Looking for an Analogue of Rice's Theorem in Complexity Theory

Bernd Borchert[*]
Universität Heidelberg

Frank Stephan[†]
Universität Heidelberg

Abstract

Rice's Theorem says that every nontrivial semantic property of programs is undecidable. It this spirit we show the following: Every nontrivial absolute (gap, relative) counting property of circuits is UP-hard with respect to polynomial-time Turing reductions.

1 Introduction

One of the nicest theorems in Recursion Theory is Rice's Theorem [17, 18]. Informally speaking it says: Any nontrivial semantic property of programs is undecidable. More formally it can be stated this way:

Theorem 1 (Rice 1953) *Let A be any nonempty proper subset of the partial recursive functions. Then the halting problem or its complement is many-one reducible to the following problem: Given a program p, does it compute a function from A?*

The theorem and its proof only use elementary notions of Recursion Theory. But the most interesting point about Rice's Theorem is that it has messages to people in practical computing: It tells programmers that for example there is no program which finds infinite loops in given programs, or that there is no program which checks if some given program does a specified job.

Intrigued by the simple beauty of Rice's Theorem we tried to find some sister of it in Complexity Theory. The first observation we made is the following. The typical undecidable problem in Recursion Theory is the halting problem, which is a problem on programs, whereas the typical hard problem in Complexity Theory is the satisfiability problem, which is a problem on circuits (in this paper we prefer circuits instead of CNF-formulas because they are the nonuniform counterpart of polynomial-time computations). Therefore, if we want to look for some analogue of Rice's Theorem in Complexity Theory we should look at problems on circuits. This

[*]Im Neuenheimer Feld 294, 69120 Heidelberg, bb@math.uni-heidelberg.de

[†]Im Neuenheimer Feld 294, 69120 Heidelberg, fstephan@math.uni-heidelberg.de; supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-1.

approach is different from the one of Kozen [14] who studies problems which have as inputs programs with given resource bounds.

Note that for circuits we have the same syntax/semantics dichotomy like we have for programs: a circuit is the description of a Boolean function like a program is the description of a partial recursive function. In other words, circuits (programs) are the syntactical objects whereas Boolean functions (partial recursive functions) are the corresponding semantic objects. Note that the question whether two syntactical objects describe the same semantic object is hard in both worlds: it is Π_2-complete for programs and co-NP-complete for circuits.

By these considerations, the perfect analogue of Rice's Theorem would be the following: Let A be any nonempty proper subset of the set of Boolean Functions. Then the following problem is NP-hard: Given a circuit c, does it compute a function from A? Unfortunately, there is the following simple counterexample for this claim. Let A be the set of Boolean functions which have the value 0 on the all-0-assignment. For a given circuit c this question can be computed in polynomial time. Therefore the statement above is false unless P=NP.

So we had to look for a more restrictive requirement for the set A for which a modification of the statement may be true. What we found is the (indeed very restrictive) requirement of *counting*: Let A be a set of Boolean functions for which membership in A only depends on the number of satisfying assignments, like for example the set of all Boolean functions which have at least one satisfying assignment, or the set of Boolean functions which have an odd number of satisfying assignments. We will call the corresponding sets of circuits *absolute counting problems*. In a similar fashion we will define *gap counting problems* and *relative counting problems*: they also incorporate the non-satisfying assignments in their definition. For example, the set of circuits which have more satisfying than non-satisfying assignments is a gap counting problem because it can be stated the following way: the gap (= difference) between the number of satisfying assignments and the number of non-satisfying assignments has to be greater than 0. And the same problem is a relative counting problem because it can be stated the following way: the relative number of satisfying assignments has to be greater than one half.

For each of these three types of counting problems a theorem in the fashion of Rice's Theorem can be shown:

> Any nontrivial absolute (gap, relative) counting property of circuits is UP-hard with respect to polynomial-time Turing reductions.

This result will be extended in terms of approximable sets and also in terms of randomized reductions.

For the following (somehow artificial) notion, which is the nonuniform counterpart of nondeterministic polynomial-time computation, a perfect analogue of Rice's Theorem could be found: Let an *existentially quantified circuit* be a circuit in which some variables are existentially quantified. Each such circuit describes in a natural way a Boolean function on its non-quantified variables. Here it can be shown: Let A be any nonempty proper subset of the set of Boolean functions which not only depends on arity. Then the following problem is NP-hard: Given an existentially quantified circuit c, does it describe a function from A?

2 Preliminaries

The standard notions of Theoretical Computer Science like words, languages, polynomial-time reductions, P, NP, etc. follow the book of Papadimitriou [16]. A central notion of this paper is the notion of a *Boolean function* which is defined to be a mapping from $\{0,1\}^n$ to $\{0,1\}$ for some integer n which is called its *arity. Circuits* are a standard way of representing Boolean functions. We will just assume that they are encoded as words in some standard way, for the details concerning circuits we refer for example to [16]. Remember that a given circuit can be evaluated on a given assignment in polynomial time. Each circuit c describes a Boolean function $F(x_1, \ldots, x_n)$. Note that here we have an example of the classical syntax/semantics dichotomy like we have it for programs: The circuits (programs) are the syntactical objects, which we have as finite words at our fingertips, whereas the Boolean functions (partial recursive functions) are the corresponding semantic objects far away in the mathematical sky[1]. Note that every Boolean function can be represented by a circuit, in fact it is represented by infinitely many circuits. The problem whether two circuits represent the same Boolean function is co-NP-complete.

As it was pointed out in the introduction one would like to have a theorem like: Every nontrivial semantic property of circuits is NP-hard. Unfortunately, there are counterexamples for this statement unless P=NP. First of all there is the counterexample consisting of the set of circuits with an odd number of satisfying assignments, this set is complete for $\oplus$P and not known to be NP-hard or co-NP-hard. So we have to replace in the statement above the class NP by some class which is contained in NP and $\oplus$P. The class UP would be a natural choice for that. But unfortunately there is an even stronger counterexample: the set of all circuits which evaluate to 0 on the all-0-assignment. This property of circuits is in fact a semantic property and it is nontrivial, but it can be checked with a polynomial-time algorithm. So we can only hope for an analogue of Rice's Theorem (stating UP-hardness) if we restrict ourselves to stronger semantic properties. We tried several approaches, for example by considering the equivalence relations like Boolean isomorphism presented in [1, 6]. Note that the above counterexample is also a counterexample under Boolean isomorphism. The restriction for which we could find the intended hardness result is the restriction to the counting properties which will be introduced in the next section.

3 Three Types of Counting Problems on Circuits

Given a Boolean function one can ask different question about the number of satisfying assignments: (a) what is the number of satisfying assignments? (b) what is the

[1]One might argue that here we have a different situation than in the case of partial recursive functions because Boolean functions are finite objects: one can represent an n-ary Boolean function just by the length-2^n 0-1-sequence of its values on the 2^n assignments. This in fact guarantees decidability of the usual semantic questions, but we are interested in finer questions: even if semantic properties are decidable, how difficult is it to decide them? And the difficulty for deciding the semantic questions (like satisfiability) stems basically from the fact that Boolean functions are not given as the length-2^n 0-1-sequence of the function values but in a *compressed* way, namely as circuits.

difference between the number of satisfying and non-satisfying assignments? (c) what is the share of the satisfying assignments compared with the total number of assignments? If the arity of the Boolean function is known then all question are equivalent. But if we do not fix the arity then the three questions are pairwise incomparable, i.e., given an answer to (a) we can not infer an answer to (b) and so on.

For a circuit c let $\#_0(c)$ and $\#_1(c)$ denote the number of non-satisfying assignments and satisfying assignments, resp., of the Boolean function represented by c. According to the different questions (a), (b) and (c) from above we will introduce the following three types of counting problems. Note that the answer to (a) is a natural number, the answer to (b) an integer and the answer to (c) a dyadic number in the interval $[0,1]$, we will denote this set by $\mathbf{D} = \{\frac{m}{2^n} \mid n, m \in \mathbf{N}, 0 \leq m \leq 2^n\}$.

Definition 2 (*a*) *Let A be a subset of* $\mathbf{N}$. *The* absolute counting problem for A, Absolute-Counting(A), *is the set of all circuits c such that $\#_1(c) \in A$.*

(*b*) *Let A be a subset of* $\mathbf{Z}$. *The* gap counting problem for A, Gap-Counting(A), *is the set of all circuits c such that $\#_1(c) - \#_0(c) \in A$.*

(*c*) *Let A be a subset of* $\mathbf{D}$. *The* relative counting problem for A, Relative-Counting(A), *is the set of all circuits c such that the relative number of accepting assignments is in A:*

$$c \in \text{Relative-Counting}(A) \quad \Leftrightarrow \quad \frac{\#_1(c)}{\#_0(c) + \#_1(c)} \in A.$$

Examples of absolute counting problems. The satisfiability problem for circuits, which we will denote here as SAT (though SAT traditionally refers to the satisfiability problem on CNF's), is by its definition an absolute counting problem, i.e. SAT = Absolute-Counting($\{1, 2, 3, \ldots\}$). Remember that SAT is NP-complete. Likewise, the set of unsatisfiable circuits equals Absolute-Counting($\{0\}$), this problem is co-NP-complete. Another example is the set of circuits with an odd number of satisfying assignments, by definition it equals Absolute-Counting($\{1, 3, 5, \ldots\}$), this problem is $\oplus$P-complete. Another example is the set 1-SAT consisting of the circuits with exactly one satisfying assignment, i.e. 1-SAT = Absolute-Counting($\{1\}$), the complexity class for which this problem is complete is usually called 1-NP.

Examples of gap counting problems. The set C$_=$SAT of circuits which have as many satisfying as non-satisfying assingments, is a gap counting problem: C$_=$SAT = Gap-Counting($\{0\}$). The set PSAT of circuits which have at least as many accepting as non-accepting assignments, is a gap counting problem: PSAT = Gap-Counting($\{0, 1, 2, 3, \ldots\}$). Remember that C$_=$SAT and PSAT are complete for the classes C$_=$P and PP, respectively. Another example is the set Gap-2-SAT consisting of the circuits with exactly two more satisfying than non-satisfying assignments, i.e. Gap-2-SAT = Gap-Counting($\{2\}$).

Examples of relative counting problems. The two gap counting problems PSAT and C$_=$SAT from above are also relative counting problems: C$_=$SAT = Relative-Counting($\{\frac{1}{2}\}$), and PSAT = Relative-Counting($\{x \in \mathbf{D}; x \geq \frac{1}{2}\}$). SAT = Relative-

Counting($\mathbf{D} - \{0\}$) is also a relative counting problem. The tautology problem equals Relative-Counting($\{1\}$).

Note that only relative counting has the following natural property. If we have a circuit $c(x_1, \ldots, x_m)$ and add some dummy variables $x_{m+1}, \ldots, x_n$ so that the whole new circuit $c'(x_1, \ldots, x_m, \ldots, x_n)$ represents a Boolean function on $n \geq m$ inputs then we have $c(x_1, \ldots, x_m) \in$ Relative-Counting(A) $\iff$ $c'(x_1, \ldots, x_m, \ldots, x_n) \in$ Relative-Counting(A). In a way one would consider the Boolean functions represented by c and c' to be "basically the same". So if we identify Boolean functions modulo independent variables then only relative counting respects this natural identification.

A *counting problem* in a general sense we define the following way. Let a sequence (A_n) be given for which A_n is a subset of $\{0, \ldots, 2^n\}$. The *counting problem for* (A_n) is the set of all circuits $c(x_1, \ldots, x_n)$ such that $\#_1(c) \in A_n$, see [12] for an analogous definition of (general) counting classes. In this way, absolute, gap and relative counting problems are counting problems. It is easy to give an example of a (general) counting problem which is nontrivial but in P, for example the set of all circuits with an odd arity (it is the counting class for the sequence $\emptyset, \{0,1,2\}, \emptyset, \{0,1,\ldots,8\}, \emptyset, \{0,1,\ldots,32\}, \emptyset, \ldots$).

It was already mentioned that the three types of counting problems from Definition 2 are incomparable as mathematical sets. But the question appears if they are comparable in terms of $\leq_m^p$-complexity. For example, the tautology problem, which is a relative counting problem, is provably not an absolute counting problem, nevertheless there is an absolute counting problem, namely the non-satisfiability problem, which has the same $\leq_m^p$-complexity (both are co-NP-complete). But even for this weaker form of comparison the three types of counting problems seem to be incomparable: PSAT and C$_=$SAT are gap and relative counting problems but do not seem to be $\leq_m^p$-equivalent to some absolute counting problem. SAT is an absolute and relative counting problem but does not seem to be $\leq_m^p$-equivalent to some gap counting problem. 1-SAT is an absolute counting problem but does not seem to be $\leq_m^p$-equivalent to some gap or relative counting problem. Gap-2-SAT is a gap counting problem but does not seem to be $\leq_m^p$-equivalent to some absolute or relative counting problem.

Remark. Analogously to the way we defined the three types of counting problems we can define three types of counting classes: for a given subset A of $\mathbf{N}$ ($\mathbf{Z}$, $\mathbf{D}$) the *absolute* (*gap, relative*) *counting class for* A consists of the languages L for which there is a polynomial-time nondeterministic machine M such that a word x is in L iff the number of accepting paths (the difference of the number of accepting paths and non-accepting paths, the share of the accepting paths compared with the total number of paths) of M on input x is in A. This definition of *gap countable classes* equals the definition of *nice gap definable classes* in Fenner, Fortnow & Kurtz [9]. As a special case of the main result in [5, 21] it follows that an absolute (gap, relative) counting problem is $\leq_m^p$-complete for the corresponding absolute (gap, relative) counting class. In other words, absolute (gap, relative) counting problems and the corresponding absolute (gap, relative) counting classes are just two sides of the same medal. Note that the $\leq_m^p$-comparability question we discussed above is therefore equivalent to the set comparability question of the three types of counting classes.

4 Some Rice-Style Theorems for Counting Problems

In this section we will state and prove the theorems which show UP-hardness of all nontrivial counting problems of the three types defined in the previous chapter. Remember that the class UP, which was defined first in [19], is the promise class consisting of the languages L such that there is a polynomial-time nondeterministic machine M such that on every input the machine M has at most one accepting path and an input x is in L if M running on input x has an accepting path. Such machines are called *unambiguous*. By definition UP is a subset of NP. A classical result tells that UP equals P if and only if one-way functions exist [11]. The primality problem is a typical example of a problem in UP not known to be in P [8]. The following theorem implies that any nontrivial absolute counting property of circuits is UP-hard.

Theorem 3 *Let A be any nonempty proper subset of* $\mathbf{N}$. *Then one of the following three classes is $\leq_m^p$-reducible to* Absolute-Counting(A)*:* NP, co-NP, UP $\oplus$ co-UP.

Proof. The proof distinguishes the following three cases.

First case: A has a maximum a. Then co-NP is $\leq_m^p$-reducible to Absolute-Counting(A): Let a language L in co-NP be given and let M be a machine for L in the sense that $x \in L$ iff no path of $M(x)$ is accepting. Let $m(x)$ denote the number of accepting paths of $M(x)$. Cook [7] established a method to construct in polynomial time a circuit $\mathtt{Cook}_x^M$ with inputs $y_1, \ldots, y_n$ (n depends on x and is bounded by a polynomial in the length of x) such that accepting computation paths of $M(x)$ and satisfying assignments of $\mathtt{Cook}_x^M(y_1, \ldots, y_n)$ correspond to each other. Therefore, $\mathtt{Cook}_x^M(y_1, \ldots, y_n)$ evaluates to 1 for exactly $m(x)$ assignments. Using a additional variables $z_1, \ldots, z_a$ the circuit given by the following specification evaluates exactly $a + m(x)$ assignments to 1:

$$d_{x,a}^M(y_1, \ldots, y_n, z_1, \ldots, z_a) = \begin{cases} \mathtt{Cook}_x^M(y_1, \ldots, y_n) & \text{if } z_1 + \ldots + z_a = 0; \\ 1 & \text{if } z_1 + \ldots + z_a = 1 \\ & \text{and } y_1 + \ldots + y_n = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Thus $d_{x,a}^M$ is in Absolute-Counting(A) iff $m(x) = 0$ iff $x \in L$. So the mapping $x \to d_{x,a}^M$ gives a $\leq_m^p$-reduction from L to Absolute-Counting(A).

Second case: $\overline{A}$ has a maximum b. Given a set L in NP recognized by the nondeterministic machine M, an analogous construction as in the previous case is used in order to obtain a circuit $d_{x,b}^M$ which evaluates exactly $b + m(x)$ assignments to 1. Now $x \in L$ iff $m(x) > 0$ iff $d_{x,b}^M \in$ Absolute-Counting(A). And so one obtains the desired $\leq_m^p$-reduction.

Third case: neither A nor $\overline{A}$ has a maximum. Then there is $a \in A$ with $a + 1 \notin A$ and $b \notin A$ with $b + 1 \in A$. For a language $L = 0L' \cup 1L''$ in UP $\oplus$ co-UP let M' and M'' be the unambiguous machines for L' and L'', respectively. The mapping which assigns to an input $0x$ the circuit $d_{x,a}^{M'}$ and to an input $1x$ the circuit $d_{x,b}^{M''}$ realizes the $\leq_m^p$-reduction from L to Absolute-Counting(A). □

Gupta [13] and Ogiwara & Hemachandra [15] introduced the class SPP as the promise class consisting of the languages L such that there is a polynomial-time nondeterministic machine M such that for every input x for the machine M either half of the computation paths are accepting of half of them plus 1 are accepting, and $x \in L$ if the second case is true. Note that the class SPP contains both UP and co-UP. Fenner, Fortnow & Kurtz [9] proved the following result which states that the class SPP is $\leq_m^p$-reducible to any gap counting problem.

Theorem 4 (Fenner, Fortnow & Kurtz 1994) *Let A be any nonempty proper subset of $\mathbf{Z}$. Then* SPP $\leq_m^p$ Gap-Counting(A).

Now that we have UP-hardness for absolute and gap counting problems we turn to relative counting problems and will show UP-hardness for the more powerful polynomial-time Turing reducibility. Note that the classes NP, co-NP, and SPP are above UP.

Theorem 5 *Let A be any nonempty proper subset of $\mathbf{D}$. Then* Relative-Counting(A) *is $\leq_m^p$-complete for* NP *or* co-NP*, or* SPP $\leq_T^p$ Relative-Counting(A).

Proof. First consider the special case that $A(p) = A(q)$ for all dyadic numbers p, q with $0 < p < q < 1$. In this special case, A is one of the following six sets: $\{0\}, \{1\}, \{0,1\}, \mathbf{D}-\{0\}, \mathbf{D}-\{1\}, \mathbf{D}-\{0,1\}$. It is easy to see that the relative counting problems for first three sets are co-NP-complete, for example, Relative-Counting($\{1\}$) is the tautology problem. Similarly the relative counting problems for the last three sets are NP-complete, for example, Relative-Counting($\mathbf{D} - \{0\}$) is the satisfiability problem.

So it remains the case where there are dyadic numbers p, q such that $0 < p < q < 1$ and $A(p) \neq A(q)$. It will be shown that in this case SPP can be $\leq_T^p$-reduced to Relative-Counting(A). Let M be a machine which witnesses that a language L is in SPP. Consider for an input x the circuit $\mathtt{Cook}_x^M(y_1, \ldots, y_n)$ defined in the proof of Theorem 3. Recall that if $x \in L$ then $\mathtt{Cook}_x^M$ evaluates $2^{n-1} + 1$ assignments to 1 and if $x \notin L$ then $\mathtt{Cook}_x^M$ evaluates 2^{n-1} assignments to 1. Furthermore there is an m such that $2^{-m} \leq p < q \leq 1 - 2^{-m}$. Now the Turing-reduction works as follows:

- Search for p', q' with $p \leq p' < q' \leq q$, $q' - p' = 2^{-m-n}$ and $A(p') \neq A(q')$.
  A query to A can be translated into a query to Relative-Counting(A) as follows: Let $r = 0.a_1 \ldots a_k < 1$ be a dyadic number and let c_r denote the circuit which assigns to $(y_1, \ldots, y_k)$ the value 1 iff $0.y_1 \ldots y_k < 0.a_0 \ldots a_k$. r is in A iff $c_r \in$ Relative-Counting(A).
  Using this mechanism it is possible to find p' and q' with interval search: starting with $p = p'$ and $q = q'$ one takes that $m+n$-bit dyadic number r which is nearest to $\frac{p'+q'}{2}$ and finds out whether $A(p') = A(r)$ or $A(q') = A(r)$. In the first case, p' is replaced by r, in the second, q' is replaced by r. This search is continued until the difference between p' and q' is 2^{-m-n}. Note that $A(p) = A(p') \neq A(q) = A(q')$.

- Now a circuit d_x is computed with Relative-Counting$(A)(d_x) = A(q')$ for $x \in L$ and Relative-Counting$(A)(d_x) = A(p')$ for $x \notin L$. Let $z = 0.z_1 \ldots z_{m+n}$ be the dyadic number determined via the binary representation of the variables.

$$d_x(z_1, \ldots, z_{m+n}) = \begin{cases} \mathtt{Cook}_x^M(z_{m+1}, \ldots, z_{m+n}) & \text{if } z < 2^{-m}; \\ 1 & \text{if } 2^{-m} \leq z < p' + 2^{-m-1}; \\ 0 & \text{if } p' + 2^{-m-1} \leq z. \end{cases}$$

  So the relative number of accepting assignments is the sum of the $p' - 2^{-m-1}$ hard wired assignments from the second line of the case-distinction plus 2^{-m-1} $(2^{-m-1} + 2^{-m-n})$ from the circuit c_x in the case of $x \notin L$ $(x \in L)$. Then the whole relative number is p' for $x \notin L$ and q' for $x \in L$. It follows that $x \in L$ iff Relative-Counting$(A)(d_x) = A(q')$. So the last query whether d_x is in Relative-Counting(A) completes the decision procedure for L.

In short words: the first part of the construction uses the fact that $A \leq_m^p$ Relative-Counting(A) in order to search sufficiently close dyadic numbers p' and q' between p and q such that $A(p') \neq A(q)$. The next step produces a circuit d_x whose relative number of satisfying assignments is p' for $x \notin L$ and q' for $x \in L$. So $L(x)$ can be computed with $m + n + 1$ queries to Relative-Counting(A). Therefore, in this second case, Relative-Counting(A) is SPP-hard with respect to polynomial-time Turing reductions. □

The preceding three Theorems 3, 4, 5 can be summarized the following way.

Conclusion 6 *Any nontrivial absolute (gap, relative) counting property of circuits is* UP*-hard with respect to polynomial-time Turing reducibility.*

So we also obtain the following conclusion.

Conclusion 7 *A nontrivial absolute (gap, relative) counting problem on circuits is not solvable in polynomial-time unless* P=UP.

5 Extensions and Limitations of the Main Results

In this section first the result is extended to randomized reductions and computations. After that it is shown that no nontrivial absolute (gap, relative) counting problem on circuits is approximable unless P=UP. Furthermore it is pointed out that it is unlikely that Theorem 5 holds with polynomial-time many-one-reduction in place of the polynomial-time Turing reduction. In the last part a perfect analogue of Rice's Theorem is given for the world of existentially quantified circuits.

5.1 Randomized Computations

Valiant and Vazarani [20] showed, that using randomized reductions, detecting unique solutions is as hard as solving the satisfiability problem. In particular they showed that every algorithm f which satisfies the following specification also already allows to solve the satisfiability problem in a randomized context:

$$\begin{aligned} \text{circuit } x \text{ has no solution} &\Rightarrow f(x) = 0; \\ \text{circuit } x \text{ has exactly one solution} &\Rightarrow f(x) = 1. \end{aligned}$$

The algorithms to reduce UP to the counting problems in Theorems 3, 4, 5 satisfy these requirements. So they allow to decide the set SAT $= \{$ circuits $x : x$ has a solution$\}$ via a nondeterministic machine which has no accepting path for $x \notin$ SAT and which has more accepting than rejecting paths for $x \in$ SAT. Thus the following holds for all three counting problems, in particular for relative counting.

Theorem 8 *If* $B =$ Relative-Counting(A) *is not trivial then* $\mathrm{NP} \subseteq \mathrm{RP}^B$.

For absolute and gap counting, the result can be improved by showing that SAT is randomized polynomial-time reducible to B or to $\overline{B}$. Valiant and Vazirani [20] defined that a randomized polynomial-time reduction from some set A to another set B is given via a machine M which computes for every x and path p a circuit $M(x,p)$ such that

$$\begin{aligned} x \in A &\Rightarrow M(x,p) \in B \text{ for at least } \frac{n(x)}{q(\text{length of } x)} \text{ paths } p; \\ x \notin A &\Rightarrow M(x,p) \notin B \text{ for all paths } p. \end{aligned}$$

where q is a suitable polynomial and $n(x)$ is the number of computational paths of M. Valiant and Vazirani [20, Theorem 1.1] constructed a randomized polynomial-time reduction from SAT to USAT_Q where

$$\mathrm{USAT}_Q(x) = \begin{cases} 0 & \text{if } x \text{ has no solution;} \\ 1 & \text{if } x \text{ has one solution;} \\ Q(x) & \text{if } x \text{ has at least two solutions;} \end{cases}$$

and where the reduction is independent of the values $Q(x)$ for all x. The result is a direct combination of this construction and the constructions for $\leq_m^p$-reducing UP to B or $\overline{B}$ in Theorems 3 and 4 which indeed are $\leq_m^p$-reductions of some suitable set USAT_Q to B or $\overline{B}$, respectively.

Theorem 9 SAT *or its complement is randomized polynomial-time reducible to any nontrivial absolute and gap counting problem.*

5.2 Approximable Sets

A set A is approximable [4] iff there is a constant j and an algorithm which computes for each input $x_1, \ldots, x_j$ in polynomial time j bits $y_1, \ldots, y_j$ such that $A(x_h) = y_h$ for some h. Beigel [2, 3] analyzed the notion of approximable sets and showed that no NP-hard set is approximable unless P = UP. This result can be transferred to the following theorem which is an extension of Conclusion 7.

Theorem 10 *If* $\mathrm{P} \neq \mathrm{UP}$ *then no nontrivial absolute* (*gap, relative*) *counting problem is approximable.*

Proof. The proofs are all direct combinations of Beigel's techniques with those to show the UP-hardness of these sets. Thus we restrict ourselves to show the most involved case of relative counting sets.

If a problem is NP-complete or co-NP-complete then it is not approximable under the hypothesis P $\neq$ UP [2, 3]. So one has only to adapt the main case in the proof of Theorem 5.

So let L be any language in UP, note that UP $\subseteq$ SPP. The first part of the Turing reduction from L to Relative-Counting(A) is exactly the same as in the proof of Theorem 5. In the second part it starts to differ at the definition of the circuit d_x. Based on the definition of d_x, n variants $d_{n,i}$ are defined via fixing one variable to 1 in the circuit $\texttt{Cook}_x^M$:

$$d_{x,i}(z_1,\ldots,z_{m+n}) = \begin{cases} \texttt{Cook}_x^M(z_{m+1},\ldots,z_{m+n}) & \text{if } z < 2^{-m} \text{ and } z_{m+i} = 1; \\ 0 & \text{if } z < 2^{-m} \text{ and } z_{m+i} = 0 \\ & \text{or } p' + 2^{-m-1} \le z; \\ 1 & \text{if } 2^{-m} \le z < p' + 2^{-m-1}. \end{cases}$$

Recall that for $x \in L$ there is exactly one satisfying assignment $(a_1,\ldots,a_n)$ and for $x \notin L$ no one. So the relative number of the accepted assignments of $d_{x,i}$ is $q' = p' + 2^{-m-n}$ if $x \in L \wedge a_i = 1$ and p' otherwise.

As Beigel [3, Theorem 9] pointed out, there is an algorithm which computes for input $(d_{x,1},\ldots,d_{x,n})$ in polynomial time $O(n^j)$ n-bit-vectors v such that one of these vectors is the characteristic function of A on the input-vector. So for each such $v = (v_1,\ldots,v_n)$ one computes the assignment $(a_1,\ldots,a_n)$ given via $a_i = 1$ for $v_i = A(q')$ and $a_i = 0$ for $v_i = A(p')$. If $\texttt{Cook}_x^M$ has a satisfying assignment, then one of these $(a_1,\ldots,a_n)$ must be one. By evaluating $\texttt{Cook}_x^M$ on these assignments it is found out whether $x \in L$ or $x \notin L$. Thus if Relative-Counting(A) is approximable then L is in P and P = UP. □

5.3 Relative Counting and BPP

For relative counting, we could not state UP-hardness in terms of $\le_m^p$-reducibility. Under the hypothesis that UP is not contained in the class BPP the following Theorem 11 gives nontrivial relative counting problems which are not UP-hard with respect to many-one reduction.

Recall that BPP, first defined in [10], is the class of all languages L such that there is an $\epsilon > 0$ and a polynomial-time nondeterministic machine M such that for all inputs x the share of accepting paths (all of them must have the same length) is either less than $\frac{1}{2} - \epsilon$ or greater than $\frac{1}{2} + \epsilon$, and x is accepted in the second case.

Theorem 11 *There is a set $A \subseteq \mathbf{D}$ such that for every computable set L the following equivalence holds: $L \le_m^p$ Relative-Counting(A) iff $L \in$ BPP.*

Proof. There are uncountably many reals r between 0 and 1 and for each such real r, the set $A = \{q \in \mathbf{D} : q < r\}$ is unique. Thus, using a natural representation of dyadic numbers by words, there is a set A of this form which is not enumerable. So fix such a set A and r.

($\Rightarrow$) : Let L be a computable set which is $\leq_m^p$-reducible to Relative-Counting(A) via a function f. Let $m(x)$ denote for each x the relative number of satisfying assingments of the circuit $f(x)$, $m(x)$ can be computed from $f(x)$ using exponential time. Since $m(x)$ is in A iff x is in L, the set

$$B = \{q \in \mathbf{D} : (\exists x \in L)\,[q \leq m(x)]\}$$

is an enumerable subset of A. Since A is not enumerable, the supremum of B must be below that of A: $\sup(B) < r$. There is a dyadic number s strictly between these two numbers. So there is some $\epsilon > 0$ such that $\sup(B) < s - 2\epsilon$ and $r > s + 2\epsilon$. Let $n(x)$ be the number of inputs of circuit $f(x)$. Now the following machine M witnesses that L is in BPP.

$$M(x)(y_0, \ldots, y_{n(x)}) = \begin{cases} 1 & \text{if } y_0 = 0 \text{ and } f(x) \text{ evaluates } (y_1, \ldots, y_{n(x)}) \text{ to } 1 \\ & \text{or } y_0 = 1 \text{ and } 0.y_1 \ldots y_{n(x)} > s; \\ 0 & \text{otherwise.} \end{cases}$$

The relative number $k(x)$ of the accepting assignments is the sum of two numbers: the number $\frac{m(x)}{2}$ from the simulation of $f(x)$ and the number $\frac{1-s}{2}$ from the hard-wired paths with $y_0 = 1$. So if $x \in L$ then $m(x) \in A$, $m(a) \leq s - 2\epsilon$ and $k(x) \leq \frac{1}{2} - \epsilon$; if $x \notin L$ then $m(x) \notin A$, $m(x) \geq s + 2\epsilon$ and $k(x) \geq \frac{1}{2} + \epsilon$. Since the constant ϵ does not depend on x and $n(x)$, M witnesses that L is in BPP.

($\Leftarrow$) : Now let $L \in$ BPP. There is a real number ϵ with $|r - \frac{1}{2}| < \epsilon < \frac{1}{2}$. For L there is now a BPP-machine M which works with this ϵ, see for example [16, Chapter 11]. Let M have path length $n(x)$ on input x. Now one assigns to each x the circuit $\neg\mathtt{Cook}_x^M(y_1, \ldots, y_{n(x)})$. The relative number of the satisfying assignments for each circuit $\mathtt{Cook}_x^M$ is above $\frac{1}{2} + \epsilon > r$ for $x \notin L$ and below $\frac{1}{2} - \epsilon < r$ for $x \in L$. So the mapping $x \rightarrow \neg\mathtt{Cook}_x^M$ is an $\leq_m^p$-reduction from L to Relative-Counting(A). $\square$

Remark. The classes PP, $C_=P$ and $\mathrm{Mod}_k\mathrm{P}$ could be defined as the class of all languages which are $\leq_m^p$-reducible to some suitable relative counting problem. For example, $C_=P$ is the set of languages $\leq_m^p$-reducible to Relative-Counting($\{\frac{1}{2}\}$). In addition to the above mentioned limitation, Theorem 11 shows that the class BPP can be defined similarly within the class of all computable sets.

5.4 A Perfect Analogue of Rice's Theorem for Existentially Quantified Circuits

It was mentioned before that the problem whether two programs compute the same partial recursive function is complete for Π_2 in the Arithmetical Hierarchy, whereas the problem whether two circuits compute the same Boolean function is complete for co-NP $= \Pi_1^p$ in the Polynomial-Time Hierarchy. So the difficulty of the two equivalence problems is on different levels in the respective hierarchies. Therefore, we have been looking for some representation of Boolean functions such that the equivalence problem is complete for Π_2^p. Such a representation is given by *existentially quantified circuits* which are defined to be circuits in which some (but not necessarily all) variables are existentially quantified (in prenex normal form). Such an existentially quantified Boolean circuit with n free variables defines a Boolean function with arity n the following obvious way: for an assignment a to the free variables all possible assignments to the quantified variables are checked and if and only if one of them lets the circuit evaluate to 1 the final value for the assignment a is 1. For example, the existentially quantified circuit $\exists z((x \vee z) \wedge (y \vee \neg z))$ describes the same Boolean function as the circuit $x \vee y$. It is easy to see that it is a Π_2^p-complete problem to check whether two given existentially quantified circuits describe the same Boolean function. Note that existentially quantified circuits are the nonuniform counterpart of polynomial-time nondeterministic computation, i.e. polynomial-size existentially quantified circuits equals NP/poly, see [16] for the notations.

We will prove an analogue of Rice's Theorem for existentially quantified circuits, even the proof is analogous.

Theorem 12 *Let A be a set of Boolean functions which not only depends on arity. Then SAT or co-SAT is $\leq_m^p$-reducible to the following problem: Given an existentially quantified Boolean circuit, does the Boolean function described by it belong to A?*

Proof. Because A does not only depends on arity there is an arity n such that one Boolean function of that arity is in A and another one is not in A. Assume as the first case that A does not contain the n-ary constant-0-function and let $f = f(x_1, \ldots, x_n)$ be a circuit which describes an n-ary Boolean function in A. We give an $\leq_m^p$-reduction of SAT to the set of existentially quantified Boolean circuits which describe a Boolean function in A. Let a circuit $c = c(x_1, \ldots, x_m)$ be given. Construct the existentially quantified circuit $e = \exists x_1 \ldots \exists x_n (c(x_1, \ldots, x_m) \wedge f(y_1, \ldots, y_n))$. If c is satisfiable then e describes the same function as f; otherwise e describes the n-ary constant 0-function.

If A does contain the n-ary constant-0-function then we can in an analogous fashion reduce co-SAT to the problem in question. $\square$

Remark. The restriction for A of being dependent on the arity is necessary because for example for the set of Boolean functions with odd arity the decision problem in question is polynomial-time computable.

If a problem is not decidable one still can have hope that it is recursively enumerable. The following extension of Rice's Theorem has a criterion for semantic properties of programs to be not even recursively enumerable.

Theorem 13 (Rice 1953) *Let A be a set of partial recursive functions. If there exist two functions f, g such that f is contained in A, g is not contained in A and $f \leq g$ (i.e. if $f(n)$ terminates then $f(n) = g(n)$) then the following problem is not recursively enumerable: Given a program p, does it compute a function from A?*

We can state the analogue of the above theorem. Also the proof is analogous.

Theorem 14 *Let A be a set of Boolean functions with two n-ary Boolean functions F and G such that F is in A, G is not in A and $F(a_1, \ldots, a_n) \leq G(a_1, \ldots, a_n)$ for all assignments $(a_1, \ldots, a_n)$. Then* co-SAT *is $\leq_m^p$-reducible to the following problem: Given an existentially quantified Boolean circuit, does the Boolean function described by it belong to A?*

Proof. Let $f(x_1,,\dots,x_n)$ be a circuit for F and let $g(x_1,,\dots,x_n)$ be a circuit for G. We give a $\leq_m^p$-reduction of co-SAT to the set of existentially quantified Boolean circuits which describe a Boolean function in A. Let a circuit $c = c(x_1,\dots,x_m)$ be given. Construct the existentially quantified circuit $e = \exists x_1 \dots \exists x_n (f(y_1,,\dots,y_n) \vee (c(x_1,\dots,x_m) \wedge g(y_1,\dots,y_n)))$. If c is not satisfiable then e describes the Boolean function F, and otherwise e describes the Boolean function G. $\square$

6 Conclusion

We presented some results in Complexity Theory which have similarities with Rice's Theorem. Our main result of that kind is that all nontrivial absolute, gap, and relative counting properties of circuits are UP-hard with respect to polynomial-time Turing reductions.

We consider the presented results as first steps of a "Rice program" in Complexity Theory: Proving lower bounds for problems which are defined by semantic properties of subrecursive objects (like circuits).

Acknowledgements

The authors are grateful for discussions with Klaus Ambos-Spies and Andre Nies. Furthermore the authors appreciate the comments by Lance Fortnow, Ulrich Hertrampf, Heribert Vollmer, Klaus Wagner, and Gerd Wechsung after a talk at Dagstuhl.

References

[1] M. Agrawal, T. Thierauf. *The Boolean isomorphism problem*, 37th Symposium on Foundations of Computer Science (FOCS), 1996, pp. 422–430.

[2] R. Beigel. *A structural theorem that depends quantitatively on the complexity of* SAT, Proc. 2th Annual IEEE Conference on Computational Complexity, 1987, pp. 28–32.

[3] R. Beigel. NP-*hard sets are* P-*superterse unless* R = NP, Technical Report 88-04, John Hopkins University, Baltimore, 1988.

[4] R. Beigel, M. Kummer, F. Stephan. *Approximable Sets*, Information and Computation, **120**, 1995, pp. 304-314.

[5] B. Borchert, A. Lozano. *Succinct circuit representations and leaf language classes are basically the same concept*, Information Processing Letters **58**, 1996, pp. 211–215.

[6] B. Borchert, D. Ranjan, F. Stephan. *The Computational Complexity of some Classical Equivalence Relations on Boolean Functions*, ECCC Report TR96-033, 1996.

[7] S. A. Cook. *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on the Theory of Computing (STOC), 1971, pp. 151–158.

[8] M. R. Fellows, N. Koblitz. *Self-witnessing polynomial-time complexity and prime factorization*, Proc. 7th Annual IEEE Conference on Computational Complexity, 1992, pp. 107–110

[9] S. A. Fenner, L. J. Fortnow, S. A. Kurtz. *Gap-definable counting Classes*, Journal of Computer and Systems Sciences **48**, 1994, pp. 116–148.

[10] J. Gill. *Computational complexity of probabilistic Turing machines*, SIAM Journal of Computing **6**, 1977, pp. 675–695.

[11] J. Grollmann, A. Selman. *Complexity measures for public-key cryptosystems*, SIAM Journal on Computing **17**, pp. 309–335

[12] T. Gundermann, N. A. Nasser, G. Wechsung. *A survey on counting classes*, Proc. 5th Annual IEEE Conference on Computational Complexity, 1990, pp. 140–153.

[13] S. Gupta. *The power of witness reduction*, Proc. 6th Annual IEEE Conference on Computational Complexity, 1991, pp. 43–59

[14] Dexter Kozen. *Indexings of subrecursive classes*, Theoretical Computer Science **11**, 1980, pp. 277–301.

[15] M. Ogiwara, L. A. Hemachandra. *A complexity theory of feasible closure properties*, Proc. 6th Annual IEEE Conference on Computational Complexity, 1991, pp. 16–29

[16] Ch. Papadimitriou. *Computational Complexity*, Addison Wesley, 1994.

[17] H. G. Rice. *Classes of recursively enumerable sets and their decision problems*, Trans. Amer. Math. Soc. **74**, 1953, pp. 358–366.

[18] H. G. Rice. *On completely recursively enumerable classes and their key arrays*, J. Symbolic Logic **21**, 1956, pp. 304–341.

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

[20] L. G. Valiant, V. V. Vazarani. *NP is as easy as detecting unique solutions*, Theoretical Computer Science **47**, 1986, pp. 85–93.

[21] H. Veith. *Succinct representations, leaf languages and projection reductions*, Proc. 11th Annual IEEE Conference on Computational Complexity, 1996, pp. 118–126.