THE TOTP FUNCTION CLASS

Andreas Nikolas Goebel

National Technical University of Athens
9, Iroon Plytexneiou, Zografou, 15780, Athens, Greece, tel.:00302107721644
E-mail: agob@corelab.ntua.gr

Abstract

Valiant has observed that there are problems in P with their counting version in #P . Pagourtzis and Zachos defined and studied 2 new classes #PE and TotP, containing #P problems with decision version in P and functions that represent all the computation paths of a poly-NDTM respectively. This paper is mainly a survey on their results about this new classes. Also we prove that TotP shares the same closure properties with #P . Furthermore it is shown that TotP is exactly the Karp closure of self-reducible function of #PE . Several interesting problems are shown to belong in TotP.

1 Introduction

Valiant has introduced the class $\#P$, which counts the accepting computation paths of a poly-NDTM. As we have seen it contains counting versions of known **NP** problems, like **SAT**, **HamiltonCycles** etc. These problems are considered hard to decide (since they are **NP**-complete). What is remarkable is that there exist other $\#P$ problems with their decision version in **P**, like **PerfectMatchings** and **DNF-SAT**.

In this paper, mainly based on the work of Pagourtzis and Zachos [5] we shall investigate the complexity of such ``hard-to-count-easy-to-decide'' problems. We notice that these problems are $\#P$-complete only under Cook reductions and not Karp reductions, leading us to an important difference between these two reductions. Most of the known classes that that can be defined via poly-NDTM's, including $\#P$, are closed under Karp reductions. On the other hand the class $FP^{\#P[1]}$, by definition, Cook[1]-reduces to $\#P$, while, as Toda and Watanabe have proven in [10], it is not contained in $\#P$ unless **PH** collapses. This means that $\#P$ is not closed under Cook[1] reduction unless **PH** collapses, which is considered unlikely. Therefore, we can say that Cook reductions blur structural differences between complexity classes of functions.

2 Definitions

The computational model is non-deterministic Turing machine. When counting the paths of an non-deterministic TM, we call it counting Turing machine (CTM).

2.1 Reductions

Definition 2.1 We say that a problem (or function) A reduces to a problem B by Karp reduction and we denote $A \leq_m^p B$, if and only if there exists a polynomial-time computable function f such that $\forall x (x \in A \Leftrightarrow f(x) \in B)$. Symbolically we have:

$$A \leq_m^p B : \exists f \in FP, \forall x (x \in A \Leftrightarrow f(x) \in B),$$

were FP is the class of all polynomial-time computable functions.

Definition 2.2 We say that a problem A (or function) reduces to a problem B by Cook (Turing) reduction and we denote $A \leq_T^p B$, if A can be computed by a deterministic TM within polynomial time with the use of an oracle for B.

This means that the TM may query the oracle as many times and for any instance of B and get an instant reply. Additionally: $A \leq_m^p B \Rightarrow A \leq_T^p B$

By P^A we denote the class of problems that can be computed by a deterministic TM within polynomial time and the use of an oracle for A, or else:

$$P^A = \{L \mid L \leq_T^p A\}$$

A widely used, special case of the Cook reduction for functions is the following:

Definition 2.3 A function f (or problem) reduces to a function g by Cook[1] reduction and we denote $A \leq_{1-T}^p B$ if and only if $\exists h_1, h_2 \in FP \, such \quad that \, \forall x \quad f(x) = h_1(x, g(h_2(x)))$.

Definition 2.4 We say that a class $\mathcal{C}$ is closed under a reduction $\leq$ if:

$$A \in \mathcal{C} and B \leq A \Rightarrow B \in \mathcal{C}$$

Proposition 2.5 If two classes $\mathcal{C}$ and $\mathcal{C}'$ are both closed under reductions and there is a problem A which is complete for both classes, then $\mathcal{C} = \mathcal{C}'$

• the function classes $\mathcal{F}$ and $\mathcal{G}$ are Cook[1]-interreducible (or indistinguishable under Cook[1] reductions)

if and only if $\mathcal{F} \subseteq FP^{\mathcal{G}[1]}$ and $\mathcal{G} \subseteq FP^{\mathcal{F}[1]}$

if and only if $FP^{\mathcal{F}[1]} = FP^{\mathcal{G}[1]}$

if and only if $P^{\mathcal{F}[1]} = P^{\mathcal{G}[1]}$

2.2 Classes

For each function $f \in \Sigma^* \to \mathbb{N}$ we define a related language:

$$L_f = x : f(x) > 0$$

For function problems this language is the decision version. In particular if a function f corresponds to the counting version of a search problem then L_f corresponds to the existence version.

Definition 2.6 $\#PE$ *is the class that contains functions of* $\#P$ *whose related language is in* ***P***.

In other words $\#PE$ contains all the ``hard-to-count-easy-to-decide'' problems, like $\#PerfectMatchings$ and $\#DNF-SAT$. We shall also see that $\#PE$ cannot contain $\#SAT$ unless $P = NP$.

The following function associated with a poly-NDTM M, will help us define the next class that we will investigate in this section:

$$tot_M(x) = (\#\ paths \quad of \quad M \quad on \quad input \quad x) - 1$$

The "minus one" in the definition of tot_M was introduced so that the function can take a zero value.

Definition 2.7 $TotP = \{tot_M : M \quad is \quad a \quad poly-NDTM\}$

3 TotP, $\#PE$ and $\#P$

For the proofs of the following propositions the reader may refer to [5]

Proposition 3.1 $\#P \subseteq TotP - FP$

where the minus sign refers to elementwise subtraction.

Now we shall see the inclusions amongst the function classes we have so far defined.

Proposition 3.2 $FP \subseteq TotP \subseteq \#PE \subseteq \#P$. *This inclusions are proper unless* $P = NP$

The classes **totP**, $\#PE$ and $\#P$ are all closed under Karp reduction so we have the following:

Corollary 3.3 ***totP***, $\#PE$ *and* $\#P$ *are not Karp equivalent unless* ***P=NP***.

A function class $\mathcal{F}$ with $f \in \mathcal{F}$ is called polynomially bounded if there exists a polynomial p such that for all x, $|f(x)| \leq p(|x|)$. We will now see that for such a function class $\mathcal{F}$ it holds $\mathcal{F} - FP \subseteq FP^{\mathcal{F}}$. Let $f \in \mathcal{F} - FP$ this mean that there exist $h_1 \in \mathcal{F}$ and $h_2 \in FP$ such that for all x, $f(x) = h_1(x) - h_2(x)$. Let M be a DTM that can query the oracle $\mathcal{F}$ once. M can calculate h_1 in polynomial time and with one oracle query it get the value of h_2, which is of polynomially bounded length as in $\mathcal{F}$. So we have proven that $f \in FP^{\mathcal{F}[1]}$. Now we can see that **totP**, $\#PE$ and $\#P$ are Cook[1]-interreducible.

Corollary 3.4 $FP^{TotP[1]} = FP^{\#PE[1]} = FP^{\#P[1]}$

Proof. $FP^{TotP[1]} \subseteq FP^{\#PE[1]} \subseteq FP^{\#P[1]} \subseteq FP^{FP^{(TotP-FP)[1]}}$. But having an **FP** oracle on a poly-DTM M doesn't increase M's computational power so $FP^{FP} = FP$. Hence $FP^{FP^{(TotP-FP)[1]}} \subseteq FP^{TotP[1]}$

By combining this latter corollary with Toda's result [9] we get the following:

$$PH \subseteq P^{TotP[1]} = P^{\#PE[1]}$$

4 Properties of TotP

In this section we will show the properties of **TotP**. We shall begin with the closure properties, which are also shared with $\#P$.

4.1 Closure Properties

Proposition 4.1 ***TotP*** *has the following closure properties:*

1. $TotP \circ FP = TotP$
2. If $f \in TotP$ and p is a polynomial, then the function

$$g(x) = \sum_{|y| \le p(|x|)} f(\langle x, y \rangle)$$

is in **TotP**

3. If $f \in TotP$ and p is a polynomial, then the function

$$g(x) = \prod_{|y| \le p(|x|)} f(\langle x, y \rangle)$$

is in **TotP**

4. If $f \in TotP$, $k \in FP$, and $k(x)$ is bounded by a polynomial in $|x|$, then the function

$$g(x) = \binom{f(x)}{k(x)}$$

is in **TotP**

Proof.

1. Given a poly-CTM M and a function $g \in FP$, we construct the poly-CTM N that simulates $M(g(x))$ for all $x \in \Sigma^*$. Obviously $tot_N = tot_M \circ g$.
2. Let $f = tot_M$ for a poly-CTM M. We can construct a poly-CTM N that first guesses a y of length $|y| \le p(|x|)$. For each y guessed it branches ($dupl_{p(|x|)}$) and then simulates M on input $\langle x, y \rangle$. Obviously $g = tot_N$.
3. Let $f = tot_M$ for a poly-CTM M. Then the poly-CTm N calculates $p(|x|)$ deterministically and then simulates $M(\langle x,1 \rangle)$. On the end of each computation path N then simulates $M(\langle x,2 \rangle)$, then $M(\langle x,3 \rangle)$, etc until $M(\langle x, p(|x|) \rangle)$ is simulated. We can see that $g = tot_N$
4. As we stated earlier we can lexicographically order the paths of a CTM. Let $f = tot_M$ for a poly-CTM M. We shall construct the CTM N as follows: First calculate $p(|x|)$ deterministically; then simulate M and on each path simulate M but branch if and only if the paths are in strictly increasing lexicographical order. The branching prevention can be achieved by changing the transition relation of a non-deterministic machine by reducing the number of the legal next actions. The reader with some elementary combinatorics knowledge may verify that $g = tot_N$.

4.2 Self-reducibility

We will formalize the notion of self-reducibility in a different way from Ko's self-reducibility.

Definition 4.2 A function $f : \Sigma^* \to \mathbb{N}$ is called poly-time self-reducible if there exist polynomials r and q and polynomial time computable functions $h : \Sigma^* \to \mathbb{N}$, $g : \Sigma^* \to \mathbb{N}$ and $t : \Sigma^* \to \mathbb{N}$ such that for all $x \in \Sigma^*$

1. $f(x) = t(x) + \sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i))$, that is, f can be processed recursively by reducing x to $h(x,i)$, $(0 \le i \le r(|x|))$, and
2. the recursion terminates after at most polynomial depth (that is, the value of f on instance $h(\ldots h(h(x,i_1),i_2)\ldots,i_{q(|x|)})$ can be computed deterministically in polynomial time).

4.3 Main Theorem

Let $\#PE_{SR}$ denote the class of all self reducible functions of $\#PE$. Now we are ready to prove the main result for the **TotP** class.

Theorem 4.3 **TotP** is exactly the closure under Karp reductions of $\#PE_{SR}$

The proof is includen in [5]. In this paper we will only show the soundess of the algorithm by induction in the depth of recursion of f(x). If $f(x) = 0$ then M stops, hence we have 1 computational leaf and

$tot_M(x) = 1 = 0 + 1 = f(x) + 1$. If $f(x) > 0$, then a non-deterministic choice is made among stopping and calling $GenTree_f(x)$. Therefore the computation has $l + 1$ leaves where l is the computation tree of $GenTree_f(x)$. We will prove, by induction on the recursion depth, that for any polynomial-time self reducible function f with $f(x) > 0$ the computation tree of $GenTree_f(x)$ has exactly $f(x)$ leaves.

- If f can be computed directly in polynomial time then $GenTree_f(x)$ spawns $f(x)$ non-deterministic branches and stops at each one of them. Thus the base of the induction holds.
- Assume now that the claim is true for all functions that can be computed with recursion depth at most k, i.e. they can be computed deterministically in polynomial time on the instance $h(\ldots h(h(x,i_1),i_2)\ldots,i_k)$ (it might have less recursion depth). Consider now a function f that requires $k+1$ recursive reductions to be calculated. $GenTree_f(x)$ first computes the functions $g(x,i), h(x,i)$ and $t(x)$ for all i, $0 \le i \le r(|x|)$. For each of the $r(|x|)+1$ different i's, $GenTree_f(x)$ creates a different subtree with $g(x,i)$ branches and at each one it computes $GenTree_f(h(x,i))$. So we have $\sum_{i=0}^{r(|x|)} g(x,i)$ different branches, and at each one the computations continue with $GenTree_f(h(x,i))$. Each $f(h(x,i))$ requires k recursive reductions to be calculated. Due to the induction hypothesis, each $GenTree_f(h(x,i))$ will have exactly $f(h(x,i))$ computation leaves. So far the $r(|x|)+1$ computation subtrees of $GenTree_f(x)$ lead to $\sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i))$ computation leaves. If we add the $t(x)$ computation leaves of the last nondeterministic branch of $GenTree_f(x)$ to the latter, it is proven that $GenTree_f(x)$ has exactly $t(x) + \sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i)) = f(x)$ computation leaves. Note that if any of the functions used above has 0 value, $GenTree_f(x)$ adds no computation paths.

From the definition of polynomial-time self reducible functions, the recursion depth is polynomial on $|x|$, hence each computation path requires at most polynomial time (definition 4.2, 4.2).

5 TotP Complete Problems Under Cook[1] Reductions

We will end by enumerating some **TotP**-complete problems. Once again the reader may refer to [5] for the proofs.

Proposition 5.1 The following problems are **TotP**-complete under Cook[1] reduction:

1. $\#PerfectMatchings$

Given a bipartite graph, count the number of perfect matchings.

2. **Permanent**

Given a $(0,1)$-matrix calculate its permanent.

3. $\#DNF-SAT$

Given a boolean formula in disjunctive normal form, count the number of its satisfying assignments.

4. $\#Non-Cliques$

Given a graph G and a positive integer k, count the number of size-k subgraphs of G that are not cliques.

5. $\#NonIndependentSets$

Given a graph G and a positive integer k, count the number of size-k subgraphs of G that are not independent sets.

6 Conclusions

So we have defined a finer distinction within the class $\#P$, the classes **TotP** and $\#PE$. We have shown several interesting and natural problems that are contained in these classes. They have been shown to be Cook[1]-complete for **TotP**. The next step is to show that they are Karp-complete for **TotP**. If that is not possible, then define a new class under which the problems will be Karp-complete.

We have also shown that this class of functions, **TotP**, shares the same closure properties that are known for $\#P$, although their computational trees have been proven to be different.

Last we have proven that **TotP** doesn't contain $\#PE$ problems with trivial related language like $\#SAT_{+1}$. We conjecture that all $\#PE$ problems that do not have a trivial related language are in **TotP**.

7 Acknowledgments

I would like to thank my professors Stathis Zachos and Aris Pagourtzis, and also the corelab students: Evangelos Babas, Aris Tentes, George Pierakos, Antonis Achileos and Georgia Kaouri for helpful discussions and observations.

References:

[1] Arnaud Durand and Miki Hermann and Phokion G. Kolaitis. Subtractive Reductions and Complete Problems for Counting Complexity Classes. MFCS, pages 323-332, 2000.
[2] Stephen A. Fenner and Lance Fortnow and Stuart A. Kurtz. Gap-Definable Counting Classes. Structure in Complexity Theory Conference, pages 30-42, 1991.
[3] Johannes Köbler and Uwe Schöning and Jacobo Torán. On Counting and Approximation. Acta Inf., 26(4):363-379, 1989.
[4] Aggelos Kiayias Aris Pagourtzis and Stathis Zachos. Cook Reductions Blur Structural Differences Between Functional Complexity Classes. Panhellenic Logic Symposium, pages 132-137, 1999.
[5] Aris Pagourtzis and Stathis Zachos. The Complexity of Counting Functions with Easy Decision Version. MFCS, pages 741-752, 2006.
[6] Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. SIAM J. Comput., 20(5):865-877, 1991.
[7] Seinosuke Toda and Mitsunori Ogiwara. Counting Classes Are at Least as Hard as the Polynomial-Time Hierarchy. Structure in Complexity Theory Conference, pages 2-12, 1991.
[8] Seinosuke Toda and Osamu Watanabe. Polynomial Time 1-Turing Reductions from #PH to #P. Theor. Comput. Sci., 100(1):205-221, 1992.
[9] Leslie G. Valiant. The Complexity of Computing the Permanent. Theor. Comput. Sci., 8:189-201, 1979.