ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών

Δομική και περιγραφική πολυπλοκότητα δύσκολων προβλημάτων μέτρησης με εύκολο πρόβλημα απόφασης

Διδακτορική διατριβή

της

Αγγελικής Χαλκή

Επιβλέπων: Αριστείδης Παγουρτζής, Καθηγητής

Αθήνα, Ιούλιος 2022

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
Computation and Reasoning Laboratory

On structural and descriptive complexity of hard counting problems the decision version of which is easy

PhD Thesis

by

Aggeliki Chalki

Supervisor: Aris Pagourtzis

Athens, July 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας, Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογισμών

Δομική και περιγραφική πολυπλοκότητα δύσκολων προβλημάτων μέτρησης με εύκολο πρόβλημα απόφασης

Διδακτορική διατριβή της Αγγελικής Χαλκή

Τριμελής συμβουλευτική επιτροπή: Αριστείδης Παγουρτζής (επιβλέπων)
Ευστάθιος Ζάχος
Δημήτριος Φωτάκης

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 22η Ιουλίου 2022.

.....	.....	.....
Αριστείδης Παγουρτζής	Ευστάθιος Ζάχος	Δημήτριος Φωτάκης
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ
.....	.....	.....
Ανδρέας Γαλάνης	Αντώνιος Αχιλλέως	Lane Hemaspaandra
Αν. Καθηγητής	Επ. Καθηγητής	Καθηγητής
University of Oxford	Reykjavik University	University of Rochester

.....
Κωνσταντίνος Κούτρας
Αν. Καθηγητής American University of the Middle East

Abstract

In this thesis, we study classes of counting problems the decision version of which is easy. The complexity class #P, introduced by Valiant [149], contains the counting versions of NP problems. We focus on #PE [122], the class of #P functions the decision version of which is in P. In fact, we are mostly interested in a subclass of #PE, namely TotP [102], which contains essentially all self-reducible #PE functions. TotP has also an interesting simple syntactic characterization: a function belongs to TotP if it counts the number of all paths of an NPTM. Notably, TotP is a robust class; it has nice closure properties and natural complete problems under parsimonious reductions, i.e. reductions that preserve the number of solutions. We study the class TotP from different angles:

1. We discuss the first complete problems for TotP under parsimonious reductions that were first presented in [22, 13]. In particular, the problem Size-of-Subtree has been introduced by Knuth as the problem of estimating the size of a backtracking procedure's tree [105] and has been studied from many perspectives.

2. Efficient and exact counting is very rare, so the main research interest in the area of counting complexity is to classify counting problems with respect to their approximability and design approximation algorithms for those that can be approximated. In this quest, properties of problems in TotP are important. We examine the relationship of TotP to the class of approximable counting problems, namely FPRAS. Some of the relevant results presented here, have appeared in [22, 24].

3. To deal with the previous question, we needed to define classes with counting problems the decision version of which is in RP. We examine one such problem, namely #Exact Matchings, with respect to its exact and approximate computation. #Exact Matchings is a generalization of counting perfect matchings in graphs that contain both black and red edges.

4. Then we turn our attention to logical characterizations of classes of counting problems. We build upon previous work in the area of descriptive complexity of counting problems. We give logical characterizations of two robust subclasses of TotP and determine their relationship with the class FPRAS [24]. Most importantly, we provide a logical characterization of TotP, which was an open question in the area of descriptive complexity [15]. To

express self-reducibility, we add recursion on functions over second-order variables to the logic **QSO** introduced in [15], which, we believe, is of independent interest.

5. We investigate the power of counting the total number of paths of an NPTM by introducing classes of decision problems defined by properties of TotP functions. These classes can be seen as tot-counterparts of traditional classes. We explore the relationship among the newly introduced classes and their '#P-definable' analogs. We also build upon a result by Curticapean [52] and we examine complete problems for some of these classes that are defined as variants of the problem of counting perfect matchings in a graph.

Acknowledgments

I have been in CoreLab for almost one quarter of my life. There are not enough words to express my gratitude to all the people that I have met and worked with during the past years.

First and foremost, I would like to thank my supervisor Aris Pagourtzis who taught me not only computational complexity, but also how to be patient, optimistic, and respectful to both science and people.

I thank Stathis Zachos for showing me how sharing can make you a better person and researcher.

I thank Dimitris Fotakis for being always positive and supportive. His attitude towards research and teaching has been very inspiring for me.

I thank Kostas Coutras for trusting me and for giving me valuable advice (that I never followed as he would say).

I thank Antonis Achilleos for bearing with me in several periods through these years. He was always present when I wanted his opinion and help. I thank him for his generosity in both research and life.

I thank Lane Hemaspaandra for providing me useful and interesting information relevant to this thesis' subject. His deep insight into structural complexity helped me improve parts of this thesis.

I thank Andreas Galanis for agreeing to be part of my committee (twice). I am very glad and honoured. I thank him for his willingness to give me his attention and invaluable feedback.

I thank Andreas Göbel for revealing some of the beauty of counting complexity to me. I also thank him for being so patient with me and for the pleasant discussions we had.

I thank Eleni Bakali, Antonis Antonopoulos, Stavros Petsalakis, and Petros Pantavos for the hours that we spent talking and writing together. This thesis could not have been completed without them.

I thank Petros and Dora for helping me and being so kind to me all these years.

I thank Elli, Christina, Giannis, JimGen, Vaggelis, Thomas, Thodoris, Vasiliki, Petros, Zachos, and every other person that has been part of the logic study group. This group has been a source of energy, ideas, and knowledge over the past years.

I thank Pouran, Eleni, Antonis, Stavros, Thanasis, Peter Pan, Natalia, Giannis, Jela, Panagiotis Pats, Eleni Psar, Alkis, Panagiotis Grontas, Marianna, Sotiris, Loukas, Stratis, Zeta, Katerina, Lydia, Dimitris, and all other members of CoreLab that have made this place so special. I thank Natalia for being my friend, so supportive and generous.

I thank my friends for their understanding and support in every choice I have made. I thank Alexandros for being there for me. I thank my parents Panagiota and Giorgos, and my brother Tolis for believing in me and for encouraging me from the start to the end.

Aggela

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A counting problem is a function that maps an instance of a decision problem to the number of solutions that the problem has on input this particular instance. For example #SAT is a function that maps an input formula to the number of its satisfying assignments. The class of functions, for which the corresponding decision problem is in the class NP, is the class #P, introduced in Valiant's seminal paper [149]. Equivalently, functions in #P count accepting paths of nondeterministic polynomial time Turing machines (NPTMs).

Interestingly, various problems from different scientific fields can be expressed as counting ones.

1. Computing the partition function, in statistical physics [81, 90, 91, 115]. For example, the partition function of the hardcore model is a generalization of the problem of counting independent sets in a graph [155].

2. Computing the volume of a convex body, in computational geometry [58].

3. Computing the permanent, in linear algebra [149].

4. Determining the probability that a given network becomes disconnected due to edge failures, in network design [97].

5. Computing the social cost of a given mixed Nash equilibrium, in selfish games in algorithmic game theory [69].

6. There are optimization problems that require counting solutions to some corresponding decision problem [129, 117].

When we consider counting, non-trivial facts hold. For example, both the NP-complete problem CNF and the polynomial-time solvable problem DNF have counting versions that are #P-complete under Turing reductions. Although no counting problem with an NP-complete decision version can be efficiently approximated unless RP = NP [59], among counting problems with a decision version in P there are some that have a polynomial-time randomized approximation algorithm (e.g. #DNF [98]) and others that cannot be approximated efficiently unless RP = NP (e.g. #IS [57]).

Since very few counting problems can be exactly computed in polynomial time (e.g. counting spanning trees [107, chapter 6]), the interest of the community has turned to the complexity of approximating them. In this quest, the class #PE [122] of problems in #P with a decision version in P is of great significance, since counting problems that admit a fully polynomial-time randomized approximation scheme (fpras) can be found only among those with an easy decision version (i.e. in BPP). In this thesis, we focus on a subclass of #PE, namely TotP [102]. Notably, almost all known counting problems that admit an fpras belong to TotP.

TotP is the class of functions that count the total number of paths of NPTMs. In fact, a function in TotP is defined to be equal to the number of computation paths of an NPTM minus one, so that functions in TotP can take zero values as well. Except for this simple structural characterization, it was shown in [123] that TotP has a noteworthy property that can be considered as an alternative definition: it is the class of all self-reducible problems with a decision version in P, which is also closed under parsimonious reductions.

TotP is a *robust* class, where we consider robustness as defined by the authors of [15]; a class is considered to be robust if either it is closed under addition, multiplication, and subtraction by one or it has natural complete problems. In particular, TotP satisfies both these requirements as we discuss later on.

Finally, TotP can also be characterized via interval size functions [84, 27]; it is the class of interval size functions which are defined on some P-definable total p-order A via polynomial-time computable boundary functions, where in addition the lexicographically nearest function for A, LN_A, is efficiently computable.

We are going to present results about TotP from the viewpoint of structural and descriptive complexity. The reader can find a guided tour to this thesis in Section 1.5.

1.1 The complexity class #P

The standard model of computation under which we study counting complexity is the nondeterministic Turing machine. In particular, we focus on polynomial-time complexity, thus the model of interest is the nondeterministic polynomial-time Turing machine (NPTM). We assume a fixed alphabet, conventionally $\Sigma = \{0, 1\}$, in which we encode both problem instances and solutions. The symbol Σ^* denotes the set of all finite strings over the alphabet Σ. The length of a string x is denoted by $|x|$. For an NPTM M, there is some polynomial p such that for any $x \in \Sigma^*$, all computation paths of M on input x have length at most $p(|x|)$. An NPTM M is in standard form if for any x, each path of $M(x)$ is encoded by a string of length exactly $p(|x|)$. We say that M is in normal form if it is in standard form and, in addition, for any x there are exactly $2^{p(|x|)}$ computation paths in $M(x)$.

A relation $R \in \Sigma^* \times \Sigma^*$ can be interpreted as assigning to each problem instance $x \in \Sigma^*$, a set of solutions $\{y \in \Sigma^* \mid R(x, y)\}$. For example, consider the relation R_{PM} which associates with each undirected graph G, the set of perfect matchings of G.

$$R_{PM} = \{(x, y) \mid x \in \Sigma^* \text{ is an encoding of a graph } G \text{ and } y \in \Sigma^* \text{ is an encoding of a perfect matching of } G\}.$$

A number of naturally defined problems are related to each relation of the above form. We are interested in the *existence*, *counting*, and *uniform generation* problems.

- Existence: Is there a $y \in \Sigma^*$, such that $R(x, y)$?
- Counting: How many y are there such that $R(x, y)$?
- Uniform generation: Generate uniformly at random a $y \in \Sigma^*$ that satisfies $R(x, y)$.

Definition 1.1. *A language L belongs to* NP *if there is a polynomial-time decidable relation R and a polynomial p such that $x \in L \Leftrightarrow \exists y \left[|y| = p(|x|) \text{ and } R(x, y)\right]$.*
Equivalently, $L \in$ NP if there is an NPTM M such that

$$x \in L \Leftrightarrow M(x) \text{ has an accepting computation path.}$$

Definition 1.2 ([149])**.** *A function $f : \Sigma^* \to \mathbb{N}$ belongs to* #P *if there exists a polynomial-time decidable relation R and a polynomial p such that for every $x \in \Sigma^*$,*

$$f(x) = \left|\{y \in \Sigma^* \mid |y| = p(|x|) \text{ and } R(x, y)\}\right|.$$

Equivalently, $\#\mathsf{P} = \{acc_M : \Sigma^* \to \mathbb{N} \mid M \text{ is an NPTM}\}$, *where* $acc_M(x) =$*#(accepting computation paths of* M *on input* x*).*

The decision version of a function $f \in \#\mathsf{P}$ is the following problem: 'Given x, is $f(x)$ non-zero?'. Equivalently, 'Given x, is there a y, such that $R(x, y)$?', or 'Given x, is there at least one accepting path of M on input x?', where M is the NPTM corresponding to f. For each function $f \in \#\mathsf{P}$ an associated language $L_f = \{x \in \Sigma^* \mid f(x) > 0\}$ can be defined. Clearly, $L_f \in \mathsf{NP}$.

1.1.1 The property of self-reducibility

Self-reducibility is an important property that has been extensively studied since Trakhtenbrot first introduced the notion of autoreducibility in 1970 [145]. Interesting approaches to this property can be found in [83], where one can see self-reducibility as a technique, and in [7], in which the authors prove that a strong self-reducibility property can amplify lower-bound results. Regarding counting solutions of a problem, self-reducibility implies that almost uniform sampling from the set of solutions is equivalent to approximate counting them [94]. The definition of self-reducibility used by Jerrum et al. in [94] is due to Schnorr [134]. Although there are problems which are not self-reducible unless $\mathsf{P} = \mathsf{NP}$, such as counting k-colourings, $k \geq 4$, in planar graphs [101], most problems satisfy this property. So self-reducibility can be seen as being the rule and not the exception.

We first give the definition of self-reducibility for a relation.

Definition 1.3 ([134, 94])**.** *A relation* $R \in \Sigma^* \times \Sigma^*$ *is* self-reducible *if the following hold.*

1. *There is a polynomial* p *such that* $R(x, y) \Rightarrow |y| = p(|x|)$.
2. *There are polynomial-time computable functions* $\psi \in \Sigma^* \times \Sigma^* \to \Sigma^*$ *and* $\sigma : \Sigma^* \to \mathbb{N}$ *such that for every* $x \in \Sigma^*$
   - *(a)* $\sigma(x) = \mathcal{O}(\log |x|)$,
   - *(b)* $p(|x|) > 0 \Rightarrow \sigma(x) > 0$,
   - *(c)* $|\psi(x, w)| \leq |x|$ *for every* $w \in \Sigma^*$ *with* $|w| = \sigma(x)$, *and*
   - *(d)* $R(x, y_1 ... y_n) \Leftrightarrow R(\psi(x, y_1 ... y_{\sigma(x)}), y_{\sigma(x)+1} ... y_n)$ *for every* $y_1 ... y_n \in \Sigma^*$.

Intuitively, the solution set associated with a given instance of a problem can be expressed in terms of the solutions sets of a number of smaller instances of the same problem. Polynomial

p gives the length of the solutions to instances. Given an instance x and initial segment w of length $\sigma(x)$ of any solution to x, $\psi(x,w)$ is an instance x', such that w concatenated with any solution to x' forms a solution to x.

Next we define a version of self-reducibility for counting functions. This kind of self-reducibility first appeared in [123] and then was generalized in [25]. Informally, a function is self-reducible if its value on an instance can be recursively computed by evaluating the same function on a polynomial number of smaller instances. By generalizing, we allow recursive computations on, not necessarily, smaller instances.

Definition 1.4 ([123, 25]). *A function $f : \Sigma^* \to \mathbb{N}$ is called* poly-time self-reducible *if for all $x \in \Sigma^*$*

(a) f can be processed recursively by reducing x to a polynomial number of instances $h(x,i)$, where h is polynomial-time computable and $0 \leq i \leq r(|x|)$ for some polynomial r. Formally, for every $x \in \Sigma^*$,

$$f(x) = t(x) + \sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i)).$$

(b) The recursion terminates after at most polynomial depth. Formally, the depth of the recursion is $q(|x|)$, for some polynomial q and for every $x \in \Sigma^*$ and $\overrightarrow{j} \in \{0,\ldots,r(|x|)\}^{q(|x|)}$,

$$f(\tilde{h}(x,\overrightarrow{j})) \text{ can be computed in polynomial time,}$$

where $\tilde{h}$ is the extension of h such that $\tilde{h}(x,\varepsilon) = x$ and $\tilde{h}(x, j_1 ... j_k) = h(\tilde{h}(x, j_1 ... j_{k-1}), j_k)$.

(c) Every instance invoked in the recursion is of polynomial size in $|x|$. Formally, there is a polynomial p, such that for every $x \in \Sigma^*$, $k \leq q(|x|)$, and $\overrightarrow{j} \in \{0,\ldots,r(p(|x|))\}^k$ it holds that $|\tilde{h}(x,\overrightarrow{j})| \in \mathcal{O}\big(p(|x|)\big)$.

Note that if the instances $h(x,i)$ are of smaller size than x, that is $|h(x,i)| < |x|$ for every x and i, $0 \leq i \leq r(|x|)$, then requirement (b) holds trivially. Moreover, (c) requires that all instances that f will be computed on, must be of polynomial length in $|x|$, which also holds if h is of decreasing length.

We prove below that self-reducibility of a relation is a more general property than the poly-time self-reducibility of the corresponding function.

Proposition 1.1. *Let $R : \Sigma^* \times \Sigma^*$ be a relation and $f : \Sigma^* \to \mathbb{N}$ be the function that for every $x \in \Sigma^*$, $f(x) = |\{y \in \Sigma^* \mid R(x,y)\}|$. If R is self-reducible, then f is poly-time self-reducible.*

Proof. The value of f on an instance x can be expressed in the following way.

$$f(x) = |\{y \mid R(x,y)\}| = \sum_{w \in \Sigma^{\sigma(x)}} |\{y' \mid R(\psi(x,w), y')\}| = \sum_{w \in \Sigma^{\sigma(x)}} f(\psi(x,w))$$

where ψ and σ are as in Definition 1.3. So poly-time self-reducibility for f is satisfied, since $|\Sigma^{\sigma(x)}|$ is polynomial in $|x|$ and ψ is polynomial-time computable, that also returns instances of smaller length. □

The inverse fact is not necessarily true. That is, if a function f, associated with a relation R as above, is poly-time reducible, then R is not necessarily self-reducible. First, $h(x,i)$ may be of greater length than x in Definition 1.4, whereas $\psi(w,x)$ of Definition 1.3, are always instances of smaller length than x. Second, even if we restrict ourselves to functions h such that $|h(x,i)| \leq |x|$, it may be the case that requirements of Definition 1.3 are not satisfied. However, as far as we know, there is no problem that has a poly-time self-reducible counting function and not a self-reducible relation.

1.1.2 Fpaus and fpras for counting problems

The probabilitic Turing machine (PTM) is the usual basis for defining randomized complexity classes and randomized algorithms. A probabilistic Turing machine T is a Turing machine equipped with special coin-tossing states. Each coin-tossing state q has two possible successor states q_h and q_t; when T enters state q, it moves on to state q_h with probability $\frac{1}{2}$ and to state q_t with probability $\frac{1}{2}$.

For two probability distributions π and π' on a countable set Ω, define the total variation distance between π and π' to be

$$||\pi - \pi'||_{TV} = \frac{1}{2} \sum_{\omega \in \Omega} |\pi(\omega) - \pi'(\omega)| = \max_{A \subseteq \Omega} |\pi(A) - \pi'(A)|.$$

A sampling problem is specified by a relation $R \subseteq \Sigma^* \times \Sigma^*$ which includes pairs (x,y) of problem instances x and solutions y. An *almost uniform sampler* for a solution set R is a randomized algorithm that takes as input an instance x of the problem and a sampling tolerance δ and outputs a solution Y (a random variable of the coin tosses made by the algorithm) such that $R(x,Y)$, and the variation distance between the distribution of Y and the uniform distribution on the set $\{y \mid R(x,y)\}$ is at most δ. If $\{y \mid R(x,y)\} = \emptyset$, we allow the almost uniform sampler to output a special symbol $\perp$. An almost uniform sampler is *fully polynomial*, denoted by fpaus, if it runs in time polynomial in $|x|$ and $\log(\delta^{-1})$.

Definition 1.5. *A* randomized approximation scheme *for a counting problem* $f : \Sigma^* \to \mathbb{N}$ *is a randomized algorithm that takes as input an instance* $x \in \Sigma^*$*, an (accuracy) error tolerance* $\varepsilon > 0$*, and a (probability) error tolerance* $\delta > 0$ *and outputs a value* $\widehat{f(x)} \in \mathbb{N}$*, such that for every* x*,*

$$\Pr[(1-\varepsilon)f(x) \leq \widehat{f(x)} \leq (1+\varepsilon)f(x)] \geq 1-\delta.$$

We speak of a fully polynomial randomized approximation scheme*, denoted by* fpras*, if the algorithm runs in time bounded by a polynomial in* $|x|$*,* ε^{-1}*, and* $\log(\delta^{-1})$*.*

Remark 1.1. *If we remove randomness from an* fpras*, then we obtain a* deterministic fully polynomial-time approximation scheme*, denoted by* fptas*. So, an* fpras *outputs a* $(1 \pm \varepsilon)$*-approximation with probability 1.*

For any self-reducible $f \in \#\mathsf{P}$ the following facts are true.

1. f admits an fpras if and only if f admits an fpaus [94].
2. Any polynomial-time approximation algorithm that provides a polynomial multiplicative approximation error can be boosted to achieve the quality of approximation demanded by an fpras [137].

So, unlike optimization problems that exhibit an hierarchy of possible degrees of approximation, there is one notion of approximation algorithm for counting problems, namely fpras.

We are going to call the counting problems that admit an fpras, approximable (and so inapproximable are the counting problems that do not admit an fpras).

We define the class FPRAS to be the class of all #P problems that are approximable. All the problems in FPRAS are AP-interreducible to each other and also the class is closed under AP reductions.

Definition 1.6 ([59, 24])**.** *A function* f *belongs to* FPRAS *if* $f \in \#\mathsf{P}$ *and there exists an* fpras *for* f*.*

#DNF (or any other counting problem that admits an fpras) can be considered as a representative of this class. We can alternatively define FPRAS to consist of all #P problems that are AP-interreducible with #DNF.

1.1.3 Reductions between counting functions

Reductions between counting functions have been introduced in the literature in a similar manner to the Turing/Cook and Karp/many-one reductions between languages. These reductions do not need to be limited to counting functions and in fact, they have been defined and used more generally. In this thesis we are going to need the following five types of reductions; below $f, g \in \#\mathsf{P}$ and FP is the class of functions computable in polynomial time.

Definition 1.7 ([149]). **Poly-time Turing reductions**. *f reduces to g under poly-time Turing reductions, denoted by $f \leq_{\mathrm{T}}^{\mathrm{p}} g$, if $f \in \mathsf{FP}^{g}$.*

Definition 1.8 ([108]). **Poly-time 1-Turing reductions**. *f reduces to g under poly-time 1-Turing reductions, denoted by $f \leq_{\text{1-T}}^{\mathrm{p}} g$, if $f \in \mathsf{FP}^{g[1]}$.*

Definition 1.9 ([136]). **Parsimonious reductions**. *f reduces to g under poly-time parsimonious reductions, denoted by $f \leq_{\mathrm{pars}}^{\mathrm{p}} g$, if there is a function $h \in \mathsf{FP}$, such that for every $x \in \Sigma^*$, it holds that $f(x) = g(h(x))$.*

Definition 1.10 ([132]). **Product reductions**. *f reduces to g under product reductions, denoted by $f \leq_{\mathrm{pr}}^{\mathrm{p}} g$, if there are $h_1, h_2 \in \mathsf{FP}$ such that for every $x \in \Sigma^*$ it holds that $f(x) = g(h_1(x)) \cdot h_2(x)$.*

Definition 1.11 ([59]). **Approximation preserving (AP) reductions**. *f reduces to g under approximation preserving reductions, denoted by $f \leq_{\mathrm{AP}} g$, if there is a probabilistic oracle Turing machine M that takes as input an instance x of f and $0 < \varepsilon < 1$ and satisfies the following three conditions:*

1. *every oracle call made by M is of the form (w, δ), where w is an instance of g, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leq poly(|x|, \varepsilon^{-1})$,*
2. *M meets the specification for being a randomized approximation scheme for f whenever the oracle meets the specification for being a randomized approximation scheme for g, and*
3. *the run-time of M is polynomial in $|x|$ and ε^{-1}.*

We also use the following notation.

(a) If $f \leq_{\mathrm{AP}} g$ (resp. $f \leq_{\mathrm{T}}^{\mathrm{p}} g$, $f \leq_{\text{1-T}}^{\mathrm{p}} g$, $f \leq_{\mathrm{pars}}^{\mathrm{p}} g$, $f \leq_{\mathrm{pr}}^{\mathrm{p}} g$) and $g \leq_{\mathrm{AP}} f$ (resp. $g \leq_{\mathrm{T}}^{\mathrm{p}} f$, $g \leq_{\text{1-T}}^{\mathrm{p}} f$, $g \leq_{\mathrm{pars}}^{\mathrm{p}} f$, $g \leq_{\mathrm{pr}}^{\mathrm{p}} f$), then we say that f and g are AP-interreducible (resp. Turing-equivalent, 1-Turing-equivalent, parsimonious-equivalent, product-equivalent), and we write $f \equiv_{\mathrm{AP}} g$ (resp. $f \equiv_{\mathrm{T}}^{\mathrm{p}} g$, $f \equiv_{\text{1-T}}^{\mathrm{p}} g$, $f \equiv_{\mathrm{pars}}^{\mathrm{p}} g$, $f \equiv_{\mathrm{pr}}^{\mathrm{p}} g$).

(b) For a class of functions F, we denote by $\text{Closure}_{\leq}(\mathsf{F})$ the closure of F under $\leq$, where $\leq \in \{\leq^{\text{p}}_{\text{T}}, \leq^{\text{p}}_{\text{1-T}}, \leq^{\text{p}}_{\text{pars}}, \leq^{\text{p}}_{\text{pr}}, \leq_{\text{AP}}\}$.

Remark 1.2. *Poly-time Turing reductions between functions were introduced by Valiant [149] and are also called Cook in [123], since they can be seen as the analog of Cook reductions between sets. Valiant [149] proved that every #P function can be solved in polynomial time using an oracle call to the problem of computing the permanent of a $(0,1)$-matrix. Zankó [158] formalized polynomial-time many-one reductions, which is the precise type of reductions under which computing the permanent of a $(0,1)$-matrix is #P-complete; polynomial-time many-one reductions are weaker than parsimonious reductions, since there is an additional poly-time computable function ϕ and $f(x) = \phi(g(h(x)))$ and they are stronger than poly-time 1-Turing reductions which allow access to the initial instance x during the polynomial-time computation, that is $f(x) = \phi(x, g(h(x)))$. Poly-time 1-Turing reductions were introduced as metric reductions by Krentel in [108], where they were used between optimization problems.*

Parsimonious reductions were defined in [136] between counting problems; here we provide a more general definition of parsimonious reductions between functions. In [123] parsimonious reductions are called Karp (or poly-time many-one) as they can be considered as the analog of Karp (or poly-time many-one) reductions between sets. Also earlier, Vollmer [153] uses the term polynomial-time functionally many-one reductions for parsimonious ones.

Here we are going to use the terms given in Definitions 1.7–1.11. In specific, when we refer to poly-time Turing (resp. poly-time 1-Turing) reductions, we just write Turing (resp. 1-Turing) reductions.

Parsimonious reductions are product reductions, where $h_2(x) = 1$ for every x. Product reductions are Turing reductions where only one oracle call is needed. The same holds for parsimonious reductions, where in addition, the answer of the oracle cannot be changed. Parsimonious and product reductions are trivially AP reductions, whereas Turing reductions and AP ones are not comparable to each other. An AP reduction from f to g means that an fpras for g yields an fpras for f. If an fpras for f is not possible, then an AP reduction from f to g means that g admits no fpras either.

Remark 1.3. *Notably, parsimonious reductions are provably stronger than poly-time many-one reductions as defined by Zankó in [158], and as a consequence, they are stronger than 1-Turing reductions. Faliszewski and Hemaspaandra have shown in [65] that the Shapley-Shubik power index is #P-complete under poly-time many-one reductions, but cannot be #P-complete under*

parsimonious reductions.

1.2 Decision versus Counting

Counting solutions using an oracle to deciding whether a solution exists, and conversely

Counting using an oracle to decision classes: Can we solve a counting problem by having an oracle to some decision problem? We provide two theorems that answer this question when 'solve' is replaced by 'approximate using either an fpras or an fptas' and the oracle calls are made to the class NP and $\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$, respectively.

Theorem 1.2 (Valiant–Vazirani bisection technique [151]). *For any $f \in \#\mathsf{P}$, there exists a probabilistic TM M equipped with an NP-oracle, which for every input $(x, \varepsilon, \delta) \in \Sigma^* \times \mathbb{R}^+ \times \mathbb{R}^+$ produces an output $M(x, \varepsilon, \delta)$ such that*

$$\Pr(M(x, \varepsilon, \delta) \text{ approximates } f(x) \text{ within ratio } (1 + \varepsilon)) \geq (1 - \delta).$$

Moreover, the running time of M is bounded by a polynomial in $|x|$, $1/\varepsilon$, and $\log(1/\delta)$.

Theorem 1.3 (Stockmeyer's Theorem [140]). *For any $f \in \#\mathsf{P}$, there exists a deterministic TM M equipped with a Σ_2^{P}-oracle, which for every input $(x, \varepsilon) \in \Sigma^* \times \mathbb{R}^+$ produces an output $M(x, \varepsilon)$ such that*

$$M(x, \varepsilon) \text{ approximates } f(x) \text{ within ratio } (1 + \varepsilon).$$

Moreover, the running time of M is bounded by a polynomial in $|x|$ and $1/\varepsilon$.

Both the Valiant–Vazirani bisection technique and Stockmeyer's Theorem can also be found in [94].

Solving decision problems using an oracle to counting: Which decision problems can be efficiently solved using an oracle to counting? Toda's Theorem states that the whole polynomial hierarchy can be solved in polynomial time using only one oracle call to the class #P.

Theorem 1.4 (Toda's Theorem [144]). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$.

The previous three theorems justify the next comment by Dyer et al. [59]: "Informally, from a complexity theoretic perspective, approximate counting is much easier than exact counting. The former lies just above NP, whereas the latter lies above the entire polynomial hierarchy."

NP-completeness versus #P-completeness

#P-completeness $\stackrel{?}{\Rightarrow}$ NP-completeness

1. Under Turing reductions: Is the decision version of a #P-complete problem *under Turing reductions*, NP-complete? The answer is *not always*. Table 1.1 includes different combinations of hard-easy counting and hard-easy decision versions.

Problem	Decision version	Counting version
SAT	NP-complete	#P-complete under parsimonious [16, Theorem 17.10]
3-COLORING	NP-complete	#P-complete under parsimonious [28]
2-COLORING	P	FP [61]
DNF	P	#P-complete under Turing
BIPARTITE PERFECT MATCHING	P	#P-complete under Turing [149]
MONOTONE SAT	trivial	#P-complete under Turing [146]
INDEPENDENT SET	trivial	#P-complete under Turing [146]

Table 1.1: The complexity status of some decision problems and their counting versions.

Some remarks on Table 1.1. The problem of counting 2 colorings of a graph, namely #2COL, can be shown to lie in FP using the dichotomy result of [61]. Alternatively, note that #2COL admits the following simple efficient algorithm: if the input graph is not bipartite, then there is no 2 coloring of the graph. Otherwise, every connected component of the graph can be colored using 2 colors in exactly two different ways. So, the number of 2 colorings of the input graph is equal to 2^k, where k is the number of connected components of the graph. The problem #DNF is #P-complete under Turing reductions since for any **CNF** formula ϕ, it holds that $\#\text{SAT}(\phi) = 2^n - \#\text{DNF}(\neg\phi)$, where n is the number of variables of ϕ.

2. Under AP reductions: The same question *with respect to AP reductions* has also a negative answer. A counterexample is the problem #IS of counting the independent sets of all sizes in a graph, which is hard under AP-reductions [59], but its decision version is trivial, since in any non-empty graph there is always an independent set of size 1.

3. Under parsimonious reductions: However, the same question *with respect to parsimonious reductions* has a positive answer: Every #P-complete problem *under parsimonious reductions* has an NP-complete decision version. For example, consider #SAT. For any #A $\in$ #P, there is a parsimonious reduction from #A to #SAT, which is also a reduction from the decision version of A to SAT.

NP-completeness $\stackrel{?}{\Rightarrow}$ #P-completeness

1. Under Turing reductions: Is the counting version of an NP-complete problem, #P-complete *under Turing reductions*? The positive answer to this question still remains a conjecture.

   Conjecture 1.1 ([59]). *Every* NP*-complete problem has a* #P*-complete counting version under Turing reductions.*

   This question was also examined in [67] as follows. Recall that a problem in NP is defined by a polynomial-time decidable relation R (as in Definition 1.1), which is not necessarily unique. In [67] every such relation is called a witnessing relation (or a witnessing scheme) and can define a counting version of the problem which belongs in #P (see Definition 1.2). A natural question arises: Do all NP-complete problems have (only) #P-complete counting versions? As the following theorem states, the answer to this question is negative with respect to 1-Turing reductions and under some structural conditions. For a definition of the class FewP [8], we refer the reader to Table 4.1 in Chapter 4.

   Theorem 1.5. ([67, Theorem 3.11]).

   (a) If there is an NP*-complete set L that with respect to some witnessing relation R_L is not* #P*-complete under 1-Turing reductions, then* $\mathsf{P} \neq \mathsf{P}^{\#\mathsf{P}}$.

   (b) If $\mathsf{P} \neq \mathsf{P}^{\#\mathsf{P}}$ *and* $\mathsf{NP} = \mathsf{FewP}$*, then each* NP*-complete set has some witnessing scheme with respect to which it fails to be* #P*-complete under 1-Turing reductions.*

2. Under AP reductions: In this case we have the following theorem.

   Theorem 1.6 ([59]). *Every* NP*-complete problem has a* #P*-complete counting version under AP reductions.*

3. Under parsimonious reductions: It has been proven that there is an NP-complete problem that its counting version is not #P-complete under parsimonious reductions unless P = NP [43]. This problem is k-Edge Coloring in k-regular graphs, where $k \geq 3$.

Approximability of a counting problem versus having an easy decision version

$f \in \mathsf{FPRAS} \stackrel{?}{\Rightarrow} f$ has an easy decision version

1. $f \in \mathsf{FPRAS} \Rightarrow L_f \in \mathsf{BPP}$: The decision version of any counting problem in FPRAS is a problem in BPP [75].

   Proposition 1.2. *If $f \in \mathsf{FPRAS}$, then $L_f \in \mathsf{BPP}$.*

   Proof. By Defintion 1.5, if we can have an $(1 \pm \varepsilon)$-approximation of $f(x)$ with bounded probability error, then we can determine whether $f(x) > 0$ with two-sided bounded probability error. □

2. If an NP-complete problem has a counting version that admits an fpras, then RP = NP [59]: We also refer the reader to Theorem 3.1 of Chapter 3 for a proof of this fact.

f has an easy decision version $\stackrel{?}{\Rightarrow} f \in \mathsf{FPRAS}$: Not always. The negative answer here is under the condition that RP ≠ NP. Table 1.2 includes two counterexamples, namely #IS and #2Sat.

Some remarks on Table 1.2. For a proof of the fact that #Sat does not have an fpras unless RP = NP the reader can also see the proof of Theorem 3.1. The problem #IS is a well-studied counting problem as it is a special case of the problem of computing the partition function of the hardcore model in statistical physics. The result of [57] was later improved by Sly [138] and Galanis et al. [73]. Inapproximability of #2Sat under the assumption RP ≠ NP is a consequence of the following simple reduction from #IS to #2Sat: consider a variable x_v for every vertex v of the input graph to #IS and write a conjunction of clauses, where a clause $(\neg x_v \vee \neg x_u)$ is added to the conjunction, for every edge (v, u) of the graph.

Furthermore it is conjectured that specific counting problems the decision version of which is in P, have no fpras; the class #RHΠ_1 [59] contains problems AP-interreducible with #BIS, that is the problem of counting independent sets in a bipartite graph, which it is widely believed not to be approximable. In fact, #BIS is considered to be of intermediate complexity, i.e. neither approximable nor as hard as #Sat, and it emerges in several approximation trichotomy results

Problem	**Decision version**	**Exact complexity**	**Approximability status**
#Sat	NP-complete	#P-complete (under parsimonious)	no fpras unless RP = NP [59]
#IS	P	#P-complete (under Turing)	no fpras unless RP = NP [57]
#2Sat	P	#P-complete (under Turing)	no fpras unless RP = NP
#DNF	P	#P-complete (under Turing)	fpras [98]
#NFA	NL	#P-complete (under Turing)	fpras [14]

Table 1.2: The exact and approximability status of some counting problems.

for classes of counting problems—see for example [60, 74]. The definition of the class $\#\mathsf{RH}\Pi_1$ is given Definition 5.1 of Chapter 5.

1.3 The complexity class TotP

1.3.1 Definition of TotP: counting all paths of an NPTM

The class that contains all the functions in #P with a decision version in P is #PE (#PEasy). Recall that given $f \in \#\mathsf{P}$, $L_f = \{x \in \Sigma^* \mid f(x) > 0\}$ is defined to be the decision version of f.

Definition 1.12 ([122]). $\#\mathsf{PE} = \{f : \Sigma^* \to \mathbb{N} \mid f \in \#\mathsf{P} \text{ and } L_f \in \mathsf{P}\}$.

The complexity class TotP is a subclass of #P that is defined as the class of functions that count the total number of computation paths of NPTMs.

Definition 1.13 ([102]). $\mathsf{TotP} = \{tot_M : \Sigma^* \to \mathbb{N} \mid M \text{ is an NPTM}\}$, *where* $tot_M(x) = \#(\text{all computation paths of } M \text{ on input } x) - 1$.

Remark 1.4. *Let M denote an NPTM and $T_{M(x)}$ denote its computation tree on input x. W.l.o.g. we can assume that, for any x, $T_{M(x)}$ is binary, i.e. every vertex of $T_{M(x)}$ has at most two children. In other words, at any time the computation of M on x is either deterministic or a nondeterministic choice is made between exactly two branches. That is because, if there are $m > 2$ nondeterministic choices for some state-symbol combination, we can modify M by adding $m - 2$ new states so that the modified Turing machine (1) simulates the computation of M on x, (2) makes only nondeterministic choices between two branches, and (3) has the same total number of paths as M.*

Therefore, the computation of any machine M on input x can be seen as a binary tree $T_{M(x)}$, where a branching is created in the computation tree whenever M has to select between two choices. Then,

$$tot_M(x) = \#(\textit{all paths of } M \textit{ on input } x) - 1 = \#(\textit{all branchings of } T_{M(x)}).$$

Sometimes we abuse notation by writing $tot_M(x) = \#(\textit{all branchings of } M \textit{ on input } x)$.

Since an NPTM has at least one computation path, the '-1' in the definition allows TotP to capture functions that take zero value on some inputs. In [80] the function $total_M(x)$ is introduced to denote the total number of M on input x without subtracting 1. Note that the functions defined via $total_M$ instead of tot_M are contained in TotP: given a function f, such that for some NPTM M and every $x \in \Sigma^*$, $f(x) = total_M(x) = \#(\text{all paths of } M \text{ on input } x)$, an NPTM M' can be easily constructed such that $total_{M'}(x) = total_M(x) + 1$, for every x. Then, it holds that $tot_{M'} \equiv total_M$ and so $f \in \mathsf{TotP}$.

The following subsection reveals how the choice of tot_M allows many natural counting problems to lie in TotP. At the beginning of Subsection 4.1 we refer to interesting decision classes defined in [80] via the function $total_M$.

1.3.2 Properties of TotP problems: self-reducibility and easy decision

The following theorem summarizes the relationship among classes defined so far. Here, FP denotes the class of natural-valued functions that are computable in polynomial time.

Theorem 1.7 ([123]). *(a)* $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$. *The inclusions are proper unless* $\mathsf{P} = \mathsf{NP}$.
(b) $\mathsf{FP}^{\mathsf{TotP}[1]} = \mathsf{FP}^{\#\mathsf{PE}[1]} = \mathsf{FP}^{\#\mathsf{P}[1]}$.
(c) TotP *is the closure under parsimonious reductions of self-reducible* #PE *functions.*

Theorem 1.7(a) and (b) state that the classes TotP, #PE, and #P are 1-Turing-equivalent, but they are not parsimonious-equivalent unless P = NP.

Theorem 1.7(c) gives an alternative characterization of problems in TotP. As a result, TotP is a very large class with problems from many different scientific fields, which share the two aforementioned simple properties of being self-reducible and having an easy decision version. At the same time, by having a simple *syntactic* characterization, TotP unifies all these problems and makes their class amenable to having complete problems.

We elaborate on the two different characterizations of TotP with an example.

Example 1.1. *Consider the problem* #BiPerfMatch *of counting perfect matchings in a bipartite graph. Its decision version is the problem of determining whether there is a perfect matching in a bipartite graph and it is in* P. #BiPerfMatch *is also self-reducible since the number of perfect matchings in* G *equals the number of perfect matchings containing some edge* e *plus the number of perfect matchings not containing edge* e*. Computing the two latter numbers is equivalent to counting perfect matchings in two subgraphs of* G*, namely* G_0 *and* G_1*, respectively.* G_0 *results from* G *by removing* e *together with its endpoints, whereas* G_1 *results from* G *by removing only* e *(without removing its endpoints).*

Let G *be an input bipartite graph and* $e_1, ..., e_m$ *be an enumeration list of its edges. Consider an NPTM* M *that, at its first step, determines whether there is a perfect matching in* G*. If the answer is no, it halts. Otherwise, it generates a dummy path and starts a recursive computation as follows. It checks whether there is a perfect matching containing the first edge appearing in the enumeration list, namely* e_1*, and whether there is one not containing* e_1*.*

- *If the answer is yes for both cases,* M *chooses nondeterministically to add* e_1 *to the perfect matching or not and proceeds recursively with* G_0 *and* G_1*, respectively. It also removes from the enumeration list all edges removed from* G*.*

- *If the answer is yes for exactly one case,* M *deterministically proceeds recursively with the corresponding subgraph, i.e. either* G_0 *or* G_1*, and it also removes from the enumeration list all edges removed from* G*. In the case of* G *consisting of just two vertices* u, v *and the edge* $e = (u, v)$*,* M *halts.*

Since M *removes at least one edge at each step, the depth of the recursion is polynomial in the size of* G*. Finally, notice that every sequence of nondeterministic choices corresponds to a perfect matching and so the definition of* TotP *is satisfied; the number of perfect matchings of*

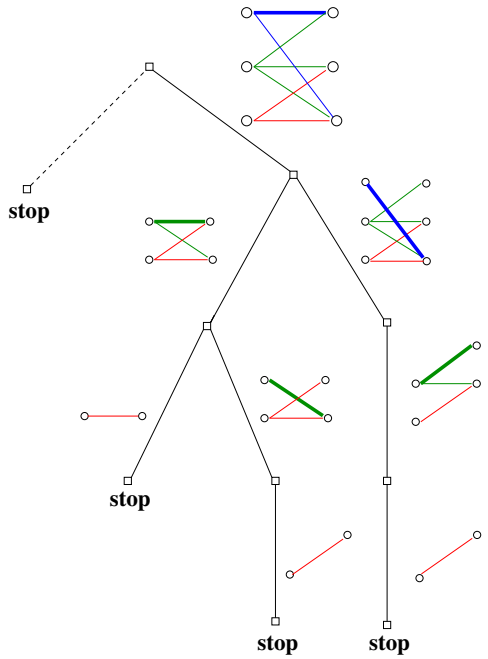


Figure 1.1: NPTM M for which it holds that $tot_M(\widehat{G}) = \#\text{BiPerfMatch}(G)$, where $\widehat{G}$ is the binary encoding of the bipartite graph G depicted on the top of the figure.

G equals the number of all paths minus one. The computation of M on input a bipartite graph with seven edges is depicted in Figure 5.3.

Using similar arguments, we can prove the following proposition.

Proposition 1.3. *The following problems belong to* TotP.

(a) #DNF *(counting satisfying assignments of a* **DNF** *formula),*

(b) #2Sat *(counting satisfying assignments of a* **2SAT** *formula),*

(c) #HornSat *(counting satisfying assignments of a* **CNF** *formula, every clause of which contains at most one positive literal),*

(d) #MonSat *(counting satisfying assignments of a* **CNF** *formula, every clause of which contains only positive literals),*

(e) #BiPerfMatch *(counting perfect matchings in a bipartite graph)*

(f) #PerfMatch *(counting perfect matchings in a general graph),*

(g) #IS *(counting independent sets of any size in a graph).*

Proposition 1.4. *If a problem in* TotP *has a polynomial number of solutions then its solutions can be enumerated in polynomial time.*

Proof. For every problem in TotP the computation tree of an NPTM can be constructed like in Example 1.1. If the number of solutions is bounded by a polynomial in the input size, then this construction is a polynomial-time enumeration of the problem's solutions. □

Proposition 1.4 is also a corollary of [133, Lemma 4.10] which states that any self-reducible relation with an easy existence version has a polynomial delay enumeration algorithm.

1.3.3 Characterization of TotP as a class of interval size functions

Interval size functions were introduced by Hemaspaandra et al. [84]. The study of this kind of functions started with the observation that most counting classes obscure factors like orders on solution sets. Interestingly, counting classes can be characterized as classes of functions the values of which are equal to the size of an interval defined by some order. For example, #P is the class of interval size functions of P-decidable partial p-orders.

A binary relation over Σ^* is a partial order if it is reflexive, antisymmetric, and transitive. A partial order A is a total order if for any $x, y \in \Sigma^*$, it holds that $(x, y) \in A$ or $(y, x) \in A$. We say that an order A is P-decidable if $A \in \mathsf{P}$ and that it is a p-order if there exists a bounding polynomial p such that for all $(x, y) \in A$ it holds that $|x| \leq p(|y|)$. We denote by $(x, y)_A$ the open interval $(x, y)_A = \{z \in \Sigma^* \mid x <_A z <_A y\}$ and we also use $[x, y]_A$, $[x, y)_A$ and $(x, y]_A$ for the closed, right-open and left-open intervals respectively. For any interval I, we denote by $||I|| = |\{z \in \Sigma^* \mid z \in I\}|$ the size of I.

Definition 1.14. *A function $f\colon \Sigma^* \to \mathbb{N}$ is called an interval size function on an order A if there exist boundary functions $b, t : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$, $f(x) = ||(b(x), t(x))_A||$.*

Proposition 1.5. *For any function f, the following are equivalent:*

1. *$f \in \#\mathsf{P}$.*
2. *there exist a partial p-order $A \in \mathsf{P}$ and functions $b, t \in \mathsf{FP}$ such that, for all $x \in \Sigma^*$, $f(x) = ||(b(x), t(x))_A||$.*
3. *there exist a total p-order $A \in \mathsf{P}$ and functions $b, t \in \mathsf{FP}$ such that, for all $x \in \Sigma^*$, $b(x) \leq_A t(x)$ and $f(x) = ||(b(x), t(x))_A||$.*

For an order A we write $x \prec_A y$ to abbreviate $(x <_A y \wedge \neg\exists z(x <_A z <_A y))$. We say that x is a predecessor of y, or y is a successor of x. If $A_\prec = \{(x, y) \mid x \prec_A y\}$ is in P, we say that A has efficient adjacency checks.

Definition 1.15. $\mathsf{IF}_{\mathsf{p}}^{\prec}$ *is the class of interval size functions defined on* P*-definable partial orders with efficient adjacency checks via polynomial-time computable boundary functions.*

Proposition 1.6. $\mathsf{IF}_{\mathsf{p}}^{\prec} = \#\mathsf{PE}$.

Bampas et al. [27] added various other feasibility constraints to interval size functions defined on P-decidable p-orders. In particular, if we add a polynomial-time computable lexicographically nearest function, then we obtain the class TotP.

Definition 1.16 ([27])**.** $\mathrm{LN}_A : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ *is the lexicographically nearest function for* A*:* $\mathrm{LN}_A(x, y, z)$ *is the string* $w \in [x, y]_A$ *such that* w *is closest to* z *in the lexicographic order (breaking ties arbitrarily). If* $[x, y]_A$ *is empty, then* $\mathrm{LN}_A(x, y, z)$ *is undefined for any* z*.*

Definition 1.17. $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$ *is the class of interval size functions defined on some polynomial-time decidable total p-order* A *via polynomial-time computable boundary functions where in addition* $\mathrm{LN}_A \in \mathsf{FP}$.

Proposition 1.7 ([27])**.** $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}} = \mathsf{TotP}$.

1.3.4 TotP is robust

The class TotP is also a robust class in the sense that it has natural complete problems (which will be shown in Chapter 2) and it is closed under addition, multiplication, and subtraction by one, as shown below. This notion of robustness was suggested by Arenas et al. [15].

Proposition 1.8. TotP *is closed under addition, multiplication, and subtraction by one.*

Proof. We are going to show that if $f, g \in \mathsf{TotP}$, then $h_1 = f + g$, $h_2 = f \cdot g$, and $h_3 = f \dot{-} 1$ also belong to TotP. Specifically, $h_3 : \Sigma^* \to \mathbb{N}$ is defined by

$$h_3(x) = \begin{cases} f(x) - 1, & \text{if } f(x) \neq 0 \\ f(x), & \text{if } f(x) = 0 \end{cases}.$$

Let M_f, M_g be NPTM such that for every $x \in \Sigma^*$, $f(x) = tot_{M_f}(x) = \#(\text{paths of } M_f \text{ on } x) - 1$ and $g(x) = tot_{M_g}(x) = \#(\text{paths of } M_g \text{ on } x) - 1$. We are going to construct M_1, M_2, and M_3 such that $h_i(x) = tot_{M_i}(x) = \#(\text{paths of } M_i \text{ on } x) - 1$, for $i \in \{1, 2, 3\}$.

- *Subtraction by one*: M_3 on x simulates M_f on x with a few modifications as follows. If M_f has only one path, then M_3 does exactly what M_f does. If M_f makes at least one

nondeterministic choice, M_3 copies the behavior of M_f, but while simulating the leftmost path, before making a nondeterministic choice, it checks whether one of the choices leads to a deterministic computation. The first time M_3 detects such a choice, it eliminates the path corresponding to the deterministic computation and continues the simulation of M_f. Notice that M_3 can recognize the leftmost path since computation paths can be lexicographically ordered. In this case, M_3 has one path less than M_f. In both cases, it holds that $h_3(x) = tot_{M_3}(x) = tot_{M_f}(x)\dot{-}1 = f(x)\dot{-}1$.

- *Addition*: If one of the machines, let's say M_f, has one computation path on input x, then $f(x) = 0$. So, on input x, M_1 checks whether either M_f or M_g has exactly one path and if this the case, it simulates the other one, i.e. M_g or M_f, respectively. Otherwise, on input x, M_1 simulates M_3 and M_g nondeterministically, i.e. in two different branches. Since #(paths of M_3 on x) $= h_3(x) + 1 = f(x)$ and #(paths of M_g on x) $= g(x) + 1$, we have that #(paths of M_1 on x) = #(paths of M_3 on x) + #(paths of M_g on x) $= f(x) + g(x) + 1$. This implies that $tot_{M_1}(x) = f(x) + g(x) = h_1(x)$.

- *Multiplication*: If one of the machines, let's say M_f, has one computation path on input x, then $f(x) = 0$. So, on x, M_2 checks whether at least one of M_f and M_g has exactly one path and if this is true, it generates one path and halts. Otherwise, consider the function $h_4 : \Sigma^* \to \mathbb{N}$ such that $h_4(x) = g(x)\dot{-}1$ for every $x \in \Sigma^*$ and the NPTM M_4 such that $h_4(x) = tot_{M_4}(x)$. On input x, M_2 generates two branches. The first branch is a dummy path. On the second branch, M_2 simulates M_3 and M_4 sequentially. So, #(paths of M_2 on x) = #(paths of M_3 on x)$\cdot$#(paths of M_4 on x)$+1 = f(x)\cdot g(x)+1 = h_2(x)$. □

For the classes #P and #PE, the following facts are known.

Proposition 1.9. *(a)* ([121]). #P *is not closed under subtraction by one unless* SPP $\subseteq$ NP. *(b)* ([123]). #PE *is not closed under subtraction by one unless* P = NP.

1.3.5 Closure of TotP under different kinds of reductions

Below we prove a proposition about the closure of TotP under different kinds of reductions that have been defined in Subsection 1.1.3.

Proposition 1.10. *(a)* TotP *is closed under parsimonious reductions.*

(b) TotP *is closed under product reductions.*

(c) TotP *is not closed under Turing reductions unless* P = NP.

(d) TotP *is not closed under approximation preserving reductions unless* P = RP.

Proof. (a) This is true by Theorem 1.7(c).
(b) Let $f \in \mathsf{TotP}$ and $g \leq^{\mathrm{p}}_{\mathrm{pr}} f$. Then for every $x \in \Sigma^*$, it holds that $g(x) = f(h_1(x)) \cdot h_2(x)$ for some $h_1, h_2 \in \mathsf{FP}$. The functions $f \circ h_1$ and h_2 are both in TotP. So, $g \in \mathsf{TotP}$, since TotP is closed under multiplication.
(c) If TotP is closed under Turing reductions, then #Sat $\in$ TotP, since #Sat $\leq^{\mathrm{p}}_{\mathrm{T}}$ Permanent and Permanent $\in$ TotP. This would imply that P = NP.
(d) TotP contains problems that are approximable, such as #DNF. Every problem in FPRAS can be trivially reduced to #DNF under aporoximation preserving reductions. So if TotP is closed under approximation preserving reduction, then FPRAS $\subseteq$ TotP, which in turn implies that P = RP. The last implication is proven in Corollary 3.5 of Chapter 3. □

1.4 Descriptive complexity of #P

In the area of descriptive complexity, we are interested in determining the type of logic that is needed to express the problems in a complexity class. In specific, to capture counting problems, the following two approaches are relevant to our work [132, 15]. We include some notation, facts, and theorems to introduce the reader to previous useful results.

A relational vocabulary $\sigma = \{\mathcal{R}_1^{a_1}, ..., \mathcal{R}_k^{a_k}\}$ is a finite set of relation symbols. Each relation symbol $\mathcal{R}_i$ has a positive integer a_i as its designated arity.

Definition 1.18. *A structure $\mathcal{A} = \langle A, R_1, ..., R_k \rangle$ over σ consists of a set A, called the universe of $\mathcal{A}$ and relations R_1,...,R_k of arities $a_1, .., a_k$ on A, which are interpretations of the corresponding relational symbols.*

A finite ordered structure is a structure with a finite universe and an extra relation $\leq$, which is interpreted as a total order on the elements of the universe.

In what follows, we shall not make a notational difference between the relations $\mathcal{R}_i$ and their interpretations R_i and denote both by R_i.

For example, a graph is represented as a finite structure using the vocabulary $\sigma_G = E^2$ corresponding to the edge relation. A boolean formula ϕ in conjunctive normal form with at most three literals per clause, called a **3CNF** formula, can be encoded as a finite structure using

the vocabulary $\sigma_{3CNF} = \{C_0^3, C_1^3, C_2^3, C_3^3\}$, where $C_i(x_1, x_2, x_3)$ iff $\neg x_1 \vee ... \vee \neg x_i \vee x_{i+1} \vee ... \vee x_3$ appears as a clause in ϕ.

We consider that a counting function takes as input a finite ordered structure. The value of the function on some input is the number of feasible solutions to the counting problem corresponding to this function.

Given a relational vocabulary σ, the set of First-Order logic formulas (**FO** formulas) over σ is given by the following grammar.

$$\phi := \ x = y \ \mid \ R(\overrightarrow{x}) \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \exists x\phi \ \mid \ \top$$

where x, y are first-order variables, $R \in \sigma$, $\overrightarrow{x}$ is a tuple of first-order variables, $\top$ represents a tautology.

The logical symbols of Second-Order logic (**SO**) include all the logical symbols of **FO** and also an infinite set of second-order variables, denoted by uppercase letters $X, Y, Z, ...$ Each second-order variable X has an arity, denoted by $\mathsf{arity}(X)$. The set of **SO** formulas over σ is given by:

$$\phi := \ x = y \ \mid \ R(\overrightarrow{x}) \ \mid \ X(\overrightarrow{y}) \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \exists x\phi \ \mid \ \exists X\phi \ \mid \ \top$$

where x, y are first-order variables, $R \in \sigma$, X is a second-order variable, $\overrightarrow{x}, \overrightarrow{y}$ are tuples of first-order variables, and $\top$ represents a tautology.

In addition, we are going to use the boolean connectives $\wedge, \rightarrow$, the quantifier $\forall$, and the symbol $\bot$ that represents the negation of a tautology.

1.4.1 #P = #FO

Definition 1.19. *Let σ be a vocabulary containing the relation symbol $\leq$. Let f be a counting function with structures $\mathcal{A}$ over σ as instances. Let $\overrightarrow{x} = (x_1, ..., x_v)$, $\overrightarrow{X} = (X_1, ..., X_r)$, , $r, v \geq 0$, and $r + v > 0$, be sequences of first-order and second-order variable symbols, respectively. We say that $f \in \#\mathsf{FO}$ if there exists a first-order formula $\phi(\overrightarrow{x}, \overrightarrow{X})$ with relation symbols from $\sigma \cup \overrightarrow{X}$ and free first-order variables from $\overrightarrow{x}$, such that,*

$$f(\mathcal{A}) = |\{\langle \overrightarrow{X}, \overrightarrow{x} \rangle \mid \mathcal{A} \models \phi(\overrightarrow{x}, \overrightarrow{X})\}|.$$

Classes $\#\Pi_n$, $\#\Sigma_n$, $n \geq 0$, are defined as in the above definition, when $\mathbf{\Pi_n}$, $\mathbf{\Sigma_n}$ formulas are used, respectively, instead of arbitrary first-order formulas. For every n, $\mathbf{\Sigma_n}$ (resp. $\mathbf{\Pi_n}$) formulas

is the set of **FO** formulas of the form $\exists\overrightarrow{x}_1\forall\overrightarrow{x}_2...\exists\overrightarrow{x}_{n-1}\forall\overrightarrow{x}_n\,\psi$ (resp. $\forall\overrightarrow{x}_1\exists\overrightarrow{x}_2...\exists\overrightarrow{x}_n\,\psi$) if n is even, or $\exists\overrightarrow{x}_1\forall\overrightarrow{x}_2...\forall\overrightarrow{x}_n\,\psi$ (resp. $\forall\overrightarrow{x}_1\exists\overrightarrow{x}_2...\forall\overrightarrow{x}_n\,\psi$) if n is odd.

For example, the problem of counting satisfying assignments of a **3CNF** formula is in $\#\mathsf{FO}$, since

$$\begin{aligned}\#3\textsc{Sat}(\mathcal{A}) = |\{\langle T\rangle \mid \mathcal{A} \models \forall x_1\forall x_2\forall x_3\big(&C_0(x_1,x_2,x_3) \rightarrow T(x_1)\vee T(x_2)\vee T(X_3)\wedge\\ &C_1(x_1,x_2,x_3) \rightarrow \neg T(x_1)\vee T(x_2)\vee T(X_3)\wedge\\ &C_2(x_1,x_2,x_3) \rightarrow \neg T(x_1)\vee \neg T(x_2)\vee T(X_3)\wedge\\ &C_3(x_1,x_2,x_3) \rightarrow \neg T(x_1)\vee \neg T(x_2)\vee \neg T(X_3)\big)\end{aligned}$$

where $\mathcal{A}$ is a structure over σ_{3CNF}. In specific, $\#3\textsc{Sat} \in \#\Pi_1$.

The class $\#\mathsf{P}$ coincides with $\#\mathsf{FO}$ and in fact, with $\#\Pi_2$. By considering $\#\mathsf{FO}$ and its subclasses, the following hierarchy is formed below $\#\mathsf{P}$.

Theorem 1.8 ([132]). *(a)* $\#\mathsf{P} = \#\mathsf{FO} = \#\Pi_2 = \#\Pi_n = \#\Sigma_n$ *over finite ordered structures.*
(b) $\#\Sigma_0 = \#\Pi_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2 = \#\mathsf{P}$.

1.4.2 $\#\mathsf{P} = \Sigma\mathsf{QSO}(\mathsf{FO})$

Given a vocabulary σ, the set of Quantitative Second-Order logic (**QSO**) formulas over σ are obtained by also using quantitative quantifiers—addition quantifier Σ and multiplication quantifier Π—over first- and second-order variables. **QSO** formulas are defined as follows.

$$\alpha := \ \phi \ \mid \ s \ \mid \ (\alpha+\alpha) \ \mid \ (\alpha\cdot\alpha) \ \mid \ \Sigma x.\alpha \ \mid \ \Pi x.\alpha \ \mid \ \Sigma X.\alpha \ \mid \ \Pi X.\alpha \tag{1.1}$$

where ϕ is an **SO** formula over σ, $s \in \mathbb{N}$, x is a first-order variable, and X is a second-order variable.

Let $\mathcal{A}$ be a structure over a vocabulary σ, v be a first-order assignment for $\mathcal{A}$, and V be a second-order assignment for $\mathcal{A}$. Then the evaluation of a **QSO** formula α over $(\mathcal{A}, V, v)$ is defined as a function $[[\alpha]]$ that on input $(\mathcal{A}, V, v)$ returns a number in $\mathbb{N}$. The function $[[\alpha]]$ is recursively defined in Table 1.3. A **QSO** formula α is said to be a sentence if it does not have any free variable, that is every variable in α is under the scope of a usual or a quantitative quantifier. Notice that if α is a **QSO** sentence over σ, then for every structure $\mathcal{A}$, first-order assignments v_1, v_2 for $\mathcal{A}$ and second-order assignments V_1, V_2 for $\mathcal{A}$, it holds that $[[\alpha]](\mathcal{A}, v_1, V_1) = [[\alpha]](\mathcal{A}, v_2, V_2)$. Thus, in such a case we use the term $[[\alpha]](\mathcal{A})$ to denote $[[\alpha]](\mathcal{A}, v, V)$ for some arbitrary first-order assignment v and some arbitrary second-order assignment V for $\mathcal{A}$.

$$[[\phi]](\mathcal{A}, v, V) = \begin{cases} 1, & \text{if } \mathcal{A} \models \phi \\ 0, & \text{otherwise} \end{cases}$$

$$[[s]](\mathcal{A}, v, V) = s$$

$$[[\alpha_1 + \alpha_2]](\mathcal{A}, v, V) = [[\alpha_1]](\mathcal{A}, v, V) + [[\alpha_2]](\mathcal{A}, v, V)$$

$$[[\alpha_1 \cdot \alpha_2]](\mathcal{A}, v, V) = [[\alpha_1]](\mathcal{A}, v, V) \cdot [[\alpha_2]](\mathcal{A}, v, V)$$

$$[[\Sigma x.\alpha]](\mathcal{A}, v, V) = \sum_{a \in A} [[\alpha]](\mathcal{A}, v[a/x], V)$$

$$[[\Pi x.\alpha]](\mathcal{A}, v, V) = \prod_{a \in A} [[\alpha]](\mathcal{A}, v[a/x], V)$$

$$[[\Sigma X.\alpha]](\mathcal{A}, v, V) = \sum_{B \subseteq A^{\mathsf{arity}(X)}} [[\alpha]](\mathcal{A}, v, V[B/X])$$

$$[[\Pi X.\alpha]](\mathcal{A}, v, V) = \prod_{B \subseteq A^{\mathsf{arity}(X)}} [[\alpha]](\mathcal{A}, v, V[B/X])$$

Table 1.3: The semantics of **QSO** formulas.

For example, the problem of counting cliques in a graph $\mathcal{G}$ can be expressed as $[[\alpha_{clique}]](\mathcal{G})$, where $\alpha_{clique} = \Sigma X.\forall x \forall y(X(x) \wedge X(y) \wedge x \neq y) \rightarrow E(x, y)$.

The syntax of **QSO** formulas is divided in two levels: the first level is composed by **SO** formulas over σ and the second level is made by quantitative operators of addition and multiplication. By parameterizing one or both of these levels, we define different sets of formulas and different counting classes. We denote **ΣQSO** the fragment of **QSO** formulas where first- and second-order products are not allowed. **ΣQSO**(**FO**) is the set of **ΣQSO** formulas obtained by restricting ϕ in (1.1) to be an **FO** formula. In general, **ΣQSO**(**L**) formulas are **ΣQSO** formulas obtained by restricting ϕ in (1.1) to be in **L**.

Definition 1.20. *We say that $f \in \Sigma\mathsf{QSO}(\mathsf{FO})$ if there exists a* **ΣQSO**(**FO**) *formula α such that $[[\alpha]](\mathcal{A}) = f(\mathcal{A})$, for every ordered finite structure $\mathcal{A}$ over σ.*

Remark 1.5. **ΣQSO**(**FO**) *denotes a set of logical formulas, whereas $\Sigma\mathsf{QSO}(\mathsf{FO})$ denotes a class of functions. For every logic* **L**, *we can define a corresponding class of functions as above and denote it by L.*

Theorem 1.9 ([15])**.** *$\#\mathsf{P} = \Sigma\mathsf{QSO}(\mathsf{FO})$ over finite ordered structures.*

Definition 1.21. *(a) A formula α in* $\mathbf{\Sigma QSO(L)}$ *is in* **L***-prenex normal form (***L***-PNF) if α is of the form $\Sigma\overrightarrow{X}.\Sigma\overrightarrow{x}.\phi(\overrightarrow{X},\overrightarrow{x})$, where $\overrightarrow{X}$ and $\overrightarrow{x}$ are sequences of second- and first-order variables, repsectively, and $\phi(\overrightarrow{X},\overrightarrow{x})$ is in* **L**.

(b) A formula α in $\mathbf{\Sigma QSO(L)}$ *is in* **L***-sum normal form (***L***-SNF) if α is of the form $\sum_{i=1}^{n}\alpha_i$, where each α_i is in* **L***-PNF.*

Proposition 1.11. ([15, Proposition 5.1]). *Every formula in* $\mathbf{\Sigma QSO(L)}$ *can be written in* **L***-SNF.*

Corollary 1.1. *Every formula in* #P *can be written in* **FO***-SNF.*

1.5 A guided tour to this thesis

Many of the results presented in Chapters 2 and 3 are from Eleni Bakali's PhD thesis [22]. However, they are presented here since they are tightly connected to further results and discussion developed within these two chapters. They are also necessary to follow several results and arguments presented later on in the context of this thesis.

Chapter 2 is about the first TotP-complete problems under parsimonious reductions. The first natural TotP-complete problem, denoted by #Tree-Monotone-Circuit-Sat, is the problem of counting the number of satisfying assignments for a circuit that is monotone under a specific partial order on the set of all assignments $\{0,1\}^n$. As a corollary, the problem of counting the satisfying assignments for monotone circuits, where the monotonicity is defined by a partial order which is part of the input, is hard with respect to both exact and approximate computation.

Among others, a particularly interesting problem, which is called Size-of-Subtree, turns out to be TotP-complete. It was first introduced by Knuth [105] as the problem of estimating the size of a backtracking tree, that is the tree produced by a backtracking procedure. Although this problem has been studied under various perspectives (see Section 2.3 for the related references), its worst case complexity had remained open. This was partially due to the fact that, unlike most problems in counting complexity, it does not relate to logical formulas or to common graph theory problems, and cannot even be expressed as a constraint satisfaction problem. TotP-completeness of Size-of-Subtree under parsimonious reductions settled its complexity status.

Combining TotP-completeness of Size-of-Subtree with an algorithm of Knuth [105] for

the problem, it can be shown that any problem in TotP admits a polynomial-time randomized approximation algorithm the error of which depends on the imbalance of the Turing machine's computation tree that corresponds to the problem. We also provide exponential-time hardness results for the problem Size-of-Subtree under variants of the exponential-time hypothesis (ETH) which were first presented in [13]. In particular, we show that under #ETH there is no deterministic algorithm that solves Size-of-Subtree in subexponential time and under the randomized version of ETH, rETH, there is no randomized algorithm that approximates Size-of-Subtree within a $(1 \pm \frac{1}{4})$-multiplicative factor in subexponential time.

Finally, we discuss TotP-completeness of a satisfiability problem, denoted by #Clustered-Monotone-Sat. An instance of #Clustered-Monotone-Sat exhibits interesting properties that allow navigation among its solutions and enumeration of them with only a polynomial delay from a solution to the next one. Since this problem is AP-interreducible with #Sat, approximating #Sat in polynomial time is equivalent to approximating it on instances for which efficient navigation among solutions is possible. This supplements the current knowledge regarding the hardness of #Sat, where the scattering of solutions of a typical **CNF** formula and the consequent lack of navigability between them is, often provably, considered as the reason of failure of many algorithms for the problem. Thus any further research regarding the unconditional (im)possibility of approximating #Sat can be restricted to instances for which an efficient algorithm for navigation between solutions exists.

In Chapter 3 the relationship between the class TotP and FPRAS is examined. Most problems proven so far to admit an fpras belong to TotP, so a reasonable question is whether FPRAS $\subseteq$ TotP. Of course, problems in FPRAS have a decision version in BPP, so if P $\neq$ BPP this is probably not the case. Therefore, a more realistic goal is to determine assumptions under which the conjecture FPRAS $\subseteq$ TotP might be true. The world can be considered as depicted in Figure 1.2, where #BPP denotes the class of problems in #P with a decision version in BPP.

This picture was refined in [22] by proving that (a) FPRAS $\not\subseteq$ TotP unless P = RP, which means that proving FPRAS $\subseteq$ TotP would be at least as hard as proving P = RP and (b) TotP $\not\subseteq$ FPRAS if and only if RP $\neq$ NP. This not only indicates that there are problems in TotP which do not admit an fpras, it also shows that TotP versus FPRAS is essentially an equivalent formulation of the RP versus NP problem.

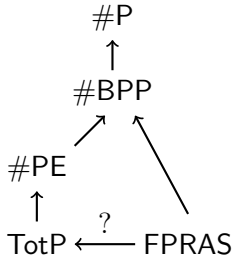


Figure 1.2: Relation of FPRAS to counting classes below #P.

Furthermore, FPRAS lies between two classes that can be seen as counting versions of RP and BPP. Moreover, the class FPRAS′ defined here as the subclass of FPRAS with zero error probability when the function value is zero, lies between two classes, namely $\#\mathsf{RP}_1$ and $\#\mathsf{RP}_2$, which can both be seen as counting versions of RP.

Both $\#\mathsf{RP}_1$ and $\#\mathsf{RP}_2$ seem to be interesting since they contain some natural counting problems the decision versions of which are in RP, but have not been shown to lie in P. However, the two classes do not coincide unless RP = NP. It was shown in [22] that $\#\mathsf{RP}_1 \subseteq \mathsf{FPRAS}$ which implies that #NonZerosForPIT and #CompositenessWitnesses, two counting problems defined here, have an fpras.

Chapters 2 and 3 also contain Notes Sections—Sections 2.6 and 3.4, respectively—in which we outline where the relevant results were first presented.

In Chapter 4 we introduce classes defined by properties of functions that count the total number of computation paths. They can be seen as tot-counterparts of classes defined via functions that count the number of accepting paths or via gap functions [66]. At first, we were interested in the class that gives information about the least significant bit of a TotP function, namely $\oplus_{\mathsf{tot}}\mathsf{P}$. According to Toda's astonishing result, $\oplus\mathsf{P}$, PP, and #P are, in a sense, at least as expressive as the polynomial hierarchy. This also holds for the class TotP, since $\mathsf{P}^{\mathsf{TotP}} = \mathsf{P}^{\#\mathsf{P}}$. Here we explore the power of the tot-definable class $\oplus_{\mathsf{tot}}\mathsf{P}$, which turns out to be exactly equal to the class $\oplus\mathsf{P}$. In an analogous manner, we define the classes $\mathsf{Gap}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Few}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Mod}_{\mathsf{ktot}}\mathsf{P}$, $\mathsf{SP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{WP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{C}_{=\mathsf{tot}}\mathsf{P}$, and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$. We compare them with their analogs, definable by either #P or GapP functions. We show that each one of them coincides with its counterpart (definable by either a #P or a GapP function), except for $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$ and $\mathsf{Few}_{\mathsf{tot}}\mathsf{P}$, which are both equal to the class P.

Moreover, building upon a result by Curticapean [52] that $\mathsf{C}_{=}\mathsf{P}$ has a complete problem definable by the TotP function #PerfMatch, we show that WPP and PP have also complete problems definable by #PerfMatch. This, together with [52], can be seen as an alternative proof of the fact that the classes $\mathsf{C}_{=\mathsf{tot}}\mathsf{P}$, $\mathsf{WP}_{\mathsf{tot}}\mathsf{P}$, and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$ coincide with $\mathsf{C}_{=}\mathsf{P}$, WPP, and PP, respectively.

Finally, we examine the complexity of the promise problem DiffPerfMatch$_{=g}$, which is essentially the problem of determining whether the number of perfect matchings of two graphs is either zero or equal to a given value. This problem is WPP-complete and also SPP-hard. First, we give a lower bound for this problem under the randomized exponential-time hypothesis

(rETH). Second, due to the hardness of DIFFPERFMATCH$_{=g}$ for SPP, a positive result for #PERFMATCH would have consequences for any problem in SPP. A seminal problem in the class is GRAPHISOMORPHISM [106, 17, 19], a well-known intermediate NP problem, the exact complexity of which is still a major open question in computational complexity.

In Chapter 5 we focus on logical characterizations of robust subclasses of TotP. We build upon previous work in descriptive complexity of counting problems [132, 15]. Arenas et al. [15] raised the question of defining classes in terms of descriptive complexity that capture either TotP or robust subclasses of TotP, as one of the most important open questions in the area. A robust class of counting problems needs either to have a natural complete problem or to be closed under addition, multiplication, and subtraction by one [15]. In specific, TotP satisfies both the above properties. Note that #P and #PE are not closed under subtraction by one (under assumptions about NP) [121, 123].

In particular, we define two subclasses of TotP, namely ΣQSO(Σ_2-2SAT) and #Π_2-1VAR, via logical characterizations; we show robustness of both classes by providing natural complete problems for them. Namely, we prove that the problem #DISJ2SAT of computing the number of satisfying assignments to disjunctions of **2SAT** formulas is complete for ΣQSO(Σ_2-2SAT) under parsimonious reductions. This reveals that problems hard for ΣQSO(Σ_2-2SAT) under parsimonious reductions cannot admit an fpras unless RP = NP. We also prove that #MONSAT is complete for #Π_2-1VAR under product reductions. Our result is the first completeness result for #MONSAT under reductions stronger than Turing. Notably, the complexity of #MONSAT has been investigated in [84, 27] and it is still open whether it is complete for TotP, or for a subclass of TotP under reductions for which the class is downwards closed. Although, #Π_2-1VAR is not known to be downwards closed under product reductions, our result is a step towards understanding the exact complexity of #MONSAT. This first part of Chapter 5 was presented in [24].

In the second and last part of Chapter 5 we define the logic $\mathbf{R\Sigma Q_{SOE}}$ and prove that it captures TotP. Arenas et al. defined recursion on the quantitative level. They extended **QSO** with function symbols over first-order variables and defined a notion of least fixed point over functions, which allows counting. Analogously to the decision classes FO(LFP) and FO(TC) that capture P and NL, respectively, they defined two classes, namely RQFO(FO) and TQFO(FO), that capture FP and #L, respectively [15]. They also mentioned that if a specific operator defined using recursion, namely the path operator, included free second order variables, it would probably give alternative ways to capture FPSPACE, or even #P.

Here we introduce function symbols defined on second order variables and a *polynomially bounded* partial fixed point operator over functions along the lines of [112]. We believe that this is of independent interest and could help to capture superclasses of #P. Then, we pose restrictions to our language, since the resulting logic is more expressive than needed.

In Chapter 6 we examine two problems that belong to $\#\mathsf{RP}_2$, i.e. the class of problems in #P that have a decision version in RP. Both these problems, namely #Exact Matchings and #Blue-Red Matchings, are generalizations of counting perfect matchings in a graph. To start with, we focus on #Exact Matchings, the problem of counting perfect matchings with exactly k red edges in a graph that has both black and red edges. We give a hardness result for the problem with respect to parameterized complexity: #Exact Matchings is #W[1]-hard.

Computing #Exact Matchings in $K_{3,3}$-free graphs has already been resolved by Mulmuley et al. who described an NC algorithm for it [119]. We show that #Exact Matchings can also be computed in polynomial time when it is restricted to K_5-free graphs. Then we turn our attention to #Exact Matchings in bipartite graphs. We have not concluded yet whether this problem has an fpras or it is inapproximable.

Motivated by a technique developed recently by Anari et al. [6], which was used to show that counting k-matchings in planar graphs has an fpras, we study matching polynomials generated by different matching problems: counting perfect matchings / all matchings / k-matchings / exact matchings. The technique of [6] connects the mixing time of a Markov chain defined on the set of matchings with the zero-free region of the generating polynomial of the problem. So we are interested in regions of the complex plane in which polynomials related to #Exact Matchings have no roots.

We consider this last chapter a starting point for dealing with the complexity of #Exact Matchings in bipartite graphs.

Every chapter includes a Discussion of results Section. Chapter 7 contains open questions.

1.6 Notes

A survey on classes of counting problems the decision version of which is easy—either in the class P or in RP—can be found in [23], which contains an overview of the work on such classes under different viewpoints: Turing machine based definitions, decsriptive complexity, approximability of them, their characterizations as classes of interval size functions, and so on.

Chapter 2

TotP-complete problems

Many counting problems are known to be #P-complete under Turing reductions. The seminal result of Valiant [149] states that computing the permanent of a matrix with entries in $\{0, 1\}$, which is equivalent to counting perfect matchings in bipartite graphs, is such a problem. Since then, many other problems that are hard for #P under Turing reductions have been determined. Especially, many dichotomy theorems for several counting classes have been proven [50, 40, 41, 43]. A dichotomy theorem for a class of counting problems states necessary and sufficient conditions under which a problem is either in FP or #P-hard under Turing reductions.

As we already mentioned, TotP and #P are equivalent under Turing reductions. Among the known #P-hard problems under Turing reductions, the ones that belong to TotP, are also TotP-complete under this kind of reductions. Some examples are #DNF,#MonSat, #BiPerfMatch (or Permanent), #IS, and #2Sat. In fact, under Turing reductions, #P is even equivalent to SpanL, a subclass of TotP defined in [9], every problem of which admits an fpras [14]. These observations support the fact that Turing reductions blur structural differences between counting classes [103].

On the contrary, parsimonious reductions can distinguish counting complexity classes inside #P. For one reason, by preserving the number of solutions, they also preserve the existence of a solution. Second, by being AP reductions, they also preserve approximability of counting problems; if a class with approximable problems was proven to be parsimonious-equivalent with #P, then all the problems in #P would be approximable, which in turn would imply RP = NP (see Theorem 3.1). Third, classes with a Turing machine-based definition, like #P and TotP, are closed under parsimonious reductions.

The above properties and observations justify the urge to study completeness under par-

simonious reductions. A first problem that is complete for TotP under such reductions is, of course, the generic one:

Definition 2.1. *(The generic* TotP*-complete problem)* f_{gen}.
Input: $(M, x, 1^t)$, *where* M *is a (binary) TM,* $x \in \{0,1\}^*$, *and* $t \in \mathbb{N}$.
Output: $f_{gen}(M, x, 1^t)$:=*(the total number of computation paths of* M *on input* x, *of length at most* t*) - 1.*

In this chapter we discuss the first natural—in the sense that no Turing machine is provided as input to their instances—TotP-complete problems under parsimonious reductions. The following results can be found in [25, 13] and in Eleni Bakali's PhD thesis [22]. We have included a Notes Section (Section 2.6) which describes in detail where each result first appeared.

2.1 The problem #Tree-Monotone-Circuit-Sat

The first problem that was shown to be TotP-complete under parsimonious reductions is #Tree-Monotone-Circuit-Sat. We provide the definitions of the tree partial order and the problem #Tree-Monotone-Circuit-Sat and the proofs of its TotP-hardness and its membership to TotP. The result can then be extended to monotone circuits with respect to other partial orders.

A tree is called (a) *binary* if every vertex has at most two children, (b) *full binary* if every vertex has either zero or two children, (c) *complete binary* if every level of it, except possibly the last one, is completely filled and all vertices in the last level are as far left as possible, (d) *perfect binary* if it is full binary and complete, i.e. all interior vertices have two children and all leaves have the same depth.

Definition 2.2. *We define the* tree *partial order, denoted by* $\leq_{tree}$, *on* $\mathbb{N}$ *as the reflexive, transitive, and antisymmetric binary relation, such that if* $y = 2x + 1$ *or* $y = 2x + 2$ *then* $x \leq_{tree} y$.

The tree partial order can be represented graphically on the plane by drawing a node for each natural number and connecting the node of a number with the nodes of its immediate successors. The resulting graph is the tree depicted in Figure 2.1 (note that labeling of the edges in this figure is not related to the partial order defined above), denoted by $T_{\mathbb{N}}$. The root of the tree is labeled with 0, its children are labeled with 1 and 2, and so on. Note that $x \leq_{tree} y$ if and only if y is some descendant of x on $T_{\mathbb{N}}$. So, by enumerating the nodes of the

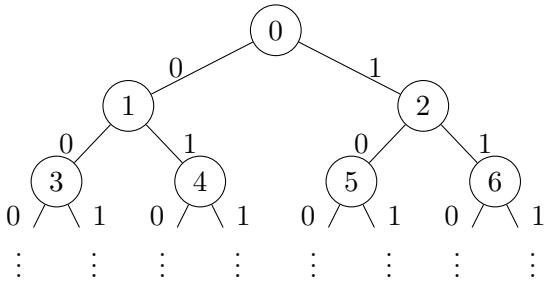


Figure 2.1: The infinite perfect binary tree $T_{\mathbb{N}}$.

infinite binary tree by the left to right 'breadth first search' (BFS) enumeration, we obtain a representation of the tree partial order.

Let $d_{T_{\mathbb{N}}}(u, v)$ denote the length of the path connecting u and v in $T_{\mathbb{N}}$. We keep $T_{\mathbb{N}}$ in mind in order to define some useful mappings.

Definition 2.3. *The following functions are defined :*

1. $\mathsf{path} : \mathbb{N} \to \{0,1\}^*$. *It maps n to the binary string that describes the path that starts from the root of $T_{\mathbb{N}}$ and ends at the vertex with label n. For example,* $\mathsf{path}(3) = 00$, $\mathsf{path}(9) = 010$, $\mathsf{path}(0) = \varepsilon$, *where ε is the empty string.*

2. $\mathsf{num} : \{0,1\}^* \to \mathbb{N}$. *It is is defined as the inverse mapping of* path.

3. $\mathsf{bin_k} : \{0, 1, \ldots, 2^k - 1\} \to \{0,1\}^k$. *It maps a vertex $n \in \{0, 1, \ldots, 2^k - 1\}$ of $T_{\mathbb{N}}$ to its binary representation padded with leading zeroes, so as to have length k. For example,* $\mathsf{bin_6}(3) = 000011$, $\mathsf{bin_4}(9) = 1001$, *and* $\mathsf{bin_3}(9)$ *is not defined.*

In addition, $\mathsf{bin_k^{-1}}$ is the inverse of $\mathsf{bin_k}$. For simplicity, we slightly abuse notation and use bin and bin^{-1}, when the length of the binary representation is clear from the context. For a vertex u of $T_{\mathbb{N}}$, $\mathsf{path}(u)$ is the concatenation of labels of edges connecting 0 with u as shown in Figure 2.1.

The functions path, num, $\mathsf{bin_k}$, and $\mathsf{bin_k^{-1}}$ are polynomial-time computable. This can be seen clearly by their equivalent definitions below.

Proposition 2.1. *(a) Let, for a binary string s,* $\mathrm{number}(s)$ *be its value in* $\mathbb{N}$.

$$\mathsf{num}(s) = \mathrm{number}(1; s) - 1,$$

where ; *denotes string concatenation.*

(b) The function path *can be computed recursively:*

- $\mathsf{path}(0) = \varepsilon$
- $\mathsf{path}(k) = \mathsf{path}\left(\lceil \frac{k}{2} \rceil - 1\right) ; \overline{\mathrm{parity}}(k)$

where $\overline{\mathrm{parity}}(k) = \begin{cases} 1 & , \textit{if } k \textit{ even} \\ 0 & , \textit{if } k \textit{ odd} \end{cases}$ *and* ; *denotes string concatenation.*

Definition 2.4. *If we restrict* $\leq_{tree}$ *on* $\{0,1,\ldots,2^k-1\}$ *and apply* $\mathsf{bin_k}$*, we obtain a partial order on* $\{0,1\}^k$*, which, abusing notation, we also denote by* $\leq_{tree}$*.*
T_k *denotes the binary tree representing* $\leq_{tree}$ *on* $\{0,1\}^k$*.*

For example the complete binary tree T_3 depicted in Figure 2.2 represents the order $\leq_{tree}$ on $\{0,1\}^3$.

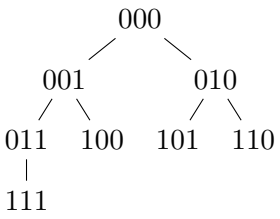


Figure 2.2: The complete binary tree T_3.

Now the problem #Tree-Monotone-Circuit-Sat can be defined as follows.

Let C_n denote a Boolean circuit (see [16, p. 107] for a formal definition) with n input gates and let $C_n(z)$ be the output of C_n on input $z \in \{0,1\}^n$.

Definition 2.5. *We call a Boolean circuit* C_n *non-increasing with respect to* $\leq_{tree}$ *if for every* $x, y \in \{0,1\}^n$*,* $x \leq_{tree} y$ *implies that* $C_n(x) \geq C_n(y)$*.*

Definition 2.6. #Tree-Monotone-Circuit-Sat*, abbreviated to* #TMC*.*
Input: A Boolean circuit C_n*, non-increasing with respect to* $\leq_{tree}$*.*
Output: $\#\mathrm{TMC}(C_n) := |\{y \in \{0,1\}^n : C_n(y) = 1\}|$*, i.e. the number of satisfying assignments for* C_n*.*

2.1.1 TotP-hardness of #Tree-Monotone-Circuit-Sat

As we discussed in Remark 1.4, for any NPTM M and input x, we have that

$$tot_M(x) = \#(\text{all paths of } M \text{ on input } x) - 1 = \#(\text{branchings of } T_{M(x)})$$

where $T_{M(x)}$ denoted the binary computation tree of M.

The sequence of nondeterministic choices of a computation of M can be represented as a binary string y (a left branching corresponds to '0' and a right branching to '1'). When we write $M(x,y)$ we refer to the output of the Turing machine M on input x and nondeterministic choices

y. Specifically, $M(x, y) = 1$ if M accepts x with nondeterministic choices y and $M(x, y) = 0$ otherwise.

#TMC was shown to be TotP-hard by reducing the computation of any function $h \in \mathsf{TotP}$ to #TMC. The key idea is the following. By definition, there is an NPTM M such that for any input x, $h(x) = tot_M(x)$. Let $T_{M(x)}$ denote the corresponding computation tree. Consider extending $T_{M(x)}$ to a perfect binary tree $S_{M(x)}$ with the same height, so that all leaves of the original $T_{M(x)}$ tree and all their descendants are labeled as 'halting'. Therefore $h(x) =$ #(branching vertices of $T_{M(x)}$) = #(non-'halting' vertices of $S_{M(x)}$).

Given an $h \in \mathsf{TotP}$ and an input x, we can construct a circuit C, non-increasing with respect to $\leq_{tree}$, such that the number of accepting inputs of C equals $h(x)$. To do that, we use a bijection between inputs of C and paths from the root to vertices of $S_{M(x)}$. C accepts an input if and only if the corresponding path ends at a non-'halting' vertex of $S_{M(x)}$, which in turn corresponds to a branching vertex of $T_{M(x)}$.

Theorem 2.1. *For any $h \in \mathsf{TotP}$, it holds that $h \leq^{\mathrm{p}}_{\mathrm{pars}}$ #TMC.*

Proof. Let $h \in \mathsf{TotP}$, and let M be the corresponding NPTM with computation binary tree $T_{M(x)}$. Recall that for every input x, $h(x) = tot_M(x) =$ #(branchings of $T_{M(x)}$). Let p be a polynomial bounding the running time of M, thus the height of $T_{M(x)}$ is at most $p(|x|)$.

We construct an NPTM M' such that for every instance x of h:

(i) $T_{M'(x)}$ is a perfect binary tree of height $p(|x|) + 1$.

(ii) #(accepting paths of $M'(x)$) = #(branchings of $T_{M(x)}$).

(iii) For $y_1, y_2 \in \{0,1\}^{p(|x|)+1}$, if $y_1 \leq_{tree} y_2$, then $M'(x, y_1) \geq M'(x, y_2)$.

In order to describe M' we make use of the functions path and bin defined in Definition 2.3.

The operation of M' on input x proceeds as follows:

1. Guess a binary string y of length $p(|x|) + 1$. Let $n_y = \mathsf{bin}^{-1}(y)$.
2. Compute $z = \mathsf{path}(n_y)$.
3. Simulate M on input x and nondeterministic choices z.
   - If the simulation reaches a halting state of M (possibly using only a prefix of z), then output 0.

- If the simulation uses all bits of z without reaching a halting state of M, output 1.

We show that the aforementioned properties (i), (ii), and (iii) hold.

(i) The computation tree of M' is a perfect binary tree of height $p(|x|)+1$, since the only nondeterministic choices are made in Step 1 (Step 3 is deterministic).

(ii) The number of accepting paths of M' equals the number of branchings of M, since M' outputs 1 if and only if z corresponds to a computation path of M ending at a branching; recall that bin and path are bijective.

(iii) To prove the third property, it suffices to show that for all y_1, y_2 such that $y_1 \leq_{tree} y_2$ if $M'(x, y_1) = 0$ then $M'(x, y_2) = 0$. If $y_1 \leq_{tree} y_2$, then $z_1 = \mathsf{path}(\mathsf{bin}^{-1}(y_1))$ is a prefix of $z_2 = \mathsf{path}(\mathsf{bin}^{-1}(y_2))$. This means that whenever M' simulates M with nondeterministic choices determined by z_2, it first passes through the same states as when it simulates M with nondeterministic choices determined by z_1. So, $M'(x, y_1) = 0$ means that the simulation of M reaches a halting state using (some of) the bits of z_1. Thus the remaining bits of z_2 are ignored and 0 is returned, therefore $M'(x, y_2) = 0$.

In order to complete the proof, we have to construct for each input x of h a circuit C_n^x with $n = p(|x|)+1$ input gates, that simulates the computation of M' on input x, i.e. for all $y \in \{0,1\}^n$, $C_n^x(y) = M'(x, y)$.

It is well known that a construction of a circuit that simulates a Turing machine can be done in polynomial time (see e.g. [124, pp. 171–172]) and the size of the circuit is quadratic to the running time of the Turing machine.

C_n^x is non-increasing w.r.t. $\leq_{tree}$ since M' has this property (due to (iii)). Thus, we have that $|\{y \in \{0,1\}^n : C_n^x(y) = 1\}| = \#acc_{M'}(x) = tot_M(x)$, i.e. $\#\mathrm{TMC}(C_n^x) = h(x)$ so the reduction is parsimonious. □

2.1.2 Membership of #Tree-Monotone-Circuit-Sat in TotP

To show that #Tree-Monotone-Circuit-Sat belongs to TotP, an NPTM can be constructed such that for every non-increasing circuit w.r.t. $\leq_{tree}$, the number of branchings of its computation tree is equal to the satisfying assignments of the circuit.

Theorem 2.2. $\#\mathrm{TMC} \in \mathsf{TotP}$.

Proof. We are going to construct an NPTM M such that for every monotone circuit C_n:

$$\#\text{TMC}(C_n) = \#(\text{branchings of } T_{M(C_n)}).$$

Since C_n is non-increasing with respect to $\leq_{tree}$, M can be constructed as follows.

1. M starts with checking whether the assignment corresponding to 0 is unsatisfying, so it first checks whether $C_n(0^n) = 0$. If this is the case, it halts. Otherwise, M chooses nondeterministically one of the successors of 0^n with respect to $\leq_{tree}$, i.e. either $0^{n-1}1$ corresponding to the natural number 1 or $0^{n-2}10$ corresponding to 2 and proceeds recursively with this one.

2. If M reaches an assignment $y \in \{0,1\}^n$ which corresponds to a leaf of T_n, then in the case of $C_n(y) = 0$, M again halts. Otherwise, it makes a nondeterministic choice between two different computation paths. At each path, M just halts without doing anything else.

Note that the depth of the computation of M is polynomial in n and every binary branching of its computation tree corresponds to a satisfying assignment of C_n. □

Note that #TMC is a promise problem, since it is not known how to check efficiently whether a circuit is non-increasing w.r.t. $\leq_{tree}$. To resolve this issue, we can simply extend the function #TMC on non-valid inputs, i.e. circuits that are not monotone w.r.t. $\leq_{tree}$, to be equal to $tot_M(x)$, where M is the NPTM which emerges from the membership of #TMC in TotP (on valid inputs).

2.1.3 Extension to monotone circuits with respect to other partial orders

The particular choice of the functions path and num, and the directly related partial order $\leq_{tree}$, is not the only possible one to yield a TotP-complete problem. It suffices to encode strings of bounded but possibly *different* length, corresponding to paths of the NPTM of a TotP function, as strings of *equal* length, corresponding to inputs to the final circuit. That is, we need a family of bijections $\{E_k\}_{k\in\mathbb{N}}$, where $E_k : \bigcup_{i=0}^{k-1}\{0,1\}^i \cup \{0^k\} \to \{0,1\}^k$. Each such encoding implies a partial order $\leq_k^*$ on $\{0,1\}^k$ for every k.

Fix such a family $\{E_k\}_{k\in\mathbb{N}}$ and let #A be the problem of counting the number of accepting inputs of a circuit monotone w.r.t. $\leq_m^*$, where m corresponds to the number of input gates of the circuit. #A can be proven to be TotP-complete by modifying the proofs of this section, provided that $\{E_k\}_{k\in\mathbb{N}}$ has the following properties.

- To show that #A is TotP-hard we need E_m^{-1} to be easily computable, so that the reduction takes polynomial time. The reduction is parsimonious since E_m is bijective. For example, simple padding would be a simpler solution, but inadequate, since it is not 1-1.
- To show that #A is in TotP, it must be easy to find the *minimum element* in $\{0,1\}^m$ with respect to $\leq_m^*$, i.e. to calculate $E_m(\varepsilon)$, where ε is the empty string. Furthermore, for every $x \in \{0,1\}^m$, it must be easy to compute the set of its *immediate successors* w.r.t. $\leq_m^*$. We note that if E_m^{-1} is easily computable, then these properties become equivalent to E_m being easily computable.

In conclusion, the proofs given above work if we choose an encoding that is an easily computable and easily invertible bijection. Although an encoding which is simple and natural was chosen (in the sense that it does not involve elaborate error correcting codes e.t.c., it is just a BFS enumeration in binary representation), we stress that for every encoding with the aforementioned properties, such as the binary representation of the depth-first-search enumeration, the corresponding problem is also TotP-complete.

This fact serves as a starting point for someone who needs to prove TotP-completeness for some other family of circuits (e.g. for monotone circuits under the standard notion of monotonicity). To that end it would not be necessary to design a reduction from scratch. It would suffice to design a—probably elaborate—encoding of the nodes of a tree (such as an error correcting code) with the aforementioned desired properties.

2.1.4 The case of the partial order being part of the input

In mathematics a monotone function is a function between two ordered sets that preserves or reverses the order of its domain. A circuit, as a function from $\{0,1\}^n$ to $\{0,1\}$, can be or not be monotone if we equip $\{0,1\}^n$ with some partial order. For example, in computer science literature ([16, Chapter 14.3]), the standard notion of a monotone circuit is defined with respect to the partial order induced by the edges of the boolean hypercube: a $y \in \{0,1\}^n$ is an immediate successor of $x \in \{0,1\}^n$ if y is obtained by flipping a bit of x from 0 to 1 and a circuit is monotone with respect to this partial order if its output does not decrease when we change the value of an input gate from 0 to 1.

We saw that for some specific partial orders, counting satisfying assignments to the corresponding monotone circuits is TotP-hard. This has an immediate consequence on the hardness of an analog of #TMC, where a partial order is given as part of the input.

Corollary 2.1. *On input a natural number $n \in \mathbb{N}$, a partial order p on $\{0,1\}^n$, and a circuit C with n input gates, which is monotone with respect to p, the problem of counting the number of satisfying assignments of C is #P-complete under both Turing and AP reductions.*

Proof. Since #Tree-Monotone-Circuit-Sat is TotP-complete under parsimonious reductions, it is also #P-complete under Turing and AP reductions. Then, the result is obtained by generalization. □

Note that for an arbitrary partial order, the problem might not belong to TotP, but it is easy to see that it always belongs to #P, hence the conclusion.

Beware that the above corollary does not imply that the problem is hard for each specific partial order, since the partial order is considered as part of the input. Theoretically speaking there may exist some partial orders for which the corresponding problem admits either an exact efficient algorithm, or an fpras.

2.2 Problems related to partially ordered sets

In this section, we are interested in problems of computing the size of a subset of a partially ordered set satisfying some additional properties.

An upper-set of a partially ordered set $(U, \leq)$ is a subset of U which is upwards closed. An uper-set of $(U, \leq)$ is called principal if it is the smallest upper set containing a particular element of U. Lower sets are defined similarly as downwards closed subsets of U.

Computing the size of the maximum lower set of $(\{0,1\}^n, \leq_{tree})$, all elements of which are accepted by a given circuit C_n, is TotP-complete under parsimonious reductions. This implies that computing the size of the maximum lower set of an arbitrary given partially ordered set, all elements of which share an arbitrary given property P, is TotP-hard under parsimonious reductions. It also implies that computing the size of a principal upper set is TotP-hard under parsimonious reductions, thus also #P-hard under both Turing and AP reductions.

Definition 2.7. *Let $(U, \leq)$ be a partially ordered set.*

(a) A subset $V \subseteq U$ is called a lower set if for all $y, x \in U$, ($y \in V$ and $x < y$) $\Rightarrow x \in V$.

(b) A subset $V \subseteq U$ is called an upper set if for all $y, x \in U$, ($y \in V$ and $x > y$) $\Rightarrow x \in V$.

(c) The smallest upper (resp. lower) set containing $x \in U$ is called principal and it is denoted

by $\uparrow x$ *(resp.* $\downarrow x$*).*

Definition 2.8. *Let* C_n *be a circuit with* n *input gates. We will call a subset* V *of* $\{0,1\}^n$ C_n-accepting *if for all* $x \in V$, $C_n(x) = 1$.

Definition 2.9. MAX-LOWER-SET-SIZE.
Input: A circuit C_n *with* n *input gates.*
Output: The size of the maximum C_n*-accepting lower set w.r.t.* $\leq_{tree}$.

Theorem 2.3. *The problem* MAX-LOWER-SET-SIZE *is* TotP*-complete under parsimonious reductions.*

Since every boolean circuit can be considered to compute a property, i.e. a predicate $P : \{0,1\}^n \to \{0,1\}$, where $P(x) = 1$ iff x has the property P, we get the following corollary.

Corollary 2.2. *Let* $(U, \leq)$ *be a partially ordered set and* P *a property (equivalently a predicate* $P : U \to \{0,1\}$*). The problem of computing the size of the maximum lower set of* U*, all elements of which share property* P*, is* TotP*-hard.*

Corollary 2.3. *Let* $(U, \leq)$ *be a partially ordered set and an element* $x \in U$*. The problem of computing the size of the principal upper set* $\uparrow x$ *is* TotP*-hard.*

Proof. Take an instance C_n for MAX-LOWER-SET-SIZE. We set $U = \{0,1\}^n$, $x = 0^n$, and we construct a partial order $\leq_p$ on $\{0,1\}^n$ such that $\uparrow 0^n$ equals to the maximum C_n-accepting lower set w.r.t. $\leq_{tree}$.

We define $\leq_p$ to be the reflexive, transitive, and antisymmetric binary relation such that $x \leq_p y$ iff $[(y = 2x + 1$ or $y = 2x + 2)$ and $C_n(x) = C_n(y)]$.

The result is immediate if we compare the above partial order with the tree partial order in Definition 2.2; observe that we get $\leq_p$ if we break every chain of subsequent elements in $(\{0,1\}^n, \leq_{tree})$ at the points where $x \leq_{tree} y$ and $C_n(x) \neq C_n(y)$. □

2.3 The problem SIZE-OF-SUBTREE

The problem SIZE-OF-SUBTREE was initially introduced by Knuth as the problem of estimating the size of a backtracking procedure's tree [105]. There is a number of papers that study this problem from many perspectives and algorithms which succeed in many special cases and practical instances. Knuth provided a probabilistic algorithm practically useful, but with an

exponential error in the worst case. Modifications and extensions of Knuth's algorithm have been presented and experimentally tested in [130, 47, 104]. However, they exhibit no significant improvement on worst case instances. There are also many heuristics and experimental results for the problem restricted to special backtracking algorithms, or special instances to them, see e.g. [36] and references therein. Surprisingly there exist fpras for random models of the problem [71, 147]. Also quantum algorithms for the problem have been studied [10]. Stockmeyer provided unconditional lower bounds for the problem under a model of computation which is not equivalent to the Turing machine, namely a variant of the (non-uniform) decision tree model [140]. Stockmeyer's result implies (unconditionally) that a family of algorithms based on a certain type of sampling methods cannot yield an fpras for the problem.

Size-of-Subtree is the estimation of the size of a tree given in succinct representation. We consider the input tree S to be a subtree of the complete binary tree T_k of height k, containing the root of T_k, or some other given vertex of T_k.

By succinct representation, we mean that the description of the tree is not polynomial to its size, but rather to its height. For example, a truth table $A_S : V(T_k) \to \{0,1\}$ that implements the indicator function of S, where $A_S(u) = 1$ if and only if vertex u of T_k belongs to S, could also be an instance to the above problem. But it would not be interesting since the size of that tree could be trivially computed in linear time with respect to the size of the input. The interesting cases are those where the indicator function A_S is implemented succinctly, e.g. by a circuit of size polynomial to the number of input gates, i.e. polynomial in $|u|$ where u is a vertex of T_k. Another example of such a succinct representation is by giving as input a backtracking procedure that generates this tree. The latter case was in fact the origin of this problem [105]. The related complexity theory question is whether we can estimate the size of such a tree without traversing it exhaustively.

Clearly, the circuit that emerges from our main reduction in Theorem 2.1 constitutes a succinct representation of the tree containing the branching vertices of the computation tree of any problem in TotP on any input. So it turns out that the problem is TotP-complete under parsimonious reductions. This resolves its worst case complexity, since it implies that it is #P-complete under both Turing and AP reductions.

Definition 2.10. Size-of-Subtree*, abbreviated to* f_{ss}*.*
Input: A polynomially computable predicate $A : T_k \to \{0,1\}$ *and a vertex* u *of* T_k*.*
Output: The size of the maximum subtree $S \subseteq A^{-1}(1)$ *with root* u*.*

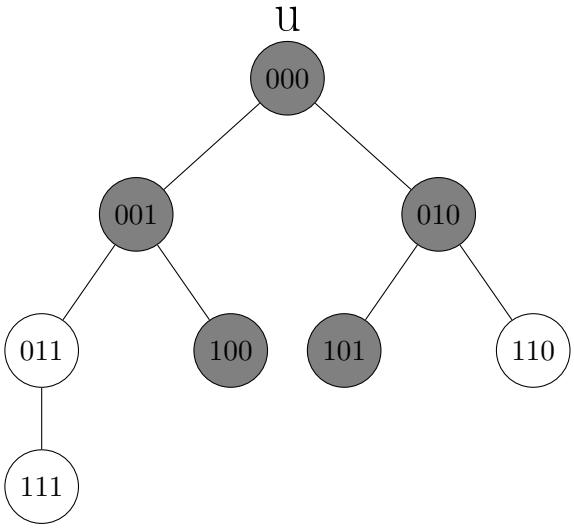


Figure 2.3: An instance of Size-of-Subtree. It holds that $u = 000$, $k = 3$, predicate A takes the value 1 on the gray vertices, and the output of the problem is equal to 5.

Note that height k can be implied by $|u|$. Equivalently, the number k could be given as part of the input. An instance of Size-of-Subtree is given in Figure 2.3.

Theorem 2.4. *f_{ss} is* TotP*-complete.*

Proof. (a) TotP-hardness: Let $g \in$ TotP and let x be some input for g. We will map x to an input for f_{ss}. Since $g \in$ TotP, there exists an NPTM M_g that runs in time $q(n)$ for some polynomial q, s.t. #(branchings of $T_{M_g(x)}$) $= g(x)$. We construct a deterministic polynomial-time TM $M'_{g;x}$ s.t. $M'_{g;x}(y) = 1$ iff the nondeterministic string y represents a computation path of M_g on input x that stops at a branching. The input to f_{ss} is $z = (M'_{g;x}, 0^{q(|x|)})$. So $f_{ss}(z) =$ #(branchings of $T_{M_g(x)}$) $= g(x)$. The details of the reduction are inherited from the proof of Theorem 2.1, thus omitted.

(b) TotP-membership: f_{ss} has an easy decision because for every input $y = (A, u)$, $f_{ss}(y) = 0$ iff $A(u) = 0$.

Also f_{ss} is self-reducible. Let v_1, v_2 be the children of u. It holds that if $A(u) = 1$, then

$$f_{ss}(A, u) = f_{ss}(A, v_1) + f_{ss}(A, v_2) + 1,$$

otherwise $f_{ss}(A, u) = 0$.

The reduction terminates after at most $n = |u|$ steps and the size of the sub-instances is polynomially related to the size of the initial instance, so the conditions of self-reducibility are satisfied. □

2.3.1 Hard instances of Size-of-Subtree

As was already mentioned, Size-of-Subtree is easy for many practical instances and for some random models of it. On the other hand there are instances for which Size-of-Subtree is hard to approximate; in other words, it cannot be efficiently approximated unless RP = NP.

Definition 2.11. *Let $n_1, n_2, n \in \mathbb{N}$ be such that $n_1 + n_2 = n$. We call (n_1, n_2)-tree a binary tree of height at most n that consists of a perfect binary tree of height n_1, from at most one leaf of which hangs another perfect binary tree of height n_2.*

Theorem 2.5. *The problem* Size-of-Subtree *restricted to $(n/2, n/2)$-trees, cannot be approximated in polynomial time within a multiplicative factor $(1 \pm \frac{1}{4})$, unless* RP = NP.

Proof. We show that Size-of-Subtree on $(n/2, n/2)$-trees is inapproximable by reducing USat—the satisfiability problem restricted to formulas that have at most one satisfying assignment—to it.

Consider the standard NPTM M for Sat: the respective computation tree on input ϕ is a perfect binary tree of height n. Each leaf corresponds to a truth assignment to the n variables and to the output '1' if and only if the assignment is satisfying for ϕ. We define another NPTM M' that simulates M and halts whenever M returns '0', else M' makes another set of n nondeterministic choices and halts.

Unsatisfiable formulas correspond to a perfect binary tree of height n (case 1), while formulas with one satisfying assignment correspond to a tree of height $2n$ consisting of a perfect binary tree of height n from one leaf of which hangs another perfect binary tree of height n (case 2).

If we could approximate the size of any $(n/2, n/2)$-tree, for any n, within a multiplicative factor $(1 \pm \frac{1}{4})$, then we could distinguish between cases 1 and 2, and thus we could distinguish between satisfiable and unsatisfiable formulas of USat. By the Valiant-Vazirani Theorem [151], which probabilistically reduces Sat to USat, we get the conclusion. □

2.3.2 On the exponential-time complexity of Size-of-Subtree

TotP-completeness of Size-of-Subtree implies exponential-time hardness results for the problem. The lower bounds shown below are relative to variants of the exponential-time hypothesis (ETH) [89], which states that 3Sat—the satifiability problem on **3CNF** formulas—cannot be

solved in subexponential time. In particular, the variants we need here are the randomized version rETH, introduced in [45], and the counting version #ETH introduced in [56]. Let n denote the number of variables of the input formula. The three aforementioned variants of the exponential time hypothesis are as stated below.

ETH: There is no deterministic algorithm that can decide 3Sat in time $\exp(o(n))$.

rETH: There is no randomized algorithm that can decide 3Sat in time $\exp(o(n))$, with error probability at most 1/3.

#ETH: There is no deterministic algorithm that can compute exactly #Sat in time $\exp(o(n))$.

Note that the hypothesis rETH is stronger than ETH, which in turn is stronger than #ETH, in the sense that rETH $\Rightarrow$ ETH $\Rightarrow$ #ETH. The weaker an assumption is, the stronger a hardness result is. On the other hand, stronger assumptions sometimes yield tighter lower bounds.

For Size-of-Subtree let N be the height of the perfect binary tree, subtree of which is the input tree.

Theorem 2.6.

(a) Under rETH there is no randomized algorithm that computes Size-of-Subtree *exactly in time* $\exp(o(N))$.

(b) Under #ETH there is no deterministic algorithm that computes Size-of-Subtree *exactly in time* $\exp(o(N/\log N))$.

(c) Under rETH there is no randomized algorithm that approximates Size-of-Subtree *within* $(1 \pm \frac{1}{4})$*-multiplicative factor in time* $\exp(o(N))$.

Proof. (a) Under rETH there is no randomized algorithm that computes #2Sat exactly in time $\exp(o(m))$, where m is the number of clauses of the imput formula ϕ [56].

Consider the following NPTM M for #2Sat. First keep a list l that indicates whether variable i has been considered. At the beginning set $l(v) \leftarrow$ 'not considered', for every variable v. Halt if all variables are set to 'considered'.

Begin with the first clause. If the first variable v_1^1 of that clause is not yet considered, check if ϕ is satisfiable with v_1^1 set either to true or false. If both hold, choose nondeterministically to set v_1^1 to one of these two values. Else, set it to the unique satisfying value, or halt if neither

holds. Then set $l(v_1^1) \leftarrow$ 'considered' and set $\phi \leftarrow$ (ϕ with v_1^1 fixed to the chosen value). Proceed with the second variable of the first clause, if it is not considered so far (and the computation has not halted yet). Repeat the same procedure for the rest of the clauses, until a halting state is reached.

M yields a computation tree that branches at most twice for each clause, thus it has height at most $2m$. Add a dummy path to M (on instances with at least one solution) and the original definition of TotP is satisfied; the number of branchings of M's computation tree equals $\#2\text{Sat}(\phi)$.

If Size-of-Subtree could be computed by a randomized algorithm in time $\exp(o(N))$, then by the above reduction the same would hold for $\#2\text{Sat}$, which contradicts rETH.

(b) Under #ETH there is no deterministic algorithm that computes the Permanent exactly in time $\exp(o(m/\log n))$, where n is the size of the input matrix $A_{n\times n}$, and m is the number of non-zero elements in $A_{n\times n}$ [56]. Equivalently, there is no such algorithm for the exact computation of #BiPerfMatch on input a bipartite graph with n vertices in each side and m edges.

Consider the NPTM M for #BiPerfMatch of Example 1.1. M has a computation tree that branches at most once for each edge, thus it has height at most m.

W.l.o.g. we assume that $n \leq m$, otherwise a perfect matching trivially does not exist. Thus if Size-of-Subtree could be computed in time $\exp(o(N/\log N))$, then by the above reduction #PerfMatch could be computed in time $\exp(o(m/\log n))$, which contradicts #ETH.

(c) Under rETH there is no randomized algorithm that decides 3USat, i.e. the satisfiability problem restricted to **3CNF** formulas that have at most one satisfying assignment, in time $\exp(o(n))$, where n is the number of variables of the input formula [45].

Using the reduction of Theorem 2.5 a formula with n variables is reducible to a tree of height $2n$. Thus, similarly to the proof of Theorem 2.5, if we could probabilistically approximate Size-of-Subtree within $(1 \pm \frac{1}{4})$-multiplicative factor in time $\exp(o(N))$, we could also probabilistically decide 3USat in time $\exp(o(n))$, which contradicts rETH. □

In Theorem 2.6, although (c) implies (a), the proof of (a) shows also that a subexponential solution to Size-of-Subtree yields a subexponential solution to $\#2\text{Sat}$, which is not implied by (c).

A more recent result by Curticapean [54] implies a tighter lower bound for Size-of-

Subtree under #ETH, than the one of Theorem 2.6(b).

Theorem 2.7. *Under #ETH there is no deterministic algorithm that computes* Size-of-Subtree *exactly in time* $\exp(o(N))$.

Proof. Under #ETH there is no deterministic algorithm that computes #BiPerfMatch exactly in time $\exp(o(n))$ on graphs with n vertices and $\mathcal{O}(n)$ edges [54].

Consider the NPTM M for #BiPerfMatch of Example 1.1 of height $\mathcal{O}(n)$ as in the above proof. If Size-of-Subtree could be computed in time $\exp(o(N))$, then #BiPerfMatch could be computed in time $\exp(o(n))$, which contradicts #ETH. □

In fact by similar fine-grained reductions between Size-of-Subtree and other problems in TotP, we can obtain the following results.

Theorem 2.8. *If* Size-of-Subtree *can be probabilistically computed in time* $\exp(o(N))$, *then* Permanent, #IS, *and* #2Sat *can be probabilistically computed in time* $\exp(o(m))$, *where* m *is the number of non-zero entries of the input matrix, the number of edges of the input graph, and the number of clauses of the input formula, respectively.*

Proof. The proof is analogous to those of Theorem 2.6(a) and (b). □

2.3.3 Implications on the approximability of TotP

A negative result

First, the following negative result about TotP is a corollary of Theorem 2.6(c).

Corollary 2.4. *Under rETH,* $\mathsf{TotP} \not\subseteq \mathsf{FPRAS}$.

Proof. By Theorem 2.6(c), under rETH there is no subexponential randomized algorithm that approximates Size-of-Subtree within $(1 \pm \frac{1}{4})$-multiplicative approximation factor. So, under rETH there is no fpras for Size-of-Subtree. □

A positive result

Second, we obtain a simple efficient randomized algorithm for TotP, which is an algorithm described by Knuth in [105].

It is known that for every problem in #P there is a trivial polynomial-time randomized approximation algorithm, such that the expected value of its output equals the exact number of solutions, but its error is exponential in the worst case ([77, chapter 6.2.2]). For any function in #P with corresponding NPTM M, this algorithm chooses a polynomial-size sample of computation paths of M uniformly at random and outputs the number of accepting paths in the sample over the size of the sample, multiplied by the number of all paths.

For the problem Size-of-Subtree there is another algorithm due to Knuth [105], no better in worst case. However, we have some additional information in the case of an exponential error. Given an input tree, the variance of this algorithm depends on the level of imbalance of the tree. A tree is considered to be *perfectly balanced* if all its vertices of the same depth have the same degree. In fact, the algorithm works for any finite tree, not necessarily binary. It chooses a random path of the tree, counts the number of vertices and children of the vertices along this path, and estimates the size of the tree to be equal to the size of the perfectly balanced tree that contains such a path.

The following theorem is a consequence of Theorem 2 of [105] and the TotP-completeness of Size-of-Subtree.

Theorem 2.9. *For any $f \in$ TotP and every input x, there is a polynomial-time randomized approximation algorithm that the expected value of its output equals $f(x)$. The variance of the output of this algorithm is given by*

$$Var(D) = \sum_{v \in S} \frac{1}{d_0^v d_1^v \dots d_{m-1}^v} \left(|S_{v_1}| - |S_{v_2}|\right)^2 ,$$

where S is the computation tree of the NPTM corresponding to f, m is the depth of vertex v in S, d_i^v, $0 \leq i \leq m-1$, is the number of children of the i^{th} vertex in the path from the root to v, and S_{v_i}, $i = 1, 2$, is the subtree of S rooted at the i^{th} child of v, denoted v_i. If vertex v has either 0 or 1 child, then its contribution to the above sum is 0.

Proof. The result comes from applying equation (13) of [105] and by simplifying terms. We used the facts that the probability of v to be encountered is $1/(d_0^v d_1^v \dots d_{m-1}^v)$ and that the tree S is binary, so every vertex has either $0, 1$ or 2 children. □

Note that the variance of the above algorithm depends on the amount of imbalance in S. For example, if S was perfectly balanced, then the variance would be 0. On the other hand, if S consisted of the root, a single path of height $n-1$ on the left, and a full complete binary tree of height $n-1$ on the right, then the variance would be exponentially large.

2.4 The problem #Clustered-Monotone-Sat

When #Tree-Monotone-Circuit-Sat is reduced to #Sat, the resulting formula has some interesting properties, that unlike an instance of #Sat in general, allow us to navigate among solutions and enumerate them one by one with only a polynomial delay from one solution to another. The problem of counting satisfying assignments of such formulas is called #Clustered-Monotone-Sat and it is a TotP-complete problem under parsimonious reductions. As a result, #Sat is AP-interreducible with #Clustered-Monotone-Sat.

This means that approximating the number of satisfying assignments of any **CNF** formula reduces to approximating the number of satisfying assignments of a formula for which navigation between solutions is feasible and conversely. So, regarding the approximability of #Sat, it suffices to either give an fpras for #Clustered-Monotone-Sat, or to prove unconditional inapproximability that holds for clustered-monotone formulas as well (and, in fact, that holds even when an algorithm for finding or counting solutions of certain prefix, or for navigating between solutions, is provided along with the input formula). Moreover, the already known conditional hardness of #Sat extends to the case of #Clustered-Monotone-Sat as well. Both these problems do not admit an fpras unless RP = NP (see Theorem 3.1).

This is also relevant to the fact that approximate counting for self-reducible problems is equivalent to uniform sampling from the set of solutions and sampling is usually accomplished by Markov chains on the set of solutions, for which navigability is an essential property [92]. From the study of random Sat certain phase transition phenomena have been discovered [46, 118, 109] and rigorously proven [1, 2]. Specifically when we consider typical random **CNF** formulas of density a (where $a = (\#\text{clauses})/(\#\text{variables})$), as we increase the density, we observe the following phenomena. First of all, the number of solutions gradually decreases. Secondly, there is a critical value a_1 such that a random formula is satisfiable with high probability for $a < a_1$ and unsatisfiable for $a > a_1$ [46, 118, 2]. There is also another critical value a_2 such that for $a > a_2$ the solution space of a typical formula shutters to a large number of clusters of fixed variables [109, 1], so that (a) it is hard to find even one solution to the formula and (b) given any cluster of solutions it is hard to find a different one.

Those results indicate which the hard instances of Sat are. It has also been proven that this shuttering phenomenon is the reason of failure of the already known algorithms for Sat (and #Sat) that are based on Markov chains, because this shuttering translates to ergodicity breaking [4], since the state space of the corresponding Markov chains is not connected. TotP-

completeness of #Clustered-Monotone-Sat implies that #Sat remains hard even when navigation among solutions is possible, i.e. when ergodicity holds.

Definition 2.12. *1. For a* **3CNF** *formula* ϕ *and* $k \in \mathbb{N}$ *we define* $f_\phi^k : \{0,1\}^k \to \mathbb{N}$ *such that* $f_\phi^k(a) = \#$*(satisfying assignments of* ϕ *with prefix* a*) for* $a \in \{0,1\}^k$.

2. A **3CNF** *formula* ϕ *with* n *variables is called* k*-clustered-monotone for some* $k \leq n$*, if for every* $a, b \in \{0,1\}^k$ *such that* $a \leq_{tree} b$, $f_\phi^k(a) = 0$ *implies* $f_\phi^k(b) = 0$.

Definition 2.13. #Clustered-Monotone-Sat*, abbreviated to* #CMS.
Input: $y = (\phi, k, M)$*, where* ϕ *is* k*-clustered monotone and* M *is the description of a function such that* $M \in \mathsf{FP}$ *and* $M(a, \phi) = f_\phi^k(a)$.
Output: $\#\mathrm{CMS}(y) :=$ *#(satisfying assingnments of* ϕ*).*

Theorem 2.10. #CMS *is* TotP*-complete.*

Proof. (a) TotP-hardness. We reduce #TMC to #CMS.

Let C_k be a Boolean circuit, non-increasing w.r.t. $\leq_{tree}$, which has k input gates, one output gate, and m more (inner) gates. We will map C_k to an input y of #CMS.

In the well known reduction of Circuit-Sat to 3SAT that can be found at [16, p. 111], a formula ϕ on $n = k + m + 1$ variables is constructed, so that each variable corresponds to a gate of C_k and

$$\#(\text{satisfying assignments for } \phi) = \#(\text{satisfying assignments for } C_k).$$

We can construct ϕ in such a way that its first k variables $x_1, \ldots, x_k$ correspond to the k input gates of C_k. For every x_i, $i \in k+1, \ldots, n$, we construct (at most four) clauses that force x_i to be set to the same value as the output of the corresponding gate of C_k on input $(x_1, \ldots, x_k)$.

We next describe an algorithm which given a formula ϕ constructed in the above way and an assignment $a \in \{0,1\}^k$ to the first k variables of ϕ, decides whether there is any satisfying assignment of ϕ with a as prefix. If such an assignment exists then it will be unique since each one of the variables $x_{k+1}, \ldots, x_{k+m+1}$ will have to be equal to the output of the corresponding gate of C_k on input a. Thus this algorithm computes f_ϕ^k in polynomial time.

The algorithm essentially simulates the original circuit C_k. It begins knowing the assigned values of $x_1, \ldots, x_k$ and it computes the values corresponding to the output and the inner gates of C_k. While there are variables which have not been assigned a truth value yet, it loops through them. If such a variable x_i corresponds to a gate g_i of C_k that its inputs correspond to variables

that have already been assigned some value by the algorithm, then the output of g_i is assigned to x_i. It is guaranteed by the construction of ϕ that such a variable x_i always exists. Finally, when values have been assigned to all variables, the algorithm outputs whether ϕ is satisfied by that assignment. If an arbitrary formula ϕ that does not correspond to a circuit is given as input to the algorithm, the algorithm will always terminate and output either `FALSE` or `TRUE`. For an exact description see Algorithm 1.

Algorithm 1

procedure A$((a_1, \ldots, a_k), \phi)$ ▷ ϕ has n variables
  for $i \leftarrow 1$, k **do** ▷ The first k variables of ϕ correspond
    $x_i \leftarrow a_i$ ▷ to the input gates of C_k
  end for
  for $i \leftarrow k+1$, n **do** ▷ The rest of the variables correspond
    $x_i \leftarrow$ `UNDEFINED` ▷ to the rest of the gates of C_k
  end for
  $count \leftarrow k$ ▷ #(variables of ϕ we have computed so far)
  while $count < n$ **do**
    $prevcount \leftarrow count$
    for each $i \in \{k+1, \ldots, n\}$ and **each** clause c that occurs in ϕ **do**
      if $x_i =$ `UNDEFINED` and (x_i appears exactly once in c) and
      (no variable set to `UNDEFINED` appears in c except for x_i) and
      (the disjunction of literals of c that do not contain x_i is `FALSE`) **then**
        if c contains the literal x_i **then**
          $x_i \leftarrow$ `TRUE`
        else
          $x_i \leftarrow$ `FALSE`
        end if
        $count \leftarrow count + 1$
      end if
    end for
    if $count = prevcount$ **then**
      return `FALSE` ▷ ϕ has not been constructed by our reduction
    end if
  end while
  return $\phi(x_1, \ldots, x_n)$
end procedure

The running time of Algorithm 1 is polynomial in the size of the input. Let M be the

description of a TM that implements this algorithm. We map C_k to the input $y = (\phi, k, M)$ for #CMS.

(b) TotP-membership: Let $y = (\phi, k, A)$ be an input to #CMS. We describe an NPTM with $\#\text{CMS}(y) + 1$ leaves, thus showing that the problem lies in TotP.

The description of the NPTM is as follows. If $A(0^k, \phi) = 0$ then HALT, else choose nondeterministically one of the following: 1. HALT, 2. EXPLORE(0^k).

The nondeterministic process EXPLORE(a) for $a \in \{0,1\}^k$ is defined as follows:

1. Let S be the set of children of a with respect to $\leq_{tree}$. For every $b \in S$, simulate $A(b, \phi)$. Let $S' = \{b \in S \mid A(b, \phi) > 0\}$. Let $\rho = |S'| \in \{0, 1, 2\}$.

2. Make a branching with $A(a, \phi) + \rho$ branches. For each one of the first $A(a, \phi)$ branches, HALT. Each one of the rest corresponds to some $b \in S'$. Run EXPLORE(b) for the corresponding b. □

Since any TotP-complete problem under parsimonious reductions is also #P-complete under AP reductions, we have the following corollary, with all its implications mentioned in the beginning of this section.

Corollary 2.5. #Sat *is AP-interreducible with* #Clustered-Monotone-Sat.

This can be generalized to the following. #Sat is AP-interreducible with the problem of counting satisfying assignments of a **CNF** formula for which an algorithm that allows efficient navigation between satisfying assignments exists. By efficient we mean that (a) it is easy to find one solution and (b) the graph that connects two solutions whenever the navigation algorithm can go from the one to the other in polynomial time, is strongly connected and of polynomial width.

2.5 Discussion of results

First and foremost, determining the first TotP-complete problems under parsimonious reductions gave some representative problems of TotP, namely the problems #Tree-Monotone-Circuit-Sat, Max-Lower-Set-Size, Size-of-Subtree, and #Clustered-Monotone-Sat.

The TotP-complete problem Size-of-Subtree can express a well-known computational problem, first introduced by Knuth and studied for years by researchers as discussed in Section 2.3. This result has two bright sides. First, the complexity of this well-studied problem was refined. Second, a simple efficient randomized algorithm was obtained for all problems in TotP. Last but not least, the reductions from several (not only counting) problems to Size-of-Subtree established exponential-time lower bounds results for this problem in Subsection 2.3.2. One of there results states that under the randomized ETH, there is no subexponential randomized algorithm that approximates Size-of-Subtree within $(1 \pm \frac{1}{4})$-multiplicative factor. We can also infer the following weaker result from the previous statement: If there is no subexponential randomized algorithm for Sat, then $\mathsf{TotP} \not\subseteq \mathsf{FPRAS}$, which is also weaker than the result of Corollary 3.1 (shown later on, in Chapter 3): If $\mathsf{NP} \neq \mathsf{RP}$, then $\mathsf{TotP} \not\subseteq \mathsf{FPRAS}$.

Also as discussed in Section 2.4, TotP-completeness of a special case of #Sat, namely #Clustered-Monotone-Sat, has as a result that #Sat $\equiv_{\text{AP}}$ #Clustered-Monotone-Sat. Other special cases of #Sat, namely #2Sat and #MonSat, are known to be AP-interreducible with #Sat as well. But AP-interreducibility of #Sat to #Clustered-Monotone-Sat has its own value, since it implies that designing an fpras for the problem of counting the number of satisfying assignments of an arbitrary **CNF** formula reduces to designing such an algorithm for the same problem on a **CNF** formula for which navigation between solutions is feasible and conversely. Thus, regarding the approximability of #Sat, it suffices to either give an fpras for #Clustered-Monotone-Sat or prove unconditional inapproximability that holds for clustered-monotone formulas as well.

2.6 Notes

The first TotP-complete problems under parsimonious reductions were first presented in [25] and in Eleni Bakali's PhD thesis [22]. An extended version of these results was recently published [13].

Most results of this chapter were presented in [22]. The results of Subsections 2.1.4 and 2.3.2 first appeared in [13]. Theorem 2.7, which states a stronger lower bound for Size-of-Subtree under #ETH (compared to Theorem 2.6(b)), was first proven here, in Subsection 2.3.2.

For algorithmic results on TotP-complete problems we refer the reader to [20] and [21], where an additive approximation algorithm for TotP problems and Markov chains for sampling among solutions to TotP-complete problems are given and discussed, respectively.

Chapter 3

Relationship between TotP and the class of approximable counting problems

The subject of this chapter is related to the following meta-question: If we can distinguish efficiently between $f(x) = 0$ and $f(x) \neq 0$, can we also achieve an efficient approximation of $f(x)$? And what about the converse: does an efficient approximation of a counting function imply that we can decide efficiently whether the function is non-zero?

As discussed in Chapter 1, if the counting version of an NP-complete decision problem has an fpras, then RP = NP [59]. In Proposition 1.2 was shown that a problem that admits an fpras, has a decision version in BPP. Remarkably, most approximable problems have a decision version in P and are also self-reducible, so they belong to TotP. Hence, a first worthwhile question to answer is whether FPRAS $\subseteq$ TotP holds. This is answered by Corollary 3.5. For the converse inclusion TotP $\subseteq$ FPRAS, we know that TotP contains problems that are inapproximable unless RP = NP, such as #IS [57]. Corollary 3.1 states that TotP $\subseteq$ FPRAS is in fact, equivalent to RP = NP.

Most of this chapter's results were shown in Eleni Bakali's PhD thesis [22]. Our presentation follows [24]. In the Discussion of results and Notes Sections of the current chapter—Sections 3.3 and 3.4, respectively—we refer the reader to the works where these results were first presented.

3.1 On #P versus FPRAS

We start with the relationship between #P and FPRAS, the class of approximable #P problems (see Definition 1.6). A corollary of Stockmeyer's Theorem (Theorem 1.3) is that RP = NP

implies the existence of an fpras for every function in #P. Inversely, if any function in #P has an fpras, then NP $\subseteq$ BPP, which in turn implies that RP = NP. These observations are summarized in the following theorem.

Theorem 3.1. #P $\subseteq$ FPRAS *if and only if* RP = NP.

Proof. If #P $\subseteq$ FPRAS, then NP $\subseteq$ BPP by Proposition 1.2 and so SAT $\in$ BPP. Since SAT is self-reducible, if it can be solved by a probabilistic Turing machine with bounded probability error, then it can be solved by a probabilistic Turing machine with no false positives, by computing a satisfying assignment [124, problem 11.5.18].

Assume that RP = NP holds. By Theorem 1.3, for any function $f \in$ #P, there exists a polynomial-time (in $|x|$ and in $1/\varepsilon$) algorithm which, using a Σ_2^{P} oracle, approximates f within ratio $(1 \pm \varepsilon)$. Since RP = NP is true, it holds that $\Sigma_2^{\mathsf{P}} = \mathsf{RP}^{\mathsf{RP}} \subseteq$ BPP [157]. Finally it is not hard to verify that the aforementioned deterministic polynomial-time algorithm with access to a BPP oracle, can be replaced by an fpras, that simulates the oracle calls itself. □

Remark 3.1. *Note that the second part of the proof of Theorem 3.1 can be obtained using the Valiant–Vazirani bisection technique (Theorem 1.2) instead of Stockmeyer's Theorem. Using a similar proof, Dyer et al. showed that* #IS *is AP-interreducible to* #SAT *even when it is restricted to graphs with maximum degree 25 [59, Theorem 4].*

3.2 On TotP versus FPRAS

First, note that TotP contains almost all counting problems known to have an fpras, such as #DNF [98], #BIPERFMATCH [93], and #NFA [14]. However it also contains problems that are inapproximable unless RP = NP. For example, #MONSAT [114] and #IS [59] belong to TotP. Of course, it includes all problems that are AP-interreducible with #BIS, the problem of counting independent sets in bipartite graphs, which we believe that they form an intermediate class with respect to approximability, i.e. neither they have an fpras nor they are AP-interreducible with #SAT [59].

So it comes natural that an analog of Theorem 3.1 holds for TotP as well. To state the theorem, we first define an ancillary class, namely the class FPRAS′.

Definition 3.1. $f \in$ FPRAS′ *if* $f \in$ FPRAS *and there exists an* fpras *for* f*, as defined in Definition 1.5, which also outputs* $\widehat{f(x)} = 0$ *with probability 1 in the case of* $f(x) = 0$.

Proposition 3.1. *If* $f \in \mathsf{FPRAS}'$, *then* $L_f \in \mathsf{RP}$.

Proof. By Proposition 1.2 there is a BPP algorithm for L_f. By the definition of FPRAS' and the fact that $f \in \mathsf{FPRAS}'$, the BPP algorithm for L_f can be modified so it has a zero error in the case of 'no' instances. So $L_f \in \mathsf{RP}$. □

3.2.1 (Non)inclusion of TotP in FPRAS

The following result is, in part, a corollary of Theorem 3.1. It also states that if an fpras exists for a TotP problem, then it can be modified so that, for every $x \in \Sigma^*$, such that $f(x) = 0$, it outputs the correct value with probability 1.

Corollary 3.1. $\mathsf{TotP} \subseteq \mathsf{FPRAS}$ *if and only if* $\mathsf{TotP} \subseteq \mathsf{FPRAS}'$ *if and only if* $\mathsf{RP} = \mathsf{NP}$.

Proof. The proof of $\mathsf{RP} = \mathsf{NP} \Longrightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS}$ is completely analogous to that of $\mathsf{RP} = \mathsf{NP} \Longrightarrow \#\mathsf{P} \subseteq \mathsf{FPRAS}$ (see Theorem 3.1).

If $\mathsf{TotP} \subseteq \mathsf{FPRAS}$, then #IS has an fpras, which implies that $\mathsf{RP} = \mathsf{NP}$ [57].

Now we prove that $\mathsf{TotP} \subseteq \mathsf{FPRAS}$ iff $\mathsf{TotP} \subseteq \mathsf{FPRAS}'$. Suppose that $\mathsf{TotP} \subseteq \mathsf{FPRAS}$ and let $f \in \mathsf{TotP}$. Then $f \in \mathsf{FPRAS}$. We can modify the fpras for f so that it outputs $\widehat{f(x)} = 0$ with probability 1 if $f(x) = 0$. We can do this since $f(x) = 0$ can be determined in polynomial time. So, $f \in \mathsf{FPRAS}'$. The other direction is trivial, since $\mathsf{FPRAS}' \subseteq \mathsf{FPRAS}$. □

Corollary 3.2. $\#\mathsf{P} \subseteq \mathsf{FPRAS}$ *if and only if* $\mathsf{TotP} \subseteq \mathsf{FPRAS}$.

Proof. By Proposition 3.1 and Corollary 3.1. □

3.2.2 Classes of counting problems the decision version of which is in RP

The classes introduced in this subsection, which contain counting problems the decision version of which is in RP, were used in the study of the opposite inclusion, i.e. whether FPRAS is a subset of TotP.

If for a problem in #P, the corresponding counting machine has an RP behavior, i.e. either the majority of the paths are accepting or all paths are rejecting, then the decision version of the problem is in RP. The first class considered below, namely $\#\mathsf{RP}_1$, contains such problems. However, this seems to be a quite restrictive requirement. Therefore, a second class, which is called $\#\mathsf{RP}_2$ is defined next.

Definition 3.2. *Let M be an NPTM in normal form. We denote by p_M the polynomial, such that on any input $x \in \Sigma^*$, M makes $p_M(|x|)$ nondeterministic choices.*
$\mathcal{MR} = \{M \mid M$ *is an NPTM in normal form and for all* $x \in \Sigma^*$ *either* $acc_M(x) = 0$ *or* $acc_M(x) > \frac{1}{2} \cdot 2^{p_M(|x|)}\}$.

Definition 3.3. $\#\mathsf{RP}_1 = \{f \in \#\mathsf{P} \mid$ *there exists an* $M \in \mathcal{MR}$ *such that for all* $x \in \Sigma^*$, $f(x) = acc_M(x)\}$.

Definition 3.4. $\#\mathsf{RP}_2 = \{f \in \#\mathsf{P} \mid L_f \in \mathsf{RP}\}$.

Note that $\#\mathsf{RP}_1$, although restrictive, contains a counting version of one of the most representative problems in RP, for which no deterministic efficient algorithms are known.

Consider the polynomial identity testing problem, denoted by PIT: Given a polynomial $p(x_1, ..., x_n)$ over some field $\mathbb{F}$, decide whether p is not identically zero. [1] This algebraic problem captures many interesting and natural computational problems, such as testing equality of two bitstrings in a distributed setting and bipartite perfect matching. Determining the computational complexity of polynomial identity testing is considered one of the most important open problems in the mathematical field of Algebraic Computing Complexity. A probabilistic solution to it is based on the following lemma.

Lemma 3.1 (Schwartz–Zippel Lemma)**.** *Let $p(x_1, ..., x_n)$ be a polynomial of degree d over a field $\mathbb{F}$. Assume that p is not identically zero. Let $S \subseteq \mathbb{F}$ be any finite set. Then, if we pick $y_1, ..., y_n$ independently and uniformly from S,*

$$Pr[p(y_1, ..., y_n) = 0] \leq \frac{d}{|S|}.$$

Assume that $d < |\mathbb{F}|$. A well-known randomized polynomial time algorithm for PIT, picks $y_1, ..., y_n$ uniformly and independently from $\mathbb{F}$ (if $\mathbb{F}$ is infinite, from a suitably large finite subset of $\mathbb{F}$). If $p(y_1, ..., y_n) \neq 0$, the algorithm announces that p is not identically zero. In this case, the algorithm never makes an error. Otherwise, if $p(y_1, ..., y_n) = 0$, the algorithm announces that p is identically zero. In this case the Schwartz–Zippel lemma guarantees that the probability of error is at most $d/|\mathbb{F}| < 1$. So, PIT $\in$ RP. Interestingly, if PIT has a deterministic polynomial time algorithm then we would obtain new circuit lower bounds for the class NEXP [95], which would be a remarkable consequence in complexity theory.

[1] In the literature, polynomial identity testing is the problem of deciding whether two polynomials are identical or, equivalently, whether a polynomial is identically zero. Here we define PIT to be the equivalent problem of deciding whether a polynomial is *not* identical zero.

We define here a counting version of PIT. For a field $\mathbb{F}$, we denote by $\mathbb{F}_{3d}$ a set of $3d$ elements of $\mathbb{F}$, which can be chosen deterministically by a Turing machine.

Definition 3.5. #NonZerosForPIT.

Input: A polynomial $p(x_1, ..., x_n)$ of degree d over a field $\mathbb{F}$, such that $|\mathbb{F}| \geq 3d$.

Output: The number of points $(y_1, ..., y_n) \in \mathbb{F}_{3d}^n$ for which $p(y_1, .., y_n) \neq 0$.

Another problem in $\#\mathsf{RP}_1$ is the problem of counting the number of compositeness witnesses as defined by the Miller-Rabin primality test, on input an integer $n > 2$: n is written as $2^s \cdot d+1$, where $s, d \in \mathbb{N}^+$ and d is odd. An integer $0 < a < n$ is called a witness for the compositeness of n if (a) or (b) holds: (a) $a^{n-1} \not\equiv 1 \pmod{n}$,
(b) $a^{n-1} \equiv 1 \pmod{n}$ is true and both the following congruence relations hold:
$a^d \not\equiv 1 \pmod{n}$ and $a^{2^r \cdot d} \not\equiv -1 \pmod{n}$, for all $0 \leq r < s$.

Definition 3.6. #CompositenessWitnesses.

Input: An integer $n > 2$.

Output: The number of integers $a \in \mathbb{Z}_n^*$ that are witnesses for the compositeness of n.

A prime number has no such witnesses, whereas for a composite number n, at least half of the integers in $\mathbb{Z}_n^*$ are Miller-Rabin witnesses. Hence there exists an NPTM $M \in \mathcal{MR}$ that has as many accepting paths as the number of witnesses. Note that in this case the decision problem is in P: given a natural number n, it can be decided whether it is a prime number in deterministic polynomial time by the AKS algorithm [3].

$\#\mathsf{RP}_2$ contains natural counting problems as well. For example, two problems in $\#\mathsf{RP}_2$ are #Exact Matchings and #Blue-Red Matchings, which are counting versions of Exact Matching [125] and Blue-Red Matching [120], respectively.

Definition 3.7. #Exact Matchings.

Input: A graph $G = (V, E)$, a subset $E' \subseteq E$ and an integer k.

Output: The number of perfect matchings of G that contain exactly k edges in E'.

Definition 3.8. #Blue-Red Matchings.

Input: A graph $G = (V, E_{red} \cup E_{blue})$ and two integers w and B.

Output: The number of matchings of size at least B with at most w edges in E_{blue} (blue edges) and at most w edges in E_{red} (red edges).

Both Exact Matching and Blue-Red Matching belong to RP (in fact in RNC) as shown in [119, 120], respectively; however, it has remained open so far whether they can be solved

in deterministic polynomial time. Therefore, it is also open whether #Exact Matchings and #Blue-Red Matchings belong to TotP.

Proposition 3.2. *(a)* #NonZerosForPIT *and* #CompositenessWitnesses *belong to* $\#\mathsf{RP}_1$. *(b)* #Exact Matchings *and* #Blue-Red Matchings *belong to* $\#\mathsf{RP}_2$.

3.2.3 Unconditional inclusions

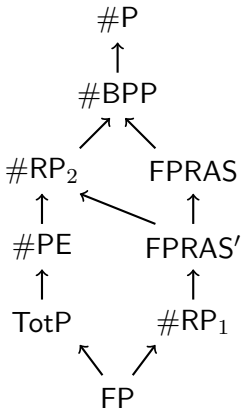


Figure 3.1: Unconditional inclusions.

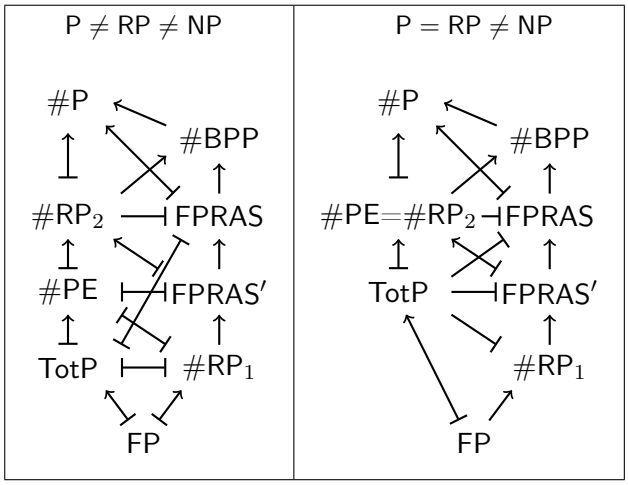


Figure 3.2: Conditional inclusions. The following notation is used: $A \to B$ denotes $A \subseteq B$, $A \dashv B$ denotes $A \not\subseteq B$, and $A \mapsto B$ denotes $A \subsetneq B$.

This subsection is about unconditional inclusions among the aforementioned classes. Subsection 3.2.4 explores possible inclusions under either the condition $\mathsf{P} \neq \mathsf{RP} \neq \mathsf{NP}$ or $\mathsf{P} = \mathsf{RP} \neq \mathsf{NP}$.

The results are summarized in Figures 3.1 and 3.2.

Theorem 3.2. *(a)* $\mathsf{FP} \subseteq \#\mathsf{RP}_1 \subseteq \#\mathsf{RP}_2 \subseteq \#\mathsf{P}$.
(b) $\mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{RP}_2$.

Proof. (a) Let $f \in \mathsf{FP}$. We will show that $f \in \#\mathsf{RP}_1$ by constructing an NPTM $M \in \mathcal{MR}$ such that on input x, $acc_M(x) = f(x)$. On input $x \in \Sigma^*$, M computes $f(x)$ and then it computes $i \in \mathbb{N}$ such that $f(x) \in (2^{i-1}, 2^i]$. M makes i nondeterministic choices $b_1, b_2, ..., b_i$. Each sequence $b = (b_1, ..., b_i) \in \{0,1\}^i$ determines a path. In particular, 0^i is the first path and $b = 1^i$ is the 2^i-th path. M returns yes on every path that is between the first and the $f(x)$-th path. So $acc_M = f(x)$ and since $f(x) > 2^{i-1}$, $M \in \mathcal{MR}$.

The other inclusions of both (a) and (b) are immediate from the definitions of the classes. □

For the class $\#\mathsf{RP}_1$ we have a guarantee that the number of accepting paths is polynomial with respect to the total number of paths. In fact, their ratio is bounded by a constant. In these cases there is an fpras for approximating the number of accepting paths using the Monte Carlo method. The following lemma is the heart of the Monte Carlo method.

Lemma 3.2 (Unbiased estimator)**.** *Let $A \subseteq B$ be two finite sets and let $p = \frac{|A|}{|B|}$. Assume we take m samples from B uniformly at random and let a be the number of them that belong to A. Then $\hat{p} = \frac{a}{m}$ is an unbiased estimator of p and if $m = \text{poly}(p^{-1}, \varepsilon^{-2}, \log(\delta^{-1}))$, then we have that*

$$\Pr[(1-\varepsilon)p \leq \hat{p} \leq (1+\varepsilon)p] \geq 1 - \delta.$$

Theorem 3.3. $\#\mathsf{RP}_1 \subseteq \mathsf{FPRAS}' \subseteq \#\mathsf{RP}_2$.

Proof. For the first inclusion, let $f \in \#\mathsf{RP}_1$. Then there exists an $M_f \in \mathcal{MR}$ such that for all x, $acc_{M_f}(x) = f(x)$. Let $q(|x|)$ be the number of nondeterministic choices of M_f and $p = \frac{f(x)}{2^{q(|x|)}}$. For any $\varepsilon > 0$ and $0 < \delta < 1$, we can compute an estimate $\hat{p}$ of p, by choosing $m = \text{poly}(p^{-1}, \varepsilon^{-1}, \log(\delta^{-1}))$ paths uniformly at random. Then we compute $\widehat{f(x)} = \hat{p} \cdot 2^{q(|x|)}$. By Lemma 3.2 we have that

$$\Pr[(1-\varepsilon)f(x) \leq \widehat{f(x)} \leq (1+\varepsilon)f(x)] \geq 1 - \delta.$$

If $f(x) \neq 0$, then $p > \frac{1}{2}$, so $m = \text{poly}(\varepsilon^{-1}, \log(\delta^{-1}))$ and so we obtain an fpras for f. If $f(x) = 0$ then $\widehat{f(x)} = 0$, hence $f \in \mathsf{FPRAS}'$.

The second inclusion is a straightforward corollary of Proposition 3.1. □

Corollary 3.3. $\#\mathsf{RP}_1 \subseteq \mathsf{FPRAS}' \subseteq \mathsf{FPRAS} \subseteq \#\mathsf{BPP}$.

Theorems 3.2 and 3.3 together with Theorem 1.7 are summarized in Figure 3.1.

Corollary 3.4. #NonZerosForPIT *and* #CompositenessWitnesses *belong to* FPRAS.

Proof. It is immediate from Proposition 3.2 and Theorem 3.3. □

3.2.4 Conditional inclusions (possible worlds) and consequences of FPRAS ⊆ TotP

Further relationships between the aforementioned classes can be determined with respect to P versus RP versus NP, so that two possible worlds inside #P emerge. All relationships proven below are demonstrated in Figure 3.2.

Corollary 3.5. *If* $\mathsf{FPRAS} \subseteq \mathsf{TotP}$ *then* $\mathsf{P} = \mathsf{RP}$.

Proof. If $\mathsf{FPRAS} \subseteq \mathsf{TotP}$, then $\#\mathsf{RP}_1 \subseteq \mathsf{TotP}$ and then for all $f \in \#\mathsf{RP}_1$, $L_f \in \mathsf{P}$. So if $A \in \mathsf{RP}$ via $M \in \mathcal{MR}$ then $\#acc_M \in \#\mathsf{RP}_1$ and thus $A = L_{\#acc_M} \in \mathsf{P}$. Thus $\mathsf{P} = \mathsf{RP}$. □

Corollary 3.6. *If* $\#\mathsf{RP}_1 = \#\mathsf{RP}_2$ *then* $\mathsf{RP} = \mathsf{NP}$.

Proof. By Theorem 3.3, if $\#\mathsf{RP}_1 = \#\mathsf{RP}_2$, then both are equal to FPRAS'. This would imply that $\mathsf{TotP} \subseteq \mathsf{FPRAS}' \subseteq \mathsf{FPRAS}$, since $\mathsf{TotP} \subseteq \#\mathsf{RP}_2$ by Theorem 3.2(b). Therefore, $\mathsf{RP} = \mathsf{NP}$ by Corollary 3.1. □

Theorem 3.4. *The inclusions depicted in Figure 3.2 hold under the corresponding assumptions on top of each subfigure.*

Proof. First note that intersections between any of the aforementioned classes are non-empty, because FP is a subclass of all of them. For the rest of the inclusions, we have the following.

- In the case of $\mathsf{P} = \mathsf{RP} \neq \mathsf{NP}$.
  - By definitions, $\#\mathsf{P} \subseteq \#\mathsf{RP}_2 \Leftrightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{P} \not\subseteq \#\mathsf{RP}_2$.
  - By Theorem 1.7, the inclusions $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$ are proper unless $\mathsf{P} = \mathsf{NP}$. Therefore, $\mathsf{P} \neq \mathsf{NP} \Rightarrow \mathsf{FP} \subsetneq \mathsf{TotP} \subsetneq \#\mathsf{PE} \subsetneq \#\mathsf{P}$.
  - By Corollary 3.1, $\mathsf{TotP} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \mathsf{TotP} \not\subseteq \mathsf{FPRAS}$.
  - By Corollary 3.1, $\mathsf{TotP} \subseteq \mathsf{FPRAS}' \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \mathsf{TotP} \not\subseteq \mathsf{FPRAS}'$.
  - By Corollary 3.1 and Theorem 3.3, $\#\mathsf{RP}_2 \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{RP}_2 \not\subseteq \mathsf{FPRAS}$.
  - By Theorem 3.3 and Corollary 3.1, $\#\mathsf{RP}_2 \subseteq \mathsf{FPRAS}' \Rightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS}' \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{RP}_2 \not\subseteq \mathsf{FPRAS}'$.
  - By Corollary 3.6, $\#\mathsf{RP}_2 = \#\mathsf{RP}_1 \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{RP}_2 \not\subseteq \#\mathsf{RP}_1$.
  - By Theorem 3.1, $\#\mathsf{P} \subseteq \mathsf{FPRAS} \Leftrightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{P} \not\subseteq \mathsf{FPRAS}$.
  - By Theorem 1.7 and Corollary 3.1, $\#\mathsf{PE} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{PE} \not\subseteq \mathsf{FPRAS}$.

- By Theorem 3.3 and the previous result, $\#\mathsf{PE} \subseteq \#\mathsf{RP}_1 \Rightarrow \#\mathsf{PE} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{PE} \not\subseteq \#\mathsf{RP}_1$.
- By Theorem 1.7 and Corollary 3.1, $\#\mathsf{PE} \subseteq \mathsf{FPRAS}' \Rightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS}' \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \#\mathsf{PE} \not\subseteq \mathsf{FPRAS}'$.
- By Corollary 3.1 and Theorem 3.3, $\mathsf{TotP} \subseteq \#\mathsf{RP}_1 \Rightarrow \mathsf{TotP} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$. Therefore, $\mathsf{RP} \neq \mathsf{NP} \Rightarrow \mathsf{TotP} \not\subseteq \#\mathsf{RP}_1$.

- In the case of $\mathsf{P} \neq \mathsf{RP} \neq \mathsf{NP}$.

  In addition to all the above results we have also the following ones.

  - By definitions, $\#\mathsf{RP}_2 \subseteq \#\mathsf{PE} \Leftrightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \#\mathsf{RP}_2 \not\subseteq \#\mathsf{PE}$.
  - As in the proof of Corollary 3.5 we can show that $\#\mathsf{RP}_1 \subseteq \#\mathsf{PE} \Rightarrow \mathsf{P} = \mathsf{RP}$ holds. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \#\mathsf{RP}_1 \not\subseteq \#\mathsf{PE}$.
  - By Theorem 3.3 and the previous result, $\mathsf{FPRAS} \subseteq \#\mathsf{PE} \Rightarrow \#\mathsf{RP}_1 \subseteq \#\mathsf{PE} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \mathsf{FPRAS} \not\subseteq \#\mathsf{PE}$.
  - Similarly, $\mathsf{FPRAS}' \subseteq \#\mathsf{PE} \Rightarrow \#\mathsf{RP}_1 \subseteq \#\mathsf{PE} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \mathsf{FPRAS}' \not\subseteq \#\mathsf{PE}$.
  - Similarly, $\#\mathsf{RP}_1 \subseteq \mathsf{TotP} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \#\mathsf{RP}_1 \not\subseteq \mathsf{TotP}$.
  - By Theorem 1.7 and the previous result, $\#\mathsf{RP}_1 \subseteq \mathsf{FP} \Rightarrow \#\mathsf{RP}_1 \subseteq \mathsf{TotP} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \#\mathsf{RP}_1 \not\subseteq \mathsf{FP}$.
  - By Corollary 3.5, $\mathsf{FPRAS} \subseteq \mathsf{TotP} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \mathsf{FPRAS} \not\subseteq \mathsf{TotP}$.
  - Similarly, $\mathsf{FPRAS}' \subseteq \mathsf{TotP} \Rightarrow \mathsf{P} = \mathsf{RP}$. Therefore, $\mathsf{P} \neq \mathsf{RP} \Rightarrow \mathsf{FPRAS}' \not\subseteq \mathsf{TotP}$. □

3.3 Discussion of results

One direction of $\#\mathsf{P} \subseteq \mathsf{FPRAS}$ iff $\mathsf{RP} = \mathsf{NP}$, namely $\#\mathsf{P} \subseteq \mathsf{FPRAS} \Rightarrow \mathsf{RP} = \mathsf{NP}$, is well-known among researchers in the area of counting complexity. The inverse direction, i.e. that $\mathsf{RP} = \mathsf{NP}$ implies $\#\mathsf{P} \subseteq \mathsf{FPRAS}$, has a proof that is based on arguments similar to those used in the proof of [59, Theorem 4] and it was first stated as a theorem in [22]. This result was also extended to the class TotP in the same work. Thus it holds that $\mathsf{TotP} \subseteq \mathsf{FPRAS}$ iff $\mathsf{RP} = \mathsf{NP}$ (stated as Corollary 3.1 here).

To prove that $\mathsf{FPRAS} \subseteq \mathsf{TotP}$ implies $\mathsf{P} = \mathsf{RP}$, the fact that $\#\mathsf{RP}_1 \subseteq \mathsf{FPRAS}$ was used, i.e. if the accepting paths are more than half of all paths, we can approximate them using the Monte

Carlo method (or an unbiased estimator). Then, the inclusion $\#\mathsf{RP}_1 \subseteq \mathsf{TotP}$ would mean that $\mathsf{RP} \subseteq \mathsf{P}$.

The class $\#\mathsf{RP}_1$ contains a counting version of the PIT (Polynomial Identity Testing) problem, which we define here and we denote by #NonZerosForPIT. Given an n-variable polynomial p over a field $\mathbb{F}$, #NonZerosForPIT is the problem of counting the non-zeros of p in a finite and sufficiently large subset of $\mathbb{F}^n$. As a result, #NonZerosForPIT has an fpras. To the extent of our knowledge, this is the only counting problem that admits an fpras, and has a decision version which is in RP, but it is not known whether it belongs to P.

Finally, the definition of $\#\mathsf{RP}_2$, the class of all #P functions with a decision version in RP turned our attention to two graph problems that are generalizations of counting matchings in graphs. In Chapter 6 we examine the problem #Exact Matchings in restricted classes of graphs with respect to its exact and approximate computation. This is the problem of counting the number of perfect matchings with exactly k red edges in a graph with both black and red edges.

3.4 Notes

The question about the relationship of TotP to FPRAS was first posed and studied in Eleni Bakali's PhD thesis [22], where most propositions and theorems of this chapter can be found. Bakali also gave a more complete picture of the conditional relationships examined in Subsection 3.2.4, i.e. she considered four different possible worlds, the ones we explored here along with two additional worlds that emerge under the assumptions $\mathsf{P} \neq \mathsf{RP} = \mathsf{NP}$ and $\mathsf{P} = \mathsf{RP} = \mathsf{NP}$.

The class FPRAS$'$ and the problems #NonZerosForPIT, #CompositenessWitnesses, #Exact Matchings, and #Blue-Red Matchings were first defined and studied in [24]. Here we provided formal definitions of these four problems.

Chapter 4

On the power of counting the total number of paths

The relation of Toda's celebrated theorem (Theorem 1.4) to the complexity classes $\oplus\mathsf{P}$ and PP motivated our study of TotP-style variants of these classes, namely $\oplus_{\mathsf{tot}}\mathsf{P}$ and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$, respectively, in this chapter.

In Toda's Theorem [144], two facts were combined to prove that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$. First, $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$, where $\oplus\mathsf{P}$ [126] is the class of languages, decidable by an NPTM, where the acceptance condition is that the number of accepting computation paths is odd. Second, $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$. In other words, $\mathsf{BPP}^{\oplus\mathsf{P}}$ is at least as powerful as the polynomial hierarchy and it can be decided in polynomial time with just one oracle call to a $\#\mathsf{P}$ function.

The second part of the proof of Toda's Theorem implies that $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$ and therefore $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$ [144]. PP [75] stands for 'Probabilistic Polynomial-time' and it is a gap-definable class; it is defined based on the difference (or the *gap*) between accepting and rejecting paths of an NPTM. A language L is in PP if there is an NPTM M such that $x \in L$ iff $M(x)$ has more accepting than rejecting paths. It is also as hard as $\#\mathsf{P}$ in the sense that $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ [26].

Apart from the classes $\oplus_{\mathsf{tot}}\mathsf{P}$ and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$, we introduce here other complexity classes defined by properties of TotP functions. We build upon the study of classes that have definitions based on the number of accepting paths, or the gap of an NPTM [66]. For example, the complexity class UP was introduced in [148] along with unambiguous NPTMs, which means NPTMs that have at most one accepting path. Over the years, many other classes of decision problems were introduced by adding constraints to the acceptance condition of NP. We are going to focus

on the following classes: UP [148], FewP [8], $\oplus$P [126],[1] $\mathsf{Mod_k P}$ [44],[2] SPP [121],[3] WPP [66], $\mathsf{C_= P}$ [136], and PP [136, 75]. The definitions of these classes are given in Table 4.1. Each of them can be defined by properties of either #P or GapP functions.

Class	Function f in:	If $x \in L$:	If $x \notin L$:
UP	#P	$f(x) = 1$	$f(x) = 0$
FewP	#P	$f(x) \leq p(\|x\|)$ for some polynomial p and $f(x) > 0$	$f(x) = 0$
$\oplus$P	#P	$f(x)$ is odd	$f(x)$ is even
$\mathsf{Mod_k P}$	#P	$f(x) \not\equiv 0 \pmod{k}$	$f(x) \equiv 0 \pmod{k}$
SPP	GapP	$f(x) = 1$	$f(x) = 0$
WPP	GapP	$f(x) = g(x)$ for some $g \in$ FP with $0 \notin \text{range}(g)$	$f(x) = 0$
$\mathsf{C_= P}$	GapP	$f(x) = 0$	$f(x) \neq 0$ [alt-def: $f(x) > 0$]
PP	GapP	$f(x) > 0$	$f(x) \leq 0$ [alt-def: $f(x) < 0$]

Table 4.1: Definitions of the classes UP, FewP, $\oplus$P, $\mathsf{Mod_k P}$, SPP, WPP, $\mathsf{C_= P}$, PP.

Remark 4.1. *(a) Note that* $\mathsf{Mod_2 P} = \oplus\mathsf{P}$ *and* $\mathsf{Mod_k P}$ *for* $\mathsf{k} > 2$ *is a generalization of* $\oplus$P*, based on congruence mod integers other than two.*

(b) The class $\mathsf{Mod_k P}$ *was defined in [35] via the acceptance condition '*$x \in L$ *iff* $f(x) \not\equiv 0 \pmod{k}$*'. On the other hand, in [44]* $\mathsf{Mod_k P}$ *was defined via the alternative condition '*$x \in L$ *iff* $f(x) \equiv 0 \pmod{k}$*' (under which the class of [35] would be* $\mathsf{coMod_k P}$*).*

(c) For the alternative definition of $\mathsf{C_= P}$*, note that given a function* $f \in$ GapP*, that satisfies the first definition, we have that the function* f^2 *belongs to* GapP *as well, since* GapP *is closed under multiplication and* f^2 *satisfies the alternative definition, i.e.* $x \in L$*, if* $f^2(x) = 0$*, whereas* $x \notin L$*, if* $f^2(x) > 0$*.*

[1] The class $\oplus$P was defined independently under the name EP in [78].
[2] $\mathsf{Mod_k P}$ was also studied independently in [35] and [85].
[3] SPP was studied independently in [66].

Known relationships among these classes are given below.

Proposition 4.1 ([131]). *(a)* $\mathsf{UP} \subseteq \mathsf{FewP} \subseteq \mathsf{NP} \subseteq \mathsf{coC_{=}P} \subseteq \mathsf{PP}$.
(b) $\mathsf{FewP} \subseteq \mathsf{SPP} \subseteq \mathsf{WPP} \subseteq \mathsf{C_{=}P} \subseteq \mathsf{PP}$.
(c) $\mathsf{SPP} \subseteq \oplus\mathsf{P} \subseteq \mathsf{Mod_kP}$.

In Subsections 4.1.1, 4.1.2, and 4.1.3, we introduce the classes $\mathsf{Gap_{tot}P}$, $\mathsf{U_{tot}P}$, $\mathsf{Few_{tot}P}$, $\oplus_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Mod_{k\,tot}P}$, $\mathsf{SP_{tot}P}$, $\mathsf{WP_{tot}P}$, $\mathsf{C_{=tot}P}$, $\mathsf{P_{tot}P}$ by properties of TotP functions. Their definitions are summarized in Table 4.2. We examine the relationship between each of them and its counterpart, defined by the same property of either a #P or a GapP function. Except for $\mathsf{U_{tot}P}$ and $\mathsf{Few_{tot}P}$ which coincide with P, all the other tot-definable classes are equal to their analogs definable by #P functions. Thus, for the classes $\oplus$P, $\mathsf{Mod_kP}$, SPP, WPP, $\mathsf{C_{=}P}$, and PP, we obtain a family of complete problems that are defined by TotP-complete problems under parsimonious reductions (and not by #P-complete), or equivalently, by problems in P (and not by NP-complete ones).

In Subsection 4.1.4 we give variants of the Valiant-Vazirani and Toda's Theorems. In Subsection 4.1.5, we present problems definable by a TotP function, that are complete for gap-definable classes, building upon a relevant result by Curticapean [52]. Finally, we study the complexity of the problem DiffPerfMatch$_{=g}$ in Subsection 4.1.6.

4.1 Tot-definable classes

Although we focus on defining tot-counterparts of the classes described in Table 4.1, counting the total number of paths of an NPTM has already been used to define analogs of the classes RP, BPP, and PP in [80], which have been denoted by $\mathsf{R_{path}}$, $\mathsf{BPP_{path}}$, and $\mathsf{PP_{path}}$, respectively. For example, $\mathsf{R_{path}}$ is the class of languages L such that there exists an NPTM M such that for all $x \in \Sigma^*$ it holds that $x \in L$ if $acc_M(x) > \frac{1}{2} \cdot total_M(x)$ and $x \notin L$ if $rej_M(x) = total_M(x)$, where $total_M(x)$ denotes the total number of paths of M on x (not minus 1) and $acc_M(x)$, $rej_M(x)$ are the number of accepting and rejecting paths of M on x, respectively. Remarkably, $\mathsf{R_{path}} = \mathsf{NP}$ and $\mathsf{PP_{path}} = \mathsf{PP}$ [80]. The relationship of $\mathsf{BPP_{path}}$ to other classes, such as $\mathsf{P^{NP[log]}}$, the Arthur–Merlin class MA, and Σ_2^{p}, has been studied in [80].

4.1.1 The class $\mathsf{Gap_{tot}P}$

For a Turing machine M, we define $\Delta M(x) = acc_M(x) - rej_M(x)$, where $acc_M(x)$, $rej_M(x)$ are the number of accepting and rejecting paths of M on x, respectively.

Definition 4.1. *A function $f : \Sigma^* \to \mathbb{N}$ is in GapP iff there exists an NPTM M such that for all inputs x, $f(x) = \Delta M(x)$.*
Equivalently, $\mathsf{GapP} = \{\Delta M : \Sigma^* \to \mathbb{N} \mid M \text{ is an NPTM}\}$.

GapP is the closure of $\#\mathsf{P}$ under subtraction and its functions can take negative values.

Theorem 4.1. *The following are equivalent:*
1. *$f \in \mathsf{GapP}$.*
2. *f is the difference of two $\#\mathsf{P}$ functions.*
3. *f is the difference of a $\#\mathsf{P}$ and an FP function.*
4. *f is the difference of an FP and a $\#\mathsf{P}$ function.*

In other words, $\mathsf{GapP} = \#\mathsf{P} - \#\mathsf{P} = \#\mathsf{P} - \mathsf{FP} = \mathsf{FP} - \#\mathsf{P}$.

If we consider the difference of two TotP functions instead of $\#\mathsf{P}$ ones, we can define a variant of GapP as follows.

Definition 4.2. *A function f belongs to the class $\mathsf{Gap_{tot}P}$ iff it is the difference of two TotP functions.*

It turns out that the class $\mathsf{Gap_{tot}P}$ is exactly the class GapP.

Proposition 4.2. $\mathsf{Gap_{tot}P} = \mathsf{GapP}$.

Proof. $\mathsf{Gap_{tot}P} \subseteq \mathsf{GapP}$ is straightforward, since $\mathsf{TotP} \subseteq \#\mathsf{P}$. For $\mathsf{GapP} \subseteq \mathsf{Gap_{tot}P}$, note that for any $\#\mathsf{P}$ function f, there exist NPTMs M, M' such that $f(x) = acc_M(x) = tot_{M'}(x) - tot_M(x)$, where we obtain M' by doubling the accepting paths of M. So for any $g \in \mathsf{GapP}$ there exist NPTMs N, N', M, M' such that $g(x) = acc_N(x) - acc_M(x) = (tot_{N'}(x) - tot_N(x)) - (tot_{M'}(x) - tot_M(x)) = (tot_{N'}(x) + tot_M(x)) - (tot_N(x) + tot_{M'}(x)) = tot_{M_1}(x) - tot_{M_2}(x)$, where M_1, M_2 can be constructed as described in Proposition 1.8. □

Corollary 4.1. $\mathsf{Gap_{tot}P} = \mathsf{GapP} = \#\mathsf{P} - \#\mathsf{P} = \mathsf{TotP} - \mathsf{TotP} = \#\mathsf{P} - \mathsf{FP} = \mathsf{FP} - \#\mathsf{P} = \mathsf{FP} - \mathsf{TotP} = \mathsf{TotP} - \mathsf{FP}$.

Proof. To prove $\#\mathsf{P} - \mathsf{FP} = \mathsf{FP} - \#\mathsf{P} = \mathsf{FP} - \mathsf{TotP} = \mathsf{TotP} - \mathsf{FP}$, we show that $\mathsf{FP} - \#\mathsf{P} \subseteq \mathsf{FP} - \mathsf{TotP}$. This is essentially a proof for $\#\mathsf{P} - \mathsf{FP} \subseteq \mathsf{TotP} - \mathsf{FP}$ as well.

Let $g \in \mathsf{FP}$ and $f = acc_M \in \#\mathsf{P}$. Then, $g(x) - f(x) = g(x) - (tot_{M'}(x) - tot_M(x))$, where M' is obtained from M by doubling its accepting paths (as in the proof of Proposition 4.2). W.l.o.g. we assume that the computation tree of M is a perfect binary tree (or in other words, M is in normal form), so we have that $g(x) - (tot_{M'}(x) - tot_M(x)) = g(x) - (tot_{M'}(x) - (2^{p(|x|)} - 1)) = g'(x) - tot_{M'}(x)$, where $g' \in \mathsf{FP}$ with $g'(x) = g(x) + 2^{p(|x|)} - 1$ for any x. □

For a $\#\mathsf{P}$ function acc_M, it holds that $rej_M(x) = 2^{p(|x|)} - acc_M(x)$ is also a $\#\mathsf{P}$ function. For example, counting unsatisfying assignments for a **CNF** formula is in $\#\mathsf{P}$. On the contrary, for a TotP function tot_M, we do not know whether $(2^{p(|x|)} - 1) - tot_M(x)$ belongs to TotP. For example, if counting unsatisfying assignments for a **DNF** formula ϕ is in TotP, then determining if there is an unsatisfying assignment for ϕ is in P, which in turn means that determining whether the **CNF** formula $\neg\phi$ has a satisfying assignment is in P (note that in this case we have to consider $p(|\phi|) = n$, where n is the number of variables of the input formula ϕ). On the other hand, Corollary 4.1 implies that $(2^{p(|x|)} - 1) - tot_M(x)$ and in fact, every $\#\mathsf{P}$ function is in $\mathsf{Gap}_{\mathsf{tot}}\mathsf{P}$.

4.1.2 The classes $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Few}_{\mathsf{tot}}\mathsf{P}$, $\oplus_{\mathsf{tot}}\mathsf{P}$, and $\mathsf{Mod}_{\mathsf{ktot}}\mathsf{P}$

The definitions of the classes $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Few}_{\mathsf{tot}}\mathsf{P}$, $\oplus_{\mathsf{tot}}\mathsf{P}$, and $\mathsf{Mod}_{\mathsf{ktot}}\mathsf{P}$ are analogous to those in Table 4.1, but instead of $\#\mathsf{P}$ or GapP functions, functions in either TotP or $\mathsf{Gap}_{\mathsf{tot}}\mathsf{P}$ are used to define them. They are summarized in Table 4.2.

The class UP appears in the well known Valiant-Vazirani Theorem which states that there is a randomized reduction from an instance of SAT to a **CNF** formula which has at most one satisfying assignment.

Theorem 4.2 (Valiant-Vazirani Theorem [151])**.** $\mathsf{NP} \subseteq \mathsf{RP}^{\mathsf{UP}}$.

Below, we explore the computational power of the *tot-definable* class $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$.

Proposition 4.3. *(a)* $\mathsf{P} = \mathsf{U}_{\mathsf{tot}}\mathsf{P}$.
(b) $\mathsf{U}_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{UP}$.
(c) If $\mathsf{UP} \subseteq \mathsf{U}_{\mathsf{tot}}\mathsf{P}$*, then* $\mathsf{P} = \mathsf{UP}$ *(and thus* $\mathsf{RP} = \mathsf{NP}$*).*

Proof. (a) Let $L \in \mathsf{U}_{\mathsf{tot}}\mathsf{P}$. Then there exists an NPTM M such that $x \in L$ iff M has 2 paths, whereas $x \notin L$ iff M has 1 path. On any input, the polynomial time Turing machine M'

Class	Function f in:	If $x \in L$:	If $x \notin L$:
$\mathsf{U_{tot}P}$	TotP	$f(x) = 1$	$f(x) = 0$
$\mathsf{Few_{tot}P}$	TotP	$f(x) \leq p(\|x\|)$ for some polynomial p and $f(x) > 0$	$f(x) = 0$
$\oplus_{\mathsf{tot}}\mathsf{P}$	TotP	$f(x)$ is odd	$f(x)$ is even
$\mathsf{Mod_{ktot}P}$	TotP	$f(x) \not\equiv 0 \pmod{k}$	$f(x) \equiv 0 \pmod{k}$
$\mathsf{SP_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = 1$	$f(x) = 0$
$\mathsf{WP_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = g(x)$ for some $g \in \mathsf{FP}$ with $0 \notin \text{range}(g)$	$f(x) = 0$
$\mathsf{C_{=tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = 0$	$f(x) \neq 0$ [alt-def: $f(x) > 0$]
$\mathsf{P_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) > 0$	$f(x) \leq 0$ [alt-def: $f(x) < 0$]

Table 4.2: Definitions of the classes $\mathsf{U_{tot}P}$, $\mathsf{Few_{tot}P}$, $\oplus_{\mathsf{tot}}\mathsf{P}$, $\mathsf{Mod_{ktot}P}$, $\mathsf{SP_{tot}P}$, $\mathsf{WP_{tot}P}$, $\mathsf{C_{=tot}P}$, $\mathsf{P_{tot}P}$.

simulates either the unique path or the two paths of M deterministically and it either rejects or accepts, respectively. This is a proof of (b) as well. Inversely, if $L \in \mathsf{P}$, then there is an NPTM N, which, for any input x, simulates the corresponding deterministic computation and generates one or two paths if the answer is 'no' or 'yes', respectively.
(c) By (a), if $\mathsf{UP} \subseteq \mathsf{U_{tot}P}$, then $\mathsf{UP} \subseteq \mathsf{P}$ and $\mathsf{RP} = \mathsf{NP}$ by the Valiant-Vazirani Theorem. □

In fact, the classes $\mathsf{U_{tot}P}$ and $\mathsf{Few_{tot}P}$ coincide.

Proposition 4.4. *(a)* $\mathsf{P} = \mathsf{U_{tot}P} = \mathsf{Few_{tot}P}$.
(b) If $\mathsf{FewP} \subseteq \mathsf{Few_{tot}P}$, *then* $\mathsf{P} = \mathsf{FewP}$.

Proof. Part (b) is immediate from (a). For (a), let $L \in \mathsf{Few_{tot}P}$. Consider the Turing machine M which has either more than 1 but polynomially many paths if $x \in L$ or just 1 path if $x \notin L$. Then an NPTM, let's say M', simulates deterministically all the paths of M and in the first case M' makes a branching forming two paths, while in the second case it halts. So, $L \in \mathsf{U_{tot}P}$. □

Next we show that the classes $\oplus_{\mathsf{tot}}\mathsf{P}$ and $\mathsf{Mod_{ktot}P}$ coincide with $\oplus\mathsf{P}$ and $\mathsf{Mod_kP}$, respec-

tively. As mentioned above, Toda showed that $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$ as an intermediate step for proving his theorem. The proof was completed by proving that $\mathsf{BPP}^{\oplus\mathsf{P}}$ can be decided in polynomial time using an oracle call to #P. In fact, only the value of a #P function modulo 2^m, for some m, is needed. Since, $\mathsf{P}^{\mathsf{TotP}} = \mathsf{P}^{\#\mathsf{P}}$, it also holds that $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{TotP}}$. It is natural to ask about the power of classes that give information about either bits of TotP functions or the value of TotP functions mod k.

Problems in $\oplus\mathsf{P}$ can be decided with the information of the rightmost bit of a #P function. The class $\oplus_{\mathsf{tot}}\mathsf{P}$ is the class of decision problems, for which the acceptance condition is that the number of all computation paths of an NPTM is even (or the number of all computation paths minus 1 is odd).

Proposition 4.5. $\oplus_{\mathsf{tot}}\mathsf{P} = \oplus\mathsf{P}$.

Proof. $\oplus_{\mathsf{tot}}\mathsf{P} \subseteq \oplus\mathsf{P}$: Consider a language $L \in \oplus_{\mathsf{tot}}\mathsf{P}$ and the NPTM M such that $x \in L$ iff $tot_M(x)$ is odd. Consider an NPTM M' that simulates M. Since, M' can distinguish the leftomost path of M, it rejects on this path and it accepts on every other path. Then, $x \in L$ iff $acc_{M'}(x)$ is odd.

$\oplus\mathsf{P} \subseteq \oplus_{\mathsf{tot}}\mathsf{P}$: Let $L \in \oplus\mathsf{P}$ and M be the NPTM such that $x \in L$ iff $acc_M(x)$ is odd. Then, we obtain M' by doubling the rejecting paths of M and add one more path. It holds that $x \in L$ iff $tot_M(x)$ is odd. □

We also give an alternative proof of Proposition 4.5. Let $\oplus$Pl-Rtw-Mon-3CNF be the problem that on input a planar **3CNF** formula where each variable appears positively and in exactly two clauses, accepts iff the formula has an odd number of satisfying assignments. The counting version of this problem, namely #Pl-Rtw-Mon-3CNF, is in TotP; it is self-reducible like every satisfiability problem and has a decision version in P, since every monotone formula has at least one satisfying assignment.

Proposition 4.6 ([150])**.** $\oplus$Pl-Rtw-Mon-3CNF *is* $\oplus\mathsf{P}$*-complete.*

Corollary 4.2. $\oplus_{\mathsf{tot}}\mathsf{P} = \oplus\mathsf{P}$.

Proof. Let $L \in \oplus\mathsf{P}$. Then, it holds that $x \in L$ iff $f(x) \equiv 1 \pmod 2$ for some $f \in \#\mathsf{P}$. By Proposition 4.6, there is an $h \in \mathsf{FP}$, such that $x \in L$ iff #Pl-Rtw-Mon-3CNF$(h(x)) \equiv 1 \pmod 2$. So, define the TotP function $g = \#$Pl-Rtw-Mon-3CNF $\circ\, h$. Then $x \in L$ iff $g(x) \equiv 1 \pmod 2$ and thus $L \in \oplus_{\mathsf{tot}}\mathsf{P}$. □

By generalizing the acceptance condition, we obtain $\mathsf{Mod}_{k\mathsf{tot}}\mathsf{P}$ classes for any $k \in \mathbb{N}$, which in fact, is the class $\mathsf{Mod}_{k}\mathsf{P}$.

Proposition 4.7. $\mathsf{Mod}_{k\mathsf{tot}}\mathsf{P} = \mathsf{Mod}_{k}\mathsf{P}$.

Proof. The proof of $\mathsf{Mod}_{k\mathsf{tot}}\mathsf{P} \subseteq \mathsf{Mod}_{k}\mathsf{P}$ is very similar to the proof of $\oplus_{\mathsf{tot}}\mathsf{P} \subseteq \oplus\mathsf{P}$ in Proposition 4.5.

For the inclusion $\mathsf{Mod}_{k}\mathsf{P} \subseteq \mathsf{Mod}_{k\mathsf{tot}}\mathsf{P}$, let $L \in \mathsf{Mod}_{k}\mathsf{P}$ and M be the NPTM such that $x \in L$ iff $acc_M(x) \equiv a \pmod{k}$, for some $m \in \{1, ..., k-1\}$. Then, M' can be obtained from M by generating k paths for every rejecting path and one more path (a dummy path). So, $\#(\text{paths of } M' \text{ on input } x) - 1 \equiv a \pmod{k}$. So, there is an NPTM M', such that $tot_{M'}(x) \equiv acc_M(x) \pmod{k}$. □

Remark 4.2. *The above proof shows that, not only equivalence or non-equivalence modulo k is preserved, but also the value of the $\#\mathsf{P}$ function modulo k is preserved.*

Conclusion 4.1. *So, we can say that if we have information about the rightmost bit of a TotP function is as powerful as having information about the rightmost bit of a $\#\mathsf{P}$ function. Toda's Theorem would be true if we used $\oplus_{\mathsf{tot}}\mathsf{P}$ instead of $\oplus\mathsf{P}$. Moreover, it holds that $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{TotP}[1]}$, where it suffices to make an oracle call to a TotP function mod 2^m, for some m. However, $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$ is defined by a constraint on a TotP function that yields only nondeterministic Turing machines with polynomially many paths. This means that $\mathsf{U}_{\mathsf{tot}}\mathsf{P}$ gives no more information than the class P and as a result, it cannot replace the class UP in the Valiant-Vazirani Theorem.*

4.1.3 The gap-definable classes $\mathsf{SP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{WP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{C}_{=\mathsf{tot}}\mathsf{P}$, and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$

By Proposition 4.2 and the definitions of the classes $\mathsf{SP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{WP}_{\mathsf{tot}}\mathsf{P}$, $\mathsf{C}_{=\mathsf{tot}}\mathsf{P}$, and $\mathsf{P}_{\mathsf{tot}}\mathsf{P}$, we have that these classes are equal to SPP, WPP, $\mathsf{C}_{=}\mathsf{P}$, and PP, respectively.

Corollary 4.3. *(a)* $\mathsf{SP}_{\mathsf{tot}}\mathsf{P} = \mathsf{SPP}$.
(b) $\mathsf{WP}_{\mathsf{tot}}\mathsf{P} = \mathsf{WPP}$.
(c) $\mathsf{C}_{=\mathsf{tot}}\mathsf{P} = \mathsf{C}_{=}\mathsf{P}$.
(d) $\mathsf{P}_{\mathsf{tot}}\mathsf{P} = \mathsf{PP}$.

The next corollary is an analog of Proposition 4.1. It is immediate from Proposition 4.1 and the propositions of Subsections 4.1.2, 4.1.3.

Corollary 4.4. *(a)* $\mathsf{P} = \mathsf{U}_{\mathsf{tot}}\mathsf{P} = \mathsf{Few}_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{coC}_{=\mathsf{tot}}\mathsf{P} \subseteq \mathsf{P}_{\mathsf{tot}}\mathsf{P}$.
(b) $\mathsf{Few}_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{SP}_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{WP}_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{C}_{=\mathsf{tot}}\mathsf{P} \subseteq \mathsf{P}_{\mathsf{tot}}\mathsf{P}$.
(c) $\mathsf{SP}_{\mathsf{tot}}\mathsf{P} \subseteq \oplus_{\mathsf{tot}}\mathsf{P} \subseteq \mathsf{Mod}_{\mathsf{ktot}}\mathsf{P}$.

4.1.4 Variants of the Valiant-Vazirani and Toda's Theorems

Consider the following satisfiability problems.

$$\oplus\textsc{Sat} = \{\phi \mid \#\textsc{Sat}(\phi) \not\equiv 0 \,(\mathrm{mod}\, 2)\}$$

$$\textsc{Mod}_k\textsc{Sat} = \{\phi \mid \#\textsc{Sat}(\phi) \not\equiv 0 \,(\mathrm{mod}\, k)\}$$

Since $\#\textsc{Sat}$ is $\#\mathsf{P}$-complete under parsimonious reductions, $\oplus\textsc{Sat}$ and $\textsc{Mod}_k\textsc{Sat}$ are complete for $\oplus\mathsf{P}$ and $\mathsf{Mod}_\mathsf{k}\mathsf{P}$, repsectively.

For any $\#\mathrm{A} \in \#\mathsf{P}$, we define two decision problems related to $\#\mathrm{A}$: $\oplus\mathrm{A} = \{x \in \Sigma^* \mid f(x) \not\equiv 0 \pmod 2\}$ and $\textsc{Mod}_k\mathsf{A} = \{x \in \Sigma^* \mid f(x) \not\equiv 0 \pmod k\}$. Then completeness results for $\oplus\mathrm{A}$ and $\textsc{Mod}_k\mathsf{A}$ can be proven, provided that $\#\mathrm{A}$ is TotP-complete under parsimonious reductions.

Proposition 4.8. *Let* $\#\mathrm{A}$ *be* TotP*-complete under parsimonious reductions. Then,*
(a) $\oplus\mathrm{A}$ *is complete for* $\oplus\mathsf{P}$*,*
(b) $\textsc{Mod}_k\mathrm{A}$ *is complete for* $\mathsf{Mod}_\mathsf{k}\mathsf{P}$*.*

Proof. (a) It holds that $\#\mathrm{A} \in \#\mathsf{P}$, thus, by definition $\oplus\mathrm{A} \in \oplus\mathsf{P}$. Since $\#\mathrm{A}$ is TotP-complete under parsimonious reductions, the problem $\oplus\mathrm{A}$ is $\oplus_{\mathsf{tot}}\mathsf{P}$-complete. By Proposition 4.5, it is also $\oplus\mathsf{P}$-complete. The proof of (b) is completely analogous. □

The following problems are related to the TotP-complete function $\#\textsc{Tree-Monotone-Circuit-Sat}$ given in Definition 2.6.

$$\oplus\mathrm{TMC} = \{C \mid C \text{ is non-increasing w.r.t. } \leq_{tree} \text{ and } \#\mathrm{TMC}(C) \not\equiv 0 \pmod 2\}$$

$$\textsc{Mod}_k\mathrm{TMC} = \{C \mid C \text{ is non-increasing w.r.t. } \leq_{tree} \text{ and } \#\mathrm{TMC}(C) \not\equiv 0 \pmod k\}$$

By Proposition 4.8, these are complete for $\oplus\mathsf{P}$ and $\mathsf{Mod}_\mathsf{k}\mathsf{P}$, respectively.

The next three corollaries are derived from the Valiant-Vazirani Theorem (Theorem 4.2), Toda's Theorem (Theorem 1.4), and the results of this section.

Corollary 4.5. *(a)* $\mathsf{NP} \subseteq \mathsf{RP}^{\oplus_{\mathsf{tot}}\mathsf{P}}$.
(b) $\textsc{Sat} \in \mathsf{RP}^{\oplus\mathrm{TMC}}$.

Proof. (a) A consequence of the Valiant-Vazirani Theorem is that $\mathsf{NP} \subseteq \mathsf{RP}^{\oplus\mathsf{P}}$. So, by Proposition 4.5, we have that $\mathsf{NP} \subseteq \mathsf{RP}^{\oplus_{\mathsf{tot}}\mathsf{P}}$.

(b) It is immediate from (a) and the $\oplus\mathsf{P}$-completeness of $\oplus$TMC. □

Corollary 4.6. *(a)* $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus_{\mathsf{tot}}\mathsf{P}}$.
(b) $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\mathrm{TMC}}$.

Proof. (a) It is immediate from Toda's Theorem and Proposition 4.5.

(b) By Toda's Theorem and the $\oplus\mathsf{P}$-completeness of $\oplus$TMC, we have that $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\mathrm{TMC}}$.

□

Remark 4.3. *Note that Proposition 4.8 allows us to replace* $\oplus$TMC *in Corollaries 4.5 and 4.6, by any other complete problem for* $\oplus\mathsf{P}$ *definable by a function that is* TotP*-complete under parsimonious reductions.*

Corollary 4.7. $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{TotP}[1]}$*, where an oracle call to the value of a* TotP *function modulo* 2^m*, for some* $m \in \mathbb{N}$*, suffices.*

Proof. In [144], Toda showed that $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$, where an oracle call to the value of a #P function modulo 2^m, for some $m \in \mathbb{N}$, is used. By the proof of Proposition 4.7, this oracle call can be replaced by an oracle call to the value of a TotP function modulo 2^m. □

The following two corollaries are also true.

Corollary 4.8. $\mathsf{BPP}^{\oplus_{\mathsf{tot}}\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{P}_{\mathsf{tot}}\mathsf{P}}$.

Corollary 4.9. $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{TotP}[1]}$.

4.1.5 Complete problems for $\mathsf{C}_{=}\mathsf{P}$, WPP, and PP definable by the TotP function #PerfMatch

Given a #P function $\#\mathrm{A} : \Sigma^* \to \mathbb{N}$, we define the following decision problems associated with #A:

DiffA$_{=0}$
Input: $(x, y) \in \Sigma^* \times \Sigma^*$.
Output: Does $\#\mathrm{A}(x) = \#\mathrm{A}(y)$ hold?

DIFFA$_{>0}$

Input: $(x, y) \in \Sigma^* \times \Sigma^*$.

Output: Does $\#\mathrm{A}(x) > \#\mathrm{A}(y)$ hold?

The next problem, namely DIFFA$_{=1}$, is a promise problem, i.e. the input to the problem belongs to one of the following classes:

$$I_{YES} = \{(x, y) \mid \#\mathrm{A}(x) = \#\mathrm{A}(y) + 1\}$$

$$I_{NO} = \{(x, y) \mid \#\mathrm{A}(x) = \#\mathrm{A}(y)\}$$

DIFFA$_{=1}$

Input: $(x, y) \in I_{YES} \cup I_{NO}$.

Output: Does $(x, y) \in I_{YES}$?

Proposition 4.9. *For any function* $\#\mathrm{A} \in \#\mathsf{P}$*, it holds that:*

(a) DIFFA$_{=0} \in \mathsf{C}_{=}\mathsf{P}$, DIFFA$_{>0} \in \mathsf{PP}$, *and* DIFFA$_{=1} \in \mathsf{SPP}$.

(b) If $\#\mathrm{A}$ *is* $\#\mathsf{P}$*-complete under parsimonious reductions, then* DIFFA$_{=0}$, DIFFA$_{>0}$, *and* DIFFA$_{=1}$ *are complete for* $\mathsf{C}_{=}\mathsf{P}$, PP, *and* SPP, *respectively.*

(c) If $\#\mathrm{A}$ *is* TotP*-complete under parsimonious reductions, then* DIFFA$_{=0}$, DIFFA$_{>0}$, *and* DIFFA$_{=1}$ *are complete for* $\mathsf{C}_{=}\mathsf{P}$, PP, *and* SPP, *respectively.*

Proof. We show the proposition for the problem DIFFA$_{=0}$. The proof for the other problems is completely analogous.

(a) We have that an instance (x, y) of DIFFA$_{=0}$ is a *yes* instance iff $\#\mathrm{A}(x) = \#\mathrm{A}(y)$ iff $\#\mathrm{A}(x) - \#\mathrm{A}(y) = 0$. The difference $\#\mathrm{A}(x) - \#\mathrm{A}(y)$ is a GapP function, since it can be written as $\#\mathrm{A}'(x, y) - \#\mathrm{A}''(x, y)$, where $\#\mathrm{A}'(x, y)$ (resp. $\#\mathrm{A}''(x, y)$) is function $\#\mathrm{A}$ on input x (resp. y), which means that $\#\mathrm{A}', \#\mathrm{A}'' \in \#\mathsf{P}$.

(b) A language $L \in \mathsf{C}_{=}\mathsf{P}$ can be decided by the value of a function $f \in \mathsf{GapP}$: $x \in L$ iff $f(x) = 0$. By definition, $f(x) = g(x) - h(x)$ for some $g, h \in \#\mathsf{P}$. Since $\#\mathsf{A}$ is $\#\mathsf{P}$-complete under parsimonious reductions, we have that $g(x) = \#\mathrm{A}(t_1(x))$ and $h(x) = \#\mathrm{A}(t_2(x))$, for some $t_1, t_2 \in \mathsf{FP}$. So, $x \in L$ iff $g(x) - h(x) = 0$ iff $\#\mathrm{A}(t_1(x)) - \#\mathrm{A}(t_2(x)) = 0$ iff $(t_1(x), t_2(x))$ is a *yes* instance for DIFFA$_{=0}$.

(c) By Corollary 4.3, a language $L \in \mathsf{C}_{=}\mathsf{P}$ can also be decided by the difference of two functions $g, h \in \mathsf{TotP}$: $x \in L$ iff $g(x) - h(x) = 0$. If $\#\mathrm{A}$ is TotP-complete, $x \in L$ is decided by $\#\mathrm{A}(t_1(x)) - \#\mathrm{A}(t_2(x))$ for some $t_1, t_2 \in \mathsf{FP}$. □

For example, the problem DIFFSAT$_{=0}$ is complete for the class $\mathsf{C}_{=}\mathsf{P}$. Note that this problem was defined in [52], where it is called SAT$_{=}$. We use a slightly different notation here, which we believe is more suitable for defining other problems as well, that lie in the gap-definable classes examined in this chapter.

The last problem we define is the promise problem DIFFA$_{=g}$. In this case, there is a function $g \in \mathsf{FP}$, $0 \notin \text{range}(g)$, such that the input $(x, y) \in \Sigma^* \times \Sigma^*$ belongs to one of the following two classes:

$$I_{YES} = \{(x, y) \mid \#\text{A}(x) = \#\text{A}(y) + g(x, y)\}$$

$$I_{NO} = \{(x, y, g) \mid \#\text{A}(x) = \#\text{A}(y)\}$$

DIFFA$_{=g}$
Input: $(x, y, g) \in I_{YES} \cup I_{NO}$.
Output: Does $(x, y, g) \in I_{YES}$?

For example, the problem DIFFSAT$_{=g}$ takes as input two **CNF** formulas such that either they have the same number of satisfying assignments or the first one has $g(x)$ more satisfying assignments than the second one. The problem is to decide which is the case. This is a generalization of the problem PROMISE-EXACT-NUMBER-SAT defined in [128]. Note that DIFFSAT$_{=g}$ corresponds to a family of problems, one for each $g \in \mathsf{FP}$, $0 \notin \text{range}(g)$.

Proposition 4.10. *(a)* DIFFA$_{=g} \in \mathsf{WPP}$.

(b) Every problem in WPP *is reducible to some* DIFFA$_{=g}$*, where*

(i) #A *is* #P*-complete under parsimonious reductions.*

(ii) #A *is* TotP*-complete under parsimonious reductions.*

Proof. The proof is analogous to the proof of Proposition 4.9. □

Remark 4.4. *Compared to Proposition 4.9, Proposition 4.10 does not provide a concrete problem that is complete for the class* WPP*. For example,* DIFFSAT$_{=g}$ *is essentially a family of problems, one for each* $g \in \mathsf{FP}$, $0 \notin \text{range}(g)$.

Curticapean gave a first $\mathsf{C}_{=}\mathsf{P}$-complete problem, defined by a function that is not #P-complete, namely #PERFMATCH [52]. By Proposition 4.9(c), we obtain a family of complete problems for the classes $\mathsf{C}_{=}\mathsf{P}$, PP, and SPP defined by functions that are not #P-complete under parsimonious reductions.

Although the problem #PerfMatch is not known to be either #P-complete or TotP-complete under parsimonious reductions, DiffPerfMatch$_{=0}$ is C$_=$P-complete [52].

Proposition 4.11 ([52])**.** DiffPerfMatch$_{=0}$ *is complete for* C$_=$P*.*

Proof. In [52] a reduction from DiffSat$_{=0}$ to DiffPerfMatch$_{=0}$ is described. Given a pair of 3CNF formulas (ϕ, ϕ'), two unweighted graphs G, G' can be constructed such that

$$\#\text{Sat}(\phi) - \#\text{Sat}(\phi') = 2^{-T}(\#\text{PerfMatch}(G) - \#\text{PerfMatch}(G'))$$

where $T \in \mathbb{N}$ can be computed in polynomial time with respect to the input (ϕ, ϕ'). □

We show the following proposition for the classes PP and WPP.

Proposition 4.12. *(a)* DiffPerfMatch$_{>0}$ *is complete for* PP*.*
(b) Every problem in WPP *is reducible to* DiffPerfMatch$_{=g}$ *defined by a function* $g \in$ FP*,* $0 \notin \text{range}(g)$*.*

Proof. (a) The reduction of [52] described briefly in the proof of Proposition 4.11, is also a reduction from DiffSat$_{>0}$ to DiffPerfMatch$_{>0}$.

(b) Given an input (ϕ, ϕ') to DiffSat$_{=g}$, which is defined by $h \in$ FP, $0 \notin \text{range}(h)$, the problem reduces to DiffPerfMatch$_{=g}$ on input (G, G'), where G, G' are the graphs of (a) and $g(G, G') = 2^T \cdot h(\phi, \phi')$. Then, $(\phi, \phi') \in$ DiffSat$_{=g}$ if

$$\#\text{Sat}(\phi) - \#\text{Sat}(\phi') = h(\phi, \phi') \text{ iff}$$

$$\#\text{PerfMatch}(G) - \#\text{PerfMatch}(G') = 2^T \cdot (\#\text{Sat}(\phi) - \#\text{Sat}(\phi'))$$

$$= 2^T \cdot h(\phi, \phi') = g(G, G').$$

Also, $(\phi, \phi') \notin$ DiffSat$_{=g}$ if

$$\#\text{Sat}(\phi) - \#\text{Sat}(\phi') = 0 \text{ iff}$$

$\#\text{PerfMatch}(G) - \#\text{PerfMatch}(G') = 2^T \cdot (\#\text{Sat}(\phi) - \#\text{Sat}(\phi')) = 0.$ □

So, any problem in the class C$_=$P (PP, WPP) is definable by the TotP function #PerfMatch. This is an alternative proof for C$_{=\text{tot}}$P = C$_=$P (P$_{\text{tot}}$P = PP and WP$_{\text{tot}}$P = WPP, respectively).

Corollary 4.10. *(a)* $\mathsf{C_{=tot}P} = \mathsf{C_{=}P}$.

(b) $\mathsf{P_{tot}P} = \mathsf{PP}$.

(c) $\mathsf{WP_{tot}P} = \mathsf{WPP}$.

Proof. (a) Let $L \in \mathsf{C_{=}P}$. Then, it holds that $x \in L$ iff $\#\textsc{Sat}(h_1(x)) - \#\textsc{Sat}(h_2(x)) = 0$ iff $\#\textsc{PerfMatch}(h_3(x)) - \#\textsc{PerfMatch}(h_4(x)) = 0$, for some $h_i \in \mathsf{FP}$, $1 \leq i \leq 4$. So, define the TotP functions $f_1 = \#\textsc{PerfMatch} \circ h_3$ and $f_2 = \#\textsc{PerfMatch} \circ h_4$. Then f_1, f_2 are TotP functions and we have that $x \in L$ iff $f_1(x) - f_2(x) = 0$.

The proofs of (b) and (c) are completely analogous. □

4.1.6 An exponential lower bound result for the problem $\textsc{DiffPerfMatch}_{=g}$

Regarding the class SPP we cannot prove a similar result for the problem $\textsc{DiffPerfMatch}_{=1}$. However, we can prove the following fact.

Proposition 4.13. *The problem* $\textsc{DiffSat}_{=1}$ *is reducible to* $\textsc{DiffPerfMatch}_{=g}$.

Proof. Consider two **3CNF** formulas (ϕ, ϕ'), with n variables and $m = \mathcal{O}(n)$ clauses, such that either $\#\textsc{Sat}(\phi) - \#\textsc{Sat}(\phi') = 1$ or $\#\textsc{Sat}(\phi) - \#\textsc{Sat}(\phi') = 0$ holds.

Then, using the polynomial-time reduction of [52] two graphs G, G' can be constructed such that

$$\#\textsc{PerfMatch}(G) - \#\textsc{PerfMatch}(G') = c^{|V|} \cdot (\#\textsc{Sat}(\phi) - \#\textsc{Sat}(\phi'))$$

where $|V| = \max\{|V(G)|, |V(G')|\}\} = \mathcal{O}(n+m)$ and $c \in (1,2)$ is a constant depending on ϕ, ϕ' and can be computed in polynomial time. Also, the graphs G and G' have $\mathcal{O}(|V|)$ edges.

So, $\textsc{DiffSat}_{=1}$ on input (ϕ, ϕ') can be reduced to $\textsc{DiffPerfMatch}_{=g}$ on input (G, G'), where $g(G, G') = c^{|V|}$. □

Remark 4.5. *Proposition 4.13 states that the smallest possible non-zero difference between the number of satisfying assignments of two given* **3CNF** *formulas can be translated to an exponentially large difference between the number of perfect matchings of two graphs. In addition, we can efficiently compute this number.*

Curticapean proved that under ETH, the problem $\textsc{DiffPerfMatch}_{=0}$ has no $2^{o(m)}$ time algorithm on simple graphs with m edges [52, Theorem 7.6]. The satisfiability of a **3CNF**

formula ϕ is reducible to the difference #PerfMatch on two different graphs, such that the number of perfect matchings of the two graphs is equal iff ϕ is unsatisfiable. The reduction follows the steps of the reductions that are used in the proof of Proposition 4.12.

Using the reduction of Proposition 4.13, we can prove the following corollary. We consider the version of rETH, which asserts that probabilistic algorithms cannot decide if a given **3CNF** formula with n variables and $\mathcal{O}(n)$ clauses is satisfiable in time $\exp(o(n))$.

Corollary 4.11. *Under rETH there is no randomized* $\exp(o(m))$ *time algorithm for* DiffPerfMatch$_{=g}$ *on simple graphs with* m *edges.*

Proof. Given rETH we cannot decide whether a given **3USat** formula ϕ with m clauses is satisfiable using a randomized algorithm that runs in time $\exp(o(m))$ [45]. By applying the reduction described in [52, Lemma 7.3] we can construct two unweighted graphs G and G' with $\mathcal{O}(m)$ vertices and edges, such that

- if ϕ is unsatisfiable, then #PerfMatch(G) − #PerfMatch$(G') = 0$,
- if ϕ is satisfiable, then #PerfMatch(G) − #PerfMatch$(G') = c^{|V|}$, where $|V| = \max\{|V(G)|, |V(G')|\}$ and $c \in (1, 2)$ can be computed in polynomial time.

So, 3USat on ϕ can be reduced to DiffPerfMatch$_{=g}$ on input (G, G'), where $g(G, G') = c^{|V|}$. Thus an $\exp(o(m))$ time randomized algorithm for DiffPerfMatch$_{=g}$ would contradict rETH. □

Remark 4.6. *A different way to read Proposition 4.13 is the following. A positive result for* #PerfMatch *would imply a corresponding positive result for* DiffPerfMatch$_{=g}$ *and therefore, for* DiffSat$_{=1}$*. Of course, this positive result would be an exponential-time algorithm for these problems!*

For example, an fpras *for* #PerfMatch *would yield an algorithm that distinguishes between*

$$\#\text{PerfMatch}(G) - \#\text{PerfMatch}(G') = c^n \text{ and } \#\text{PerfMatch}(G) - \#\text{PerfMatch}(G') = 0$$

with high probability in time $\mathcal{O}(\frac{2^m}{c^n})$*, where*

$$n = \max\{|V(G)|, |V(G')|\} \text{ and } m = \max\{|E(G)|, |E(G')|\} = \mathcal{O}(n).$$

So, in time $\mathcal{O}(d_1^n)$*, where* $d_1 \in (1, 2)$*.*

Because of the reduction of Proposition 4.13, the above algorithm is also an algorithm that distinguishes between $\#\text{SAT}(\phi) - \#\text{SAT}(\phi') = 1$ *and* $\#\text{SAT}(\phi) - \#\text{SAT}(\phi') = 0$ *with high probability in time* $\mathcal{O}(d_2^n)$, $d_2 \in (1, 2)$, *where* n *is the number of variables in* ϕ, ϕ'. *The same kind of algorithm would then exist for all the problems in* SPP. *Among them*

1. *is the well-studied* GraphIsomorphism *[17], which is one of the* NP *problems that has not been proven to be either* NP*-complete, or polynomial-time solvable so far [106, 19],*
2. *all the problems in* UP, *since* UP $\subseteq$ SPP *(see Proposition 4.1).*

4.2 Discussion of results

We conclude that knowing the least (resp. most) significant bit of a TotP function is as powerful as knowing the least (resp. most) significant bit of a #P function. Also, the closure of TotP under subtraction is the class GapP, i.e. it coincides with the closure of #P under subtraction. This means that every gap-definable class is equivalent to a class defined by some property of the gap of two TotP functions.

For the classes $\oplus$P, Mod_kP, SPP, WPP, $\text{C}_=$P, and PP we obtained a family of complete problems that are defined by TotP-complete problems under parsimonious reductions (and not by #P-complete), or equivalently, by problems in P (and not by NP-complete ones). This was not surprising for $\oplus$P and $\text{C}_=$P, since complete problems defined by TotP functions (not even known to be TotP-complete) were already known (see Proposition 4.6 and Proposition 4.11, respectively). Interestingly, for the class $\text{C}_=$P this complete problem is defined by the function #PerfMatch. We determined analogous complete problems for WPP and PP.

Moreover, we showed that every SPP problem is decidable by the difference of #PerfMatch on two graphs, which is promised to be either exponentially large or zero. The value of the difference is exponentially large iff the instance of the SPP problem is a 'yes' instance. We commented on the fact that an fpras for #PerfMatch would yield a randomized exponential-time algorithm for every SPP problem (see Remark 4.6), such as GraphIsomorphism and USat.

Chapter 5

Descriptive complexity of counting problems the decision version of which is easy

In order to determine classes the problems of which admit efficient exact or approximate computation, Arenas et al. [15] suggest to focus on classes that are contained in TotP and are *robust* in the following sense: either they are closed under addition, multiplication, and subtraction by one, or they have natural complete problems under parsimonious reductions. In general, when we consider counting complexity classes, closure under addition and multiplication is desirable, but also the three aforementioned closure properties allow us to manipulate witnesses of the corresponding computational problems (add or remove witnesses). The latter requirement guarantees that the class has a natural representative and the choice of parsimonious reductions is due to the properties of these reductions we discuss in Chapter 2.

In the next paragraph we refer to some classes which are subclasses of TotP and either satisfy the requirement of being robust or are contained in FPRAS. Most of them are defined in the context of descriptive complexity.

$\#\Sigma_1$ and $\#\mathsf{R}\Sigma_2$ are classes of counting problems that are reducible to #DNF under product reductions [132], so problems that admit an fpras. The class $\Sigma\mathsf{QSO}(\Sigma_2\text{-Horn})$ has a natural complete problem, namely #DisjHornSat, the problem of counting satisfying assignments for disjunctions of **Horn** formulas [15]. The class $\Sigma\mathsf{QSO}(\Sigma_1[\mathsf{FO}])$ is not known to have a natural complete problem, but it is closed under addition, multiplication, and subtraction by one [15]. SpanL, which is the class of functions that count the number of different outputs of nondeterministic polynomial-time transducers, has a natural complete problem, namely #NFA, the problem of counting the number of strings smaller than a string x that are accepted by a given

NFA [9]. A quasi-polynomial randomized approximation scheme for #NFA was known for 25 years [96, 79], before an fpras was designed for the problem [14]. The latter result yields an fpras for every problem in the class SpanL. It also makes SpanL the first and only class so far with a Turing machine-based definition that is a subclass of FPRAS, but is not contained in FP (under standard assumptions).

Already known relationships among the aforementioned classes and some other ones defined in [132, 15] are depicted in Figure 5.1.

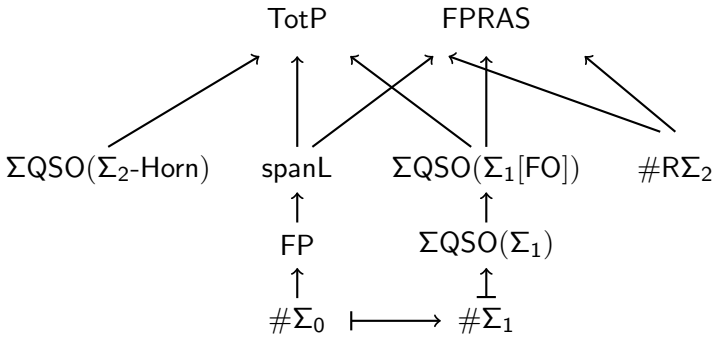


Figure 5.1: Inclusions and separations that have been shown in [132, 123, 15, 14] under no assumption. The following notation is used: $A \to B$ denotes $A \subseteq B$ and $A \mapsto B$ denotes $A \subsetneq B$.

5.1 Two robust subclasses of TotP: ΣQSO(Σ$_2$-2SAT) and #Π$_2$-1VAR

In this section, we complement previous work on robust subclasses of TotP. We introduce two classes in the context of descriptive complexity, that have natural complete problems, the decision versions of which are in the class P. Analogously to the classes #Σ$_1$ and ΣQSO(Σ$_2$-Horn) for which the problems #DNF and #DisjHornSat are complete, repsectively, the classes ΣQSO(Σ$_2$-2SAT) and #Π$_2$-1VAR introduced here, have the complete problems #Disj2Sat and #MonSat, respectively.

We prove that several natural counting problems lie in these classes. Finally, we do not expect these classes to be contained in FPRAS, since we prove that this would imply RP = NP.

5.1.1 The class ΣQSO(Σ$_2$-2SAT)

First, we define the syntax and semantics of **ΣQSO(Σ$_2$-2SAT)** formulas. To this end, we will make use of the framework introduced by Arenas et al. [15] and described here in Subsection 1.4.2.

We define a literal to be either of the form $X(\overrightarrow{x})$ or $\neg X(\overrightarrow{x})$, where X is a second-order variable and $\overrightarrow{x}$ is a tuple of first-order variables. A 2SAT clause over σ is a formula of the form $\phi_1 \vee \phi_2 \vee \phi_3$, where each of the ϕ_i's, $1 \leq i \leq 3$, can be either a literal or a first-order formula over σ. In addition, at least one of them is a first-order formula. The set of $\mathbf{\Sigma_2\text{-}2SAT}$ formulas over σ are defined as follows.

$$\psi := \exists \overrightarrow{x} \forall \overrightarrow{y} \bigwedge_{j=1}^{k} C_j(\overrightarrow{x}, \overrightarrow{y})$$

where $\overrightarrow{x}, \overrightarrow{y}$ are sequences of first-order variables, $k \in \mathbb{N}$, and C_j are 2SAT clauses for every $1 \leq j \leq k$.

The set of $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ formulas over σ is given by the following grammar.

$$\alpha := \ \phi \ \mid \ s \ \mid \ (\alpha + \alpha) \ \mid \ \Sigma x.\alpha \ \mid \ \Sigma X.\alpha \tag{5.1}$$

where ϕ is a $\mathbf{\Sigma_2\text{-}2SAT}$ formula, $s \in \mathbb{N}$, x is a first-order variable, and X is a second-order variable. So the syntax of $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ formulas includes only the counting operators of addition $+$, Σx, ΣX, and in (5.1), ϕ is restricted to be a $\mathbf{\Sigma_2\text{-}2SAT}$ formula.

Since $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ is a fragment of **QSO** formulas, the semantics of $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ formulas can be obtained from Table 1.3 .

At this point, it is clear that for any $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ formula α, a function $[[\alpha]]$ is defined.

Below we show that the class $\#\mathsf{RH}\Pi_1$ defined in [59] is contained in the class $\Sigma\mathsf{QSO}(\Sigma_2\text{-}\mathsf{2SAT})$ defined presently.

Definition 5.1 ([59]). *A function f is in the class $\#\mathsf{RH}\Pi_1$ if, on any input structure $\mathcal{A}$, f can be expressed in the form*

$$f(\mathcal{A}) = |\{\langle \overrightarrow{X}, \overrightarrow{x} \rangle : \mathcal{A} \models \forall \overrightarrow{y} \psi(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{X})\}|$$

*where ψ is an unquantified **CNF** formula, in which each clause has at most one occurrence of an unnegated variable from $\overrightarrow{X}$ and at most one occurrence of a negated variable from $\overrightarrow{X}$.*

Proposition 5.1. $\#\mathsf{RH}\Pi_1 \subseteq \Sigma\mathsf{QSO}(\Sigma_2\text{-}\mathsf{2SAT})$.

Proof. Alternatively, the function f on a structure $\mathcal{A}$ can be expressed in the form

$$f(\mathcal{A}) = [[\Sigma \overrightarrow{X}.\Sigma \overrightarrow{x}.\forall \overrightarrow{y} \psi(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{X})]](\mathcal{A}).$$

The **Restricted-Horn** formula ψ is also a **2SAT** formula. Therefore, $f \in \Sigma\mathsf{QSO}(\Sigma_2\text{-}\mathsf{2SAT})$. □

Proposition 5.2. *The following graph problems belong to* $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$.

1. #2Col *(counting 2 colorings of a graph).*
2. #IS *(counting independent sets of any size in a graph).*

Proof. 1. Consider an input graph represented as a finite ordered structure $\mathcal{G} = \langle V, E\rangle$. Then, #2Col is defined by:

$$\alpha_{2Col} = \Sigma X.\forall x\forall y\big(E(x,y) \rightarrow (X(x) \vee X(y)) \wedge (\neg X(x) \vee \neg X(y))\big).$$

2. The problem #IS is defined by:

$$\alpha_{IS} = \Sigma X.\forall x\forall y\big(E(x,y) \rightarrow (\neg X(x) \vee \neg X(y))\big).$$

□

The class $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$ contains problems that are tractable, such as #2Col, which is known to be computable in polynomial time [61]. It also contains the problem #IS, which is AP-interreducible with #Sat [59]. Finally, all the problems in $\#\mathsf{RH}\Pi_1$, such as #BIS, #1P1NSat, and #Downsets [59], belong to $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$ as well. The last three problems are complete for $\#\mathsf{RH}\Pi_1$ under approximation preserving reductions and are believed to be of intermediate complexity: neither they admit an fpras nor they are as hard as #Sat.

We next show that a generalization of #2Sat, which we will call #Disj2Sat, is complete for $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$ under parsimonious reductions.

Membership of #Disj2Sat in $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$

In propositional logic, a **2SAT** formula is a conjunction of clauses that each one contains at most two literals. In this subsection we assume that clauses of **2SAT** formulas consist of exactly two literals, since we can rewrite any clause of the form l, where l is a literal, as $l \vee l$.

Definition 5.2. #Disj2Sat.
Input: A formula ϕ which is a disjunction of **2SAT** *formulas.*
Output: The number of satisfying assignments of ϕ.

For example, the formula

$$\big((x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)\big) \vee \big((x_2 \vee \neg x_3) \wedge (x_3)\big)$$

is a disjunction of two **2SAT** formulas, where $\big((x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)\big)$ is one of its two disjuncts, $(x_1 \vee x_2)$ is one of its four clauses, and $\neg x_1$ is one of its seven literals. The clause (x_3) can be rewritten as $(x_3 \vee x_3)$.

Theorem 5.1. #Disj2Sat $\in \Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$.

Proof. Consider the vocabulary $\sigma = \{C_1^3, C_2^3, C_3^3, C_4^3, D^2\}$. This vocabulary can encode any formula which is a disjunction of **2SAT** formulas. More precisely, $C_1(c,x,y)$ iff clause c is of the form $x \vee y$, $C_2(c,x,y)$ iff c is $\neg x \vee y$, $C_3(c,x,y)$ iff c is $x \vee \neg y$, $C_4(c,x,y)$ iff c is $\neg x \vee \neg y$, and $D(d,c)$ iff clause c appears in the disjunct d.

Let ϕ be an input to #Disj2Sat encoded by an ordered structure $\mathcal{A} = \langle A, C_1, C_2, C_3, C_4, D\rangle$, where the universe A consists of elements representing variables, clauses, and disjuncts. Then, it holds that the number of satisfying assignments of ϕ is equal to $[[\Sigma T.\psi(T)]](\mathcal{A})$, where

$$\begin{aligned}\psi(T) := \exists d \forall c \forall x \forall y \big((\neg D(d,c) \vee \neg C_1(c,x,y) \vee T(x) \vee T(y)) \wedge \\ (\neg D(d,c) \vee \neg C_2(c,x,y) \vee \neg T(x) \vee T(y)) \wedge \\ (\neg D(d,c) \vee \neg C_3(c,x,y) \vee T(x) \vee \neg T(y)) \wedge \\ (\neg D(d,c) \vee \neg C_4(c,x,y) \vee \neg T(x) \vee \neg T(y))\big)\end{aligned}$$

Thus, #Disj2Sat is defined by $\Sigma T.\psi(T)$ which is a $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ formula. □

Hardness of #Disj2Sat

Suppose we have a formula α in $\mathbf{\Sigma QSO(\Sigma_2\text{-}2SAT)}$ and an input structure $\mathcal{A}$ over a vocabulary σ. We describe a polynomial-time reduction that given α and $\mathcal{A}$, it returns a propositional formula $\phi_{\alpha_\mathcal{A}}$ which is a disjunction of **2SAT** formulas and it holds that $[[\alpha]](\mathcal{A}) = \#\text{Disj2Sat}(\phi_{\alpha_\mathcal{A}})$. The reduction is a parsimonious reduction, i.e. it preserves the values of the functions involved.

Theorem 5.2. #Disj2Sat *is hard for* $\Sigma\mathsf{QSO}(\Sigma_2\text{-}2\mathsf{SAT})$ *under parsimonious reductions.*

Proof. By Proposition 1.11, α can be written in $\mathbf{\Sigma_2\text{-}2SAT}$-SNF, that is, in the following form.

$$\sum_{i=1}^{m} \Sigma \overrightarrow{X}_i . \Sigma \overrightarrow{x} . \exists \overrightarrow{y} \forall \overrightarrow{z} \bigwedge_{j=1}^{n} C_j^i(\overrightarrow{X}_i, \overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$$

where each $\overrightarrow{X}_i$ is a sequence of second-order variables and each C_j^i is a 2SAT clause. Each term of the sum can be replaced by $\Sigma \overrightarrow{X} . \Sigma \overrightarrow{x} . \exists \overrightarrow{y} \forall \overrightarrow{z} \bigwedge_{j=1}^{n} C_j^i(\overrightarrow{X}_i, \overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}) \wedge \bigwedge_{X \notin \overrightarrow{X}_i} \forall \overrightarrow{u} X(\overrightarrow{u})$ where $\overrightarrow{X}$

is the union of all $\overrightarrow{X}_i$. Now we have expressed α in the following form.

$$\sum_{i=1}^{m} \Sigma \overrightarrow{X}. \Sigma \overrightarrow{x}. \exists \overrightarrow{y} \forall \overrightarrow{z} \bigwedge_{j=1}^{n} \phi_j^i(\overrightarrow{X}, \overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}).$$

The next step is to expand the first-order and sum quantifiers and replace their variables with first-order constants from the universe A.

In this way, we obtain

$$\alpha_{\mathcal{A}} := \sum_{i=1}^{m} \sum_{\overrightarrow{a} \in A^{|\overrightarrow{x}|}} \Sigma \overrightarrow{X}. \bigvee_{\overrightarrow{b} \in A^{|\overrightarrow{y}|}} \bigwedge_{i=1}^{n} \bigwedge_{\overrightarrow{c} \in A^{|\overrightarrow{z}|}} \phi_j^i(\overrightarrow{X}, \overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}).$$

Each first-order subformula of ϕ_j^i has no free-variables and is either satisfied or not satisfied by $\mathcal{A}$, so we can replace it by $\top$ or $\bot$ respectively. Also, after grouping the sums and the conjunctions, we get

$$\sum_{i=1}^{m'} \Sigma \overrightarrow{X}. \bigvee_{j=1}^{n_1} \bigwedge_{k=1}^{n_2} \psi_{j,k}^i(\overrightarrow{X}).$$

The formulas $\psi_{j,k}^i(\overrightarrow{X})$ are conjunctions of clauses that consist of $\bot$, $\top$, and at most two literals of the form $X_t(\overrightarrow{a}_l)$ or $\neg X_t(\overrightarrow{a}_l)$ for some second-order variable X_t and some tuple of first-order constants $\overrightarrow{a}_l$. We can eliminate the clauses that contain a $\top$ and remove $\bot$ from the clauses that contain it. After this simplification, some combinations of variable-constants may not appear in the remaining formula. For any such combination $X(\overrightarrow{a})$, we add a clause $\psi_{X,\overrightarrow{a}} := X(\overrightarrow{a}) \vee \neg X(\overrightarrow{a})$, since $X(\overrightarrow{a})$ can have any truth value.

So, we have reformulated the above formula and we obtain

$$\sum_{i=1}^{m'} \Sigma \overrightarrow{X}. \bigvee_{j=1}^{n_1} \bigwedge_{k=1}^{n_2'} \psi_{j,k}^i(\overrightarrow{X}).$$

After replacing every appearance of $X_t(\overrightarrow{a}_l)$ by a propositional variable x_{tl}, the part $\bigvee_{j=1}^{n_1} \bigwedge_{k=1}^{n_2'} \psi_{j,k}^i(\overrightarrow{X})$ becomes a disjunction of **2SAT** formulas. Finally, we introduce m' new propositional variables $x_1, ...x_{m'}$ and define

$$\phi_{\alpha_{\mathcal{A}}} := \bigvee_{i=1}^{m'} \bigvee_{j=1}^{n_1} \bigwedge_{k=1}^{n_2'} \psi_{j,k}^i \wedge x_i \bigwedge_{l \neq i} \neg x_l.$$

The formula $\phi_{\alpha_{\mathcal{A}}}$ is a disjunction of **2SAT** formulas and the number of its satisfying assignments is equal to $[[\alpha]](\mathcal{A})$. Moreover, every transformation we described above, requires polynomial time in the size of the input structure $\mathcal{A}$. □

It is known that #2SAT has no fpras unless RP = NP, since it is equivalent to counting all independent sets in a graph [59]. Thus, problems hard for ΣQSO(Σ_2-2SAT) under parsimonious reductions, cannot admit an fpras unless RP = NP.

Proposition 5.3. ΣQSO(Σ_2-2SAT) $\not\subseteq$ FPRAS *unless* RP = NP.

Inclusion of ΣQSO(Σ_2-2SAT) in TotP

Several problems in ΣQSO(Σ_2-2SAT), like #1P1NSAT, #IS, #2COL, and #2SAT are also in TotP. We next prove that this is not a coincidence.

Theorem 5.3. ΣQSO(Σ_2-2SAT) $\subseteq$ TotP.

Proof. Since TotP is exactly the closure under parsimonious reductions of self-reducible functions in #PE [123], it suffices to show that the ΣQSO(Σ_2-2SAT)-complete problem #DISJ2SAT is such a function.

First of all, the decision version DISJ2SAT of #DISJ2SAT belongs to P. Thus #DISJ2SAT $\in$ #PE. Secondly, every counting function associated with the problem of counting satisfying assignments of a propositional formula is self-reducible. So #DISJ2SAT has this property as well. Therefore, any **ΣQSO(Σ_2-2SAT)** formula α defines a function $[[\alpha]]$ that belongs to TotP. □

Corollary 5.1. #RH$\Pi_1 \subseteq$ TotP.

Proof. Immediate from Proposition 5.1 and Theorem 5.3. □

5.1.2 The class #Π_2-1VAR

To define the second class #Π_2-1VAR, we will use the framework presented in [132] and described in Subsection 1.4.1.

We say that a counting problem #B belongs to the class #Π_2-1VAR if for any ordered structure $\mathcal{A}$ over a vocabulary σ, which is an input to #B, it holds that

$$\#\mathrm{B}(\mathcal{A}) = |\{\langle X\rangle : \mathcal{A} \models \forall \vec{y} \exists \vec{z} \psi(\vec{y}, \vec{z}, X)\}|.$$

The formula $\psi(\vec{y}, \vec{z}, X)$ is of the form $\phi(\vec{y}, \vec{z}) \wedge X(\vec{z})$, where ϕ is a first-order formula over σ and X is a positive appearance of a second-order variable. We call the formula ψ a variable, since

it contains only one second-order variable. Moreover, we allow counting only the assignments to the second-order variable X under which the structure $\mathcal{A}$ satisfies $\forall\overrightarrow{y}\exists\overrightarrow{z}\psi(\overrightarrow{y},\overrightarrow{z},X)$.

Proposition 5.4. *The following graph problems belong to* #Π_2-1VAR.

1. #VC *(counting vertex covers of any size in a graph).*
2. #IS *(counting independent of any size in a graph).*
3. #DS *(counting dominating sets of any size in a graph).*
4. #EC *(counting edge covers of any size in a graph).*

Proof. 1. An input graph G to #VC can be encoded as a finite structure $\mathcal{G}$ using the vocabulary $\sigma = \{V^1, EndPoint^2\}$. The universe is the set of vertices and edges. $V(u)$ iff vertex u is a vertex and $EndPoint(e,u)$ iff vertex u is an endpoint of edge e. Then,

$$\#\mathrm{VC}(\mathcal{G}) = |\{\langle VC\rangle \mid \mathcal{G} \models \forall x\exists y\big((V(x) \vee EndPoint(x,y)) \wedge V(y) \wedge VC(y)\big)\}|.$$

So, #VC $\in$ #Π_2-1VAR.

2. An input graph G to #IS is encoded as a finite structure $\mathcal{G}$ exactly as in the case of #VC. Then,

$$\#\mathrm{IS}(\mathcal{G}) = |\{\langle NIS\rangle \mid \mathcal{G} \models \forall x\exists y\big((V(x) \vee EndPoint(x,y)) \wedge V(y) \wedge NIS(y)\big)\}|$$

where $NIS(y)$ indicates that vertex y is not in the independent set. #IS is expressed in the exact same way as #VC, since the number of vertex covers in a graph is equal to the number of independent sets. Hence, #IS $\in$ #Π_2-1VAR.

3. An input graph G to #DS can be encoded as a finite structure $\mathcal{G}$ over the vocabulary $\sigma = \{E^2\}$, where the universe is the set of vertices and E is the edge relation. Then,

$$\#\mathrm{DS}(\mathcal{G}) = |\{\langle DS\rangle \mid \mathcal{G} \models \forall x\exists y\big((x = y \vee E(x,y)) \wedge DS(y)\big)\}|.$$

4. An input graph G to #EC can be encoded as a finite structure $\mathcal{G}$ as in the case of #DS. Then,

$$\#\mathrm{EC}(\mathcal{G}) = |\{\langle EC\rangle \mid \mathcal{G} \models \forall x\exists y\big(E(x,y) \wedge EC(x,y)\big)\}|.$$

□

Completeness of #MonSat for #Π_2-1VAR

Definition 5.3. #MonSat.

Input: A formula ϕ in CNF, in which all variables appear positive.

Output: The number of satisfying assignments of ϕ.

Theorem 5.4. #MonSat $\in$ #Π_2-1VAR.

Proof. Consider the vocabulary $\sigma = \{Var^1, C^2\}$ with $Var(x)$ to indicate that x is a variable and the binary relation $C(c,x)$ to indicate that the variable x appears in the clause c. Given a σ-structure $\mathcal{A} = \langle A, Var, C\rangle$ that encodes a formula ϕ, which is an input to #MonSat, it holds that #MonSat(ϕ)=$|\{\langle T\rangle : \mathcal{A} \models \forall c \exists x\big((Var(c) \vee C(c,x)) \wedge Var(x)\big) \wedge T(x)\big)\}|$. Therefore, #MonSat $\in$ #Π_2-1VAR. □

Theorem 5.5. #MonSat *is hard for* #Π_2-1VAR *under product reductions.*

Proof. We show that there is a polynomial-time product reduction from any #B $\in$ #Π_2-1VAR to #MonSat. This means that there are polynomial-time computable functions g and h, such that for every strucrure $\mathcal{A}$ that is an input to #B we have #B($\mathcal{A}$) = #MonSat$\big(g(\mathcal{A})\big) \cdot h(\mathcal{A})$.

Suppose we have a problem #B $\in$ #Π_2-1VAR and a structure $\mathcal{A}$ over σ. Then, there exists a formula ψ of the form $\psi(\overrightarrow{y}, \overrightarrow{z}, X) = \phi(\overrightarrow{y}, \overrightarrow{z}) \wedge X(\overrightarrow{z})$ such that #B($\mathcal{A}$) = $|\{\langle X\rangle : \mathcal{A} \models \forall \overrightarrow{y} \exists \overrightarrow{z} \psi(\overrightarrow{y}, \overrightarrow{z}, X)\}|$.

The formula $\forall \overrightarrow{y} \exists \overrightarrow{z} \psi(\overrightarrow{y}, \overrightarrow{z}, X)$ can be written in the form

$$\bigwedge_{\overrightarrow{a} \in A^{|\overrightarrow{y}|}} \bigvee_{\overrightarrow{b} \in A^{|\overrightarrow{z}|}} \phi(\overrightarrow{a}, \overrightarrow{b}) \wedge X(\overrightarrow{b}).$$

By substituting first-order subformulas by $\top$ or $\bot$ and simplifying, we obtain $\chi_{\psi_{\mathcal{A}}} := \bigwedge_{i=1}^{n_1} \bigvee_{j=1}^{n_2} X(\overrightarrow{b}_{i,j})$, where each $\overrightarrow{b}_{i,j}$ is a tuple of first-order constants. To define $\chi_{\psi_{\mathcal{A}}}$, we have simplified the subformulas containing $\bot$ and $\top$. As a result, there may be some combinations of the second-order variable X and first-order constants that do not appear in $\chi_{\psi_{\mathcal{A}}}$. Let $n(\mathcal{A})$ be the number of these combinations. The last transformation consists of replacing every $X(\overrightarrow{b}_{i,j})$ with a propositional variable x_{ij}, so we get the output of the function g, which is $g(\mathcal{A}) := \bigwedge_{i=1}^{n_1} \bigvee_{j=1}^{n_2} x_{i,j}$. This formula has no negated variables, so it can be an input to #MonSat. Finally, since the missing $n(\mathcal{A})$ variables can have any truth value, we have #B($\mathcal{A}$) = #MonSat$\big(g(\mathcal{A})\big) \cdot 2^{n(\mathcal{A})}$. □

#MonSat does not admit an fpras if a variable can appear in 6 clauses unless RP = NP. In the case of **monotone CNF** formulas where each variable appears at most 5 times, the problem has an fptas [114]. So, we have the following result.

Proposition 5.5. $\#\Pi_2$-1VAR $\not\subseteq$ FPRAS *unless* RP = NP.

Inclusion of $\#\Pi_2$-1VAR in TotP

Theorem 5.6. $\#\Pi_2$-1VAR $\subseteq$ TotP.

Proof. #MonSat $\in$ TotP, since it has an easy decision version; a **monotone CNF** formula has always a satisfying assignment, the one in which every variable is set to true. Also, TotP is closed under product reductions by Proposition 1.10. Thus, every counting problem in $\#\Pi_2$-1VAR belongs to TotP. □

The following corollary can be obtained using standard reductions between counting problems. However, it is interesting that any problem in $\#\Pi_2$-1VAR inherits positive results for #MonSat, since product reductions preserve the existence of fptas and fpras [132].

Corollary 5.2. *(a)* #VC, #IS, *and* #EC *admit a fully polynomial-time deterministic scheme (fptas) in graphs with maximum degree 5.*

(b) #DS *admits an fptas in graphs with maximum degree 4.*

Proof. Consider #VC. If each vertex has degree at most 5, then after the reduction to #MonSat, we come up with a formula which is equivalent to a monotone formula in which every variable appears in at most 5 clauses. But #MonSat restricted to such inputs, admits an fptas [114] and product reductions preserve the existence of fptas (or fpras) [132].

Similar arguments hold for #IS, #DS, and #EC as well. □

Remark 5.1. *Corollary 5.2 is not surprising. We could obtain the same result via immediate reductions from* #VC, #IS, #DS, *and* #EC *to* #MonSat. *In addition the following algorithms are known for the aforementioned problems: (a)* #EC *admits an fptas [113], (b)* #IS *in graphs with maximum degree* 5 *has an fptas [155], (c) the problem of counting dominating sets in regular graphs admits an fptas when the maximum degree of the graph is* ≤ 5 *or* ≥ 199 *[37].*

The introduction of $\#\Pi_2$-1VAR *can yield similar results for any problem that lies in this class.*

Although completeness of #MonSat for #Π_2-1VAR under product reductions, is the first completeness result for #MonSat under reductions stronger than Turing, we would like to prove #MonSat complete for a class under reductions for which the class is downwards closed. This is a question raised by Hemaspaandra et al. [84]. Regarding the closure under parsimonious and product reductions of #MonSat and #Π_2-1VAR, we can say the following.

Proposition 5.6. *(a)* $\text{Closure}_{\leq^{\text{p}}_{\text{pars}}}(\#\text{MonSat}) \subseteq \text{Closure}_{\leq^{\text{p}}_{\text{pr}}}(\#\text{MonSat})$.

(b) $\text{Closure}_{\leq^{\text{p}}_{\text{pr}}}(\#\Pi_2\text{-1VAR}) \subseteq \text{Closure}_{\leq^{\text{p}}_{\text{pr}}}(\#\text{MonSat}) \subseteq \text{Closure}_{\leq^{\text{p}}_{\text{pr}}}(\text{TotP})$. *The first inclusion is strict unless the class* #Π_2-1VAR *is closed under product reductions. The second one is strict unless* #MonSat *is complete for* TotP *under product reductions.*

Relationships among TotP, FPRAS, and the classes defined here are depicted in Figure 5.2.

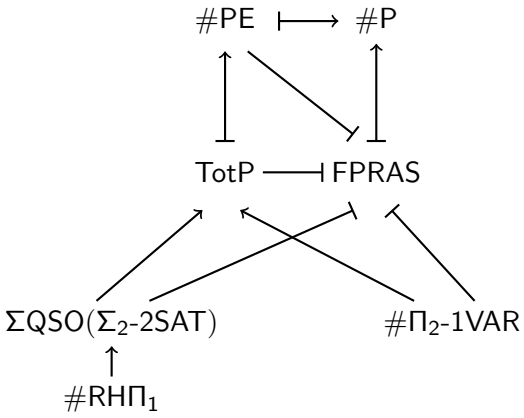


Figure 5.2: Inclusions and separations in the case of P = RP ≠ NP. The following notation is used: $A \to B$ denotes $A \subseteq B$, $A \dashv B$ denotes $A \not\subseteq B$, and $A \mapsto B$ denotes $A \subsetneq B$.

5.2 A logical characterization of TotP

The logical characterization of #P is based on writing a **ΣQSO**(**FO**) formula that is essentially a sum over all accepting paths. In the case of TotP we start from the logic **ΣQSO**(∃**SO**) which allows us to express the existence of a branching. Once we find a branching we want to add 1 and continue recursively. Since the Turing machine is of polynomial time, the recursion suffices to be of polynomial depth.

So, at first, we describe a logic, namely $\mathbf{R\Sigma Q}_{\exists\mathbf{SO}}$, which is more expressive than **ΣQSO**(**FO**). It is **ΣQSO**(∃**SO**) equipped with recursion. In this logic, we can express every TotP problem based on the fact that it is self-reducible. We are going to give an example for #DNF.

Formally, in Subsection 5.2.1, we define logics equipped with a polynomially-bounded partial fixed point operator over functions $f : (\mathcal{P}(A^k))^l \to \mathbb{N}$, which means that a function f takes l sets in $\mathcal{P}(A^k)$ as arguments (in other words, l relations of arity k).

5.2.1 Functions over relations and recursion in QSO

First, we add the infinite set SOFS of second-order function symbols f to $\mathbf{\Sigma QSO}(\exists\mathbf{SO})$. Each symbol has an associated arity, denoted by $\mathsf{arity}(f)$. The set of $\mathbf{SO_k\text{-}F\Sigma QSO}(\exists\mathbf{SO})$ formulas over a vocabulary σ is defined by the following grammar:

$$\alpha := \phi \mid s \mid f(X_1, ..., X_l) \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma X.\alpha \tag{5.2}$$

where $f \in$ SOFS, $\mathsf{arity}(f) = l$, and $X_1, ..., X_l$ is a sequence of second-order variables with $\mathsf{arity}(X_i) = k$ for every $1 \leq i \leq l$. Subscript $\mathbf{k}$ designates that for all $i \in \{1, ..., l\}$, X_i is of arity k. As we already explained, ϕ is restricted to be an existential second-order formula.

Given a structure $\mathcal{A}$ over σ, F is a function assignment for $\mathcal{A}$, if for every $f \in$ SOFS with $\mathsf{arity}(f) = l$,

$$F(f) : (\mathcal{P}(A^k))^l \to \mathbb{N}.$$

Given first- and second-order assignments v and V for $\mathcal{A}$, respectively, it holds that:

$$[[f(X_1, ..., X_l)]](\mathcal{A}, v, V, F) = F(f)(V(X_1), ..., V(X_l)).$$

The domain of the function symbols we introduce here, consists of sets of k-tuples of the universe. That's why we call them second-order function symbols. Function symbols defined in [15], representing functions $h : A^l \to \mathbb{N}$, can be seen as first-order function symbols.

In the same way, given a logic $\mathbf{L}$, we obtain $\mathbf{SO_k\text{-}FL}$ by adding the function symbols in SOFS into the syntax.

We define now the logic $\mathbf{RSO_k\text{-}\Sigma QSO}(\exists\mathbf{SO})$, where $\mathbf{RSO_k}$ stands for second-order recursion. The set of $\mathbf{RSO_k\text{-}\Sigma QSO}(\exists\mathbf{SO})$ formulas over σ includes $\mathbf{\Sigma QSO}(\exists\mathbf{SO})$ formulas and the formula $[\text{pbfp}_f\, \alpha](\overrightarrow{X})$, where $\overrightarrow{X} = (X_1, ..., X_l)$ is a sequence of l distinct second-order variables and $\alpha(\overrightarrow{X}, f)$ is an $\mathbf{SO_k\text{-}F\Sigma QSO}(\exists\mathbf{SO})$ formula with only one function symbol f with $\mathsf{arity}(f) = l$. The free variables of $[\text{pbfp}_f\, \alpha](\overrightarrow{X})$ are $X_1, ..., X_l$ (and f is not considered to be free).

Finally, $\mathbf{RSO\text{-}\Sigma QSO}(\exists\mathbf{SO}) = \bigcup_k \mathbf{RSO_k\text{-}\Sigma QSO}(\exists\mathbf{SO})$. **RSO** stands for second-order recursion, $\mathbf{\Sigma QSO}$ represents that addition quantifiers over second-order variables are used, and

($\exists$**SO**) represents that ϕ in (7.1) is an existential second-order formula. In sequel we denote the logic **RSO-ΣQSO**($\exists$**SO**) by $\mathbf{R\Sigma Q_{\exists SO}}$.

Define $\mathcal{SOF}_k$ to be the set of all functions $h : (\mathcal{P}(A^k))^l \to \mathbb{N}$. Let $\mathcal{A}$ be a structure over σ and $[\text{pbfp}_f\, \alpha](\overrightarrow{X})$ with $\mathsf{arity}(f) = l$. To define the semantics of $[\text{pbfp}_f\, \alpha](\overrightarrow{X})$, we interpret $\alpha(\overrightarrow{X}, f)$ as an operator T_α on $\mathcal{SOF}_k$. For every $h \in \mathcal{SOF}_k$ and $(S_1, ..., S_l) \in (\mathcal{P}(A^k))^l$, it holds that:

$$T_\alpha(h)(\overrightarrow{S}) = [[\alpha(\overrightarrow{X}, f)]](\mathcal{A}, V, F)$$

where V is a second-order assignment for $\mathcal{A}$ such that $V(X_i) = S_i$, $i \in \{1, ..., l\}$, and F is a function assignment for $\mathcal{A}$ such that $F(f) = h$.

Our first attempt would be to use the Tarksi-Knaster Theorem to prove that a least fixed point exists for an order-preserving T_α. In this direction, we define the partial order $\leq_f$ on $\mathcal{SOF}_k$ as follows: For two functions f, g with $\mathsf{arity}(f) = \mathsf{arity}(g) = l$, it holds that $f \leq_f g$ if $f(\overrightarrow{S}) \leq g(\overrightarrow{S})$ for every $\overrightarrow{S}$. However, the requirement of $(\mathcal{SOF}_k, \leq_f)$ being a complete lattice, is not satisfied. A second idea would be to consider the support of a function and define h to be a fixed point of T_α if $\mathsf{supp}(T_\alpha(h)) = \mathsf{supp}(h)$ [15]. But there are cases in which the support of the operator reaches a least fixed point, but the function we are interested in, still increases.

We resolve this problem by defining the partial fixed point of an operator T_α along the lines of [112]. Let us consider the sequence of functions $\{h_i\}_{i\in\mathbb{N}}$, $h_i : (\mathcal{P}(A^k))^l \to \mathbb{N}$, where $h_0(\overrightarrow{S}) = 0$ for every $\overrightarrow{S} \in (\mathcal{P}(A^k))^l$ and let h_{i+1} be defined as $T_\alpha(h_i)$ for every $i \in \mathbb{N}$. Then, there are two possibilities: either there exists some $n \in \mathbb{N}$ such that $h_{n+1}(\overrightarrow{S}) = h_n(\overrightarrow{S})$ for every $\overrightarrow{S} \in (\mathcal{P}(A^k))^l$, and thus we say that $h_j = h_n$ for every $j \geqslant n$, or there is no such n. If the first case holds, then $n \leq 2^{l\cdot|A|^k}$. So, it would be reasonable to define $\text{pfp}(T_\alpha)$ as follows:

$$\text{pfp}(T_\alpha) = \begin{cases} f_n & \text{if } f_n = f_{n+1}, \\ f_0 & \text{if } f_n \neq f_{n+1} \text{ for all } n \leq 2^{l\cdot|A|^k}. \end{cases}$$

Here we slightly change this definition. Let m be the maximum arity of a second-order variable bounded by a quantitative quantifier, i.e. Σ or Π, in α. Then, we define the p-bounded (partial) fixed point of an operator T_α to be the following.

$$\text{pbfp}(T_\alpha) = \begin{cases} f_n & \text{if } f_n = f_{n+1} \text{ for some } n \leq |A|^m \\ f_{|A|^m} & \text{if } f_n \neq f_{n+1} \text{ for all } n \leq |A|^m. \end{cases}$$

We define the semantics of $[\text{pbfp}_f\, \alpha](\overrightarrow{X})$ to be the p-bounded fixed point of T_α. More precisely,

$$[[\,[\text{pbfp}_f\, \alpha](\overrightarrow{X})\,]](\mathcal{A}, V) = \begin{cases} f_n(V(\overrightarrow{X})) & \text{if } f_n = f_{n+1} \text{ for some } n \leq |A|^m, \\ f_l(V(\overrightarrow{X})) & \text{otherwise, where } l = |A|^m. \end{cases} \tag{5.3}$$

We give an example of a TotP problem that belongs to $\mathsf{R\Sigma Q}_{\exists\mathsf{SO}}$.

Example 5.1. *The problem* #DNF *is in* RSO_1-$\Sigma\mathsf{QSO}(\exists\mathsf{SO})$. *Let* $\sigma = \{V^1, D^1, Pos^2, Neg^2\}$ *be the vocabulary of a* **DNF** *formula and* $\mathcal{A}_\phi = \langle A = \{v_1, v_2, ..., v_n, d_1, ..., d_{n'}\}, V, D, Pos, Neg\rangle$ *be a structure over* σ, *encoding an input* ϕ. *The universe contains* n *variables and* n' *disjuncts, relations* $V(v)$ *and* $D(d)$ *indicate that* v, d *are a variable and a disjunct repsectively, whereas* $Pos(d, v)$ *(resp.* $Neg(d, v)$*) means that variable* v *appears positive (resp. negative) in disjunct* d.

#DNF *is defined as the p-bounded fixed point of formula* $\mathsf{dnf}(True, False, f)$ *given below.*

$$\begin{aligned} \mathsf{dnf}(True, False, f) :=& \exists T\, \mathsf{sat}(T) \wedge \mathsf{EndOfRecursion}+ \\ & \Sigma T_1.\Sigma F_1.\, (\mathsf{t}_1(T_1) \wedge \mathsf{f}_1(F_1)) \cdot f(T_1, F_1)+ \\ & \Sigma T_0.\Sigma F_0.\, (\mathsf{t}_0(T_0) \wedge \mathsf{f}_0(F_0)) \cdot f(T_0, F_0) \end{aligned}$$

Now we give details for each subformula of $\mathsf{dnf}(True, False, f)$. *The idea behind this formula is that we start with empty relations* $True$ *and* $False$ *and at each recursive step, we fix a variable* v_i *to be either true or false. When all variables are set to either true or false, the recursion ends.*

1. $\mathsf{EndOfRecursion} := \forall v(V(v) \rightarrow True(v) \vee False(v))$. *Formula* $\mathsf{EndOfRecursion}$ *is true when we have assigned a truth value to every variable.*

2. $\mathsf{sat}(T) := \exists d \forall v\big(D(d) \wedge (Pos(d, v) \rightarrow T(v)) \wedge (Neg(d, v) \rightarrow \neg T(v)) \wedge (True(v) \rightarrow T(v)) \wedge (False(v) \rightarrow \neg T(v))\big)$. *Formula* $\mathsf{sat}(T)$ *states that there is a satisfying assignment for the formula* ϕ, *when we have fixed some variables to either true or false. The fixed truth values are given by the relations* $True$ *and* $False$.

3. *The minimum variable that has not been assigned a truth value yet can be defined by the formula* $\mathsf{min}(v_{min}) := \exists d\big(Pos(d, v_{min}) \vee Neg(d, v_{min})\big) \wedge \big(\neg True(v_{min}) \wedge \neg False(v_{min})\big) \wedge \forall v\big((\neg True(v) \wedge \neg False(v)) \rightarrow v \geqslant v_{min}\big)$.

4. *Formula* $t_1(T_1)$ *says that there exists a variable that appears in formula* ϕ *and has not been assigned a truth value yet and sets the value of the minimum such variable to true. So,*

$t_1(T_1) = \exists v_{min}\Big(\mathsf{min}(v_{min}) \wedge T_1(v_{min}) \wedge \forall u\big(u \neq v_{min} \rightarrow (T_1(u) \leftrightarrow True(u))\big)\Big)$. *In other words,* $t_1(T_1)$ *defines the relation* T_1 *that agrees with relation* $True$ *on the values of all variables, except one;* T_1 *is also true for the minimum variable assigned neither to true nor to false at the current step of recursion.*

5. *Formula* $f_1(F_1)$ *defines relation* F_1 *to be exactly the relation* $False$*. So,* $f_1(F_1) = \forall v(F_1(v) \leftrightarrow False(v))$.

6. *Similarly,* $t_0(T_0)$ *defines* T_0 *such that it is the relation* $True$*, whereas* $f_0(F_0)$ *defines* F_0 *to be the extension of* $False$ *that also holds for the minimum variable not assigned to either true or false yet.*

Then, $[[\,[\mathrm{pbfp}_f\,\mathsf{dnf}](True, Fasle)\,]](\mathcal{A}_\phi, V)$ *gives the number of satisfying assignments of* ϕ*, where* V *is a second-order assignment for* $\mathcal{A}_\phi$ *such that* $V(True) = \emptyset$ *and* $V(False) = \emptyset$*. Note that after* n *steps all variables are set to true or false and the recursion requires* $n+1$ *steps until* $f_0, f_1, ...$ *stabilizes. The maximum arity of a second-order variable bounded by the quantifier* Σ *is* $m = 1$ *and the size of the universe is* $n+n'$*. So,* $\#\mathrm{DNF}(\mathcal{A}_\phi) = f_{n+1}(\emptyset, \emptyset)$*, where* $n + 1 \leq |A|^m = n + n'$.

5.2.2 A logic for expressing TotP functions

Let N be an NPTM that has a binary computation tree and uses time $n^c - 1$ for inputs of size n. We assume that N is the quintuple $N = \{\mathcal{Q}, \Sigma, \delta, q_I, q_F\}$, where $\mathcal{Q} = \{q_0, ..., q_{d-1}\}$ is the set of states of size $|\mathcal{Q}| = d$, $\Sigma = \{0, 1\}$ the alphabet, $q_I = q_0$ is the initial state, and $q_F = q_{d-1}$ is the final state. Let also $k = \max\{c, \lceil \log d \rceil\}$. Here we assume that the tape of machine N is of length n^k and that once the machine accepts or rejects, it clears its tape, it moves its cursor all the way to the left, and enters a unique final state q_F. The transition relation δ maps a pair in $\mathcal{Q} \times (\Sigma \cup \{\llcorner\})$ to at most two triples in $\mathcal{Q} \times (\Sigma \cup \{\llcorner\}) \times \{Stay, Left, Right\}$. If δ maps a pair to only one triple, then we say that only choice 0 is possible. An input x of size n is encoded by a structure $\mathcal{A}_x = \langle \{0, 1, ..., n-1\}, B\rangle$, where B represents the positions where x is 1.

We are going to describe the formula $\mathsf{tot}(T, E, P, Q, f)$ such that the interpretation of $[\mathrm{pbfp}_f\,\mathsf{tot}](T, E, P, Q)$ gives the number of branchings of N, which is sufficient by Remark 1.4. Second-order variables T (contents of the tape), E (end of zeros and ones on the tape), P (position of the cursor), Q (state of the machine), each of arity k, will encode a configuration of NPTM N.

To construct tot we need the following subformulas.

$\mathsf{DetComp}(S_0, ..., S_m, \overrightarrow{t_*})$ which expresses the existence of a deterministic computation on an input, encoded by the k-ary relations T, E, P, Q until time $\overrightarrow{t_*}$.

$\mathsf{NonDetChoice}(S_0, ..., S_m, \overrightarrow{t_*})$ which expresses that $\overrightarrow{t_*}$ is the first time step a nondeterministic choice is made during the computation.

$\Delta_{\mathsf{i}}(S_0, ..., S_m, T_i, E_i, P_i, Q_i, \overrightarrow{t_*})$ which defines the contents of the tape, the position of the cursor, and the state of N at time $\overrightarrow{t_*} + 1$ if nondeterministic choice i was made at time $\overrightarrow{t_*}$, where $i \in \{0, 1\}$.

Formula $\mathsf{DetComp}(S_0, ..., S_m, \overrightarrow{t_*})$ is the formula given in the proof of Fagin's Theorem [88] with a few modifications.

$$\mathsf{DetComp}(S_0, ..., S_m, \overrightarrow{t_*}) := \mathsf{input}(S_0, ..., S_m) \wedge \mathsf{transition}(S_0, ..., S_m, \overrightarrow{t_*}) \wedge \mathsf{nonfinal}(S_0, ..., S_m, \overrightarrow{t_*}) \wedge \mathsf{deterministic}(S_0, ..., S_m, \overrightarrow{t_*})$$

where S_i, $0 \leq i \leq m$, are $2k$-ary relations such that $S_i(\overrightarrow{s}, \overrightarrow{t})$ encodes that cell $\overrightarrow{s}$ contains the symbol i at time $\overrightarrow{t}$. Symbol i is one of 0, 1, $\sqcup$ when the cursor is not on cell $\overrightarrow{s}$, or a combination $(q, \sigma) \in Q \times \{0, 1, \sqcup\}$, when the cursor is on cell $\overrightarrow{s}$. Recall that the input structure has n elements ordered by a total order $\leq$, so we can use k-tuples of these n elements, to encode n^k cells (or time steps) and define the lexicographic order on them, which again we denote by $\leq$. We are going to use the expressions $\overrightarrow{s} \leq \overrightarrow{u}$, $\overrightarrow{s} < \overrightarrow{u}$, $\overrightarrow{s} + 1$, $\overrightarrow{s} + 1$, $\overrightarrow{s_{max}}$, $\overrightarrow{s_{min}}$ which are all definable in first-order logic. Also, when we write $\overrightarrow{s_i}$, we mean the i-th smallest cell with respect to $\leq$. We use similar notation for time steps.

As for the second-order variables T, E, P, Q, $T(\overrightarrow{s_i})$ is true iff cell $\overrightarrow{s_i}$ has the symbol 1, $E(\overrightarrow{s_i})$ denotes that all cells greater than $\overrightarrow{s_i}$ have the symbol $\sqcup$, $P(\overrightarrow{s_i})$ indicates that the cursor is on cell $\overrightarrow{s_i}$, and $Q(\overrightarrow{s_i})$ means that N is in state q_i. When we write $E(\overrightarrow{s_i})$ (resp. $P(\overrightarrow{s_i})$, $Q(\overrightarrow{s_i})$), it is a shorthand for $E(\overrightarrow{s_i}) \wedge \forall \overrightarrow{u}(\overrightarrow{u} \neq \overrightarrow{s_i} \rightarrow \neg E(\overrightarrow{u}))$ (resp. $P(\overrightarrow{s_i}) \wedge \forall \overrightarrow{u}(\overrightarrow{u} \neq \overrightarrow{s_i} \rightarrow \neg P(\overrightarrow{u}))$, $Q(\overrightarrow{s_i}) \wedge \forall \overrightarrow{u}(\overrightarrow{u} \neq \overrightarrow{s_i} \rightarrow \neg Q(\overrightarrow{u})))$.

From now on, we denote the quadruple (T, E, P, Q) by $\overrightarrow{C}$ and the sequence $(S_0, ..., S_m)$ by $\overrightarrow{S}$. Note that we need both sequences $\overrightarrow{C}$ and $\overrightarrow{S}$. Relations $\overrightarrow{C}$ encode a configuration,

let's say c, which is the starting point of a deterministic computation. Then, $\overrightarrow{S}$ encode the deterministic computation of N starting at c. When the first nondeterministic choice is made, the configuration of the machine after this choice is copied again into $\overrightarrow{C}$ and the recursive computation continues.

Formulas input, transition, nonfinal, and deterministic are defined below. Given a configuration of N, formula input is used to force sequence $\overrightarrow{S} = (S_0, ..., S_m)$ to encode this configuration at time $\overrightarrow{0}$. The configuration is not necessarily the initial configuration of N and it corresponds to some state q_i, $i \in \{0, ..., d-1\}$, some contents of the tape, and a position of the cursor. These information will be passed to $\overrightarrow{S}$ and the time will be set to $\overrightarrow{0}$. So, for each state q_i we will need a different subformula $\mathsf{state_i}$, $0 \leq i \leq d-1$, which has free second-order variables $\overrightarrow{S_i} = (S_0, S_1, S_\sqcup, S_{(0,q_i)}, S_{(1,q_i)}, S_{(\sqcup,q_i)})$.

$$\begin{aligned}\mathsf{state_i}(\overrightarrow{S_i}) := \forall\overrightarrow{s}\Big(E(\overrightarrow{s}) \rightarrow \forall\overrightarrow{u}\big(\overrightarrow{u} \leq \overrightarrow{s} \rightarrow &\big(\neg T(\overrightarrow{u}) \wedge \neg P(\overrightarrow{u}) \rightarrow S_0(\overrightarrow{u}, \overrightarrow{0})\big)\wedge\\ &\big(T(\overrightarrow{u}) \wedge \neg P(\overrightarrow{u}) \rightarrow S_1(\overrightarrow{u}, \overrightarrow{0})\big)\wedge\\ &\big(\neg T(\overrightarrow{u}) \wedge P(\overrightarrow{u}) \rightarrow S_{(0,q_i)}(\overrightarrow{u}, \overrightarrow{0})\big)\wedge\\ &\big(T(\overrightarrow{u}) \wedge P(\overrightarrow{u}) \rightarrow S_{(1,q_i)}(\overrightarrow{u}, \overrightarrow{0})\big)\wedge\\ \overrightarrow{u} > \overrightarrow{s} \rightarrow &\big(\neg P(\overrightarrow{u}) \rightarrow S_\sqcup(\overrightarrow{u}, \overrightarrow{0})\big)\wedge\\ &\big(P(\overrightarrow{u}) \rightarrow S_{(\sqcup,q_i)}(\overrightarrow{u}, \overrightarrow{0})\big)\big)\Big)\end{aligned}$$

Furthermore, for every tuple $(\overrightarrow{u}, \overrightarrow{0})$, we want exactly one S_l, $0 \leq l \leq m$, to be true. For example, if for some $\overrightarrow{u}$ we have that $T(\overrightarrow{u}) \wedge P(\overrightarrow{u})$ is true, this should imply that $S_{(1,q_i)}(\overrightarrow{u}, \overrightarrow{0}) \wedge \bigwedge_{j \neq (1,q_i)} \neg S_j(\overrightarrow{u}, \overrightarrow{0})$ is also true. So, we add similar conjunctions to every subformula above, such that whenever $S_l(\overrightarrow{u}, \overrightarrow{0})$ holds, no other S_j, $j \neq l$, holds for the tuple $(\overrightarrow{u}, \overrightarrow{0})$.

Now, we can define formula input as follows.

$$\mathsf{input}(\overrightarrow{S}) := \bigwedge_{0 \leq i \leq d-1} \forall\overrightarrow{v}\big(\overrightarrow{v} = \overrightarrow{s_i} \wedge Q(\overrightarrow{v})\big) \rightarrow \mathsf{state_i}(\overrightarrow{S_i})$$

Formula transition describes all the deterministic transitions that can be made by N until time step $\overrightarrow{t_*}$.

$$\begin{aligned}\mathsf{transition}(\overrightarrow{S}, \overrightarrow{t_*}) := \forall\overrightarrow{s}\forall\overrightarrow{t}(\overrightarrow{t} < \overrightarrow{t_*}) \rightarrow\\ \bigwedge_{(i_1,i_2,i_3) \rightarrow^d (i_4)} S_{i_1}(\overrightarrow{s}-1, \overrightarrow{t}) \wedge S_{i_2}(\overrightarrow{s}, \overrightarrow{t}) \wedge S_{i_3}(\overrightarrow{s}+1, \overrightarrow{t}) \rightarrow S_{i_4}(\overrightarrow{s}, \overrightarrow{t}+1)\wedge\\ \bigwedge_{i \neq i_4} S_{i_1}(\overrightarrow{s}-1, \overrightarrow{t}) \wedge S_{i_2}(\overrightarrow{s}, \overrightarrow{t}) \wedge S_{i_3}(\overrightarrow{s}+1, \overrightarrow{t}) \rightarrow \neg S_i(\overrightarrow{s}, \overrightarrow{t}+1)\end{aligned}$$

where $(i_1, i_2, i_3) \rightarrow^d (i_4)$ means that the configuration corresponding to $\overrightarrow{s}-1$, $\overrightarrow{s}$ and $\overrightarrow{s}+1$ containing i_1, i_2, and i_3, respectively, is mapped deterministically to the configuration that cell $\overrightarrow{s}$ contains i_4. In other words, the conjunction is over all possible transitions that are deterministic. Trivial transitions for cell contents that do not change and a conjunction which states that for every tuple $(\overrightarrow{s}, \overrightarrow{t})$ with $\overrightarrow{t} > \overrightarrow{t_*}$ no S_j is true are also included in the formula transition.

Formula nonfinal guarantees that the computation has not reached its final state before time $\overrightarrow{t_*}$.

$$\mathsf{nonfinal}(\overrightarrow{S}, \overrightarrow{t_*}) := \forall \overrightarrow{t} \big(\overrightarrow{t} < \overrightarrow{t_*} \rightarrow \neg S_{(\sqcup, q_F)}(\overrightarrow{s}_{min}, \overrightarrow{t}) \big).$$

Formula deterministic states that the computation from time $\overrightarrow{0}$ until time $\overrightarrow{t_*}-1$ is deterministic. Equivalently, at all time steps before $\overrightarrow{t_*}$, only choice 0 can be made by N, since there is no possibility of a branching. Let $\gamma_1, ..., \gamma_p \in \mathcal{Q} \times \{0, 1, \sqcup\}$ be exactly the symbol combinations that lead to a nondeterministic choice. Then deterministic can be written as follows:

$$\mathsf{deterministic}(\overrightarrow{S}, \overrightarrow{t_*}) := \forall \overrightarrow{t} \forall \overrightarrow{s} \big(\overrightarrow{t} < \overrightarrow{t_*} \rightarrow \bigwedge_{\gamma \in \{\gamma_1, ..., \gamma_p\}} \neg S_\gamma(\overrightarrow{s}, \overrightarrow{t}) \big).$$

Formula $\mathsf{NonDetChoice}(\overrightarrow{S}, \overrightarrow{t_*})$ expresses that $\overrightarrow{t_*}$ is a time step at which a nondeterministic choice is made by N.

$$\mathsf{NonDetChoice}(\overrightarrow{S}, \overrightarrow{t_*}) := \exists \overrightarrow{s} \bigvee_{\gamma \in \{\gamma_1, ..., \gamma_p\}} S_\gamma(\overrightarrow{s}, \overrightarrow{t_*})$$

Formula $\Delta_i(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*})$, $i \in \{0, 1\}$, defines relations $\overrightarrow{C_i} = (T_i, E_i, P_i, Q_i)$, which encode the configuration of N at time step $\overrightarrow{t_*}+1$, if nondeterministic choice i was made at $\overrightarrow{t_*}$.

For example, if nondeterministic choice 1 was made at $\overrightarrow{t_*}$ and $S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*})$ represents that this choice was made while N was in state q_5, the cursor was on cell $\overrightarrow{s_3}$, and $\overrightarrow{s_3}$ contained the symbol 0, then the following formula encodes the transition that is made.

$$\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge ... \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge ... \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow$$
$$\big(Q_1(\overrightarrow{s_{i_1}}) \wedge E_1(\overrightarrow{s_{i_2}}) \wedge P_1(\overrightarrow{s_{i_3}}) \wedge \bigwedge_{\substack{\overrightarrow{s} \text{ contains } 1 \\ \text{at time } \overrightarrow{t_*}+1}} T_1(\overrightarrow{s})\big)$$

where $\overrightarrow{s_{i_2}}$ is the greatest cell containing either 0 or 1, $\overrightarrow{s_{i_3}} \in \{\overrightarrow{s_2}, \overrightarrow{s_3}, \overrightarrow{s_4}\}$ is the position of the cursor, and q_{i_1} is the state of N at time step $\overrightarrow{t_*}+1$. The first line represents the contents of

the tape at $\overrightarrow{t_*}$. We define formula $\gamma_j^{\Delta_i}(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*})$ to be a conjunction of such formulas so that it describes the transition from any configuration of the machine where $S_{\gamma_j}(\overrightarrow{s}, \overrightarrow{t_*})$ holds for some $\overrightarrow{s}$, to the configuration determined by choice i. The latter configuration is encoded by T_i, E_i, P_i, Q_i. So, for $i \in \{0, 1\}$:

$$\Delta_{\mathsf{i}}(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*}) := \bigwedge_{j \in \{1,\ldots,p\}} \gamma_j^{\Delta_i}(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*})$$

Finally, let $\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) := \mathsf{DetComp}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \mathsf{NonDetChoice}(\overrightarrow{S}, \overrightarrow{t_*})$. Formula tot is defined as follows.

$$\begin{aligned}\mathsf{tot}(\overrightarrow{C}, f) :=& \exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) + \\ & \Sigma \overrightarrow{C_0}.\, \big(\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \big(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_0(\overrightarrow{S}, \overrightarrow{C_0}, \overrightarrow{t_*})\big) \cdot f(\overrightarrow{C_0})\big) + \\ & \Sigma \overrightarrow{C_1}.\, \big(\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \big(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_1(\overrightarrow{S}, \overrightarrow{C_1}, \overrightarrow{t_*})\big) \cdot f(\overrightarrow{C_1})\big).\end{aligned}$$

We have that

$$tot_N(x) = [[\,[\mathrm{pbfp}_f\, \mathsf{tot}](\overrightarrow{C})\,]](\mathcal{A}_x, V_I)$$

where V_I is a second-order assignment for $\mathcal{A}_x$ such that $V_I(T) = T_I$, $V_I(E) = E_I$, $V_I(P) = P_I$, $V_I(Q) = Q_I$, and T_I, E_I, V_I, Q_I are relations which represent the initial configuration of N. We describe how this is possible by giving an FO formula that defines T_I, E_I, V_I, Q_I.

Let B be the relation that encodes the input structure as a binary string of length n. We assume that the cursor is on the first cell when N starts its computation and the input ends at the $(n-1)$-th cell. Let $\mathsf{inputrule_i}$, $0 \leq i \leq 2^n - 1$, be formulas that describe how relation B defines relation T. For example, $\mathsf{inputrule_0}(T) = \forall x \neg B(x) \rightarrow \forall \overrightarrow{s} \neg T(\overrightarrow{s})$. Then,

$$\psi_I := \exists T_I \exists E_I \exists P_I \exists Q_I \big(Q_I(\overrightarrow{s_0}) \wedge E_I(\overrightarrow{s_{n-1}}) \wedge P_I(\overrightarrow{s_0}) \wedge \bigwedge_{0 \leq i \leq 2^n - 1} \mathsf{inputrule_i}(T_I)\big)$$

Note that tot is an $\mathbf{RSO_k\text{-}\Sigma QSO}(\exists \mathbf{SO})$ formula. The operator T_{tot} reaches a fixed point in polynomially many steps; since the machine N runs for $n^c - 1$ steps, the formula $\mathsf{nonfinal}$ will become false after at most $n^c - 1$ recursive steps. So, $f_{n^c}(T_I, E_I, P_I, Q_I) = f_{n^c-1}(T_I, E_I, P_I, Q_I)$. The maximum arity of a second-order variable bounded by the quantifier Σ is $k \geqslant c$. So the interpretation of $[\mathrm{pbfp}_f\, \mathsf{tot}](\overrightarrow{C})$ returns the correct value, which is the number of branchings of N.

A formula for counting the number of computation paths of N.

We can also write a formula, such that its interpretation is equal to the total number of paths of N. Let $\mathsf{FinalState}$ be the formula

$$\mathsf{input}(\overrightarrow{S}) \wedge \mathsf{transition}(\overrightarrow{C}, \overrightarrow{t_*}) \wedge \mathsf{deterministic}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \mathsf{final}(\overrightarrow{S}, \overrightarrow{t_*})$$

which is true if the machine reaches a final state at $\overrightarrow{t_*}$ after a deterministic computation starting from $\overrightarrow{0}$.

$$\begin{aligned}
&\mathsf{tot}_{\mathsf{paths}}(\overrightarrow{C}, f) := \exists\overrightarrow{S}\exists\overrightarrow{t_*}\ \mathsf{FinalState}(\overrightarrow{S}, \overrightarrow{t_*}) + \\
&\Sigma\overrightarrow{C_0}.\left(\exists\overrightarrow{S}\exists\overrightarrow{t_*}\left(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_0(\overrightarrow{S}, \overrightarrow{C_0}, \overrightarrow{t_*})\right) \cdot f(\overrightarrow{C_0})\right) + \\
&\Sigma\overrightarrow{C_1}.[\exists\overrightarrow{S}\exists\overrightarrow{t_*}\left(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_1(\overrightarrow{S}, \overrightarrow{C_1}, \overrightarrow{t_*})\right) \cdot f(\overrightarrow{C_1})].
\end{aligned}$$

In this case it holds that

$$[[\,[\mathrm{pbfp}_f\,\mathsf{tot}_{\mathsf{paths}}](\overrightarrow{C})\,]](\mathcal{A}_x, V_I) = \#(\text{paths of } N \text{ on input } x) = tot_N(x) + 1.$$

5.2.3 A logic that captures TotP

Actually, we do not need the expressive power of $\mathbf{R\Sigma Q_{\exists SO}}$ to write formula tot. First, notice that all second-order formulas in tot can be reduced to propositional formulas the satisfiability of which belongs to P. In more detail, they are reducible to disjunctions of **Horn** formulas. Second, the formula

$$\exists\overrightarrow{S}\exists\overrightarrow{t_*}\ \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*})$$

appears as a subformula in every summand that the function symbol f appears. Thus, we can write tot in the form

$$\begin{aligned}
\mathsf{tot}(\overrightarrow{X}, f) = &\exists\overrightarrow{X}\exists\overrightarrow{x}\,\phi_0(\overrightarrow{X}, \overrightarrow{x}) + \\
&\Sigma\overrightarrow{Y}.\exists\overrightarrow{X}\exists\overrightarrow{x}\left(\phi_0(\overrightarrow{X}, \overrightarrow{x}) \wedge \phi_1(\overrightarrow{X}, \overrightarrow{Y})\right) \cdot f(\overrightarrow{Y}) + \\
&\Sigma\overrightarrow{Y}.\exists\overrightarrow{X}\exists\overrightarrow{x}\left(\phi_0(\overrightarrow{X}, \overrightarrow{x}) \wedge \phi_2(\overrightarrow{X}, \overrightarrow{Y})\right) \cdot f(\overrightarrow{Y})
\end{aligned}$$

Finally, formula Δ_0 (resp. Δ_1) defines unique relations T_0, E_0, P_0, and Q_0 (resp. T_1, E_1, P_1, and Q_1). Although the operators ΣT_0, ΣE_0, ΣP_0, and ΣQ_0 require the evaluation of the succeeding formula

$$\exists\overrightarrow{S}\exists\overrightarrow{t_*}\left(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_0(\overrightarrow{S}, T_0, E_0, P_0, Q_0, \overrightarrow{t_*})\right) \cdot f(T_0, E_0, P_0, Q_0)$$

for every possible assignment for T_0, E_0, P_0, Q_0, the formula will be evaluated to zero for all assignments but one. For the unique assignment defined by Δ_0, the formula will be evaluated to some natural number.

In what follows, first we give some formal definitions based on the observations we just made. Then, we define a subclass of $\mathsf{R\Sigma Q}_{\exists\mathsf{SO}}$ and show that it captures the class TotP.

Definition 5.4. *We say that ϕ is an* **SOE** (**SO-Easy**) *formula if ϕ is an existential second-order formula $\phi = \exists\overrightarrow{X}\psi(\overrightarrow{X})$, where*

- $\overrightarrow{X} = (X_1, ..., X_k)$, $k \in \mathbb{N}$, *is a sequence of second-order variables and*
- $\psi(\overrightarrow{X})$ *is a* $\mathbf{\Sigma_2}$**-Horn** *formula, which means that ψ is of the form*

$$\psi(\overrightarrow{X}) = \exists\overrightarrow{x}\bigvee_i \forall\overrightarrow{y}\,\chi_i(\overrightarrow{X}, \overrightarrow{x}, \overrightarrow{y})$$

where each χ_i is an unquantified first-order formula in CNF form, in which each clause may contain negative occurrences and at most one positive occurence of a relation X_j, $1 \leq j \leq k$.

For an **SOE** formula, **E** stands for easy, since its satisfiability is reducible to the satisfiability of a propositional formula, which can be solved in deterministic polynomial time. In fact, there is a product reduction between the counting versions of these problems.

Let #DisjHornSat denote the problem of counting satisfying assignments of a disjuction of propositional **Horn** formulas.

Proposition 5.7. *Given a structure $\mathcal{A}$ and an* **SOE** *formula $\phi = \exists\overrightarrow{X}\psi(\overrightarrow{X})$, the problem of computing $|\{\langle\overrightarrow{X}\rangle : \mathcal{A} \models \psi(\overrightarrow{X})\}|$ can be reduced to #DisjHornSat under product reductions.*

Proof. Formula

$$\psi(\overrightarrow{X}) = \exists x_1 ... \exists x_n \bigvee_{1\leq i\leq t} \forall y_1 ... \forall y_m\, \chi_i(\overrightarrow{X}, x_1, ..., x_n, y_1, ..., y_m)$$

can be written as $\bigvee_{w\in A^n} \bigvee_{1\leq i\leq t} \bigwedge_{u\in A^m} \chi_i(\overrightarrow{X}, w, u)$. Each clause in any χ_i is a disjunction of some (positive or negative) X_j and atomic formulas. The latter are either true or false in $\mathcal{A}$. If an atomic formula is not satisfied in $\mathcal{A}$, then it is removed from its clause, whereas if it is satisfied in $\mathcal{A}$, its clause is removed from the formula.

By the previous transformations, we obtained a formula which is a disjunction of **CNF** formulas, each clause of which has some literals of the form $\neg X_j(\overrightarrow{x_l})$ and at most one literal of the form $X_j(\overrightarrow{x_l})$, where $1 \leq j \leq k$ and $\overrightarrow{x_l} \in A^{\mathsf{arity}(X_j)}$.

The last transformation consists of replacing every $X_j(\overrightarrow{x_l})$ by a propositional variable z_{jl}. Let $g(\mathcal{A}) = \bigvee\bigwedge\bigvee z_{jl}$ be the resulting formula. Now there may be some combinations of the

second-order variables $\overrightarrow{X}$ and tuples of elements of A, that do not appear in $g(\mathcal{A})$. Let $n(\mathcal{A})$ be the number of these combinations. Then,

$$|\{\langle\overrightarrow{X}\rangle : \mathcal{A} \models \psi(\overrightarrow{X})\}| = \#\textsc{DisjHornSat}(g(\mathcal{A})) \cdot 2^{n(\mathcal{A})}.$$

□

Definition 5.5. *An* **SO** *formula $\phi(R)$ with free second-order variable R of arity k, defines R uniquely if the following condition holds.*

$$\begin{gathered}\textit{For any structure } \mathcal{A}, \textit{if } \mathcal{A} \models \phi(R/S) \textit{ and } \mathcal{A} \models \phi(R/S'), \textit{ then} \\ \textit{for every } (x_1, ..., x_k) \in A^k, \textit{ it holds that } S(x_1, ..., x_k) \textit{ iff } S'(x_1, ..., x_k).\end{gathered} \tag{5.4}$$

Remark 5.2. *Consider an* **SOE** *formula $\phi(R)$ that defines a relation R uniquely and let $\mathcal{A}$ be a structure. Then by reducing ϕ to a disjunction of* **Horn** *formulas, we can find the unique relation S such that $\mathcal{A} \models \phi(R/S)$.*

Definition 5.6. *We say that a formula α is a* $\mathbf{\Sigma QSO}(\mathbf{SOE})$ *formula over a vocabulary σ if it is given by the following grammar*

$$\alpha := \phi \mid s \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma X.\psi(X) \circ \alpha(X) \tag{5.5}$$

where ϕ is restricted to be an **SOE** *formula, $\psi(X)$ is an* **SOE** *formula that defines X uniquely and $\circ \in \{\wedge, \cdot\}$.*

The set of $\mathbf{RSO_k\text{-}\Sigma QSO}(\mathbf{SOE})$ *formulas over σ is the set of* $\mathbf{\Sigma QSO}(\mathbf{SOE})$ *formulas together with formula* $[\mathrm{pbfp}_f\, \beta](\overrightarrow{X})$*, where β is of the following form:*

$$\begin{aligned}&\exists\overrightarrow{Z}\,\phi_0(\overrightarrow{Z}) + \\ &\Sigma\overrightarrow{Y}.\exists\overrightarrow{Z}\left(\phi_0(\overrightarrow{Z}) \wedge \phi_1(\overrightarrow{Z}, \overrightarrow{Y})\right) \cdot f(\overrightarrow{Y}) + \\ &\qquad\qquad ... \qquad\qquad + \\ &\Sigma\overrightarrow{Y}.\exists\overrightarrow{Z}\left(\phi_0(\overrightarrow{Z}) \wedge \phi_r(\overrightarrow{Z}, \overrightarrow{Y})\right) \cdot f(\overrightarrow{Y})\end{aligned} \tag{5.6}$$

where $\overrightarrow{X} = (X_1, ..., X_n)$, $\overrightarrow{Y} = (Y_1, ..., Y_n)$, with $\mathsf{arity}(X_j) = \mathsf{arity}(Y_j) = k$, $k \in \mathbb{N}$, for every $j \in \{1, ..., n\}$, $\overrightarrow{Z} = (Z_1, ..., Z_m)$ with $arity(Z_i) = k_i$, $i \in \{1, ..., m\}$, and $\exists\overrightarrow{Z}\left(\phi_0(\overrightarrow{Z}) \wedge \phi_i(\overrightarrow{Z}, \overrightarrow{Y})\right)$, $i \in \{1, ..., r\}$, defines relations $\overrightarrow{Y}$ uniquely.

Definition 5.7. *We say that $f \in \mathsf{RSO_k\text{-}\Sigma QSO(SOE)}$ if there exists an* $\mathbf{RSO_k\text{-}\Sigma QSO}(\mathbf{SOE})$ *formula α such that $[[\alpha]](\mathcal{A}) = f(\mathcal{A})$, for every finite ordered structure $\mathcal{A}$ over σ.*

Analogously to $\mathsf{R\Sigma Q_{\exists SO}}$, we denote the union of $\mathsf{RSO_k\text{-}\Sigma QSO(SOE)}$, for every $k \in \mathbb{N}$, by $\mathsf{R\Sigma Q_{SOE}}$.

Definition 5.8. $\mathsf{R\Sigma Q_{SOE}} = \bigcup_k \mathsf{RSO_k\text{-}\Sigma QSO(SOE)}$.

Now we prove the main theorems of this work.

Theorem 5.7. *Every function in* TotP *belongs to* $\mathsf{R\Sigma Q_{SOE}}$.

Proof. We prove that formula tot is a formula in $\mathsf{R\Sigma Q_{SOE}}$.

- Formula $\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*})$ can be written in the equivalent following form.

$$\exists \overrightarrow{S} \exists \overrightarrow{t_*} \mathsf{branch}(\overrightarrow{S}, \overrightarrow{t_*}) := \exists \overrightarrow{S} \exists \overrightarrow{t_*} \exists \overrightarrow{s_*} \bigvee_{\gamma \in \{\gamma_1,\ldots,\gamma_p\}} \mathsf{input} \wedge \mathsf{transition} \wedge \mathsf{nonfinal} \wedge \mathsf{deterministic} \wedge S_\gamma(\overrightarrow{s_*}, \overrightarrow{t_*}).$$

  This formula is an **SOE** formula, since the first-order formula

$$\exists \overrightarrow{t_*} \exists \overrightarrow{s_*} \bigvee_{\gamma \in \{\gamma_1,\ldots,\gamma_p\}} \mathsf{input} \wedge \mathsf{transition} \wedge \mathsf{nonfinal} \wedge \mathsf{deterministic} \wedge S_\gamma(\overrightarrow{s_*}, \overrightarrow{t_*})$$

  is a $\mathbf{\Sigma_2}$**-Horn** formula.

- Formula Δ_i, $i \in \{0,1\}$, can be easily written as an equivalent $\mathbf{\Sigma_2}$**-Horn** formula. For example, subformula

$$\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow \big(Q_1(\overrightarrow{s_{i_1}}) \wedge E_1(\overrightarrow{s_{i_2}}) \wedge P_1(\overrightarrow{s_{i_3}}) \wedge \bigwedge_{\substack{\overrightarrow{s} \text{ contains } 1 \\ \text{at time } \overrightarrow{t_*}+1}} T_1(\overrightarrow{s})\big)$$

  can be replaced by

$$\begin{aligned} &\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow Q_1(\overrightarrow{s_{i_1}}) \wedge \\ &\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow E_1(\overrightarrow{s_{i_2}}) \wedge \\ &\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow P_1(\overrightarrow{s_{i_3}}) \wedge \\ \bigwedge_{\substack{\overrightarrow{s} \text{ contains } 1 \\ \text{at time } \overrightarrow{t_*}+1}} &\big(S_{i_1}(\overrightarrow{s_0}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{(0,q_5)}(\overrightarrow{s_3}, \overrightarrow{t_*}) \wedge \ldots \wedge S_{i_2}(\overrightarrow{s_{max}}, \overrightarrow{t_*})\big) \rightarrow T_1(\overrightarrow{s}). \end{aligned}$$

  Recall that in fact we have $E_1(\overrightarrow{s_{i_2}}) \wedge \bigwedge_{i \neq i_2} \neg E_1(\overrightarrow{s_i})$ instead of $E_1(\overrightarrow{s_{i_2}})$, which also gives the right form here. The same holds for relations P_1 and Q_1.

- Formula $\exists \overrightarrow{S} \exists \overrightarrow{t_*} \big(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_i(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*})\big)$, $i \in \{0,1\}$, satisfies condition (5.4) for relations T_i, E_i, P_i, Q_i. Note that, given a structure $\mathcal{A}$, there are unique relations $S_1, \ldots, S_m$, such that encode the computation of the machine N on input $\mathcal{A}$. Furthermore, formula Δ_i gives unique relations T_i, E_i, P_i, Q_i which depend on $S_1, \ldots, S_m$.

- It is immediate from the previous facts that tot is of form (5.6) with only two summands.

□

It remains to prove that every function in $\mathsf{R\Sigma Q_{SOE}}$ is a TotP function.

Theorem 5.8. *Every function in* $\mathsf{R\Sigma Q_{SOE}}$ *belongs to* TotP.

Proof. We need to prove that for any formula $\alpha \in \mathbf{R\Sigma Q_{SOE}}$ there is an NPTM M_α with a second-order assignment V stored in memory, such that for every input structure $\mathcal{A}$, $[[\alpha]](\mathcal{A}, V) = tot_{M_\alpha}(\mathcal{A})$. Equivalently,

$$\#(\text{paths of } M_\alpha \text{ on input } \mathcal{A}) = [[\alpha]](\mathcal{A}, V) + 1.$$

We prove this by induction on the structure of α.

- If $\alpha = \phi$, then ϕ is an **SOE** formula, so it can be verified whether $(\mathcal{A}, V) \models \phi$ in deterministic polynomial time. If $(\mathcal{A}, V) \models \phi$ is true, then M_α generates two paths and halts. Otherwise, it just halts.

- If $\alpha = s$, then M_α generates $s + 1$ paths and halts.

- If either $\alpha = \alpha_1 + \alpha_2$, or $\alpha = \alpha_1 \cdot \alpha_2$, then by induction hypothesis, there are M_{α_i}, $i = 1, 2$, such that $\#(\text{paths of } M_{\alpha_i} \text{ on input } \mathcal{A}) = [[\alpha_i]](\mathcal{A}, V) + 1$. There are also NPTMs M'_{α_i} such that $\#(\text{paths of } M'_{\alpha_i} \text{ on input } \mathcal{A}) = [[\alpha_i]](\mathcal{A}, V)$ as described in the proof of Proposition 1.8.

  - If $\alpha = \alpha_1 + \alpha_2$, then M_α simulates M'_{α_1} and M'_{α_2} nondeterministically and generates an additional (dummy) computation path.
  - If $\alpha = \alpha_1 \cdot \alpha_2$, then M_α simulates M'_{α_1} and M'_{α_2} sequentially and generates an additional (dummy) computation path.

  So, in both cases, $\#(\text{paths of } M_\alpha \text{ on input } \mathcal{A}) = [[\alpha]](\mathcal{A}, V) + 1$.

- If $\alpha = \Sigma Y.\beta(Y) = \Sigma Y.\psi(Y) \circ \beta'(Y)$, then M_α can determine the unique relation Y, which is defined by ψ, in deterministic polynomial time and then evaluate β' in polynomial time, by induction hypothesis.

- If $\alpha = [\text{pbfp}_f\, \beta](\overrightarrow{X})$, then β has form (5.6) and M_α can evaluate $[[\alpha]](\mathcal{A}, V)$ as follows. It starts a recursive computation on input β where $\overrightarrow{X} = (X_1, ..., X_n)$ are replaced by $V(X_1), ..., V(X_n)$. First, it checks whether $\mathcal{A} \models \exists \overrightarrow{Z}\, \phi_0(\overrightarrow{Z})$ is true.

1. If $\mathcal{A} \models \exists \overrightarrow{Z} \phi_0(\overrightarrow{Z})$ is false, then M_α halts.
2. Otherwise, for every summand determines the unique relations $\overrightarrow{Y_i}$ which are defined by $\exists \overrightarrow{Z} \big(\phi_0(\overrightarrow{Z}) \wedge \phi_i(\overrightarrow{Z}, \overrightarrow{Y})\big)$. For every $\overrightarrow{Y_i}$, it checks whether $f(\overrightarrow{Y_i})$ is not zero. This can be done by checking $\mathcal{A} \models \exists \overrightarrow{Z} \phi_0(\overrightarrow{Z})$, where $\overrightarrow{X}$ are replaced by $\overrightarrow{Y_i}$. For every $\overrightarrow{Y_i}$ with $f(\overrightarrow{Y_i}) \neq 0$, M_α generates a different path. On each of the generating paths, the computation continues recursively on input $\beta(X_1/Y_{i_1}, ..., X_n/Y_{i_n}, f)$.
3. If M_α is at the first step of the recursion and there is at least a sequence $\overrightarrow{Y_i}$ such that $f(\overrightarrow{Y_i}) \neq 0$, then it also generates a dummy path, in which it does nothing but halts.

By definition of the p-bounded fixed point of the operator T_β, if the chain $\{f_j\}_{j \in \mathbb{N}}$ stabilizes for some $j \leq |A|^k$, where k is the arity of each X_i, it means that after k recursive steps, M_α evaluates formulas to zero on every path and halts. In the case of the chain not reaching a fixed point, then M_α stops the recursion after $|A|^k$ steps. In both cases, the number of paths of M_α is equal to $[[\,[\mathrm{pbfp}_f\, \beta](\overrightarrow{X})\,]](\mathcal{A}, V) + 1$. □

Corollary 5.3. $\mathsf{TotP} = \mathsf{R\Sigma Q_{SOE}}$ *over finite ordered structures.*

5.2.4 An alternative way to define $\mathsf{R\Sigma Q_{SOE}}$ and capture TotP

Although the logic just defined captures TotP, it seems that natural TotP problems cannot be expressed using this language in an easy and natural way. That's why we provide an alternative (and very similar) definition of $\mathbf{R\Sigma Q_{SOE}}$, that will help us to express #DNF.

Definition 5.9. *We say that a formula α is a* $\mathbf{\Sigma QSO(SOE)}$ *formula over a vocabulary σ if it is given by the following grammar*

$$\alpha := \phi \mid s \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma X.\psi(X) \circ \alpha(X) \tag{5.7}$$

where ϕ is restricted to be an **SOE** *formula, $\psi(X)$ is an* **SOE** *formula that defines X uniquely and $\circ \in \{\wedge, \cdot\}$.*

The set of $\mathbf{RSO_k\text{-}\Sigma QSO(SOE)}$ *formulas over σ is the set of* $\mathbf{\Sigma QSO(SOE)}$ *formulas together with formula $s + [\mathrm{pbfp}_f\, \beta](\overrightarrow{X})$, where $s \in \mathbb{N}$ and β is of the following form:*

$$\begin{aligned} &\exists \overrightarrow{Z} \big(\phi_{01}(\overrightarrow{Z}) \wedge \phi_{02}(\overrightarrow{Z})\big) + \\ &\Sigma \overrightarrow{Y}. \exists \overrightarrow{Z} \big(\phi_{01}(\overrightarrow{Z}) \wedge \phi_1(\overrightarrow{Z}, \overrightarrow{Y})\big) \cdot f(\overrightarrow{Y}) + \\ &\Sigma \overrightarrow{Y}. \exists \overrightarrow{Z} \big(\phi_{02}(\overrightarrow{Z}) \wedge \phi_2(\overrightarrow{Z}, \overrightarrow{Y})\big) \cdot f(\overrightarrow{Y}) \end{aligned} \tag{5.8}$$

where $\overrightarrow{X} = (X_1, ..., X_n)$, $\overrightarrow{Y} = (Y_1, ..., Y_n)$, *with* $\mathsf{arity}(X_j) = \mathsf{arity}(Y_j) = k$, $k \in \mathbb{N}$, *for every* $j \in \{1, ..., n\}$, $\overrightarrow{Z} = (Z_1, ..., Z_m)$ *with* $\mathsf{arity}(Z_i) = k_i$, $i \in \{1, ..., m\}$, $\exists \overrightarrow{Z} \big(\phi_{0i}(\overrightarrow{Z}) \wedge \phi_i(\overrightarrow{Z}, \overrightarrow{Y})\big)$, $i \in \{1, 2\}$, *defines relations* $\overrightarrow{Y}$ *uniquely.*

Definition 5.10. *We say that* $f \in \mathsf{RSO_k\text{-}\Sigma QSO(SOE)}$ *if there exists an* $\mathbf{RSO_k\text{-}\Sigma QSO(SOE)}$ *formula* α *such that* $[[\alpha]](\mathcal{A}) = f(\mathcal{A})$, *for every finite ordered structure* $\mathcal{A}$ *over* σ.

Definition 5.11. $\mathsf{R\Sigma Q_{SOE}} = \bigcup_k \mathsf{RSO_k\text{-}\Sigma QSO(SOE)}$

In the following example, given a **DNF** formula ϕ, a binary tree is constructed such that the number of satisfying assignments of ϕ is equal to

- the number of its paths,
- or equivalently, the number of its branchings minus 1.

Example 5.2. *Let* ϕ *be an input* **DNF** *formula and* $x_1, ..., x_n$ *be an enumeration list of its variables. Consider an NPTM* M *that makes the following recursive computation on input* ϕ. M *determines in deterministic polynomial time whether there is a satisfying assignment for* ϕ. *If the answer is no or the enumeration list is empty, it halts. Otherwise,* M *picks the first variable appearing in the list, namely* x_1, *and checks whether there is a satisfying assignment for* ϕ_0, *i.e.* ϕ *with* x_1 *assigned to false, and whether there is one for* ϕ_1, *which is obtained from* ϕ *by making* x_1 *true.* M *removes* x_1 *from the list and then*

- *if the answer is yes for both cases,* M *chooses nondeterministically to make* x_1 *either false or true and proceeds recursively with* ϕ_0 *and* ϕ_1, *respectively.*
- *if the answer is yes for only one case,* M *deterministically proceeds recursively with the corresponding formula, i.e. either* ϕ_0 *or* ϕ_1.

Since M *removes at least one variable from the list at each step, the depth of the recursion is polynomial in the size of* ϕ.

The computation of M *on input a* **DNF** *formula with three variables is depicted in Figure 5.3.*

Example 5.3. *The problem* #DNF *belongs to* $\mathsf{R\Sigma Q_{SOE}}$ *since it holds that* $\#\mathrm{DNF}(\phi) = [[1 + [\mathrm{pbfp}_f\, \mathsf{dnf_2}](True, Fasle)\,]](\mathcal{A}_\phi, V)$, *where* $\mathsf{dnf_2}$ *is given below.*

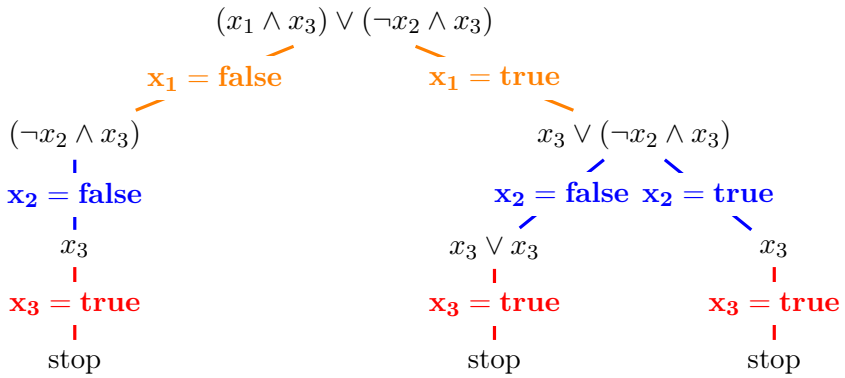


Figure 5.3: NPTM M for which it holds that $tot_M(x) = \#\text{DNF}(\phi)$, where x is a binary encoding of $\phi = (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$.

$$\mathsf{dnf}_2(True, False, f) :=$$

$$\exists T \exists v_{min}\big(\mathsf{min}(v_{min}) \wedge T(v_{min}) \wedge \mathsf{sat}(T)\big) \wedge \big(\exists T \exists v_{min}(\mathsf{min}(v_{min}) \wedge \neg T(v_{min}) \wedge \mathsf{sat}(T)\big) +$$

$$\Sigma T_1.\Sigma F_1.\Big(\big(\exists T \exists v_{min}(\mathsf{min}(v_{min}) \wedge T(v_{min}) \wedge \mathsf{sat}(T)) \wedge \mathsf{t}_1(T_1) \wedge \mathsf{f}_1(F_1)\big) \cdot f(T_1, F_1)\Big) +$$

$$\Sigma T_0.\Sigma F_0.\Big(\big(\exists T \exists v_{min}(\mathsf{min}(v_{min}) \wedge \neg T(v_{min}) \wedge \mathsf{sat}(T)) \wedge \mathsf{t}_0(T_0) \wedge \mathsf{f}_0(F_0)\big) \cdot f(T_0, F_0)\Big).$$

The second-order assignment V and subformulas appearing in dnf_2 *are defined in Example 5.1.*

Note that $[[\,[\text{pbfp}_f\, \mathsf{dnf}_2](True, Fasle)\,]](\mathcal{A}_\phi, V)$ *is equal to the number of branchings that exist in an NPTM constructed as in Example 5.2.*

Theorem 5.9. *Every function in* TotP *belongs to* $\mathsf{R\Sigma Q_{SOE}}$.

Proof. The proof is the same as the proof of Theorem 5.7 where only the last case has changed. Formula tot is of form $s + [\text{pbfp}_f\, \beta](\overrightarrow{X})$, where $s = 0$ and β is of form (5.8) with $\exists \overrightarrow{Z} \phi_{01}(\overrightarrow{Z}) = \exists \overrightarrow{Z} \phi_{02}(\overrightarrow{Z})$ to be the formula $\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*})$. □

Theorem 5.10. *Every function in* $\mathsf{R\Sigma Q_{SOE}}$ *belongs to* TotP.

Proof. We need to prove that for any formula $\alpha \in \mathbf{R\Sigma Q_{SOE}}$ there is an NPTM M_α with a second-order assignment V stored in memory, such that for every input structure $\mathcal{A}$, $[[\alpha]](\mathcal{A}, V) = tot_{M_\alpha}(\mathcal{A})$. Equivalently,

$$\#(\text{paths of } M_\alpha \text{ on input } \mathcal{A}) = [[\alpha]](\mathcal{A}, V) + 1.$$

We prove this by induction on the structure of α.

- If $\alpha = \phi$, then ϕ is an **SOE** formula, so it can be verified whether $(\mathcal{A}, V) \models \phi$ in deterministic polynomial time. If $(\mathcal{A}, V) \models \phi$ is true, then M_α generates two paths and halts. Otherwise, it just halts.

- If $\alpha = s$, then M_α generates $s+1$ paths and halts.
- If either $\alpha = \alpha_1 + \alpha_2$, or $\alpha = \alpha_1 \cdot \alpha_2$, then by induction hypothesis, there are M_{α_i}, $i = 1, 2$, such that $\#(\text{paths of } M_{\alpha_i} \text{ on input } \mathcal{A}) = [[\alpha_i]](\mathcal{A}, V) + 1$. There are also NPTMs M'_{α_i} such that $\#(\text{paths of } M'_{\alpha_i} \text{ on input } \mathcal{A}) = [[\alpha_i]](\mathcal{A}, V)$ as described in the proof of Proposition 1.8.
  - If $\alpha = \alpha_1 + \alpha_2$, then M_α simulates M'_{α_1} and M'_{α_2} nondeterministically and generates an additional (dummy) computation path.
  - If $\alpha = \alpha_1 \cdot \alpha_2$, then M_α simulates M'_{α_1} and M'_{α_2} sequentially and generates an additional (dummy) computation path.

  So, in both cases, $\#(\text{paths of } M_\alpha \text{ on input } \mathcal{A}) = [[\alpha]](\mathcal{A}, V) + 1$.
- If $\alpha = \Sigma Y.\beta(Y) = \Sigma Y.\psi(Y) \circ \beta'(Y)$, then M_α can determine the unique relation Y which is defined by ψ in deterministic polynomial time and then evaluate β' in polynomial time by inductive hypothesis.
- If $\alpha = s + [\text{pbfp}_f\, \beta](\overrightarrow{X})$, then β has form (5.8). M_α evaluates $[[\alpha]](\mathcal{A}, V)$ as follows. It generates $s+1$ different paths. On each of the first s paths, it halts. On the last path, it starts a recursive computation on input β where $\overrightarrow{X} = (X_1, ..., X_n)$ are replaced by $V(X_1), ..., V(X_n)$. First, it checks whether either $\mathcal{A} \models \exists \overrightarrow{Z} \phi_{01}(\overrightarrow{Z})$ or $\mathcal{A} \models \exists \overrightarrow{Z} \phi_{02}(\overrightarrow{Z})$ holds.
  1. If both do not hold, then it halts.
  2. If both hold, M_α generates two different paths, one for each ϕ_{0i}. On the path that corresponds to ϕ_{01}, $i \in \{1, 2\}$, it determines the unique relations $\overrightarrow{Y_i}$ which are defined by $\exists \overrightarrow{Z} \big(\phi_{0i}(\overrightarrow{Z}) \wedge \phi_i(\overrightarrow{Z}, \overrightarrow{Y})\big)$ and it proceeds recursively on input β where $\overrightarrow{X}$ are replaced by $\overrightarrow{Y_i}$.
  3. If exactly one of them holds, w.l.o.g let's assume that only $\mathcal{A} \models \exists \overrightarrow{Z} \phi_{01}(\overrightarrow{Z})$ is true, then M_α determines the unique relations $\overrightarrow{Y_1}$ which are defined by $\exists \overrightarrow{Z} \big(\phi_{01}(\overrightarrow{Z}) \wedge \phi_1(\overrightarrow{Z}, \overrightarrow{Y})\big)$ and it proceeds recursively on input β where $\overrightarrow{X}$ are replaced by $\overrightarrow{Y_1}$.

  By definition of the p-bounded fixed point of the operator T_β, if the chain $\{f_j\}_{j \in \mathbb{N}}$ stabilizes for some $j \leq |A|^k$, where k is the arity of each X_i, it means that after k recursive steps, M_α evaluates formulas to zero on every path and halts. In the case of the chain not reaching a fixed point, then M_α stops the recursion after $|A|^k$ steps.

In both cases, the number of binary branchings of M_α on the last of its initial $s+1$ paths is equal to $[[\,[\text{pbfp}_f\,\beta](\overrightarrow{X})\,]](\mathcal{A},V)$. The number of all paths of M_α is equal to $s+[[\,[\text{pbfp}_f\,\beta](\overrightarrow{X})\,]](\mathcal{A},V)+1$. □

5.3 Discussion of results

Building upon previous work, we gave logical characterizations of two subclasses of TotP, namely ΣQSO(Σ$_2$-2SAT) and #Π$_2$-1VAR, that have natural complete problems. Both these classes are not subclasses of FPRAS unless RP = NP, since their complete problems #Disj2Sat and #MonSat, respectively, are AP-interreducible with #Sat. Regarding the latter, which is complete for #Π$_2$-1VAR under product reductions, we provided Proposition 5.6, which is about the closure of #MonSat, #Π$_2$-1VAR, and TotP under product reductions and their relationship.

Most significantly, we answered an open question in the area of descriptive complexity of counting classes, about the logical characterization of TotP: RΣQ$_{\text{SOE}}$ = TotP over finite ordered structures.

For an arbitrary TotP problem and its related binary NPTM M, computing the number of branchings of M can be expressed in the logic $\mathbf{R\Sigma Q_{SOE}}$. Conversely, given a formula α in $\mathbf{R\Sigma Q_{SOE}}$, there is an NPTM such that the number of its paths (minus one) is the value of the interpretation of α. We achieved our goal by starting with a very rich logic, namely $\mathbf{\Sigma QSO}(\exists\mathbf{SO})$, and implementing the following steps.

1. We expressed the existence of a branching in the computation of an NPTM. This requires an existential second-order formula which is **SOE** (**SOEasy**), i.e. it can be determined whether it is satisfied in the input structure in polynomial time.

2. We added recursion to our logic, so we can add 1 whenever we find a branching and continue recursively. Since the Turing machine is of polynomial length, we need recursion of polynomial depth. Therefore, we introduced a *polynomially* bounded fixed point (p-bounded fixed point).

3. We ensured that any sum over a second-order variable has to be evaluated on a single interpretation of the second-order variable. Also this interpretation can also be determined efficiently.

The recursion and fixed point introduced in this work could give logical characterizations

of superclasses of #P, or even #P. They can also be restricted; for example, specific operators, such as the path operator, using free second-order variables could be defined and added to the syntax of some logic (as Arenas et al. suggested [15]). We believe that this is an interesting and meaningful line of future work.

In Section 5.2.4, we provided an equivalent, slightly different logic, that captures TotP and allows us to express TotP-problems in a more natural and easy way.

5.4 Notes

The two classes ΣQSO(Σ_2-2SAT) and #Π_2-1VAR studied in Section 5.1 were first defined in [24].

Chapter 6

Counting matchings in graphs with edge colors

In this chapter our aim is to examine counting problems the decision version of which is in RP. We are interested in the following two problems defined in Chapter 3. In specific, we would like to conclude whether these problems have an fpras or are hard to approximate.

#Exact Matchings (Definition 3.7 restated).
Input: A graph $G = (V, E)$, a subset $E' \subseteq E$, and an integer k.
Output: The number of perfect matchings of G that contain exactly k edges in E'.

#Blue-Red Matchings (Definition 3.8 restated).
Input: A graph $G = (V, E_{red} \cup E_{blue})$ and two integers w and B.
Output: The number of matchings of size at least B with at most w edges in E_{blue} (blue edges) and at most w edges in E_{red} (red edges).

The corresponding decision versions Exact Matching [125] and Blue-Red Matching [120] are known to belong to the class RNC [119, 120] and it is not known whether they can be solved in deterministic polynomial time.

#Blue-Red Matchings is at least as hard as #Exact Matchings, which, in turn, is at least as hard as #PerfMatch. To start with, we are going to examine #Exact Matchings since it seems to be 'easier' than #Blue-Red Matchings. In general graphs, it is a long-standing open question whether #PerfMatch, a special case of #Exact Matchings, has an fpras. So, it is reasonable that we focus on restricted classes of graphs, such as planar, $K_{3,3}$-free, K_5-free, and bipartite graphs.

Next section discusses related work on problems of counting matchings.

6.1 Related work on counting matchings

The problem #PerfMatch of counting perfect matchings in a general graph appeared along with the definition of the complexity class #P [149] and was among the first problems to be proven #P-complete under Turing reductions. In fact, its special case #BiPerfMatch, which is equivalent to the problem of computing the permanent of an $(n \times n)$ matrix with entries in $\{0, 1\}$, is #P-complete under Turing reductions [149].

Definition 6.1. #BiPerfMatch.

Input: A bipartite graph $G = (U \cup V, E)$.

Output: The number of perfect matchings of G.

Definition 6.2. Permanent.

Input: An $(n \times n)$ matrix A with entries $a_{ij} \in \{0, 1\}$.

Output: The permanemt of matrix A, that is

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Given a bipartite graph $G = (U \cup V, E)$, the biadjacency matrix $A = (a_{ij})$ of G is the $|U| \times |V|$ matrix such that $a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$. If $|U| = |V|$, then A is a square matrix and $\text{perm}(A) = \#\text{BiPerfMatch}(G)$.

Chien [48] reduced the problem of counting perfect matchings in general graphs to computing the determinant of random matrices. This determinant-based algorithm generalizes the Godsil-Gutman estimator [76] and has an exponential running time in the worst case. Fürer and Kasiviswanathan [72] proposed a simpler randomized algorithm together with some of its variants.

An fpras for the problem #BiPerfMatch was presented by Jerrum, Sinclair, and Vigoda [93] using a Markov chain Monte Carlo (MCMC) approach. This algorithm is also an fpras for computing the permanent of an arbitrary matrix with non-negative entries. However, this Markov chain is not rapidly mixing in general graphs [139].

In planar graphs—which are also the graphs that exclude K_5 and $K_{3,3}$ as minors—counting perfect matchings can be solved in deterministic polynomial time by the FKT algorithm [100, 142]. This problem is even in NC [152]. Membership in NC is also true for the same problem defined on $K_{3,3}$-free graphs [152]. Counting perfect matchings in K_5-free graphs is in TC [141].

Counting problem	Complexity	Exact algorithm
#PerfMatch in planar graphs	FP	FKT and NC algorithm
#PerfMatch in $K_{3,3}$-free graphs	FP	NC algorithm
#PerfMatch in K_5-free graphs	FP	TC algorithm
#PerfMatch in K_8-free graphs	#P-complete	—

Table 6.1: The complexity of #PerfMatch in some minor-free graphs.

However, it was shown in [55] that counting perfect matchings in K_8-free graphs is #P-complete. The complexity of the four aforementioned problems is summarized in Table 6.1. Thilikos and Wiederrecht recently proved in [143] a sharp complexity dichotomy for the problem of counting perfect matchings in minor-closed graph classes. They provided a polynomial-time decidable criterion to classify the problem on any graph G which excludes a finite set $\mathcal{F}$ of graphs as minors, as either polynomial-time computable or #P-complete.

On the contrary, the problem #AllMatchings of counting the matchings of all sizes is #P-hard under Turing reductions even when restricted on planar graphs [90]. Jerrum and Sinclair have given an fpras for this problem defined on general graphs [92].

Counting matchings of size k in planar graphs is #P-complete under Turing reductions [90]. Counting matchings of size k in bipartite graphs has an fpras [70], since there is an easy reduction from counting k-matchings to counting perfect matchings. This reduction works for general and bipartite graphs, but not for planar graphs. Recently Anari et al. proved that counting k-matchings in planar graphs admits an fpras [6]. They also gave an fpras for two versions of weighted counting (for all weights) in planar graphs: (i) counting weighted matchings of size k and (ii) counting weighted matchings of any size. The problem of counting weighted matchings is defined below.

Definition 6.3. #Weighted Matchings.
Input: A graph $G = (V, E)$ and a function $w : E \to \mathbb{R}$.
Output: Compute the sum

$$\#\text{Weighted Matchings}(G) = \sum_{M \in \mathcal{M}(G)} \prod_{e \in M} w(e)$$

where $\mathcal{M}(G)$ is the set of matchings of G.

Remark 6.1. *This problem can be defined such that we count weighted k-matchings or weighted*

perfect matchings by just considering $\mathcal{M}(G)$ to be the set of k-matchings or perfect matchings, respectively.

Results for the aforementioned variants of counting matchings defined on either general, bipartite or planar graphs are gathered in Table 6.2.

Counting problem	**Complexity**	**Approximation algorithm**
#PerfMatch	#P-complete	a ras achieving $(1 \pm \varepsilon)$-approximation with $\mathcal{O}(\varepsilon^{-2} \cdot 3^{n/2})$ trials
#PerfMatch in bipartite graphs	#P-complete	fpras
#AllMatchings	#P-complete	fpras
#AllMatchings in bipartite graphs	#P-complete	fpras
#AllMatchings in planar graphs	#P-complete	fpras
#k-Matchings	#P-complete	—
#k-Matchings in bipartite graphs	#P-complete	fpras
#k-Matchings in planar graphs	#P-complete	fpras

Table 6.2: The complexity of counting perfect matchings, all matchings, and k-matchings in general, bipartite, and planar graphs.

Parameterized complexity

Counting k-matchings on unweighted graphs without multiple edges or self-loops is #W[1]-hard [51]. The best known algorithms for counting k-matchings exhibit time bounds of the type $f(k)n^{\Theta(k)}$. Among them is the algorithm of [156] with a runtime of $\mathcal{O}(2^{k+o(k)}\binom{n}{k/2})$. Arvind and Raman [18] gave a randomized algorithm with running time $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$, approximation ratio $1/k^{\mathcal{O}(k)}$, and error probability $2^{-n^{\mathcal{O}(1)}}$ for approximately counting the number of matchings of size k in a graph with n vertices.

In [68], it was conjectured that counting k-matchings in bipartite graphs is #W[1]-hard in

the parameter k. The conjecture was proved by Curticapean [51]. He also proved that counting k-matchings in planar graphs is #W[1]-hard [53]. For some bipartite graphs (with $f(k)$ vertices in one partition, for some function f) the problem of counting k matchings is fixed-parameter tractable [156].

Parameterized complexity of the problem #k-Matchings is also included in Table 6.4.

6.2 The problems Exact Matching and Blue-Red Matching

The problem Exact Matching is defined on a graph $G = (V, E)$ that has a set $E' \subseteq E$ of *red* edges. We are going to call the edges in $E \setminus E'$ *black*.

Definition 6.4 ([125]). Exact Matching.
Input: A graph $G = (V, E)$, a set of red edges $E' \subseteq E$, and a positive integer k.
Output: Determine whether G contains a perfect matching involving exactly k edges in E'.

Let $G = (V, E_{blue} \cup E_{red})$ be a graph in which each edge is colored either blue or red; E_{blue} is the set of blue edges and E_{red} the set of red edges. A matching M in G is called w-blue-red matching if $M \cap E_{blue} \leq w$ and $M \cap E_{red} \leq w$, that is, if it contains at most w edges of each color.

Definition 6.5 ([120]). *(a) Optimization version:* Blue-Red Matching.
Input: A graph $G = (V, E_{blue} \cup E_{red})$ and a positive integer w.
Output: Find a w-blue-red matching of maximum cardinality.

(b) Decision version: Blue-Red Matching(D).
Input: A graph $G = (V, E_{blue} \cup E_{red})$, a positive integer w, and a bound B.
Output: Determine whether G contains a w-blue-red matching of cardinality at least B.

The problem Blue-Red Matching$_{\text{multi}}$, that is a generalization of Blue-Red Matching in multigraphs, can be reduced to BRM in simple graphs with red, blue, and uncolored (white) edges. We specify a third set of initially uncolored edges as follows. Let $G = (V, E_{blue} \cup E_{red} \cup E_{white})$ be a graph in which E_{blue}, E_{red}, and E_{white} are sets of blue, red, and white edges, respectively. A matching M in G is called w-blue-red-white matching if there exists a partition $\{E_{wb}, E_{wr}\}$ of E_{white} such that $M \cap (E_{blue} \cup E_{wb}) \leq w$ and $M \cap (E_{red} \cup E_{wr}) \leq w$.

Definition 6.6. *(a) Optimization version:* Blue-Red Matching$_{\text{multi}}$.
Input: A graph $G = (V, E_{blue} \cup E_{red} \cup E_{white})$ and a positive integer w.

Output: Find a w-blue-red-white matching of maximum cardinality.

(b) Decision version: Blue-Red Matching$_{\text{MULTI}}$(D).

Input: A graph $G = (V, E_{blue} \cup E_{red} \cup E_{white})$*, a positive integer* w*, and a bound* B*.*

Output: Determine whether G *contains a* w*-blue-red-white matching of cardinality at least* B*.*

The problem Blue-Red Matching(D) is at least as hard as Exact Matching, which in turn, is a generalization of Perfect Matching. This fact is shown in the following proposition.

Proposition 6.1 ([120])**.** Perfect Matching $\leq^{\text{l}}_{\text{m}}$ Exact Matching $\leq^{\text{l}}_{\text{m}}$ Blue-Red Matching(D)*, where* $\leq^{\text{l}}_{\text{m}}$ *denotes the log-space many-one reduction between languages.*

Proof. Perfect Matching $\leq^{\text{l}}_{\text{m}}$ Exact Matching: Given a graph $G = (V, E)$ we construct G' by adding two vertices and one red edge between them. Then, there is a perfect matching in G iff there is an exact matching with $k = 1$ red edge in G'.

Exact Matching $\leq^{\text{l}}_{\text{m}}$ Blue-Red Matching(D): Consider a graph $G = (V, E)$, a set of red edges $E' \subseteq E$, and a positive integer k. If $|V|$ is an odd number or $k > \frac{|V|}{2}$, then G does not contain a perfect matching involving exactly k edges in E'. In that case we construct a 'no' instance of BRM(D) (for example, any instance with $2w < B$).

Otherwise, let $w = \max\{k, \frac{|V|}{2} - k\}$ and $r = w - \min\{k, \frac{|V|}{2} - k\}$. Graph G^* is obtained from G by adding $2r$ new vertices $u_1, ..., u_r, v_1, ..., v_r$ and r edges $\{u_1, v_1\}, ..., \{u_r, v_r\}$. The additional edges are colored blue if $k > \frac{|V|}{2} - k$, otherwise they are colored red. Furthermore, edges in $E \setminus E'$ are colored blue and edges in E' remain red in G^*. Let $B = 2w$. The above construction requires logarithmic space. It is not hard to check that G contains a perfect matching involving exactly k edges in E' if and only if G^* contains a w-blue-red matching of cardinality B. □

We define here two variants of Blue-Red Matching, namely the problems Exact BRM and Exact-Equal BRM.

Definition 6.7. *(a)* Exact BRM.

Input: A graph $G = (V, E_{blue} \cup E_{red} \cup E_{white})$ *and a pair of positive integers* (k_1, k_2)*.*

Output: Determine whether G *contains a perfect matching involving exactly* k_1 *edges in* E_{blue} *and exactly* k_2 *edges in* E_{red}*.*

(b) We define Exact-Equal BRM *to be the special case of the problem* Exact BRM*, when* $k_1 = k_2$*.*

We prove below that these problems are harder than Blue-Red Matching.

Proposition 6.2. Blue-Red Matching(D) $\leq_{\mathrm{T}}^{\mathrm{p}}$ Exact BRM $\leq_{\mathrm{m}}^{\mathrm{l}}$ Exact-Equal BRM, *where* $\leq_{\mathrm{T}}^{\mathrm{p}}$ *and* $\leq_{\mathrm{m}}^{\mathrm{l}}$ *denote the poly-time Turing reduction and the log-space reduction between languages, respectively.*

Proof. Blue-Red Matching(D) $\leq_{\mathrm{T}}^{\mathrm{p}}$ Exact BRM: Notice that there are no white edges in the initial instance of Blue-Red Matching(D), but the resulting instance of Exact BRM can have white edges.

If $B > 2w$, we construct a 'no' instance of Exact BRM.

Let $w \leq B \leq 2w$. Given an input $\langle G = (V, E_{red} \cup E_{blue}), w, B\rangle$ to Blue-Red Matching(D), we construct $2w - B + 1$ instances of the Exact BRM problem, namely the graphs G_m, $0 \leq m \leq 2w - B$, where

$$\langle G_m = (V', E_{red} \cup E_{blue} \cup E_{white}), (k_{1m}, k_{2m})\rangle.$$

Let $V = \{v_1, ..., v_n\}$ be the set of vertices in G. We add n new vertices $u_1, ..., u_n$. We add white edges $\{u_i, u_j\}$ for every $i \neq j$. We also add white edges $\{v_i, u_i\}$ for every $i \in \{1, ..., n\}$. In other words, we connect every vertex in V with a vertex in a clique of size n. In the m-th instance, we set the positive integers to be $k_{1m} = w - m$, $k_{2m} = B - w + m$, i.e. we construct an instance for every possible combination of red and blue edges that add up to B. Note that if G contains a w-blue-red matching of cardinality $> B$, then G contains a w-blue-red matching of cardinality exactly B.

It is not hard to see that there exists a w-blue-red matching of cardinality B in G if and only if there exists a perfect matching with exactly k_{1m} blue edges and exactly k_{2m} red edges in at least one of the graphs G_m.

If $B < w$, then we construct $B + 1$ instances of Exact BRM in the same way and we set $k_{1m} = B - m$ and $k_{2m} = m$ in the m-th instance, where $0 \leq m \leq B$.

Exact BRM $\leq_{\mathrm{m}}^{\mathrm{l}}$ Exact-Equal BRM: Let $\langle G = (V, E_{red} \cup E_{blue} \cup E_{white}), (k_1, k_2)\rangle$ be an instance of Exact BRM. Without loss of generality, assume that $k_2 > k_1$. We construct an instance $\langle G' = (V', E_{red} \cup E'_{blue} \cup E_{white}), k_2\rangle$ of Exact-Equal BRM by adding $2(k_2 - k_1)$ vertices $u_1, .., u_{k_2-k_1}, v_1, ..., v_{k_1-k_2}$ and the $k_2 - k_1$ blue edges $\{u_i, v_i\}$. □

Corollary 6.1. Perfect Matching $\leq_{\mathrm{m}}^{\mathrm{l}}$ Exact Matching $\leq_{\mathrm{m}}^{\mathrm{l}}$ Blue-Red Matching(D) $\leq_{\mathrm{T}}^{\mathrm{p}}$ Exact BRM $\leq_{\mathrm{m}}^{\mathrm{l}}$ Exact-Equal BRM.

Decision problem	Complexity
Perfect Matching	P and RNC
Planar Exact Matching	NC
Bipartite Exact Matching	P
Exact Matching	RNC
Blue-Red Matching$_{\text{MULTI}}$(D)	RNC

Table 6.3: The complexity of the problem Perfect Matching and some of its variants discussed here.

The following theorem gives the already known complexity of decision problems discussed in this section. In Theorem 6.2, we show that Exact Matching is in P, by reducing it to a max-flow problem. This fact will be significant when we study its counting version, i.e. #BiExact Matchings. We gather all these results in Table 6.3.

Theorem 6.1. *(a)* ([62]). Perfect Matching *is in* P.
(b) ([99]). Perfect Matching *is in* RNC.
(c) ([12]). Planar Perfect Matching *is in* NC.
(d) ([119]). Exact Matching *is in* RNC.
(e) ([120]). Blue-Red Matching$_{\text{MULTI}}$(D) *is in* RNC.

In the proof of Theorem 6.2 we are using a generalization of the Max Flow problem, namely Max Flow with Lower Bounds. In this modified problem, the input consists of a directed graph $G = (V, E)$, nodes s, t, and two functions $\mathfrak{l}, \mathfrak{u} : E \to \mathbb{R}$ and we seek a flow f with maximum value such that $\mathfrak{l}(e) \leq f(e) \leq \mathfrak{u}(e)$ at every edge e. There is a folklore polynomial-time reduction from Max Flow with Lower Bounds to Max Flow and so the following proposition holds.

Proposition 6.3 ([63])**.** Max Flow with Lower Bounds *is in* P.

Theorem 6.2. Bipartite Exact Matching *is in* P.

Proof. We are going to describe a reduction from Bipartite Exact Matching to Max Flow with Lower Bounds. Given an instance $G = \langle (U \cup V, E), E_{red}, k \rangle$, $|U| = |V| = n$, of Bipartite Exact Matching, we construct a directed graph $G' = (V', E')$ which consists of all vertices of G together with two source vertices s_1, s_2 and a sink vertex t. G' contains the directed edges (s_1, s_2), (s_2, u), for every $u \in U$, (u, v) for every edge $(u, v) \in E$, and (v, t)

for every $v \in V$. The lower and upper bounds are set as follows, where $w_{red} = 2(n-k)$ and $w = w_{red} - 1 = 2(n-k) - 1$.

- For $e = (s_1, s_2)$ we set $\mathfrak{u}(e) = \mathfrak{l}(e) = kw_{red} + (n-k)w$.
- For every $e = (u, v)$ which is a red edge in G, we set $\mathfrak{u}(e) = \mathfrak{l}(e) = w_{red}$.
- For every $e = (u, v)$ which is not a red edge in G, we set $\mathfrak{u}(e) = \mathfrak{l}(e) = w$.
- For every $e = (s_2, u)$ or $e = (v, t)$, we set $\mathfrak{l}(e) = w$ and $\mathfrak{u}(e) = w_{red}$.

An example of a resulting graph G' is shown in Figure 6.1. We prove that there is an exact matching with k red edges in G iff there is a flow of value $kw_{red} + (n-k)w$ in graph G'. If there is an exact matching with k red edges and $(n-k)$ black edges in G, then it is easy to see that there is a flow of value $kw_{red} + (n-k)w$ in graph G'. For the inverse direction, we show that for any flow of value $kw_{red} + (n-k)w$, the following facts hold.

1. At most n edges that have their endpoints in U and V can carry a non-negative flow. The flow that leaves from a vertex $u \in U$ is routed on exactly one edge. Since the amount of flow that reaches vertex u is either w or w_{red} and the lower and upper bounds of edges (u, v), $u \in U$, $v \in V$, are tight, this flow can be routed on either exactly one red or exactly one black edge.

2. A flow of value $kw_{red} + (n-k)w$ has to use exactly k red edges and $(n-k)$ black edges that have their endpoints in U and V.

   (a) A flow of value $kw_{red} + (n-k)w$ needs at least k red edges. Suppose that k' red edges are used, with $k' < k$, then the rest of the flow must be rooted on black edges, which can carry a flow of value $w < w_{red}$ each. So, more than $(n-k')$ black edges will be needed, which contradicts 1.

   (b) At most k red edges can be used in the solution. Suppose that instead of using k red edges and $n-k$ black edges, we replace $2 \leq m \leq n-k$ black edges with $l < m$ red edges. We can say that $l = m - i$ for some $i \in \{1, \ldots, m-1\}$. Then

$$\begin{aligned} mw &= lw_{red} \Leftrightarrow \\ \frac{w_{red}}{w} &= \frac{m}{l} \Leftrightarrow \\ \frac{2(n-k)}{2(n-k)-1} &= \frac{m}{m-i} \Leftrightarrow \\ m &= i \cdot 2(n-k). \end{aligned}$$

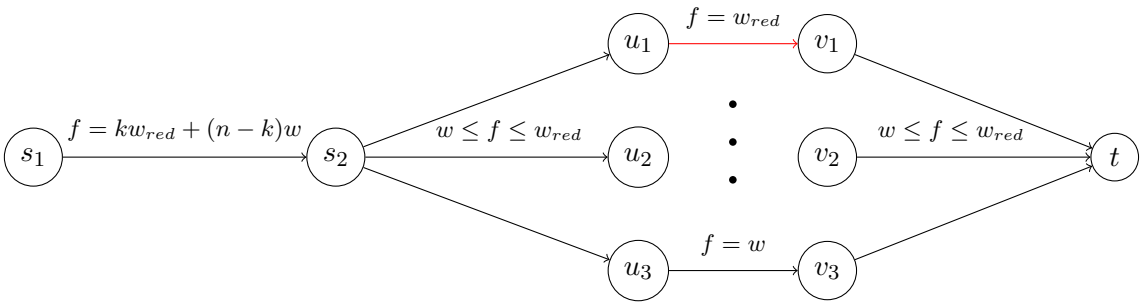


Figure 6.1: The resulting instance of Max Flow with Lower Bounds. When we write $f = c$ on an edge e, for some $c \in \mathbb{R}$, we mean that $\mathfrak{l}(e) = \mathfrak{u}(e) = c$. For example, if $e \in E_{red}$, then $\mathfrak{l}(e) = \mathfrak{u}(e) = w_{red}$. The red color of edge (u_1, v_1) denotes that edge $(u_1, v_1) \in E_{red}$ in graph G.

Contradiction, since $m \leq n - k$. □

6.2.1 Optimization version of Exact Matching in bipartite graphs

Our motivation to study the optimization version of Exact Matching in bipartite graphs is the following. There are randomized algorithms [29, 33] which approximate the value of a counting function by using oracle calls to the corresponding optimization problem. This method produces a very crude estimate, but gives some general information about the value of the function for large n. For example, it allows to determine whether it is exponentially large in n. In the case of a polynomial-time optimization problem, these randomized algorithms run in polynomial time.

Definition 6.8. Minimum Weight Perfect Matching in Bipartite Graphs.
Input: A bipartite graph $G = (U \cup V, E)$ and a weight function $w : E \to \mathbb{R} \cup \{\infty\}$.
Output: Find a perfect matching M minimizing $w(M) = \sum_{e \in M} w(e)$.

The following is an Integer Linear Programming (ILP) formulation for the minimum weight perfect matching in a bipartite graph.

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v)} w(u,v) \cdot x_{uv} \\ \text{subject to} \quad & \sum_{v \in V} x_{uv} = 1 \text{ for all } u \in U \\ & \sum_{u \in U} x_{uv} = 1 \text{ for all } v \in V \\ & x_{uv} \in \{0,1\} \text{ for all } u \in U, v \in V. \end{aligned}$$

The Linear Programming (LP) relaxation of the problem is as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{(u,v)} w(u,v) \cdot x_{uv} \\
\text{subject to} \quad & \sum_{v \in V} x_{uv} = 1 \text{ for all } u \in U \\
& \sum_{u \in U} x_{uv} = 1 \text{ for all } v \in V \\
& x_{uv} \geq 0 \text{ for all } u \in U, v \in V.
\end{aligned}$$

In the above LP instance, the constraint matrix of the polytope is totally unimodular, i.e. every square submatrix has determinant 0, +1 or -1. As a result, any extreme point of the polytope defined by these constraints is integral. So there is a polynomial-time algorithm for finding minimum weight perfect matching in a bipartite graph.

For the Exact Matching problem, the optimization version is defined as follows.

Definition 6.9. Minimum Weight Exact Matching in Bipartite Graphs.
Input: A bipartite graph $G = (U \cup V, E)$, a subset $E' \subseteq E$, an integer k, and a weight function $w : E \to \mathbb{R} \cup \{\infty\}$.
Output: Find an exact matching M minimizing $w(M) = \sum_{e \in M} w(e)$.

This problem is described by the following ILP.

$$\begin{aligned}
\text{minimize} \quad & \sum_{(u,v) \in E \setminus E'} w(u,v) \cdot x_{uv} + \sum_{(u,v) \in E'} w(u,v) \cdot y_{uv} \\
\text{subject to} \quad & \sum_{v \in V} (x_{uv} + y_{uv}) = 1 \text{ for all } u \in U \\
& \sum_{u \in U} (x_{uv} + y_{uv}) = 1 \text{ for all } v \in V \\
& \sum_{(u,v) \in E'} y_{uv} = k \\
& x_{uv}, y_{uv} \in \{0,1\} \text{ for all } u \in U, v \in V.
\end{aligned}$$

The LP relaxation is given by the same constraints but instead of the variables being in $\{0,1\}$, we have $x_{uv}, y_{uv} \geq 0$. In this case, the constraint matrix is not in general totally unimodular. This is a corollary of the following propositions that can be found in [135].

Proposition 6.4. *Let M be a $\{0, \pm 1\}$ matrix with at most three non-zero entries in each column. Then M is totally unimodular if and only if each submatrix of M with at most two non-zero entries in each column is totally unimodular.*

Proposition 6.5. *Let M be a $\{0, \pm 1\}$ matrix with exactly two non-zero entries in each column. Then M is totally unimodular if and only if the rows of M can be split into two classes such that for each column: if the two non-zeros in the column have the same sign then they are in different classes and if they have opposite sign then they are both in one and the same class.*

Note that every instance of the above ILP has at most three non-zero entries. Also, one can easily construct an instance, such that the constraint matrix has a submatrix for which the condition of Proposition 6.5 does not hold.

We have not concluded whether the optimization version of #Exact Matchings in bipartite graphs is either in P or NP-hard. However, our attempt to prove that it is in P by using the ILP formulation of the problem has failed.

6.3 The problem #Exact Matchings

6.3.1 #Exact Matchings in general graphs

The reductions of Proposition 6.1 between the decision problems Perfect Matching, Exact Matching, and Blue-Red Matching(D), are parsimonious reductions between their counting versions.

Proposition 6.6. #PerfMatch $\leq^{\text{p}}_{\text{pars}}$ #Exact Matchings $\leq^{\text{p}}_{\text{pars}}$ #Blue-Red Matchings.

Using polynomial interpolation, we show next that #Exact Matchings is reducible to #PerfMatch in graphs with edge weights. Reductions that use polynomial interpolation are Turing reductions. Another example of such a reduction is the one from #PerfMatch to #Matchings given by Valiant [149].

Proposition 6.7. #Exact Matchings $\leq^{\text{p}}_{\text{T}}$ #Weighted PerfMatch.

Proof. Let $\langle G = (V, E), E', k\rangle$ be an input to the problem #Exact Matchings. Define the polynomialLet where m_k is the number of exact matchings with k red edges. We can evaluate $P(x)$ at $n/2 + 1$ points using an oracle for #Weighted PerfMatch as follows. For every integer $1 \leq \rho \leq n/2+1$, we construct a graph $G_\rho = (V, E)$, which is G with the weight function $w_\rho : E \to \mathbb{N}$, that assigns weight ρ to every (red) edge in E' and 1 to every (black) edge in $E \setminus E'$. Then, we have that #Weighted PerfMatch$(G_\rho) = P(\rho)$. So, we make $n/2+1$ oracle calls, one for each graph G_ρ, $1 \leq \rho \leq n/2 + 1$. Using polynomial interpolation, we can recover

the coefficients of polynomial $P(x)$. In particular, m_k is the output of the problem #Exact Matchings on input $\langle G = (V, E), E', k \rangle$. □

Parameterized complexity

Let p#Cliques be the problem of counting cliques of size k in a graph G, parameterized by k. Define the class #W[1] as the set of parameterized counting problems $\#A$ with $\#\text{A} \leq^{\text{T}}_{\text{fpt}}$ p#Cliques. Here, $\#\text{A} \leq^{\text{T}}_{\text{fpt}}$ #B means that #A admits an fpt-algorithm that solves instances (x, k) of #A with oracle access to #B, under the restriction that all oracle queries (y, k') satisfy the condition $k' \leq g(k)$ for some computable $g : \mathbb{N} \to \mathbb{N}$. The following theorem gives evidence that #W[1] $\neq$ FPT, where FPT is the class of fixed-parameter tractable problems.

Theorem 6.3 ([68]). *Under #ETH, it holds that* #W[1] $\neq$ FPT.

For more details we refer the reader to [68].

The problem #k-Matchings parameterized by k has been proven to be #W[1]-complete [51]. Below, we show a reduction from #k-Matchings to #Exact Matchings that implies #W[1]-hardness for #Exact Matchings parameterized by k. The results discussed in this chapter with respect to parameterized complexity are shown in Table 6.4.

Proposition 6.8. $\#k$-Matchings $\leq^{\text{p}}_{\text{pr}}$ #Exact Matchings.

Proof. Let $\langle G_1 = (V_1, E_1), k \rangle$, $|V_1| = n$, be an instance of #k-Matchings. We construct $G_2 = (V_2, E_2)$ and E_2' as follows. First, we add n new vertices $v_1, \ldots, v_n$ to G_1 and the black edges (v_i, v_j) for every $i, j \in \{1, \ldots, n\}$ (a black n-clique). For every vertex $u_i \in V_1$, $i \in \{1, \ldots, n\}$, we add the black edge (u_i, v_i). The set E_2' of red edges in G_2 consists of all the edges in E_1.

Every k-matching in G_1 can be extended to $\frac{(2k)!}{2^k k!}$ exact matchings in G_2. Let M be a k-matching in G_1. The k edges in M are k red edges in G_2. An exact matching M' in G_2 that extends M contains also an edge (u_i, v_i) for every matched vertex $v_i \in V_1$ not matched by M. These extra edges are $(n - 2k)$ in total. For the rest $2k$ vertices in the clique that are still unmatched, there are $\frac{(2k)!}{2^k k!}$ different ways that can be matched (the number of perfect matchings in a complete graph with $2k$ vertices). M' contains the edges of one of these possible matchings among the $2k$ vertices. Moreover, every exact matching with k red edges in G_2 is obtained uniquely by a k-matching in G_1.

So $\#k$-Matchings$(G_1) = \#$Exact Matchings$(G_2) \cdot \frac{2^k k!}{(2k)!}$. □

Corollary 6.2. #Exact Matchings *is* #W[1]*-hard.*

Proof. By Proposition 6.8 and #W[1]-hardness of #k-Matchings [51], we have that #Exact Matchings is also #W[1]-hard. □

Counting problem	Parameterized Complexity	Exact algorithm	Approximation algorithm
#k-Matchings	#W[1]-complete	$\mathcal{O}(2^{k+o(k)}\binom{n}{k/2})$	fptras
#Planar-k-Matchings	#W[1]-complete	—	—
#Bipartite-k-Matchings	#W[1]-complete	—	—
#Bipartite-k-Matchings with $f(k)$ vertices in one partition	—	fpt	—
#Exact Matchings	#W[1]-hard	—	—

Table 6.4: The parameterized complexity of the problems #k-Matchings and #Exact Matchings.

6.3.2 #Exact Matchings in $K_{3,3}$-free graphs

#Exact Matchings was shown to be in NC by Vazirani [152]. This result is based on the fact that we can efficiently compute the number of perfect matchings in any graph that has an efficiently computable pfaffian orientation.

We roughly describe how we can compute the number of perfect matchings in a graph that has a pfaffian orientation, such as a planar or a $K_{3,3}$-free graph. For a detailed proof, we refer to [42, Chapter 4]. An orientation $\overrightarrow{G}$ of an undirected graph G is an assignment of a direction to each of its edges. An orientation is pfaffian, if for any two perfect matchings M, M' in G, every cycle in $M \cup M'$ is oddly oriented, which means that when traversing the cycle, in either direction, the number of co-oriented edges is odd. Given a pfaffian orientation, we define the

skew adjacency matrix $A_S(\overrightarrow{G}) = (a_{ij} : 0 \leq i, j \leq n-1)$ of G by $a_{ij} = \begin{cases} +1, & \text{if } (i,j) \in E(\overrightarrow{G}) \\ -1, & \text{if } (j,i) \in E(\overrightarrow{G}) \\ 0, & \text{otherwise} \end{cases}$.

Then, it holds that $\#\textsc{PerfMatch}(G) = \sqrt{\det(A_S(\overrightarrow{G}))}$.

Theorem 6.4 ([152]). *$\#$Exact Matchings in $K_{3,3}$ graphs is in* NC.

Proof. Let $\overrightarrow{G}$ be a pfaffian orientation of a $K_{3,3}$-free graph $G = (V, E)$ with a set E' of red edges, which can be computed in NC [152]. We assign to every edge $e \in E$ the polynomial

$$p_e(x) = \begin{cases} x, & \text{if } e \in E', \\ 1, & \text{otherwise.} \end{cases}$$

We construct the matrix $A_S(\overrightarrow{G}) = (a_{ij} : 0 \leq i, j \leq n-1)$ of G by

$$a_{ij} = \begin{cases} +p_e(x), & \text{if } (i,j) \in E(\overrightarrow{G}) \\ -p_e(x), & \text{if } (j,i) \in E(\overrightarrow{G}) \\ 0, & \text{otherwise} \end{cases}.$$

Then $\sqrt{\det(A_S(\overrightarrow{G}))} = \sum_{k=0}^{n/2} m_k x^k$, where m_k is the number of perfect matchings containing k red edges. So we compute the determinant of A_S using the parallel determinant algorithm of [39], compute its square root by interpolation, and return the coefficient of x^k. □

Since planar graphs are $K_{3,3}$-free, we have the following corollary.

Corollary 6.3. *$\#$Exact Matchings in planar graphs is in* NC.

6.3.3 #Exact Matchings in K_5-free graphs

To compute the number of perfect matchings, the pfaffian orientation technique is not applicable to K_5-free graphs, because some K_5-free graphs have no such orientation. The graph $K_{3,3}$ is such an example [116].

An algorithm for the problem $\#\textsc{PerfMatch}$ in K_5-free graphs, that uses dynamic programming, was described in [141]. It is based on the decomposition of K_5-free graphs into 2-, 3-, and 4-connected components studied by Wagner [154]. The components will be planar at some point, except for one type of component, namely the Möbius ladder M_8, which has constant

size. The algorithm of [141] reduces the problem of computing the number of perfect matchings in K_5-free graphs to the one for planar graphs. To solve the problem for an input K_5-free graph, the challenge was to combine the results for its planar components correctly. For details, we refer to [141].

It is not hard to see that the aforementioned algorithm works even when the input is a K_5-free graph with edge weights and we want to solve #Weighted PerfMatch as defined in Definition 6.3 (and Remark 6.1). This observation implies the following result.

Proposition 6.9. *#Exact Matchings in K_5-free graphs is in* P.

Proof. As shown in Proposition 6.7, #Exact Matchings $\leq_{\mathrm{T}}^{\mathrm{p}}$ #Weighted PerfMatch. In the reduction, the oracle calls are to #Weighted PerfMatch on graphs with edge weights being natural numbers. Since #Weighted PerfMatch in K_5-free graphs has a polynomial-time algorithm in such graphs, #Exact Matchings in K_5-free graphs is in P. □

It is clear that if we restrict ourselves to a class of graphs that #Weighted PerfMatch is polynomial-time computable, then #Exact Matchings is also polynomial-time computable. So, if the dichotomy result of [143] can be extended to the weighted version of counting perfect matchings, then it also holds for the problem #Exact Matchings.

6.3.4 #Exact Matchings in bipartite graphs

#Exact Matchings in bipartite graphs seems to be non-trivial, since neither approximability nor hardness of approximation can be proven easily. After the completion of this thesis, this problem remains open with respect to approximability.

Below we mention some cases where #Exact Matchings in bipartite graphs has an fpras. The proof for each of them requires just some simple observations.

At the end of the current subsection, we propose and give a high level description of an approach that we believe is promising for showing that an fpras for #Exact Matchings in bipartite graphs exists. This approach when applied on a counting (or sampling) problem, it makes use of the generating polynomial of the problem. This fact justifies the subject of the next section, that is Section 6.4, which deals with various polynomials generated by matching problems.

First, note that if we restrict ourselves to bipartite graphs, #Exact Matchings is a

problem in TotP.

Proposition 6.10. *#Exact Matchings in bipartite graphs is in* TotP.

Proof. By Proposition 6.2, the decision version of #Exact Matchings in bipartite graphs is in P. It is not hard to see that the problem is also self-reducible. So, it belongs to TotP. □

Case 1. The class of exact matchings is of polynomial size

By Proposition 1.4, if a TotP problem has polynomially many solutions, then there is a polynomial-time algorithm that enumerates them.

Case 2. The class of exact matchings is dense

Let $I = \langle G = (U \cup V, E), E', k\rangle$ be an instance of #Exact Matchings. Let $\mathcal{M}_k$ denote the class of exact matchings with exactly k red edges and $\mathcal{M}$ denote the set of all perfect matchings of G. Suppose that $\frac{|\mathcal{M}|}{|\mathcal{M}_k|} \leq p(|I|)$, for some polynomial p. Then we can simply call the fpaus for the problem of perfect matchings in bipartite graphs [93] and return its output if it contains exactly k red edges. Repeating this $p(|I|)$ times, we expect to find an exact matching with k red edges.

Case 3. A few vertices in U are adjacent to red edges

Consider that we are given an instance $\langle G = (U \cup V, E), E', k\rangle$ of #Exact Matchings in bipartite graphs, in which red edges (i.e. edges in E') are adjacent to only $k + \mathcal{O}(1)$ vertices in U. We choose a set S_k of exactly k vertices in U and do the following. We are using a modified biadjacency matrix $A' = (a'_{ij})$ of G so

$$a'_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E' \text{ and } i \in S_k \\ & \text{or } (i,j) \in E \text{ and } i \notin S_k \\ 0, & \text{otherwise} \end{cases}$$

Then $\text{perm}(A')$ gives the number of perfect matchings of G in which all the vertices in S_k are covered by red edges. The sum of the number of such perfect matchings for all choices of S_k gives the number of exact matchings with k red edges. Since there are $\binom{k+\mathcal{O}(1)}{k} = \mathcal{O}(k^{\mathcal{O}(1)})$ different choices of sets S_k, then by calling the fpras for computing each one of these $\mathcal{O}(k^{\mathcal{O}(1)})$

many permanents [93], we obtain an fpras for the number of exact matchings. Of course, the above arguments hold also in the case that red edges are adjacent to only $k + \mathcal{O}(1)$ vertices in V. So we conclude to the following fact.

Proposition 6.11. *There is an* fpras *for* #Exact Matchings *in any bipartite graph* $G = (U \cup V, E)$ *with a set* $E' \subseteq E$ *of red edges, where the edges in* E' *are adjacent to only* $k + \mathcal{O}(1)$ *vertices in* U *(or* $k + \mathcal{O}(1)$ *vertices in* V*).*

A promising approach

Here we are going to describe the technique presented in [6], which is used to sample efficiently solutions of particular problems. The main idea is that there is a connection between efficient sampling from a probability distribution and the zero-free region of the polynomial *generated* by the distribution.

Because of the equivalence between approximate counting and approximate sampling for self-reducible problems [94], this sampling technique yields fpras for the corresponding counting problems. For example, as we have already mentioned in Section 6.1, it yields an fpras for counting k-matchings in planar graphs.

Let $\mu : \binom{[n]}{k} \to \mathbb{R}_{\geq 0}$ be a density function on the family of subsets of size k out of a ground set of n elements. Our goal is to approximately sample from the probability distribution $\mathbb{P}_\mu[S] \propto \mu(S)$ efficiently. To do so we are going to define the following Down-Up Random Walk on subsets of size ℓ and subsets of size k, for some $l \leq k$.

Down-Up Random Walk.

For a density $\mu : \binom{[n]}{k} \to \mathbb{R}_{\geq 0}$ and an integer $\ell \leq k$, we define the $k \leftrightarrow \ell$ down-up random walk as the sequence of random sets $S_0, S_1, \ldots$ generated by the following algorithm:

for $t = 0, 1, \ldots$ **do**

Select T_t uniformly at random from subsets of size ℓ of S_t. (step 1.)

Select S_{t+1} with probability $\propto \mu(S_{t+1})$ from supersets of size k of T_t. (step 2.)

Note that each step of this random walk can be efficiently implemented as long as $k - \ell = \mathcal{O}(1)$ and we have oracle access to μ. This is because the number of supersets of T_t is at most $n^{k-\ell} = \text{poly}(n)$, so we can enumerate them in polynomial time.

Down-Up Random Walk is time-reversible, has μ as its stationary distribution, and has positive real eigenvalues [11]. Rapid mixing of the $k \leftrightarrow l$ down-up walk is established by

showing that the generating polynomial of μ is zero-free in a symmetric sector of the complex plane centered around the real axis. The generating polynomial of μ is defined next.

Definition 6.10. *Let $\mu : \binom{[n]}{k} \to \mathbb{R}_{\geq 0}$ be a density function. The multivariate polynomial g_μ defined as follows:*

$$g_\mu(z_1, \ldots, z_n) := \sum_S \mu(S) \prod_{i \in S} z_i$$

is called the generating polynomial and encodes μ in its coefficients.

Theorem 6.5. *Let $\mu : \binom{[n]}{k} \to \mathbb{R}_{\geq 0}$ be a density function and g_μ be the corresponding generating polynomial. Suppose there is an open sector $\Gamma \subseteq \mathbb{C}$ of aperture $\Omega(1)$ centered around the positive real axis in the complex plane, such that*

$$z_1, \ldots, z_n \in \Gamma \Rightarrow g_\mu(z_1, \ldots, z_n) \neq 0.$$

Then for an appropriate value $\ell = k - \mathcal{O}(1)$, the $k \leftrightarrow \ell$ has relaxation time $k^{\mathcal{O}(1)}$.

We will see in Definition 6.12 that a polynomial that has a zero-free region like the one described in the above theorem, is called α-sector-stable, where α is the aperture of the open sector Γ.

The relaxation time is the inverse of spectral gap for a time-reversible Markov chain with positive eigenvalues. If in addition, the starting point has not terribly small probability, the mixing time can be polynomially bounded [111]. So, by the following corollary, we have that the $k \leftrightarrow \ell$ random walk has a polynomially bounded mixing time.

Corollary 6.4. *[111] Let $\mu : \binom{[n]}{k} \to \mathbb{R}_{\geq 0}$ be a density function and g_μ be the corresponding generating polynomial. Suppose there is an open sector $\Gamma \subseteq \mathbb{C}$ of aperture $\Omega(1)$ centered around the positive real axis in the complex plane, such that*

$$z_1, \ldots, z_n \in \Gamma \Rightarrow g_\mu(z_1, \ldots, z_n) \neq 0.$$

Let also $\ell = k - \mathcal{O}(1)$ be the value promised by Theorem 6.5. If the $k \leftrightarrow \ell$ down-up random walk is started from S_0, then

$$t_{mix}(\varepsilon) \leq \mathcal{O}\Big(k^{\mathcal{O}(1)} \cdot \log\big(\frac{1}{\varepsilon \cdot \mathbb{P}_\mu[S_0]}\big)\Big)$$

where $t_{mix}(\varepsilon)$ is the smallest time t such that S_t is ε-close in total variation distance to the distribution defined by μ.

Corollary 6.4 establishes a connection between sector-stability and polynomially bounded mixing time. For the interested reader, α-sector-stability implies that μ is spectrally independent, i.e. an associated pairwise influence matrix has a bounded largest eigenvalue for the distribution and all of its conditional distributions. Using the results of other recent works [11, 5], rapid mixing time is obtained. For details, we refer to [6, 11, 5].

Application of the Down-Up Random Walk in our case

We give some general ideas of how this sampling technique could be used in the case of exact matchings in bipartite graphs.

Let $\langle G = (U \cup V, E), E', k\rangle$ be an instance of #Exact Matchings in bipartite graphs with $|U| = |V| = n$, $|E| = m$, and $|E'| = r$. Given $S \subseteq U \cup V$, we denote by $G[S]$ (resp. $G[\overline{S}]$) the subgraph of G induced by S (resp. the complement $\overline{S}$ of S). Given $T \subseteq E'$, we denote by V_T the set of vertices in $U \cup V$ covered by T.

First idea: If the following hold then the Down-Up Random Walk can be used for sampling uniformly at random an exact matching of G.

1. The generating polynomial g_μ is α-sector-stable for a constant α, where $\mu : \binom{[2n]}{2k} \to \mathbb{R}_{\geq 0}$ is a density function defined on sets of vertices of size $2k$ such that

$$\mu(S) = \#(\text{perfect matchings in } G[S] \text{ with edges in } E') \cdot \#(\text{perfect matchings in } G[\overline{S}] \text{ with edges in } E \setminus E').$$

2. If step 2 of the Down-Up Random Walk is implemented using an fpras for computing μ instead of a polynomial-time algorithm (or an oracle), then the same facts hold, i.e. the Markov chain of the random walk has μ as its stationary and has rapid mixing time.

If 1 and 2 are true, then we implement the $k \leftrightarrow \ell$ down-up random walk to efficiently sample a set S of $2k$ vertices with probability $\mathbb{P}_\mu \propto \mu$. Then we choose u.a.r. a perfect matching in $G[S]$ with edges in E' and a perfect matching in $G[\overline{S}]$ with edges in $E \setminus E'$. Combining these two perfect matchings we obtain an exact matching with k red edges that has been chosen u.a.r.

Second idea: If the following hold then the Down-Up Random Walk can be used for sampling uniformly at random an exact matching of G.

1. The generating polynomial g_μ is α-sector-stable for a constant α, where $\mu : \binom{[r]}{k} \to \mathbb{R}_{\geq 0}$ is

a density function defined on sets of red edges of size k such that

$$\mu(T) = \begin{cases} \#(\text{perfect matchings in } G[\overline{V_T}] \text{ with edges in } E \setminus E'), & \text{if } T \text{ is a matching} \\ 0, & \text{otherwise} \end{cases}$$

2. If step 2 of the Down-Up Random Walk is implemented using an fpras for computing μ instead of a polynomial-time algorithm (or an oracle), then the same facts hold, i.e. the Markov chain of the random walk has μ as its stationary and has rapid mixing time.

If 1 and 2 are true, then we implement the $k \leftrightarrow \ell$ down-up random walk to efficiently sample a set T of k red edges with probability $\mathbb{P}_\mu \propto \mu$. Then we choose u.a.r. a perfect matching in $G[\overline{V_T}]$ with edges in $E \setminus E'$. Combining T with the perfect matching $G[\overline{V_T}]$ we obtain an exact matching with k red edges that has been chosen u.a.r.

In the following section we define matching polynomials. Those that are multiaffine and homogeneous, i.e. each of their terms contains the same number of variables, are generating polynomials of some density μ. The polynomials described in the first and second ideas above are given in Subsections 6.4.4 and 6.4.5, respectively.

6.4 Matching polynomials

Let $G = (V, E)$ be a weighted graph, i.e. a graph with edge weights. We are using v to denote a vertex in V, e to denote an edge in E, and w_e to denote the weight of the edge e.

We are going to consider univariate and multivariate polynomials over $\mathbb{C}$ that have real coefficients. Some first useful notions are the following.

- A multivariate polynomial is *multiaffine* if each variable has degree at most one.
- A multivariate polynomial is *homogeneous* if all non-zero terms have the same degree. In specific, a multivariate multiaffine polynomial is homogeneuous if each non-zero term contains the same number of variables.

We are interested in the zero-free regions of these polynomials. Definition 6.12 essentially gives characterizations of polynomials with respect to their zero-free regions.

Definition 6.11. *We denote by*

(a) $\mathcal{H}_+$*: the open upper half-plane of* $\mathbb{C}$*.*

(b) $\mathcal{D}$*: the open unit disc, i.e.* $\mathcal{D} = \{z \in \mathbb{C} \mid |z| < 1\}$.

(c) Γ_α*: the open sector of aperture* $\alpha\pi$ *centered around the positive axis, i.e.*

$$\Gamma_\alpha := \{\exp(x + iy) \mid x \in \mathbb{R}, y \in (-\alpha\pi/2, \alpha\pi/2)\}.$$

Definition 6.12 ([38, 6])**.** *(a) For an open subset* $\mathcal{U} \subseteq \mathbb{C}^n$*, we call a polynomial* $p \in \mathbb{C}[x_1, \ldots, x_n]$ $\mathcal{U}$*-stable if*

$$(z_1, \ldots, z_n) \in \mathcal{U} \Longrightarrow p(z_1, \ldots, z_n) \neq 0.$$

(b) A polynomial $p \in \mathbb{C}[x_1, \ldots, x_n]$ *is said to be stable if*

$$(z_1, \ldots, z_n) \in \mathcal{H}_+^n \Longrightarrow p(z_1, \ldots, z_n) \neq 0.$$

Additionally, if $p \in \mathbb{R}[x_1, \ldots, x_n]$*, we say that* p *is real-stable.*
A univariate polynomial is real-stable iff it is real rooted.

(c) A polynomial $p \in \mathbb{C}[x_1, \ldots, x_n]$ *is said to be Hurwitz-stable if*

$$(z_1, \ldots, z_n) \in \Gamma_1^n \Longrightarrow p(z_1, \ldots, z_n) \neq 0.$$

(d) A polynomial $p \in \mathbb{C}[x_1, \ldots, x_n]$ *is said to be* α*-sector-stable if*

$$(z_1, \ldots, z_n) \in \Gamma_\alpha^n \Longrightarrow p(z_1, \ldots, z_n) \neq 0.$$

Next we introduce the notion of same-phase stability.

Definition 6.13 ([110])**.** *A polynomial* $p \in \mathbb{R}[x_1, \ldots, x_n]$ *is said to be same-phase stable if one of the following equivalent conditions is satisfied.*

(i) For every $\overrightarrow{t} \in \mathbb{R}_+^n$*, the univariate restriction* $p(\overrightarrow{t}x)$ *is real-stable (and therefore real rooted).*

(ii) If $\arg(z_1) = \arg(z_2) = \ldots = \arg(z_n)$*, then* $p(z_1, \ldots, z_n) = 0$ *implies* $z_k \notin \mathcal{H}_+$ *for some* k*.*

6.4.1 The univariate matching polynomial

Let $A = (a_{ij})$ be the $n \times n$ symmetric adjacency matrix of G, which means that the entry a_{ij} is equal to w_e, where $e = (i, j)$. The **univariate matching polynomial** is defined as follows.

Definition 6.14. *We define the univariate matching polynomial by*

$$p_A(x) = \sum_{0}^{\lfloor \frac{n}{2} \rfloor} haf_m(A)x^m$$

where

$$haf_m(A) = \sum_{\{i_1,j_1\},\ldots,\{i_m,j_m\}} a_{i_1,j_1} \cdots a_{i_m,j_m}$$

where the sum is taken over all unordered collections of m pairwise disjoint unordered pairs $\{i_1, j_1\}, ..., \{i_m, j_m\}$. We agree that $h_0(A) = 1$.

Note that the coefficient $h_m(A)$ is the sum of all weighted matchings with m edges in a complete weighted graph. In statistical physics $p_A(x)$ is the partition function of the *monomer-dimer model*, where edges of the matching correspond to dimers and the vertices not covered by the matching correspond to monomers (single atoms).

Theorem 6.6 ([82])**.** *Let A be a $n \times n$ symmetric matrix with non-negative real entries. Then the matching polynomial $p_A(x)$ is real-stable. More precisely, the roots of the matching polynomial $p_A(x)$ are negative real.*

Remark 6.2. *It also holds that the polynomial*

$$p'_A(x) = \sum_{0}^{\lfloor \frac{n}{2} \rfloor} haf_m(A)x^{\frac{n}{2}-m}$$

is real-stable. Proofs of the above theorem and this remark can be found in [82, 32].

The above result is important because Barvinok's technique [32, Section 2] can be applied to the partition function of the monomer-dimer model. Barvinok related a zero-free region of a univariate polynomial with its efficient evaluation. The polynomial is considered on the complex plane, i.e. the variable x takes complex values. Barvinok states that if a polynomial $P(x) = \sum_{i=1}^{n} c_i x^i$ of degree n is zero-free in a strip containing $[0, 1]$ (on the complex plane), then $P(1)$ can be $(1 \pm \varepsilon)$-approximated using $c_0, \ldots, c_k$ for some $k = \mathcal{O}(\log \frac{n}{\varepsilon})$. The basic idea is to truncate the Taylor expansion of $\log P(x)$ at $x = 0$. Let $g(x) := \log P(x)$ and for $k \geq 0$,

$$T_k(g)(x) := \sum_{i=0}^{k} \frac{g^{(i)}(0)}{i!} x^i, \text{ where } g^{(i)} \text{ is the } i\text{-th derivative of } g.$$

Then Barvinok states that if $P(x)$ is zero-free in the disk of radius $\beta > 1$ centered at the origin, then there exists a constant C_β such that for any $0 < \varepsilon < 1$,

$$\left| \frac{\exp(T_k(g)(1))}{P(1)} - 1 \right| \leq \varepsilon, \text{ for some } k = C_\beta \log \frac{n}{\varepsilon}.$$

So, when the polynomial is zero-free in the disk of radius $\beta > 1$, then we can approximately evaluate $P(1)$ using the first $\mathcal{O}(\log \frac{n}{\varepsilon})$ terms of the Taylor expansion of $\log P(x)$ at the origin. If the polynomial is zero-free in a strip of $[0, 1]$, then we can apply a transformation to transform it into a polynomial that is zero-free in the disk of radius > 1.

This technique has been used to construct deterministic quasi-polynomial-time approximation algorithms for evaluating a number of graph partition functions (for general graphs). Related work can be found in [30, 19, 34, 31]. Patel and Regts [127] gave a polynomial-time algorithm for computing the first $\mathcal{O}(\log n)$ coefficients of a large class of graph polynomials when the graph has bounded degree. In particular, they obtained fptas for evaluating the independence polynomial, the Tutte polynomial, and computing partition functions of spin and edge-coloring models in the case of bounded degree graphs.

6.4.2 The multivariate (vertex) matching polynomial

Below we define the **multivariate (vertex) matching polynomial**.

Definition 6.15. *Let $G = (V, E)$ be a graph with edge weights w_e, $e \in E$. We define the multivariate (vertex) matching polynomial by*

$$p_V(x_{v_1}, ..., x_{v_n}) = \sum_{M \in \mathcal{M}} \text{weight}(M) \prod_{v \in M} x_v$$

where $\mathcal{M}$ is the set of matchings, $\text{weight}(M) = \prod_{e \in M} w_e$, *and $n = |V|$.*

Note that each term corresponds to a different matching and each variable x_v of a term corresponds to a vertex that belongs to this matching. The coefficient of the term is the weight of the matching. Matchings that contain the same vertices contribute similar terms to the multivariate matching polynomial.

Remark 6.3. *Alternatively, we can define the multivariate matching polynomial to be the following.*

$$p'_V(x_{v_1}, ..., x_{v_n}) = \sum_{M \in \mathcal{M}} \text{weight}(M) \prod_{v \notin M} x_v.$$

The only difference here is that a term, corresponding to a matching M, contains a variable for each vertex not covered by M.

Remark 6.4. *Note that these polynomials are multiaffine, i.e. multivariate polynomials in which each variable has degree at most one.*

Theorem 6.7 ([82, 49]). *Let $G = (V, E)$ be a graph with non-negative edge weights w_e, $e \in E$. The polynomials*

$$p_V(x_{v_1}, ..., x_{v_n}) = \sum_{M \in \mathcal{M}} \text{weight}(M) \prod_{v \in M} x_v$$

and

$$p'_V(x_{v_1}, ..., x_{v_n}) = \sum_{M \in \mathcal{M}} \text{weight}(M) \prod_{v \notin M} x_v$$

are Hurwitz-stable, that is, if $\text{Re}(z_1) > 0, ..., \text{Re}(z_n) > 0$, then $p_V(z_1, ..., z_n) \neq 0$ (resp. $p'_V(z_1, ..., z_n) \neq 0$). In other words, they are zero-free in the right half of the complex plane.

Remark 6.5. *The stability of p_V and p'_V is also true for the left half-plane (i.e. if $\text{Re}(z_i) < 0$, for every $1 \leq i \leq n$, then $p_V(z_1, ..., z_n) \neq 0$ and $p'_V(z_1, ..., z_n) \neq 0$), but it does not hold for other rotations of the right half-plane.*

To encode k-matchings of a graph G, we use multivariate (multiaffine) homogeneous polynomials, i.e. each term contains the same number of variables. Since the variables correspond to vertices here, each term representing a k-matching M contains $2k$ variables corresponding to the $2k$ vertices covered by M (alt. $n - 2k$ variables, one for each vertex not covered by M). The **multivariate homogeneous matching polynomial** is defined below. In fact, given a graph G, there is a family of such polynomials for G, one for each $1 \leq k \leq n$.

Definition 6.16. *Let $G = (V, E)$ be a graph with edge weights w_e, $e \in E$. We define the multivariate homogeneous matching polynomial by*

$$p_V^k(x_{v_1}, ..., x_{v_n}) = \sum_{M \text{ matching of size } k} \text{weight}(M) \prod_{v \in M} x_v$$

where $\text{weight}(M) = \prod_{e \in M} w_e$ and $n = |V|$.

Proposition 6.12 ([6]). *Let $G = (V, E)$ be a graph with non-negative edge weights w_e, $e \in E$. For any k, the polynomial*

$$p_V^k(x_{v_1}, ..., x_{v_n}) = \sum_{M \text{ matching of size } k} \text{weight}(M) \prod_{v \in M} x_v$$

is $\frac{1}{2}$-sector-stable.

Corollary 6.5 ([6]). *Let $G = (V, E)$ be a graph with non-negative edge weights w_e, $e \in E$. The following polynomials are $\frac{1}{2}$-sector-stable.*

$$p'^k_V(x_{v_1}, ..., x_{v_n}) = \sum_{M \text{ matching of size } k} \text{weight}(M) \prod_{v \notin M} x_v,$$

$$p_V^{2n}(x_{v_1},...,x_{v_n},x'_{v_1},...,x'_{v_n}) = \sum_{M \text{ matching}} \text{weight}(M) \prod_{v\in M} x_v \prod_{v\notin M} x'_v.$$

The former encodes k-matchings, whereas the latter is homogeneous in $2n$ variables and encodes all matchings.

6.4.3 The multivariate edge matching polynomial

Analogously, the **multivariate edge matching polynomial** is defined as follows.

Definition 6.17. *Let $G = (V, E)$ be a graph with edge weights w_e, $e \in E$. We define the multivariate edge matching polynomial by*

$$p_E(x_{e_1},\dots,x_{e_m}) = \sum_{M\in\mathcal{M}} \text{weight}(M) \prod_{e\in M} x_e$$

where $\mathcal{M}$ is the set of matchings, $\text{weight}(M) = \prod_{e\in M} w_e$*, and $m = |E|$.*

In [110], the authors proved that the polynomial just defined is same-phase stable.

Proposition 6.13 ([110])**.** *Let $G = (V, E)$ be a graph with non-negative edge weights w_e, $e \in E$. The polynomial $p_E(x_{e_1},\dots,x_{e_m})$ is same-phase stable.*

Proof. Let $G = (V, E)$ be a graph with non-negative edge weights w_{e_i}, $e_i \in E$, and let $(t_1,\dots,t_m) \in \mathbb{R}_+^m$. The univariate restriction $p_E(t_1x,\dots,t_mx)$ of the multivariate edge matching polynomial for G, is the univariate matching polynomial for a graph $G' = (V, E)$ with edge weights $w'_{e_i} = w_{e_i} \cdot t_i$. Since the latter is real-stable, by Definition 6.13, $p_E(x_{e_1},\dots,x_{e_m})$ is same-phase stable. □

Analogously to the multivariate homogeneous matching polynomial with variables corresponding to vertices, we define here the **multivariate homogeneous edge matching polynomial**. This polynomial encodes k-matchings, but each term contains exactly k variables, since now variables represent edges (instead of vertices).

Definition 6.18. *Let $G = (V, E)$ be a graph with edge weights w_e, $e \in E$. We define the multivariate homogeneous edge matching polynomial by*

$$p_E^k(x_{e_1},\dots,x_{e_m}) = \sum_{M \text{ matching of size } k} \text{weight}(M) \prod_{e\in M} x_e$$

where $\text{weight}(M) = \prod_{e\in M} w_e$ *and $m = |E|$.*

As Proposition 6.12 states the multivariate homogeneous matching polynomial p_V^k is $\frac{1}{2}$-sector-stable. We do not have an analogous result for p_E^k. In fact, this would be a strong result. The following proposition states the implications that α-sector-stability of p_E^k would have.

Proposition 6.14. *If the polynomial $p_E^k(x_{e_1}, \ldots, x_{e_m})$ is α-sector-stable for a constant α, then we have an* fpras *for counting k-matchings in general graphs.*

Proof. We can efficiently sample a k-matching u.a.r. by applying the Markov chain of the Down-Up Random Walk described in Subsection 6.3.4. In specific, the stationary distribution $\pi \propto \mu$ is the uniform distribution on k-matchings. Also step 2 can be implemented efficiently; to select S_{t+1} with probability $\propto \mu(S_{t+1})$ we look at all the $k - \ell$ sets of edges (the number of which is $\mathcal{O}(n^{k-l})$) and among them we keep the ones that will give us a matching (together with the ℓ edges in T_t). We choose one of them u.a.r. □

However, the multivariate homogeneous edge matching polynomial is $\alpha(k)$-sector-stable with $\alpha(k) = \frac{1}{2k}$ depending on k, i.e. the number of variables in each term. In fact, we prove below that every multivariate homogeneous polynomial, where each term has degree k, is $\frac{1}{2k}$-sector-stable.

Proposition 6.15. *Let $p^k \in \mathbb{R}[x_1, \ldots, x_n]$ be a multivariate homogeneous polynomial, where each term has degree k. Then p^k is $\frac{1}{2k}$-sector-stable.*

Proof. The proof is based on the following facts. The product of two complex numbers z_1 and z_2 is a complex number w that has absolute value $|w| = |z_1| \cdot |z_2|$ and argument (i.e. the angle between the positive real axis and the line joining the origin and w, represented as a point in the complex plane) $\arg(w) = \arg(z_1) + \arg(z_2)$. If $\arg(z_i) \in (-\frac{\pi}{2k}, \frac{\pi}{2k})$ for every $i \in \{1, \ldots, n\}$, then the product of k z_i's is a complex number w with $\arg(w) \in (-\frac{\pi}{2}, \frac{\pi}{2})$. This means that w belongs to the right half-plane. The sum of such complex numbers cannot be zero. □

6.4.4 The multivariate homogeneous red-black polynomial

Now we define a polynomial that encodes exact matchings using variables that correspond to vertices. We call this polynomial the **multivariate homogeneous red-black polynomial**. Intuitively, a term gives information about the number of 'red perfect matchings' that cover $2k$ specific vertices times the number of 'black perfect matchings' that cover the rest $2n - 2k$ vertices.

Let $G = (U \cup V, E)$, $|U| = |V| = n$, be an unweighted bipartite graph and $E' \subseteq E$. Let $S = \{u_1, ..., u_k, v_1, ..., v_k\}$ be a set of $2k$ vertices such that $u_1, ..., u_k \in U$ and $v_1, ..., v_k \in V$. We define the following two weights for S.

- $\mathrm{W}_{\mathrm{red}}(S)$ is defined to be the number of perfect matchings that exist in $G[S]$ (i.e. the induced subgraph of G by S) consisting only of red edges (i.e. edges in E').
- $\mathrm{W}_{\mathrm{black}}(S)$ is defined to be the number of perfect matchings in $G[\overline{S}]$ consisting only of black edges (i.e. edges in $E \setminus E'$), where $\overline{S} = (U \cup V) \setminus S$.

Definition 6.19. *Let $G = (U \cup V, E)$, $|U| = |V| = n$, be an unweighted bipartite graph and $E' \subseteq E$. We define the multivariate homogeneous red-black polynomial by*

$$p^k_{rb}(x_{u_1}, \ldots, x_{u_n}, x_{v_1}, \ldots, x_{v_n}) = \sum_{\substack{S \subseteq U \cup V \\ |S \cap V| = |S \cap U| = k}} \mathrm{W}_{\mathrm{red}}(S) \cdot \mathrm{W}_{\mathrm{black}}(\overline{S}) \cdot \prod_{v \in S} x_v.$$

Remark 6.6. *Alternatively, let R be the biadjacency matrix of $G_{red} = (U \cup V, E')$ and B be the biadjacency matrix of $G_{black} = (U \cup V, E \setminus E')$. For a set $S \subseteq U \cup V$, we denote by R_S (resp. B_S) the submatrix of R (resp. B) which contains only rows and columns that correspond to vertices in S. Then,*

$$p^k_{rb}(x_{u_1}, \ldots, x_{u_n}, x_{v_1}, \ldots, x_{v_n}) = \sum_{\substack{S \subseteq U \cup V \\ |S \cap V| = |S \cap U| = k}} \mathrm{perm}(R_S) \cdot \mathrm{perm}(B_{\overline{S}}) \prod_{v \in S} x_v.$$

Note that a term of this polynomial consists of the product of $2k$ variables corresponding to $2k$ vertices, times a coefficient that is equal to the number of exact matchings of G, in which these $2k$ vertices are covered by red edges.

We can obtain the homogeneous red-black polynomial by applying an operator on the homogeneous (vertex) matching polynomial defined in Subsection 6.4.2. Let $\langle G = (U \cup V, E), E', k \rangle$ be an instance of #Exact Matchings where G is an unweighted bipartite graph. Consider the polynomials

$$p^k_{V\,red}(x_{u_1}, \ldots, x_{u_n}, x_{v_1}, \ldots, x_{v_n}) = \sum_{\substack{S \subseteq U \cup V \\ |S \cap V| = |S \cap U| = k}} \mathrm{W}_{\mathrm{red}}(S) \prod_{v \in S} x_v$$

$$p'^k_{V\,black}(x_{u_1}, \ldots, x_{u_n}, x_{v_1}, \ldots, x_{v_n}) = \sum_{\substack{S \subseteq U \cup V \\ |S \cap V| = |S \cap U| = k}} \mathrm{W}_{\mathrm{black}}(\overline{S}) \prod_{v \in S} x_v$$

Note that the former is the matching polynomial p_V^k defined on the graph $G_{red} = (U \cup V, E')$, whereas the latter is polynomial p'^k_V defined on $G_{black} = (U \cup V, E \setminus E')$. Then the polynomial

$$p_{rb}^k(x_{u_1}, \ldots, x_{u_n}, x_{v_1}, \ldots, x_{v_n}) = \sum_{\substack{S \subseteq U \cup V \\ |S \cap V| = |S \cap U| = k}} \operatorname{perm}(R_S) \cdot \operatorname{perm}(B_{\overline{S}}) \prod_{v \in S} x_v$$

is called the Schur product (or the Hadamard-Schur product) of $p^k_{V\,red}$ and $p'^k_{V\,black}$ [87].

Recall that $\mathcal{D} = \{z \in \mathbb{C} \mid |z| < 1\}$ is the open unit disc. In [86] it was shown that $\mathcal{D}^n$-stability is preserved under the Schur product.

Theorem 6.8 ([86]). *Suppose that for each $S \subset \{1, 2, \ldots, n\}$, complex number a_S and b_S are given. Suppose that*

$$\sum_{S \subset \{1,2,\ldots,n\}} a_S \prod_{j \in S} z_j \neq 0 \quad \text{and} \quad \sum_{S \subset \{1,2,\ldots,n\}} b_S \prod_{j \in S} z_j \neq 0$$

whenever $|z_j| < 1$ for all j with $1 \leq j \leq n$. Then also

$$\sum_{S \subset \{1,2,\ldots,n\}} a_S b_S \prod_{j \in S} z_j \neq 0$$

whenever $|z_j| < 1$ for all j with $1 \leq j \leq n$.

So the following corollary holds for polynomial p_{rb}^k.

Corollary 6.6. *If both multivariate homogeneous matching polynomials $p_V^k(x_1, \ldots, x_n)$ and $p'^k_V(x_1, \ldots, x_n)$ defined on an unweighted bipartite graph are $\mathcal{D}^n$-stable, then the multivariate homogeneous red-black polynomial $p_{rb}^k(x_1, \ldots, x_n)$ defined on an unweighted bipartite graph with both black and red edges, is $\mathcal{D}^n$-stable.*

Proof. It is immediate from Theorem 6.8 and the fact that $p_{rb}^k(x_1, \ldots, x_n)$ is the Schur product of $p_V^k(x_1, \ldots, x_n)$ and $p'^k_V(x_1, \ldots, x_n)$. □

Remark 6.7. *The assumption of Corollary 6.6 is very strong. If $p_V^k(x_1, \ldots, x_n)$ is $\mathcal{D}^n$-stable, then $p'^k_V(x_1, \ldots, x_n) = x_1 \cdots x_n \cdot p_V^k(x_1^{-1}, \ldots, x_n^{-1})$ is $\overline{\mathcal{D}}^n$-stable, where $\overline{\mathcal{D}} = \{z \in \mathbb{C} \mid |z| > 1\}$. This is true because if $x \in \overline{\mathcal{D}}$, then $x \neq 0$ and $x^{-1} \in \mathcal{D}$. The converse fact is also true, that is, if $p'^k_V(x_1, \ldots, x_n)$ is $\mathcal{D}^n$-stable, then $p_V^k(x_1, \ldots, x_n)$ is $\overline{\mathcal{D}}^n$-stable. So, the assumption implies that both multivariate homogeneous matching polynomials $p_V^k(x_1, \ldots, x_n)$ and $p'^k_V(x_1, \ldots, x_n)$ are not only $\mathcal{D}^n$-stable, but also $\overline{\mathcal{D}}^n$-stable.*

6.4.5 The multivariate homogeneous exact matching polynomial

In this section, we define a polynomial that encodes exact matchings using variables corresponding to edges, namely the **multivariate homogeneous exact matching polynomial**. Since we want to focus on solving the #Exact Matchings problem on unweighted bipartite graphs, we define this polynomial for such graphs.

Let $G = (U \cup V, E)$, $|V| = |U| = n$, be an unweighted bipartite graph and $E' \subseteq E$. Let also $\mathcal{M}^k_{red}$ be the set of k-matchings of G containing only red edges.

Let $A = (a_{ij})$ be the $n \times n$ biadjacency matrix of G such that,

$$a_{ij} = \begin{cases} 1 & \text{if } (u_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

Let

- $A_{(ij)}$ denote the $(n-1) \times (n-1)$ matrix obtained from A by crossing out the i-th row and the j-th column and
- $A|^{(0)}$ denote matrix A where every a_{ij}, $(u_i, v_j) \in E'$ is set to 0.

Let M_k be a set of pairs $\{i_1, j_1\}, ..., \{i_k, j_k\}$ such that $i_1 \neq i_2 \neq ... \neq i_k$, $j_1 \neq j_2 \neq ... \neq j_k$ and $(i_1, j_1), ..., (i_k, j_k) \in E'$. We define the weight of M_k as follows

$$\mathrm{W}(M_k) = \mathrm{perm}(A_{(i_1,j_1),...,(i_k,j_k)}|^{(0)})$$

where $\mathrm{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$, for any square matrix A.

Note that edges $(i_1, j_1), ..., (i_k, j_k)$ form a matching in $\mathcal{M}^k_{red}$. The weight of this matching is the number of perfect matchings with only black edges in the remaining subgraph of G. In other words, it is the permanent of $A_{(i_1,j_1),...,(i_k,j_k)}|^{(0)}$, where $A_{(i_1,j_1),...,(i_k,j_k)}|^{(0)}$ is the biadjacency matrix of G after deleting all the vertices of M_k and deleting all red edges of G.

Definition 6.20. *Let $G = (U \cup V, E)$, $|V| = |U| = n$, be an unweighted bipartite graph, $E' \subseteq E$, and $\mathcal{M}^k_{red}$ be the set of k-matchings of G containing only edges in E'. We define the multivariate homogeneous exact matching polynomial by*

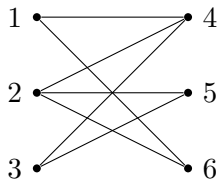
$$p^k_{em}(x_{e_1}, \ldots, x_{e_m}) = \sum_{M_k \in \mathcal{M}^k_{red}} \mathrm{W}(M_k) \cdot \prod_{e \in M_k} x_e.$$

6.4.6 Examples of expressing the matching polynomials using the permanent

The rationale behind this subsection is the following. The permanent of the biadjacency matrix of a bipartite graph gives the number of perfect matchings of the graph. We examine here how coefficients of the aforementioned polynomials, other useful quantities, or maybe an entire polynomial can be expressed using the permanent of appropriate matrices. To start with, we simplify the form of the polynomials as much as possible, so the following examples are for unweighted bipartite graphs.

Example 6.1 (Multivariate homogeneous matching polynomial)**.**

Consider the matrix $A_1 = \begin{bmatrix} z_1z_4 & 0 & z_1z_6 \\ z_2z_4 & z_2z_5 & z_2z_6 \\ z_3z_4 & z_3z_5 & 0 \end{bmatrix}$ *that corresponds to the following graph* G_1*.*



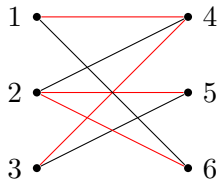
Note that:

1. *The sum of permanents of* $k \times k$ *submatrices of* A_1 *gives the multivariate homogeneous matching polynomial* $p_V^k(z_1, \ldots, z_6)$*. So, it is* $\frac{1}{2}$*-sector-stable by Proposition 6.12.*

2. $\operatorname{perm}(A_1)$ *is of the form* $C \cdot z_1 \cdots z_6$*, where* $C \in \mathbb{N}$ *is the number of perfect matchings of* G_1*. So* $\operatorname{perm}(A) = 0$ *if and only* $z_i = 0$ *for some* $1 \leq i \leq 6$*.*

Example 6.2 (Multivariate non-homogeneous red-black polynomial)**.**

Consider the matrix $A_2 = \begin{bmatrix} z_1z_4 & 1 & 0 \\ 1 & z_2z_5 & z_2z_6 \\ z_3z_4 & 1 & 1 \end{bmatrix}$ *that corresponds to the following graph* G_2*.*



Note that $\operatorname{perm}(A_2)$ *is the multivariate non-homogeneous red-black polynomial, i.e. the sum of the homogeneous red-black polynomials* $p_{rb}^k(z_1, \ldots, z_6)$*, for every* $0 \leq k \leq n$*. A term of this*

polynomial is the product of some variables times a coefficient that is equal to the number of exact matchings in which only the vertices corresponding to the variables (of the term) are covered by red edges.

Example 6.3 (Multivariate homogeneous red-black polynomial in $2n$ variables)**.**

(a) Consider the matrix $A_3 = \begin{bmatrix} z_1z_4 & z'_1z'_5 & 0 \\ z'_2z'_4 & z_2z_5 & z_2z_6 \\ z_3z_4 & z'_3z'_5 & z'_3z'_6 \end{bmatrix}$ *that corresponds to graph* G_2 *of Example 6.2.*

Note that:

1. $\mathrm{perm}(A_3)$ *gives the following homogeneous polynomial in* $12 = 2\times$(*the number of vertices of* G_2) *variables.*

$$g(z_1,\dots,z_6,z'_1,...,z'_6) = \sum_{\substack{S\subseteq U\cup V \\ |S\cap\overline{V}|=|S\cap U|}} \mathrm{W}_{\mathrm{red}}(S)\cdot \mathrm{W}_{\mathrm{black}}(\overline{S})\cdot \prod_{v\in S} z_v \cdot \prod_{v\notin S} z'_v$$

where $\mathrm{W}_{\mathrm{red}}(S)$ *and* $\mathrm{W}_{\mathrm{red}}(S)$ *are defined in Subsection 6.4.4.*

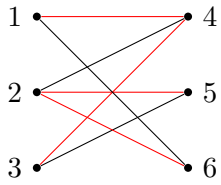
2. *If polynomial* g *is* α*-sector-stable, then by [6, Proposition 54], if we set all* z'_v *to 1, we obtain an* α*-sector-stable polynomial. In specific, this is the non-homogeneous red-black polynomial.*

(b) Consider the matrices $A_{31} = \begin{bmatrix} z_1z_4 & 0 & 0 \\ 0 & z_2z_5 & z_2z_6 \\ z_3z_4 & 0 & 0 \end{bmatrix}$ *and* $A_{32} = \begin{bmatrix} 0 & z'_1z'_5 & 0 \\ z'_2z'_4 & 0 & 0 \\ 0 & z'_3z'_5 & z'_3z'_6 \end{bmatrix}$ *that correspond to graph* G_2.

Fix a k*. Let* A_{1S} *be a* $k\times k$ *submatrix of* A_{31}*, where* S *denotes the set of rows and columns in* A_{1S}*. Then we denote by* $A_{2\overline{S}}$ *the matrix that we obtain from* A_{32} *by keeping the rows and columns in* $\overline{S}$ *(i.e. by deleting all rows and columns in* S*). Consider the polynomial that is the sum of products* $A_{1S}\cdot A_{2\overline{S}}$ *for all possible* S*. This polynomial is* $g(z_1,...,z_6,z'_1,...z'_6)$ *restricted to terms with* $2k$ *variables among* $z_1,...,z_6$ *and* $12-2k$ *variables among* $z'_1,...,z'_6$*. So, it encodes exact matchings with* k *red edges and it is a generalization of* p^k_{rb} *(i.e. if we set all* $z'_1,\dots,z'_6$ *to 1, we get* p^k_{rb}*).*

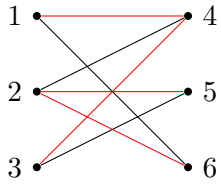
Example 6.4 (Multivariate homogeneous red-black polynomial)**.**

Consider the matrices $R = \begin{bmatrix} z_1z_4 & 0 & 0 \\ 0 & z_2z_5 & z_2z_6 \\ z_3z_4 & 0 & 0 \end{bmatrix}$ *and* $B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ *that correspond to the same graph* G_2 *given below.*



Fix a k. Consider a term to be the permanent of a $k \times k$ submatrix R_S of R times the permanent of the matrix $B_{\overline{S}}$. The sum of such terms for all possible S gives the multivariate homogeneous red-black polynomial.

Example 6.5 (Multivariate non-homogeneous exact matching polynomial)**.**
Consider the matrices $A_4 = \begin{bmatrix} e_1 & 1 & 0 \\ 1 & e_5 & e_6 \\ e_7 & 1 & 1 \end{bmatrix}$, $R' = \begin{bmatrix} e_1 & 0 & 0 \\ 0 & e_5 & e_6 \\ e_7 & 0 & 0 \end{bmatrix}$ *and* $B' = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ *that correspond to the same graph G_2 given below.*



Note that:

1. $\mathrm{perm}(A_4)$ *is the multivariate non-homogeneous exact matching polynomial, i.e. the sum of the multivariate homogeneous exact matching polynomials $p^k_{em}(z_1, \ldots, z_6)$, for every $0 \leq k \leq n$. A term corresponds to exact matchings containing the red edges appearing as variables in the term. The coefficient of the term is the number of black perfect matchings (perfect matchings containing only black edges) in the remaining subgraph.*

2. *Fix a k. Let a term be the permanent of a $k \times k$ submatrix R'_S of R' times the permanent of the matrix $B'_{\overline{S}}$. The polynomial that is the sum of all these terms, gives the homogeneous exact matching polynomial $p^k_{em}(z_1, \ldots, z_6)$.*

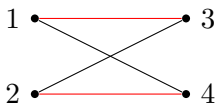
Example 6.6 (Univariate matching polynomial)**.**
This example is about bipartite graphs that the diagonal of their biadjacency matrix has only zero entries. Let $A_5 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ *be the biadjacency matrix of a bipartite graph $G_3 = (U \cup V, E)$,*

$|U| = |V| = n$, *that has only black edges. Then*

$$\text{perm}(xI + A_5) = \sum_{M \text{ matching of size } n-k} x^k$$

which is a univariate polynomial encoding all matchings of G_3. *So, it is real stable [32, Chapter 5].*

Example 6.7 (Univariate exact matching polynomial)**.**
Let $A_6 = \begin{bmatrix} x & 1 \\ 1 & x \end{bmatrix}$ *represent the graph* G_4 *which is shown below.*



Then $\text{perm}(A_6) = x^2 + 1$ *is the univariate exact matching polynomial, i.e. the polynomial obtained from the multivariate non-homogeneous exact matching polynomial by replacing all variables by* x. *In general, the permanent of the biadjacency matrix with entries equal to* x *and* 1 *for red and black edges, respectively, may be sector stable, but it is not real stable (since* i *is a root of* $x^2 + 1$*).*

6.4.7 Facts and remarks about the matching polynomials defined here

Remark 6.8. *Let* A *be a matrix with non-negative entries and zero diagonal entries. Then the permanental polynomial* $\text{perm}(xI + A)$ *of* A *is a real-stable polynomial. This is based on Example 6.6.*

Remark 6.9. *Let* $A = (a_{ij})$ *be the biadjacency matrix of the unweighted bipartite graph* $G = (U \cup V, E)$ *which also has a set of red edges* $E' \subseteq E$, *such that*

$$a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \setminus E' \\ x, & \text{if } (i,j) \in E' \\ 0, & \text{otherwise} \end{cases}$$

Then $\text{perm}(A)$ *is the univariate exact matching polynomial*

$$p_{em}(x) = \sum_{\substack{M \text{ perfect} \\ \text{matching}}} C_M \cdot x^{|E' \cap M|}$$

where $|E' \cap M|$ is the number of red edges in M and the coefficient C_M is the number of black perfect matchings in the subgraph induced by the vertices not covered by the red edges of M.
This polynomial is not real-stable. This is based on Example 6.7.

In Example 6.5 we expressed the multivariate non-homogeneous exact matching polynomial (i.e. the sum of the multivariate homogeneous exact matching polynomials p^k_{em}, $0 \leq k \leq n$) as the permanent of a relevant matrix (denoted by A_4 in Example 6.5), which contains a variable e_i, $1 \leq i \leq E'$, for every red edge, 1 for each black edge, and 0 otherwise. Below we prove that this polynomial has a zero-free region, which is a circle around the point $(0, 1)$, under some conditions on the sum of the rows and columns of this relevant matrix. The proof is based on the following theorem.

Theorem 6.9. ([32, Theorem 5.5.3]). *Let $A = (a_{ij})$ be a $n \times n$ array of n^2 complex numbers, such that*

$$\sum_{1 \leq i \leq n} |1 - a_{ij}| \leq \frac{\alpha \cdot n}{4} \quad \text{and} \quad \sum_{1 \leq j \leq n} |1 - a_{ij}| \leq \frac{\alpha \cdot n}{4}$$

for every $1 \leq j \leq n$, and $1 \leq i \leq n$, respectively, where

$$\alpha \approx 0.2784645428$$

is the positive solution of the equation

$$xe^{x+1} = 1.$$

Then $\mathrm{perm}(A) \neq 0$.

Corollary 6.7. *Let $G = (U \cup V, E)$, $|U| = |V| = n$, $E' \subseteq E$, be a bipartite graph, that has $r = |E'|$ red edges, $b = |E \setminus E'|$ black edges, and $c = n^2 - (r + b)$ pairs of vertices that are not connected by an edge. Let also $e_1, \ldots, e_r$ be an enumeration of edges in E'.*
Consider the biadjacency matrix $A = (a_{ij})$ of G such that

$$a_{ij} = \begin{cases} e_\ell, & \text{if } (i,j) = e_\ell \\ 1, & \text{if } (i,j) \in E \setminus E' \\ 0, & \text{otherwise} \end{cases}$$

Let D_r be the maximum degree of a vertex with respect to red edges (i.e. the maximum number of red edges adjacent to some vertex) and D_c be the maximum number of non-neighbors of a vertex in G. If all e_ℓ, $1 \leq \ell \leq r$, belong to the closed disc of center $(1, 0)$ and radius $\frac{\alpha n - 4D_c}{4D_r}$, then $\mathrm{perm}(A) \neq 0$, *where $\alpha \approx 0.2784645428$. The polynomial* $\mathrm{perm}(A)$ *is called the multivariate*

non-homogeneous exact matching polynomial of G.
In other words, the multivariate non-homogeneous exact matching polynomial of G is $\mathcal{C}^r$-stable, where $\mathcal{C} \subseteq \mathbb{C}$ is the circle with center $(1,0)$ and radius $\frac{\alpha n - 4D_c}{4D_r}$.

Proof. By Theorem 6.9, if the sum of $|1 - a_{ij}|$ in every row and every column is $\leq \frac{\alpha \cdot n}{4}$, then $perm(A) \neq 0$.

But the sum of a row (or a column) depends on the number of zeros (no edge) and the number of e_ℓ's (red edge) in the row. So, the sum of any row (or column) is upper bounded by $\sum_{i=1}^{D_r} |1 - e_\ell| + D_c$, which should be:

$$\sum_{i=1}^{D_r} |1 - e_\ell| + D_c \leq \frac{\alpha n}{4} \qquad \Longrightarrow$$

$$\sum_{i=1}^{D_r} |1 - e_\ell| \leq \frac{\alpha n - 4D_c}{4} \qquad \text{where } \alpha n > 4D_c.$$

It suffices to have

$$|1 - e_\ell| \leq \frac{\alpha n - 4c}{4r} \qquad \text{for every } e_\ell, 1 \leq \ell \leq r \qquad \Longrightarrow$$

$$(1 - \mathrm{Re}(e_\ell))^2 + \mathrm{Im}(e_\ell)^2 \leq (\frac{\alpha n - 4D_c}{4D_r})^2 \qquad \text{for every } e_\ell, 1 \leq \ell \leq r$$

If all e_i belong to the closed disc of centre $(1,0)$ and radius $\frac{\alpha n - 4D_c}{4D_r}$, then the multivariate non-homogeneous exact matching polynomial is not zero.

For the inequality $\alpha n > 4D_c$, we have that

$$\alpha n > 4D_c \Longleftrightarrow \alpha n > 4(n-d) \Longleftrightarrow (4-\alpha)n < 4d \Longleftrightarrow d > \frac{4-\alpha}{4} n$$

where d is the minimum degree of a vertex. □

For the sake of completeness, below we include three more propositions that have already been stated earlier in this section.

Proposition 6.16 (Proposition 6.14 restated)**.** *If the polynomial $p_E^k(x_{e_1}, \ldots, x_{e_m})$ is α-sector-stable for a constant α, then we have an* fpras *for counting k-matchings in general graphs.*

Proposition 6.17 (Proposition 6.15 restated)**.** *Let $p^k \in \mathbb{R}[x_1, \ldots, x_n]$ be a multivariate homogeneous polynomial, where each term has degree k. Then p^k is $\frac{1}{2k}$-sector-stable.*

Proposition 6.18 (Corollary 6.6 restated)**.** *If both multivariate homogeneous matching polynomials $p^k_V(x_1,\ldots,x_n)$ and $p'^k_V(x_1,\ldots,x_n)$ defined on an unweighted bipartite graph are $\mathcal{D}^n$-stable, then the multivariate homogeneous red-black polynomial $p^k_{rb}(x_1,\ldots,x_n)$ defined on an unweighted bipartite graph with both black and red edges, is $\mathcal{D}^n$-stable.*

6.5 Discussion of results

In this chapter our goal was to examine #Exact Matchings, the problem of counting perfect matchings with k edges in graphs with both black and red edges, with respect to its exact and approximate complexity.

The decision version of the problem, namely Exact Matching is known to be in RNC [119]. We prove here that Exact Matching restricted to bipartite graphs can be reduced to a max-flow problem and so it is in P.

#Exact Matchings was known to be computable in polynomial time in $K_{3,3}$-free graphs. We prove here that #Exact Matchings is reducible to #Weighted PerfMatch using polynomial interpolation, so the polynomial-time algorithm for #PerfMatch in K_5-free graphs applies to #Exact Matchings in K_5-free graphs as well.

In the case of #Exact Matchings in bipartite graphs we are left with several open questions. We include below most of them and we add another one in Chapter 7. For details, the reader can see Subsections 6.2.1 and 6.3.4.

1. In Subsection 6.2.1 we analyzed our motivation to determine whether the optimization version of #Exact Matchings in bipartite graphs is in P. A polynomial-time computable optimization version would yield a randomized algorithm described in [29], which distinguishes between #Exact Matchings(G) being polynomially or exponentially large. We have not concluded whether the optimization version of #Exact Matchings in bipartite graphs is either in P or NP-hard. However, our attempt to prove that it is in P by using the ILP formulation of the problem failed. Moreover, it is worth-noting that a modification of the algorithm in [29], based on the RNC algorithm for the decision version, and not on a polynomial-time algorithm for the optimization version, may work for #Exact Matchings. This is because the RNC algorithm for the decision version, can be extended to output the minimum exact matching, where the m edges of the graph are randomly assigned weights in $\{1,\ldots,2m\}$.

2. Can we design an fpaus for exact matchings in bipartite graphs? More precisely, can we do one of the following?

   (a) Can we modify the MCMC for sampling uniformly at random perfect matchings in bipartite graphs so it works for exact matchings with exactly k red edges?

   (b) Can we prove α-sector-stability either for the multivariate homogeneous red-black polynomial or for the multivariate homogeneous exact matching polynomial and then apply the technique of [6] (as was described in Subsection 6.3.4)?

3. Can we prove that #Exact Matchings in bipartite graphs is AP-interreducible with a problem that is not expected to be approximable, such as #BIS, or even #PerfMatch, which has an open approximability status? In fact, it would suffice to prove that it is at least as hard as one of these problems, so we would focus on the question we pose in question 1 and not on designing an fpras for the problem.

Regarding the question 2(b), we have stated some first facts and remarks about matching polynomials in Section 6.4. However, at the time this thesis is completed, we are far from proving α-sector-stability for the polynomials we mention in question 2(b).

Regarding the parameterized complexity of #Exact Matchings, we proved a hardness result. By reducing #k-Matchings to #Exact Matchings, we concluded that the latter is #W[1]-hard.

Chapter 7

Conclusion

To conclude, we state below some open questions that can set the ground for future work.

1. We have presented some TotP-complete problems under parsimonious reductions. However, these problems are not among the well-known problems in TotP, for example #IS, Permanent, etc. If any problem that admits an fpras, like Permanent, is TotP-complete under parsimonious reductions, then RP = NP. If #PerfMatch was proven to be TotP-complete, then its approximability status would be resolved, since its TotP-hardness would imply that it has no fpras unless RP = NP. In Chapter 5 we referred to an open question in relation to #MonSat; which class is #MonSat complete for, with respect to reductions under which the class is closed? Can it be that it is complete for TotP under parsimonious reductions?

2. The syntax of the logic $\mathbf{R\Sigma Q_{SOE}}$, which captures TotP over finite ordered structures, is such that only one p-bounded fixed point appears. If we allow nested p-bounded fixed points, will we obtain a logic that still captures TotP? Such questions arise quite naturally when we deal with logics with fixed points. For example, first-order logic with nested least fixed points has been proven to be the same as first-order logic with one least fixed point [112].

3. Classes defined in the context of descriptive complexity are not closed under parsimonious or product reductions in general. For example, every problem that is reducible to #MonSat under product reductions, does not necessarily belong to the class $\#\Pi_2$-1VAR defined in Chapter 5. The introduction of reductions defined in some logic (e.g. first-order reductions) between counting problems, under which the aforementioned classes would

be closed, could refine the classification of these problems in the context of descriptive complexity. Such reductions have been defined and widely used between decision problems [88].

4. We would like to give logical characterizations of classes defined in Chapter 4 and examine whether using them, we can obtain alternative proofs of well-known results, such as Toda's Theorem, closure properties of classes, etc. Note that, for example, the logical characterization of NL has given an alternative proof of the closure of NL under complement [112].

5. The most significant question that this thesis has not succeeded to answer is whether #Exact Matchings in bipartite graphs has an fpras. Can we prove α-sector-stability either for the multivariate homogeneous red-black polynomial or for the multivariate homogeneous exact matching polynomial and then apply the technique of [6] (as was described in Subsection 6.3.4 of Chapter 6)? If not, can we show hardness of approximation for the problem? In the latter case, can we prove that the optimization version of the problem is polynomial-time computable and obtain a randomized algorithm for #Exact Matchings in bipartite graphs along the lines of [29]?

6. Regarding the exact computation of #Exact Matchings, can the dichotomy result of [143] be extended to the problem of counting perfect matchings in weighted graphs? If the answer is yes, then the same dichotomy theorem holds for the problem #Exact Matchings on minor-closed graph classes.

Appendix A
Extended abstract in greek - Εκτεταμένη περίληψη στα Ελληνικά

Εισαγωγή

Με τον όρο πρόβλημα μέτρησης αναφερόμαστε σε μια συνάρτηση που αντιστοιχίζει μια είσοδο σε ένα πρόβλημα απόφασης στο πλήθος των λύσεων που έχει το πρόβλημα για αυτή την είσοδο. Για παράδειγμα, το #Sat είναι μια συνάρτηση που αντιστοιχίζει έναν προτασιακό τύπο στο πλήθος των αναθέσεων αληθοτιμών για τις οποίες ο τύπος ικανοποιείται. Η κλάση τέτοιων συναρτήσεων, για τις οποίες το αντίστοιχο πρόβλημα απόφασης βρίσκεται στην κλάση NP, είναι η κλάση #P, που όρισε ο Valiant [149]. Ισοδύναμα, η #P είναι η κλάση των συναρτήσεων που μετρούν τα μονοπάτια αποδοχής μη-ντετερμινιστικών μηχανών Turing πολυωνυμικού χρόνου, για τις οποίες θα χρησιμοποιούμε το συμβολισμό NPTMs.

Είναι ενδιαφέρον ότι πολλά προβλήματα από διαφορετικά επιστημονικά πεδία μπορούν να εκφραστούν ως μετρητικά. Αναφέρουμε ενδεικτικά τα παρακάτω.

1. Η συνάρτηση διαμέρισης στη στατιστική φυσική [81, 90, 91, 115].
2. Ο υπολογισμός του όγκου ενός κυρτού πολύτοπου στην υπολογιστική γεωμετρία [58].
3. Ο υπολογισμός της permanent στη γραμμική άλγεβρα [149].
4. Υπάρχουν προβλήματα βελτιστοποίησης με αβεβαιότητα που απαιτούν τη μέτρηση του πλήθους των λύσεων. Για την ακρίβεια απαιτείται η μέτρηση λύσεων που είναι προσεγιστικά βέλτιστες (και όχι απαραίτητα βέλτιστες λύσεις) [129, 117].

Πολύ λίγα προβλήματα μέτρησης μπορούν να επιλυθούν ακριβώς σε πολυωνυμικό χρόνο (π.χ. το πρόβλημα #2Col). Για αυτό το λόγο μας ενδιαφέρει να εξετάσουμε ποια μετρητικά προβλήματα

μπορούν να προσεγγιστούν. Τα μετρητικά προβλήματα που μπορούν να προσεγγιστούν με τη χρήση ενός πλήρως πολυωνυμικού πιθανοτικού προσεγγιστικού σχήματος (fpras), είναι προβλήματα με εύκολο πρόβλημα απόφασης (για την ακρίβεια πρόβλημα απόφασης στην κλάση BPP). Στην αναζήτηση προσεγγίσιμων μετρητικών προβλημάτων, η κλάση #PE, η οποία περιέχει τα προβλήματα στη #P, τα οποία έχουν αντίστοιχο πρόβλημα απόφασης στην P, είναι κεντρικής σημασίας. Σε αυτή τη διατριβή, εστιάζουμε σε μια υποκλάση της #PE, την κλάση TotP. Σχεδόν όλα τα γνωστά προβλήματα μέτρησης που επιδέχονται fpras ανήκουν στην TotP και οι κλάσεις που αποτελούνται μόνο από τέτοια προβλήματα περιέχονται στην TotP.

Η TotP είναι η κλάση των συναρτήσεων που μετρούν το συνολικό αριθμό των μονοπατιών των NPTMs. Εκτός από αυτόν τον απλό δομικό χαρακτηρισμό, η TotP έχει έναν ενδιαφέροντα εναλλακτικό ορισμό. Είναι η κλάση όλων των αυτοαναγώγιμων μετρητικών προβλημάτων με πρόβλημα απόφασης στην P, η οποία επίσης είναι κλειστή ως προς φειδωλές αναγωγές [123]. Η TotP είναι μια εύρωστη κλάση, όπου χρησιμοποιούμε τον όρο 'εύρωστη κλάση' όπως ορίστηκε από τους συγγραφείς του άρθρου [15]: μια κλάση θεωρείται εύρωστη αν είτε είναι κλειστή ως προς την πρόσθεση, τον πολλαπλασιασμό και την αφαίρεση κατά ένα, είτε έχει φυσικά πλήρη προβλήματα [15]. Συγκεκριμένα, η TotP ικανοποιεί και τις δύο αυτές συνθήκες όπως θα δείξουμε στην παρούσα διατριβή.

Η TotP έχει επίσης έναν χαρακτηρισμό μέσω συναρτήσεων που υπολογίζουν το μέγεθος διαστημάτων [84, 27]. Η TotP είναι η κλάση των συναρτήσεων μέτρησης του μεγέθους διαστήματος, το οποίο ορίζεται μέσω κάποιας πολυωνυμικής ολικής διάταξης A για την οποία (1) υπάρχει πολυωνυμικού χρόνου αλγόριθμος που δεδομένης εισόδου υπολογίζει το κάτω και άνω όριο του διαστήματος, (2) υπάρχει πολυωνυμικού χρόνου συνάρτηση LN_A που δεδομένου στοιχείου, επιστρέφει το λεξικογραφικά πλησιέστερο σε αυτό που ανήκει σε κάποιο δεδομένο διάστημα.

Σε αυτή τη διατριβή, θα παρουσιάσουμε αποτελέσματα δομικής και περιγραφικής πολυπλοκότητας για την κλάση TotP.

Δίνουμε τους ορισμούς των κλάσεων #P, #PE και TotP, κάποιες προτάσεις που ισχύουν για τις μεταξύ τους σχέσεις, παραδείγματα προβλημάτων που ανήκουν στην TotP και χρήσιμες ιδιότητες αυτής της κλάσης.

Ορισμός 7.1. *(α)* ([149]). $\#\mathsf{P} = \{acc_M : \Sigma^* \to \mathbb{N} \mid M \text{ είναι μία } NPTM\}$, όπου $acc_M(x) = \#(\text{μονοπάτια αποδοχής της } M \text{ με είσοδο } x)$.

(β) ([122]). $\#\mathsf{PE} = \{f : \Sigma^* \to \mathbb{N} \mid f \in \#\mathsf{P} \text{ και } L_f \in \mathsf{P}\}$, όπου $L_f = \{x \mid f(x) > 0\}$.

(γ) ([102]). $\mathsf{TotP} = \{tot_M : \Sigma^* \to \mathbb{N} \mid M \text{ είναι μία } NPTM\}$, όπου

$tot_M(x) = \#(\text{όλα τα μονοπάτια της } M \text{ με είσοδο } x) - 1$.

Χρησιμοποιούμε τα παρακάτω είδη αναγωγών μεταξύ μετρητικών συναρτήσεων. Η κλάση FP είναι η κλάση των συναρτήσεων που μπορούν να υπολογιστούν σε πολυωνυμικό χρόνο.

Turing αναγωγές. Η f ανάγεται στη g ως προς Turing αναγωγές, το οποίο συμβολίζουμε $f \leq_{\mathrm{T}}^{\mathrm{p}} g$, αν ισχύει $f \in \mathsf{FP}^g$. Αν η αναγωγή χρησιμοποιεί μόνο μία κλήση στο μαντείο, συμβ. $f \in \mathsf{FP}^{g[1]}$, τότε γράφουμε $f \leq_{\text{1-T}}^{\mathrm{p}} g$.

Φειδωλές αναγωγές. Η f ανάγεται στη g ως προς φειδωλές αναγωγές, το οποίο συμβολίζουμε $f \leq_{\mathrm{pars}}^{\mathrm{p}} g$, αν υπάρχει συνάρτηση $h \in \mathsf{FP}$, τέτοια ώστε για κάθε $x \in \Sigma^*$, ισχύει $f(x) = g(h(x))$.

Αναγωγές γινομένου. Η f ανάγεται στη g ως προς αναγωγές γινομένου, το οποίο συμβολίζουμε $f \leq_{\mathrm{pr}} g$, αν υπάρχουν $h_1, h_2 \in \mathsf{FP}$ τέτοιες ώστε για κάθε $x \in \Sigma^*$ ισχύει $f(x) = g(h_1(x)) \cdot h_2(x)$.

Αναγωγές που διατηρούν την προσεγγισιμότητα (AP αναγωγές). Η f ανάγεται στη g ως προς αναγωγές που διατηρούν την προσεγγισιμότητα, το οποίο συμβολίζουμε $f \leq_{\mathrm{AP}} g$, αν υπάρχει πιθανοτική μηχανή Turing M που καλεί ένα μαντείο, η οποία παίρνει ως είσοδο ένα στιγμιότυπο x της f και $0 < \varepsilon < 1$ και ικανοποιεί τις παρακάτω τρεις συνθήκες:

1. για κάθε κλήση του μαντείου η είσοδος είναι της μορφής (w, δ), όπου w είναι στιγμιότυπο της g, και $0 < \delta < 1$ είναι μια σταθερά σφάλματος τέτοια ώστε $\delta^{-1} \leq poly(|x|, \varepsilon^{-1})$,
2. αν το μαντείο είναι ένα πιθανοτικό προσεγγιστικό σχήμα για την g, τότε η M είναι ένα πιθανοτικό προσεγγιστικό σχήμα για την f.
3. η M είναι πολυωνυμικού χρόνου ως προς $|x|$ και ε^{-1}.

Αν για δύο συναρτήσεις f, g ισχύει $f \leq g$ και $g \leq f$ για κάποια αναγωγή $\leq$, τότε λέμε ότι οι f και g είναι ισοδύναμες ως προς τη $\leq$.

Διαισθητικά, ένα μετρητικό πρόβλημα είναι αυτοαναγώγιμο αν η τιμή του μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο δεδομένων των τιμών του ίδιου προβλήματος για (πολυωνυμικά πολλές) μικρότερες εισόδους.

Θεώρημα 7.1 ([123]). *(α)* $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$. *Οι εγκλεισμοί είναι αυστηροί εκτός αν* $\mathsf{P} = \mathsf{NP}$.
(β) $\mathsf{FP}^{\mathsf{TotP}[1]} = \mathsf{FP}^{\#\mathsf{PE}[1]} = \mathsf{FP}^{\#\mathsf{P}[1]}$.
(γ) Η TotP *είναι η κλάση των αυτοαναγώγιμων* #PE *συναρτήσεων, η οποία, επιπλέον, είναι κλειστή ως προς φειδωλές αναγωγές.*

Τα παρακάτω προβλήματα είναι αυτοαναγώγιμα και ανήκουν στη #PE, άρα είναι προβλήματα της κλάσης TotP.

1. #DNF: το πρόβλημα μέτρησης των αναθέσεων αληθοτιμών που ικανοποιούν έναν προτασιακό τύπο σε κανονική διαζευκτική μορφή.

2. #2Sat: το πρόβλημα μέτρησης των αναθέσεων αληθοτιμών που ικανοποιούν έναν προτασιακό τύπο σε κανονική συζευκτική μορφή όπου κάθε απλή πρόταση περιέχει δύο λεκτικά.

3. #HornSat: το πρόβλημα μέτρησης των αναθέσεων αληθοτιμών που ικανοποιούν έναν προτασιακό τύπο σε κανονική συζευκτική μορφή όπου κάθε απλή πρόταση περιέχει το πολύ ένα θετικό λεκτικό.

4. #MonSat: το πρόβλημα μέτρησης των αναθέσεων αληθοτιμών που ικανοποιούν έναν προτασιακό τύπο σε κανονική συζευκτική μορφή όπου κάθε απλή πρόταση περιέχει μόνο θετικά λεκτικά.

5. #BiPerfMatch: το πρόβλημα μέτρησης των τέλειων ταιριασμάτων ενός διμερούς γράφου.

6. #PerfMatch: το πρόβλημα μέτρησης των τέλειων ταιριασμάτων ενός γράφου.

Γενικά για έναν προτασιακό τύπο που είναι σε κανονική συζευτκική μορφή θα χρησιμοποιούμε την αγγλική συντομογραφία CNF (2CNF, 3CNF στην περίπτωση που θέλουμε να διευκρινίσουμε ότι κάθε απλή πρόταση περιέχει 2 ή 3 λεκτικά αντίστοιχα).

TotP-πλήρη προβλήματα

Η #P είναι ισοδύναμη με την TotP ως προς Turing αναγωγές, αλλά ακόμα και με μια υποκλάση της TotP, την SpanL [9], που περιέχει μόνο προσεγγίσιμα προβλήματα. Αυτό σημαίνει ότι όλα τα προβλήματα που είναι #P-πλήρη ως προς Turing αναγωγές, είναι επίσης TotP-πλήρη ως προς Turing αναγωγές. Για παράδειγμα, όλα τα προβλήματα 1-6 που αναφέρονται παραπάνω είναι TotP-πλήρη ως προς Turing αναγωγές. Αυτή η παρατήρηση διακαολογεί τον ισχυρισμό ότι οι Turing αναγωγές θολώνουν τις δομικές διαφορές μεταξύ των μετρητικών κλάσεων [103].

Αντίθετα, χρησιμοποιώντας φειδωλές αναγωγές μπορούμε να διακρίνουμε διαφορές μεταξύ μετρητικών κλάσεων και των δύσκολων προβλημάτων τους. Ένα βασικό χαρακτηριστικό αυτών των αναγωγών είναι ότι διατηρώντας το πλήθος των λύσεων, διατηρούν την ύπαρξη fpras. Αν $f \leq^{\mathrm{P}}_{\mathrm{pars}} g$ και η g επιδέχεται fpras, τότε η f επιδέχεται fpras.

TotP-πλήρη προβλήματα προς φειδωλές αναγωγές

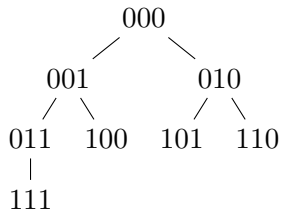
Σε αυτή τη διατριβή αποδεικνύουμε ότι τα παρακάτω προβλήματα είναι TotP-πλήρη ως προς φειδωλές αναγωγές.

Ορίζουμε τη μερική διάταξη $\leq_{tree}$ πάνω στο σύνολο $\mathbb{N}$ να είναι η ανακλαστική, μεταβατική και αντισυμμετρική σχέση για την οποία ισχύει: αν $y = 2x + 1$ ή $y = 2x + 2$ τότε $x \leq_{tree} y$.

Πρόβλημα 1. #Tree-Monotone-Circuit-Sat.
Είσοδος: Ένα λογικό κύκλωμα C_n, μη-αύξον ως προς $\leq_{tree}$.
Έξοδος: #Tree-Monotone-Circuit-Sat$(C_n) := |\{y \in \{0,1\}^n : C_n(y) = 1\}|$.

Έστω $(U, \leq)$ ένα μερικά διατεταγμένο ζεύγος. Το $V \subseteq U$ λέγεται *κλειστό από κάτω* αν για κάθε $y, x \in U$, ισχύει $(y \in V$ και $x < y) \Rightarrow x \in V$. Έστω C_n ένα λογικό κύκλωμα με n πύλες εισόδου. Το $V \subseteq \{0,1\}^n$ λέγεται C_n*-αποδεκτό* αν για κάθε $x \in V$, $C_n(x) = 1$.

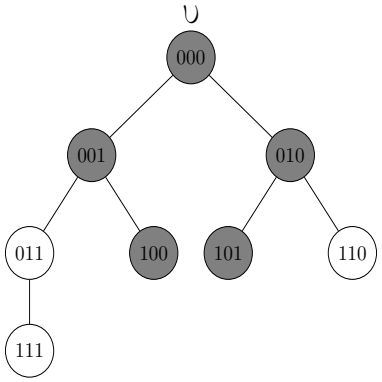
Πρόβλημα 2. Max-Lower-Set-Size.
Είσοδος: Ένα λογικό κύκλωμα C_n με n πύλες εισόδου.
Έξοδος: Το μέγεθος του μέγιστου C_n-αποδεκτού συνόλου που είναι κλειστό από κάτω ως προς τη μερική διάταξη $\leq_{tree}$.



Αν περιορίσουμε τη διάταξη $\leq_{tree}$ στο σύνολο $\mathbb{N}_{k-1} = \{0, 1, \ldots, 2^k - 1\}$ και επιπλέον χρησιμοποιήσουμε τις δυαδικές αναπαραστάσεις των στοιχείων του $\mathbb{N}_{k-1}$, παίρνουμε μια μερική διάταξη πάνω στο $\{0,1\}^k$, την οποία επίσης θα συμβολίζουμε $\leq_{tree}$. Ονομάζουμε T_k το δυαδικό δέντρο που αναπαριστά την $\leq_{tree}$ στο $\{0,1\}^k$.

Για παράδειγμα, το πλήρες δυαδικό δέντρο T_3 που απεικονίζεται παραπάνω αναπαριστά τη διάταξη $\leq_{tree}$ στο $\{0,1\}^3$.

Πρόβλημα 3. Size-of-Subtree.
Είσοδος: Ένα κατηγόρημα $A : T_k \to \{0,1\}$, το οποίο είναι υπολογίσιμο σε πολυωνυμικό χρόνο και μια κορυφή u του T_k.
Έξοδος: Το μέγεθος του μέγιστου υποδέντρου $S \subseteq A^{-1}(1)$ με ρίζα u.



Για παράδειγμα, στο διπλανό σχήμα δίνεται ένα στιγμιότυπο του Size-of-Subtree. Ισχύει ότι $u = 000$, $k = 3$ και το κατηγόρημα A παίρνει την τιμή 1 για τις γκρι κορυφές. Η έξοδος του προβλήματος είναι 5.

Παρακάτω δίνουμε τον όρισμο για έναν *k-clustered-monotone* τύπο. Κρατάμε τον αγγλικό όρο εδώ γιατί το όνομα του επόμενου προβλήματος #Clustered-Monotone-Sat προέρχεται από αυτόν.

1. Έστω ϕ ένας **3-CNF** τύπος και $k \in \mathbb{N}$. Ορίζουμε τη συνάρτηση $f_\phi^k : \{0,1\}^k \to \mathbb{N}$ τέτοια ώστε $f_\phi^k(a) = \#(\text{αναθέσεις αληθοτιμών που ικανοποιούν τη } \phi \text{ και έχουν πρόθεμα } a)$ για κάποιο $a \in \{0,1\}^k$.

2. Ένας **3-CNF** τύπος ϕ με n μεταβλητές λέγεται *k-clustered-monotone* για κάποιο $k \leq n$, αν για κάθε $a, b \in \{0,1\}^k$ τέτοια ώστε $a \leq_{tree} b$, αν $f_\phi^k(a) = 0$ τότε $f_\phi^k(b) = 0$.

Πρόβλημα 4. #Clustered-Monotone-Sat.
Είσοδος: $y = (\phi, k, M)$, όπου ο ϕ είναι k-clustered monotone, και M είναι η περιγραφή μιας συνάρτησης για την οποία ισχύει $M \in \mathsf{FP}$ και $M(a, \phi) = f_\phi^k$.
Έξοδος: #Clustered-Monotone-Sat$(y) := \#(\text{αναθέσεις αληθοτιμών που ικανοποιούν τη } \phi)$.

Συνέπειες

1. Ένας απλός αλγόριθμος επίλυσης TotP-δύσκολων προβλημάτων

Το πρόβλημα Size-of-Subtree είχε οριστεί καταρχάς από τον Knuth [105] ως το πρόβλημα υπολογισμού του μεγέθους του δέντρου ενός backtracking προγράμματος. Στο ίδιο άρθρο ο Knuth πρότεινε έναν απλό αποδοτικό πιθανοτικό προσεγγιστικό αλγόριθμο, ο οποίος ακολουθεί τον παρακάτω αναδρομικό υπολογισμό. Ξεκινώντας από την κορυφή u, υπολογίζει το πλήθος των παιδιών της u για τα οποία το κατηγόρημα A παίρνει την τιμή 1.

1. Αν το A δεν παίρνει την τιμή 1 για κανένα από τα παιδιά της u, σταματάει.

2. Αν για ένα παιδί της u το A παίρνει την τιμή 1, συνεχίζει αναδρομικά με αυτό το παιδί.

3. Αν το A παίρνει την τιμή 1 και για τα δύο παιδιά της, επιλέγει ένα από αυτά με πιθανότητα 1/2 και συνεχίζει αναδρομικά.

Ο αλγόριθμος θεωρεί ότι κάθε κορυφή είναι αντιπροσωπευτική για το επίπεδό της, δηλ. ότι όλες οι κορυφές που ανήκουν στο ίδιο επίπεδο με αυτή έχουν το ίδιο πλήθος παιδιών για τα οποία το κατηγόρημα A παίρνει την τιμή 1. Μέ βάση αυτή την υπόθεση, δίνει μια εκτίμηση για το ζητούμενο, δηλ. το μέγεθος του μέγιστου υποδέντρου με ρίζα u που περιέχει μόνο κορυφές για τις οποίες το A παίρνει την τιμή 1. Η αναμενόμενη τιμή του αλγορίθμου είναι η τιμή που θέλουμε να υπολογίσουμε. Η διακύμανση μπορεί να είναι εκθετική στη χειρότερη περίπτωση.

2. Εκθετικού χρόνου κάτω φράγματα για το πρόβλημα Size-of-Subtree

Θα δείξουμε κάτω φράγματα για τον υπολογισμό του Size-of-Subtree, όταν κάποια από τις παρακάτω υποθέσεις ισχύει.

- Υπόθεση εκθετικού χρόνου (ETH): Δεν υπάρχει ντετερμινιστικός αλγόριθμος που να αποφασίζει το 3Sat σε χρόνο $\exp(o(n))$.
- Πιθανοτική υπόθεση εκθετικού χρόνου (rETH): Δεν υπάρχει πιθανοτικός αλγόριθμος που να αποφασίζει το 3Sat σε χρόνο $\exp(o(n))$, με πιθανότητα σφάλματος το πολύ 1/3.
- Μετρητική υπόθεση εκθετικού χρόνου (#ETH): Δεν υπάρχει ντετερμινιστικός αλγόριθμος που να υπολογίζει ακριβώς το #Sat σε χρόνο $\exp(o(n))$.

Παρακάτω, το N συμβολίζει το ύψος του πλήρως δυαδικού δέντρου που είναι μέρος ενός στιγμιοτύπου του Size-of-Subtree.

Θεώρημα 7.2. *(α΄) Αν ισχύει η rETH, δεν υπάρχει πιθανοτικός αλγόριθμος που να υπολογίζει το* Size-of-Subtree *ακριβώς σε χρόνο* $\exp(o(N))$.

(β΄) Αν ισχύει η #ETH δεν υπάρχει ντετερμινιστικός αλγόριθμος που να υπολογίζει το Size-of-Subtree *ακριβώς σε χρόνο* $\exp(o(N))$.

(γ΄) Αν ισχύει η rETH δεν υπάρχει πιθανοτικός αλγόριθμος που να προσεγγίζει το Size-of-Subtree *με πολλαπλασιαστικό παράγοντα* $(1 \pm \frac{1}{4})$ *σε χρόνο* $\exp(o(N))$.

3. Το #Sat είναι ισοδύναμο με το #Clustered-Monotone-Sat ως προς AP αναγωγές

Κάθε TotP-πλήρες πρόβλημα ως προς φειδωλές αναγωγές είναι ισοδύναμο με το #Sat ως προς αναγωγές που διατηρούν την προσεγγισιμότητα. Γνωρίζουμε ήδη ότι τα προβλήματα #2Sat και #MonSat έχουν επίσης αυτή τη σχέση με το #Sat. Αυτό όμως που κερδίσαμε εδώ είναι το εξής: Το #Sat είναι AP-ισοδύναμο με το #Clustered-Monotone-Sat, δηλ. ένα πρόβλημα μέτρησης αναθέσεων αληθοτιμών που ικανοποιούν έναν τύπο, το οποίο έχει τις παρακάτω ιδιότητες:

1. μπορούμε να αποφασίσουμε σε πολυωνυμικό χρόνο αν ένας δεδομένος τύπος είναι ικανοποιήσιμος,
2. για κάθε λύση υπάρχει κάποια γειτονιά αναθέσεων αληθοτιμών, για τις οποίες μπορεί να αποφασιστεί αποδοτικά αν περιέχουν κάποια λύση.

Σχέση μεταξύ των κλάσεων TotP και FPRAS

Σε αυτή την ενότητα, εξετάζουμε υπό ποιες συνθήκες ισχύει TotP $\subseteq$ FPRAS, καθώς και ο αντίστροφος εγκλεισμός FPRAS $\subseteq$ TotP.

Καταρχάς ισχύει το παρακάτω θεώρημα για την κλάση #P.

Θεώρημα 7.3. #P $\subseteq$ FPRAS *αν και μόνο αν* RP = NP.

Το παραπάνω θεώρημα μπορεί να επεκταθεί για την TotP. Επίσης, κάθε συνάρτηση $f \in$ TotP η οποία επιδέχεται fpras, ανήκει στην κλάση FPRAS′, δηλ. μπορεί να επιλυθεί από fpras που δεν κάνει ποτέ λάθος στην περίπτωση που ισχύει $f(x) = 0$. Ισχύει λοιπόν το παρακάτω θεώρημα.

Θεώρημα 7.4. TotP $\subseteq$ FPRAS *αν και μόνο αν* TotP $\subseteq$ FPRAS′ *αν και μόνο αν* RP = NP.

Ενώ για τον αντίστροφο εγκλεισμό έχουμε το εξής αποτέλεσμα.

Θεώρημα 7.5. *Αν* FPRAS $\subseteq$ TotP *τότε* P = RP.

Η απόδειξη του τελευταίου θεωρήματος υπήρξε αφορμή για τον ορισμό δύο κλάσεων, των #RP$_1$ και #RP$_2$, που περιέχουν προβλήματα μέτρησης με αντίστοιχο πρόβλημα απόφασης στην κλάση RP.

Έστω M μία NPTM σε κανονική μορφή, δηλ. για είσοδο x, η M έχει $2^{p(|x|)}$ μονοπάτια για κάποιο πολυώνυμο p, ή αλλιώς η M κάνει $p(|x|)$ μη-ντετερμινιστικές επιλογές. Ορίζουμε το

σύνολο $\mathcal{MR} = \{M \mid M$ είναι NPTM σε κανονική μορφή και για κάθε $x \in \Sigma^*$ είτε $acc_M(x) = 0$ είτε $acc_M(x) > \frac{1}{2} \cdot 2^{p(|x|)}\}$.

Ορισμός 7.2. $\#\mathsf{RP}_1 = \{f \in \#\mathsf{P} \mid$ *υπάρχει* $M \in \mathcal{MR}$ *τέτοια ώστε για κάθε* $x \in \Sigma^*$, $f(x) = acc_M(x)\}$.

Ορισμός 7.3. $\#\mathsf{RP}_2 = \{f \in \#\mathsf{P} \mid L_f \in \mathsf{RP}\}$, *όπου* $L_f = \{x \mid f(x) > 0\}$.

Τα παρακάτω προβλήματα ανήκουν στην κλάση $\#\mathsf{RP}_1$. Το πρώτο πρόβλημα σχετίζεται με το πρόβλημα Polynomial Identity Testing, στο οποίο δεδομένου ενός πολυωνύμου μας ενδιαφέρει αν είναι ταυτοτικά ίσο με το μηδενικό πολυώνυμο. Είναι γνωστό ότι υπάρχει αποδοτικός πιθανοτικός αλγόριθμος για αυτό το πρόβλημα, αλλά αν αποδειχθεί ότι επιλύεται σε ντετερμινιστικό πολυωνυμικό χρόνο, τότε θα αποκτήσουμε ένα νέο κάτω φράγμα για την κλάση NEXP σε σχέση με την επίλυση των προβλημάτων της από κυκλώματα πολυωνυμικού μεγέθους [95].

#NonZerosForPIT.
Είσοδος: Ένα πολυώνυμο $p(x_1, ..., x_n)$ βαθμού d πάνω σε ένα σώμα $\mathbb{F}$, τέτοιο ώστε $|\mathbb{F}| \geq 3d$.
Έξοδος: Ο αριθμός των σημείων $(y_1, ..., y_n) \in \mathbb{F}_{3d}^n$ για τα οποία ισχύει $p(y_1, .., y_n) \neq 0$.

Το σύνολο $\mathbb{F}_{3d}$ είναι ένα σύνολο $3d$ στοιχείων του $\mathbb{F}$ που μπορούν να επιλεγούν ντετεριμινιστικά από μία μηχανή Turing.

Το δεύτερο πρόβλημα είναι μια μετρητική εκδοχή του προβλήματος απόφασης αν ένας φυσικός αριθμός είναι σύνθετος. Δεδομένου $n > 2$, ο n μπορεί να γραφεί ως $2^s \cdot d + 1$, όπου $s, d \in \mathbb{N}^+$ και d είναι περιττός. Ένας φυσικός αριθμός $0 < a < n$ ονομάζεται *πιστοποιητικό για τη συνθετότητα του* n, αν ισχύει το (α) ή το (β):
(α) $a^{n-1} \not\equiv 1 \pmod{n}$,
(β) $a^{n-1} \equiv 1 \pmod{n}$ και ισχύουν οι δύο επόμενες σχέσεις:
$a^d \not\equiv 1 \pmod{n}$ και $a^{2^r \cdot d} \not\equiv -1 \pmod{n}$, για όλα τα $0 \leqslant r < s$.

Τα δύο επόμενα προβλήματα είναι οι μετρητικές εκδοχές των προβλημάτων Exact Matching [125] και Blue-Red Matching [120], τα οποία ανήκουν στην κλάση RP [119, 120], αλλά δε γνωρίζουμε αν ανήκουν στην P.

#Exact Matchings.
Είσοδος: Ένας γράφος $G = (V, E)$, ένα υποσύνολο των ακμών $E' \subseteq E$ και ένας ακέραιος k.
Έξοδος: Το πλήθος των τέλειων ταιριασμάτων του G που περιέχει ακριβώς k ακμές από το E'.

#Blue-Red Matchings.
Είσοδος: Ένας γράφος $G = (V, E_{red} \cup E_{blue})$, και δύο ακέραιοι w και B.

Έξοδος: Το πλήθος των ταιριασμάτων μεγέθους τουλάχιστον B που περιέχουν το πολύ w ακμές από το E_{blue} και το πολύ w ακμές από το E_{red}.

Η ισχύς που προκύπτει από τη μέτρηση του πλήθους όλων των μονοπατιών

Ορίζουμε την κλάση $\mathsf{Gap_{tot}P}$ ως την κλειστότητα της TotP ως προς αφαίρεση. Με βάση ιδιότητες συναρτήσεων της TotP και της $\mathsf{Gap_{tot}P}$, ορίζουμε τις παρακάτω κλάσεις προβλημάτων απόφασης.

Κλάση	Συνάρτηση f στην κλάση:	Αν $x \in L$:	Αν $x \notin L$:
$\mathsf{U_{tot}P}$	TotP	$f(x) = 1$	$f(x) = 0$
$\mathsf{Few_{tot}P}$	TotP	$f(x) \leq p(\|x\|)$ για κάποιο πολυώνυμο p και $f(x) > 0$	$f(x) = 0$
$\oplus_{\mathsf{tot}}\mathsf{P}$	TotP	$f(x)$ είναι περιττός	$f(x)$ είναι άρτιος
$\mathsf{Mod_{k\,tot}P}$	TotP	$f(x) \not\equiv 0 \pmod{k}$	$f(x) \equiv 0 \pmod{k}$
$\mathsf{SP_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = 1$	$f(x) = 0$
$\mathsf{WP_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = g(x)$ για κάποια $g \in$ FP με $0 \notin \mathrm{range}(g)$	$f(x) = 0$
$\mathsf{C_{=tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) = 0$	$f(x) \neq 0$ [εναλλ. ορισμός: $f(x) > 0$]
$\mathsf{P_{tot}P}$	$\mathsf{Gap_{tot}P}$	$f(x) > 0$	$f(x) \leq 0$ [εναλλ. ορισμός: $f(x) < 0$]

Αποδεικνύεται ότι η κλάση $\mathsf{Gap_{tot}P}$ ταυτίζεται με τη GapP, δηλ. την κλειστότα της #P ως προς αφαίρεση. Επίσης, εκτός από τις κλάσεις $\mathsf{U_{tot}P}$ και $\mathsf{Few_{tot}P}$ που είναι ίσες με την κλάση P, αποδείξαμε ότι κάθε μία από τις υπόλοιπες είναι ίση με την αντίστοιχη κλάση που ορίζεται μέσω κάποιας #P ή GapP συνάρτησης, όπως φαίνεται παρακάτω.

$\mathsf{U_{tot}P} = \mathsf{P}$
$\mathsf{Few_{tot}P} = \mathsf{P}$
$\oplus_{\mathsf{tot}}\mathsf{P} = \oplus\mathsf{P}$
$\mathsf{Mod_{k\,tot}P} = \mathsf{Mod_kP}$
$\mathsf{SP_{tot}P} = \mathsf{SPP}$
$\mathsf{WP_{tot}P} = \mathsf{WPP}$
$\mathsf{C_{=tot}P} = \mathsf{C_{=}P}$
$\mathsf{P_{tot}P} = \mathsf{PP}$

Για τις κλάσεις $\oplus\mathsf{P}$ και $\mathsf{C_{=}P}$ αυτό ήταν αναμενόμενο, επειδή ήταν ήδη γνωστά πλήρη προν=βλήματα για τις δύο αυτές κλάσεις που ορίζονται από κάποια TotP συνάρτηση. Πιο συγκεκριμένα, τα προβλήματα αυτά ήταν το $\oplus$Pl-Rtw-Mon-3CNF [150] και το DiffPerfMatch$_{=0}$ [52].

Πλήρη προβλήματα για τις κλάσεις $\oplus\mathsf{P}$, $\mathsf{Mod_kP}$, SPP, WPP, $\mathsf{C_{=}P}$ και PP

- Για κάθεμία από τις κλάσεις $\oplus\mathsf{P}$, $\mathsf{Mod_kP}$, SPP, WPP, $\mathsf{C_{=}P}$ ανδ PP αποκτήσαμε μια οικογένεια πλήρων προβλημάτων, τα οποία ορίζονται μέσω κάποιου TotP-πλήρους προβλήματος ως προς φειδωλές αναγωγές και όχι μέσω κάποιου #P-πλήρους (ή NP-πλήρους).

- Ακολουθώντας το αποτέλεσμα του Curticapean [52] για την WPP για την κλάση $\mathsf{C_{=}P}$, δείξαμε ότι οι κλάσεις WPP και PP έχουν πλήρη προβλήματα, που ορίζονται μέσω της TotP συνάρτησης #PerfMatch, τα οποία ονομάσαμε DiffPerfMatch$_{=g}$ και DiffPerfMatch$_{>0}$ αντίστοιχα.

- Το πρόβλημα DiffPerfMatch$_{=g}$ είναι το πρόβλημα του να αποφασίσουμε αν η διαφορά της συνάρτησης #PerfMatch σε δύο δεδομένους γράφους είναι ίση με 0 ή με μία πολυωνυμικά υπολογίσιμη τιμή, όταν έχουμε την υπόσχεση ότι ένα από τα δύο ισχύει. Αποδείξαμε ότι αν ισχύει η πιθανοτική υπόθεση εκθετικού χρόνου (rETH), τότε το DiffPerfMatch$_{=g}$ δεν μπορεί να αποφασιστεί από κάποιον υποεκθετικό αλγόριθμο.

- Το πλήρες πρόβλημα DiffSat$_{=1}$ για την SPP ανάγεται στο να αποφασίσουμε αν η διαφορά της συνάρτησης #PerfMatch σε δύο δεδομένους γράφους είναι ίση με 0 ή με μία εκθετικά μεγάλη τιμή. Θα άξιζε να εξετάσουμε πώς ένα ενδεχόμενο θετικό αποτέλεσμα για το πρόβλημα #PerfMatch μπορεί να δώσει κάποιο αποτέλεσμα για τα προβλήματα της SPP, όπως είναι το Graph Isomorphism και το USat.

Περιγραφική πολυπλοκότητα δύσκολων μετρητικών προβλημάτων με εύκολο πρόβλημα απόφασης

Για την περιγραφική πολυπλοκότητα, το ερώτημα που πρέπει να απαντηθεί είναι το εξής: Ποια λογική είναι κατάλληλη ώστε να μπορούν να εκφραστούν τα προβλήματα μιας κλάσης πολυπλοκότητας και μόνο αυτά;

Για την κλάση NP, αποδείχθηκε από τον Fagin [64] ότι χρειαζόμαστε την υπαρξιακή δευτεροβάθμια λογική, που συμβολίζουμε **∃SO**, ισχύει δηλ. NP = ∃SO για πεπερασμένες δομές.

Για την κλάση #P, αρχικά στο άρθρο [132], δείχθηκε ότι #P = #FO για πεπερασμένες διατεταγμένες δομές. Πρόσφατα στο άρθρο [15], οι συγγραφείς ορίζουν την κλάση ΣQSO(FO) και αποδεικνύουν ότι και αυτή ταυτίζεται με τη #P για πεπερασμένες διατεταγμένες δομές. Για παράδειγμα, ο τύπος

$$\alpha = \Sigma X.\forall x \forall y \big(X(x) \wedge X(y) \wedge x \neq y) \rightarrow E(x,y)$$

εκφράζει το πλήθος των κλικών σε ένα γράφο. Η ερμηνεία του τύπου α σε κάποια πεπερασμένη δομή, που κωδικοποιεί ένα γράφο στη συγκεκριμένη περίπτωση, δίνει ένα φυσικό αριθμό, ο οποίος ισούται με το ζητούμενο. Γράφουμε $[[\alpha]](\mathcal{G}) = \#\textsc{Cliques}(G)$.

Στο άρθρο [15] διατυπώθηκαν δύο ερωτήματα που θα μας απασχολήσουν σε αυτή την ενότητα.

1. Ποιος είναι ο λογικός χαρακτηρισμός της TotP;
2. Μπορούμε να ορίσουμε εύρωστες υποκλάσεις της TotP και να προσδιορίσουμε τη σχέση τους με την κλάση FPRAS;

Λογικός χαρακτηρισμός της TotP

Για να δώσουμε ένα λογικό χαρακτηρισμό της κλάσης TotP, χρησιμοποιούμε αναδρομή πολυωνυμικού βάθους. Παρακάτω θα ορίσουμε το πολυωνυμικά φραγμένο σταθερό σημείο ενός τελεστή που δρα πάνω σε συναρτήσεις. Στη συνέχεια, θα περιγράψουμε πώς μας βοηθά να εκφράσουμε το πλήθος των μονοπατιών μιας μηχανής Turing.

Παίρνουμε ως βάση τη λογική **ΣQSO**(**∃SO**) και προσθέτουμε σε αυτή ένα συναρτησιακό σύμβολο που παίρνει ως ορίσματα δευτεροβάθμιες μεταβλητές. Οι τύποι αυτής της λογικής δίνονται από την παρακάτω γραμματική.

$$\alpha := \phi \mid s \mid f(X_1, ..., X_l) \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma X.\alpha \tag{7.1}$$

όπου η ϕ είναι μια υπαρξιακή δευτεροβάθμια πρόταση, f είναι ένα συναρτησιακό σύμβολο και $X_1, \ldots, X_l$ είναι δευτεροβάθμιες μεταβλητές. Προσθέτουμε επίσης στη λογική τον τύπο $[\mathrm{pbfp}_f\, \alpha](\overrightarrow{X})$, όπου το σύμβολο pbfp προέρχεται από το polynomially-bounded fixed point (πολυωνυμικά φραγμένο σταθερό σημείο) και θα αποτιμάται με αναδρομή πολυωνυμικού βάθους.

Ορίζουμε το $\mathcal{SOF}_k$ ως το σύνολο όλων των συναρτήσεων $h : (\mathcal{P}(A^k))^l \to \mathbb{N}$. Έστω $\mathcal{A}$ μια δομή στο λεξιλόγιο σ και $[\mathrm{pbfp}_f\, \alpha](\overrightarrow{X})$ όπου η f έχει πλειομέλεια l. Για να ορίσουμε τη σημασιολογία του $[\mathrm{pbfp}_f\, \alpha](\overrightarrow{X})$, ερμηνεύουμε το $\alpha(\overrightarrow{X}, f)$ ως έναν τελεστή T_α στο $\mathcal{SOF}_k$. Για κάθε $h \in \mathcal{SOF}_k$ και $(S_1, ..., S_l) \in (\mathcal{P}(A^k))^l$, ισχύει ότι:

$$T_\alpha(h)(\overrightarrow{S}) = [[\alpha(\overrightarrow{X}, f)]](\mathcal{A}, V, F)$$

όπου το V είναι μια δευτεροβάθμια ανάθεση στη δομή $\mathcal{A}$ έτσι ώστε $V(X_i) = S_i$, $i \in \{1, ..., l\}$ και η F είναι ανάθεση για την f στη δομή $\mathcal{A}$ τέτοια ώστε $F(f) = h$.

Ορίζουμε την ακολουθία συναρτήσεων $\{h_i\}_{i\in\mathbb{N}}$, $h_i : (\mathcal{P}(A^k))^l \to \mathbb{N}$, τέτοια ώστε

- $h_0(\overrightarrow{S}) = 0$ για κάθε $\overrightarrow{S} \in (\mathcal{P}(A^k))^l$ και
- h_{i+1} να οριστεί ως $T_\alpha(h_i)$ για κάθε $i \in \mathbb{N}$.

Υπάρχουν δύο περιπτώσεις: είτε υπάρχει $n \in \mathbb{N}$ τέτοιο ώστε $h_{n+1}(\overrightarrow{S}) = h_n(\overrightarrow{S})$ για κάθε $\overrightarrow{S} \in (\mathcal{P}(A^k))^l$, και τότε ισχύει $h_j = h_n$ για κάθε $j \geqslant n$, είτε δεν υπάρχει τέτοιο n. Μας ενδιαφέρει αν υπάρχει κάποιο τέτοιο n που να είναι μικρότερο από το $|A|^m$, όπου m είναι η μέγιστη πλειομέλεια κάποιας δευτεροβάθμιας μεταβλητής που βρίσκεται στην εμβέλεια του ποσοδείκτη Σ. Στη συνέχεια, ορίζουμε το πολυωνυμικά φραγμένο σταθερό σημείο ενός T_α ως εξής:

$$\mathrm{pbfp}(T_\alpha) = \begin{cases} f_n & \text{αν } f_n = f_{n+1} \text{ για κάποιο } n \leqslant |A|^m \\ f_{|A|^m} & \text{αν } f_n \neq f_{n+1} \text{ για κάθε } n \leqslant |A|^m. \end{cases}$$

Ορίζουμε τη σημασιολογία του $[\mathrm{pbfp}_f\, \alpha](\overrightarrow{X})$ ως pbfp του T_α, δηλ:

$$[[\,[\mathrm{pbfp}_f\, \alpha](\overrightarrow{X})\,]](\mathcal{A}, V) = \begin{cases} f_n(V(\overrightarrow{X})) & \text{αν } f_n = f_{n+1} \text{ για κάποιο } n \leqslant |A|^m, \\ f_l(V(\overrightarrow{X})) & \text{αλλιώς, όπου } l = |A|^m. \end{cases}$$

Χωρίς βλάβη της γενικότητας, θεωρούμε ότι για κάθε TotP συνάρτηση υπάρχει μια δυαδική μηχανή Turing N, τέτοια ώστε η συνάρτηση δίνει το πλήθος των μονοπατιών της N μείον 1 ή ισοδύναμα, το πλήθος των διακλαδώσεών της. Μπορούμε να εκφράσουμε το πλήθος των διακλαδώσεων της N με τον τύπο $[[\,[\mathrm{pbfp}_f\, \mathsf{tot}](\overrightarrow{C})\,]](\mathcal{A}_x, V_I)$, όπου

- Ο τύπος tot είναι ο εξής:

$$\begin{aligned}\mathsf{tot}(\overrightarrow{C}, f) :=& \exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) + \\ & \Sigma \overrightarrow{C_0}.\, \left(\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \left(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_0(\overrightarrow{S}, \overrightarrow{C_0}, \overrightarrow{t_*})\right) \cdot f(\overrightarrow{C_0})\right) + \\ & \Sigma \overrightarrow{C_1}.\, \left(\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \left(\mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*}) \wedge \Delta_1(\overrightarrow{S}, \overrightarrow{C_1}, \overrightarrow{t_*})\right) \cdot f(\overrightarrow{C_1})\right).\end{aligned}$$

- Η $\mathcal{A}_x$ είναι μια πεπερασμένη διατεταγμένη δομή που κωδικοποιεί την είσοδο της μηχανής Turing.

- V_I είναι μια δευτεροβάθμια ανάθεση στη δομή $\mathcal{A}_x$ τέτοια ώστε $V_I(T) = T_I$, $V_I(E) = E_I$, $V_I(P) = P_I$, $V_I(Q) = Q_I$ και T_I, E_I, V_I, Q_I είναι σχέσεις που κωδικοποιούν την αρχική κατάσταση της μηχανής N.

- Ο τύπος $\exists \overrightarrow{S} \exists \overrightarrow{t_*}\, \mathsf{branching}(\overrightarrow{S}, \overrightarrow{t_*})$ εκφράζει ότι υπάρχει ένας ντετερμινιστικός υπολογισμός της N που οδηγεί σε μια διακλάδωση τη χρονική στιγμή $\overrightarrow{t_*}$.

- $\Delta_i(\overrightarrow{S}, \overrightarrow{C_i}, \overrightarrow{t_*})$, $i = 0, 1$, εκφράζει ότι τη χρονική στιγμή $\overrightarrow{t_*}$ η N κάνει τη μη-ντετερμινιστική επιλογή i και οι $\overrightarrow{C_i}$ κωδικοποιούν την κατάσταση της μηχανής ακριβώς μετά από αυτή την επιλογή.

- Το $f(\overrightarrow{C_i})$ εκφράζει ότι η διαδικασία συνεχίζεται αναδρομικά.

Για τους ακριβείς ορισμούς των παραπάνω τύπων παραπέμπουμε στο Κεφάλαιο 5.

Η κλάση $\mathsf{R\Sigma Q_{SOE}}$ προκύπτει αν περιορίσουμε τη λογική που μόλις ορίσαμε. Παρατηρούμε ότι κάθε υπαρξιακός δευτεροβάθμιος τύπος μπορεί να ελεγχθεί κατά πόσο ισχύει σε μία δομή σε πολυωνυμικό χρόνο. Ονομάζουμε κάθε τέτοιο τύπο **SOE** και δίνουμε ένα συντακτικό ορισμό. Επίσης, κάθε ποσοδείκτης Σ εφαρμόζεται σε κάποια δευτεροβάθμια μεταβλητή, αλλά το άθροισμα χρειάζεται να αποτιμηθεί το πολύ για μία αποτίμηση αυτής της μεταβλητής. Αυτό επίσης μπορεί να γίνει σε πολυωνυμικό χρόνο. Η λογική που ορίζουμε ονομάζεται $\mathbf{R\Sigma Q_{SOE}}$, και ισχύει ότι οι συναρτήσεις που ορίζονται μέσω αυτής αποτελούν την κλάση TotP.

Θεώρημα 7.6. $\mathsf{R\Sigma Q_{SOE}} = \mathsf{TotP}$ *για πεπερασμένες διατεταγμένες δομές.*

Δύο εύρωστες υποκλάσεις της TotP

Στα άρθρα [132, 15] είχαν οριστεί οι παρακάτω υποκλάσεις της TotP και είχε προσδιοριστεί η σχέση κάποιων από αυτές με την FPRAS.

- Η $\#\Sigma_1$ τα προβλήματα της οποίας ανάγονται στο #DNF ως προς αναγωγές γινομένου [132]. Ισχύει $\#\Sigma_1 \subseteq$ FPRAS.
- Η ΣQSO(Σ_2-HORN) για την οπόια το #DisjHornSat είναι πλήρες ως προς φειδωλές αναγωγές [15]. Είναι ανοιχτό ερώτημα η σχέση αυτής της κλάσης με την FPRAS.
- Η ΣQSO(Σ_1[FO]), η οποία είναι κλειστή ως προς πρόσθεση, πολλαπλασιασμό και αφαίρεση κατά ένα. Ισχύει ΣQSO(Σ_1[FO]) $\subseteq$ FPRAS.

Ορίσαμε τις παρακάτω κλάσεις και αποδείξαμε ότι δεν είναι υποκλάσεις της FPRAS εκτός αν οι κλάσεις RP και NP είναι ίσες.

- Η ΣQSO(Σ_2-2SAT) για την οποία το #Disj2Sat είναι πλήρες ως προς φειδωλές αναγωγές.
- Η $\#\Pi_2$-1VAR για την οποία το #MonotoneSat είναι πλήρες ως προς αναγωγές γινομένου.

Μέτρηση ταιριασμάτων σε γράφους με μαύρες και κόκκινες ακμές

Σε αυτή την ενότητα εξετάζουμε το πρόβλημα #Exact Matchings. Υπενθυμίζουμε τον ορισμό του παρακάτω.

#Exact Matchings.
Είσοδος: Ένας γράφος $G = (V, E)$, ένα υποσύνολο των ακμών $E' \subseteq E$ και ένας ακέραιος k.
Έξοδος: Το πλήθος των τέλειων ταιριασμάτων του G που περιέχει ακριβώς k ακμές από το E'.

Το πρόβλημα αυτό γενικεύει το #PerfMatch και έχει πρόβλημα απόφασης που ανήκει στην κλάση RP, και πιο συγκεκριμένα στην κλάση RNC [119]. Δείξαμε ότι επίσης ανάγεται στο πρόβλημα #Weighted Perf Match μέσω μιας αναγωγής που χρησιμοποιεί πολυωνυμική παρεμβολή.

Το πρόβλημα #Exact Matchings σε $K_{3,3}$-ελεύθερους γράφους είναι αποδοτικά επιλύσιμο [152]. Δείξαμε ότι το ίδιο ισχύει για το ίδιο πρόβλημα σε K_5-ελεύθερους γράφους. Στην περίπτωση του #Exact Matchings σε διμερείς γράφους, αποδείξαμε ότι ανήκει στην κλάση TotP, δηλ. το αντίστοιχο πρόβλημα απόφασης ανήκει στην P. Παραμένει ανοιχτή η ακριβής πολυπλοκότητα αυτού του μετρητικού προβλήματος μετά την ολοκλήρωση αυτής της διατριβής.

Ως προς την παραμετρική πολυπλοκότητα, δείξαμε ότι το πρόβλημα #Exact Matchings είναι #W[1]-δύσκολο, περιγράφοντας μία αναγωγή από το πρόβλημα #k-Matchings [51].

Appendix B
Glossary - Γλωσσάριο

αναγωγή γινομένου	product reduction
απλή πρόταση	clause
αυτοαναγώγιμος	self-reducible
εύρωστη κλάση	robust class
$K_{3,3}$-ελεύθερος γράφος	$K_{3,3}$-free graph
K_5-ελεύθερος γράφος	K_5-free graph
κλειστό από κάτω σύνολο	lower set
λεκτικό	literal
λογικό κύκλωμα	boolean circuit
μαντείο	oracle
Μετρητική υπόθεση εκθετικού χρόνου	#Exponential-time hypothesis (#ETH)
μονοπάτι αποδοχής	accepting path
Πιθανοτική υπόθεση εκθετικού χρόνου	randomized exponential-time hypothesis (rETH)
σταθερό σημείο	fixed point
συζευκτική κανονική μορφή	conjunctive normal form (CNF)
συνάρτηση διαμέρισης	partition function
συνάρτηση μέτρησης του μεγέθους διαστήματος	interval size function
ταίριασμα	matching
Υπόθεση εκθετικού χρόνου	Exponential-time hypothesis (ETH)
φειδωλή αναγωγή	parsimonious reduction
φυσικό πρόβλημα	natural problem

References

[1] Dimitris Achlioptas, Amin Coja-Oghlan, and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Structures and Algorithms*, 38(3):251–268, 2011.

[2] Dimitris Achlioptas and Cristopher Moore. Random k-SAT: Two moments suffice to cross a sharp threshold. *SIAM Journal on Computing*, 36(3):740–762, 2006.

[3] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160 (2):781–793, 2004.

[4] Mikhail Alekhnovich and Eli Ben-Sasson. Linear upper bounds for random walk on small density random 3-CNFs. *SIAM Journal on Computing*, 36(5):1248–1263, 2007.

[5] Vedat Levi Alev and Lap Chi Lau. Improved analysis of higher order random walks and applications. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1198–1211. ACM, 2020.

[6] Yeganeh Alimohammadi, Nima Anari, Kirankumar Shiragur, and Thuy-Duong Vuong. Fractionally log-concave and sector-stable polynomials: counting planar matchings and more. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC '21, Virtual Event, Italy, June 21-25, 2021*, pages 433–446. ACM, 2021.

[7] Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *Journal of the ACM*, 57(3):14:1–14:36, 2010.

[8] Eric Allender and Roy S. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.

[9] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, 1993.

[10] Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 989–1002. ACM, 2017.

[11] Nima Anari, Kuikui Liu, and Shayan Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1319–1330. IEEE, 2020.

[12] Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC. *Journal of the ACM*, 67(4):21:1–21:34, 2020.

[13] Antonis Antonopoulos, Eleni Bakali, Aggeliki Chalki, Aris Pagourtzis, Petros Pantavos, and Stathis Zachos. Completeness, approximability and exponential time results for counting problems with easy decision version. *Theoretical Computer Science*, 915:55–73, 2022.

[14] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. *SIGMOD Record*, 49(1):52–59, 2020.

[15] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science*, 16(1), 2020.

[16] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 2009.

[17] Vikraman Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5):835–852, 2006.

[18] Vikraman Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *Proceedings of the 13th International Symposium on Algorithms and Computation, ISAAC 2002 Vancouver, BC, Canada, November 21-23, 2002*, volume 2518 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2002.

[19] László Babai. Graph isomorphism in quasipolynomial time (extended abstract). In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016.

[20] Eleni Bakali. Self-reducible with easy decision version counting problems admit additive error approximation. connections to counting complexity, exponential time complexity, and circuit lower bounds. *CoRR*, abs/1611.01706, 2016.

[21] Eleni Bakali. On Markov chains for #Clustered-Monotone-Sat and other hard counting versions with easy decision version. *International Journal of Scientific and Engineering Research*, 9(3):1203–1211, 2018.

[22] Eleni Bakali. *On properties of counting functions with easy decision version: completeness, approximability, Markov chains, phase transitions.* PhD thesis, National Technical University of Athens, 2018.

[23] Eleni Bakali, Aggeliki Chalki, Andreas Göbel, Aris Pagourtzis, and Stathis Zachos. Guest column: A panorama of counting problems the decision version of which is in P. *SIGACT News*, 53(3):46–68, 2022.

[24] Eleni Bakali, Aggeliki Chalki, and Aris Pagourtzis. Characterizations and approximability of hard counting classes below #P. In *Proceedings of the 16th International Conference on Theory and Applications of Models of Computation, TAMC 2020, Changsha, China, October 18-20, 2020*, volume 12337 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2020. extended version: *CoRR*, abs/2003.02524.

[25] Eleni Bakali, Aggeliki Chalki, Aris Pagourtzis, Petros Pantavos, and Stathis Zachos. Completeness results for counting problems with easy decision. In *Proceedings of the 10th International Conference on Algorithms and Complexity, CIAC 2017, Athens, Greece, May 24-26, 2017*, volume 10236 of *Lecture Notes in Computer Science*, pages 55–66, 2017.

[26] José L. Balcázar, Ronald V. Book, and Uwe Schöning. The polynomial-time hierarchy and sparse oracles. *Journal of the ACM*, 33(3):603–617, 1986.

[27] Evangelos Bampas, Andreas-Nikolas Göbel, Aris Pagourtzis, and Aris Tentes. On the connection between interval size functions and path counting. *Computational Complexity*, 26(2):421–467, 2017.

[28] Régis Barbanchon. On unique graph 3-colorability and parsimonious reductions in the plane. *Theoretical Computer Science*, 319(1-3):455–482, 2004.

[29] Alexander Barvinok. Approximate counting via random optimization. *Random Structures and Algorithms*, 11:187–198, 1997.

[30] Alexander Barvinok. Computing the partition function for cliques in a graph. *Theory of Computing*, 11:339–355, 2015.

[31] Alexander Barvinok. Approximating permanents and hafnians. *Discrete Analysis*, 2:34 pp., 2017.

[32] Alexander Barvinok. *Combinatorics and Complexity of Partition Functions*. Springer, 2017.

[33] Alexander Barvinok and Alex Samorodnitsky. Random weighting, asymptotic counting, and inverse isoperimetry. *Israel Journal of Mathematics*, 158:159–191, 03 2007.

[34] Alexander Barvinok and Pablo Soberón. Computing the partition function for graph homomorphisms with multiplicities. *Journal of Combinatorial Theory, Series A*, 137:1–26, 2016.

[35] Richard Beigel and John Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103(1):3–23, 1992.

[36] Gleb Belov, Samuel Esler, Dylan Fernando, Pierre Le Bodic, and George L. Nemhauser. Estimating the size of search trees by sampling with domain knowledge. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 473–479. ijcai.org, 2017.

[37] Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, Heng Guo, and Daniel Stefankovic. Approximation via correlation decay when strong spatial mixing fails. *SIAM Journal on Computing*, 48(2):279–349, 2019.

[38] Julius Borcea and Petter Brändén. The Lee-Yang and Pólya-Schur programs. I. Linear operators preserving stability. *Inventiones mathematicae*, 177:541–569, 2009.

[39] Allan Borodin, Stephen A. Cook, and Nicholas Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1):113–136, 1983.

[40] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *Journal of the ACM*, 60(5):34:1–34:41, 2013.

[41] Andrei A. Bulatov, Martin E. Dyer, Leslie Ann Goldberg, Markus Jalsenius, and David Richerby. The complexity of weighted boolean #CSP with mixed signs. *Theoretical Computer Science*, 410(38-40):3949–3961, 2009.

[42] Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems*, volume 1. Cambridge University Press, 2017.

[43] Jin-Yi Cai, Heng Guo, and Tyson Williams. The complexity of counting edge colorings and a dichotomy for some higher domain Holant problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 601–610. IEEE Computer Society, 2014.

[44] Jin-Yi Cai and Lane A. Hemachandra. On the power of parity polynomial time. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science, STACS 89, Paderborn, FRG, February 16-18, 1989*, volume 349 of *Lecture Notes in Computer Science*, pages 229–239. Springer, 1989.

[45] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-SAT: An isolation lemma for k-CNFs. *Journal of Computer and System Sciences*, 74(3):386–393, 2008.

[46] Peter C. Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI 1991, Sydney, Australia, August 24-30, 1991*, pages 331–340. Morgan Kaufmann, 1991.

[47] Pang Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, 1992.

[48] Steve Chien. A determinant-based algorithm for counting perfect matchings in a general graph. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 728–735. SIAM, 2004.

[49] Youngbin Choe, James G. Oxley, Alan D. Sokal, and David G. Wagner. Homogeneous multivariate polynomials with the half-plane property. *Advances in Applied Mathematics*, 32(1-2):88–187, 2004.

[50] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.

[51] Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, ICALP 2013,*

Riga, Latvia, July 8-12, 2013, Part I, volume 7965 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2013.

[52] Radu Curticapean. *The simple, little and slow things count: on parameterized counting complexity.* PhD thesis, Saarland University, 2015.

[53] Radu Curticapean. Counting matchings with k unmatched vertices in planar graphs. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 33:1–33:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[54] Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. *Information and Computation*, 261:265–280, 2018.

[55] Radu Curticapean and Mingji Xia. Parameterizing the permanent: Hardness for K_8-minor-free graphs. *CoRR*, abs/2108.12879, 2021.

[56] Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms*, 10(4):21:1–21:32, 2014.

[57] Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM Journal on Computing*, 31(5):1527–1541, 2002.

[58] Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991.

[59] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

[60] Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *Journal of Computer and System Sciences*, 76(3-4):267–277, 2010.

[61] Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3-4):260–289, 2000.

[62] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[63] Jeff Erickson. *Algorithms.* University of Illinois, Lecture Notes, 2015.

[64] Ronald Fagin. Generalized first-order spectra and polynomial time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974.

[65] Piotr Faliszewski and Lane A. Hemaspaandra. The complexity of power-index comparison. *Theoretical Computer Science*, 410(1):101–107, 2009.

[66] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[67] Sophie Fischer, Lane A. Hemaspaandra, and Leen Torenvliet. Witness-isomorphic reductions and the local search problem. In *Complexity, logic, and recursion theory*, chapter 7, pages 207–223. CRC Press, 1997.

[68] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

[69] Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009.

[70] Shmuel Friedland and Daniel Levy. A polynomial-time approximation algorithm for the number of k-matchings in bipartite graphs. *CoRR*, abs/cs/0607135, 2006.

[71] Martin Fürer and Shiva Prasad Kasiviswanathan. An almost linear time approximation algorithm for the permanent of a random (0-1) matrix. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2004, Chennai, India, December 16-18, 2004*, volume 3328 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2004.

[72] Martin Fürer and Shiva Prasad Kasiviswanathan. Approximately counting perfect matchings in general graphs. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005*, pages 263–272. SIAM, 2005.

[73] Andreas Galanis, Qi Ge, Daniel Stefankovic, Eric Vigoda, and Linji Yang. Improved inapproximability results for counting independent sets in the hard-core model. *Random Structures and Algorithms*, 45(1):78–110, 2014.

[74] Andreas Galanis, Leslie Ann Goldberg, and Mark Jerrum. A complexity trichotomy for approximately counting list H-colorings. *ACM Transactions on Computation Theory*, 9(2), April 2017.

[75] John Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[76] Chris D. Godsil and Ivan Gutman. On the theory of the matching polynomial. *Journal of Graph Theory*, 5(2):137–144, 1981.

[77] Oded Goldreich. *Computational Complexity - A Conceptual Perspective*. Cambridge University Press, 2008.

[78] Leslie M. Goldschlager and Ian Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.

[79] Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997.

[80] Yenjo Han, Lane A. Hemaspaandra, and Thomas Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.

[81] Frank Harary. *Graph Theory and Theoretical Physics*. Academic Press, 1967.

[82] Ole J. Heilmann and Elliott Lieb. Theory of monomer-dimer systems. *Communications in Mathematical Physics*, 27:166, 1972.

[83] Lane A. Hemaspaandra. The power of self-reducibility: Selectivity, information, and approximation. In *Complexity and Approximation - In Memory of Ker-I Ko*, volume 12000 of *Lecture Notes in Computer Science*, pages 19–47. Springer, 2020.

[84] Lane A. Hemaspaandra, Christopher M. Homan, Sven Kosub, and Klaus W. Wagner. The complexity of computing the size of an interval. *SIAM Journal on Computing*, 36(5):1264–1300, 2007.

[85] Ulrich Hertrampf. Relations among mod-classes. *Theoretical Computer Science*, 74(3):325–328, 1990.

[86] Aimo Hinkkanen. Schur products of certain polynomials. In *Lipa's Legacy: Proceedings of the Bers Colloquium, United States*, volume 211 of *Contemporary Mathematics*, pages 285–295. American Mathematical Society, 1997.

[87] Roger A. Horn. The hadamard product. *Proceedings of Symposia in Applied Mathematics*, 40:87–169, 1990.

[88] Neil Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer, 1999.

[89] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[90] Mark Jerrum. Two-dimensional monomer-dimer systems are computationally intractable. *Journal of Statistical Physics*, 48(1-2):121–134, 1987.

[91] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.

[92] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: An approach to approximate counting and integration. In *Approximation Algorithms for NP-Hard Problems*, chapter 12, pages 482–520. PWS Publishing Company, USA, 1996.

[93] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004.

[94] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[95] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[96] Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 551–557. ACM/SIAM, 1995.

[97] David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, 1999.

[98] Richard M Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[99] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[100] Pieter W. Kasteleyn. The statistics of dimers on a lattice : I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27(12):1209–1225, December 1961.

[101] Samir Khuller and Vijay V. Vazirani. Planar graph coloring is not self-reducible, assuming P != NP. *Theoretical Computer Science*, 88(1):183–189, 1991.

[102] Aggelos Kiayias, Aris Pagourtzis, Kiron Sharma, and Stathis Zachos. Acceptor-definable counting classes. In *Proceedings of the 8th Panhellenic Conference on Informatics, PCI 2001, Nicosia, Cyprus, November 8-10, 2001, Revised Selected Papers*, volume 2563 of *Lecture Notes in Computer Science*, pages 453–463. Springer, 2001.

[103] Aggelos Kiayias, Aris Pagourtzis, and Stathis Zachos. Cook reductions blur structural differences between functional complexity classes. In *Proceedings of the 2nd Panhellenic Logic Symposium, PLS 1999, Delphi, Greece, July 13-17, 1999*, pages 132–137, 1999.

[104] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh. Estimating search tree size. In *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1014–1019. AAAI Press, 2006.

[105] Donald Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29:122–136, 1974.

[106] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhäuser/Springer, 1993.

[107] William Kocay and Donald L. Kreher. *Graphs, Algorithms, and Optimization.* CRC Press, 2017.

[108] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[109] Florent Krzakala, Andrea Montanari, Federico Ricci-Tersenghi, Guilhem Semerjian, and Lenka Zdeborová. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Sciences*, 104(25):10318–10323, 2007.

[110] Jonathan Leake and Nick Ryder. Generalizations of the matching polynomial to the multivariate independence polynomial. *Algebraic Combinatorics*, 2(5):781–802, 2016.

[111] David A. Levin and Yuval Peres. *Markov Chains and Mixing Times*. American Mathematical Society, 2017.

[112] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[113] Chengyu Lin, Jingcheng Liu, and Pinyan Lu. A simple FPTAS for counting edge covers. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 341–348. SIAM, 2014.

[114] Jingcheng Liu and Pinyan Lu. FPTAS for counting monotone CNF. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1531–1548. SIAM, 2015.

[115] Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures and Algorithms*, 15(3-4):229–241, 1999.

[116] William McCuaig, Neil Robertson, Paul D. Seymour, and Robin Thomas. Permanents, pfaffian orientations, and even directed circuits (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 402–405. ACM, 1997.

[117] Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory of Computing Systems*, 58(1):45–59, 2016.

[118] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992*, pages 459–465. AAAI Press / The MIT Press, 1992.

[119] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 3 1987.

[120] Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Randomized and approximation algorithms for blue-red matching. In *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science 2007, MFCS 2007, Český Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 715–725. Springer, 2007.

[121] Mitsunori Ogiwara and Lane A. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.

[122] Aris Pagourtzis. On the complexity of hard counting problems with easy decision version. In *Proceedings of the 3rd Panhellenic Logic Symposium, PLS 2001, Anogia, Greece, July 17-21, 2001*, pages 21–29, 2001.

[123] Aris Pagourtzis and Stathis Zachos. The complexity of counting functions with easy decision version. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science 2006, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 741–752. Springer, 2006.

[124] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[125] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted minimum spanning tree problems (extended abstract). In *Proceedings of the 6th Colloquium on Automata, Languages and Programming, ICALP 1979, Graz, Austria, July 16-20, 1979*, volume 71 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1979.

[126] Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science, Dortmund, Germany, January 5-7, 1983*, volume 145 of *Lecture Notes in Computer Science*, pages 269–276. Springer, 1983.

[127] Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM Journal on Computing*, 46(6):1893–1919, 2017.

[128] Tayfun Pay and James L. Cox. An overview of some semantic and syntactic complexity classes. *Electronic Colloquium on Computational Complexity*, page 166, 2018.

[129] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79(6):984–1001, 2013.

[130] Paul W. Purdom. Tree size by partial backtracking. *SIAM Journal on Computing*, 7(4):481–491, 1978.

[131] Rajesh P. N. Rao, Jörg Rothe, and Osamu Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, 1994.

[132] Sanjeev Saluja, K.V. Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of #P functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995.

[133] Johannes Schmidt. *Enumeration: Algorithms and complexity.* Diplomarbeit, Leibniz Universität Hannover, 2009.

[134] Claus-Peter Schnorr. Optimal algorithms for self-reducible problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming, ICALP 1976, University of Edinburgh, UK, July 20-23, 1976*, pages 322–337. Edinburgh University Press, 1976.

[135] Alexander Schrijver. *Theory of Linear and Integer programming.* Wiley-Interscience, 1986.

[136] Janos Simon. *On some central problems in computational complexity.* PhD thesis, Cornell University, 1975.

[137] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.

[138] Allan Sly. Computational transition at the uniqueness threshold. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 287–296. IEEE Computer Society, 2010.

[139] Daniel Stefankovic, Eric Vigoda, and John Wilmes. On counting perfect matchings in general graphs. In *Proceedings of the 13th Latin American Theoretical Informatics Symposium, LATIN 2018, Buenos Aires, Argentina, April 16-19, 2018*, volume 10807 of *Lecture Notes in Computer Science*, pages 873–885. Springer, 2018.

[140] Larry Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[141] Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. *Theory of Computing Systems*, 59(3):416–439, 2016.

[142] Harold N. V. Temperley and Michael E. Fisher. Dimer problem in statistical mechanics-an exact result. *The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics*, 6(68):1061–1063, 1961.

[143] Dimitrios M. Thilikos and Sebastian Wiederrecht. Killing a vortex. *CoRR*, abs/2207.04923, 2022.

[144] Seinosuke Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[145] Boris A. Trakhtenbrot. Autoreducibility. *Doklady Akademii Nauk SSSR*, 192:6:1224 – 1227, 1970.

[146] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[147] Radislav Vaisman and Dirk P. Kroese. Stochastic enumeration method for counting trees. *Methodology and Computing in Applied Probability*, 19(1):31–73, 3 2017.

[148] Leslie G. Valiant. Relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

[149] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[150] Leslie G. Valiant. Accidental algorithms. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, 21-24 October 2006, Berkeley, California, USA*, pages 509–517. IEEE Computer Society, 2006.

[151] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[152] Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computation*, 80(2):152–164, 1989.

[153] Heribert Vollmer. On different reducibility notions for function classes. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science, STACS 94, Caen, France, February 24-26, 1994*, volume 775 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 1994.

[154] Klaus Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[155] Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006, Seattle, WA, USA, May 21-23, 2006*, pages 140–149. ACM, 2006.

[156] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

[157] Stathis Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36(3):433–451, 1988.

[158] Viktória Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):77–82, 1991.