silvestri. A uniform approach to define complexity classes. Theoretical Computer Sci., 104 (1992), pp 263-283.

[9] S. Fenner, L. Fortnow, S. A. Kurtz and L. Li. An oracle builders toolkit. Proceedings of Structure in Complexity Theory, (1993), pp 120-131.

[10] M. Furst, J. Hopcroft and E. Luks. A subexponential algorithm for trivalent graph isomorphism. Congressus Numerantium 28, (1980), pp 421-446.

[11] M. R. Garey and D.S. Johnson. Computers and Intractability. A guide to the theory of NP-Completeness. WH Freeman and Co., (1979).

[12] J. Gill. Computational Complexity of probabilistic Turing machines. SIAM Journal of Computing, 6 (1977), pp 675-695.

[13] L. M. Goldschlager and I. Parberry. On the construction of parallel Computers from various bases of Boolean functions. Theoretical Computer Science, 43 (1986), pp 43-58.

[14] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of Interactive Proof Systems. SIAM Journal of Computing, 18 (1989), pp 186-208.

[15] S. Goldwasser and Sipser. Private coins versus public coins in interactive proof systems. Proc. 18th ACM Symposium on Theory of Computing, (1986), pp 59-68.

[16] S. Grollmann and A. L. Selman. Complexity measures for public key cryptosystems. SIAM Journal of Computing, 17 (1988), pp 309-335.

[17] P. Hinman and S. Zachos. Probabilistic machines, oracles and quantifiers. Proceedings Oberwolfach Recursion-Theoretic Week, Lecture Notes in Mathematics, Vol. 1141, Springer-Verlag, (1984), pp 159-192.

[18] D. S. Johnson. The NP-completeness column: An ongoing guide. Jr. of Algorithms 5, (1984), pp 433-447.

[19] J. Köbler, U. Schöning, J. Torán. The Graph isomorphism problem: Its structural complexity. Birkhäuser, Boston (1993).

[20] J. Köbler, U. Schöning, S. Toda and J. Torán. Turing machines with few accepting computations and low set for PP. Structure in Complexity Theory (1989), pp 208-215.

[21] C. Lautemann. BPP and the polynomial hierarchy. Info. Processing letters 17 (1983), pp 215-218.

[22] C. H. Papadimitriou. Computational Complexity. Addison-Wesley, New York, (1994).

[23] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). Proceedings 14th ACM Symposium on Theory of Computing (1982), pp 255-260.

[24] C. H. Papadimitriou and S. Zachos. Two remarks on the power of counting. 6th GI conference on Theoretical Computer Science. Springer Verlag, (1983), 269-276.

[25] U. Schöning. Graph Isomorphism is in the low hierarchy. Journal of Computer and System Sci., 37 (1988), pp312-323.

[26] U. Schöning. Probabilistic complexity classes and lowness. Journal of Computer and System Sci., 39 (1989), pp84-100.

[26A] A. Shamir. IP = PSPACE. Proceedings, 31st IEEE Symposium on the Foundations of Computer Science, (1990), pp 11-15.

[27] J. Simon. On some central problems in computational complexity. PhD dissertation Cornell Univ. 1975.

[28] M. Sipser. A Complexity theoretic approach to randomness in Proceedings, 15th ACM Symposium on Theory of Computing, (1983), pp 330-335.

[29] L. J. Stockmeyer. The Polynomial Hierarchy. Theoretical Computer Sci., 3 (1976), pp 1-22.

[30] S. Toda. PP is as hard as the Polynomial Hierarchy. SIAM Journal on Computing, 20 (1991), pp 865-877.

[31] S. Toda and M. Ogiwara, Counting classes are at least as hard as the polynomial hierarchy. SIAM Journal on Computing, vol.21, 2 (1992), pp 316-328.

[32] J. Torán. Complexity classes defined by counting Quantifiers. Journal of ACM, vol. 38, 3 (1991), pp 753-774.

[33] L. G. Valiant. The relative complexity of checking and evaluating Information Processing letters, 5 (1976), pp 20-23.

[34] **L. G. Valiant.** The complexity of computing the permanent. Theoretical Computer Science, 8 (1979), pp 189-201.

[35] **L. G. Valiant and V.V. Vazirani.** NP is as easy as detecting unique solutions. Theoretical Computer Science, 47 (1986), pp 85-93.

[36] **K. Wagner.** The complexity of Combinatorial Problems with succinct input representations. Acta informatica 23 (1986), pp 325-356.

[37] **C. Wrathall.** Complete sets and the Polynomial Hierarchy. Theoretical Computer Science, 3 (1976), pp 23-33.

[38] **S. Zachos.** Robustness and probabilistic Computational Complexity Classes under definitional Perturbations. Information and Control 54 (1982), pp 143-154.

[39] **S. Zachos.** Probabilistic Quantifiers and games. Journal of Computer and System Sciences 36, 3 (1988), pp 433-451.

[40] **S. Zachos and H. Heller.** On BPP T/R 252, LCS, MIT, (1983).

[41] **S. Zachos and H. Heller.** A decisive characterization of BPP. Information and Control 69, (1986) pp 125-135.

[42] **S. Zachos.** Probabilistic oracles are less powerful than Nondeterministic oracles, T/R (1987), Brooklyn College.

[43] **S. Zachos.** Collapsing Probabilistic hierarchies, Proc. Conference on Computational Complexity Theory, Santa Barbara, (1983) pp 75-81.

[44] **S. Zachos and M. Fürer.** Probabilistic quantifiers vs distrustful adversaries. Proceedings FST-TCS, Lecture notes in Computer science, pp 443-455, Springer-Verlag, New York/ Berlin (1987), Vol 287.