

The Complexity of Modular Counting in
Constraint Satisfaction Problems

John Faben

Submitted for the degree of Doctor of Philosophy

April 26, 2012

Abstract

Constraint Satisfaction Problems are a broad class of combinatorial problems, including several classical decision problems such as graph colouring and SAT, and a range of problems from other areas, including statistical physics, DNA sequencing and scheduling problems. There are a variety of dichotomy theorems for various subclasses of CSP in various complexity classes, one of the most notable is Bulatov’s proof that a dichotomy holds for $\#CSP$ [Bul08], that is, all $\#CSP$ problems are either $\#P$ -complete or in FP .

In this thesis, we look at the complexity of modular counting in some restricted classes of CSP, answering questions of the sort “What is the number of solutions to this CSP *modulo* k ?” for various k . In particular, we discuss CSP restricted to relations on the domain of size 2, Boolean CSP or Generalised Satisfiability, and CSP restricted to a single, symmetric, binary relation, or Graph Homomorphism. In [CH96], Creignou and Hermann proved a dichotomy theorem for counting solutions to Boolean CSP problems, and characterised the easy cases. We provide a proof of a dichotomy theorem for $\#_k$ Boolean CSP for all k , characterising the easy cases.

In [DG00], Dyer and Greenhill proved a dichotomy theorem for counting the

number of homomorphisms into a fixed graph H , or H -colouring, and provided a characterisation of the types of H for which the problem is tractable. We give some results on $\oplus H$ -colouring. We give a conjectured dichotomy for the general $\oplus H$ -colouring problem, based on a condition related to the automorphism group of H . We show that this dichotomy holds for the case in which H is restricted to be a tree, and give some results about the complexity of determining, given an arbitrary H , whether the associated H -colouring problem is tractable assuming the dichotomy holds.

Acknowledgements

With thanks to my supervisor Mark Jerrum for his patience in reading drafts of the thesis, and insight which helped produce its contents. To my examiners, Graham Brightwell and Andrei Krokhin for not understanding several of my proofs, thus forcing me to explain them better. To Jess for beating me in the thesis race, and inspiring me to finish. To all the other students in 202 for making the office somewhere I actually wanted to spend time. And to my parents and godmother for all the help they've given me over the decade I've spent at university.

Contents

1	Introduction	9
1.1	Overview of the contents of this thesis	13
2	CSP	15
2.1	Definition of CSP	15
2.2	Computational Complexity and CSP	16
3	Modular Counting Classes	21
3.1	The classes $\#_k\text{P}$	22
3.1.1	Completeness	25
3.1.2	$\#_k\text{P}$ vs. Mod_kP	27
3.2	Relations between $\#_k\text{P}$ classes	31
3.3	Modular Counting Problems	34
3.4	The Power of Modular Counting	37
4	$\#_k$ Independent Set	41
4.1	Independent Set	42
4.2	Bipartite Independent Set	47

5	Boolean CSP	53
5.1	Boolean CSP	53
5.2	Preliminaries	55
5.3	The classes $\#_k$ -SAT	57
6	Graph Homomorphism	65
6.1	Graph Homomorphisms	65
6.2	H-colouring and Complexity	67
6.3	Pinning colours to vertices	68
6.3.1	The Lovász vector of a Graph	68
6.3.2	Building Gadgets	75
6.4	Reduction by Involutions	86
6.5	Trees	90
6.5.1	Involution-Free Trees	92
6.5.2	The Reduction	98
6.5.3	A Dichotomy for Trees	102
6.6	Other Graphs	103
7	Complexity of finding the Reduced Form	105
7.1	Complexity of finding the Reduced Form	105
7.1.1	Modular Graph Automorphism	105
7.1.2	Other Moduli	111
7.2	Complexity of the dichotomy	115

Chapter 1

Introduction

Constraint Satisfaction Problems, or CSPs, form a general problem framework with applications from statistical physics to artificial intelligence, and which contains a variety of commonly studied combinatorial problems, including SAT.

The framework is one in which values from a given domain are assigned to some variables, with constraints on the types of assignments which are allowed to given tuples of variables.

Two examples of well-known problems that can be described in the CSP framework are SAT and 3-colouring. In SAT, the clauses form the constraints, and the variables are assigned values from the Boolean domain. In 3-colouring, the variables are the vertices of the graph, the constraints are given by requirement that vertices which share an edge are given different colours, and the domain is the set of three colours. Further examples are given in Chapter 2.

The general CSP decision problem is NP-complete, including within it such problems as 3-SAT and 3-colouring. When discussing questions of complexity,

therefore, we usually consider CSPs in which the relations are restricted to come from a given set, or constraint language. This is referred to as non-uniform CSP.

There are two types of constraint language with which we will concern ourselves in this thesis: Boolean CSP and Graph Homomorphism.

Boolean CSP is CSP in which the domain of the relations in the constraint language is of size two. In other words, the variables can be mapped to one of two values, which are usually thought of either as 0 and 1 or as True and False. When Schaefer first studied Boolean CSP in [Sch78], he referred to it as Generalised Satisfiability.

Graph homomorphism is a natural generalization of graph colouring. In a graph colouring problem, we place colours on the vertices of a graph in such a way that no two adjacent vertices are the same colour. In graph homomorphism problems, we might have more general restrictions on the placements of the colours - *e.g.* we might have a problem in which red vertices can be adjacent to vertices of any colour (including red) but blue and green vertices can only be adjacent to red. We can draw a graph on the colours, which has an edge between two colours if and only if those two colours are allowed to be adjacent to each other. Then a colouring of a graph with colours which obey these adjacencies is a homomorphism from that graph into our new colour graph.

The problem of finding homomorphisms into a given graph H is often referred to as H -colouring. It can be seen that the classical graph colouring problem is just the case of homomorphisms into a complete graph, or K_n -colouring, as colours can be adjacent to any other colour, but not to themselves, as there are no loops. If we allow loops on both G and H , the graph homomorphism

problem is the restriction of CSP to a single, symmetric, binary relation.

There is a famous conjecture by Feder and Vardi [FV98] that for every constraint language, Γ , the decision problem associated with an arbitrary CSP whose constraints all consist of relations which are in Γ is either in P or NP-complete, depending on Γ . There has been a large amount of work on this conjecture, and several special cases of it have been proven ([Bul06] [Sch78] [HN90]). More discussion of this is given in Chapter 2. Note that there can be no such dichotomy for the whole of NP, due to a result of Ladner from [Lad75] that if $P \neq NP$ then there is an infinite hierarchy of classes of intermediate difficulty.

The complexity of the decision versions of CSP problems is well studied - *i.e.* the answer to the question “is it possible to satisfy the given constraints?”. There are a variety of other questions we can ask about a given CSP instance: how many ways are there of satisfying the given constraints; what is the maximum number of constraints we can satisfy; can we approximate the number of ways of satisfying the constraints; what is the parity of the number of ways of satisfying the constraints? In this thesis, we will ask questions of the last sort, although we will not restrict our attention to parity, and will instead ask “what is the number of solutions *modulo* k ?” for various k .

Modular counting complexity is an area in which there has been relatively little progress since it was introduced by Valiant in his 1978 paper *The Complexity of Computing the Permanent* [Val78]. The equivalent of #P for counting *modulo* k is denoted $\#_k P$, with $\oplus P$ used to denote counting *modulo* 2. However, there has been an increase in interest in the area in recent years, partly inspired

by Valiant's paper *Completeness for Parity Problems* [Val05] which was among the first to demonstrate $\oplus\text{P}$ -hardness for some natural problems for which the $\#\text{P}$ reductions from SAT are not parsimonious, *i.e.* do not preserve the number of solutions exactly. Some $\#_k\text{P}$ -hardness results and surprising algorithms for some $\#_k\text{P}$ problems have emerged from work on Holographic Algorithms *e.g.* [Val06]. More details of this are given in Chapter 3.

In this thesis, we combine these two areas. For certain restricted cases of CSP, we ask the question: what is the complexity of determining the number of solutions *modulo* k , for natural numbers $k > 1$? We find results for the complexity of Boolean CSP, which is CSP restricted to a domain of size two, and for graph homomorphism. For the former set of problems we completely classify the complexity of computing the number of solutions *modulo* k for all k . We also conjecture a dichotomy for the complexity of computing the number of homomorphisms into a fixed graph, H , *modulo* 2, and prove that this dichotomy holds when H is restricted to be a tree.

1.1 Overview of the contents of this thesis

We consider the complexity of modular counting in two restrictions of CSP, the restriction of the domain to size two, or Boolean CSP, and the restriction to a single, symmetric, binary relation, or graph homomorphism. We give a complete dichotomy for all k for $\#_k$ Boolean CSP, with a characterisation of the easy problems. We provide some potentially useful technology for proving results about $\#_k$ Graph Homomorphism, and prove a dichotomy theorem for \oplus Graph Homomorphism where the target graph for the homomorphisms is restricted to be a tree.

Chapter 2 Contains a description and formal definition of the CSP problem, and a summary of some of the many dichotomy results for various restrictions of CSP in various complexity classes.

Chapter 3 Contains a discussion of the relevant complexity theoretic foundations for the results in the remainder of the thesis. We consider various relations between $\#_k$ P classes. In particular, we use a result of Beigel and Gill from [BG92] along with the Chinese Remainder Theorem to show that, in essence, the only modular counting classes we need concern ourselves with are those in which counting is done *modulo* a fixed prime.

Chapter 4 Contains a series of reductions which show that counting independent sets in a graph is $\#_k$ P-complete for all k . In fact, we show that this is true even if the graphs are restricted to be bipartite. These reductions preserve the number of solutions *modulo* k exactly, and proceed

directly from SAT. The reductions in later chapters proceed from these two problems.

Chapter 5 Contains a dichotomy for modular counting in the Boolean CSP, or Generalised Satisfiability Problem. That is, for any constraint language Γ which consists only of relations on the domain of size 2, either we prove that the associated $\#_k\text{CSP}(\Gamma)$ problem is solvable in polynomial time or that it is hard for a relevant complexity class. The result is very similar to that found in [CH96] for counting in the usual sense, just including one new easy type of problem for the case where $k = 2$, which results from symmetries of the solution space in those types of problem.

Chapter 6 Contains some results on the complexity of counting graph homomorphisms *modulo* integers, graph homomorphisms being the special case of CSP with one binary, symmetric relation. In Sections 6.1 and 6.3 we develop some general techniques, which we then use to prove a dichotomy for $\bigoplus H$ -colouring in the case where H is a tree in Section 6.5. We offer a conjecture for a dichotomy for the general $\bigoplus H$ -colouring problem. This conjecture is based on a condition related to the automorphism group of H . We give details of this condition in Section 6.4.

Chapter 7 Contains results on the complexity of computing the condition given in Section 6.4 for a given graph H .

Chapter 2

CSP

2.1 Definition of CSP

In a CSP, we are given a set of variables, a domain from which the variables can take values, and a set of constraints, which is a tuple of variables along with a relation on the domain. We satisfy the constraints if we can assign values from the domain to the variables such that the resulting tuples of domain values are all in the relevant relations. Formally:

Definition 2.1.1. *A CSP is a triple (X, C, D) , where C is a set of constraints over some variables X from a domain D . Each constraint is a pair (t, R) , where t is an n -tuple of variables and R is an n -ary relation on D (i.e. a subset of D^n), for some integer n . An evaluation of a CSP is a map $v : X \rightarrow D$. An evaluation v satisfies a constraint $((x_1, \dots, x_n), R)$ if $(v(x_1), \dots, v(x_n)) \in R$. A solution to a CSP is an evaluation which satisfies all of the constraints.*

Definition 2.1.2. *Let Γ be a set of relations on a domain D , or a constraint*

language. We call $\text{CSP}(\Gamma)$ the set of CSP problems (X, C, D) such that D is the domain of Γ , and every constraint in C is of the form (t, R) , where R is a relation in Γ and t is an n -tuple of variables from X .

Example 2.1.3. 3-colouring is a CSP problem, the set of variables, X , is the set of vertices of a graph, the constraints are of the form $((a, b), \neq)$ for each adjacent pair of vertices a and b and the domain is a set of three colours.

Example 2.1.4. 3-SAT is a CSP. The set of variables is usually denoted x_1, \dots, x_n . The constraints are of the form $((x_i, x_k, x_j), R_l)$ where R_l is one of the four relations: $\{0, 1\}^3 \setminus \{(0, 0, 0)\}$, $\{0, 1\}^3 \setminus \{(1, 0, 0)\}$, $\{0, 1\}^3 \setminus \{(1, 1, 0)\}$ and $\{0, 1\}^3 \setminus \{(1, 1, 1)\}$ and (x_i, x_k, x_j) is some 3-tuple of variables from the set $\{x_1, \dots, x_n\}$.

2.2 Computational Complexity and CSP

The general CSP problem, defined above, contains such problems as 3-colouring and 3-SAT, and so is NP-complete. We therefore discuss questions of complexity for the case in which the sorts of relations allowed in the constraint set are restricted. In particular, we will usually regard Γ as fixed, and consider the complexity of $\text{CSP}(\Gamma)$. Note that by fixing Γ , we must also fix the domain D of the relations in Γ . We refer to Γ as a constraint language.

The dichotomy conjecture for CSP, due to Feder and Vardi [FV98], states that for every finite constraint language Γ , the associated decision problem, “Given an arbitrary CSP from $\text{CSP}(\Gamma)$, is there any way of satisfying the constraints?”, is either in P or NP-complete.

Conjecture 2.2.1. [FV98] *For every constraint language Γ , $CSP(\Gamma)$ is either in P or NP -complete.*

This conjecture has been the subject of a lot of recent work. The decision version of the conjecture is still open, but has been shown to be true for a variety of restricted cases of the CSP problem, including graph homomorphism [HN90], Boolean CSP [Sch78] and CSP restricted to a domain of size 3 [Bul06].

Research has not been restricted to the decision versions of CSP problems. Indeed, there are a wide variety of counting variants of CSP problems that have been studied. Of particular interest is the case of the complexity of exact counting in CSP, or $\#CSP$.

Definition 2.2.2. *Given a constraint language Γ , we define $\#CSP(\Gamma)$ to be the problem of determining the number of satisfying assignments to a set of Γ constraints.*

Example 2.2.3. *The problem of counting the number of 3-colourings of a graph G , $\#3$ -colouring, is the problem $\#CSP(\Gamma)$ where Γ is the disequality relation on a domain of size three. i.e. the domain of Γ is some set of three colours and the constraints are disequalities on pairs of vertices.*

Example 2.2.4. *The problem of determining the number of solutions to linear equations over three variables in $GF(2)$ is given by $\#CSP(\Gamma)$ where, $\Gamma = \{R_0, R_1\}$, where R_0 and R_1 are relations defined over $\{0, 1\}^3$ such that $(x, y, z) \in R_i \iff x \oplus y \oplus z \equiv i$. That is, $R_0 = \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\}$ and $R_1 = \{(1, 1, 1), (1, 0, 0), (0, 1, 0), (0, 0, 1)\}$.*

In [Bul06], Bulatov proved a dichotomy theorem for the general #CSP problem, the problem of exactly counting the number of solutions to a given CSP.

Theorem 2.2.5. *[Bul06] For every constraint language, Γ , either $\#CSP(\Gamma)$ is in FP, or it is #P-complete.*

In principle, Bulatov left open the question of the decidability of the dichotomy itself, *i.e.* it was still possible that there might exist no algorithm which could determine, for a given constraint language Γ , whether $CSP(\Gamma)$ was #P-complete or in FP.

Dyer and Richerby settled the question of decidability of the dichotomy in [DR11]. They gave an algorithm which decides for a given #CSP problem whether it lies in FP or is #P-complete. Their proof relies on a combinatorial condition on the structure of the relations in Γ which they refer to as strong balance. They show that if a constraint language Γ is strongly balanced then $CSP(\Gamma)$ is in FP, and that it is #P-complete otherwise. They also show that determining strong balance is not only decidable, but in fact, lies in NP.

Theorem 2.2.6. *[DR11] If Γ is strongly balanced, $\#CSP(\Gamma)$ is in FP. Otherwise, $\#CSP(\Gamma)$ is #P-complete. Moreover, the dichotomy is decidable.*

Their proof relies on interpolation techniques for the hardness results, so cannot be adapted easily for the modular counting classes dealt with in this thesis.

Other variants that have been studied include the complexity of approximating the number of solutions to a given CSP problem and the weighted CSP

problem. In the weighted CSP problem, the relations which form part of the constraints in the CSP problem are replaced with cost functions. Let x be an n -tuple of variables, and f be a function from D^n into some set, usually \mathbb{N} , \mathbb{R} or \mathbb{Q} . Then the weight of a constraint (x, f) in a given assignment is the value of $f(v)$, where v is the vector of values assigned to the tuple x by the assignment. The weight of an assignment is the sum of the weights of the constraints in the assignment. Note that if the functions are into \mathbb{R} they must be into some polynomial-time computable subset of \mathbb{R} , *e.g.* the algebraic numbers, for us to talk about the complexity of computing such a function.

Dyer, Goldberg and Jerrum gave a dichotomy for weighted Boolean $\#$ CSP restricted to non-negative weights in [DGJ09]. They showed that the non-negative weighted Boolean CSP problem can be computed in polynomial time if the functions involved are either of product type or pure affine, and is $\#$ P-complete otherwise. Product type means that the functions can be decomposed into smaller, easy to compute functions, and the functions are the products of these easy to compute functions. For a function to be pure affine means that the function is a constant multiple of some function which is defined by a system of linear equations over $\text{GF}(2)$.

Cai *et al* proved a dichotomy for weighted Boolean CSP with arbitrary complex weights in [CCL10].

This dichotomy was also found independently for real weights by Bulatov, Dyer, Goldberg, Jalsenius and Richerby in [BDG⁺09].

The same authors along with Jerrum have recently extended the results of Bulatov in [Bul08] for $\#$ CSP to provide a dichotomy theorem for general

weighted $\#CSP$ with positive rational weights in [BDG⁺10]. Cai, Chen and Yu gave a dichotomy for general non-negative weighted $\#CSP$ in [CCL11], this dichotomy resting on a fairly straightforward condition on the functions which can be described by the weighted $\#CSP$, which they describe as the functions being balanced. They also show that the problem of determining whether a given weighted $\#CSP$ is balanced is in NP.

Dyer, Goldberg and Jerrum proposed a trichotomy for the problem of approximating the number of solutions to Boolean CSP problems in [DGJ10]. They showed that every Boolean CSP problem can either be approximated efficiently or can be reduced by approximation-preserving reductions to one of SAT or Bipartite Independent Set, and conjecture that these latter two classes are not inter-reducible.

Yamakami extended this approximation result to the weighted version, in which the weights are allowed to take complex values, in [Yam11].

Chapter 3

Modular Counting Classes

In this section, we will look in some detail at the complexity classes that we study in the rest of the thesis. These complexity classes, $\#_k\text{P}$, formalize the notion of counting solutions to NP problems *modulo* k for $k = 2, 3, 4, 5, \dots$. We give a formal definition of these as function classes, and discuss their relationship to the language classes, which are the decision problems: is the number of solutions *modulo* k non-zero? We find that the language classes actually capture most of the power of the function classes. Using results of Beigel and Gill [BG92], along with the Chinese Remainder Theorem, we then show that in some sense the only interesting $\#_k\text{P}$ classes are the $\#_p\text{P}$ with p prime. In Section 3.3, we look at a variety of previously known modular counting complexity results, including problems for which the modular counting version is easy where counting is known to be hard as well as problems for which hardness results for modular counting have been proven, but not via parsimonious reductions from SAT. Finally, in Section 3.4 we summarise some results on the power

of modular counting, in particular recalling the Valiant-Vazirani Theorem and Toda’s Theorem.

3.1 The classes $\#_k\mathbf{P}$

Previous work dealing with the complexity of modular counting (*e.g.* [CH89] [Her90]) has tended to define the relevant complexity class as $\text{Mod}_k\mathbf{P}$, the set of languages which have non-zero number of accepting paths *modulo* k for some Turing Machine M . Formally, $\text{Mod}_k\mathbf{P}$ contains for every function $f \in \#P$ the language:

$$\{x \in \Sigma^* \mid f(x) \not\equiv 0 \pmod{k}\}.$$

For the purposes of this thesis, we have chosen to define a slightly different set of classes, which we refer to here as $\#_k\mathbf{P}$, and which we think more intuitively capture the notion of counting *modulo* k . Analogous to $\#P$, we define $\#_k\mathbf{P}$ to be the class of problems “compute $f(x)$ *modulo* k ” where $f(x)$ is the number of accepting paths of a polynomial time Turing Machine on input x , *i.e.* f is a $\#P$ function. Like $\#P$, this is a class of function problems rather than a class of decision problems. Below we give formal definitions of $\#P$ and $\#_k\mathbf{P}$.

Definition 3.1.1. A **language** over an alphabet Σ is a subset of Σ^* , that is, a set of strings of symbols from Σ .

Definition 3.1.2. A **counting problem** is a function $\phi : \Sigma^* \rightarrow \mathbb{N}$.

Definition 3.1.3. Let M be a non-deterministic Turing Machine. A compu-

tational path on an input x is a sequence of configurations which the machine can enter when given x as input. We write $\mathbf{Paths}_M(x)$ for the set of possible computational paths that the machine M can take on input x . We say that a computational path is **accepting** if it finishes in the accepting state. We write $\mathbf{acc}_M(x)$ for the set of accepting paths of the machine M on the input x .

The class $\#P$ formalises the idea of counting the number of accepting paths of non-deterministic polynomial time Turing Machines. In particular, a counting problem is in $\#P$ if it can be described by the number of accepting paths of some non-deterministic polynomial time Turing Machine.

Definition 3.1.4. Let $\#acc_M$ be the function mapping from an input x to the number of accepting paths of the non-deterministic Turing Machine M on input x . So $\#acc_M(x)$ is the number of accepting paths of the machine M on input x . The class $\#P$ consists of all functions $\#acc_M$ for all non-deterministic polynomial time Turing Machines M .

Definition 3.1.5. Let $\#acc_{M,k}$ be the function mapping from an input x to the number of accepting paths of the non-deterministic Turing Machine M on input x modulo k (so $\#acc_{M,k} : \Sigma^* \rightarrow \{0, \dots, k-1\}$ with $\#acc_{M,k}(x) \equiv \#acc_M(x) \pmod{k}$). The class $\#_kP$ consists of all functions $\#acc_{M,k}$ for all non-deterministic polynomial time Turing Machines M .

Given a counting problem in $\#P$, say $\#A$, we write $\#_kA$ for the $\#_kP$ problem of determining the number of solutions to A modulo k . So where $\#A : \Sigma^* \rightarrow \mathbb{N}$ is a function defined from strings to the natural numbers,

$\#_k A : \Sigma^* \rightarrow \{0, \dots, k-1\}$ is the function defined from strings into the integers modulo k with $\#_k A(x) \equiv \#A(x) \pmod{k}$.

Definition 3.1.6. Let F be a boolean formula, then we define $SAT(F)$ to be the set of satisfying assignments of F , so that $|SAT(F)|$ is the number of satisfying assignments of F .

Example 3.1.7. We define $\#SAT$ to be the problem: given a boolean formula F in conjunctive normal form, compute $|SAT(F)|$.

Example 3.1.8. We define $\#_k SAT$ to be the problem: given a boolean formula, F in conjunctive normal form, compute $|SAT(F)|$ modulo k .

For each k , we define $Mod_k P$, which in contrast to $\#P$ is a class of languages. $Mod_k P$ can answer questions of the form “Does this problem have a non-zero number of solutions modulo k ?”. What we mean by this is that for every language $L \in Mod_k P$ there is some Turing Machine M for which the number of accepting paths on an input x is congruent to 0 (mod k) exactly when $x \in L$.

Definition 3.1.9. A language L is in $Mod_k P$ if there exists a Turing Machine M such that:

$$x \in L \iff \#acc_{M,k}(x) \not\equiv 0 \pmod{k} \quad (3.1)$$

Analogously to $\#_k A$, we associate with a counting problem $\#A \in \#P$ the decision problems $Mod_k A$, where $Mod_k A$ is the decision problem “does A have a non-zero number of solutions modulo k ?”. So $Mod_k A(x)$ returns True if $\#A(x) \not\equiv 0 \pmod{k}$ and False otherwise.

Example 3.1.10. We define $\text{Mod}_k \text{SAT}$ to be the problem: given a boolean formula, F in conjunctive normal form, determine whether $|\text{SAT}(F)| \not\equiv 0 \pmod{k}$.

It should be noted that previous papers have used both $\#_k P$ and $\text{Mod}_k P$ to refer to the decision class defined above as $\text{Mod}_k P$.

3.1.1 Completeness

We will need the notion of a reduction which is parsimonious *modulo* k : just as a parsimonious reduction from one counting problem to another is one which preserves the number of solutions exactly, so a reduction which is parsimonious *modulo* k is one which preserves exactly the number of solutions *modulo* k .

Definition 3.1.11. Given two counting problems $\#A$ and $\#B$, we say there is a ***parsimonious reduction*** from $\#A$ to $\#B$ if there exists a function f , computable in polynomial time, such that for all possible input x , $A(x) = B(f(x))$, i.e. f is a transformation of an instance of A into an instance of B with the same number of solutions.

Definition 3.1.12. Given two $\#_k P$ counting problems, $\#_k A$ and $\#_k B$, we say there is a ***parsimonious $\#_k$ -reduction*** from $\#A$ to $\#B$ if there exists a polynomial-time computable function f such that $A(x) \equiv B(f(x)) \pmod{k}$. In this case we say $\#A \leq_{\#_k} \#B$.

We note here that a reduction which is parsimonious is also parsimonious *modulo* k for all k .

Again, in an analogy with $\#P$ -completeness, we define the notion of $\#_kP$ -completeness with respect to polynomial-time Turing reducibility. Essentially, a problem $\#_kA$ is $\#_kP$ -complete if every problem in $\#_kP$ can be solved in polynomial time given an oracle for $\#_kA$.

Definition 3.1.13. *We say that a problem A is **polynomial-time Turing reducible** (or just *Turing reducible*) to a problem B if every instance of problem A can be solved in polynomial time using an oracle for problem B . We write $A \leq_p^T B$.*

Definition 3.1.14. *A counting problem A is **$\#_kP$ -hard** if, for every problem B in $\#_kP$, $B \leq_p^T A$, i.e. if every problem in $\#_kP$ is polynomial-time Turing reducible to A .*

Definition 3.1.15. *A counting problem $\#_kA$ is **$\#_kP$ -complete** if A is in $\#_kP$ and A is $\#_kP$ -hard.*

When discussing problems of $\#_kP$ complexity, we will need an example of a natural $\#_kP$ -complete problem. As one might expect, such an example is given by the problem of counting the number of solutions to a SAT formula *modulo* k . In [Sim75], Simon showed that the reduction in Cook's theorem can be made parsimonious (and hence parsimonious *modulo* k for all k). This immediately gives the fact that $\#_k\text{SAT}$ is $\#_kP$ -complete for all k .

Simon proved the following.

Theorem 3.1.16. *[Sim75] There exists a parsimonious reduction from any problem in NP to SAT .*

And we have the following two corollaries of this theorem.

Theorem 3.1.17. *For all k , $\#_k \text{SAT}$ is $\#_k P$ -complete.*

Proof. Follows directly from Theorem 3.1.16, as any parsimonious reduction is also parsimonious *modulo* k for all k . \square

In the next section, we will also be interested in questions of $\text{Mod}_k P$ completeness, which we also define in terms of polynomial time Turing reductions: so a problem in $\text{Mod}_k P$ is $\text{Mod}_k P$ complete iff every other problem in $\text{Mod}_k P$ is polynomial time Turing reducible to it. It follows from Simon's results that $\text{Mod}_k \text{SAT}$ is $\text{Mod}_k P$ -complete for all k .

Theorem 3.1.18. *[Sim75] For all k , $\text{Mod}_k \text{SAT}$ is $\text{Mod}_k P$ -complete.*

Proof. Follows directly from Theorem 3.1.16, as any parsimonious reduction is also parsimonious *modulo* k for all k . \square

3.1.2 $\#_k P$ vs. $\text{Mod}_k P$

Whilst, as we will see, there can be differences between the $\#_k P$ version and the $\text{Mod}_k P$ version of a problem for some k , we will show in this section that there are important structural similarities. In fact, only one notion of hardness is needed for both classes. A decision problem which is $\text{Mod}_k P$ -hard is also $\#_k P$ -hard. In other words, an oracle for any $\text{Mod}_k P$ -complete problem can be used to solve any problem in $\#_k P$. We will use some results of Beigel and Gill on the closure properties of $\#P$, the following lemma can be found in [BG92].

Lemma 3.1.19. *If $f(x) \in \#P$ then $f(x) + 1 \in \#P$.*

Proof. Given a polynomial-time Turing machine M such that $f(x)$ is the number of accepting paths of M on input x , we build a Turing machine which behaves as following on input x .

1. Guess 1 or 2
2. If the guess in step 1 was 1, accept.
3. If the guess in step 1 was 2, guess an element of $\text{Paths}(M, x)$, accept iff this is an accepting path.

It is obvious that this machine has exactly one more accepting path than M on any input. □

We now show that the notion of hardness we get from considering the Mod_kP classes is enough to capture all of the power of the $\#_k\text{P}$ classes. In fact, $\#_k\text{P}$ is contained in the closure of Mod_kP under polynomial time Turing reductions. Which it to say that a problem which is hard for Mod_kP is already hard for $\#_k\text{P}$.

Theorem 3.1.20. *Any problem in $\#_k\text{P}$ is polynomial-time Turing reducible to Mod_kSAT .*

Proof. By Theorem 3.1.18, we can use an oracle for Mod_kSAT to determine whether $f(x) \not\equiv 0 \pmod{k}$ in polynomial time for any input x and any function $f \in s\#P$.

On the other hand, using Lemma 3.1.19 we can also use the oracle to decide whether $f(x) + 1 \not\equiv 0 \pmod{k}$ and, iteratively, $f(x) + 2, f(x) + 3, \dots, f(x) +$

$k - 1$. It is clear that the answer to exactly one of these questions will be no, and the value of $f(x) \pmod{k}$ will be uniquely determined by solution to the congruence $f(x) + j \equiv 0 \pmod{k}$, which implies $f(x) \equiv k - j \pmod{k}$.

So, using an oracle for Mod_kSAT , we can determine the value *modulo* k of an arbitrary function in $\#P$ in polynomial time, and Mod_kSAT is indeed $\#_kP$ -hard, as required. \square

It seems intuitively that there should be problems for which determining the number of solutions *modulo* k exactly is harder than deciding whether the number of solutions *modulo* k is non-zero. We give here an example of such a problem. This problem is relatively artificial, leaving open the question of whether there are ‘natural’ problems with this property.

The idea here is to find a problem in $\#P$ for which the number of solutions is always non-zero *modulo* 3, but for which deciding whether the answer is congruent to 1 or 2 is $\#_3P$ -hard. So the $\#_3P$ version of this problem will be $\#_3P$ -complete, but the Mod_3P version of the problem will be trivial. We use the following fact of arithmetic.

Lemma 3.1.21. *Take any $x \in \mathbb{Z}$, then $x^2 + 1 \not\equiv 0 \pmod{3}$.*

Proof. Clearly it suffices to consider the three cases $x \equiv 0, x \equiv 1$ and $x \equiv 2 \pmod{3}$, but $0^2 + 1 \equiv 1 \pmod{3}$, $1^2 + 1 \equiv 2 \pmod{3}$ and $2^2 + 1 \equiv 2 \pmod{3}$, none of which are congruent to zero. \square

Definition 3.1.22. *Let $\#\text{SATsquaredPlusOne}$ be the function which maps F to $|\text{SAT}(F)|^2 + 1$ where F is a SAT formula and $\text{SAT}(F)$ is the set of satisfying assignments of F .*

Lemma 3.1.23. *$\#SAT^2$ is in $\#P$.*

Proof. We know that $\#SAT$ is in $\#P$. Now, given a SAT formula F , we can create a new SAT formula F^2 such that $|\text{SAT}(F^2)| = |\text{SAT}(F)|^2$ as follows. Assume F uses the variables x_1, \dots, x_k . Then F^2 uses variables x_1, \dots, x_k and y_1, \dots, y_k with the clauses of F^2 being copies of each of the clauses in F , along with copies of each of the clauses in F with y_i in place of x_i , so that, for example, if $x_i \vee x_j \vee \bar{x}_k$ is a clause in F then F^2 contains clauses $x_i \vee x_j \vee \bar{x}_k$ and $y_i \vee y_j \vee \bar{y}_k$. We claim that F^2 has $|\text{SAT}(F)|^2$ satisfying assignments: the number of ways of satisfying the clauses containing x_i in F^2 is equal to the number of satisfying assignments of F , as is the number of ways of satisfying the clauses containing y_i , picking any two of these gives a satisfying assignment of F^2 and any satisfying assignment of F^2 must satisfy both of these sets of constraints.

Clearly F^2 can be constructed from F in polynomial time, so the function which maps F to $|\text{SAT}(F)|^2$ is in $\#P$. On other hand, we know that if a function $f(x)$ is $\#P$ then $f(x) + 1$ is in $\#P$ by 3.1.19. \square

Theorem 3.1.24. *$\#_3SAT^2$ is $\#_3P$ -hard.*

Proof. As noted in Theorem 3.1.18 Mod_3SAT is Mod_3P -complete. We will show that an oracle for $\#_3SAT^2$ can be used to solve Mod_3SAT in polynomial time and then, using Theorem 3.1.20 we will be able to show that $\#_3SAT^2$ is hard for $\#_3P$.

Consider an arbitrary SAT formula F . If $|\text{SAT}(F)|^2 + 1 \equiv 1 \pmod{3}$, then the number of satisfying assignments of F is congruent to 0 modulo 3, and if

$|\text{SAT}(F)|^2 + 1 \equiv 2 \pmod{3}$, then $|\text{SAT}(F)| \equiv 1 \pmod{3}$ or $|\text{SAT}(F)| \equiv 2 \pmod{3}$, as in the proof of Lemma 3.1.21. So, given the value of $|\text{SAT}(F)|^2 + 1 \pmod{3}$, we can determine whether or not $|\text{SAT}(F)| \not\equiv 0 \pmod{3}$, but F was an arbitrary SAT formula, so this is exactly equivalent to solving Mod_3SAT , and $\#_3\text{SAT}^2 + 1$ is $\text{Mod}_3 P$ -hard, and so $\#_3 P$ -hard by Theorem 3.1.20.

□

On the other hand, $|\text{SAT}(F)|^2 + 1 \not\equiv 0 \pmod{3}$ by Lemma 3.1.21, so $\text{Mod}_3\text{SAT}^2 + 1$ is trivial.

3.2 Relations between $\#_k P$ classes

Here we discuss relations between the different $\#_k P$ classes for $k \in \mathbb{N} \setminus \{1\}$. Using some surprising and powerful results from Beigel and Gill [BG92] along with the Chinese Remainder Theorem, we show that, in essence, there is only one notion of hardness for all of p, p^2, p^3, \dots for each p prime, and that these are the only notions of hardness with which we need concern ourselves.

We begin with the following straightforward observation, essentially: the ability to count *modulo* some number k allows us to count *modulo* any of its factors.

Lemma 3.2.1. *For all natural numbers a, b , if $a \mid b$ then any problem in $\#_a P$ can be solved in polynomial time with an oracle for any $\#_b P$ -hard problem.*

Proof. By Theorem 3.1.17, $\#_a \text{SAT}$ is complete for $\#_a P$, but given an instance of SAT , F , we can compute the number of solutions to $F \pmod{b}$ in polynomial

time using our oracle for a $\#_bP$ -hard problem, but then the number of solutions *modulo* a is uniquely determined by the number of solutions *modulo* b . \square

We will now prove the main result of this section: that if we want to prove a hardness result for all *moduli* it suffices to prove hardness *modulo* p for p prime. First, the Chinese Remainder Theorem shows that the ability to calculate *modulo* the prime power factors of a number is essentially the same as the ability to calculate *modulo* that number.

Lemma 3.2.2. *[The Chinese Remainder Theorem] For any co-prime natural numbers $\{n_1, n_2 \dots n_k\}$ and any set of integers $\{a_1, a_2 \dots a_k\}$, there exists a unique solution, modulo $\prod_{i=1}^k n_i$ to the set of congruences*

$$x \equiv a_i \pmod{n_i}, \quad i \in 1, \dots, k \quad (3.2)$$

Furthermore, given the a_i and the n_i , there exists an algorithm which can solve the above congruences for x in polynomial time.

Lemma 3.2.3. *Let $a_1 \dots a_n$ be co-prime, then, for any problem A , if $\#_{a_i} A$ are all in FP then $\#_a A$ is in FP , where $a = \prod_{i=1}^n a_i$.*

Proof. By assumption, we can calculate the number of solutions to A *modulo* each of the a_i in polynomial time. We can then apply the Chinese Remainder Theorem to calculate the number of solutions to A *modulo* $\prod_{i=1}^n a_i$ in polynomial time. \square

Lemma 3.2.4. *For any natural number k , if $a_1 \dots a_n$ are co-prime factors of k such that $\prod_{i=1}^n a_i = k$, and $\#_{a_i} A$ is $\#_{a_i}P$ -hard for each of the a_i , then $\#_k A$ is*

$\#_k P$ -hard.

Proof. We give a reduction from $\#_k \text{SAT}$ to $\#_k A$. Assume we have an oracle for $\#_k A$, we will show that this can be used to compute the number of solutions modulo k of a SAT formula, f , in polynomial time.

First, note that an oracle for $\#_k A$ is also an oracle for $\#_{a_i} A$ for all a_i , so we can use the oracle to find the number of solutions to any $\#_{a_i} A$ problem. But $\#_{a_i} A$ is $\#_{a_i} P$ -hard by assumption, so we can compute the number of solutions to f modulo a_i in polynomial time using the $\#_k A$ oracle as a $\#_{a_i} A$ oracle. But then once we know the number of solutions to f modulo each of the a_i , we can apply the Chinese Remainder Theorem to compute the number of solutions to f modulo k . \square

On the other hand, in [BG92], Beigel and Gill show that $\text{Mod}_p P$ is equal to $\text{Mod}_{p^l} P$ for all primes p and all integers l .

Theorem 3.2.5. [BG92] For all primes p and all integers l , $\text{Mod}_p P = \text{Mod}_{p^l} P$.

And so, using Theorem 3.1.20 and the above result, we see that an oracle for $\text{Mod}_p P$ can be used to solve any problem in $\#_{p^l} P$ for all primes p and integers l .

Since it is obvious that an oracle for $\#_p P$ can be used as an oracle for $\text{Mod}_p P$, we have the following theorem.

Theorem 3.2.6. For a given $\#_k P$ problem A , if $\#_{p_i} A$ is $\#_{p_i} P$ -hard for all prime factors, p_i of some integer k , then $\#_k A$ is $\#_k P$ -hard.

Proof. Given that the problems $\#_{p_i}A$ are all hard for all of the p_i , we get hardness *modulo* all powers of the p_i from 3.2.5 and 3.1.20, and then hardness *modulo* k from Lemma 3.2.4. \square

So, if we can prove hardness *modulo* the prime factors of a number, this suffices to prove hardness *modulo* the number itself and, in fact, vice versa - an algorithm which computes the number of solutions *modulo* all of the prime factors of a natural number also allows us to compute the number of solutions *modulo* the number itself.

3.3 Modular Counting Problems

Part of the reason for interest in modular counting comes from the fact that the relationship between $\oplus P$ and NP is ambiguous. There are whole classes of problem for which determining the parity is easy, but counting the number of solutions exactly is $\#P$ -hard. On the other hand, as we will see in the next section, $\oplus P$ appears to be quite a powerful class of problems. There are several classical results, and one more recent result, which suggest that modular counting classes are objects of interest.

In particular, we have the following two results, from Valiant's 1978 paper, in which he first introduced the idea of modular counting problems.

Theorem 3.3.1. [Val78] *Computing the permanent of a 0-1 matrix is $\#P$ -hard.*

Note that Valiant actually proved more than this: computing the permanent of a 0-1 matrix *modulo* k for $k \neq 2^n$ is as hard as recognising unique solutions to

NP problems. On the other hand, Valiant also makes the following observation, which follows from the fact that the permanent of a matrix is equal to its determinant *modulo 2*.

Theorem 3.3.2. [Val78] *Computing the parity of the permanent of a 0-1 matrix can be done in polynomial time.*

There are a variety of other problems for which parity can be computed in polynomial time, but for which the corresponding #P problem is #P-complete. A whole class of such problems is given by the following theorem, which states that the parity of the Tutte polynomial, $T(G; a, b)$, of any graph, G , at any integer points a, b can be calculated in polynomial time.

Theorem 3.3.3. [Wel93] *For integers a, b and any graph G , the parity of $T(G; a, b)$ can be calculated in polynomial time.*

On the other hand, Annan [Ann94] and Goodall [Goo04] showed that computing the value of the Tutte polynomial *modulo* primes other than 2 cannot be done in polynomial time, except at a few special points, unless $\text{RP} = \text{NP}$.

As Valiant noted in [Val05], there are not a great deal of hardness results for modular counting complexity for problems for which the corresponding counting problem is known to be #P-hard, but not via parsimonious reductions from SAT. In [Val05], he demonstrated that several problems were $\oplus\text{P}$ -hard: counting solutions to Monotone SAT problems, counting solutions to SAT formulae in which each variable appears once negated and once non-negated, and counting Hamiltonian circuits in planar graphs of regular degree 3. Note that there

cannot exist a parsimonious reduction from SAT for the first of these problems unless $P = NP$, as the corresponding decision problem is in P .

In [Val06], Valiant proved the following pair of, perhaps counter-intuitive, theorems, giving an example of a natural problem in which the number of solutions *modulo 7* can be computed in polynomial time, whereas the corresponding problem *modulo 2* is $\oplus P$ -hard. So, in particular, the complexity of a modular counting problem can depend, for non-trivial reasons, on the characteristic of the field in which we want to evaluate it.

Definition 3.3.4. *$\#Pl-3/2Bip-VC$ is the problem of counting the number of vertex covers in a planar, bipartite graph which is bi-regular, with vertices of degree 3 on one side and vertices of degree 2 on the other side.*

Theorem 3.3.5. [Val06] *There exists a polynomial time algorithm for $\#_7Pl-3/2Bip-VC$.*

Theorem 3.3.6. [Val06] *The problem $\oplus Pl-3/2Bip-VC$ is $\oplus P$ -complete.*

Finally, there have been two more recent dichotomy results for modular counting in problems related to CSP. In [GLV11] Guo, Lu and Valiant proved a dichotomy theorem for the parity version of the symmetric Boolean Holant problem. Holant is a framework which generalises CSP and Graph Homomorphism, in particular, allowing one to express matching. They showed that there are four types of polynomial-time tractable problems, one of which is the vanishing type, *i.e.* those for which there are always zero solutions *modulo 2*. In all other cases, determining the value of a parity Holant function is $\oplus P$ -complete.

In [GHLX11], the authors extended the main result of Chapter 5 of this thesis, Theorem 5.3.11, to the question of modular weighted Boolean CSP. They also utilise heavily the work of Cai *et al* on the weighted version of the problem for exact counting in [CCL10]. Similar to normal weighted CSP, a weighted modular CSP consists of functions from the domain into the field of size k rather than just relations on the domain.

Theorem 3.3.7. [GHLX11] *Let $k > 1$ and \mathcal{F} be a set of functions. Then $\#_k \text{CSP}(\mathcal{F})$ is either in FP or $\#_p \text{P}$ -hard for some p with $p \mid k$.*

3.4 The Power of Modular Counting

We now turn to a slightly different topic, the question of the power of these modular counting classes. In particular, we will discuss the results of Valiant and Vazirani [VV86] and Toda [Tod91].

The Valiant-Vazirani theorem implies that any NP problem can be recognised with a BPP machine equipped with an oracle for $\oplus\text{P}$. If we are allowed randomization, then $\oplus\text{P}$ is at least as hard as NP. The actual result proved by Valiant and Vazirani was summarised in the title of their paper, *NP is as easy as detecting unique solutions*. They gave a randomised reduction from any problem in NP to a promise version of UP, which is the class of NP problems with at most one solution. An instance of the promise version of a problem is an instance of the problem along with a promise that the instance is in a given class. In this case, the algorithm would be given an NP problem along with a promise that it had at most one solution. The algorithm should behave

correctly on input where the promise is true, but may do anything otherwise.

Theorem 3.4.1. [VV86] *If there exists a polynomial time algorithm for solving problems in promise-UP, then $NP = RP$.*

It is an easy corollary of the Valiant-Vazirani Theorem that any problem in NP can be reduced to a problem in $\oplus P$ by randomised reductions, since any machine which can determine the parity of the number of solutions to a problem can certainly give the correct answer to any problem whose number of solutions is known to be 0 or 1.

Toda's Theorem makes a more powerful statement about the power of a probabilistic Turing Machine equipped with a parity oracle. It says, in essence, that any such machine can solve any problem in the polynomial hierarchy. We give the following definition.

Definition 3.4.2. *For any class of \mathbf{C} , we define $BP.\mathbf{C}$ to be the class of sets L such that for some set $A \in \mathbf{C}$, some polynomial p , some $\epsilon > 0$, and all $x \in \Sigma^*$,*

$$x \in L \implies \Pr\{w \in \{0, 1\}^{p(|x|)} : x\#w \in A\} \geq \frac{1}{2} + \epsilon \quad (3.3)$$

$$x \notin L \implies \Pr\{w \in \{0, 1\}^{p(|x|)} : x\#w \in A\} \leq \frac{1}{2} - \epsilon \quad (3.4)$$

So, for a class of sets \mathbf{C} , $BP.\mathbf{C}$ is the class of problems from which there exist probabilistic polynomial time reductions to problems in \mathbf{C} . Then Toda's Theorem says:

Theorem 3.4.3. [Tod91] *The polynomial hierarchy, PH, is contained in the class $BP.\oplus P$*

In other words, any problem in the Polynomial Hierarchy can be reduced, via randomised reductions, to some problem in $\oplus P$. As Welsh remarks in [Wel93], this seems to be quite a powerful result: it is by no means obvious that an oracle for $\oplus P$ without randomness can be used to settle questions in NP, but with randomness, much more can be achieved.

Chapter 4

$\#_k$ Independent Set

An independent set in a graph is a set of vertices such that the induced graph on that set is the empty graph, *i.e.* such that there are no edges between any pair of vertices in the set. The problem of determining whether there is an independent set of size k in a graph is one of Karp's original 21 NP-complete problems [Kar75], and the complexity of the problem is well-studied. In this section we will show that the problems $\#_k$ INDEPENDENT-SET and $\#_k$ BIPARTITE-INDEPENDENT-SET, respectively counting the number of independent sets in a graph and counting the number of independent sets in a graph which is restricted to be bipartite, are both $\#_k$ P-complete for all integer k . These problems will be used as the targets for reductions in later chapters.

Reductions here are from the problems $\#_k$ SAT, which we defined in Chapter 3, namely, the problem of counting the number of satisfying assignments of a boolean formula in conjunctive normal form *modulo* k . We know that this problem is $\#_k$ P-complete for all k from Theorem 3.1.17 in Chapter 3.

The reductions used in the proofs which follow all have the same basic structure. Given a SAT-formula, F , we produce a graph in which the independent sets with a certain property each correspond to a unique satisfying assignment of F , and in which the independent sets which do not have this property can be partitioned into k subsets of equal size; the total number of independent sets of the latter sort therefore being zero *modulo* k . This allows us to produce a formula for the number of independent sets *modulo* k , as described in Lemma 4.1.5.

As discussed in Chapter 3 it will suffice to prove hardness of these counting problems for all prime k , we make use of this fact in proving the hardness of counting independent sets in bipartite graphs (Theorem 4.2.1).

4.1 Independent Set

Definition 4.1.1. *An independent set of a graph G is a set of vertices in $V(G)$ such that there are no edges between any two of them.*

Definition 4.1.2. *We write $\mathcal{I}(G)$ for the set of independent sets of a graph G .*

Definition 4.1.3. *We write $G[X]$, $X \subset V(G)$ for the subgraph of G induced by the set of vertices X . This is the graph with vertex set X and an edge between two elements of X if and only if there is an edge between these two elements in G .*

Definition 4.1.4. *We will use $N(x)$ to represent the neighbourhood in G of a vertex $x \in V(G)$. That is, the set of vertices $\{y \in V(G) \mid (x, y) \in E(G)\}$.*

We will also use $N(X)$ to represent the open neighbourhood of a set of vertices $X \subset V(G)$. That is, the set of vertices $\{y \in V(G) \setminus X \mid \exists x \in X, (x, y) \in E(G)\}$.

In the next lemma, we describe the general structure of the gadgets we will use in the reduction from SAT to INDEPENDENT-SET.

Lemma 4.1.5. *Consider a graph G with the following structure:*

The vertex set of G consists of a set of vertices X , along with n copies of a graph H , $\{H_1, \dots, H_n\}$, each of which contains a distinguished vertex $h_i \in V(H_i)$. Note that a different h_i could be chosen for each H_i , although in practice we will use the same vertex of H as the distinguished vertex in all of our reductions. The edges in G either go between vertices in one copy of H , between vertices in X or between some distinguished vertex h_i and a vertex in X . Furthermore, H has the property that the total number of independent sets in H is congruent to zero modulo k .

The total number of independent sets in G is congruent modulo k to:

$$\sum_{I_0 \in \mathcal{I}(G[X])} \prod_{i=1}^n [\min\{|I_0 \cap N_G(h_i)|, 1\} \times |\mathcal{I}(H_i \setminus \{h_i\})|] \quad (4.1)$$

Proof. The relevant intuition for this proof is that if we have two sets of vertices, say X and Y , satisfying $N_G(X) \cap Y = \emptyset$ then $|\mathcal{I}(G[X \cup Y])| = |\mathcal{I}(G[X])| \times |\mathcal{I}(G[Y])|$. This is because any independent set which lies entirely in $X \cup Y$ is the union of an independent set whose vertices are contained in X and an independent set whose vertices are contained in Y , and each such union is an independent set of $X \cup Y$.

We note that if $J \in \mathcal{I}(G)$ is an independent set in G then $I = J \cap X$ is an independent set in $G[X]$. We partition the independent sets of G according to their intersection with X - we then count the number of independent sets in each partition *modulo* k and take the sum.

Let I be an independent set in X and let $[I]$ denote the set of independent sets in G whose intersection with X is I . Now we consider two cases.

First, assume that there is some subgraph H_i such that the neighbourhood of H_i does not share any vertices with I (*i.e.* such that $I \cap N_G(H_i) = \emptyset$). Now, any independent set in $[I]$ can be written as the union of an independent set in H_i and an independent set in $G \setminus H_i$ the intersection of which with X is I . Furthermore, every such union is an independent set in $[I]$. Then the total number of independent sets in $[I]$ is congruent *modulo* k to $|\mathcal{I}(H_i)|$ multiplied by the number of independent sets in $G \setminus H_i$ whose intersection with X is I , but since $|\mathcal{I}(H_i)|$ is congruent to zero *modulo* k , then $|[I]|$ is congruent to zero *modulo* k . Note that in this case the product term in the summation above always evaluates to zero - giving a correct count *modulo* k of the size of $[I]$.

Now, assume that for all i , the neighbourhood of H_i (and therefore the neighbourhood of h_i) does contain some vertex in I ($I \cap N(H_i) \neq \emptyset$). Then any independent set in $[I]$ can be written as the union of I and n different independent sets $\{J_1, \dots, J_n\}$ such that J_i is entirely contained in $V(H_i \setminus \{h_i\})$ and, once again, each such union is an independent set in $[I]$. The total number of such unions is clearly $\prod_{i=1}^n |\mathcal{I}(G; H_i \setminus \{h_i\})|$, and since the minimum of $|I \cap N_G(H_i)|$ and 1 is equal to 1 for all i in this case, this is equal to the product given in the theorem statement. \square

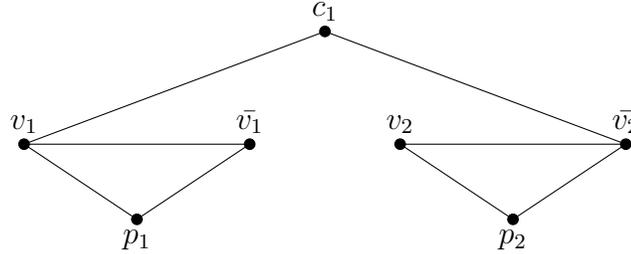
Theorem 4.1.6 which we prove next is in fact a consequence of Theorem 4.2.1 which we prove below. However, since the reduction used here is probably easier to follow, and is similar in structure to that used in the later proof, we will give the construction of this reduction explicitly.

Theorem 4.1.6. \oplus INDEPENDENT-SET is \oplus P-complete

Proof. We proceed by reduction from \oplus SAT. Given a CNF formula F with clauses $\{C_1, \dots, C_m\}$ and variables $\{x_1, \dots, x_n\}$, considered as an instance of \oplus SAT, we construct a graph, G , with vertices $\{v_i, \bar{v}_i, p_i \mid i = 1, \dots, n\}$, corresponding to each variable in F . There are also vertices $\{c_j \mid j = 1, \dots, m\}$, each corresponding to one clause in F . There are three types of edges in the graph. Each pair (v_i, \bar{v}_i) is linked by an edge, and each vertex p_i is linked by an edge to both v_i and \bar{v}_i . Finally, a vertex v_i (\bar{v}_i) is linked to a vertex c_j if and only if the literal x_i (\bar{x}_i) appears in the clause C_j . An example of the graph derived from the SAT formula with the single clause $x_1 \vee \bar{x}_2$ is given in figure 4.1. We claim that the parity of the number of independent sets in G is equal to the parity of the number of satisfying assignments of F .

This graph satisfies the conditions of Lemma 4.1.5. The special subgraph H is the graph on one vertex, which has $2 \equiv 0 \pmod{2}$ independent sets as required. The p_i and c_j are the copies of H and the set X is the vertices $v_i, \bar{v}_i, i \in \{1, \dots, n\}$. It therefore suffices for us to show that the independent sets, I , of $G[X]$ which satisfy $I \cap N_G(p_i) \neq \emptyset$ and $I \cap N_G(c_j) \neq \emptyset$ for all i and j are in one-to-one correspondence with the satisfying assignments of F .

We note that an independent set, I , with the required property must contain exactly one of v_i and \bar{v}_i for each i . It must contain at least one in order to

Figure 4.1: Graph derived from formula $x_1 \vee \bar{x}_2$

ensure that p_i has a neighbour in I , and it cannot contain more than one as $(v_i, \bar{v}_i) \in E(G)$. We now consider the assignment of truth values to variables in F given by setting $s(x_i)$ to true if $v_i \in I$ and setting it to false if $\bar{v}_i \in I$. To see that this assignment is satisfying, let C_j be a clause in F , then the vertex c_j has some neighbour in I , which is either v_i or \bar{v}_i for some i , and the literal x_i or \bar{x}_i , which appears in C_j , is set to true by the construction of s .

Conversely, given a satisfying assignment, s of F , then by reversing this process, *i.e.* taking v_i in I iff $s(x_i) = \text{True}$ and $\bar{v}_i \in I$ otherwise, we get an independent set of $G[X]$ such that $p_i, c_j \in N_G(I)$ for all i and j , by the same reasoning as in the previous paragraph.

Now using the formula in Lemma 4.1.5, we see that the number of independent sets of G modulo 2 is equal to the number of satisfying assignments of F modulo 2, giving the desired reduction.

□

Theorem 4.1.7. $\#_k$ -INDEPENDENT-SET is $\#_k P$ -complete for all k

Proof. Whilst it is possible to give a construction along the lines of that given above (the special subgraphs being copies of K_k), this theorem is again an

immediate consequence of theorem 4.2.1, so this time we will not detail the construction explicitly. \square

4.2 Bipartite Independent Set

Theorem 4.2.1. *$\#_k$ BIPARTITE-INDEPENDENT-SET is $\#_k$ P-complete for all k .*

Proof. We begin by noting that it actually suffices to show that the problem of counting *modulo* p is $\#_p$ P-complete for all primes p by Theorem 3.2.6 from Chapter 3. So for this proof, we will assume that k is a prime.

We proceed by reduction from $\#_k$ -SAT. Given a SAT formula F with clauses $\{C_1, \dots, C_m\}$ and variables $\{x_1, \dots, x_n\}$, and a prime k , we construct a graph as described below, which we will call G .

With each variable x_i in F we associate a subgraph of G as follows; the subgraph contains special vertices v_i, \bar{v}_i the presence or absence of which in an independent set will correspond to the truth or otherwise of the literals x_i, \bar{x}_i of F . The graph also has vertices p_i and \bar{p}_i - these are connected to v_i and \bar{v}_i respectively, and are both connected by an edge to one vertex, h_i in H_i . Each of these H_i is a copy of H , a bipartite graph with a distinguished vertex h , having the property that the number of independent sets in H is a multiple of k and that the number of independent sets in $H \setminus \{h\}$ is not a multiple of k (we show in Lemma 4.2.5 that such graphs exist for all prime k). For each variable, there is also another copy of the same graph, H_i^* . The distinguished vertex of H_i^* , h_i^* is linked by an edge to each of v_i and \bar{v}_i . Finally for each clause C_j in

F we add another copy of this bipartite graph H , denoted C_j , one vertex of which, c_j is linked to each of the v_i, \bar{v}_i which represent a literal present in the clause C_j .

Formally, the vertex set of G will be $\{v_i, \bar{v}_i, p_i, \bar{p}_i \mid i = 1, \dots, n\} = X$, along with $\{V(H_i), V(H_i^*) \mid i = 1, \dots, n\}$ and $\{V(C_j) \mid j = 1, \dots, m\}$, where each of H_i, H_i^* and C_j is a copy of H . We then add the edges

$$\{(v_i, p_i), (\bar{v}_i, \bar{p}_i), (p_i, h_i), (\bar{p}_i, h_i), (v_i, h_i^*), (\bar{v}_i, h_i^*) \mid i = 1 \dots n\}$$

and $(v_i, c_j), (\bar{v}_i, c_j)$ such that the literals x_i, \bar{x}_i respectively appear in the clause C_j .

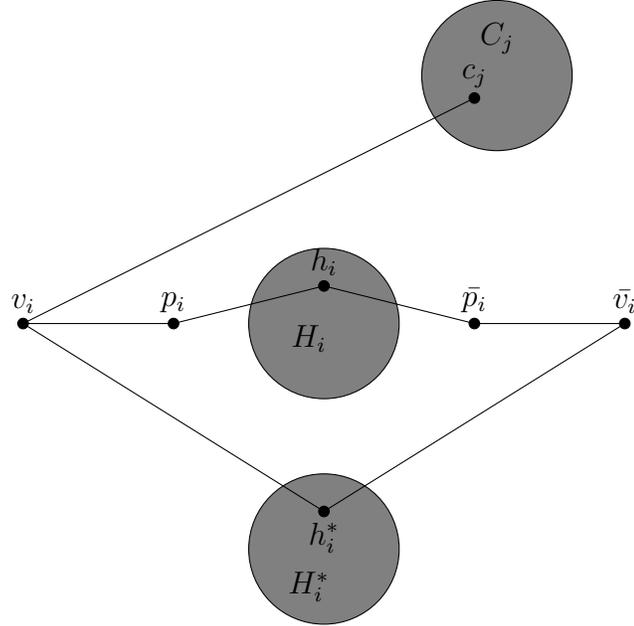
An example of the subgraph associated with a variable x_i such that the literal x_i appears in the clause C_j is given in figure 4.2.

Using Lemma 4.1.5 it suffices to show that the independent sets, I , of G which satisfy the following conditions:

$$\begin{aligned} I &\subset X \\ I \cap N(H_i) &\neq \emptyset, \forall i \\ I \cap N(H_i^*) &\neq \emptyset, \forall i \\ I \cap N(C_j) &\neq \emptyset, \forall j \end{aligned} \tag{4.2}$$

are in one-to-one correspondence with the satisfying assignments of F , and that we can produce a bipartite graph H with the desired property.

Note that each independent set which does obey the conditions above is

Figure 4.2: Subgraph associated with the variable x_i

associated with exactly x^{2n+m} independent sets of G , where x is the number of independent sets in $H \setminus \{h\}$. But since $x \not\equiv 0 \pmod{k}$ by assumption, and k is assumed to be prime, x is invertible *modulo* k , so we can derive the number of satisfying assignments of F *modulo* k from the number of independent sets of G *modulo* p by dividing everything by x^{2n+m} , giving the required reduction. That the relevant H can be constructed for all prime p is shown in Lemma 4.2.5 below.

Let I be an independent set in G satisfying 4.2. Then for all i , either $\{v_i, \bar{p}_i\} \subset I$ or $\{\bar{v}_i, p_i\} \subset I$. To see this, we note that both h_i and h_i^* have some neighbour in I by assumption, but then the only neighbours of h_i in X are v_i and \bar{v}_i , so one of these two must be in I . Similarly, the neighbours of h_i^* in G

are p_i and \bar{p}_i - so one of this pair must be in I , but then since I is independent and $(v_i, p_i), (\bar{v}_i, \bar{p}_i) \in E(G)$ we have the stated result. Let s be the assignment of truth values to variables in F given by $s(x_i) = \text{true} \iff v_i \in I$. We claim that this is a satisfying assignment of F .

Indeed, let C_j be a clause of F . Then there is some element of I which is a neighbour of c_j , the distinguished node in C_j . This is either v_i or \bar{v}_i for some i , but then the literal x_i (\bar{x}_i) appears in the clause C_j , and this literal is true by construction of s , therefore the clause C_j is satisfied.

Similarly, if s is a satisfying assignment of F , then the independent set constructed analogously to that above (with $\{v_i, \bar{p}_i\} \subset I$ if $s(x_i) = \text{true}$ and $\{\bar{v}_i, p_i\} \subset I$ otherwise) is an independent set of X which satisfies the conditions given in 4.2. \square

We now show that a subgraph H which satisfies the conditions given in the proof of the above theorem exists for all prime k . We will appeal to Fermat's Little Theorem.

Theorem 4.2.2 (Fermat's Little Theorem). *Let p be a prime, and let a be an integer coprime to p . Then*

$$a^{p-1} \equiv 1 \pmod{p} \tag{4.3}$$

We will also need basic notions about the multiplicative group of integers modulo p .

Theorem 4.2.3. *For all prime p , the multiplicative group of integers modulo p consists of the congruence classes of all integers which are not multiples of p .*

i.e. if a number is non-zero modulo p then it is also invertible modulo p .

We now prove that for all p it is possible to construct a subgraph with the properties required of H in the above construction. We begin with the following lemma.

Lemma 4.2.4. *The complete bipartite graph $K_{n,n}$ on $2n$ vertices has exactly $2^{n+1} - 1$ independent sets.*

Proof. Any independent set in $K_{n,n}$ is contained entirely in one of the two vertex classes, and every subset of either of the vertex classes is independent. Then there are 2^n independent sets in each class, but the empty set is in both, so there are in fact $2^n + 2^n - 1 = 2^{n+1} - 1$ independent sets in $K_{n,n}$. \square

Lemma 4.2.5. *For all primes p it is possible to construct a bipartite graph H , containing a distinguished node h , with the following properties.*

- (i) *The number of independent sets in H is congruent to zero modulo p .*
- (ii) *The number of independent sets in $H \setminus \{h\}$ is invertible modulo p (i.e. is not a multiple of p).*

Indeed, for $p > 2$, the graph $K_{p-2,p-2}$ is such an H (any node of the graph can be chosen as the distinguished node, since they are indistinguishable).

Example 4.2.6. *An example of a subgraph H which would satisfy the above conditions for $p = 2$ is the graph on one vertex, where the distinguished vertex, h will clearly be the unique vertex in the graph. This graph has precisely 2 independent sets (\emptyset and $\{h\}$), whereas $H \setminus \{h\} = \emptyset$ has precisely one.*

Proof of Lemma 4.2.5. We note that the graph K_1 provides an example of such a graph for $p = 2$ (as explained in Example 4.2.6), so we restrict our attention to the case $p > 2$.

By lemma 4.2.4, $K_{p-2,p-2}$ has $2^{p-1} - 1$ independent sets. But by Fermat's Little Theorem, $2^{p-1} \equiv 1 \pmod{p}$, therefore the number of independent sets of $K_{p-2,p-2}$ is congruent to zero *modulo* p .

On the other hand, the number of independent sets in $K_{p-2,p-3}$ (*i.e.* H with a vertex deleted) is equal to $2^{p-2} + 2^{p-3} - 1$ (by similar arguments to the proof of Lemma 4.2.4), but this is just $(2^{p-1} - 1) - 2^{p-3}$, and since $2 \not\equiv 0 \pmod{p}$, we have that $2^{p-3} \not\equiv 0 \pmod{p}$, and so $(2^{p-1} - 1) - 2^{p-3} \equiv -2^{p-3} \not\equiv 0 \pmod{p}$. \square

Chapter 5

Boolean CSP

5.1 Boolean CSP

The restriction of CSP to a two-element domain, often referred to in the literature as the Generalised Satisfiability Problem, or the Boolean Constraint Satisfaction Problem, still gives a very general class of problems, which provide the base cases for the reductions in a wide variety of complexity theoretic proofs. They were first studied by Schaefer in [Sch78], where he proved a dichotomy theorem for the decision version of these problems, assuming $P \neq NP$.

The Generalised Satisfiability Problem is this: given a set S of boolean relations, the S -satisfiability problem is the question of determining whether or not a given S -formula is satisfiable, where an S -formula is a conjunction of S -relations. The set of all satisfiable S -formulae is denoted by $SAT(S)$. For example, if S were the set of all eight 3-ary boolean relations which can be expressed as a conjunction of three literals, $SAT(S)$ would be the well-known

3-SAT language. Another well-known example is the NP-complete problem 1-in-3 SAT, where S is equal to the three Boolean relations on three variables which are true when exactly one of the variables is true, *i.e.* $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$

Schaefer showed that the decision versions of Generalised Satisfiability Problems can be divided into two classes - those which are NP-complete, and those which can be solved in polynomial time, depending on what type of logical relations are contained in the set S . This is in contrast with a result of Ladner that, under the assumption $P \neq NP$, there is an infinite hierarchy of problems of increasing complexity between problems in P and problems which are NP complete [Lad75].

A dichotomy theorem for the counting version of the Boolean CSP problem was proved by Creignou and Hermann in [CH96]. If a set of relations S is affine, meaning that each relation can be written as a linear equation over $GF(2)$, then $\#SAT(S)$ can be solved in polynomial time using Gaussian elimination. Creignou and Hermann showed that if this is not the case, then $\#SAT(S)$ is $\#P$ -complete. A revised version of their proof appears in the monograph [CKS01], results from which are used in Section 5.3.

Given this counting dichotomy, we are motivated to pose the question: among those Generalised Satisfiability Problems for which the counting problem is known to be $\#P$ -complete, are there any for which the number of solutions is easy to count *modulo* some integer k ? The answer is almost always no. The dichotomy we find in this section is identical to that found in [CH96] except for one small difference caused by symmetries of the solution space in cases where

$k \equiv 2 \pmod{4}$.

5.2 Preliminaries

In [CKS01], the counting dichotomy for Generalised Satisfiability Problems is established via reductions from problems which they refer to as $\#SAT(OR_0)$, $\#SAT(OR_1)$ and $\#SAT(OR_2)$. These are the problems of counting the number of satisfying assignments of boolean formulae whose constraints are defined by relations of the form $x_i \vee x_j$, $\bar{x}_i \vee x_j$ and $\bar{x}_i \vee \bar{x}_j$ respectively. In this paper, we will use essentially the same reductions to find a dichotomy for counting *modulo* k for all integer k . We therefore begin by proving the following $\#_k$ -hardness results.

Theorem 5.2.1. *$\#_kSAT(OR_0)$, $\#_kSAT(OR_1)$ and $\#_kSAT(OR_2)$ are $\#_kP$ -complete for all k .*

This theorem is proved by reduction from $\#_kINDEPENDENT-SET$, the problem of counting the number of independent sets in a general graph *modulo* k , and from $\#_kBIPARTITE-INDEPENDENT-SET$, the problem of counting the number of independent sets in a bipartite graph *modulo* k , both of which are proved to be $\#_kP$ -hard for all k in Chapter 4. The counting reduction from Bipartite Independent set to OR_1 was first given by Linial in [Lin86]. The reduction from $INDEPENDENT-SET$ to $\#OR_2$ is well-known.

We note first that $\#_kSAT(OR_2)$ is trivially reducible to $\#_kSAT(OR_0)$, simply by taking the negation of each literal in the formula. We then make use of the following two lemmas:

Lemma 5.2.2. $\#_k \text{INDEPENDENT-SET} \leq_p^T \#_k \text{SAT}(\text{OR}_2)$ for all k .

Proof. With a graph G on vertices $\{v_1, \dots, v_n\}$ we associate the OR_2 formula F on the variables $\{x_1, \dots, x_n\}$ such that the clause $\bar{x}_i \vee \bar{x}_j$ appears in F if and only if there is an edge between vertices v_i and v_j in G . Then given an independent set I in G , the truth assignment s , which satisfies $s(x_i) = \text{true} \iff v_i \in I$, is a satisfying assignment for F . Similarly given a satisfying assignment for f , the corresponding vertex set (*i.e.* the vertex set which satisfies the same condition, $v_i \in I \iff s(x_i) = (\text{true})$) is independent. So the satisfying assignments of F are in one-to-one correspondence with the independent sets of G , and the reduction is parsimonious.

Since the reduction given above preserves the number of solutions exactly, it preserves the number of solutions *modulo* k for all k . \square

Lemma 5.2.3. $\#_k \text{BIPARTITE-INDEPENDENT-SET} \leq_p^T \#_k \text{SAT}(\text{OR}_1)$ for all k .

Proof. With a bipartite graph G which can be partitioned into sets of vertices $V = \{v_1, \dots, v_n\}$ and $W = \{w_1, \dots, w_m\}$ we associate a OR_1 formula F on the variables $\{x_1, \dots, x_n\}$ and $\{y_1, \dots, y_m\}$. The clause $x_i \vee \bar{y}_j$ appears in F if and only if there is an edge between vertices v_i and w_j in G . Then given an independent set in G , the truth assignment S which satisfies $s(x_i) = \text{False} \iff v_i \in I$ and $s(y_j) = \text{True} \iff w_j \in I$ is a satisfying assignment for F and vice versa, given a a satisfying assignment, the corresponding vertex set is independent. So satisfying assignments of F are in one-to-one correspondence with the independent sets of G , and the reduction is parsimonious.

Note that this reduction is essentially the same as the reduction from independent set to OR_2 , with a vertex being in the independent set having a different truth value on different sides of the bipartition. Since the reduction preserves the number of solutions exactly, it preserves the number of solutions *modulo* k for all k . \square

5.3 The classes $\#_k$ -SAT

We now know that $\#_k\text{SAT}(OR_0)$, $\#_k\text{SAT}(OR_1)$ and $\#_k\text{SAT}(OR_2)$ are $\#_k\text{P}$ -complete for all integer k . We proceed to give reductions from these base problems to Generalised Satisfiability Problems - the reductions are in most cases identical to those used by Creignou et. al. in [CKS01].

We will use the following definitions:

- True and False, the functions of one variable which evaluate to true if and only if the given variable is true (*e.g.* $\text{True}(x)$ evaluates to true iff x is true).
- $XOR(x, y)$, the function which evaluates to true when exactly one of x and y is true and false otherwise.
- Given an element of $\text{GF}(2)^n$, say $x = (x_1, x_2, \dots, x_n)$ we define $1 - x$ to be the element of $\text{GF}(2)^n$ which differs from x in every coordinate.
- A constraint is **C-closed** if the corresponding relation is such that whenever some assignment s satisfies the relation the assignment $1 - s$ also satisfies the relation. In other words, a constraint is C-closed if the set of

satisfying assignments of the constraint is closed under component-wise negation. A set of constraints is \mathcal{C} -closed if every constraint in the set is \mathcal{C} -closed.

- a relation is called **0-valid** (**1-valid**) if it is satisfied by setting all of the variables in the relation to be False (True). A set of relations is 0-valid (1-valid) if every relation in the set is 0-valid (1-valid).
- a constraint set is **affine** if each of the constraints in the set can be expressed as a the solution to a set of linear equations in $\text{GF}(2)$.

Definition 5.3.1. *Let \mathbf{x} and \mathbf{y} be two sets of variables. A family of constraints, \mathcal{F} over \mathbf{x} and \mathbf{y} , **faithfully implements** a boolean function $f(\mathbf{x})$ iff there exists an \mathcal{F} -collection of constraints, \mathcal{C} such that there is exactly one way to satisfy all of the constraints in \mathcal{C} whenever $f(\mathbf{x})$ evaluates to true, and no ways to satisfy them all whenever $f(\mathbf{x})$ evaluates to false. The variables \mathbf{x} are called function variables, and the variables \mathbf{y} auxiliary variables. Note that this means that for any assignment of the variables \mathbf{x} , there is exactly one assignment to the variables \mathbf{y} which satisfies \mathcal{C} if $f(\mathbf{x})$ is true, and no ways of satisfying them all otherwise.*

Example 5.3.2. *The constraint family $\{OR_0, \text{False}\}$ faithfully implements the function True through the constraint applications $\{OR_0(x, y), \text{False}(y)\}$, where y is an auxiliary variable.*

We note that for our purposes a slightly weaker definition of faithful implementation would suffice, with “exactly one” replaced with “congruent to one

modulo k ". However, it turns out that the reductions we need are faithful in the original sense, and therefore we use this definition in order to be able to appeal directly to the results of [CKS01].

Lemma 5.3.3. *Given an integer k and a constraint set \mathcal{F} , if $\#_k\text{SAT}(\mathcal{F})$ is $\#_k P$ -hard and every constraint of \mathcal{F} can be faithfully implemented by \mathcal{F}' , then $\#_k\text{SAT}(\mathcal{F}')$ is also $\#_k P$ -hard.*

Proof. This proof is essentially identical to the proof of Theorem 5.15 in [CKS01]. Given an \mathcal{F} -collection of constraint applications on a variable set \mathbf{x} , say \mathcal{C} , we transform this using faithful implementations to an \mathcal{F}' -collection of constraint applications on a new variable set, (\mathbf{x}, \mathbf{y}) , say \mathcal{C}' . Each clause, C of \mathcal{C} is replaced by a family of \mathcal{F}' constraints which faithfully implements that clause, so the \mathbf{x} variables in \mathcal{C}' are the variables of \mathcal{C} and the \mathbf{y} variables are the auxiliary variables associated with each clause. Since the implementations are faithful, each satisfying assignment of \mathcal{C} can be extended in a unique way to a satisfying assignment of \mathcal{C}' . Therefore there is a one-to-one correspondence between satisfying assignments of \mathcal{C} and satisfying assignments of \mathcal{C}' . This gives a parsimonious reduction from $\#\text{SAT}(\mathcal{F})$ to $\#\text{SAT}(\mathcal{F}')$, which clearly implies the desired result. \square

We will make use of the following lemmas, taken from [CKS01] and stated here without proof.

Lemma 5.3.4. [CKS01] *If a constraint family \mathcal{F} is not 0-valid (1-valid) and*

(i) if \mathcal{F} is C -closed, then \mathcal{F} faithfully implements XOR.

(ii) if \mathcal{F} is not C -closed, then \mathcal{F} faithfully implements True (False).

Lemma 5.3.5. [CKS01] Take a function f . If f is not affine, then $\{f, \text{False}, \text{True}\}$ faithfully implements at least one of the three functions OR_0 , OR_1 and OR_2 . Furthermore, if f is 0-valid (1-valid) then $\{f, \text{False}\}$ ($\{f, \text{True}\}$) faithfully implements one of OR_1 or OR_2 (OR_0 or OR_1).

Lemma 5.3.6. Let \mathcal{F} be a non- C -closed family of functions. Then if \mathcal{F} is both 0-valid and 1-valid, \mathcal{F} faithfully implements OR_1 .

We also need the following lemmas, which have been adapted from the versions given in [CKS01].

Lemma 5.3.7. Let \mathcal{F} be a C -closed set of functions. If p is an odd prime, and if $\#_p \text{SAT}(\mathcal{F} \cup \{\text{False}, \text{True}\})$ is $\#_p P$ -hard, and if \mathcal{F} can faithfully implement the XOR function, then $\#_p \text{SAT}(\mathcal{F})$ is $\#_p P$ -hard.

Proof. We will use the following reduction: Let \mathcal{C} be an $\mathcal{F} \cup \{\text{False}, \text{True}\}$ -collection of constraint applications on variables \mathbf{x} , let y_0, y_1 be two new variables, replace with y_0 any variable constrained to be false, and replace with y_1 any variable constrained to be true. Now add the constraint $XOR(y_0, y_1)$. We now have \mathcal{C}' , an $\mathcal{F} \cup XOR$ collection of constraint applications on variables \mathbf{x}, y_0, y_1 . Clearly any satisfying assignment of \mathcal{C} can be extended to a satisfying assignment of \mathcal{C}' by setting $s'(y_0) = 0$ and $s'(y_1) = 1$. Conversely, let s' be a satisfying assignment of \mathcal{C}' then either $s'(y_0) = 0$ and $s'(y_1) = 1$, in which case s' restricted to \mathbf{x} is a satisfying assignment of \mathcal{C} or $s'(y_0) = 1$ and $s'(y_1) = 0$, in which case it is easy to check that $s(x) = 1 - s'(x)$ satisfies all constraints in \mathcal{C} . So \mathcal{C}' has precisely twice as many satisfying assignments as \mathcal{C} .

Now since p is prime and $p \geq 3$, we can divide by two *modulo* p , giving a Turing reduction from $\#_p\text{SAT}(\mathcal{F} \cup \{\text{False}, \text{True}\})$ to $\#_p\text{SAT}(\mathcal{F} \cup \text{XOR})$. Finally, since \mathcal{F} can faithfully implement XOR, we have $\#_p$ P-hardness of $\#_p\text{SAT}(\mathcal{F})$ by Lemma 5.3.3. \square

Lemma 5.3.8. *Let \mathcal{F} be a C-closed set of functions. For all integer $l > 1$, if $\#_{2^{l-1}}\text{SAT}(\mathcal{F} \cup \{\text{False}, \text{True}\})$ is \oplus P-hard and if \mathcal{F} can faithfully implement the XOR function then $\#_{2^l}\text{SAT}(\mathcal{F})$ is \oplus P-hard.*

Proof. The reduction used is the same as in the previous proof. Given a $\mathcal{F} \cup \{\text{False}, \text{True}\}$ -formula, F , we can construct a \mathcal{F} formula, F' , with twice as many satisfying assignments as F . There are only 2^{l-1} values that $|SAT(F')| \pmod{2^l}$ can take, and these are in one-to-one correspondence with the different values that $|SAT(F)| \pmod{2^{l-1}}$ can take, so any algorithm for computing $\#_{2^l}\text{SAT}(F)$ can also be used to compute $\#_{2^{l-1}}\text{SAT}(F)$ in polynomial time. \square

Lemma 5.3.9. *Let \mathcal{F} be a C-closed set of functions. If p is an odd prime, and if $\#_p\text{SAT}(\mathcal{F} \cup \{\text{False}\})$ is $\#_p$ P-hard then $\#_p\text{SAT}(\mathcal{F})$ is $\#_p$ P-hard.*

Proof. We construct a \mathcal{F} formula from a given $\mathcal{F} \cup \{\text{False}\}$ formula by replacing all variables which are constrained to be false with a new variable x_0 . This formula then has twice as many satisfying assignments as the original, and we proceed as in the proof of Lemma 5.3.7. \square

Lemma 5.3.10. *Let \mathcal{F} be a C-closed set of functions. For all integer $l > 1$, if $\#_{2^l}\text{SAT}(\mathcal{F} \cup \{\text{False}\})$ ($\#_{2^l}\text{SAT}(\mathcal{F} \cup \{\text{True}\})$) is \oplus P-hard, and if \mathcal{F} faithfully implements the False (True) function, then $\#_{2^l}\text{SAT}(\mathcal{F})$ is \oplus P-hard.*

Proof. Using the same reduction as in the proof of the previous lemma, and then the same reasoning as in the proof of Lemma 5.3.8 we obtain the desired result. \square

Finally, we require the observation that for any C-closed set of functions, the number of satisfying assignments *modulo* 2 is always congruent to zero – as for any satisfying assignment \mathbf{s} , the assignment $1 - \mathbf{s}$ is also satisfying.

Theorem 5.3.11. *Given a constraint set \mathcal{F} , and an integer k , the problem $\#_k\text{SAT}(\mathcal{F})$ is in FP if \mathcal{F} is an affine family of constraints, or if $k = 2$ and \mathcal{F} is C-closed. It is $\#_{\frac{k}{2}}\text{P}$ -hard if \mathcal{F} is C-closed, $k > 2$ and $k \equiv 2 \pmod{4}$ and it is otherwise $\#_k\text{P}$ -complete.*

Proof. First we note that $\#_k\text{SAT}(\mathcal{F})$ is clearly in $\#_k\text{P}$. Now, if every constraint in \mathcal{F} is affine, then we can consider solving $\#\text{SAT}(\mathcal{F})$ as the problem of solving a system of linear equations over $\text{GF}(2)$; this can be done in polynomial time using Gaussian elimination. Since we can solve $\#\text{SAT}(\mathcal{F})$ in polynomial time, we can clearly solve $\#_k\text{SAT}(\mathcal{F})$ in polynomial time for all k . Also, if \mathcal{F} is C-closed, then clearly \mathcal{F} has an even number of satisfying assignments, so the problem $\#_2\text{SAT}(\mathcal{F})$ is trivial, and can certainly be solved in polynomial time.

Now, consider the situation in which none of these apply. We will prove that $\#_p\text{SAT}(\mathcal{F})$ is $\#_p\text{P}$ -hard for all sets of relations \mathcal{F} and all odd primes p , and that if \mathcal{F} is C-closed then $\#_4\text{SAT}(\mathcal{F})$ is $\oplus\text{P}$ -hard. Now consider any integer k , if k is odd, then we will have hardness *modulo* all of the prime factors of k , and so hardness *modulo* k by Theorem 3.2.6. On the other hand, if k is even and divisible by 4, then we will again have that $\#_k\text{SAT}(\mathcal{F})$ is hard *modulo* each of

the prime factors of k , and so hard *modulo* k . Finally, if k is even and congruent to 2 *modulo* 4, then we will have shown that $\#_k\text{SAT}(\mathcal{F})$ is hard *modulo* each of the *odd* prime factors of k , and so hard *modulo* $\frac{k}{2}$ (as in this case 2 appears exactly once in the prime decomposition of k and so the product of the odd prime factors of k is equal to $\frac{k}{2}$), but we do not show that it is hard *modulo* k .

In the following, suppose \mathcal{F} contains a function, g , which is not affine.

g is neither 0-valid nor 1-valid Then family $\{g, \text{False}, \text{True}\}$ can faithfully implement one of OR_0 , OR_1 and OR_2 (Lemma 5.3.5). Hence by Lemma 5.3.3 and Theorem 5.2.1, $\#_k\text{SAT}(\mathcal{F} \cup \{\text{False}, \text{True}\})$ is $\#_k\text{P}$ -complete for all k . If \mathcal{F} contains a function which is not C-closed, we can faithfully implement False and True by Lemma 5.3.4, so we get $\#_k\text{P}$ -hardness for $\#_k\text{SAT}(\mathcal{F})$. Otherwise we can faithfully implement XOR by Lemma 5.3.4 and we get $\#_p\text{P}$ -hardness for all odd primes p using Lemma 5.3.7, and $\bigoplus\text{P}$ -hardness for $k = 4$ using Lemma 5.3.8.

g is 0-valid but not 1-valid (or vice versa) In this case, $\{g, \text{False}\}$ can faithfully implement one of the functions OR_1 or OR_2 (Lemma 5.3.5). Also g itself can faithfully implement False since it is 0-valid but not 1-valid. Thus \mathcal{F} can faithfully implement one of OR_1 or OR_2 . Then by the Lemma 5.3.3 and Theorem 5.2.1, we get $\#_k\text{P}$ -hardness for $\#_k\text{SAT}(\mathcal{F})$. Note that in this case g itself is not C-closed as $g(0) = \text{true}$ and $g(1) = \text{false}$ so we don't need to deal with the possibility that \mathcal{F} is C-closed.

g is 0-valid and 1-valid Then if g is not C-closed, g can faithfully implement OR_1 (Lemma 5.3.6) which gives $\#_k\text{P}$ -hardness for $\#_k\text{SAT}(\mathcal{F})$, and

so hardness *modulo* all primes. Otherwise $\{g, \text{False}\}$ can faithfully implement one of OR_1 or OR_2 (Lemma 5.3.5), which gives $\#_k\text{P}$ -hardness of $\#_k\text{SAT}(\mathcal{F}, \text{False})$. Therefore we can use Lemmas 5.3.9 and 5.3.10 to get $\#_p\text{P}$ -hardness *modulo* all odd primes, and $\oplus\text{P}$ -hardness *modulo* 4 as above.

□

Chapter 6

Graph Homomorphism

6.1 Graph Homomorphisms

Graph homomorphism is a natural generalisation of graph colouring, in which the restrictions on adjacencies between colours can be more general than in the usual graph colouring problem.. A homomorphism from a graph G to a graph H is an edge-preserving map between the vertices (see Definition 6.1.1). It is sometimes referred to as H -colouring (where the target graph for the homomorphism is H). Graph colouring is the special case of homomorphisms into the complete graph.

Definition 6.1.1. *A homomorphism from a graph G into another graph H is a map $\phi : V(G) \rightarrow V(H)$ satisfying the property that if $(u, v) \in E(G)$ then $(\phi(u), \phi(v)) \in E(H)$.*

Example 6.1.2. *A homomorphism from a graph G into the complete graph K_n is an n -colouring of G .*

Example 6.1.3. *A homomorphism from a graph G into the graph on two vertices with an edge and one loop, H_1 can be considered as an independent set of G . The vertices mapped to the unlooped vertex in H_1 form an independent set (as none of them can be pairwise adjacent) and, conversely, given an independent set, it is possible to map the vertices of the independent set to the unlooped vertex and the vertices of its complement to the looped vertex. So there is a natural one-to-one correspondence between homomorphisms into this graph and independent sets.*

We will also need to discuss the automorphism groups of graphs. There will be particular reference to automorphisms of order 2, or involutions.

Definition 6.1.4. *An automorphism of a graph G is an injective homomorphism from G to itself. In other words, an automorphism of a graph G is a permutation of the vertices of G , ϕ , such that $(\phi(u), \phi(v)) \in E(G) \iff (u, v) \in E(G)$.*

Definition 6.1.5. *An involution of a graph G is an automorphism of order 2. That is, an automorphism σ , of a graph, is an involution if σ is not the identity and for all $v \in V(G)$, $\sigma(\sigma(v)) = v$.*

In this chapter, we will mostly be concerned with the problem of determining whether the number of H -colourings is odd or even (*i.e.* the restriction of the counting version to counting *modulo* 2). Some of our results also apply to other *moduli*, and we will make it clear when this is the case.

We give a dichotomy theorem for counting H -colourings *modulo* 2 in the case where H is a tree: either the associated $\oplus P$ counting problem is $\oplus P$ -

complete or it can be solved in polynomial time. The dichotomy is based on a condition on the structure of the tree and some of its subtrees. We conjecture that the same condition gives a dichotomy for general graphs. In the next section, we give a summary of previous work on the complexity of counting H -colourings in various counting classes.

6.2 H-colouring and Complexity

The complexity of the decision version of the graph homomorphism problem was completely classified by Hell and Nešetřil in [HN90]. For a given graph H , deciding whether an arbitrary graph has a homomorphism into H can be done in polynomial time iff H has a loop or is bipartite. Hell and Nešetřil showed that this decision problem is NP-complete in all other cases.

The problem of exactly counting the number of homomorphisms into a fixed graph H was considered by Dyer and Greenhill in [DG00]; they gave a complete characterisation, again with a dichotomy theorem: the counting problem associated with a graph homomorphism problem is polynomial time solvable if the graph concerned is either a complete graph with loops everywhere or a complete bipartite graph without loops, and it is $\#P$ -complete otherwise.

We denote by $\text{hom}(G, H)$ the total number of homomorphisms from G to H . Note that we will often refer to the vertices of H as ‘colours’, reserving the word ‘vertex’ for vertices of G . As mentioned above, the complexity of exactly counting the number of homomorphisms into a given graph H was characterised by Dyer and Greenhill in [DG00]. They proved the following theorem.

Theorem 6.2.1. *[DG00] If a graph H is a complete bipartite graph with no loops or a complete graph with loops everywhere, then exactly counting H -colourings can be done in polynomial time. Otherwise, the problem is $\#P$ -complete.*

Clearly if the number of homomorphisms into a graph H can be counted exactly in polynomial time, then the parity can be determined in polynomial time. We will show that there are some cases in which symmetries of H can make the related modular counting problem trivial even when the exact counting problem is $\#P$ -hard.

6.3 Pinning colours to vertices

We would like to be able to count the number of colourings of a given graph G in which certain vertices are forced to be coloured with certain colours from H : this would then allow us to reduce the H -colouring problem to colouring with certain subgraphs of the vertices of H , which we could then show to be hard. We achieve this by building gadgets in which only certain sets of colours can be used at certain vertices in the gadgets: these gadgets can then be attached to vertices of G restricting them to be coloured with the same sets of colours.

6.3.1 The Lovász vector of a Graph

We give here a modified version of a theorem from [HN04]. This theorem will allow us to build gadgets to determine the parity of the number of colourings

of graphs in which certain small sets of vertices are restricted to be coloured with specified sets of colours.

In essence, we want to show that for any two distinct colours h_1, h_2 in a given H , there exists some graph with a specified vertex (Γ, γ) such that the number of ways of colouring Γ with γ coloured h_1 is different *modulo 2* to the number of ways of colouring Γ such that γ is coloured h_2 . In fact, as we can see, we can find such Γ for all prime moduli. We can then use these Γ to determine the parity of the number of colourings of some instance graph G in which a given vertex, say v , is coloured with some subset of the colours in H in the following way. We attach a copy of Γ to G , identifying γ and v . If h_i is a colour of H for which there are an even number of colourings of Γ with h_i at γ , then there are an even number of extensions of any colouring of G which uses h_i at v to a colouring of this new graph; similarly, if h_i is a colour such that there are an odd number of colourings of Γ with h_i at γ , then there are an odd number of extensions of any colouring of G which uses h_i at v , so the number of colourings of the new graph is congruent *modulo 2* to the number of colourings of G in which v is coloured with those colours from H for which there are an odd number of colourings of Γ with that colour at γ .

We first prove an exact counting version of the theorem we want to use to build our gadgets. Given two graphs H and H' with specified vertices h and h' , such that there is no isomorphism from H to H' as graphs with specified vertices, *i.e.* no isomorphism from H to H' which maps h to h' , there is some graph G , also with a specified vertex, v , such that the number of homomorphisms from G to H in which v is coloured with h is different from the number

of homomorphisms from G to H' in which v is coloured with h' . To prove this, we use essentially the same strategy as the proof in [HN04]. In fact, we prove the contrapositive.

Before we give the proof, we must formally define the notion of homomorphism between two graphs with specified vertices.

Definition 6.3.1. A **graph with a specified vertex** is a pair (G, v) where G is a graph and $v \in V(G)$ is a vertex of G .

Definition 6.3.2. A homomorphism between two graphs with specified vertices $(G, v), (G', v')$ is a graph homomorphism $\phi : V(G) \rightarrow V(G')$ with $\phi(v) = v'$.

Definition 6.3.3. We denote the number of homomorphisms between a graph with specified vertex (G, g) and another graph with specified vertex (H, h) by $\text{hom}^*((G, g), (H, h))$. If the specified vertex is implied by the context we will sometimes suppress it in the above notation, and just write $\text{hom}^*(G, H)$.

Similarly, we denote the number of injective homomorphisms from a graph with specified vertex (G, g) to (H, h) by $\text{inj}^*((G, g), (H, h))$ and, again, we may suppress the specified vertices if they are implied by the context, instead writing $\text{inj}^*(G, H)$.

In the following, we will also require the concept of the quotient of a graph with respect to a partition of its vertices:

Definition 6.3.4. The **quotient** of a graph G with respect to a partition Θ of its vertices is the graph G/Θ whose vertices are the elements of Θ , with edges between two vertices of G/Θ iff there is an edge between some pair of vertices

which lie in the corresponding sets of vertices of G . If G is a graph with specified vertex, then G/Θ is a graph with specified vertex, and the specified vertex of G/Θ is the vertex corresponding to the set of the partition which contains the specified vertex of G .

Finally, we will use the concept of the Lovász vector of a graph with specified vertex. For us, this will be the vector which counts, for a given graph with specified vertex (H, h) , the number of homomorphisms into (H, h) from every other finite graph with specified vertex.

Definition 6.3.5. *Let G_1, G_2, \dots be a fixed enumeration of all pairwise non-isomorphic graphs with specified vertices (so these graphs are pairwise non-isomorphic, and each graph with a specified vertex is isomorphic to at least one of them). Then we call the Lovász vector of a graph (H, h) the countable sequence $(\text{hom}(G_i, H))_{i \geq 1}$.*

Now, we prove the following theorem, essentially stating that if the number of (H, h) -colourings of every graph with specified vertex is the same as the number of (H', h') -colourings, then (H, h) and (H', h') are isomorphic.

Theorem 6.3.6. *Let H, H' be graphs with specified vertices h and h' , then (H, h) and (H', h') are isomorphic iff for every graph G with specified vertex g we have:*

$$\text{hom}^*(G, H) = \text{hom}^*(G, H') \quad (6.1)$$

The Lovász vector of a graph with specified vertex, as defined in Definition 6.3.5, is the vector which counts homomorphisms into (H, h) from every other

graph. The theorem asserts that this vector is sufficient to reconstruct (H, h) . In fact, we will see from the proof that we only need finitely many terms of the sequence to reconstruct (H, h) .

Proof. Clearly the condition is necessary - two isomorphic graphs have the same Lovász vector. Now we need to prove that it is sufficient. This proof is similar to the proof of Theorem 2.11 in [HN04]. We first observe that, in order to show that H and H' are isomorphic, it is sufficient to prove that for every graph with specified vertex (G, g) :

$$\text{inj}^*(G, H) = \text{inj}^*(G, H') \quad (6.2)$$

To see this, note that if we take $G = H$ in the above, we find there is an injective homomorphism from H to H' (since there is certainly an injective homomorphism from H to itself). Similarly, if we take $G = H'$ we find an injective homomorphism the other way, so this equality allows us to deduce injective homomorphisms from H to H' and from H' to H , and thus an isomorphism between H and H' .

We will prove that equation 6.1 implies equation 6.2 by induction on the size of G . If G only has one vertex then every homomorphism from G to any other graph is injective, so the equality holds. Now assume that the equality is true for all graphs with specified vertices which have fewer vertices than G . The proof strategy is essentially to count those homomorphisms which are not injections, and show that there are the same number of these, so there must be the same number of injective homomorphisms.

The number of homomorphisms is equal to the number of injective homomorphisms plus the number of homomorphisms which are not injective. In order to count the number of homomorphisms which are not injective, we consider the different ways in which the colours of H can be assigned to vertices of G . A colouring of G with H induces a partition of G in the obvious way, with vertices which are given the same colour assigned to the same part of the partition. If we call this partition Θ , then any colouring of G can be considered as an injective colouring of G/Θ , since each vertex of G/Θ is associated with exactly one colour from H by definition. Let i be the partition consisting of a single block for each vertex (*i.e.* the partition associated with injective homomorphisms from G to H). Then we have both:

$$\text{hom}^*(G, H) = \text{inj}^*(G, H) + \sum_{\Theta \neq i} \text{inj}^*(G/\Theta, H) \quad (6.3)$$

$$\text{hom}^*(G, H') = \text{inj}^*(G, H') + \sum_{\Theta \neq i} \text{inj}^*(G/\Theta, H') \quad (6.4)$$

Since G/Θ is necessarily smaller than G if $\Theta \neq i$, we know by the induction hypothesis that $\text{inj}^*(G/\Theta, H) = \text{inj}^*(G/\Theta, H')$ and since $\text{hom}^*(G, H) = \text{hom}^*(G, H')$ by assumption, we do have $\text{inj}^*(G, H) = \text{inj}^*(G, H')$, as required.

Note that the largest G considered in the given inductive argument have the same number of vertices as H , so for two H, H' , there must be two G with at most as many vertices as H, H' for which the number of colourings of G with H is different to the number of colourings of G with H' . \square

Since we are actually interested in the modular counting version of the

graph homomorphism problem, we will use the following version of the above theorem, in which the terms of the Lovász vector are taken *modulo* k .

Definition 6.3.7. *Let G_1, G_2, \dots be a fixed enumeration of all the graphs with specified vertices, then the **mod** k **Lovász vector** of a graph is the countable sequence $(\text{hom}(G_i, H) \pmod{k})_{i \geq 1}$.*

We have proven that the Lovász vector is sufficient to characterise any graph with a specified vertex. We will actually use the following slightly modified version of the theorem.

Lemma 6.3.8. *For any prime p , if two graphs with specified vertices neither of which have an automorphism of order p have the same mod p Lovász vector, they are isomorphic.*

Proof. The only part of the above proof that changes is the part where we claim it is sufficient to prove that $\text{inj}^*(G, H) = \text{inj}^*(G, H')$. Working *modulo* p , we require the assumption that H and H' have no automorphisms of order p in order that the number of injective homomorphisms from H to itself is non-zero *modulo* p . (indeed, as we show below, the graphs which have no automorphisms of order p are exactly those with a non-zero number of injective homomorphisms to themselves *modulo* p). The remainder of the proof is valid if all calculations are done *modulo* p . □

We now demonstrate the fact mentioned in the previous proof, that the graphs with no automorphisms of order p are exactly those with a non-zero number of automorphisms *modulo* p . To do this, we appeal to a general theorem

from group theory, which we apply to the automorphism group of the graph. We will require Cauchy's Group Theorem: that if a prime number divides the order of a group then the group contains at least one element of that order. For a proof, see *e.g.* [McK59]. We will also use a corollary of Lagrange's Theorem: that if a group contains an element of order p then p divides the order of the group (since the subgroup generated by an element of order p is a cyclic group of order p).

Theorem 6.3.9 (Cauchy's Group Theorem). *If a prime p divides the order of a finite group G , then G contains at least one element of order p .*

Theorem 6.3.10 (Lagrange's Theorem). *For any finite group G the order of any subgroup of G divides the order of G .*

Lemma 6.3.11. *For any prime, p , a graph has an automorphism of order p if and only if the order of its automorphism group is divisible by p .*

Proof. The automorphisms of a graph form a group. If this group contains an element of order p , then it is of even order by Lagrange's Theorem. If the order of its automorphism group is divisible by p , then it contains an automorphism of order p by Cauchy's Group Theorem. \square

6.3.2 Building Gadgets

In the following we return to $\bigoplus H$ -colouring, and are only interested in automorphisms of order two, or involutions. It will be useful to consider the case where H and H' have the same underlying graph but different specified vertices

(note that for H and H' to be non-isomorphic as graphs with specified vertices, this requires that there is no automorphism of H which takes h to h' , *i.e.* that h and h' lie in different orbits of the automorphism group of H). Since we will no longer be able to use the previous naming convention for the specified vertices, we will refer to the two specified vertices in H as x and y . In the following, we will be assuming that H is involution-free. As we will see in Section 6.4, it suffices to consider the complexity of $\bigoplus H$ -colouring for involution-free H .

Lemma 6.3.8 allows us to construct the following useful gadgets: given an involution-free graph H and two colours x and y which are in different orbits of $\text{Aut}(H)$, there is a graph Γ with a vertex γ such that the number of H -colourings of Γ with x at γ differs in parity from the number of H -colourings of Γ with y at γ . To prove this, we use the fact that (H, x) and (H, y) have different parity Lovász vectors.

Lemma 6.3.12. *Given an involution-free graph H and two vertices x and y which lie in different orbits of $\text{Aut}(H)$, there exists a graph Γ with specified vertex γ such that $\text{hom}^*((\Gamma, \gamma), (H, x)) \not\equiv \text{hom}^*((\Gamma, \gamma), (H, y)) \pmod{2}$.*

Proof. Since (H, x) and (H, y) are non-isomorphic as graphs with specified vertices, they have different parity Lovász vectors by 6.3.8. Simply take (Γ, γ) to be the first graph with specified vertex for which the corresponding entries of the parity Lovász vectors of (H, x) and (H, y) differ. \square

We will use this gadgetry to get a reduction from the problem of counting H -colourings in which a given vertex of G is forced to be coloured with colours from a specified orbit of $\text{Aut}(H)$ to the problem of counting H -colourings of G

modulo 2.

Theorem 6.3.13. *Given an involution-free graph H , and an oracle for $\bigoplus H$ -colouring, it is possible to determine the parity of the number of H -colourings of a graph G in which a specific vertex of G is coloured with vertices from a given orbit of the automorphism group of H in polynomial time.*

Note that this would be a trivial result if we were able to build a gadget Γ such that $\text{hom}^*((\Gamma, \gamma), (H, x))$ is odd while $\text{hom}^*((\Gamma, \gamma), (H, y))$ is even for all $y \neq x$. Then we could just attach a copy of Γ at the vertex of G that we want to colour with x , identifying this vertex with the specified vertex of Γ and count H -colourings of the new graph. Unfortunately, Lemma 6.3.12 doesn't allow us to construct such a gadget: indeed, it doesn't even guarantee the existence of a gadget with an odd number of colourings in which the specified vertex is coloured x . However, we can construct a series of gadgets which allow us to count colourings in which the given vertex is fixed to a given orbit of colours, essentially by developing a sort of algebra on the gadgets, as described below.

Definition 6.3.14. *With each gadget Γ we associate a vector $v_H(\Gamma)$ indexed by the vertices of H , which we label h_1, \dots, h_n , such that the i^{th} entry of the vector is a 1 if there are an odd number of H -colourings of Γ which use colour h_i at γ and 0 otherwise.*

Note that if two vertices i and j are in the same orbit of the automorphism group of H then the i^{th} and j^{th} entries of $v_H(G)$ are the same for all G . So, in the following we will consider the vectors $v_H^*(G)$ which are indexed by orbits of the automorphism group of H rather than individual vertices of H , the entry of

$v_H^*(G)$ associated with a given orbit being the entry of $v_H(G)$ associated with each of the colours in that orbit.

We define an operation on graphs with specified vertices below. Given two such objects, we will combine them by identifying their specified vertices. We thereby get a new graph with specified vertex. The specified vertex of the new graph is the vertex created by identifying the specified vertices of the original two.

This operation allows us in some sense to take the intersection of the sets of colours, h , such that there are an odd number of colourings of Γ in which the specified vertex is coloured h . This is equivalent to saying that the new gadget we get by intersecting two gadgets has the associated vector obtained by taking the coordinate-wise product of the vectors associated with the individual gadgets.

Definition 6.3.15. *We define the operation $*$: $GF(2)^n \rightarrow GF(2)^n$ to be the coordinate-wise product of two vectors, so the i^{th} entry of $v * w$ is the i^{th} entry of v multiplied by the i^{th} entry of w .*

Definition 6.3.16. *Given a graph with specified vertex (Γ, γ) and another graph with specified vertex (Π, π) , we define the new graph with specified vertex $\Gamma \cdot \Pi$ to be the graph obtained by identifying the specified vertices of each. The specified vertex of $\Gamma \cdot \Pi$ is the vertex formed by the specified vertices of the other two graphs.*

Lemma 6.3.17. *Given a graph with specified vertex (Γ, γ) and a graph with specified vertex (Π, π) along with a graph H , such that (Γ, γ) and (Π, π) have*

vectors $v_H^*(\Gamma)$ and $v_H^*(\Pi)$ respectively, the graph $\Gamma \cdot \Pi$ has associated vector $v_H^*(\Gamma) * v_H^*(\Pi)$.

Proof. If there is a zero in the i^{th} place of either of the vectors $v_H^*(\Gamma)$ or $v_H^*(\Pi)$, then there are an even number of colourings of the respective graph with the colour h_i at the specified vertex. But then there are an even number of colourings of the new graph which use h_i at the specified vertex, as any colouring of the rest of the graph can be extended in an even number of ways to a colouring of the relevant gadget. Alternatively, if there is a 1 in both places, then there are an odd number of extensions of any colouring which uses h_i at the specified vertex, v of $\Gamma \cdot \Pi$ to a colouring of the whole graph, since there are an odd number of colourings of Γ with this property and also an odd number of colourings of Π , and so there is a 1 in the corresponding place of $\Gamma \cdot \Pi$. \square

We now introduce a formal sum on graphs, which preserves addition of these vectors.

Definition 6.3.18. For a set of graphs $\Gamma_1, \Gamma_2, \dots, \Gamma_k$, we define $v(\Gamma_1 + \Gamma_2 + \dots + \Gamma_k)$ to be $v(\Gamma_1) + v(\Gamma_2) + \dots + v(\Gamma_k)$.

Definition 6.3.19. We will say that a vector v is **implementable** for some H if there is a set of gadgets $\{\Gamma_1, \dots, \Gamma_k\}$ such that v is equal to $v_H^*(\Gamma_1 + \Gamma_2 + \dots + \Gamma_k)$.

Lemma 6.3.20. The set of vectors which are implementable for a given H is closed under the operations of vector addition and point-wise multiplication (or the operation $*$, as defined in Definition 6.3.15).

Proof. Given two gadgets Γ_1 and Γ_2 , we can obtain a gadget $\Gamma_1 \cdot \Gamma_2$ with the vector $v(\Gamma_1) * v(\Gamma_2)$ by identifying the specified vertices of the two original gadgets, as in Lemma 6.3.17. Then, given an arbitrary set of gadgets $\Gamma_1, \dots, \Gamma_k$ and Π_1, \dots, Π_l , we have the following:

$$v(\Gamma_1 + \Gamma_2 + \dots + \Gamma_k) * v(\Pi_1 + \Pi_2 + \dots + \Pi_l) \quad (6.5)$$

$$= v(\Gamma_1) * v(\Pi_1 + \Pi_2 + \dots + \Pi_l) + \dots + v(\Gamma_k) * v(\Pi_1 + \Pi_2 + \dots + \Pi_l) \quad (6.6)$$

$$= v(\Gamma_1) * v(\Pi_1) + v(\Gamma_1) * v(\Pi_2) + \dots + v(\Gamma_k) * v(\Pi_l) \quad (6.7)$$

$$= v(\Gamma_1 \cdot \Pi_1) + v(\Gamma_1 \cdot \Pi_2) + \dots + v(\Gamma_k \cdot \Pi_l) \quad (6.8)$$

$$= v(\Gamma_1 \cdot \Pi_1 + \dots + \Gamma_k \cdot \Pi_l) \quad (6.9)$$

□

Lemma 6.3.21. *For any involution free graph, H , the all-ones vector is implementable, and for any pair of orbits in H there is at least one implementable vector which has a 1 at every vertex in one of the two orbits and a 0 at every vertex in the other orbit.*

Proof. The all-ones vector is implementable using the graph on one vertex. The gadgets whose vectors distinguish between distinct orbits of colours in H are obtained using Lemma 6.3.12. □

We now know that the set of implementable vectors is closed under the operations of coordinate-wise addition and coordinate-wise multiplication, that for any pair of colours which are not in the same orbit it contains a vector

which has different entries at these two places, and that it contains the all-ones vector. In the following lemma, we prove that these facts are enough to enable us to count colourings in which a specific vertex is required to be coloured with vertices from a given orbit of $\text{Aut}(H)$.

Lemma 6.3.22. *Consider a set, S , of vectors in $GF(2)^n$ which contains the all-ones vector $(1, 1, \dots, 1)$ and has the property that for any two coordinates i and j there is some vector in the set whose i^{th} coordinate differs from its j^{th} coordinate. The closure of this set under the operations of coordinate-wise multiplication and coordinate-wise addition includes each of the vectors in the standard basis.*

Proof. We proceed by induction. If $n = 1$ the lemma clearly holds, as the all-ones vector is the only vector in the standard basis. Now, assume that the lemma holds for all $k < n$, and we attempt to construct the vectors in the standard basis in $GF(2)^n$.

By induction, we can construct vectors which agree with the standard basis in the first $n - 1$ places, without being able to control what happens in the n^{th} place (note that the restriction of the set of vectors S to the first $n - 1$ places still satisfies the conditions of the lemma). *i.e.* we can certainly obtain vectors

of each of the following forms, where the x_i can be either 0 or 1

$$\begin{aligned} & (1 \ 1 \ 1 \ 1 \ \dots \ 1 \ 1 \ 1) \\ & (1 \ 0 \ 0 \ 0 \ \dots \ 0 \ 0 \ x_1) \\ & (0 \ 1 \ 0 \ 0 \ \dots \ 0 \ 0 \ x_2) \\ & (0 \ 0 \ 1 \ 0 \ \dots \ 0 \ 0 \ x_3) \\ & (0 \ 0 \ 0 \ 0 \ \dots \ 0 \ 1 \ x_n) \end{aligned}$$

This leaves several cases:

Case 1. The x_i are all equal to zero. In this case, we already have the first $n - 1$ vectors from the standard basis, and we can just take the sum of all $n - 1$ vectors with the all-ones vector, which has a 1 in the last place and zeros everywhere else, to get the last one.

Case 2. There are at least two i, j such that $x_i, x_j = 1$. But then the product of these two vectors is the vector $(0, 0, \dots, 0, 1)$. To obtain the remaining vectors from the standard basis, we just take the sum of this vector with any of those from the original list which had a 1 in the n^{th} place, *i.e.* e_i is the sum of this vector with the vector which had a 1 in the i^{th} place and a 1 in the n^{th} place.

Case 3. There is exactly one vector in the list, v with a 1 as the n^{th} entry. Say this vector has a 1 in the i^{th} and n^{th} places. By assumption, there is some vector in S which has different values in the n^{th} and i^{th} places. The product of this with v is a vector with exactly one 1, in either the i^{th} or the n^{th} place, and the sum of this basis vector with v is the other of e_i and e_n . \square

Lemma 6.3.23. *For any involution-free graph H , and any orbit of $\text{Aut}(H)$ the vector which consists of 1s for every vector in the orbit and 0s elsewhere is*

implementable.

Proof. By Lemma 6.3.20 the set of vectors we can implement is closed under the operations of addition and coordinate-wise multiplication; by Lemma 6.3.21 we can implement the all ones vector and, for each pair of orbits, a vector which has 1s in the places corresponding to the vertices in one of the pair and 0s in the vertices corresponding to the other, and by Lemma 6.3.22 the closure of any set with the containing such pairs of vectors under coordinate-wise multiplication and addition contains the basis vectors. \square

Now we need to show that our definition of “implementable” actually does what we want it to do. That is: for any H , if some vector v is implementable by H then it is possible to determine, in polynomial time, the parity of the number of H -colourings of a graph G in which only colours with a 1 in the corresponding place of v are used.

Lemma 6.3.24. *Let H be an involution-free graph, G an arbitrary graph, and v some vector implementable for H , then for any vertex $x \in V(G)$ we can determine the parity of the number of H -colourings of G in which x is coloured with colours such that there is a 1 in the corresponding entry of v .*

Proof. By definition, $v = v_H^*(\Gamma_1 + \Gamma_2 + \dots + \Gamma_k)$ for some set of graphs $\{\Gamma_1, \dots, \Gamma_k\}$. We can count the number of colourings of G which use only colours such that there is a 1 in the corresponding place of $v_H^*(\Gamma_i)$ by taking a copy of G and a copy of Γ_i and identifying x with the specified vertex of Γ_i , getting the new graph $(G, x) \cdot (\Gamma_i, \gamma_i)$, and counting the number of H -colourings of this graph.

To see that the parity of the number of colourings of the resulting graph is the same as the parity of the number of colourings of G which use only the desired colours from H at x , consider colourings which do use one of those colours at x , and those which do not. We claim that for each colouring of the first type, there is an odd number of extensions to a colouring of $(G, x) \cdot (\Gamma_i, \gamma_i)$, and that for each colouring of the second type there is an even number of such extensions.

Take a colouring of G which has a colour at x such that there is a 1 in the corresponding place of $v_H^*(\Gamma_i)$. Then, by definition, there are an odd number of colourings of Γ_i with this colour at γ_i , but then there are an odd number of ways of extending the initial colouring of G to a colouring of $(G, x) \cdot (\Gamma_i, \gamma_i)$. Similarly, if there is a zero in the corresponding place of $v_H^*(\Gamma_i)$, then there are an even number of colourings of Γ_i with that colour at its specified vertex, and so an even number of extensions of the given colouring of G to a colouring of $(G, x) \cdot (\Gamma_i, \gamma_i)$.

So we can determine the parity of the number of colourings of G which use only those colours for which there is a 1 in the corresponding place of $v_H^*(\Gamma_i)$ for each i . We now claim that the parity of the number of colourings of G with those colours which have a 1 in the corresponding place of v is the sum of these numbers *modulo 2*.

We demonstrated above that the parity of the number of H -colourings of $(G, x) \cdot (\Gamma_i, \gamma_i)$ is the parity of the number of ways of colouring G such that x is coloured using only colours from H such that there is a 1 in the corresponding place of $v_H^*(\Gamma_i)$. That is, the parity number of H -colourings of $(G, x) \cdot (\Gamma_i, \gamma_i)$

is the parity of the sum of the number of H -colourings of G with each of the relevant colours of H at x . But then the sum of these numbers is just the sum of these sums. The number of H -colourings of G which use a colour $h \in H$ at x is counted an odd number of times in this sum if and only if there are an odd number of Γ_i such that there is a 1 in the corresponding place of $v_H^*(\Gamma_i)$, but this is exactly the same condition for there to be a 1 in the corresponding place of v , which is the sum *modulo 2* of the $v_H^*(\Gamma_i)$, so the sum of these is exactly the sum *modulo 2* of the numbers of colourings of G which use one of the colours which have a 1 in the corresponding place of v at x . \square

Combining these results, we know that we can generate a combination of gadgets such that they vector corresponding to this combination of gadgets contains 1s only in a single orbit of $\text{Aut}(H)$, and that for any implementable vector, v , we can use a $\bigoplus H$ -colouring oracle to count H -colourings of G which use only the vertices of H which have ones in the corresponding places of v *modulo 2*. This finally allows us to prove Theorem 6.3.13, which stated that given any graph G and any involution-free graph H , it is possible to use an H -colouring oracle to determine the number of colourings of G in which some specified vertex is restricted to be coloured with vertices from some any given orbit of H .

Proof of Theorem 6.3.13. By Lemma 6.3.22 it is possible to implement the vectors which contain only 1s in the places corresponding to colours from some given orbit of H . But then, by Lemma 6.3.24 we can use a $\bigoplus H$ -colouring oracle to count H -colourings of G which use only these colours at some specified

vertex in G . □

6.4 Reduction by Involutions

As mentioned above, there is a certain property which a graph H can satisfy, relating to the structure and symmetry groups of H and some subgraphs of H which implies that the $\oplus H$ -colouring problem is polynomial time soluble. In fact, for connected H which meet this condition, determining the parity of the number of H -colourings is equivalent to checking whether or not the instance graph has any edges, combined with counting the number of singleton vertices. This condition is based on an operation we refer to as “reducing H by involutions” which essentially consists of picking an involution of H and deleting all vertices which are not fixed by it. We will argue below that for any graph G , this process preserves the number of H -colourings of G modulo 2. That is, the number of H -colourings of G has the same parity as the number of colourings of G with the graph one gets by reducing H by any of its involutions.

Definition 6.4.1. *Let H be a graph, and σ an automorphism of H . We denote by H^σ the subgraph of H induced by the fixed points of σ .*

Lemma 6.4.2. *If H is a graph, and σ an involution of H , the number of H -colourings of any graph, G , is congruent modulo 2 to the number of H^σ -colourings of G .*

Proof. We will in fact show that the number of H -colourings of G which are not H^σ -colourings is even. This will suffice, because an H -colouring which is

not an H^σ -colouring is one which uses at least one colour in $V(H)\setminus V(H^\sigma)$. The number of H -colourings of G is just the number of H -colourings which use at least one vertex from $V(H)\setminus V(H^\sigma)$ plus the number of colourings of G with the vertices of H^σ . But as the former is even (*i.e.* congruent to 0 modulo 2) this is congruent to the number of colourings of G with H^σ .

To see that the number of H -colourings which use at least one vertex of $H\setminus H^\sigma$ is even, we show that we can partition the set of such colourings into sets of size two. The basic idea here is that with each colouring which uses at least one vertex which is moved by σ we can associate the colouring gained by first applying σ to H and then colouring G . Formally, given any colouring $\phi : V(G) \rightarrow V(H)$, consider the alternative colouring $\sigma \circ \phi$. This is still an H -colouring of G , as both σ and ϕ are edge-preserving. It is different from ϕ as there is some vertex $v \in G$ such that $\phi(v) \in V(H)\setminus V(H^\sigma)$, and so $\sigma(\phi(v)) \neq \phi(v)$. On the other hand $\sigma \circ \sigma \circ \phi$ is just ϕ , as σ is an involution. So σ acts as an involution on the set of H -colourings of G which use at least one colour from $V(H)\setminus V(H^\sigma)$. Since this involution has no fixed points, the size of this set must be even. \square

Note that the above argument does not rely on any special properties of the modulus 2. In fact, by an exactly parallel argument, we get the following.

Theorem 6.4.3. *For all integer k , if H is a graph, and σ an automorphism of H of order k , the number of H -colourings of any graph, G , is congruent modulo k to the number of H^σ -colourings of G .*

We define the following reduction system on the set of unlabelled graphs.

Definition 6.4.4. Given a graph H , we define reduction as the relation \rightarrow_k , with $H \rightarrow_k K$ iff there exists an automorphism of order k of H , say σ , such that $H^\sigma = K$. If there is some way of reducing H to K , i.e. if there exists a sequence of graphs H_1, H_2, \dots, H_n such that $H \rightarrow_k H_1 \rightarrow_k H_2 \rightarrow_k \dots \rightarrow_k H_n \rightarrow_k K$ we say that $H \rightarrow_k^* K$.

Definition 6.4.5. We say that a graph K is a reduced form associated with a graph H if K has no automorphisms of order k and $H \rightarrow_k^* K$. Note that the condition that H has no automorphisms of order k is equivalent to saying that there is no graph G such that $K \rightarrow G$. If k is prime, then for each H there is only one such K , up to isomorphism, by Theorem 6.4.8 below.

Example 6.4.6. In Figure 6.1 we give an example of a graph G , along with two ways of reducing G by involutions. On the right-hand side we reduce G by using the involution σ which swaps each of the pairs of vertices a and e , b and f , c and d , leaving behind only the involution-free graph on the vertices g and h . On the left-hand side, we begin with the involution τ which swaps e and f , and have to reduce the resulting graph by involutions twice more before we get to the involution-free graph $((G^\tau)^\upsilon)^\eta$ which is isomorphic to the graph G^σ , as demonstrated in Theorem 6.4.8.

If one reduces a given graph by involutions until an involution-free graph is obtained, this involution-free graph is uniquely determined (up to isomorphism) by the original. Thus, it makes sense to talk about ‘the’ involution-free graph obtained from a given H . We will need to use a minor modification of Lemma 6.3.8 in order to show this. Lemma 6.3.8 said that the parity Lovász vector

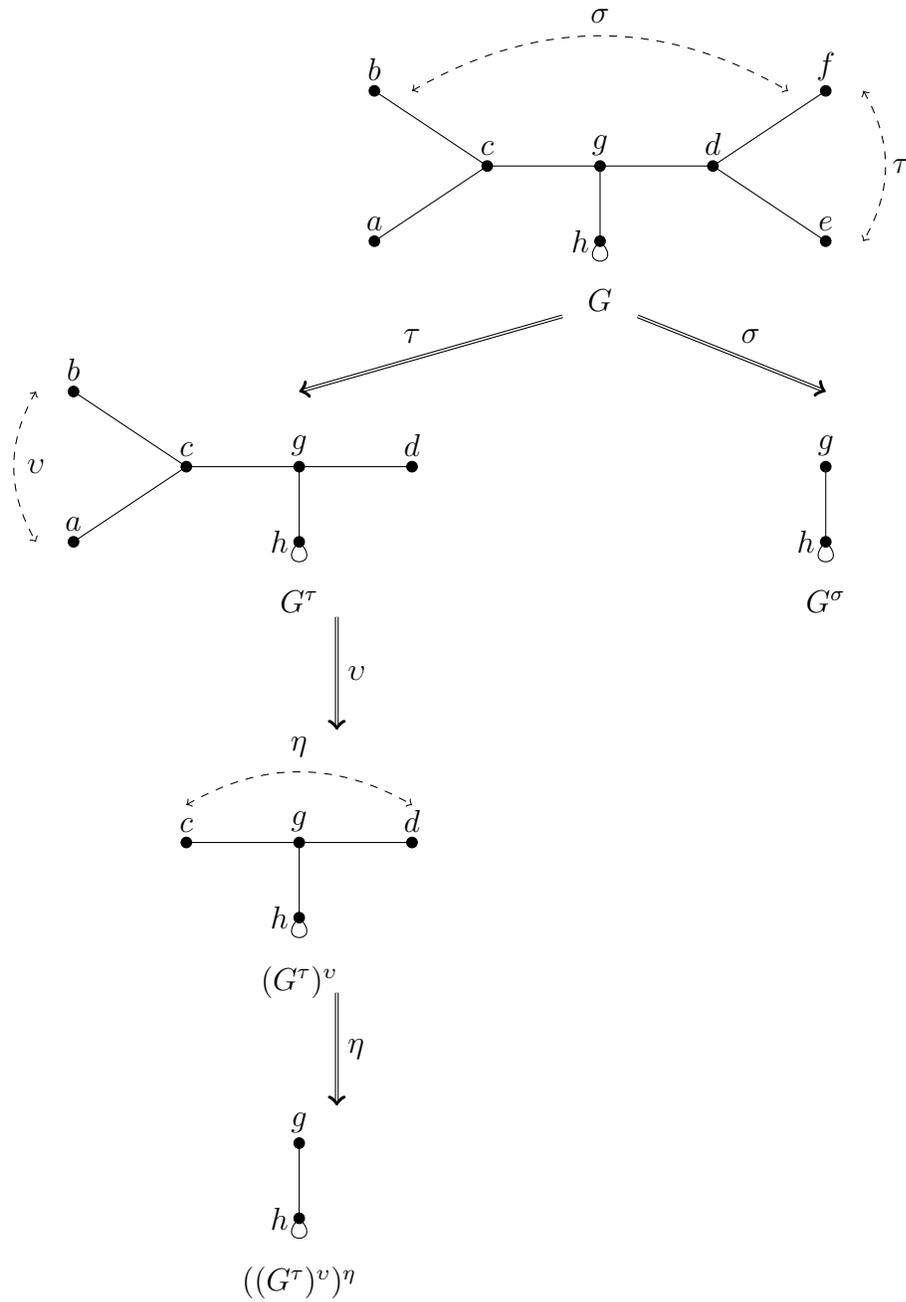


Figure 6.1: An example of a graph G with the sequence of reductions we get from G if we start with each of the involutions σ and τ .

was enough to characterise involution-free graphs with specified vertices. Here, we need the version for graphs without specified vertices. In fact, there is no reason to restrict ourselves to involutions and *modulo 2*. The reduced form associated with any graph in \rightarrow_p is unique for all prime p .

Lemma 6.4.7. *For any prime p , if two graphs with no automorphisms of order p have the same mod p Lovász vector then they are isomorphic.*

Proof. The proof is almost identical to that of Lemma 6.3.8 but homomorphisms are all homomorphisms in the normal graph sense instead of homomorphisms between graphs with specified vertices. \square

Theorem 6.4.8. *Given a graph G , and a prime p there is (up to isomorphism) exactly one graph, G^* such that G^* has no automorphisms of order p and $G \rightarrow_p G^*$.*

Proof. We consider the mod p Lovász vector of G . Lemma 6.4.2 says the reduction operation we described above preserves the mod p Lovász vector. On the other hand, Lemma 6.4.7 above says that the mod p Lovász vector characterises (isomorphism classes of) graphs with no automorphisms of order p . So any two graphs with no automorphism of order p which we can reach from the same G have the same mod p Lovász vector by the first Lemma and so are isomorphic by the second. \square

6.5 Trees

As we have seen, if we apply the reduction operations defined in Definition 6.4.4 to any graph H , this preserves the parity of the number of H -colourings

of any graph G . In particular, if a given H reduces to a graph, say H' such that the H' -colouring problem lies in P, then the H -colouring problem also lies in P. There are certain graphs for which the H -colouring problem obviously lies in P: namely, the graph on one vertex, the graph on one vertex with a loop, the graph on two vertices, one with a loop and one without, and the null graph

Lemma 6.5.1. *Counting the number of H -colourings of a given graph G can be done in polynomial time if H is one of the null graph (the graph on no vertices), the graph on one vertex, the graph on one vertex with a loop, or the graph on two disconnected vertices, one with a loop and one without.*

Proof. If H is the null graph then there is no H -colouring of G , so the counting problem is obviously trivial. If H is the graph on one vertex then G has exactly one H -colouring if and only if G has no edges, and zero otherwise, which can be determined in polynomial time. If H is the graph on one vertex with a loop, then there is exactly one H -colouring of G . If H is the graph on two vertices one with a loop and one without then there are exactly $2^{|Is(G)|}$ colourings of G , where $Is(G)$ is the set of isolated vertices of G . Each isolated vertex can be coloured with either the looped vertex or the unlooped vertex of H independently, and all the vertices which form part of a connected component of size greater than one must be coloured with the looped vertex. \square

Corollary 6.5.2. *If the reduced form associated with a given H in the reduction system defined in Definition 6.4.4 is one of the null graph, the graph on one vertex, the graph on one vertex with a loop or the graph on two vertices, one with a loop and one without, then H -colouring is in P.*

Proof. This follows directly from Lemma 6.5.1 and the fact that the reduction system preserves the number of H -colourings, as shown in Lemma 6.4.2 \square

We conjecture that for general graphs, the reduction given in Corollary 6.5.2, that is, H reducing by involutions to one of the four trivial graphs, is the only way in which the $\oplus H$ -colouring problem can fail to be $\oplus P$ -complete. Note that this does encompass all of the easy cases in [DG00]. A complete graph with loops everywhere reduces to the empty graph if it has an odd number of vertices and the graph on one vertex with a loop if it has an even number. On the other hand, a complete bipartite graph reduces to the graph on one vertex if there are an odd number of vertices in total, and the empty graph otherwise.

In this section, we will prove that this conjecture is true for trees. In particular, if the reduced form in the reduction system defined in 6.4.4 associated with a given tree T is the graph on one vertex or the empty graph then the associated $\oplus T$ -colouring problem can be solved in polynomial time. Otherwise, it is $\oplus P$ -complete.

6.5.1 Involution-Free Trees

We begin by exploring some structure in involution-free trees. These trees have quite a lot of structure, and we will exploit this when we build gadgets for our reductions from $\oplus \text{INDEPENDENT-SET}$ to $\oplus H$ -colouring in the next section.

Lemma 6.5.3. *An involution-free tree on more than one vertex has two vertices of degree 2 which are adjacent to leaves.*

Proof. The argument given below is very similar to the standard argument given to show that any tree has at least two leaves.

The first observation to make is that any involution-free tree contains some path of length at least 3. If the maximum-length path in a tree is of length 1, then the tree consists of a single edge, and so has an involution. If it is of length 2, then the tree is a star, and exchanging any two of its leaves is an involution.

Consider a longest path in an involution-free tree, and label the vertices of this path $p_0, p_1 \dots p_M$. Note that p_0 and p_M are both leaves. Then we claim that both vertices p_1 and p_{M-1} are degree 2. Note that p_1 and p_{M-1} are in fact distinct vertices, as $M \geq 3$. Assume the degree of p_1 is greater than 2, and consider a vertex, v , adjacent to p_1 which is neither p_0 nor p_2 . This vertex cannot have any neighbours which are not already in the path (as this would contradict maximality of the path). It also cannot have any neighbours which are in the path (as this would create a cycle, contradicting the fact that G is a tree). Therefore, it cannot have any neighbours other than p_1 . But then exchanging this vertex with p_0 is an involution of G , so there is no such vertex, and p_1 is degree 2 as claimed. An analogous argument shows that p_{M-1} must be degree 2. \square

In order to fully exploit the power of the results proved in Section 6.4, we need to use the fact that an involution-free tree has no automorphisms. Although this fact is no doubt well-known, I have not been able to find a proof of it in the literature, so give one below. We in fact prove the contrapositive: if a tree has at least one automorphism, then it has an involution.

We will use the following well-known result on trees, a proof of which can

be found in [Ser03], it states that a tree either contains vertex which is fixed by all of its automorphisms, or an edge which is fixed by all of its automorphisms.

Lemma 6.5.4 ([Ser03], p. 20). *Every tree has either a vertex which is fixed by every automorphism or an edge which is fixed by every automorphism, in the sense that either both of its endpoints are fixed or they are mapped to one another.*

Definition 6.5.5. *A rooted tree (T, r) is a tuple consisting of a tree T along with a root r , which is one of its vertices. An automorphism of a rooted tree is an automorphism of T which fixes r .*

Lemma 6.5.6. *If a rooted tree has any non-trivial automorphisms, then it has an involution.*

Proof. Consider a non-trivial automorphism ϕ of a rooted tree T ; we will use ϕ to construct an involution on the vertices of T . Take the vertex $v \in V(T)$ with $\phi(v) \neq v$ such that the distance from v to the root is as small as possible; such a vertex exists by the assumption that ϕ is a non-trivial automorphism.

Now, the subtree rooted at v must be isomorphic to the subtree rooted at $\phi(v)$, as there is an automorphism which maps one to the other. Also, v and $\phi(v)$ share a neighbour. In order to see this, consider w , the neighbour of v which is fixed by ϕ . Since $w = \phi(w)$ and v is adjacent to w , we know that $\phi(v)$ is also adjacent to $\phi(w)$, which is equal to w .

So the map which exchanges v and $\phi(v)$, and exchanges the subtrees rooted at each of these vertices, and fixes all vertices which are closer to the root than

v and all vertices which are in neither of these subtrees is an automorphism of T . This automorphism is an involution, and we are done. \square

Lemma 6.5.7. *An involution-free tree has trivial automorphism group.*

Proof. We prove the contrapositive. Consider an automorphism, ϕ of an arbitrary tree, T . By Lemma 6.5.4, T either has a vertex which is fixed by every automorphism or an edge which is fixed by every automorphism. If T has a vertex which is fixed by every automorphism, then any automorphism of T is also an automorphism of the rooted tree (T, r) , where r is this vertex and so, as T has a non-trivial automorphism by assumption, (T, r) has an involution by Lemma 6.5.6, and this is clearly an involution of T . On the other hand, T may have an edge which is fixed by every automorphism. In this case, either ϕ fixes this edge or it exchanges its endpoints. If ϕ exchanges the endpoints, then ϕ is of even order, and T has an involution by Lagrange's Theorem. Otherwise, ϕ is an automorphism of the rooted tree (T, r) , where r is either endpoint of the edge which is fixed by $\text{Aut}(T)$, and so (T, r) , and hence T has an involution as in the previous case. \square

Finally, we require the following fact concerning the number of walks of various lengths between vertices in involution-free trees - this will enable us to count the number of ways of colouring paths which connect vertices known to be coloured with these colours when we are counting H -colourings of instance graphs in the following section.

Lemma 6.5.8. *Let H be an involution-free tree, let e_0 be a vertex of degree 2 which is adjacent to a leaf in H , and let e_k be a vertex of even degree such that*

there are no vertices of even degree on the path joining e_0 and e_k , where $k \geq 1$ is the length of the path joining e_0 and e_k . We will name the vertices on this path $e_0, o_1, o_2, \dots, o_{k-1}, e_k$.

Then there are an even number of vertices v such that both:

1. v is a neighbour of the first vertex on this path other than e_0 , i.e. $v \in N_H(o_1)$ (or $v \in N_H(e_1)$ in the case where $k = 1$) and
2. the number of walks of length k from v to e_k in H is odd.

Proof. We will refer in this proof to the vertices o_1 and o_2 , which do not exist if $k = 1$ or $k = 2$, we deal with this at the end of this proof. For now, assume $k \geq 3$. We want to prove that there are an even number of neighbours of o_1 from which there are an odd number of walks of length k to e_k in H . There are an odd number of paths of length k from e_k to each of the neighbours of o_1 other than o_2 : there is, in fact, one such walk, and it is the unique path connecting the neighbour to e_k in the tree. We claim that there are an even number of walks of length k from e_k to o_2 .

A walk of length k from e_k to o_2 traverses exactly 1 edge more than once, as there is a unique path of length $k - 2$ from e_k to o_2 . Two such walks which traverse the same edge more than once are identical. There is therefore a one-to-one correspondence between these walks and the edges which are traversed at least twice by at least one of them. We claim that the number of such edges is even.

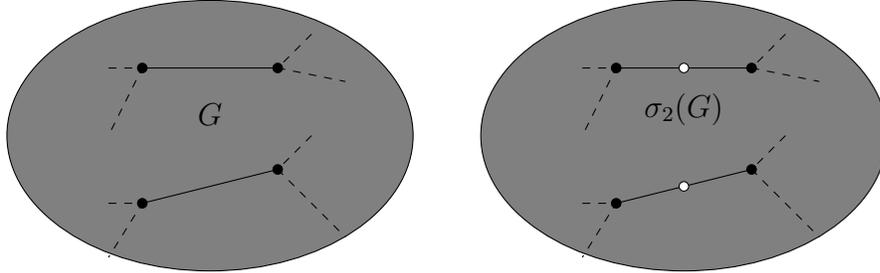
Any edge which is adjacent to any of the vertices in $\{o_2, o_3 \dots e_k\}$, and only those edges, may be traversed more than once, so it suffices to show that there

are an even number of such edges. To see this, note that the only edges in this set which are adjacent to more than one of the vertices in the set are: $\{(o_2, o_3), (o_3, o_4) \dots (o_{k-1}, e_k)\}$, there are the same number of edges in this set as the number of vertices of odd degree in $\{o_2 \dots e_k\}$. The total number of edges is then just the sum of the vertex degrees minus the number of edges which are adjacent to more than one of the vertices, but the sum of the vertex degrees is $k - 2 \pmod{2}$ (as there are $k - 2$ vertices of odd degree) and the number of repeated edges is $k - 2$, so the parity of the total number of edges is $(k - 2) - (k - 2) \equiv 0 \pmod{2}$.

As noted above, if $k = 1$ or if $k = 2$ the vertices o_1 or o_2 may not exist. However, the theorem still holds.

In particular, if $k = 1$ then we actually have two adjacent vertices of even degree and the first vertex on the path which is not e_0 is in fact e_1 , which is of even degree. Clearly there are an even number of vertices adjacent to e_1 with an odd number of length 1 walks to e_1 , these being exactly the neighbours of e_1 .

If $k = 2$, then again the vertex whose neighbours we are interested in is of odd degree, call it o_1 , and there are an odd number of walks of length 2 from e_2 to each of the neighbours of o_1 other than itself: in fact, there is exactly one such walk, the path joining the two vertices. On the other hand, e_2 is of even degree, so there are an even number of walks of length 2 from e_2 to itself. Since o_1 has an odd number of neighbours, this leaves an even number of neighbours of o_1 which have an odd number of length 2 walks to e_2 , as claimed. \square

Figure 6.2: The 2-stretch of G

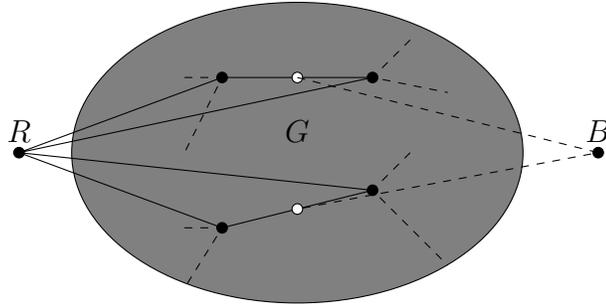
6.5.2 The Reduction

Theorem 6.5.9. *Given an involution-free tree H with more than one vertex, $\oplus H$ -colouring is $\oplus P$ -complete. In fact, there is a reduction from $\oplus IS$ to $\oplus H$ -colouring.*

Definition 6.5.10. *Given a graph G , we call $\sigma_2(G)$ the graph obtained by replacing every edge in G with a path of length 2. We refer to the newly introduced vertices as stretch vertices, and the original vertices of G as G -vertices. The construction is illustrated in Figure 6.2*

The graph defined above, $\sigma_2(G)$, is usually referred to as the 2-stretch of G , and it is an established result that counting H -colourings of $\sigma_2(G)$ is equivalent to counting H^2 -colourings of G , where H^2 is the multigraph whose adjacency matrix is the square of the adjacency matrix of H (see, *e.g.* [DG00]).

We will use a variant of this stretch operation in which we count only those colourings of $\sigma_2(G)$ in which both the stretch vertices and the G vertices are coloured with specific subsets of the colours in H . This is achieved using gadgetry based on the principles established in Section 6.3.

Figure 6.3: The construction of G^*

We now detail the reduction from Independent Set, first, given any graph G , we will construct a graph G^* . We then claim that the number of H -colourings of G^* with certain vertices restricted to be coloured with certain colours from H is congruent *modulo 2* to the number of independent sets in G .

For a given involution-free tree H , pick a vertex of degree 2, e_0 , adjacent to a leaf, and a vertex of even degree, e_k such that the unique path of length k in H from e_0 to e_k does not contain any vertex of even degree (exactly as in the statement of Lemma 6.5.8). Note that, as H is involution-free, there are two vertices of even degree, and at least one vertex of degree two which is adjacent to a leaf in H by Lemma 6.5.3, and we can choose e_0 and e_k with the above properties.

Now, given a graph G , first create $\sigma_2(G)$, then add two new vertices R and B . Add an edge between each of the original vertices of G (G -vertices) and R , and a path of length k from every one of the new vertices (stretch vertices) of $\sigma_2(G)$ to B . We call this new graph G^* , the construction is illustrated in Figure 6.3

Now, using the technology described in Theorem 6.3.13 and the fact that

the orbit of a vertex in an involution-free tree is trivial by Lemma 6.5.7 we can determine the parity of the number of H -colourings of G^* in which R is restricted to be coloured with e_0 and B is restricted to be coloured with e_k using only a $\oplus H$ -colouring oracle. We claim that this number is congruent (*modulo* 2) to the number of independent sets in G . We will use what we know about the number of walks of length k between the colours e_0 and e_k from Lemma 6.5.8.

Lemma 6.5.11. *Let H be an involution-free tree, let e_0 be a vertex of degree 2 adjacent to a leaf, and e_k a vertex of even degree at distance $k \geq 1$ from e_0 such that there are no vertices of even degree on the path of length k joining them.*

The number of H -colourings of G^ in which R is coloured with e_0 and B is coloured with e_k is congruent modulo 2 to the number of independent sets in G .*

Proof. First consider the G -vertices in G . They are all neighbours of a vertex which is coloured with e_0 , they must therefore be coloured with colours which are adjacent to e_0 in H . But e_0 was chosen to be one of the vertices of degree 2 adjacent to a leaf in H , so G -vertices can only be coloured with either the leaf adjacent to e_0 (which we will call l) or with the first vertex on the path linking e_0 and e_k , which we will call v_1 in the remainder of this proof. This vertex is o_1 , except in the case $k = 1$ where it is e_1 .

Now, consider the stretch vertices. These are connected to a vertex which is coloured e_k by a path of length k . So, consider the colour used at a given stretch vertex, s . If there are an even number of walks of length k from e_k to this colour in H , then there are an even number of colourings of G^* which

use that colour at s , as there are an even number of ways of colouring the path joining s and B , and the total number of colourings is the product of the number of ways of colouring this path with the number of ways of colouring the rest of the graph.

We therefore need to count colourings of G^* in which the colours used at the stretch vertices are such that there are an odd number of paths of length k between them and e_k in H .

Note that these colours must also be adjacent to either v_1 or l in H (as the G -vertices are all coloured with either v_1 or l , and every stretch vertex is adjacent to a G -vertex), and therefore, in fact, must be adjacent to v_1 , as the only neighbour of l is e_0 , which is also a neighbour of v_1 .

Now, we are reduced to considering colourings of G^* in which the following conditions hold. The G -vertices are coloured either l or v_1 , while the ‘stretch’ vertices are coloured with one of the neighbours of v_1 which has an odd number of length k walks from itself to e_k . We claim that the parity of the number of such colourings is equal to the parity of the number of ways of colouring G with the two colours l and v_1 such that no two vertices coloured with v_1 are adjacent.

Consider a colouring of G with the colours v_1 and l . If there are two vertices of G which are adjacent in G and both coloured with v_1 then there are an even number of extensions of this colouring to an H -colouring of G^* : the stretch vertex between the two G -vertices in G^* can be coloured with any one of the neighbours of v_1 which are at distance k from e_k in H , and there are an even number of such vertices by Lemma 6.5.8.

On the other hand, if there are no two such vertices, there is exactly one extension of the given colouring of G to an H -colouring of G^* : every one of the stretch vertices is adjacent to a vertex which is coloured l , so the stretch vertices must all be coloured e_0 , and as there is only one path of length k from e_0 to e_k in H , this determines the colouring of the vertices on the paths linking the stretch vertices to B .

So the number of colourings of G^* with H such that R is coloured e_0 and B is coloured e_k is congruent *modulo* 2 to the number of colourings of G in which each vertex is either coloured with l or v_1 and adjacent vertices may not both be coloured with v_1 . But these are exactly the independent sets of G - vertices coloured v_1 are 'in' the independent set and vertices coloured l are 'out'. \square

Proof of Theorem 6.5.9. by Theorem 6.3.13 and Lemma 6.5.7 we can count H -colourings of G^* in which R is coloured e_0 and B is coloured e_k in polynomial time if equipped with an H -colouring oracle, but we know that the number of such colourings is congruent *modulo* 2 to the number of independent sets in G . Since clearly G^* can be constructed from G in polynomial time, this gives us a polynomial time Turing reduction from \bigoplus IS to $\bigoplus H$ -colouring. \square

6.5.3 A Dichotomy for Trees

Theorem 6.5.12. *If H is a tree, then $\bigoplus H$ -colouring is $\bigoplus P$ -complete if the tree obtained by reducing H by involutions is non-trivial (i.e. has more than 1 vertex). Otherwise it is solvable in polynomial time.*

Proof. By Lemma 6.4.2, the number of H -colourings of a graph G is congruent

modulo 2 to the number of H' -colourings, where H' is any graph obtained from H by reducing H by any of its involutions. Also, if H is a tree then any graph H' which can be reached from H by reduction by involutions is also a tree. It therefore suffices to consider involution-free trees.

If H is an involution-free tree, and H contains more than one vertex, then Theorem 6.5.9 shows that $\oplus H$ -colouring is $\oplus P$ -complete. On the other hand, if H contains either 0 or 1 vertices then $\#H$ -colouring (and hence $\oplus H$ -colouring) is polynomial time solvable by Lemma 6.5.1. \square

6.6 Other Graphs

As noted earlier, we conjecture not only that there is a dichotomy for the complexity of $\oplus H$ -colouring for general H , but that this dichotomy is the same as that for trees. In other words, that the only way in which a $\oplus H$ -colouring problem can be polynomial-time solvable is if H reduces by involutions to one of the four trivial graphs: the graph on no vertices, the graph on one vertex with a loop, the graph on one vertex without a loop and the graph consisting of two disconnected vertices, one with a loop and one without.

We now show that we can restrict our attention to connected H . That is, if an involution-free graph H has any connected component H_1 for which $\oplus H_1$ -colouring is $\oplus P$ -hard, then the parity colouring problem associated with H is itself $\oplus P$ -hard.

Theorem 6.6.1. *Let H be an involution-free graph, if H_1 is a connected component of H and H_1 -colouring is $\oplus P$ hard, then H -colouring is $\oplus P$ hard.*

Proof. Take any graph G , assume that G is connected (since the number of H -colourings of G is just the product of the number of H -colourings of each of its connected components). We can use an oracle for H -colouring to determine the parity of the number of colourings of G in which only colours from H_1 are used in the following way: let $v \in V(G)$ be any vertex of G . For each colour $h_i \in V(H_1)$, we can count the colourings of G in which v is coloured h_i using Theorem 6.3.13. Notice that the size of the orbit of h_i in $\text{Aut}(H)$ is odd, as H has no involutions, so the parity of the number of colourings of G with h_i at v is the same as the parity of the number of colourings of G which use any of the vertices in the orbit of h_i at v .

But we can do this for every vertex in H_1 , and since G is connected, any colouring which uses a vertex from H_1 at v can use only colours from H_1 anywhere in G . Conversely, any colouring of G which uses only colours from H_1 must use some colour from H_1 at v , so this does indeed allow us to count all such colourings of G . \square

Note that this actually allows us to strengthen Theorem 6.5.12: the H -colouring problem associated with any forest H is polynomial-time solvable if the reduced form associated with the forest in the reduction system described in Section 6.4 is the empty graph or the graph on one vertex, and $\oplus\text{P}$ -complete otherwise.

Chapter 7

Complexity of finding the Reduced Form

We now take things in a slightly different direction, and discuss the complexity of determining the reduced form associated with a given graph in the reduction systems described in 6.4.4. Given a graph G , we want to know the complexity of deciding which involution-free (or more generally, automorphism-of-order- p -free) graph we will reach if we reduce it by involutions (automorphisms of order p).

7.1 Complexity of finding the Reduced Form

7.1.1 Modular Graph Automorphism

It is well-known that the problem of determining whether two graphs are isomorphic is polynomial-time equivalent to the the problem of determining the

size of the automorphism group of a given graph (see, *e.g.* [Hof82]). It is not known whether either of these problems is reducible to the problem of determining whether or not a given graph has a non-trivial automorphism. In [ABL00], Arvind, Beigel and Lozano introduce the problem of modular graph automorphism. That is, determining whether the size of automorphism group of a graph is a multiple of k . The motivation for this was to determine exactly how much one needs to know about the automorphism group of a graph before one is able to solve the graph isomorphism problem. Here we will show that the complexity of finding the reduced form associated with a given graph H is related to the complexity of these problems.

We first define a few languages based on decision problems related to the graph automorphism and isomorphism problems.

Definition 7.1.1. ***GA** is the Graph Automorphism decision problem:*

$$GA = \{G : |Aut(G)| > 1\}$$

Definition 7.1.2. ***GI** is the Graph Isomorphism decision problem:*

$$GI = \{(G, H) : G, H \text{ are graphs and there is an isomorphism from } G \text{ to } H\}$$

Definition 7.1.3. *For all k , **MOD_kGA** is the modular Graph Automorphism problem:*

$$MOD_k GA = \{G : |Aut(G)| \equiv 0 \pmod{k}\}$$

In [ABL00] it is shown that the MOD_kGA problems all lie between the

graph automorphism decision problem and the graph isomorphism decision problem under polynomial time reductions. That is, there is a polynomial time reduction from GA to each of the MOD_kGA problems, and a polynomial time reduction from each of the MOD_kGA problems to GI. We show that the problem of determining whether a given graph reduces to the empty graph under the reduction system defined in definition 6.4.4, for reduction by involutions, is also intermediate for these classes. Specifically, we show that it lies between GA and MOD_2GA .

Whilst they acknowledge that no proofs of strict inclusions exist regarding the GA and GI problems - indeed, it is still possible that all of the problems lie in P - the authors of [ABL00] conjecture that none of the problems GA, MOD_kGA and GI are in fact polynomial-time equivalent. They offer the fact that MOD_2GA is poly-time equivalent to Tournament Isomorphism as partial evidence in favour of this conjecture.

Definition 7.1.4. We define the language ***ReducesEmpty***, the set of graphs which reduce to the graph on no vertices in the reduction system for defined in 6.4.4: $\text{ReducesEmpty} = \{G : G \rightarrow_2^* \emptyset\}$.

We note that the reductions used in [ABL00] are of a slightly stronger sort than those we give here. In particular, they use Karp reductions, whereas we will continue to use polynomial-time Turing reductions.

Definition 7.1.5. We say that a language A is Karp-reducible to a language B , and write $A \leq_p^m B$, if there exists a function f , computable in polynomial time, such that for a given string X , $X \in A \iff f(X) \in B$.

Note that a Karp reduction is a polynomial-time Turing reduction, in which the oracle may only be called once, and the result of the oracle call for problem B is returned directly as the output of the Turing Machine for problem A without post-processing.

It is known that for all natural numbers $k > 1$, $GA \leq_m^p \text{MOD}_k GA \leq_m^p GI$. We show that $GA \leq_p^T \text{ReducesEmpty} \leq_p^T GI$. In fact, that $GA \leq_p^T \text{ReducesEmpty}$ and that $\text{ReducesEmpty} \leq_p^T \text{MOD}_2 GA$.

We will use a result from [ABL00] which states that the search problem for $\text{MOD}_2 GA$ is many-one reducible to the decision problem. As shown in Lemma 6.3.11, the order of the automorphism group of a graph is divisible by 2 iff the graph has an automorphism of order 2, and the search problem in question is the production of such an automorphism.

Lemma 7.1.6. *$\text{ReducesEmpty} \leq_p^T \text{MOD}_2 GA$.*

Proof. As noted in the paragraph preceding this lemma, if we have an oracle for $\text{MOD}_2 GA$, we can use it to find an automorphism of order 2 in a given graph in polynomial time. This enables us to find the reduced form associated with any graph using a $\text{MOD}_2 GA$ oracle in polynomial time in the following manner: we use the $\text{MOD}_2 GA$ oracle to find an involution of G , reduce G by this involution and use the oracle to find an involution of the new graph. Repeat until the current graph is involution-free. There is only one such involution-free graph for a given initial graph by the Theorem 6.4.8 in Section 6.4. This graph is the empty graph if and only if the reduced form associated with G is the empty graph. □

We now show that an oracle for `ReducesEmpty` can be used to determine whether or not an arbitrary graph has any non-trivial automorphisms. In order to do so, we will attach gadgets to vertices in copies of G which act as “labels”, so that two labelled vertices can only be mapped to one another.

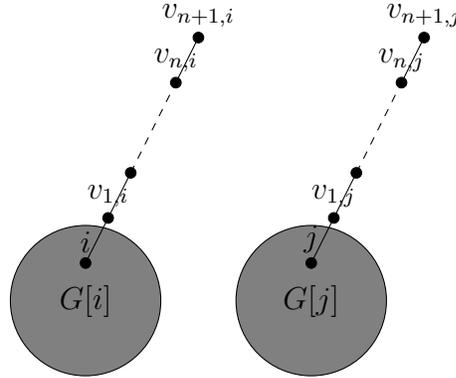
Theorem 7.1.7. $GA \leq_p^T \text{ReducesEmpty}$.

Definition 7.1.8. Consider a graph G and a vertex $i \in V(G)$. Then we call $G[i]$ the graph obtained by joining a path of length $|V(G)| + 1$ to G at i . For two vertices i and j , we use $G[i, j]$ to refer to the disjoint union of $G[i]$ and $G[j]$

Lemma 7.1.9. For all connected finite graphs G , except in the case where G is a path and i is one of its endpoints, any automorphism of $G[i]$ gives an automorphism of G which fixes i when it is restricted the subgraph of $G[i]$ generated by the vertices of G .

Proof. The path of length $|V(G)| + 1$ attached to i must be mapped to itself, as there can be no other path of this length in $G[i]$ which consists entirely of vertices of length 2 with a leaf as an endpoint. So any automorphism of G fixes the entire path and therefore fixes i . Also, any automorphism is an edge-preserving bijective map between $G[i]$ and itself, so is an edge-preserving bijective map between the subgraph of $G[i]$ induced by the original vertices of G and itself, which is therefore an automorphism of G . \square

Lemma 7.1.10. A graph G has at least one non-trivial automorphism iff there exists a pair of vertices $i, j \in V(G)$ with $i \neq j$ such that $G[i, j] \rightarrow_* \emptyset$

Figure 7.1: $G[i, j]$

Proof. For one direction: if there exists such a pair of vertices, then there is at least one involution in $G[i, j]$. This involution either associates at least two vertices which are in different components of $G[i, j]$ or not. In the first case, the involution must swap the two components, so the graphs $G[i]$ and $G[j]$ are isomorphic, and there must be an automorphism of G which maps i to j , and G has a non-trivial automorphism. In the second case, the involution must be an automorphism of G which fixes i (or j) by Lemma 7.1.9, and so G has a non-trivial automorphism.

For the other direction; if G has a non-trivial automorphism, σ then there exists at least one pair of vertices i, j such that $j = \sigma(i) \neq i$. Then consider the graph $G[i, j]$. The claim is that this graph reduces to the empty graph. In fact, this graph has an involution with no fixed points, which exchanges the two components: $\sigma(G[i])$ is isomorphic to $G[j]$ and $\sigma^{-1}(G[j])$ is isomorphic to $G[i]$, and so $G[i, j]$ has an involution which consists of applying σ to the $G[i]$ component and σ^{-1} to the $G[j]$ component and then exchanging the two. \square

Proof of Theorem 7.1.7. Given a graph G , we can construct all of the graphs $G[i, j]$ for each pair of vertices $i, j \in V(G)$ in polynomial time. There are polynomially many such $G[i, j]$. We then call the *ReducesEmpty* oracle on each of these $G[i, j]$. By the previous Lemma, if we get any positive responses to the oracle call, then we know G contains at least one non-trivial automorphism. If not, then G contains no non-trivial automorphisms. Finally, to deal with the exception in the statement of Lemma 7.1.9, we note that if G is a path then G does contain a non-trivial automorphism, and that this can easily be checked. \square

7.1.2 Other Moduli

Once again, an equivalent statement holds for moduli other than two. This time, for automorphisms of arbitrary order in a graph.

Definition 7.1.11. *We define the problem ReducesEmpty_k , the set of graphs which reduce to the empty graph when reduced by automorphisms of order k : $\text{ReducesEmpty}_k = \{G : G \rightarrow_k^* \emptyset\}$.*

It is clearly the case that ReducesEmpty_k can be reduced to the graph isomorphism problem, since the latter is equivalent to determining the entire automorphism group of a graph [Luk82], and so can be used in particular to find all automorphisms of order k . For prime k , we have a stronger result, again using the results of Arvind, Beigel and Lozano from [ABL00].

Lemma 7.1.12. *For all prime p , $\text{ReducesEmpty}_p \leq_p^T \text{MOD}_p \text{GA}$.*

Proof. The proof is essentially identical to that of Lemma 7.1.6. Arvind and Beigel showed that, for prime p the MOD_pGA search problem is equivalent to finding an automorphism of order p , so we can find the reduced form associated with a given graph by repeated application of this. \square

Now we will prove an analogue of Theorem 7.1.7 for moduli other than 2. Our construction will use a graph, Cyc_n , whose purpose, in essence, is to act as a directed cycle in the construction: its automorphism group is exactly the cyclic group of order n .

Lemma 7.1.13. *Let Cyc_n be the graph consisting of the following vertices and edges. $V(\text{Cyc}_n)$ consists of n sets of 3 vertices, $\{a_i, b_i, c_i : i = 1, \dots, n\}$. $E(\text{Cyc}_n)$ consists of all of the edges $(a_i, b_i), (b_i, c_i), (a_i, c_i)$, along with the edges $(a_i, a_{i+1}), i = 1, \dots, n - 1$ and (a_n, a_1) and the edges $(c_i, a_{i+1}), i = 1, \dots, n - 1$ and (c_n, a_1) . Then the automorphism group of Cyc_n is exactly the cyclic group of order n . The sets $\{a_i\}, \{b_i\}$ and $\{c_i\}$ form the orbits of vertices under the group.*

Proof. Consider any automorphism, ϕ of Cyc_n which is not the identity. We claim that $\phi(a_1)$ is equal to a_i for some i , that $\phi(b_1) = b_i, \phi(c_1) = c_i$ and that $\phi(a_j) = a_{j+i-1 \pmod n}$.

It is clear that a_1 must be mapped to some a_i , as the a_i are the only vertices of degree 4 in Cyc_n . But then consider $\phi(b_1)$. Since $(a_1, b_1) \in E(\text{Cyc}_n)$, $(\phi(a_1), \phi(b_1)) \in E(\text{Cyc}_n)$, but then $\phi(b_1) = b_i$, as b_i is the only vertex of degree 2 adjacent to a_i in Cyc_n . Similarly, $\phi(c_1) = c_i$ as c_i is the only vertex of degree 3 adjacent to both a_i and b_i in Cyc_n , and $\phi(a_2) = a_{i+1 \pmod n}$ as $a_{i+1 \pmod n}$ is

the other vertex of degree 4 adjacent to c_i . \square

Let $G_n[i, j]$ be the graph obtained by taking one copy of $G[i]$ along with $n - 1$ disjoint copies of $G[j]$ along with a copy of Cyc_n . Attach each vertex in the copy of $G[i]$ to a_1, b_1 and c_1 , attach every vertex in the first copy of $G[j]$ to a_2, b_2 and c_2 and each vertex in the k^{th} copy of $G[j]$ to a_{k+1}, b_{k+1} and c_{k+1} . Automorphisms of this graph must either map the $G[i]$'s and $G[j]$'s around in a cycle or stay within one of the $G[j]$'s. In particular, since the vertices which were in the copy of Cyc_n are all of higher degree than any other vertices, they must be mapped to each other, and so either the cycle must be fixed by the automorphism, or it must map these vertices around in a cycle, as in Lemma 7.1.13.

Lemma 7.1.14. *For any integer k , a graph G has at least one non-trivial automorphism iff there exists a pair of vertices $i, j \in V(G)$ such that $G_k[i, j]$ reduces to the empty graph when reduced by automorphisms of order k .*

Proof of Lemma 7.1.14. First, assume there is such a pair, i, j , then $G_k[i, j]$ has an automorphism of order k , say σ . Either this automorphism maps vertices in one of the subgraphs of $G_k[i, j]$ isomorphic to $G[j]$ into another or not.

In the former case, the automorphism must map the a vertices of the copy of Cyc_n around in a cycle (as these vertices are the only vertices of degree $4 + |V(G)|$ in $G_k[i, j]$), in which case the copy of vertex i in the subgraph of $G_k[i, j]$ isomorphic to $G[i]$ must be mapped to a copy of vertex j in a subgraph isomorphic to $G[j]$. But then $G[i]$ and $G[j]$ are isomorphic, and there must be an automorphism of G which maps i to j as above.

If the automorphism does not associate vertices in different components of $G_k[i, j]$ then it must be an automorphism of the original graph G , again, as in the proof of Lemma 7.1.10.

For the other direction, if G has a non-trivial automorphism, σ , then pick a pair of vertices i and j such that $j = \sigma(i) \neq i$. Note that in this case $G[j]$ is isomorphic to $\sigma(G[i])$. Now we can find an automorphism of $G_k[i, j]$ which has no fixed points. We do this by applying σ to the subgraph which was isomorphic with $G[i]$ and σ^{-1} to the $k - 1^{\text{th}}$ copy of $G[j]$. Now consider the automorphism which maps the a_i vertices of Cyc_n to $a_{i+1 \pmod k}$, maps $\sigma(G[i])$ onto the first copy of $G[j]$, maps each of the copies of $G[j]$ to the next one around the cycle using the identity, and finally maps $\sigma^{-1}(G[j])$ onto $G[i]$. This is clearly order k , as the cycle formed by the a_i vertices is length k , and it is an automorphism, as we know that $G[j]$ is isomorphic to $\sigma(G[i])$ and $G[i]$ is isomorphic to $\sigma^{-1}(G[j])$. So the reduced form associated with $G_k[i, j]$ is the empty graph, as required. \square

Theorem 7.1.15. *For all k , $GA \leq_p^T \text{ReducesEmpty}_k$.*

Proof. Given a graph G , we can construct all of the graphs $G_k[i, j]$ for each pair of vertices $i, j \in V(G)$ in polynomial time. There are polynomially many such $G_k[i, j]$. We then call the ReducesEmpty_k oracle on each of these $G_k[i, j]$. By Lemma 7.1.14, if we get any positive responses to the oracle call, then we know G contains at least one non-trivial automorphism. If not, then G contains no non-trivial automorphisms. \square

We now have the following results, on the complexity of determining whether

the reduced form associated with H is isomorphic to any given graph K .

Definition 7.1.16. For any graph K , we define the problem $\text{Reduces}K_k$, the set of graphs which reduce to K when reduced by automorphisms of order k : $\text{Reduces}K_k = \{G : G \rightarrow_k^* K\}$.

Theorem 7.1.17. For any graph H and any graph K with no automorphisms of order p , $GA \leq_p^T \text{Reduces}K_k$

Proof. By Theorem 7.1.15, it suffices to show that we can solve ReducesEmpty_k with an oracle for $\text{Reduces}K_k$. Consider the graph consisting of H along with one disjoint copy of K . Since the order in which we choose to apply automorphisms does not matter, we can simply reduce the copy of H until we find its reduced form before touching any of the vertices in the copy of K . But then the reduced form associated with this new graph is the reduced form associated with $K \cup H^*$, where H^* is the reduced form associated with H . And this is isomorphic to K if and only if the reduced form associated with H is the empty graph. \square

7.2 Complexity of the dichotomy

Note that this doesn't quite answer the question we might like to ask, inspired by the conjecture in Section 6.5. If the conjecture is true, then the question "Is this $\bigoplus H$ -colouring problem $\bigoplus P$ -complete?" is equivalent to asking "Is the reduced form associated with H anything other than one of four easy cases enumerated in Corollary 6.5.2?". We would like to know the complexity of answering this question, given H . This can be done with slight modifications to the proofs above.

Definition 7.2.1. We define the class IsEasy_2 to be the set of graphs whose reduced form in \rightarrow_2 is one of the null graph, the graph on one vertex with a loop or the graph on two vertices, one of which has a loop.

Lemma 7.2.2. $\text{IsEasy}_2 \leq_p^T \oplus \text{GA}$.

Proof. As in the proof of Lemma 7.1.6, we can use a $\oplus \text{GA}$ oracle to find an involution in H if one exists. We can then repeatedly reduce H by involutions until we find an involution-free graph. Then $H \in \text{IsEasy}_2$ iff this involution-free graph is one of the four easy graphs. \square

Theorem 7.2.3. $\text{GA} \leq_p^T \text{IsEasy}_2$.

Proof. We proceed by mimicking the proof of Theorem 7.1.7. It suffices to show that an equivalent of Lemma 7.1.10 for IsEasy_2 holds. In particular, we show that a graph G has a non-trivial automorphism iff there exists a pair of vertices i, j such that $G[i, j] \in \text{IsEasy}_2$. We can then construct and check each of the $G[i, j]$ with an IsEasy_2 oracle in polynomial time, and G has non-trivial automorphism iff any of these is in IsEasy_2 .

In Lemma 7.1.10, we showed that G has a non-trivial automorphism iff there exists a pair of vertices i, j such that $G[i, j] \rightarrow_2 \emptyset$. Now, if $G[i, j] \rightarrow_2 \emptyset$ then certainly $G[i, j] \in \text{IsEasy}_2$, so it suffices to show that for any pair of vertices i, j if $G[i, j] \in \text{IsEasy}_2$ then G has a non-trivial automorphism. But exactly the same argument as in the proof of Lemma 7.1.10 applies. If the reduced form associated with $G[i, j]$ is not equal to $G[i, j]$, then $G[i, j]$ has an involution. This involution either swaps the components of $G[i, j]$ or not. If not, then it is an involution of G , and we are done. If it does swap the components, then,

unless G is just a path, the copy of i with the long path attached to it must be mapped to the copy of j with the long path attached to it, and G has an automorphism which swaps i and j . Finally, if G is just a path then it does have a non-trivial automorphism, and it is easy to check if G is a path. \square

Bibliography

- [ABL00] V. Arvind, R. Beigel, and A. Lozano, *The complexity of modular graph automorphism*, SIAM J. Comput. **30** (2000), no. 4, 1299–1320.
- [Ann94] J.D Annan, *The complexity of counting problems*, Ph.D. thesis, University of Oxford, 1994.
- [BDG⁺09] Andrei Bulatov, Martin Dyer, Leslie Ann Goldberg, Markus Jalsenius, and David Richerby, *The complexity of weighted boolean #csp with mixed signs*, Theoretical Computer Science **410** (2009), no. 38–40, 3949 – 3961.
- [BDG⁺10] Andrei A. Bulatov, Martin E. Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby, *The complexity of weighted and unweighted #csp*, CoRR **abs/1005.2678** (2010).
- [BG92] Richard Beigel and John Gill, *Counting classes: Thresholds, parity, mods, and fewness*, Theoretical Computer Science **103** (1992), 3–23.

- [Bul06] Andrei A. Bulatov, *A dichotomy theorem for constraint satisfaction problems on a 3-element set*, J. ACM **53** (2006), 66–120.
- [Bul08] Andrei A. Bulatov, *The complexity of the counting constraint satisfaction problem*, ICALP (1), 2008, pp. 646–661.
- [CCL10] J. Cai, Xi Chen, and Pinyan Lu, *Graph homomorphisms with complex values: A dichotomy theorem*, ICALP (1), 2010, pp. 275–286.
- [CCL11] Jin. Cai, Xi Chen, and Pinyan Lu, *Non-negatively weighted #csp: An effective complexity dichotomy*, IEEE Conference on Computational Complexity, 2011, pp. 45–54.
- [CH89] J. Cai and L. A. Hemachandra, *On the power of parity polynomial time*, Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89 (New York, NY, USA), Springer-Verlag New York, Inc., 1989, pp. 229–239.
- [CH96] Nadia Creignou and Miki Hermann, *Complexity of generalized satisfiability counting problems*, Inf. Comput. **125** (1996), no. 1, 1–12.
- [CKS01] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan, *Complexity classifications of boolean constraint satisfaction problems*, SIAM Monographs on Discrete Mathematics and Applications 7 (2001).
- [DG00] Martin Dyer and Catherine Greenhill, *The complexity of counting graph homomorphisms*, Random Structures and Algorithms **17** (2000), 260–289.

- [DGJ09] Martin Dyer, Leslie Ann Goldberg, and Mark Jerrum, *The complexity of weighted boolean csp*, SIAM J. Comput. **38** (2009), 1970–1986.
- [DGJ10] Martin Dyer, Leslie Ann Goldberg, and Mark Jerrum, *An approximation trichotomy for boolean #csp*, Journal of Computer and System Sciences **76** (2010), no. 3-4, 267 – 277.
- [DR11] Martin E. Dyer and David Richerby, *The #csp dichotomy is decidable*, STACS, 2011, pp. 261–272.
- [FV98] Tomas Feder and Moshe Y. Vardi, *The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory*, Siam Journal on Computing **28** (1998), 57–104.
- [GHLX11] Heng Guo, Sangxia Huang, Pinyan Lu, and Mingji Xia, *The complexity of weighted boolean #csp modulo k*, STACS, 2011, pp. 249–260.
- [GLV11] Heng Guo, Pinyan Lu, and Leslie Valiant, *The complexity of symmetric boolean parity holant problems*, Automata, Languages and Programming (Luca Aceto, Monika Henzinger, and Jir Sgall, eds.), Lecture Notes in Computer Science, vol. 6755, Springer Berlin / Heidelberg, 2011, pp. 712–723.
- [Goo04] A. J. Goodall, *The tutte polynomial modulo a prime*, Advances in Applied Mathematics **32** (2004), no. 1-2, 293 – 298, Special Issue on the Tutte Polynomial.

- [Her90] U. Hertrampt, *Relations among mod-classes*, Theor. Comput. Sci. **74** (1990), no. 3, 325–328.
- [HN90] Pavol Hell and Jaroslav Nešetřil, *On the complexity of h -coloring*, J. Comb. Theory Ser. B **48** (1990), no. 1, 92–110.
- [HN04] ———, *Graphs and homomorphisms*, Oxford University Press, 2004.
- [Hof82] C.M. Hoffmann, *Group-theoretic algorithms and graph isomorphism*, Lecture notes in computer science, Springer, 1982.
- [Kar75] R. M. Karp, *Reducibility among combinatorial problems*, Requirements Engineering, 1975.
- [Lad75] Richard E. Ladner, *On the structure of polynomial time reducibility*, J. ACM **22** (1975), no. 1, 155–171.
- [Lin86] Nathan Linial, *Hard enumeration problems in geometry and combinatorics*, SIAM J. Algebraic Discrete Methods **7** (1986), no. 2, 331–335.
- [Luk82] Eugene M. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, Journal of Computer and System Sciences **25** (1982), no. 1, 42 – 65.
- [McK59] James H. McKay, *Another proof of cauchy’s group theorem*, American Mathematical Monthly **66** (1959).

- [Sch78] T.J. Schaefer, *The complexity of satisfiability problems*, Proceedings, 10th Symposium on Theory of Computing, San Diego CA (1978).
- [Ser03] J.P. Serre, *Trees*, Springer monographs in mathematics, Springer, 2003.
- [Sim75] Janos Simon, *On some central problems in computational complexity.*, Ph.D. thesis, Cornell University, Ithaca, NY, USA, 1975.
- [Tod91] Seinosuke Toda, *Pp is as hard as the polynomial-time hierarchy*, SIAM J. Comput. **20** (1991), no. 5, 865–877.
- [Val78] Leslie G. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science (1978), no. 8, 189–201.
- [Val05] ———, *Completeness for parity problems*, Proceedings, 11th International Computing and Combinatorics Conference (2005), 1–9.
- [Val06] ———, *Accidental algorithms*, FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06) (Washington, DC, USA), IEEE Computer Society, 2006, pp. 509–517.
- [VV86] L. G. Valiant and V. V. Vazirani, *Np is as easy as detecting unique solutions*, Theoretical Computer Science **47** (1986), 85 – 93.
- [Wel93] D. J. A. Welsh, *Complexity: knots, colourings and counting*, Cambridge University Press, New York, NY, USA, 1993.

- [Yam11] Tomoyuki Yamakami, *Approximate counting for complex-weighted boolean constraint satisfaction problems*, Proceedings of the 8th international conference on Approximation and online algorithms (Berlin, Heidelberg), WAOA'10, Springer-Verlag, 2011, pp. 261–272.