Complexity Results for Boolean Constraint Satisfaction Problems

Von der Fakultät für Elektrotechnik und Informatik der
Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor der Naturwissenschaften
Dr. rer. nat.
genehmigte Dissertation
von

Dipl.-Inform. Michael Bauland

geboren am 22. August 1977 in Gronau (Westf.)

Hannover, 2007

Umschlagsgrafik: Anna Bauland

Referent: Prof. Dr. Heribert Vollmer
Korreferent: Prof. Dr. Martin Mundhenk
Tag der Promotion: 29.01.2007

Kurzzusammenfassung

Meine Dissertation befasst sich mit Boole'schen *Constraint Satisfaction Problemen* (kurz: CSP). Ein Constraint besteht aus einer Menge von Variablen und einer (Boole'schen) Relation, die die Belegungen bestimmter Tupel von Variablen einschränkt. Ein CSP ist dann die Frage, ob es zu einer gegebenen Menge von Constraints eine Belegung aller Variablen gibt, die alle Constraints gleichzeitig erfüllt. Dieses CSP und einige seiner Derivationen werden unter komplexitätstheoretischen Aspekten betrachtet.

Als wichtiges Instrument zur Bestimmung der Komplexität von CSP wird die *algebraische Methode* verwendet. Diese nutzt die von Emil Post gefundene vollständige Klassifikation aller unter Superposition abgeschlossener Klassen Boole'scher Funktionen (Clones). Der Abschluss einer Menge von Funktionen unter Superposition bedeutet dabei, dass die Menge der Funktionen unter beliebigen Kompositionen abgeschlossen ist. Der nach ihm benannte Post'sche Graph zeigt die vollständige Inklusionsstruktur der Clones. Hiermit konnten in Verbindung mit der Galoistheorie schon viele elegante Beweise geführt werden. Unter anderem wurde so auch das Dichotomieergebnis von Thomas Schaefer erneut bewiesen. Dieses besagt, dass das Boole'sche CSP in Abhängigkeit von den zugelassenen Boole'schen Constraints entweder in **P** oder **NP**-vollständig ist. Die behandelten Themen der Arbeit sind dabei im Einzelnen:

- Im ersten Teil dieser Arbeit wird das Resultat von Schaefer genauer untersucht. Es werden hierbei alle in **P** liegenden Fälle bis zur **L**-Isomorphie betrachtet, um so eine vollständige Klassifikation der Komplexität zu erhalten.

- Im Weiteren werden *quantifizierte* CSP (QCSP) betrachtet. Der Unterschied zu CSP liegt darin, dass bei QCSP die Variablen entweder existentiell oder universell quantifiziert sind. Als Spezialfall treten dabei auch die aus der Datenbanktheorie bekannten *Conjunctive Queries* auf, bei denen es keine universell quantifizierten Variablen gibt. Diese Arbeit untersucht dabei zum einen die Komplixität des QCSP sowie zum anderen die des *Counting*-Problems für QCSP (#QCSP). Letzteres stellt sich der Frage, wie viele erfüllende Belegungen ein quantifiziertes Constraint mit zusätzlichen freien Variablen hat.

- Abschließend wird eine Reihe von Äquivalenz- und Isomorphieproblemen komplexitätstheoretisch bestimmt. Dabei wird vor allem das Problem der isomorphen Implikation (gegeben zwei Constraintformeln, gibt es eine Permutation der Variablen, so dass die eine Formel die andere impliziert) als natürliche Erweiterung des Äquivalenzproblems (gegeben zwei Constraintformeln, ergibt sich bei allen möglichen Belegungen der Variablen der gleiche Wahrheitswert) näher betrachtet. Weiterhin werden die Probleme der Äquivalenz und der Implikation detaillierter untersucht, indem die zu vergleichenden Constraintformeln aus unterschiedlichen Relationsklassen stammen können.

Abstract

My thesis is concerned with Boolean *constraint satisfaction problems* (CSP). A constraint consists of a set of variables and a (Boolean) relation, which restricts the assignment of certain tuples of variables. A CSP then is the question, whether there is an assignment for all variables to a given set of constraints, such that all constraints are satisfied simultaneously. We examine this CSP and some of its derivations from a complexity theoretical point of view.

As the most important tool to determine the complexity of CSP we employ the *algebraic method*. It uses the complete classification of all classes of Boolean functions that are closed under superposition (clones). This classification has been obtained by Emil Post. The closure of a set of functions under superposition means, that the set of functions is closed under arbitrary composition. Post's lattice shows the complete inclusion structure of all clones. Hereby and in connection with the Galois theory it has been possible to obtain elegant proofs for several problems. Among others, it has been possible to reprove a dichotomy result first obtained by Thomas Schaefer. His result states that the Boolean CSP is either in **P** or **NP**-complete, depending on the set of constraints allowed. An outline of the topics of this thesis follows.

- In the first part of this thesis we examine the result by Schaefer in more detail. We look at all cases in **P** up to **L**-isomorphism, in order to obtain a complete classification of the complexity of CSP in the Boolean case.

- Further, we look at *quantified* CSP (QCSP). The distinction to CSP is, that variables are universally and existentially quantified. As a special case *conjunctive queries*, known from database theory, will occur. These are quantified formulae without universal quantifiers. This section of the thesis examines on the one hand the complexity of QCSP and on the other hand also the complexity of the corrresponding *counting* version #QCSP, which is the problem to find out how many satisfying assignments there are to a quantified formula with additional free variables.

- Finally, we determine the complexity of some equivalence and isomorphism problems in the constraint context. Our main attention is directed to the problem of isomorphic implication (given two constraint formulae, is there a permutation of the variables such that the first one implies the second one), which is a natural extension of the equivalence problem (given two constraint formulae, do they evaluate to the same value for all possible variable assignments). Further, we look at the problems of equivalence and implication in more detail, in such a way that the constraint formulae to be compared may originate from different sets of relations.

Acknowledgement

I would like to thank everybody who helped me with the development of this thesis and who supported me in my work. Especially I would like to thank my doctoral adviser Heribert Vollmer, who provided me with this opportunity in the first place, and my colleagues Henning and Ilka Schnoor. You three were a great help and in particular, thank you Henning, without you answering those many questions of mine patiently and in detail, I wouldn't have come so far.

Then I would like to thank Edith Hemaspaandra for inviting me to the cold and snowy Rochester and giving my research a good start. I'd like to thank Martin Mundhenk, Nadia Creignou, Steffen Reith, Elmar Böhler, and Thomas Schneider for many fruitful discussions.

Thank you, Ingo Fliegen, Michael Thomas, and Michael Pangborn for reading (parts of) my thesis and your ensuing comments and suggestions.

I'd also like to thank my family, especially my dear little sister Anna, for being there for me, whenever I needed you. And finally, last but not at all least, I would like to thank Miriam for simply everything. You were my greatest support.

This thesis is about all kinds of beautiful problems. Beautiful problems are very important for complexity theory, since they allow to prove beautiful theorems, and publish beautiful papers.

The need for more beauty in complexity theory was first examined by Cook. In his seminal paper [Coo71], he showed that the most beautiful problem in the world is the *satisfiability problem*.

Building on his result, many more problems were proven to be incredible beautiful. The main tool used for this is the *beauty reduction*, which is a polynomial-time procedure making every problem at least as beautiful as this introduction.

Henning Schnoor (when asked, how to write an introduction)

Contents

1 Introduction

Computer science as a whole contains many fields. They can be very broadly divided into practical and theoretical areas. A member of the former area, for example, is the field of software engineering, of designing and implementing specific software and tools that are of direct use for people or companies. Another example of practical computer science is the field of computer graphics and multimedia, where one goal might be the modeling of three dimensional objects in computers or the automatic recognition of people's faces by computer programs. In the area of theoretical computer science most research is not directly useful for everyday life; however, it forms a basis for most other fields of computer science. Developing computer software would be less advanced if it were not for the research in the theory of programming languages. Similarly the complexity theory is a broad field, that allows people to know in advance, for example, whether it makes sense to tackle a certain problem or whether the problem simply cannot ever be solved by any computers, no matter how advanced they may become. That is the field, where this thesis is located.

The general idea behind complexity theory is to analyze different problems or tasks a computer might encounter, and compare them to one another. For this, many so called complexity classes have been introduced, which incorporate problems that are similar to each other in the way computers deal with them; that is, they have a resembling running time or they utilize similar amounts of memory space. Two very famous complexity classes are **P** and **NP**. The former contains, colloquially spoken, all problems that are *easy to solve*, whereas the latter contains all problems where it is *easy to verify*, whether a given solution is correct. A formal definition of these and other complexity classes will be given later in this thesis. Trivially, all problems in **P** are also in **NP**. The question, whether also all problems in **NP** are in **P** (thus making the two classes equal), is probably the best examined yet still unsolved problem in computer science. Though most people believe the two classes to be different, nobody has yet been able to prove or disprove their equality.

Of course, complexity theory is a very broad area in itself, since there is a vast number of different problems to consider. To name only a few, there are, for example, cryptographic problems and their protocols, optimization problems, interactive proof systems, probabilistic algorithms, and so forth. It is way beyond the scope of this thesis to discuss all of them. Though, another major issue in complexity theory, which we will take a closer look at, are Boolean formulae. A Boolean formula is a mapping from a list of input variables to one output variable, with the restriction that only values from $\{0,1\}$ are allowed. Instead of 0 one often uses *false* and 1 is often identified with *true*. So, a very obvious question is to decide, given a certain input to the variables, whether a

Boolean formula evaluates to true or false. Another, related question is, whether there exists any input to the variables such that a given formula evaluates to true. The latter one is well-known under the name *satisfiability problem* for Boolean formulae. These formulae are the most natural kind of problems in complexity theory when dealing with the polynomial hierarchy, which is a hierarchy of complexity classes based upon the classes **NP** and **P**, because they provide – in different variations – complete problems for most classes of the polynomial hierarchy. For instance, Stephan A. Cook showed already in 1971, that the above mentioned satisfiability problem (known as SAT) is, in its general version, one of the most difficult problems in the class **NP** [Coo71]. Those most difficult problems will be called **NP**-*complete*. Naturally, the complexity of such problems depends to a large extend on the type of Boolean formula that is used. Even though SAT without any restrictions is **NP**-complete, the problem becomes trivial, when we only allow the formulae to be a conjunction of positive literals. Then, any such formula is always satisfiable. The questions that arise are: For what type of formulae is the problem still difficult? What changes it to become easy? And where exactly is that borderline? Another interesting question that comes to mind is, whether this borderline is sharp, or what kind of complexity classes between the "difficult" (**NP**-complete) and "easy" (in **P**) ones can be assumed by Boolean formulae. This question is especially reasonable, because Richard E. Ladner proved that unless **P** = **NP**, there are infinitely many complexity classes between **P** and **NP** [Lad75]. However, an intricacy is, that there are, of course, infinitely many different kinds of Boolean formulae; and it is not clear, how to look at all of these at the same time.

A good way to get to work with this infinite number of Boolean formulae is with the help of *constraints*. Basically, a constraint is simply a relation; and a constraint formula is the conjunction of such constraints. A set of constraints will be called *constraint language*. The disadvantage of using constraints is, that it is not possible to express every Boolean formula with the help of constraints. However, this drawback is compensated on the one hand by the fact that constraints still cover an infinite number of Boolean formulae, and most of the "important" problems, that have already been analyzed through other means, can be modeled by a constraint formula (e. g., 3-SAT, 2-SAT, Horn-formulae, etc.). On the other hand a great advantage of constraint formulae in comparison to "usual" Boolean formulae (of course, every constraint formula is also a Boolean formula, just a restricted version) is, that it is possible to analyze the complexity of constraint formulae in a very succinct way. As we will see, there are several properties of constraints (or constraint languages) and the complexity of all problems, that we will examine, depends solely on these properties. This allows us to state complexity results for an infinite number of Boolean formulae. This method has first been used by Thomas Schaefer in 1978. He classified the *constraint satisfaction problem* (CSP), which is the constraint version of SAT. Surprisingly, he found out, that the problem is in **P**, if the constraint language has some certain well-defined properties, and in all other cases, the CSP is already **NP**-complete [Sch78]. This theorem of his will be used as a starting point for this thesis, which is organized as follows:

After we have introduced all necessary notations and definitions in Chapter 2, we will refine Schaefer's Theorem in Chapter 3 in such a way that we will take a closer look at the Boolean constraint satisfaction problem. Schaefer obtained a dichotomatic complexity result by separating the problem into **P** and **NP**-complete cases. We will examine the tractable cases in more detail and show that these can be divided further into four cases (or five, if we count the trivial ones as a case of their own).

In Chapter 4 we generalize the constraint satisfaction problem by introducing quantifiers for the variables. Then the question is not anymore, whether there is a satisfying assignment for a given formula, but whether the variables are quantified in such a way, that the formula is true. In addition to the decision problem, we also consider its corresponding counting version: There not all variables need to be quantified and we are interested in the number of solutions for the not quantified variables.

Finally, we consider graph related problems in Chapter 5. We start with the constraint analogon to graph isomorphism – the equivalence problem – and its one-sided version – implication. Then, we move on to the isomorphic implication problem, which is the constraint counterpart to subgraph isomorphism. Incidentally, the isomorphic implication problem discloses an interesting and new approach, which could lead to a proof that graph isomorphism is in **P**, which it is not known to be.

The results of Chapter 3 are based on a joint work with Eric Allender, Neil Immerman, Henning Schnoor, and Heribert Vollmer [ABI+05]. Chapter 4 relies on the previous publications [BCC+04] and [BBC+05], which are joint works with Elmar Böhler, Philippe Chapdelaine, Nadia Creignou, Steffen Reith, Henning Schnoor, and Heribert Vollmer. Parts of Chapter 5 have been obtained together with Edith Hemaspaandra and have been published in [BH05].

2 Preliminaries

This chapter contains definitions and theorems, which will be used extensively throughout the paper. We will start with some basic mathematical definitions including Boolean formulae. Then follows an introduction to complexity classes and reductions. Thereafter we will define constraints and finally we will introduce some closure properties, which will enable us to give short and elegant proofs as we will see in Chapters 3 and 4.

2.1 Basics

A basic mathematical object is the *set*, which is a collection of elements. Each such element in a set is called a *member* of that particular set. There are two different ways to describe a set. One is to list all of its elements (e. g., $A = \{a_1, a_2, a_3, a_4\}$) and the other is to describe, what properties all elements in the set possess (e. g., $A = \{a \mid a \text{ has property } E\}$). Instead of listing each and every element, which is not possible for infinite sets and infeasible for very large sets, one can also use ellipses (...). If an element a is a member of a set A, we also write $a \in A$, and accordingly $a \notin A$, if a is not a member of A. Further, we will write $A \subseteq B$ if every member of A is also a member of B, and say A is a *subset* of B; we write $A \subset B$ if $A \subseteq B$, but not $B \subseteq A$, and say A is a *proper subset* of B. If on the other hand $A \subseteq B$ and $B \subseteq A$, we say that A *equals* B, denoted by $A = B$. The set with no elements is called the *empty set* and will be denoted by $\emptyset$. The number of the elements in a set is called its *cardinality* and is denoted by $|A|$ for a set A.

Example 2.1.1 *The sets A, B, and C will be defined by listing all elements, whereas D and E use the description technique: $A = \{2, 3, 5\}$, $B = \{1, 2, \dots\}$, $C = \{1, \dots, 131072\}$, $D = \{a \mid a \text{ is a natural number}\}$, $E = \{a \mid a \text{ is prime}\}$. Obviously, the following holds: $1 \notin A$, $2 \in A$, $A \subset E \subseteq B = D$, $|A| = 3$, and $|\emptyset| = 0$. The set D of natural numbers will be used quite often and is usually denoted by $\mathbb{N}$.*

There are three basic operations on sets: the *union* ($\cup$), *intersection* ($\cap$), and the *difference* ($\setminus$). The union of A and B is defined to be the set of elements, which are in A or in B or in both. An element is in the intersection of A and B, if and only if it is an element of A and an element of B. And finally, $a \in A \setminus B$ if and only if $a \in A$ and $a \notin B$.

An object similar to the set is the *tuple*. It is also a collection of elements; however, contrary to a set, a tuple has to be finite and all elements are ordered. Tuples are written in parentheses (e. g., $(1, 3, 1)$). If n is the number of elements in a tuple u, we also call it an n-tuple or an n-ary tuple. For $n = 2$ and $n = 3$ we use the short forms of *pair* and *triple*. Similarly to sets, the number of elements in a tuple u is also denoted by $|u|$.

Instead of tuple, we sometimes also use the more general expression vector, which, for the scope of this thesis, should always denote a tuple.

There exists a direct connection between sets and tuples, namely the *cross product* (also called *Cartesian product*) of sets, which yields a set of tuples. For $A_1, \dots, A_n$ sets, define $A_1 \times \cdots \times A_n = \{(a_1, \dots, a_n) \mid a_1 \in A_1, \dots, a_n \in A_n\}$. As a special case, A^n for A a set and $n \in \mathbb{N}$ is, analogously to the power of numbers, defined as $A \times \cdots \times A$. Hereby $A^0 = \{()\}$ and A^1 formally is $\{(a) \mid a \in A\}$; however, to simplify matters we will define $A^1 = A$.

An n-ary Boolean *relation* R is a subset of $\{0,1\}^n$. An n-ary Boolean *function* (sometimes also called *mapping*) f is a subset of $\{0,1\}^{n+1}$ with the additional property that if $(a_1, \dots, a_n, b) \in f$ and $(a_1, \dots, a_n, b') \in f$, then $b = b'$, where $a_1, \dots, a_n, b, b' \in \{0,1\}$. Instead of $(a_1, \dots, a_n, b) \in f$ we also write $f(a_1, \dots, a_n) = b$ and say that f maps $(a_1, \dots, a_n)$ to b. Accordingly, instead of $f \subseteq \{0,1\}^{n+1}$ we mainly write $f\colon \{0,1\}^n \to \{0,1\}$, where $\{0,1\}^n$ is the *input* and $\{0,1\}$ is the *output*. As a generalization of the Boolean function we allow the input and output to be arbitrary sets A and B instead of $\{0,1\}^n$; thus, yielding a function $f\colon A \to B$. A function f is called *injective*, if it has the property that $f(a) = f(b)$ implies $a = b$ for any $a, b \in A$. An injective function $f\colon A \to B$ is called *bijective*, if for all $b \in B$ there is an $a \in A$ such that $f(a) = b$, that is, every element from B is mapped to. A *permutation* π is a special kind of function, that maps from one set A to itself, such that $A = \{\pi(a) \mid a \in A\}$.

In the following we will use the standard correspondence between Boolean relations and Boolean formulae; that is, an n-ary Boolean relation R corresponds to an n-ary Boolean formula f in such a way, that a vector v is in R if and only if $f(v) = 1$.

A finite non-empty set Σ is called *alphabet*. A sequence $w = a_1 \dots a_n$ of n elements of Σ for $n \geq 0$ is called a *word* (or sometimes also *string*) over the alphabet Σ and $|w| = n$ is called its *length*. In the special case of $n = 0$ we write $w = \varepsilon$ and call ε the *empty word*. The set of all words over Σ is denoted by Σ^*. Let $A \subseteq \Sigma^*$; then, A is called a *language* over Σ. If $A \subseteq \Sigma^*$ is a language, we define the *complement* of A as $\overline{A} = \Sigma^* \setminus A$.

Example 2.1.2 *Let $\Sigma = \{0,1\}$ be an alphabet. Then the set of all words over Σ is the set $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$. Possible languages over Σ would be $A = \emptyset$, $B = \Sigma^*$, $C = \{1, 10\}$, and $D = \{\varepsilon, 0, 00, 000, \dots\}$. Obviously $\overline{A} = B$ and $\overline{B} = A$.*

Finally, we will define some notions of the propositional logic, in particular formulae and their assignments. Let $X = \{x_1, \dots, x_n\}$ be a set of variables. Then any variable $x_i \in X$ is a *propositional formula*. If φ and ψ are propositional formulae, then also $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \vee \psi$ are propositional formulae. Instead of $\neg\varphi$ we sometimes also write $\overline{\varphi}$. If φ is a propositional formula and $x_1, \dots, x_n$ are all variables occurring in φ, we say that φ is a formula *over* $X = \{x_1, \dots, x_n\}$. For the following, let φ be a propositional formula over variables $X = \{x_1, \dots, x_n\}$. A function I from X to $\{0,1\}$ is called a *truth assignment* (or *assignment*) of φ. We now define inductively what it means that a truth assignment I *satisfies* a given formula φ. If $\varphi = x_i$ for a variable x_i, then I satisfies φ if and only if $I(x_i) = 1$. If $\varphi = \psi_1 \wedge \psi_2$ ($\varphi = \psi_1 \vee \psi_2$), then I satisfies φ if and only if I satisfies ψ_1 and ψ_2 (ψ_1 or ψ_2, resp.). This way, a propositional formula can also be seen

formula	name	description	abbreviation for
φ	*id*	identity	n/a
$\neg\varphi$, $\overline{\varphi}$	*not*	negation	n/a
$\varphi \wedge \psi$	*and*	conjunction	n/a
$\varphi \vee \psi$	*or*	disjunction	n/a
$\varphi \rightarrow \psi$	*impl*	implication	$\neg\varphi \vee \psi$
$\varphi \leftrightarrow \psi$	*equiv*	equivalence	$(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
$\varphi \oplus \psi$	*xor*	exclusive or	$\neg(\varphi \leftrightarrow \psi)$

Table 2.1: List of important Boolean functions

as a Boolean function and for a formula φ over variables $X = \{x_1, \dots, x_n\}$ we thus write $\varphi(x_1, \dots, x_n)$. Depending on the assignment, it calculates either 0 or 1. Table 2.1 gives an overview of the most important Boolean functions. When a formula is considered as a function, the truth assignments are usually written in form of a tuple instead of a function. Thus, $(\alpha_1, \dots, \alpha_n) \in \{0,1\}^n$ is said to satisfy a formula φ, if $\varphi(\alpha_1, \dots, \alpha_n) = 1$. We denote with $\text{sat}(\varphi)$ ($\text{unsat}(\varphi)$) the set of satisfying (unsatisfying, resp.) assignments of φ; that is, $\text{sat}(\varphi)$ ($\text{unsat}(\varphi)$) is the set $A \subseteq \{0,1\}^n$ such that for every $\alpha \in A$ we have that $\varphi(\alpha) = 1$ ($\varphi(\alpha) = 0$, resp.). Further we write $\#\text{sat}(\varphi)$ for $|\text{sat}(\varphi)|$ and $\#\text{unsat}(\varphi)$ for $|\text{unsat}(\varphi)|$ to denote the number of satisfying respectively unsatisfying assignments of φ.

There are two main normal forms of propositional formulae. One is the *conjunctive normal form* (CNF) and the other is the *disjunctive normal form* (DNF). A formula φ is said to be in CNF if it is of the form $\varphi = c_1 \wedge \cdots \wedge c_n$, where $c_i = l_{j_1} \vee \cdots \vee l_{j_{m_i}}$ for $1 \leq i \leq n$ and all l_k are variables or negated variables. The c_i's are called *clauses* and the l_k's are *literals*. A DNF is similarly a disjunction of conjunctions; that is, $\varphi = d_1 \vee \cdots \vee d_n$, where $d_i = l_{j_1} \wedge \cdots \wedge l_{j_{m_i}}$ for $1 \leq i \leq n$ and all l_k are variables or negated variables. The d_i's are then called *disjuncts* and the l_k's are again literals. It can be shown that any propositional formula can be transformed into a CNF and a DNF. If we only allow up to three literals per clause (disjunct), the resulting formula is said to be a 3-CNF (3-DNF, resp.).

An extension to this is a formula with *quantifiers*, where a quantifier is a symbol $\forall$ or $\exists$. For this we first have to introduce the notion of *free variables*. Let φ be a formula over variables X. Then the free variables of φ are all occurring variables. If φ is a (quantified) formula with free variables $X = \{x_1, \dots, x_n\}$, then $\exists x_i\varphi$ and $\forall x_i\varphi$ with $x_i \in X$ are *quantified formulae* with free variables $X \setminus \{x_i\}$. An assignment I of the free variables satisfies an existentially quantified formula $\exists x\varphi$ if and only if there is a value $\alpha \in \{0,1\}$ such that $I \cup \{x = \alpha\}$ satisfies φ. An assignment I of the free variables satisfies a universally quantified formula $\forall x\varphi$ if and only if for all values $\alpha \in \{0,1\}$, we have that $I \cup \{x = \alpha\}$ satisfies φ. If a formula φ has no free variables, it is said to be *closed*, and it is either true or false.

2.2 Complexity

In complexity theory a main objective is to classify problems by their complexity, that is, by their difficulty. In order to do this, we first have to clarify two things: What kind of difficulty are we looking for? And how exactly do we measure it? For example, if the problem is to get from one place to another, we could define the difficulty as the time that is needed to do the journey. Another difficulty might be, how much money it takes to do the journey. Now measuring the time seems not to be a problem (you just take a stop watch and see how long the journey takes), whereas the measurement of money is not so clear anymore. If you take a short trip of a few hundred metres, you can certainly do it for free if you just walk. However, on a trip from Paris to Rome, you could also just walk, but it would take you days and you would have to sleep and eat meanwhile, which would probably cost you some money. So, the question is, what kind of money is included and what is not. Also the measurement of time is not always obvious. The fastest way from Paris to Rome is certainly to fly. However, do you include the time you need to buy a plane ticket, get to the airport, wait for the next plane, and so on? Or do you just take the flying time from one airport to the other?

2.2.1 A Basic Model

Our problems are of course somewhat more theoretical, though, the general principle remains the same. You might, for example, be given a list of numbers and want to sort them. One important complexity is, of course, the time that is needed to sort those numbers. Similar to the practical example above, this certainly depends on the help you have. If you do it by hand, it will take much longer than if you do it with the help of a computer. But, even with a computer, there are large differences of speed mainly depending on the age of the computer, but also depending on how the data is fed to the computer. We therefore would like to have a fixed computational model, which is supplied by the *Turing machine*, or short TM. Basically, a TM is an automaton with a finite number of states. It has an infinite tape divided into cells and a head, which can look at one cell at a time. Each cell contains exactly one out of finitely many symbols. At the start of each run of a TM the input data is assumed to be on the tape starting at the head's position. Depending on the symbol at the head's current location and the state of the machine, it may enter a new state, may change the symbol, and afterwards move the head one cell to the left, one cell to the right, or leave it at the current position. Although this model is very simple, it is as powerful as any programming language. A detailed definition of the Turing machine is omitted here, but a very good introduction can be found in, for example, [Sip97] and [HU79].

Although a TM could produce any kind of output, we will almost only deal with TMs for *decision problems* (with a little exception in Chapter 4). These are TMs designed for a specific language and they decide, whether a word is in that language or not; that is, they either accept or reject their input by entering a unique accepting or rejecting state. For our purposes we still need to define four variants of such a TM.

- The first one is a generalization. Instead of having only one tape, we allow the TM to have a constant number k of tapes. Accordingly, there are also k heads, one for each tape. Such a TM is then called *k-tape Turing machine* or in case k is not known or not important, we call it a *multi-tape Turing machine*. The action of the k-tape TM depends on its state and the content of the tapes at each head's position. All heads can be moved independently from each other.

- Another variant is that the TM has an additional input tape. This input tape is a read-only tape. The purpose will be seen later.

- The third variant is a *nondeterministic* TM (NTM). In contrast to a (deterministic) TM, an NTM is allowed to have more than one possible action for any given pair of state and read symbol. Nondeterminism can then be explained in two ways. The first being that whenever there is more than one possible action the NTM could do, it branches and follows all possibilities in parallel. Each sequence of possible branchings is called a *computation path* or simply path of the TM. The NTM accepts an input if and only if at least one of those paths terminates in an accepting state. We also call such a path an accepting path. The other explanation is that by some "divine intuition" the TM always chooses the right action if such exists, whenever several possibilities occur. Thus, it "guesses" a computation path and if there is at least one accepting path, the NTM will always choose an accepting path.

- Finally, we define the so-called *oracle Turing machine*. This is a general TM with an additional oracle A. Such a TM possesses a separate write-only oracle tape and three distinguished states $q_?$, q_0, and q_1. Whenever the state $q_?$ is reached, the TM either enters one of the distinguished states q_0 or q_1 depending on whether the word on the oracle tape is in A or not. Thus, it has access to the language A with almost no cost. Intuitively such an oracle can be considered as a subroutine of a programming language. Except from this peculiarity the oracle TM works as usual.

A formal definition of nondeterministic and oracle Turing machines can again be found in many text books about theoretical computer science, see, for example, [Sip97] and [HU79] for nondeterminism and [Pap94] for oracles.

2.2.2 Complexity Classes

We finally have a palpable and well-defined model for computations, which allows us to define complexity classes. The *time* that is needed to solve a problem, is defined as the steps it takes a TM to decide it. A second important complexity is the *space* a TM uses during its computation. This is defined as the number of cells on the work tape that are being visited by the TM during its computation. For space complexity we always assume a TM with additional input tape, but the visited cells of that tape are not considered. However, in complexity theory we are not interested in the exact amount of steps taken or cells visited, but just in their order of magnitude. Therefore, the *O-notation* is very useful. If $f, g\colon \mathbb{N} \to \mathbb{N}$, then $f \in O(g)$[1] (pronounced, f is in big oh of g), if there exist $c, n_0 \in \mathbb{N}$, such that for all $n > n_0$ we have $f(n) \leq c \cdot g(n)$. Intuitively this means that for large enough n, the function f is not larger than g (apart from a constant factor).

Of course, time and space depend on the TM and its actual input. So, if we want to classify the complexity of a given problem, we have to leave open the choice of TM and somehow include the input, making the time a function depending on the input size. Since we are only dealing with decision problems, we can say that a complexity class is a set of languages. Thus, the following definitions seem reasonable.

Definition 2.2.1 *Let $s, t\colon \mathbb{N} \to \mathbb{N}$ and $k \in \mathbb{N}$. We define* $\mathbf{TIME}(t)$ *as the set of languages, that can be decided by a k-tape TM in time $O(t)$. Similarly we define* $\mathbf{SPACE}(s)$ *as the set of languages, that can be decided by a k-tape TM with additional input tape in space $O(s)$.*

Similar classes can be defined for nondeterministic Turing machines instead of deterministic ones. Here, the time and space that is needed is always considered to be the maximum through all computation paths.

Definition 2.2.2 *Let $s, t\colon \mathbb{N} \to \mathbb{N}$ and $k \in \mathbb{N}$. We define* $\mathbf{NTIME}(t)$ *as the set of languages, that can be decided by a k-tape NTM in time $O(t)$. Similarly we define* $\mathbf{NSPACE}(s)$ *as the set of languages, that can be decided by a k-tape NTM with additional input tape in space $O(s)$.*

Definition 2.2.3 *Let $\mathcal{C}$ be a set of languages. The* complement *of $\mathcal{C}$ is defined as* $\mathbf{co}\mathcal{C} = \{\overline{A} \mid A \in \mathcal{C}\}$.

With the help of these formalities, we can finally define the complexity classes, which we will encounter throughout this paper.

Definition 2.2.4 *Let $k \geq 1$.*

L *is the class of languages, that can be accepted by a TM in logarithmic space; that is,* $\mathbf{L} = \mathbf{SPACE}(\log n)$.

NL *is the class of languages, that can be accepted by an NTM in logarithmic space; that is,* $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

$\oplus$**L** *is the class of languages, that can be accepted by an NTM in logarithmic space with the additional property, that an input is accepted if and only if the number of accepting paths is odd.*

P *is the class of languages, that can be accepted by a TM in polynomial time; that is,* $\mathbf{P} = \mathbf{TIME}(n^{O(1)})$.

[1] Some text books often write $f = O(g)$ instead, though the meaning remains the same

NP *is the class of languages, that can be accepted by an NTM in polynomial time; that is,* $\mathbf{NP} = \mathbf{NTIME}(n^{O(1)})$.

$\mathbf{\Sigma_0^P}$ $= \mathbf{P}$.

$\mathbf{\Sigma_k^P}$ *is the class of languages, that can be accepted by an NTM in polynomial time with access to a* $\mathbf{\Sigma_{k-1}^P}$*-oracle; that is,* $\mathbf{\Sigma_k^P} = \mathbf{NP}^{\mathbf{\Sigma_{k-1}^P}}$.

$\mathbf{\Delta_0^P}$ $= \mathbf{P}$.

$\mathbf{\Delta_k^P}$ *is the class of languages, that can be accepted by a TM in polynomial time with access to a* $\mathbf{\Sigma_{k-1}^P}$*-oracle; that is,* $\mathbf{\Delta_k^P} = \mathbf{P}^{\mathbf{\Sigma_{k-1}^P}}$.

$\mathbf{\Pi_0^P}$ $= \mathbf{coP} = \mathbf{P}$.

$\mathbf{\Pi_k^P}$ $= \mathbf{co\Sigma_k^P}$

$\mathbf{\Theta_0^P}$ $= \mathbf{P}$.

$\mathbf{\Theta_k^P}$ *is the class of languages, that can be accepted by a TM in polynomial time with a parallel access to an* $\mathbf{\Sigma_{k-1}^P}$*-oracle; that is, all oracle questions have to be calculated before the first one is asked.*

PH $= \bigcup_{k\geq 0}(\mathbf{\Delta_k^P} \cup \mathbf{\Sigma_k^P} \cup \mathbf{\Pi_k^P})$

PSPACE *is the class of languages, that can be accepted by a TM in polynomial space; that is,* $\mathbf{PSPACE} = \mathbf{SPACE}(n^{O(1)})$.

The class **PH** stands for the *polynomial hierarchy*, which was defined by Meyer and Stockmeyer [MS72]. In Figure 2.1 the inclusion structure of the complexity classes is displayed. A line between two classes means that the lower one is a subclass of the higher one. For almost all inclusions it is not known, whether they are strict or not. So far it has only been proved that $\mathbf{NL} \subset \mathbf{PSPACE}$ (see, for example, the space hierarchy theorem in [Pap94]). For all other inclusions strictness is strongly assumed, especially for the famous $\mathbf{P} = \mathbf{NP}$ problem. An indication to their inequality can be seen through reductions and completeness. For a more detailed description of complexity classes, their dependencies and different characterizations, see, for example, [Pap94], [Pip97], or [HO02].

2.2.3 Reductions

A very important tool in complexity theory is the *reduction*. By reducing one language to another language it is possible to compare their complexity.

Definition 2.2.5 *Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be two languages. We say that A is polynomial-time many-one (log-space many-one)* reducible *to B, in signs $A \leq_m^{\mathbf{P}} B$ ($A \leq_m^{\log} B$, resp.), if there exists a function $f\colon \Sigma^* \rightarrow \Delta^*$, such that f is computable in polynomial time (logarithmic space, resp.) and for all $x \in \Sigma^*$ we have that $x \in A$ if and only if $f(x) \in B$. Such a function is then called* reduction function.

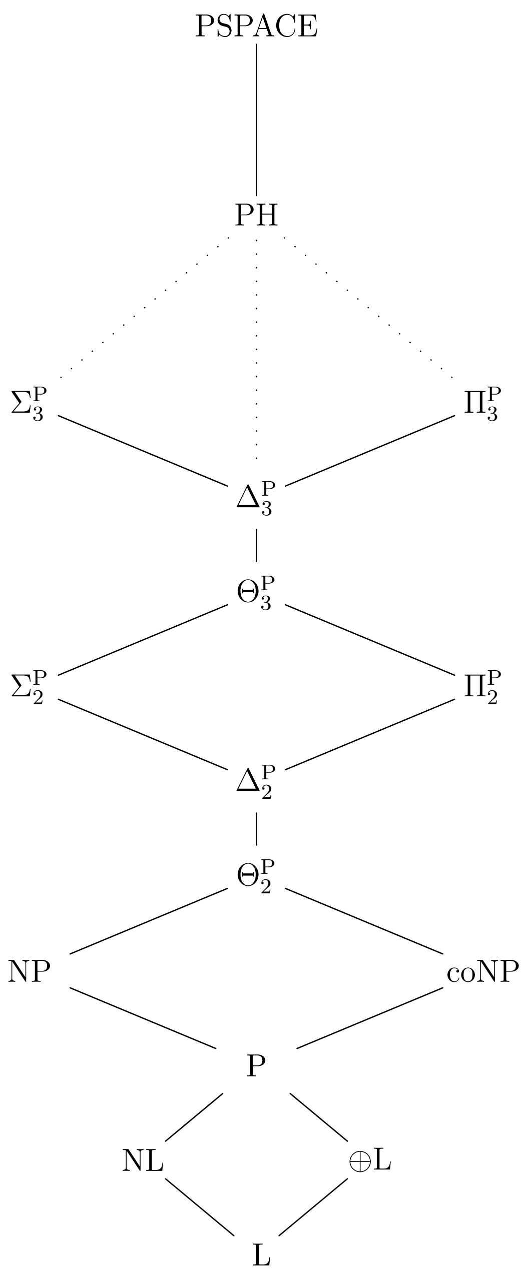

Figure 2.1: Inclusion structure of the complexity classes

The polynomial-time many-one reduction is sometimes also called *Karp reduction* due to Karp, who first introduced this notion of reducibility [Kar72].

Colloquially it means that if a language A can be reduced to a language B, then A cannot be much more difficult than B. Here, the "much" is implicit in the type of reduction, since the reduction function has some power of its own. Thus, for example, the polynomial-time many-one reduction (or in short pm-reduction) is only useful for comparing languages up to a polynomial-time degree. Though it is possible to reduce any language $A \in \mathbf{P}$ to a language $B \in \mathbf{L}$ when using pm-reductions, this is not possible with log-space reductions; then, unless $\mathbf{P} = \mathbf{L}$, there are languages $A \in \mathbf{P}$ with $A \not\leq_m^{\log} B$ for all $B \in \mathbf{L}$.

Next to the two many-one reductions in Definition 2.2.5 there are also other notions of reducibilities. For this thesis the following two are also needed.

Definition 2.2.6 *Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be two languages. We say that A is polynomial-time* truth-table *reducible to B, in signs $A \leq_{tt}^{\mathbf{P}} B$, if A is decidable by an oracle TM M with oracle B in polynomial time, such that M has to compute all oracle queries before the first one is asked. If we additionally require that M accepts if and only if all oracle queries return 1, we have a* conjunctive truth-table *reduction from A to B, in signs $A \leq_{ctt}^{\mathbf{P}} B$.*

When comparing complexities of problems it is also important to be able to express, what it means that two languages have the same complexity referring to a certain reduction.

Definition 2.2.7 *Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be two languages and let $\leq_x^y$ be any reduction. If $A \leq_x^y B$ and $B \leq_x^y A$, we say that A and B are x-y-equivalent and write $A \equiv_x^y B$.*

Since we are now able to compare different problems with each other, we can also talk about *most difficult problems* in a certain complexity class. Therefore, we will introduce the notions of *completeness* and *hardness*.

Definition 2.2.8 *Let A be a language, $\mathcal{C}$ be a complexity class, and $\leq$ be a reduction. We say that A is* $\leq$-hard *for $\mathcal{C}$ (or* $\mathcal{C}$-hard*, if the reduction type is known from the context) if for any language $B \in \mathcal{C}$, we have that $B \leq A$. We say that A is* $\leq$-complete *for $\mathcal{C}$ (or* $\mathcal{C}$-complete*) if additionally $A \in \mathcal{C}$.*

The first problem ever shown to be **NP**-complete is SAT. This is the problem, given a propositional formula φ, to decide, whether there exists a satisfying assignment of φ. This result is due to a work by Stephan A. Cook [Coo71] and was later independently also proved by Leonid Levin [Lev73]. Once the first **NP**-complete problem was known, it was possible to show **NP**-completeness for a whole lot of further problems by simply reducing SAT to these problems, since the reduction is transitive. Garey and Johnson produced a very large compilation of **NP**-complete problems in [GJ79], which is still today a good source of candidates for reductions.

2.3 Constraints

Informally, a constraint can be seen as a kind of restriction to some solution. For example, you have a box of red and blue candies and are allowed to take four candies. Obviously you have five possibilities to choose from[2]. One constraint could be that they must not be all equal. A second constraint could be that the majority should be red. Now the only solution is that you take one blue and three red. Thus, each constraint restricts the set of possibilities.

Formally, an n-ary *constraint* R is an n-ary Boolean relation. A *constraint application* C is an application of an n-ary constraint to an n-tuple of not necessarily distinct variables. An assignment I is said to *satisfy* a constraint application $C = R(x_1, \dots, x_n)$ if and only if $(I(x_1), \dots, I(x_n)) \in R$.

A finite set of constraints is called *constraint language* and will usually be denoted by S. Then, we have an S-clause, which is a constraint application of some constraint $C \in S$. A conjunction of such clauses is called S-formula. If φ is an S-formula and the set of variables occurring in φ is X, then φ is an S-formula over variables X.

Example 2.3.1 *Let $S = \{R_{\text{nae}}, R_{\text{1in3}}, R_{\text{eq}}\}$ be a constraint language with the constraints*

$$\begin{aligned} R_{\text{nae}} &= \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}, \\ R_{\text{1in3}} &= \{(1,0,0),(0,1,0),(0,0,1)\}, \text{ and} \\ R_{\text{eq}} &= \{(0,0),(1,1)\}. \end{aligned}$$

Informally, R_{nae} is the ternary constraint such that not all three variables have the same value, R_{1in3} is true if and only if exactly one of the three variables is set to 1, and finally R_{eq} is the equivalence. For variables x, y, and z, the expressions $R_{\text{1in3}}(x,y,z)$ and $R_{\text{1in3}}(x,x,x)$ are constraint applications, where the former is satisfiable, but the latter is not.

A possible S-clause would be $R_{\text{1in3}}(x,y,z) \wedge R_{\text{eq}}(x,z)$. A satisfying assignment for this clause would have to map x to 0, y to 1, and z to 0. Also, this clause is obviously equivalent to the clause $R_{\text{1in3}}(x,y,x) \wedge R_{\text{1in3}}(z,y,z)$.

There are several properties that classify special types of constraints. The ones which we will use most often are given in the following definition.

Definition 2.3.2 *Let C be a k-ary constraint. Then we say*

- *C is* 0-valid*, if $(0, \dots, 0) \in C$.*
- *C is* 1-valid*, if $(1, \dots, 1) \in C$.*
- *C is* Horn *(sometimes also called* weakly negative*), if C is equivalent to a* CNF *with at most one positive literal in each clause.*
- *C is* anti-Horn *(sometimes also called* weakly positive*), if C is equivalent to a* CNF *with at most one negative literal in each clause.*

[2]Assuming you will actually take four and are not satisfied with three or less

- *C is* bijunctive*, if C is equivalent to a* 2-CNF*; that is, a* CNF *where each clause has at most two literals.*
- *C is* affine*, if C is equivalent to an XOR-CNF; that is, a* CNF *with $\oplus$-clauses.*
- *C is* 2-affine*, if C is affine and bijunctive.*
- *C is* complementive*, if $(\alpha_1, \dots, \alpha_k) \in C$ implies $(\neg\alpha_1, \dots, \neg\alpha_k) \in C$.*
- *C is* Schaefer*, if C is Horn, anti-Horn, bijunctive, or affine.*

These properties can also be extended to constraint languages. A constraint language S is called 0-valid, 1-valid, Horn, anti-Horn, bijunctive, affine, 2-affine, complementive, or Schaefer if and only if every constraint in S has that property.

The following problem, which has first been considered by T. Schaefer in 1978 [Sch78], is the basis of this work. All other problems, which we will look at, are in some way or another derived from it.

Definition 2.3.3 *Let S be a constraint language. The constraint satisfaction problem over S, denoted by* $\mathrm{CSP}(S)$*, is the problem to decide, whether a given S-formula is satisfiable.*

Example 2.3.4 *The well-known* 3-SAT *problem is equivalent to the constraint satisfaction problem over the constraint language $S_{3\text{-SAT}}$, which is defined as $S_{3\text{-SAT}} = \{(x_1 \vee x_2 \vee x_3), (\overline{x_1} \vee x_2 \vee x_3), (\overline{x_1} \vee \overline{x_2} \vee x_3), (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})\}$.*

Schaefer came to the result that depending on the constraint language allowed, the problem is either in **P** or **NP**-complete. He also gave easy criteria to decide which of the two cases occurs.

Theorem 2.3.5 ([Sch78]) *Let S be a constraint language. If S is 0-valid, 1-valid, or Schaefer, then* $\mathrm{CSP}(S)$ *is in* **P***; otherwise* $\mathrm{CSP}(S)$ *is* **NP***-complete.*

This result is quite surprising, since assuming that $\mathbf{P} \neq \mathbf{NP}$ there are infinitely many complexity classes between **P** and **NP** [Lad75].

A question that arises is, how one can determine, whether a given constraint language is, for example, Horn or affine. For this the closure properties of the next section are very helpful. But first, we will introduce the notion of *conjunctive queries*, which can be used to measure the expressive power of a set of constraints. They were thus also an important tool in the method of proof used by Schaefer to prove Theorem 2.3.5.

Definition 2.3.6 *Let S be a constraint language. Then* $\mathrm{COQ}(S)$ *is the set of all relations that can be represented by formulae of the form*

$$\varphi(x_1, \dots, x_k) = \exists y_1 \exists y_2 \dots \exists y_l\, \psi(x_1, \dots, x_k, y_1, \dots, y_l),$$

where ψ is an S-formula. $\mathrm{COQ}(S)$ *is called the set of conjunctive queries over S.*

2.4 Closure Properties and Post's Lattice

2.4.1 Functions and Clones

For a set B of Boolean functions, we say that B is closed under *superposition*, if it contains the identity function *id* and it is closed under

1. introduction of fictive variables: Let f be an n-ary Boolean function in B. Then $g(x_1, \ldots, x_{n+1}) = f(x_1, \ldots, x_n)$ is in the closure of B.

2. permutation of variables: Let f be an n-ary Boolean function in B and let π be a permutation on $\{x_1, \ldots, x_n\}$. Then $g(x_1, \ldots, x_n) = f(\pi(x_1), \ldots, \pi(x_n))$ is in the closure of B.

3. identification of variables: Let f be an n-ary Boolean function in B. Then the function $g(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, x_1)$ is in the closure of B.

4. substitution: Let f be an n-ary function and g be an m-ary function in B. Then $h(x_1, \ldots, x_{m+n-1}) = f(x_1, \ldots, x_{n-1}, g(x_n, \ldots, x_{m+n-1}))$ is in the closure of B.

Points 2 through 4 are equivalent to arbitrary composition of functions. For a more detailed explanation of superposition see, for example, [BCRV03]. A set closed under superposition is also called a *clone* or simply *closed*. The clone of a set of functions B is denoted by $[B]$. If for two sets B and B' we have that $B = [B']$, then B' is said to be a *base* of B. Naturally, a clone may have different bases.

Example 2.4.1 *Let* BF *be the set of all Boolean functions. Then* $\mathrm{BF} = [\{\mathit{and}, \mathit{not}\}] = [\{\mathit{or}, \mathit{not}\}]$.

In the first half of the last century E. Post did some extensive research on the properties of Boolean functions. In that context he identified all closed classes and their inclusion structure [Pos41]. This inclusion structure constitutes a lattice and is therefore often called *Post's lattice* (see Figure 2.2 for a graphical representation[3]). Additionally, Post found a finite base for each of the Boolean clones (see Table 2.2). A proof for the finite bases and a nice general overview of this topic can be found in [Pip97]. Jablonski et al. give a very detailed and complete compendium of Post's classes in [JGK70][4].

2.4.2 Relations and Co-Clones

Similar to the Boolean functions there is also the notion of closure for relations. A relation R is *closed* under a k-ary Boolean function f, if for any, not necessarily distinct, k vectors $m_1, \ldots, m_k \in R$, we have that

$$(f(m_1[1], \ldots, m_k[1]), f(m_1[2], \ldots, m_k[2]), \ldots, f(m_1[n], \ldots, m_k[n])) \in R,$$

[3]by courtesy of Steffen Reith

[4]This version is a German translation of the Russian original work.

Name	**Definition**	**Base**
BF	All Boolean functions	$\{\mathit{or}, \mathit{and}, \mathit{not}\}$
R_0	$\{f \in \mathrm{BF} \mid f$ is 0-reproducing$\}$	$\{\mathit{and}, \mathit{xor}\}$
R_1	$\{f \in \mathrm{BF} \mid f$ is 1-reproducing$\}$	$\{\mathit{or}, \mathit{equiv}\}$
R_2	$\mathrm{R}_1 \cap \mathrm{R}_0$	$\{\mathit{or}, x \wedge (y \leftrightarrow z)\}$
M	$\{f \in \mathrm{BF} \mid f$ is monotonic$\}$	$\{\mathit{or}, \mathit{and}, 0, 1\}$
M_0	$\mathrm{M} \cap \mathrm{R}_0$	$\{\mathit{or}, \mathit{and}, 0\}$
M_1	$\mathrm{M} \cap \mathrm{R}_1$	$\{\mathit{or}, \mathit{and}, 1\}$
M_2	$\mathrm{M} \cap \mathrm{R}_2$	$\{\mathit{or}, \mathit{and}\}$
S_0^n	$\{f \in \mathrm{BF} \mid f$ is 0-separating of degree $n\}$	$\{\mathit{impl}, \mathrm{dual}(h_n)\}$
S_0	$\{f \in \mathrm{BF} \mid f$ is 0-separating$\}$	$\{\mathit{impl}\}$
S_{02}^n	$\mathrm{S}_0^n \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z}), \mathrm{dual}(h_n)\}$
S_{02}	$\mathrm{S}_0 \cap \mathrm{R}_2$	$\{x \vee (y \wedge \overline{z})\}$
S_{01}^n	$\mathrm{S}_0^n \cap \mathrm{M}$	$\{\mathrm{dual}(h_n), 1\}$
S_{01}	$\mathrm{S}_0^n \cap \mathrm{M}$	$\{x \vee (y \wedge z), 1\}$
S_{00}^n	$\mathrm{S}_0 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z), \mathrm{dual}(h_n)\}$
S_{00}	$\mathrm{S}_0 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \vee (y \wedge z)\}$
S_1^n	$\{f \in \mathrm{BF} \mid f$ is 1-separating of degree $n\}$	$\{x \wedge \overline{y}, h_n\}$
S_1	$\{f \in \mathrm{BF} \mid f$ is 1-separating$\}$	$\{x \wedge \overline{y}\}$
S_{12}^n	$\mathrm{S}_1^n \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z}), h_n\}$
S_{12}	$\mathrm{S}_1 \cap \mathrm{R}_2$	$\{x \wedge (y \vee \overline{z})\}$
S_{11}^n	$\mathrm{S}_1^n \cap \mathrm{M}$	$\{h_n, 0\}$
S_{11}	$\mathrm{S}_1 \cap \mathrm{M}$	$\{x \wedge (y \vee z), 0\}$
S_{10}^n	$\mathrm{S}_1^n \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z), h_n\}$
S_{10}	$\mathrm{S}_1 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{x \wedge (y \vee z)\}$
D	$\{f \mid f$ is self-dual$\}$	$\{x\overline{y} \vee x\overline{z} \vee (\overline{y} \wedge \overline{z})\}$
D_1	$\mathrm{D} \cap \mathrm{R}_2$	$\{xy \vee x\overline{z} \vee y\overline{z}\}$
D_2	$\mathrm{D} \cap \mathrm{M}$	$\{xy \vee yz \vee xz\}$
L	$\{f \mid f$ is linear$\}$	$\{\mathit{xor}, 1\}$
L_0	$\mathrm{L} \cap \mathrm{R}_0$	$\{\mathit{xor}\}$
L_1	$\mathrm{L} \cap \mathrm{R}_1$	$\{\mathit{equiv}\}$
L_2	$\mathrm{L} \cap \mathrm{R}_2$	$\{x \oplus y \oplus z\}$
L_3	$\mathrm{L} \cap \mathrm{D}$	$\{x \oplus y \oplus z \oplus 1\}$
V	$\{f \mid f$ is constant or an n-ary *or*-function$\}$	$\{\mathit{or}, 0, 1\}$
V_0	$[\{\mathit{or}\}] \cup [\{0\}]$	$\{\mathit{or}, 0\}$
V_1	$[\{\mathit{or}\}] \cup [\{1\}]$	$\{\mathit{or}, 1\}$
V_2	$[\{\mathit{or}\}]$	$\{\mathit{or}\}$
E	$\{f \mid f$ is constant or an n-ary *and*-function$\}$	$\{\mathit{and}, 0, 1\}$
E_0	$[\{\mathit{and}\}] \cup [\{0\}]$	$\{\mathit{and}, 0\}$
E_1	$[\{\mathit{and}\}] \cup [\{1\}]$	$\{\mathit{and}, 1\}$
E_2	$[\{\mathit{and}\}]$	$\{\mathit{and}\}$
N	$[\{\mathit{not}\}] \cup [\{0\}] \cup [\{1\}]$	$\{\mathit{not}, 1\}$
N_2	$[\{\mathit{not}\}]$	$\{\mathit{not}\}$
I	$[\{\mathit{id}\}] \cup [\{0\}] \cup [\{1\}]$	$\{\mathit{id}, 0, 1\}$
I_0	$[\{\mathit{id}\}] \cup [\{0\}]$	$\{\mathit{id}, 0\}$
I_1	$[\{\mathit{id}\}] \cup [\{1\}]$	$\{\mathit{id}, 1\}$
I_2	$[\{\mathit{id}\}]$	$\{\mathit{id}\}$

Table 2.2: List of all closed classes of Boolean functions and their bases.
The function h_n is defined as:

$$h_n(x_1, \ldots, x_{n+1}) = \bigvee_{i=1}^{n+1} x_1 \wedge x_2 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{n+1}$$

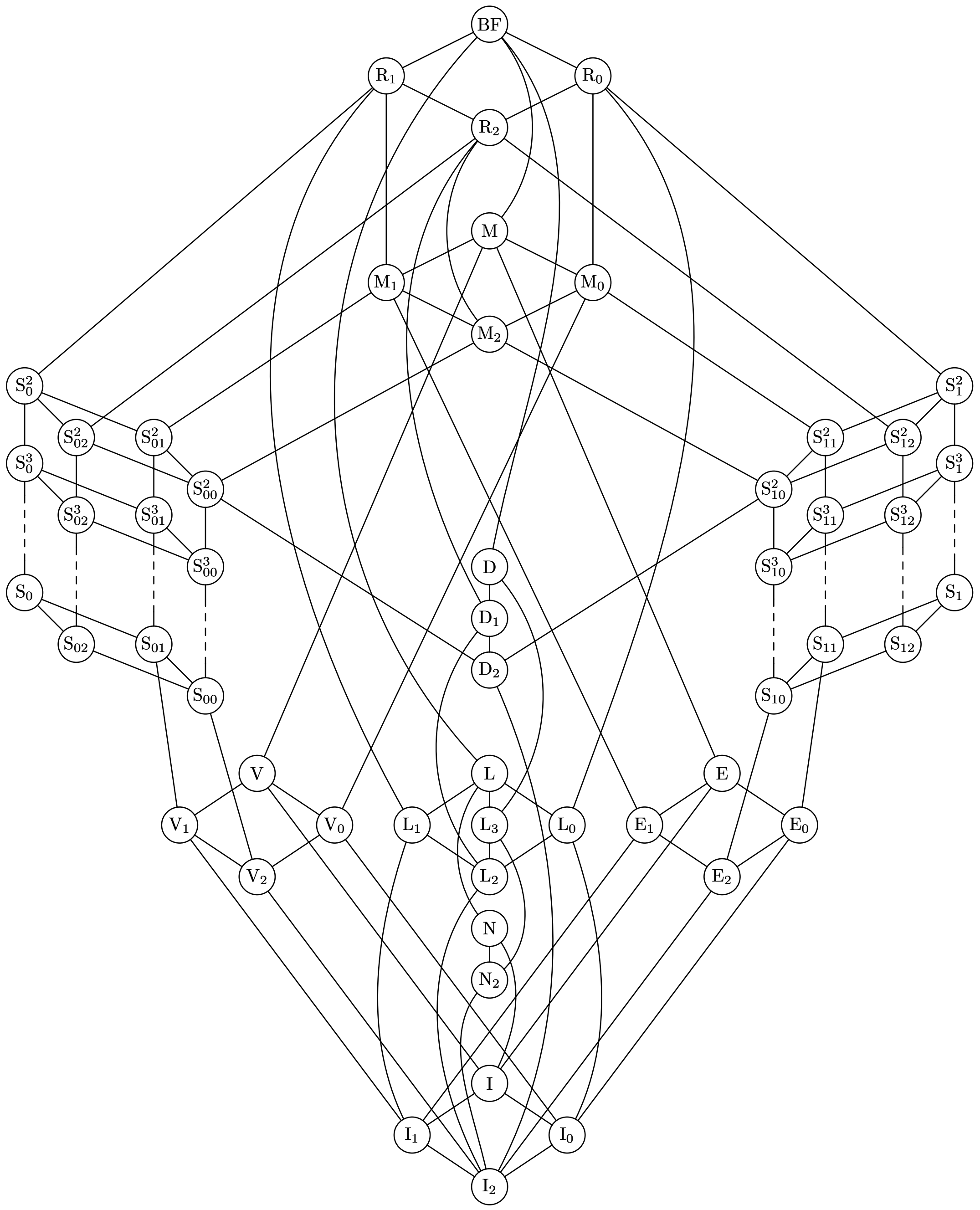

Figure 2.2: Graph of all closed classes of Boolean functions

Clone	**Remark**	**Base(s) of corresponding co-clone**
BF		$\{x \leftrightarrow y\}$, $\{\emptyset\}$
R_0		$\{\overline{x}\}$
R_1	dual of R_0	$\{x\}$
R_2	$\mathrm{R}_0 \cap \mathrm{R}_1$	$\{x, \overline{x}\}$, $\{x \wedge \overline{y}\}$
M		$\{x \rightarrow y\}$
M_0	$\mathrm{M} \cap \mathrm{R}_0$	$\{x \rightarrow y, \overline{x}\}$, $\{\overline{x} \wedge (y \rightarrow z)\}$
M_1	$\mathrm{M} \cap \mathrm{R}_1$	$\{x \rightarrow y, x\}$, $\{x \wedge (y \rightarrow z)\}$
M_2	$\mathrm{M} \cap \mathrm{R}_2$	$\{x \rightarrow y, x, \overline{x}\}$, $\{x \rightarrow y, \overline{x \rightarrow y}\}$, $\{x\overline{y} \wedge (u \rightarrow v)\}$
S_0^n		$\{\mathit{or}^n\}$
S_0	$\bigcap_{n\geq 2} \mathrm{S}_0^n$	$\{\mathit{or}^n \mid n \geq 2\}$
S_{02}^n	$\mathrm{S}_0^n \cap \mathrm{R}_2$	$\{\mathit{or}^n, x, \overline{x}\}$
S_{02}	$\mathrm{S}_0 \cap \mathrm{R}_2$	$\{\mathit{or}^n \mid n \geq 2\} \cup \{x, \overline{x}\}$
S_{01}^n	$\mathrm{S}_0^n \cap \mathrm{M}$	$\{\mathit{or}^n, x \rightarrow y\}$
S_{01}	$\mathrm{S}_0 \cap \mathrm{M}$	$\{\mathit{or}^n \mid n \geq 2\} \cup \{x \rightarrow y\}$
S_{00}^n	$\mathrm{S}_0^n \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{\mathit{or}^n, x, \overline{x}, x \rightarrow y\}$
S_{00}	$\mathrm{S}_0 \cap \mathrm{R}_2 \cap \mathrm{M}$	$\{\mathit{or}^n \mid n \geq 2\} \cup \{x, \overline{x}, x \rightarrow y\}$
S_1^n	dual of S_0^n	$\{\mathit{nand}^n\}$
S_1	dual of S_0	$\{\mathit{nand}^n \mid n \geq 2\}$
S_{12}^n	dual of S_{02}^n	$\{\mathit{nand}^n, x, \overline{x}\}$
S_{12}	dual of S_{02}	$\{\mathit{nand}^n \mid n \geq 2\} \cup \{x, \overline{x}\}$
S_{11}^n	dual of S_{01}^n	$\{\mathit{nand}^n, x \rightarrow y\}$
S_{11}	dual of S_{01}	$\{\mathit{nand}^n \mid n \geq 2\} \cup \{x \rightarrow y\}$
S_{10}^n	dual of S_{00}^n	$\{\mathit{nand}^n, x, \overline{x}, x \rightarrow y\}$
S_{10}	dual of S_{00}	$\{\mathit{nand}^n \mid n \geq 2\} \cup \{x, \overline{x}, x \rightarrow y\}$
D		$\{x \oplus y\}$
D_1	$\mathrm{D} \cap \mathrm{R}_1$	$\{x \oplus y, x\}$
D_2	$\mathrm{D} \cap \mathrm{M}$	$\{x \oplus y, x \rightarrow y\}$, $\{x\overline{y} \vee \overline{x}yz\}$
L		$\{\mathit{even}^4\}$
L_0	$\mathrm{L} \cap \mathrm{R}_0$	$\{\mathit{even}^4, \overline{x}\}$, $\{\mathit{even}^3\}$
L_1	$\mathrm{L} \cap \mathrm{R}_1$	$\{\mathit{even}^4, x\}$, $\{\mathit{odd}^3\}$
L_2	$\mathrm{L} \cap \mathrm{R}_2$	$\{\mathit{even}^4, x, \overline{x}\}$, every $\{\mathit{even}^n, x\}$ where $n \geq 3$ is odd
L_3	$\mathrm{L} \cap \mathrm{D}$	$\{\mathit{even}^4, x \oplus y\}$, $\{\mathit{odd}^4\}$
V		$\{x \vee y \vee \overline{z}\}$
V_0	$\mathrm{V} \cap \mathrm{R}_0$	$\{x \vee y \vee \overline{z}, \overline{x}\}$
V_1	$\mathrm{V} \cap \mathrm{R}_1$	$\{x \vee y \vee \overline{z}, x\}$
V_2	$\mathrm{V} \cap \mathrm{R}_2$	$\{x \vee y \vee \overline{z}, x, \overline{x}\}$
E	dual of V	$\{\overline{x} \vee \overline{y} \vee z\}$
E_0	dual of V_1	$\{\overline{x} \vee \overline{y} \vee z, \overline{x}\}$
E_1	dual of V_0	$\{\overline{x} \vee \overline{y} \vee z, x\}$
E_2	dual of V_2	$\{\overline{x} \vee \overline{y} \vee z, x, \overline{x}\}$
N		$\{\mathit{dup}^3\}$
N_2	$\mathrm{N} \cap \mathrm{L}_3$	$\{\mathit{dup}^3, \mathit{even}^4, x \oplus y\}, \{R_{\mathrm{nae}}\}$
I	$\mathrm{L} \cap \mathrm{M}$	$\{\mathit{even}^4, x \rightarrow y\}$
I_0	$\mathrm{L} \cap \mathrm{M} \cap \mathrm{R}_0$	$\{\mathit{even}^4, x \rightarrow y, \overline{x}\}$, $\{\mathit{dup}^3, x \rightarrow y\}$
I_1	$\mathrm{L} \cap \mathrm{M} \cap \mathrm{R}_1$	$\{\mathit{even}^4, x \rightarrow y, x\}$, $\{x \vee (y \oplus z)\}$
I_2	$\mathrm{L} \cap \mathrm{M} \cap \mathrm{R}_2$	$\{\mathit{even}^4, x \rightarrow y, x, \overline{x}\}, \{R_{\mathrm{1in3}}\}$, $\{x \rightarrow (y \oplus z)\}$

Table 2.3: Bases for all Boolean co-clones.
For the definition of the relations dup^3, even^i, and odd^i see Definition 2.4.5.

where $m_i[j]$ is the value of the jth position of the vector m_i. In other words, the coordinate-wise application of f to any vectors from R yields again a vector from R. This notion of closure of relations is well-known and an easy way to determine what type of relation we are dealing with (compare Example 2.4.2, and for a more detailed description including proofs, see [Sch78] or [CKS01]). Such a function is then also called *polymorphism* of R. The set of all polymorphisms of R is denoted by $\mathrm{Pol}(R)$. If S is a set of relations, then $\mathrm{Pol}(S)$ is the set of functions, which are polymorphisms of every relation in S.

Example 2.4.2 *Let R be a constraint and let $\wedge$, $\vee$, $\oplus$, and maj be applied on vectors coordinate-wise. Here, maj is the ternary majority function, which yields 1 if and only if at least two of its arguments are 1. Then*

- *R is Horn if and only if m, $m' \in R$ implies $m \wedge m' \in R$.*
- *R is anti-Horn if and only if m, $m' \in R$ implies $m \vee m' \in R$.*
- *R is bijunctive if and only if m, m', $m'' \in R$ implies $\mathsf{maj}(m, m', m'') \in R$.*
- *R is affine if and only if m, m', $m'' \in R$ implies $m \oplus m' \oplus m'' \in R$.*
- *R is 0-valid if and only if $m \in R$ implies that the all-0 vector is in R.*
- *R is 1-valid if and only if $m \in R$ implies that the all-1 vector is in R.*

It can easily be shown that $\mathrm{Pol}(S)$ is a clone for any constraint language S, since $\mathrm{Pol}(S)$ is closed under superposition. The following two propositions will turn out to be very useful for our later proofs.

Proposition 2.4.3 *Let S be a constraint language. Then*

1. *S is not Schaefer if and only if $\mathrm{Pol}(S) \subseteq \mathrm{N}$,*
2. *S is 1-valid if and only if $\mathrm{Pol}(S) \supseteq \mathrm{I}_1$, and*
3. *S is 0-valid if and only if $\mathrm{Pol}(S) \supseteq \mathrm{I}_0$.*

Proof. For the first item, note that a constraint language is Schaefer if and only if every relation in S is Horn, anti-Horn, bijunctive, or affine. Looking at Example 2.4.2 and Table 2.2, a relation R is Horn if and only if $\mathrm{Pol}(R) \supseteq \mathrm{E}_2$; a relation R is anti-Horn if and only if $\mathrm{Pol}(R) \supseteq \mathrm{V}_2$; a relation R is bijunctive if and only if $\mathrm{Pol}(R) \supseteq \mathrm{D}_2$; and a relation R is affine if and only if $\mathrm{Pol}(R) \supseteq \mathrm{L}_2$. A further look at Figure 2.2 shows, that all clones except the ones below N are covered by Schaefer cases.

For the cases 2 and 3 it is clear by definition that every relation is b-valid, for $b \in \{0, 1\}$, if and only if the all-b vector is in the relation and this is the case if and only if it is closed under the constant b function, that is, closed under I_b. $\square$

Proposition 2.4.4 $\mathrm{Pol}(R_{1\text{in}3}) = \mathrm{I}_2$.

Proof. First, note that trivially $\mathrm{Pol}(R_{1\text{in}3}) \supseteq \mathrm{I}_2$, since I_2 only contains the identity function and every relation is closed under the identity. In order to prove $\mathrm{Pol}(R_{1\text{in}3}) = \mathrm{I}_2$, we have to show that none of the seven supersets of I_2 is contained in $\mathrm{Pol}(R_{1\text{in}3})$; that is, $R_{1\text{in}3}$ is not closed under those functions. The result then follows, because $\mathrm{Pol}(S)$ is a clone for any set of relations S.

In the following all functions are considered to be applied coordinate-wise. A look at Figure 2.2 shows that we have to examine the classes I_1, I_0, N_2, V_2, E_2, L_2, and D_2. The clones I_1 and I_0 contain the constant 1 respectively constant 0 function. Since neither $(1,1,1) \in R_{1\text{in}3}$ nor $(0,0,0) \in R_{1\text{in}3}$, $\mathrm{Pol}(R_{1\text{in}3}) \not\supseteq \mathrm{I}_1$ and $\mathrm{Pol}(R_{1\text{in}3}) \not\supseteq \mathrm{I}_0$. Obviously $(1,0,0),(0,1,0),(0,0,1) \in R_{1\text{in}3}$. The clone N_2 contains the function *not*, but *not*$((1,0,0)) = (0,1,1) \notin R_{1\text{in}3}$. The clone V_2 contains the function *or*, but $(1,0,0) \vee (0,1,0) = (1,1,0) \notin R_{1\text{in}3}$. Similarly *and* $\in \mathrm{E}_2$, but $(1,0,0) \wedge (0,1,0) = (0,0,0) \notin R_{1\text{in}3}$. The clone L_2 contains the ternary *xor* function, but $(1,0,0) \oplus (0,1,0) \oplus (0,0,1) = (1,1,1) \notin R_{1\text{in}3}$. And finally, the clone D_2 contains the ternary majority function, but *maj*$((1,0,0),(0,1,0),(0,0,1)) = (0,0,0) \notin R_{1\text{in}3}$. □

For a set of Boolean functions B, one can also define a dual operator to $\mathrm{Pol}(S)$, namely $\mathrm{Inv}(B) = \{R \mid B \subseteq \mathrm{Pol}(R)\}$ as the set of *invariants* of B. It turns out that the sets $\mathrm{Inv}(B)$ behave similarly to $\mathrm{Pol}(S)$. All sets $\mathrm{Inv}(B)$ contain the equivalence relation and they are closed under certain operations (see [Pip97] for more details). Therefore, we denote the closure of a constraint language S by $\langle S\rangle$ and call it the *co-clone* generated by S. Table 2.3 displays a list of bases for every co-clone. We adopted the table from the result obtained by Böhler et al. in [BRSV05]. In order to read the table properly, we need to define the following relations.

Definition 2.4.5
- *dup*$^3 = \{0,1\}^3 \setminus \{(0,1,0),(1,0,1)\}$.
- *even*$^i = \{v \subseteq \{0,1\}^i \mid v \textit{ contains an even number of 1's}\}$.
- *odd*$^i = \{v \subseteq \{0,1\}^i \mid v \textit{ contains an odd number of 1's}\}$.

In [BHRV04] it is shown that this closure corresponds to conjunctive queries in the way that $\langle S\rangle = \mathrm{COQ}(S \cup \{R_{\mathrm{eq}}\})$.

The two operators Inv and Pol exhibit an interesting *Galois connection*: For any set B of Boolean functions, $\mathrm{Pol}(\mathrm{Inv}(B)) = [B]$ and for any set S of constraint languages, $\mathrm{Inv}(\mathrm{Pol}(S)) = \langle S\rangle$. One useful property is thus that if for an arbitrary constraint language S and an arbitrary set of Boolean functions B it holds that if $\mathrm{Pol}(S) \subseteq B$, then $\langle S\rangle \supseteq \mathrm{Inv}(B)$, and vice versa. Colloquially this means that the smaller the set of polymorphisms is, the greater is the expressive power of the corresponding conjunctive queries. A basic introduction to the Galois theory can be found in [Pip97, Pös01] and a comprehensive study in [PK79]. The use of this connection can be seen in the following theorem, which is proved, for example, in [Dal00] and [JCG97].

Theorem 2.4.6 *Let S_1 and S_2 be two constraint languages. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then every constraint $R \in S_1$ can be represented by a formula*

$$\begin{aligned} R(x_1, \ldots, x_n) \Leftrightarrow \exists y_1 \ldots \exists y_m R_1(z_{1,1}, \ldots, z_{1,n_1}) \wedge \cdots \wedge R_k(z_{k,1}, \ldots, z_{k,n_k}) \\ \wedge (x_{i_1} = x_{i_2}) \wedge (x_{i_3} = x_{i_4}) \wedge \cdots \wedge (x_{i_{r-1}} = x_{i_r}) \end{aligned}$$

where $R_i \in S_2$ and $z_{i,j} \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$.

Obviously the corollary follows directly.

Corollary 2.4.7 *Let S_1 and S_2 be two constraint languages. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then $\mathrm{COQ}(S_1) \subseteq \mathrm{COQ}(S_2 \cup \{R_{\mathrm{eq}}\})$.*

This result was used in [JCG97] to obtain the following result.

Theorem 2.4.8 *Let S_1 and S_2 be two constraint languages. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then $\mathrm{CSP}(S_1) \leq_m^{\mathbf{P}} \mathrm{CSP}(S_2)$.*

3 The Complexity of Satisfiability Problems

3.1 Introduction

In 1978 Thomas J. Schaefer published his paper "The Complexity of Satisfiability Problems" [Sch78], which had a big impact on the complexity theory community. Up to then it was known that the general satisfiability problem of Boolean formulae is **NP**-complete and some restrictions (like Horn-formulae) were known to be in **P**. Also, the restriction to a CNF with just three literals per clause (i. e., 3-SAT) stays **NP**-complete whereas the same problem with only two literals per clause (i. e., 2-SAT) is already in **P**. It was not clear, whether it is always the case, that the restrictions are either in **P** or **NP**-complete. Especially since Ladner proved that if $\mathbf{P} \neq \mathbf{NP}$, then there are infinitely many complexity classes between **P** and **NP** [Lad75], it was considered rather unlikely. Schaefer examined an infinite family of Boolean formulae, namely the class of those that can be represented by constraints (which cover all of the aforementioned restrictions). For those he proved that if a formula is Horn, anti-Horn, bijunctive, affine, 1-valid, or 0-valid, then the corresponding satisfiability problem is polynomial-time decidable. In all other cases it is **NP**-complete. This dichotomy theorem was the main result of his paper and is now often called Schaefer's Theorem. Schaefer also considered two other problems in that paper. For one, he took a look at quantified constraints, which we will address in the next chapter. On the other hand he also began with a refinement of his dichotomy theorem by looking closer at the results inside **P**. This is what this chapter is about. Schaefer only presented some results below **P** and all without proof. It is also not clear whether the case list presented by him is exhaustive.

For a closer inspection of the results in **P** one has to use an appropriate reduction. In our cases (as did Schaefer) we will use the logarithmic space reduction and obtain a complete list of complexity classes for the CSP. We show that next to the **NP**-complete problems, there are constraint languages for which the CSP is complete for **P**, $\oplus$**L**, and **NL**. In all other cases we have membership in **L**, some of which are trivial (i. e., always satisfiable). Our proofs rely heavily on the connection between complexity of constraint languages and universal algebra, in particular the theory of polymorphisms and clones as introduced in Section 2.4.

3.2 A Complete Classification

The following theorem lists the complete classification of the Boolean constraint satisfaction problem (see Definition 2.3.3) depending on the set of constraints allowed. This theorem is a refinement of Theorem 5.1 from [Sch78] and Theorem 6.5 from [CKS01].

Theorem 3.2.1 *Let S be a constraint language.*

- *If* $\mathrm{Pol}(S) \in \{\mathrm{I}_2, \mathrm{N}_2\}$*, then* $\mathrm{CSP}(S)$ *is* $\leq_m^{\log}$*-complete for* $\mathbf{NP}$.
- *If* $\mathrm{Pol}(S) \in \{\mathrm{V}_2, \mathrm{E}_2\}$*, then* $\mathrm{CSP}(S)$ *is* $\leq_m^{\log}$*-complete for* $\mathbf{P}$.
- *If* $\mathrm{Pol}(S) \in \{\mathrm{L}_2, \mathrm{L}_3\}$*, then* $\mathrm{CSP}(S)$ *is* $\leq_m^{\log}$*-complete for* $\oplus\mathbf{L}$.
- *If* $\mathrm{S}_{00} \subseteq \mathrm{Pol}(S) \subseteq \mathrm{S}_{00}^2$ *or* $\mathrm{S}_{10} \subseteq \mathrm{Pol}(S) \subseteq \mathrm{S}_{10}^2$ *or* $\mathrm{Pol}(S) \in \{\mathrm{D}_2, \mathrm{M}_2\}$*, then* $\mathrm{CSP}(S)$ *is* $\leq_m^{\log}$*-complete for* $\mathbf{NL}$.
- *If* $\mathrm{S}_{02} \subseteq \mathrm{Pol}(S) \subseteq \mathrm{R}_2$ *or* $\mathrm{S}_{12} \subseteq \mathrm{Pol}(S) \subseteq \mathrm{R}_2$ *or* $\mathrm{Pol}(S) \in \{\mathrm{D}_1, \mathrm{D}\}$*, then* $\mathrm{CSP}(S)$ *is in* $\mathbf{L}$.
- *Otherwise* $\mathrm{Pol}(S) \supseteq \mathrm{I}_0$ *or* $\mathrm{Pol}(S) \supseteq \mathrm{I}_1$ *and every S-formula is satisfiable, and therefore* $\mathrm{CSP}(S)$ *is trivial.*

This listing is exhaustive: As can be seen in Figure 3.1, every clone is covered. Obviously the trivial cases are also in $\mathbf{L}$, but due to their special status of being always satisfiable, we separate them from the $\mathbf{L}$ cases. The proof of Theorem 3.2.1 follows from the lemmas in the remainder of this chapter.

3.2.1 Upper Bounds

First, we will state some results, which are already well-known, see, for example, [Sch78, BCRV04].

Proposition 3.2.2 *Let S be a constraint language.*

1. *If* $\mathrm{Pol}(S) \in \{\mathrm{I}_2, \mathrm{N}_2\}$*, then* $\mathrm{CSP}(S)$ *is* $\mathbf{NP}$*-complete. Otherwise,* $\mathrm{CSP}(S) \in \mathbf{P}$.
2. $\mathrm{Pol}(S) \supseteq \mathrm{L}_2$ *implies* $\mathrm{CSP}(S) \in \oplus\mathbf{L}$.
3. $\mathrm{Pol}(S) \supseteq \mathrm{D}_2$ *implies* $\mathrm{CSP}(S) \in \mathbf{NL}$.
4. $\mathrm{Pol}(S) \supseteq \mathrm{I}_0$ *or* $\mathrm{Pol}(S) \supseteq \mathrm{I}_1$ *implies that* $\mathrm{CSP}(S)$ *is trivial.*

The first statement is just a reformulation of Schaefer's Theorem in the way of Post's classes. By Proposition 2.4.3 and a look at Figure 2.2 it is obvious that if a constraint language is neither Schaefer nor 0-valid nor 1-valid, its polymorphisms can only be the clones I_2 or N_2. For the second item note that a base of L_2 is the ternary $\oplus$-function (see Table 2.2). Therefore, we are dealing with affine constraints (see Example 2.4.2) and that

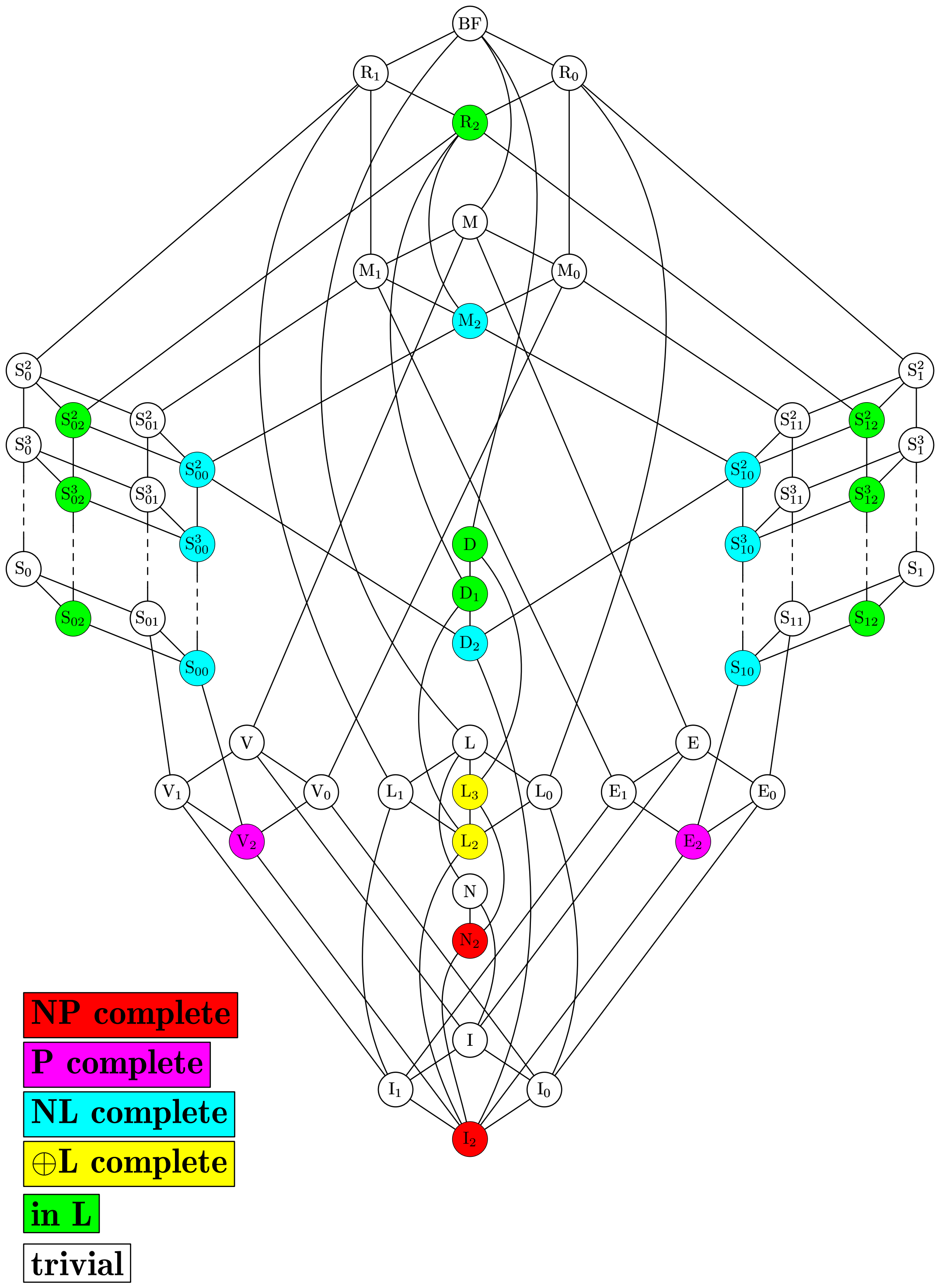

Figure 3.1: The complexity of the constraint satisfaction problem

is the problem to check, whether a system of linear equations over $\{0,1\}$ is consistent, which is in $\oplus\mathbf{L}$ [CKS01, Sch78]. Thirdly, a base for D_2 is $\{xy \vee yz \vee xz\}$ (see Table 2.2), which is the ternary majority function. Hence, we are dealing with bijunctive constraints. Satisfiability is thus equivalent to searching for a path in a directed graph, which is in **NL** [Sch78]. Finally, all instances of $\mathrm{CSP}(S)$, where $\mathrm{Pol}(S) \supseteq \mathrm{I}_0$ or $\mathrm{Pol}(S) \supseteq \mathrm{I}_1$ are satisfiable by the all-0 or the all-1 vector.

Before we start proving lower bounds, we need the result of Theorem 2.4.8 extended to logarithmic space reductions.

Lemma 3.2.3 *Let S_1 and S_2 be two constraint languages. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then $\mathrm{CSP}(S_1) \leq_m^{\log} \mathrm{CSP}(S_2)$.*

Proof. Because of Theorem 2.4.6 we can reduce $\mathrm{CSP}(S_1)$ to $\mathrm{CSP}(S_2 \cup R_{\mathrm{eq}})$ by locally replacing every relation from S_1 by a conjunctive query over S_2 plus some equality constraints. Local replacement is clearly possible in logarithmic space. The existential quantifiers of the conjunctive query can simply be omitted, since we are only looking at the satisfiability of formulae. The equality constraints on the other hand can be eliminated by identifying those variables that are connected by an =-path. By [Rei05] we know that searching for a path in an undirected graph can also be computed in logarithmic space. □

The following lemma proves the remaining **NL** lower bounds by utilizing a similar algorithm as was used in the proof of Theorem 6.5 in [CKS01].

Lemma 3.2.4 *Let S be a constraint language such that $\mathrm{Pol}(S) \supseteq \mathrm{S}_{00}$ or $\mathrm{Pol}(S) \supseteq \mathrm{S}_{10}$. Then $\mathrm{CSP}(S) \in \mathbf{NL}$.*

Proof. Let $\mathrm{Pol}(S) \supseteq \mathrm{S}_{00}$. Since $\mathrm{S}_{00} = \bigcup_{k\geq 2} \mathrm{S}_{00}^k$, there can be no finite set S such that $\mathrm{Pol}(S) = \mathrm{S}_{00}$. However, S is finite, and hence, there exists an $k \geq 2$ with $\mathrm{Pol}(S) \supseteq \mathrm{S}_{00}^k$. As can be seen in the list of bases for co-clones (see Table 2.3), the set $S' = \{\mathsf{or}^k, x, \overline{x}, x \rightarrow y\}$ is a base for the co-clone of S_{00}^k; and thus $\mathrm{Pol}(S') = \mathrm{S}_{00}^k$. Thus, we have $\mathrm{Pol}(S') \subseteq \mathrm{Pol}(S)$ and by Lemma 3.2.3 it follows that $\mathrm{CSP}(S) \leq_m^{\log} \mathrm{CSP}(S')$. Therefore, it suffices to prove that $\mathrm{CSP}(S') \in \mathbf{NL}$. Now the algorithm works as shown in Figure 3.2:

With the notion $\rightarrow$-=-path we mean a sequence $yR_1z_1, z_1R_2z_2, \ldots, z_{m-1}R_mx$ for $R_i \in \{\rightarrow, =\}$. The algorithm clearly works in logarithmic space, since next to saving the pointers to variables of the formula, it is only necessary to look for an $\rightarrow$-=-path, which is an instance of the graph accessibility problem on directed graphs; and thus it is in **NL**.

For the correctness of the algorithm observe that a positive clause (including positive literals) in the formula is satisfiable if and only if at least one of its variables can be set to 1. And that is the case, if there is no $\rightarrow$-=-path to a negated variable.

The S_{10} case works analogously. Just replace or by nand and look for an $\rightarrow$-=-path from a positive literal to variables in the nand-clause (including negative literals) instead. □

A similar algorithm also works for proving the **L**-membership for constraint languages with polymorphisms above S_{02} or S_{12}, as we will show next.

```
INPUT: formula $\varphi$ over relations from $S'$ plus equality
  for every clause $\psi = x_{i_1} \vee \cdots \vee x_{i_k}$ in $\varphi$ do
    for every $y \in \{x_{i_1}, \ldots, x_{i_k}\}$ do
      for every clause $\overline{x}$ in $\varphi$ do
        if there is an $\rightarrow$-$=$-path from $y$ to $x$ then
          $y$ cannot be set to 1
        end if
      end for
    end for
    if at least one $y \in \{x_{i_1}, \ldots, x_{i_k}\}$ could be set to 1 then
      $\psi$ is satisfiable
    else
      reject
    end if
  end for
  accept
```

Figure 3.2: Algorithm for checking satisfiability

Lemma 3.2.5 *Let S be a constraint language such that* $\mathrm{Pol}(S) \supseteq \mathrm{S}_{02}$ *or* $\mathrm{Pol}(S) \supseteq \mathrm{S}_{12}$. *Then* $\mathrm{CSP}(S) \in \mathbf{L}$.

Proof. Let $\mathrm{Pol}(S) \supseteq \mathrm{S}_{12}$. Similar to the previous proof, there is no finite set S such that $\mathrm{Pol}(S) = \mathrm{S}_{12}$. So, we have $\mathrm{Pol}(S) \supseteq \mathrm{S}_{12}^k$ for some k and accordingly it suffices to prove $\mathrm{CSP}(S') \in \mathbf{L}$ for $S' = \{\mathit{nand}^k, x, \overline{x}\}$, which is a base for S_{12}^k (see Table 2.3).

The algorithm is essentially the same as in the previous proof. The only difference is that instead of looking for an $\rightarrow$-$=$-path, we are looking for an $=$-path. This is the graph accessibility problem for undirected graphs, which is in $\mathbf{L}$.

Again the case that $\mathrm{Pol}(S) \supseteq \mathrm{S}_{02}$ works analogously, since we have *or*-clauses instead of *nand*-clauses. $\square$

Finally, the remaining upper bound occurs if the polymorphisms are either D_1 or D. In these cases the result follows immediately from [AG00].

Lemma 3.2.6 *Let S be a constraint language, such that* $\mathrm{Pol}(S) \in \{\mathrm{D}_1, \mathrm{D}\}$. *Then* $\mathrm{CSP}(S) \in \mathbf{L}$.

Proof. Let $R = x_1 \wedge (x_2 \oplus x_3)$. Table 2.3 shows that a base for the co-clone of D_1 is $\{x \oplus y, x\}$; thus $\mathrm{Pol}(\{R\}) = \mathrm{D}_1$. Since $\mathrm{D}_1 \subseteq \mathrm{D}$ and $\mathrm{Pol}(\{R\}) = \mathrm{D}_1$, by Lemma 3.2.3 it suffices to prove that $\mathrm{CSP}(\{R\}) \in \mathbf{L}$. Àlvarez and Greenlaw showed that the satisfiability problem for formulae that are conjunctions of clauses of the form x or $x \oplus y$ is complete for **SL** (see Problem 4.1 in Section 7 of [AG00]). Thus, by [Rei05] we have membership in $\mathbf{L}$ for $\mathrm{CSP}(\{x_1 \wedge (x_2 \oplus x_3)\})$. $\square$

3.2.2 Lower Bounds

We will start with the **P** lower bounds, since the lower bounds for **NP** are already known (see Proposition 3.2.2).

Lemma 3.2.7 *Let S be a constraint language such that* $\mathrm{Pol}(S) \in \{\mathrm{E}_2, \mathrm{V}_2\}$*. Then* $\mathrm{CSP}(S)$ *is $\leq_m^{\log}$-hard for* **P***.*

Proof. A base for the clone E_2 is made up of the *and*-function and similarly $[\{\mathit{or}\}] = \mathrm{V}_2$. As stated in Example 2.4.2 (and proved in [CKS01]) a relation is Horn if and only if it is closed under conjunction, and a relation is anti-Horn if and only if it is closed under disjunction. It is well-known that the satisfiability problem for both Horn and anti-Horn formulae are $\leq_m^{\log}$-complete for **P**. For Horn see, for example, [GHR95], and for anti-Horn there is a trivial reduction from Horn satisfiability: Simply negate all literals of the Horn formula. The result is an anti-Horn formula, which is satisfiable if and only if the original Horn formula is (via the dual assignment). □

Lemma 3.2.8 *Let S be a constraint language such that* $\mathrm{Pol}(S) \subseteq \mathrm{M}_2$*. Then* $\mathrm{CSP}(S)$ *is $\leq_m^{\log}$-hard for* **NL***.*

Proof. Looking at the list of bases for co-clones (see Table 2.3), one can see that a possible base for the co-clone of M_2 is $\{x \to y, x, \overline{x}\}$. Thus, for every S with $\mathrm{Pol}(S) \subseteq \mathrm{M}_2$ we can express $x \to y$, x, and $\overline{x}$. Therefore, the complement of the graph accessibility problem for directed graphs easily reduces to $\mathrm{CSP}(S)$: Let G be a directed graph and s, t be vertices. For every edge (u, v) in G add a constraint $u \to v$. Further add constraints s for the source and $\overline{t}$ for the target. Obviously there is no path in G from s to t if and only if all constraints are simultaneously satisfiable. Immerman [Imm88] and Szelepcsényi [Sze88] showed that **NL** is closed under complement, so the lemma follows with Lemma 3.2.3. □

In order to show $\oplus$**L**-hardness we will reduce from a problem called $\mathrm{SAT}^{\mathrm{C}}(B)$. This is the satisfiability problem for Boolean B-circuits, where B is a set of Boolean formulae. A B-circuit C is an acyclic graph. The vertices of C are called *gates*. Every gate has exactly one outgoing edge (out-degree is 1) and none or some incoming edges (in-degree is ≥ 0). A gate g with in-degree 0 is called an input gate and is assigned one input variable x. The value of g is $\mathrm{val}(g) = x$. All other gates are assigned a function from B, which they apply to the values of their incoming edges and the result is propagated to their outgoing edge. Every B-circuit has exactly one distinguished output gate. This is the gate calculating the function of the circuit. A formal definition can be found in [Rei01].

Lemma 3.2.9 *Let S be a constraint language such that* $\mathrm{Pol}(S) \in \{\mathrm{L}_2, \mathrm{L}_3\}$*. Then* $\mathrm{CSP}(S)$ *is $\leq_m^{\log}$-hard for* $\oplus$**L***.*

Proof. From [Rei01] we know that $\mathrm{SAT}^{\mathrm{C}}(\{x \oplus y\})$ is $\leq_m^{\log}$-complete for $\oplus$**L**. Thus, when reducing from $\mathrm{SAT}^{\mathrm{C}}(\mathrm{L}_0)$ we have a Boolean circuit C with only $\oplus$-gates. The idea of the reduction is to create a constraint for each gate in C such that the function of the gate is expressed by this constraint.

First, assume that $\mathrm{Pol}(S) = \mathrm{L}_2$. We show that $\mathrm{SAT}^{\mathrm{C}}(\mathrm{L}_0) \leq_m^{\log} \mathrm{CSP}(S)$. Let C be an $\{x \oplus y\}$-circuit. For each gate g in C use a new variable x_g. Then create constraints as follows:

- If g is an $(x \oplus y)$-gate with g_x and g_y being its predecessor gates, create a constraint $x_g = x_{g_x} \oplus x_{g_y}$.
- If g is the output gate, create a constraint x_g.

Thus, we have a conjunction of constraints of the type $x_1 = x_2 \oplus x_3$ and x. Call this conjunction φ. Looking at the list of bases in Table 2.3, we see that the relation represented by x is in the base of the co-clone of L_2; that is, $\{1\}$ is closed under L_2. Furthermore, the relations represented by $\overline{x}$ and $\mathit{even}^4(u, v, w, x)$ are also in the base of the co-clone of L_2, where $\mathit{even}^4(u, v, w, x)$ returns 1 if and only if an even number of the four variables is 1 (see Definition 2.4.5). Since $x_1 = x_2 \oplus x_3$ is equivalent to $\mathit{even}^4(x_1, x_2, x_3, 0)$ with 0 being equivalent to $\exists x\ \mathit{even}^4(x, x, x, \overline{x})$, the constraint $x_1 = x_2 \oplus x_3$ is also closed under L_2. Since we only have local replacements, φ is computable in logarithmic space. We still need to show that $C \in \mathrm{SAT}^{\mathrm{C}}(\mathrm{L}_0)$ if and only if $\varphi \in \mathrm{CSP}(S)$.

Let $C \in \mathrm{SAT}^{\mathrm{C}}(\mathrm{L}_0)$. Let further $I = (\alpha_1, \ldots, \alpha_n)$ be a satisfying assignment of C. Now assign to each variable in φ the value the corresponding gate in C has, when C is given the assignment I to its input gates. Obviously, all introduced constraints are satisfied with this variable assignment.

Let $\varphi \in \mathrm{CSP}(S)$ and let $x_1, \ldots, x_n$ be the variables of φ. Let further $I = (\alpha_1, \ldots, \alpha_n)$ be a satisfying assignment of φ. Now assign to each input gate g of C the value of the corresponding variable in φ; that is, $\mathrm{val}(g) = \alpha_g$. It can easily be shown by induction that for all $g \in C$, $\mathrm{val}(g) = \alpha_g$. Since this is true for the output gate as well, and the clause x_g (for $g \in C$ the output gate of the circuit) exists in φ, the circuit value is 1.

The same proof does not work for $\mathrm{Pol}(S) = \mathrm{L}_3$, since it is neither possible to express x nor $\overline{x}$. However, when looking at Post's graph (see Figure 2.2) it is obvious that L_3 is the union of L_2 and N_2. Thus, basically we have negation as a new polymorphism in addition to the functions from L_2. Now it is possible to extend any relation from $\mathrm{Inv}(\mathrm{L}_2)$ such that it is also closed under N_2 by simply doubling the truth-table. More formally, for a constraint language S with $\mathrm{Pol}(S) = \mathrm{L}_2$ we generate a constraint language S', such that $\mathrm{CSP}(S) \leq_m^{\log} \mathrm{CSP}(S')$ and $\mathrm{Pol}(S') = \mathrm{L}_3$. Let therefore R be an n-ary relation closed under L_2. We define $\overline{R} = \{(\overline{x_1}, \ldots, \overline{x_n}) \mid (x_1, \ldots, x_n) \in R\}$ and the $(n+1)$-ary relation $R' = (\{0\} \times R) \cup (\{1\} \times \overline{R})$. By construction R' is closed under N_2. For closure under L_2 we have to show that for any vectors $u = (u_0, u_1, \ldots, u_n), v = (v_0, v_1, \ldots, v_n), w = (w_0, w_1, \ldots, w_n) \in R'$ also $u \oplus v \oplus w \in R'$. Note, that for any vectors a, b, and c it holds that (1) $a \oplus b \oplus c = \overline{a} \oplus \overline{b} \oplus c$ and (2) $a \oplus b \oplus c = \overline{\overline{a} \oplus \overline{b} \oplus \overline{c}}$. Because of (1) we only have to consider the two cases that $u_0 = v_0 = w_0 = z$ for $z \in \{0, 1\}$. If $z = 0$, then $u \oplus v \oplus w \in R'$, since R is closed under L_2. For the case $z = 1$, let $u' = (v_1, \ldots, v_n)$, $v' = (v_1, \ldots, v_n)$, and $w' = (v_1, \ldots, w_n)$. Since $u', v', w' \in \overline{R}$, it follows that $\overline{u'}, \overline{v'}, \overline{w'} \in R$.

Since R is closed under L_2, also $\overline{u'} \oplus \overline{v'} \oplus \overline{w'} \in R$. Then $\overline{\overline{u'} \oplus \overline{v'} \oplus \overline{w'}} \in \overline{R}$ and by (1) and (2) $u' \oplus v' \oplus w' \in \overline{R}$. Thus, R' is also closed under L_2, and hence also under L_3. Let

$$\varphi \quad = \quad \bigwedge_{i=1}^{n} R_n(x_{i_1}, \ldots, x_{i_{n_i}})$$

be an instance of $\mathrm{CSP}(S)$ and let $S' = \{R' \mid R \in S\}$. We set

$$\varphi' \quad = \quad \bigwedge_{i=1}^{n} R'_n(t, x_{i_1}, \ldots, x_{i_{n_i}}),$$

where t is a new variable. We now show that $\varphi \in \mathrm{CSP}(S)$ if and only if $\varphi' \in \mathrm{CSP}(S')$.

Let $\varphi \in \mathrm{CSP}(S)$ and let I be a satisfying assignment for φ. Then obviously $I \cup \{t = 0\}$ is a satisfying assignment for φ'.

Let $\varphi' \in \mathrm{CSP}(S')$ and let I' be a satisfying assignment for φ'. Without loss of generality let I' be an assignment that sets t to 0 (otherwise $\overline{I'}$ is a satisfying assignment for φ', which sets t to 0). Then by construction of φ', the assignment I' without t is also a satisfying assignment for φ. This concludes the reduction and thus $\mathrm{CSP}(S)$ is $\leq_m^{\log}$-hard for $\oplus\mathbf{L}$, if $\mathrm{Pol}(S) = \mathrm{L}_3$. $\square$

This concludes the proofs for Theorem 3.2.1. Looking at Post's lattice one can easily verify that all cases have been covered.

3.3 Conclusion

In this chapter we have been able to refine Schaefer's Theorem, which states that all constraint satisfaction problems are either difficult (i. e., **NP**-complete) or easy (i. e., in **P**). We proved that there are very different complexity classes for the easy cases, if we take a reduction that is fine enough. There are some that are actually easy (namely those that are always satisfiable), whereas some others are more difficult, like the Horn formulae that are **P**-complete. The open question that remains is, whether it is possible to get an even more precise refinement by looking at the cases which we classified to be in **L**. Thus, instead of using logspace reductions, one could use AC^0 reductions[1]. A result in this direction has already been obtained by Allender et al. [ABI+05].

[1]Sometimes also called FO reductions. These are reductions computable by first-order formulae with addition and multiplication operators. For more details see [Vol99, Imm99].

4 Quantified Constraints

4.1 Introduction

As briefly mentioned in the previous chapter, quantified constraints, that is, constraints with all variables existentially or universally quantified, have already been looked at in Schaefer's work [Sch78]. Though, he only presented a result without proof. Dalmau [Dal97] and Creignou, Khanna, and Sudan [CKS01] continued his ideas and confirmed the dichotomy result that the problem to decide, whether a fully quantified formula is true or false, is either in **P** or **PSPACE**-complete. Edith Hemaspaandra [Hem04] was the first to combine this problem with the satisfiability problems of quantified formulae restricted to a bounded alternation of quantifiers. The latter problems are complete for the classes in the polynomial hierarchy [MS72]. We reprove her results using the Galois theory from Section 2.4, which yields a much shorter proof. However, this result just serves as a starting point for our main study, the complexity of quantified counting problems; that is, we are given a quantified constraint formula with some free variables and we want to count the number of satisfying assignments to the formula. For this purpose we introduce a new type of reduction, which we call complementive reduction. This is a generalization of the subtractive reduction introduced by Durand, Hermann, and Kolaitis [DHK00]. The advantage of complementive reductions is that they are strict enough to close most relevant counting classes, but at the same time they are wide enough to allow us to obtain hardness results for many problems. Complementive reductions are likely to turn out useful in other contexts, especially when the problems have some symmetric properties.

As a special case of quantified formulae, we also implicitly get a full classification for conjunctive queries. These conjunctive queries play an important role in database theory, since they represent a broad class of different queries, and their expressive power is equivalent to select-join-project queries in relational algebra. Additionally, conjunctive query containment is considered to be a fundamental problem in database query evaluation and optimization [AHV95]. Recent research points out that query containment is a central problem in several database and knowledge applications, such as data integration [Len02] or data warehousing [Wid95].

4.2 Prerequisites

Apart from the general preliminaries of Chapter 2, we need some extra concepts only for this chapter. On the one hand we have the quantified formulae and related problems and on the other hand there are the counting complexity classes and their reductions.

4.2.1 Quantified Formulae

Let S be a constraint language. A quantified S-formula φ is a closed formula of the form $\varphi = Q_1X_1Q_2X_2 \dots Q_nX_n\psi$, where $Q_1, \dots, Q_n \in \{\forall, \exists\}$ and ψ is an S-formula over the variables $X_1 \cup \dots \cup X_n$. If the set S is not important, we also call φ a quantified Boolean formula or short QBF. The *quantified constraint satisfaction problem* QCSP(S) is the problem to decide, whether a given quantified S-formula is true. This problem is trivially in **PSPACE**, since one simply has to do an exhaustive search on all possible assignments. The general problem, that is, with no restrictions to the set of constraints or the number of quantifiers, is **PSPACE**-complete. It is still **PSPACE**-complete, if the constraint language is restricted to $S_{\text{3-SAT}}$, that is, the constraints necessary to build a 3-CNF formula (see Example 2.3.4). As it was the case with the basic CSP, also QCSP(S) has been classified according to the allowed constraint languages S. Surprisingly the result is again a dichotomy, though there are infinitely many classes between **P** and **PSPACE**-complete.

Theorem 4.2.1 ([Sch78, Dal97, CKS01]) *Let S be a constraint language. If S is Schaefer, then* QCSP(S) *is in* **P***; otherwise,* QCSP(S) *is* **PSPACE***-complete.*

The second possibility to restrict a QBF is on the number of quantifiers, or better quantifier alternations, which such a formula may possess. The result therefor is also well-known, since these problems are prototypical for the polynomial hierarchy. We define $\mathrm{Q}_\exists\mathrm{SAT}_i$ ($\mathrm{Q}_\forall\mathrm{SAT}_i$) as the set of all true, closed Boolean formulae, starting with an existential quantifier (starting with a universal quantifier, resp.) and having exactly $i-1$ quantifier alternations. Wrathall showed that each of these problems $\mathrm{Q}_\exists\mathrm{SAT}_i$ for $i \geq 1$ is complete for the class $\mathbf{\Sigma_i^P}$ [Wra77]. Accordingly each problem $\mathrm{Q}_\forall\mathrm{SAT}_i$ for $i \geq 1$ is complete for the class $\mathbf{\Pi_i^P}$. As it is the case with QCSP, these problems remain complete for their respective classes, if the formulae are restricted to normal forms: For $\mathrm{Q}_\exists\mathrm{SAT}_i$ with odd i and $\mathrm{Q}_\forall\mathrm{SAT}_i$ with even i this holds for restriction to 3-CNF and for $\mathrm{Q}_\exists\mathrm{SAT}_i$ with even i and $\mathrm{Q}_\forall\mathrm{SAT}_i$ with odd i this holds for restriction to 3-DNF. These results are also due to [Wra77]. Hemaspaandra examined both restrictions at the same time by looking at alternation-bounded quantified generalized Boolean formulae [Hem04]. For this she introduced the following notations, which we will adopt in this thesis.

Definition 4.2.2 *Let S be a constraint language and $i \geq 1$; let $X_1, \dots, X_i$ be pairwise disjunct sets of variables, and let ψ be a quantifier-free S-formula defined over variables $\bigcup_{j=1}^{i} X_j$. Then*

- *a $\Sigma_i(S)$-formula φ is an expression of the form $\varphi = \exists X_1 \forall X_2 \dots Q_iX_i\psi$, where $Q_i = \exists$ for i odd and $Q_i = \forall$ for i even, and*
- *a $\Pi_i(S)$-formula φ is an expression of the form $\varphi = \forall X_1 \exists X_2 \dots Q_iX_i\psi$, where $Q_i = \exists$ for i even and $Q_i = \forall$ for i odd.*

4.2.2 Counting

In this section we will introduce counting problems, their complexity classes, and the necessary reductions to classify these problems. First, what is a counting problem? So far we have always encountered decision problems, that is, problems, whose instances are either true or false, that belong to a language or do not belong to it. A prominent example is the SAT problem. A formula is either satisfiable or unsatisfiable. However, counting problems are interested in numbers. The matching counting problem for SAT is #SAT and the question is, given a Boolean formula φ, how many (different) satisfying assignments for φ are there. More generally let $B \subseteq \Sigma^* \times \Gamma^*$ be a binary relation; then y is called a *witness* for x if $(x, y) \in B$. We only consider relations B such that for each x there are only finitely many witnesses. In the example of SAT the corresponding binary relation SAT$'$ is defined as $\{(\varphi, I) \mid \varphi$ is a Boolean formula such that I satisfies $\varphi\}$ and thus a witness for a formula would be a satisfying truth assignment to its variables. Further, the *witness function* for the relation B is the function $w \colon \Sigma^* \to \mathcal{P}^{<\omega}(\Gamma^*)$, with $w(x) = \{y \in \Gamma \mid y$ is a witness for $x\}$, where $\mathcal{P}^{<\omega}(\Gamma^*)$ is the collection of all finite subsets of Γ^*. As a shorthand we also write $B(x)$ instead of $w(x)$. Thus, the counting problem for a given B is the problem to determine the cardinality of $w(x)$ and is usually denoted by $\#B$. In the case of SAT (as for many other decision problems), the counting version is simply denoted by #SAT instead of #SAT$'$ for its binary relation.

The first one to examine counting complexity classes in detail was Valiant [Val79a, Val79b]. He introduced therefor the class $\#\mathbf{P}$ as the class of functions f, such that there exits a polynomial time NTM M and for all x, $f(x)$ is the number of accepting computation paths of M with input x. In the case of a language in **NP**, these accepting paths correspond to the number of witnesses. Thus, for all problems A in **NP**, their counting version $\#A$ is in $\#\mathbf{P}$. For the prototypical problem #SAT, Valiant also proved that it is $\#\mathbf{P}$-complete under parsimonious reductions. A *parsimonious reduction* (denoted by $\leq_!$) is a polynomial-time reduction that additionally preserves the cardinality of the witness sets; that is, for two counting problems $\#A$ and $\#B$, $\#A \leq_! \#B$ if and only if there exists a total, polynomial-time computable function f such that for every x holds $|w_A(x)| = |w_B(f(x))|$, where w_A and w_B are the witness functions of the counting problems $\#A$ respectively $\#B$. Another complexity class in this context is **FP**. This is the set of functions, which are computable in polynomial time. Hence, trivially $\mathbf{FP} \subseteq \#\mathbf{P}$.

A framework for higher complexity counting classes has been introduced by Toda [Tod91]. It is based on predicates and focuses on the complexity of membership in the witness sets. Specifically, if $\mathcal{C}$ is a complexity class of decision problems, then $\#{\cdot}\mathcal{C}$ is the class of all counting problems whose witness function w satisfies the following conditions:

- There is a polynomial $p(n)$ such that for every x and every $y \in w(x)$, we have that $|y| \leq p(|x|)$.
- The witness recognition problem "given x and y, is $y \in w(x)$?" is in $\mathcal{C}$.

In particular, $\#{\cdot}\mathbf{NP}$ is the class of counting problems associated with decision problems, for which the witness size is polynomially bounded and the witness recognition problem is in $\mathbf{NP}$. Following Toda [Tod91], the inclusions $\#\cdot\mathbf{\Sigma^P_k} \subseteq \#\cdot\mathbf{\Pi^P_k}$ and $\#\cdot\mathbf{\Pi^P_k} \subseteq \#\cdot\mathbf{\Sigma^P_{k+1}}$ among counting classes hold for each k. In particular, we have the inclusion $\#\cdot\mathbf{NP} \subseteq \#\cdot\mathbf{coNP}$.

In [Val79b] Vailant used a different type of reduction in his $\#\mathbf{P}$-completeness proofs, namely the so called counting reduction. A *counting reduction* (denoted by $\leq_{cnt}$) allows in contrast to a parsimonious reduction an additional application of a function from $\mathbf{FP}$ after the reduction function has been applied. It therefore is sometimes also called a *weakly parsimonious reduction*. The reason he used this instead of the parsimonious reduction is, that for many problems that are known to be $\#\mathbf{P}$-complete under counting reductions, it has not been possible to prove completeness under parsimonious reduction. The drawback of counting reductions however, as has been proved by Toda and Watanabe [TW92], is that the higher counting complexity classes are not closed thereunder. In Section 4.4 we will therefore introduce a new type of reduction, that on the one hand is wide enough to allow us to obtain completeness results and on the other hand is strict enough to leave the $\mathbf{\Pi^P_k}$-classes closed.

4.3 Quantified Constraint Satisfaction Problems

We will now define the quantified constraint satisfaction problem for a bounded number of quantifier alternations. The definition originates from [Hem04].

Definition 4.3.1 *Let S be a constraint language and let $i \geq 1$.*

- *For i odd, a $\mathrm{QCSP}_i(S)$-formula is a $\Sigma_i(S)$-formula and $\mathrm{QCSP}_i(S)$ is the problem to decide, whether a given $\mathrm{QCSP}_i(S)$-formula is true.*
- *For i even, a $\mathrm{QCSP}_i(S)$-formula is a $\Pi_i(S)$-formula and $\mathrm{QCSP}_i(S)$ is the problem to decide, whether a given $\mathrm{QCSP}_i(S)$-formula is false.*

The reason for defining different problems for i even and i odd is that we are dealing with constraints. According to Wrathall [Wra77] the general $\mathrm{Q}_\exists\mathrm{SAT}_i$ problems are complete for their respective class in the polynomial hierarchy if we are dealing with CNF in the case that i is odd and DNF in the case that i is even. But, a formula in DNF cannot be naturally modeled in a constraint context. Hence, in the even cases we are not looking, whether the formula is true, but whether it is false. Obviously the negation of a DNF is a CNF and thus can be expressed as a constraint language.

We will now show that the Galois connection between constraint languages and their closure properties is also applicable to quantified constraints with bounded alternations.

Proposition 4.3.2 *Let S_1 and S_2 be two constraint languages such that S_2 does not only contain the full relation and let $i \geq 1$. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then $\mathrm{QCSP}_i(S_1) \leq^{\log}_m \mathrm{QCSP}_i(S_2)$.*

Proof. Let φ be a $\mathrm{QCSP}_i(S_1)$-formula and let $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$. By Corollary 2.4.7 we know that we can express any relation from S_1 by an existentially quantified formula using relations from $S_2 \cup \{R_{\text{eq}}\}$. Thus, given φ we can similarly create a $\text{QCSP}_i(S_2)$-formula φ' by locally replacing all relations from S_1 by an existentially quantified $(S_2 \cup \{R_{\text{eq}}\})$-formula. The newly introduced existentially quantified variables can be moved to the right end of the quantifier block, because all other variables do not depend on them. Since $\text{QCSP}_i(S)$-formulae always end with a block of existential quantifiers, the resulting formula φ' is a $\text{QCSP}_i(S_2 \cup \{R_{\text{eq}}\})$-formula equivalent to φ.

In the next step, we need to remove the equality constraints in order to obtain a $\text{QCSP}_i(S_2)$-formula φ'' that is equivalent to φ. Therefore, we check, whether there are two variables x and y that are connected via an $=$-path, such that x is universally quantified after y is quantified. In this case φ' (and accordingly φ) is false. Hence, we need to build a false $\text{QCSP}_i(S_2)$-formula, which is only possible, if S_2 does not only contain the full relation. If no such variables x and y exist, all variables connected by an $=$-path are variables of which at most the first one in the quantifier sequence is universally quantified. Call this variable x. All other variables connected to x can be renamed to x and their corresponding existential quantifiers can be deleted. Finding variables connected by an $=$-path is simply searching for a path in an undirected graph, which is computable in logarithmic space by Reingold [Rei05]. □

We can now reprove a theorem first obtained by E. Hemaspaandra [Hem04]. Using the algebraic approach, this yields a much shorter proof.

Theorem 4.3.3 *Let S be a constraint language and let $i \geq 1$. If S is Schaefer, then* $\text{QCSP}_i(S)$ *is in* $\mathbf{P}$*; otherwise* $\text{QCSP}_i(S)$ *is* $\mathbf{\Sigma_i^P}$*-complete under logspace reductions.*

The polynomial result for the Schaefer cases already carries over from the general case (see Theorem 4.2.1). Likewise the case $i = 1$ is covered by Schaefer's Theorem, since deciding, whether a formula with only existentially quantified variables is true, is the same as testing satisfiability for a formula. Before we can prove the remainder of this theorem, we need some additional results.

Proposition 4.3.4 *Let $i \geq 1$; then,* $\text{QCSP}_i(R_{\text{nae}})$ *is* $\mathbf{\Sigma_i^P}$*-complete under logspace reductions.*

Proof. Since the general problem $\text{Q}_\exists\text{SAT}_i$ for arbitrary formulae is $\mathbf{\Sigma_i^P}$-complete, it follows that $\text{QCSP}_i(R_{\text{nae}})$ is trivially in $\mathbf{\Sigma_i^P}$. So, we only need to show hardness. From Proposition 2.4.4 we know that the relation $R_{1\text{in}3}$ is only closed under the identity function and thus $\text{Pol}(R_{1\text{in}3}) = \text{I}_2$. Taking a look at Figure 2.2 it is obvious that $\text{Pol}(S_{3\text{-SAT}}) \supseteq \text{I}_2$. Thus, according to Proposition 4.3.2 $\text{QCSP}_i(S_{3\text{-SAT}})$ is logspace reducible to $\text{QCSP}_i(R_{1\text{in}3})$; this yields $\mathbf{\Sigma_i^P}$-hardness for $\text{QCSP}_i(R_{1\text{in}3})$, since by Wrathall $\text{QCSP}_i(S_{3\text{-SAT}})$ is $\mathbf{\Sigma_i^P}$-complete under logspace reductions [Wra77]. We now show that $\text{QCSP}_i(R_{1\text{in}3}) \leq_m^{\log} \text{QCSP}_i(R_{\text{nae}})$, which concludes the proof. Let

$$\varphi \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R_{1\text{in}3}(x_{j_1}, x_{j_2}, x_{j_3})$$

be a $\mathrm{QCSP}_i(R_{\mathrm{1in3}})$-formula. We define the following relation as a conjunction of R_{nae}-relations.

$$\begin{aligned} R_{\mathrm{2in4}}(x_1,x_2,x_3,x_4) \quad = \quad & R_{\mathrm{nae}}(x_1,x_2,x_3) \ \wedge \ R_{\mathrm{nae}}(x_1,x_2,x_4) \\ & \wedge \ R_{\mathrm{nae}}(x_1,x_3,x_4) \ \wedge \ R_{\mathrm{nae}}(x_2,x_3,x_4). \end{aligned}$$

It is obvious, that $R_{\mathrm{2in4}}(x_1,x_2,x_3,x_4)$ is true if and only if exactly two of the four variables are set to true. We then create φ' by locally replacing every R_{1in3} relation by its R_{2in4} counterpart:

$$\varphi' \quad = \quad Q_1 t Q_1 X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R_{\mathrm{2in4}}(x_{j_1}, x_{j_2}, x_{j_3}, t).$$

We need to show that φ is true if and only if φ' is true. Let ψ and ψ' be the formula φ respectively φ' without their quantifiers; that is, $\varphi = Q_1X_1 \ldots \exists X_i \psi$ and $\varphi' = Q_1X_1 \ldots \exists X_i \psi'$. Assume that $Q_1 = \forall$ (the case $Q_1 = \exists$ then follows trivially). If $t = 1$, we have that $R_{\mathrm{2in4}}(x_1,x_2,x_3,1) = R_{\mathrm{1in3}}(x_1,x_2,x_3)$ and thus ψ and $\psi'[t/1]$ are equivalent. If $t = 0$, we have that $R_{\mathrm{2in4}}(x_1,x_2,x_3,0) = R_{\mathrm{1in3}}(\overline{x_1},\overline{x_2},\overline{x_3})$ and thus $\psi'[t/0]$ and $\mathrm{Neg}(\psi)$ are equivalent, where $\mathrm{Neg}(\psi)$ is the formula obtained from ψ by negating all literals. Obviously I is a satisfying assignment for ψ, if and only if $\overline{I}$ is a satisfying assignment for $\mathrm{Neg}(\psi)$; therefore, φ is true if and only if φ' is true. Since φ' originates from φ by local replacements only, the reduction is surely computable in logarithmic space. □

Lemma 4.3.5 *Let S be a constraint language and let $i \geq 2$. If S is not Schaefer, then $\mathrm{QCSP}_i(S)$ is $\mathbf{\Sigma_i^P}$-complete under logspace reductions.*

Proof. Let S be a constraint language such that S is not Schaefer. According to Proposition 2.4.3 we have $\mathrm{Pol}(S) \subseteq \mathrm{N}$. Since the general problem $\mathrm{Q_\exists SAT}_i$ for arbitrary formulae is already $\mathbf{\Sigma_i^P}$-complete it remains to show hardness for $\mathrm{Pol}(S) = \mathrm{N}$. Let the relation R be defined as

$$R \quad = \quad \{t_1,t_2,x_1,x_2,x_3 \mid t_1 = t_2 \text{ or } R_{\mathrm{nae}}(x_1,x_2,x_3)\}.$$

It is obvious that R is closed under the constant 0 and under the constant 1, since $(0,0,0,0,0),(1,1,1,1,1) \in R$. It is also closed under negation: Let $(v_1,v_2,v_3,v_4,v_5) \in R$. If $v_1 = v_2$, then trivially $(\overline{v_1},\overline{v_2},\overline{v_3},\overline{v_4},\overline{v_5}) \in R$. Therefore, let without loss of generality $v_1 = 0$ and $v_2 = 1$. By definition $R_{\mathrm{nae}}(v_3,v_4,v_5)$ holds and then also $R_{\mathrm{nae}}(\overline{v_3},\overline{v_4},\overline{v_5})$ holds. Consequently $(\overline{v_1},\overline{v_2},\overline{v_3},\overline{v_4},\overline{v_5}) \in R$.

Since $\{\mathit{not}, \{0\}, \{1\}\}$ is a base for N, we know that $\mathrm{Pol}(\{R\}) \supseteq \mathrm{N}$. Now we prove that $\mathrm{QCSP}_i(R)$ is $\mathbf{\Sigma_i^P}$-hard by a logspace reduction from $\mathrm{QCSP}_i(R_{\mathrm{nae}})$, which is $\mathbf{\Sigma_i^P}$-complete by Proposition 4.3.4. Let

$$\varphi \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R_{\mathrm{nae}}(x_{j_1}, x_{j_2}, x_{j_3})$$

be an instance of $\mathrm{QCSP}_i(R_{\mathrm{nae}})$. Then construct an instance of $\mathrm{QCSP}_i(R)$ as follows.

$$\varphi' \quad = \quad Q_1 X_1 \dots \forall X_{i-1} \forall t_1 \forall t_2 \exists X_i \bigwedge_{j=1}^{s} R(t_1, t_2, x_{j_1}, x_{j_2}, x_{j_3}).$$

Since φ' must be true for all combinations of truth assignments to t_1 and t_2, the constraint $R_{\mathrm{nae}}(x_{j_1}, x_{j_2}, x_{j_3})$ has to hold in every relation R. Hence, it is clear that φ is true if and only if φ' is true and the reduction holds. Since we are always just replacing one relation by another one locally, the reduction is clearly computable in logarithmic space. $\square$

4.4 Complementive Reductions

The remainder of this chapter is dedicated to our main theorem, namely the complete classification of the counting problem of quantified constraints for a given constraint language S. The challenge we are confronted with is to decide, what reduction we should use for our results. Generally the parsimonious reduction is the reduction of choice in the context of counting problems, since all higher complexity counting classes $\#\cdot\mathbf{\Sigma_k^P}$ and $\#\cdot\mathbf{\Pi_k^P}$ are closed under this reduction. However, there are some problems like $\mathrm{QCSP}_i(R_{\mathrm{nae}})$ for which this reduction does not work, since R_{nae} is complementive and thus always leads to an even number of solutions. In order to resolve this problem we will introduce *complementive reductions*, which are not as strict as parsimonious reductions, but still supply closure of the counting classes.

We call a finite alphabet *even*, if it contains an even number of elements. A permutation π on an even alphabet is called *bipartite*, if there exists a partition of Γ into two disjoint sets Γ_0 and Γ_1 such that the following holds:

- $\Gamma = \Gamma_0 \cup \Gamma_1$
- $\Gamma_0 \cap \Gamma_1 = \emptyset$
- $|\Gamma_0| = |\Gamma_1|$
- for all $x \in \Gamma_i$ we have $\pi(x) \in \Gamma_{i-1}$ for each $i = 0, 1$.

For each string $x_1 \dots x_k \in \Gamma^*$ we homomorphically enlarge every permutation π on Γ to the strings in Γ^* by means of the identity $\pi(x_1 \dots x_k) = \pi(x_1) \dots \pi(x_k)$.

A set of strings $E \subseteq \Gamma^*$ over an even alphabet Γ is called *complementive*, if there exists a bipartite permutation π_E on Γ such that $x \in E$ holds if and only if $\pi_E(x) \in E$. If we know that a set of strings E is complementive, we always assume that we are effectively given the permutation π_E. Given two alphabets Σ and Γ with Γ being even, we call a relation B between strings from Σ and Γ *complementive*, if for each string $y \in \Sigma^*$ the set $B(y)$ is complementive with respect to the same bipartite permutation π_B.

Definition 4.4.1 *Let Σ and Γ be two alphabets, Γ being even, and let $\#A$ and $\#B$ be two counting problems determined by the relations A and B between the strings from Σ and Γ, where B is complementive.*

- *We say that the counting problem $\#A$ reduces to the counting problem $\#B$ via a* strong complementive reduction*, if there exist two polynomial-time computable functions f and g such that for every string $x \in \Sigma^*$ the following holds:*
  - $B(g(x)) \subseteq B(f(x))$
  - $2 \cdot |A(x)| = |B(f(x))| - |B(g(x))|$
- *Let $\#A$ and $\#B$ be two counting problems. A* complementive reduction *$\#A \leq_{compl} \#B$ from $\#A$ to $\#B$ is a sequence of parsimonious and strong complementive reductions; that is, $\#A \leq_{compl} \#B$ if and only if $\#A \leq_{x_1} \#X_1 \leq_{x_2} \#X_2 \leq_{x_3} \ldots \leq_{x_n} \#B$, where $\leq_{x_i}$ for $1 \leq i \leq n$ is either a parsimonious or a strong complementive reduction.*

We will now show that this reduction, which is obviously a special case of counting reductions, is reasonable for our aims. Theorem 4.4.2 first proves that all complexity classes $\#\cdot\mathbf{\Pi_k^P}$ are closed under complementive reductions. Proposition 4.4.3 will then demonstrate that the same statement unfortunately does not hold for $\#\cdot\mathbf{\Sigma_k^P}$ classes. However, it nevertheless makes sense to use complementive reductions for hardness results in the hierarchy of the classes $\#\cdot\mathbf{\Sigma_k^P}$. Since $\#\cdot\mathbf{\Sigma_k^P} \subseteq \#\cdot\mathbf{\Pi_k^P} \subseteq \#\cdot\mathbf{\Sigma_{k+1}^P}$ holds and the $\#\cdot\mathbf{\Pi_k^P}$ classes are closed, a problem complete for $\#\cdot\mathbf{\Sigma_{k+1}^P}$ cannot be in $\#\cdot\mathbf{\Sigma_k^P}$, unless $\#\cdot\mathbf{\Sigma_{k+1}^P} = \#\cdot\mathbf{\Pi_k^P}$. By Vollmer [Vol94] the latter would imply $\mathbf{UP}^{\mathbf{\Sigma_k^P}} = \mathbf{NP}^{\mathbf{\Sigma_k^P}}$, where **UP** is the class of languages that are acceptable by an NTM such that there is always at most one accepting computation path. Thus, it is reasonable to assume that a problem complete for $\#\cdot\mathbf{\Sigma_{k+1}^P}$ is not in $\#\cdot\mathbf{\Sigma_k^P}$.

Theorem 4.4.2 *Let $k \geq 1$; then, $\#\mathbf{P}$ and all higher complexity classes $\#\cdot\mathbf{\Pi_k^P}$ are closed under complementive reductions.*

Proof. Note that for $k = 0$ we have $\#\mathbf{P} = \#\cdot\mathbf{\Pi_0^P}$; thus we take $k \geq 0$ arbitrary, but fixed. First, we will prove that $\#\cdot\mathbf{\Pi_k^P}$ is closed under *strong* complementive reductions. Therefore, assume two counting problems $\#A$ and $\#B$, such that $\#B$ is in $\#\cdot\mathbf{\Pi_k^P}$, $\#A$ is reducible to $\#B$ with a strong complementive reduction, and B is complementive. We will show that then $\#A$ is also in $\#\cdot\mathbf{\Pi_k^P}$. By definition we have two polynomial-time computable functions f and g, such that $B(g(x)) \subseteq B(f(x))$ and $2 \cdot |A(x)| = |B(f(x))| - |B(g(x))|$. We will now construct a relation A' as follows. A tuple (x, y') belongs to A', if y' is of the form

$$y' \quad = \quad f(x)\$g(x)\$y$$

with $(f(x), y) \in B$, $(g(x), y) \notin B$, and $\mathrm{last}(y) \in \Gamma_0$, where \$ is a delimiter symbol that is neither contained in Σ nor in Γ, last(y) returns the last element of the string y, and Γ_0 is one of the two partition sets of Γ, both of which are defined by the bipartite permutation π_B. The following simple algorithm decides, whether a given pair (x, y') is in A' or not.

1. Divide the string y' into its three parts $f(x)$, $g(x)$, and y.
2. Check whether $\text{last}(y) \in \Gamma_0$; if not, reject.
3. Check whether $(f(x), y) \in B$; if not, reject.
4. Check whether $(g(x), y) \notin B$; if not, reject.
5. Accept.

Obviously steps 1 and 2 of the algorithm can be computed in polynomial time. By definition of the counting classes, the test whether $(f(x), y)$ belongs to B in step 3 is in $\mathbf{\Pi_k^P}$ and thus also in $\mathbf{P}^{\mathbf{\Sigma_k^P}}$. Finally, in step 4 we need to test, whether $(g(x), y)$ does not belong to B, that is, the complement of step 3, which is in $\mathbf{\Sigma_k^P}$ and thus also in $\mathbf{P}^{\mathbf{\Sigma_k^P}}$. Consequently A' is in $\mathbf{P}^{\mathbf{\Sigma_k^P}}$ and hence $\#A'$ is in $\#{\cdot}\mathbf{P}^{\mathbf{\Sigma_k^P}}$, which is the same as $\#{\cdot}\mathbf{\Pi_k^P}$ by Toda [Tod91]. By construction of A' we know that the number of elements in $A'(x)$ is made up of the number of elements in $B(f(x))$ minus those that are in $B(g(x))$ and the whole number is divided by two, since only the strings ending with an element of Γ_0 are taken (which are exactly half of the strings, because B is complementive). Therefore, $2 \cdot |A'(x)| = |B(f(x))| - |B(g(x))|$ and hence $|A'(x)| = |A(x)|$. It follows that the counting problem $\#A$ is in $\#{\cdot}\mathbf{\Pi_k^P}$, because it is as difficult to determine the solutions of A as it is for A'.

The closure of $\#{\cdot}\mathbf{\Pi_k^P}$ under complementive reductions now follows inductively on the number of parsimonious and strong complementive reductions, since for both of them $\#{\cdot}\mathbf{\Pi_k^P}$ is closed. □

Proposition 4.4.3 *For every $k \in \mathbb{N}$, each higher complexity class $\#{\cdot}\mathbf{\Sigma_k^P}$ is not closed under complementive reductions, unless $\#{\cdot}\mathbf{\Sigma_k^P} = \#{\cdot}\mathbf{\Pi_k^P}$.*

Proof. Let $k \in \mathbb{N}$. We will reduce a $\#{\cdot}\mathbf{\Pi_k^P}$-complete problem to a problem in $\#{\cdot}\mathbf{\Sigma_k^P}$ via a complementive reduction. Note that the problem, given a quantified formula

$$\varphi(y_1, \dots, y_n) \quad = \quad \forall X_1 \exists X_2 \dots Q_k X_k \psi(x_1, \dots, x_m, y_1, \dots, y_n),$$

where ψ is a quantifier-free formula and $x_1, \dots, x_m \in \bigcup_{i=1}^k X_i$, is $\#{\cdot}\mathbf{\Pi_k^P}$-complete according to Durand et al. [DHK00]. The problem stays $\#{\cdot}\mathbf{\Pi_k^P}$-complete if for k odd ψ is in DNF and for k even ψ is in CNF. Similarly the problem given a quantified formula starting with an existential quantifier and having $k-1$ quantifier alternations is $\#{\cdot}\mathbf{\Sigma_k^P}$-complete. We will do a case distinction on the parity of k: First, assume that k is odd and we are given φ_o as the quantified formula φ from above in such a way that the quantifier-free formula ψ_o is in DNF. Construct two formulae ξ_o and ϑ_o as follows:

$$\begin{array}{rcl} \xi_o(y_0, \dots, y_n) & = & y_0 \vee y_1 \vee \dots \vee y_n \vee \overline{y_0} \vee \overline{y_1} \vee \dots \vee \overline{y_n} \\ \vartheta_o(y_0, \dots, y_n) & = & (y_0 \vee \neg\varphi_o(y_1, \dots, y_n)) \wedge (\overline{y_0} \vee \neg\varphi_o(\overline{y_1}, \dots, \overline{y_n})). \end{array}$$

Obviously the negation of φ_o can be propagated to appear only at the literal level. The resulting formula then starts with an existential quantifier and is in CNF. By the distributive law we can thus transform ϑ_o into a quantified CNF.

In the case that k is even, we are given φ_e as the quantified formula φ from above in such a way that the quantifier-free formula ψ_e is in CNF. Let $\xi_e = \xi_o$ and construct

$$\vartheta_e(y_0, \dots, y_n) \quad = \quad (y_0 \wedge \neg\varphi_e(y_1, \dots, y_n)) \vee (\overline{y_0} \wedge \neg\varphi_e(\overline{y_1}, \dots, \overline{y_n})).$$

Propagation of the negation and application of the distributive law yields a quantified DNF ϑ_e, whose first quantifier is a universal one. All those transformation can be computed in polynomial time and linear space. Also note that all formulae ξ_o, ξ_e, ϑ_o, and ϑ_e are complementive. Since the ξ-formulae represent the full relation, obviously $\#\mathrm{sat}(\xi_p) = 2 \cdot 2^n$ and

$$\mathrm{sat}(\vartheta_p) \quad \subseteq \quad \mathrm{sat}(\xi_p)$$

for $p \in \{o, e\}$. Evidently $\#\mathrm{sat}(\psi_p) = 2 \cdot \#\mathrm{sat}(\neg\varphi_p) = 2 \cdot (2^n - \#\mathrm{sat}(\varphi_p)) = 2 \cdot 2^n - 2 \cdot \#\mathrm{sat}(\varphi_p)$ for $p \in \{o, e\}$. Combined we have

$$2 \cdot \#\mathrm{sat}(\varphi_p) \quad = \quad \#\mathrm{sat}(\xi_p) - \#\mathrm{sat}(\vartheta_p)$$

for $p \in \{o, e\}$. This constitutes for both the odd and the even cases the complementive reduction we were looking for. □

4.5 Counting Problems for Quantified Constraints

In this section we want to classify the counting problem associated with the quantified constraint satisfaction problem. The previous results will serve as a basis to start from. First, we will have to clarify what counting for quantified constraints means. Since a quantified Boolean formula is just true or false, there is nothing to count. Thus, we consider formulae with some free variables apart from the quantified variables. Analogously to $\mathrm{Q}_\exists\mathrm{SAT}_i$ and $\mathrm{Q}_\forall\mathrm{SAT}_i$, we define $\#\mathrm{Q}_\exists\mathrm{SAT}_i$ and $\#\mathrm{Q}_\forall\mathrm{SAT}_i$ as the problems to count the number of satisfying assignments for a given quantified formula with free variables and $i - 1$ quantifier alternations starting with an existential respectively universal quantifier. These problems are complete for the classes $\#\cdot\mathbf{\Sigma_i^P}$ respectively $\#\cdot\mathbf{\Pi_i^P}$ under parsimonious reductions and remain complete when restricted to CNF (for $\#\mathrm{Q}_\exists\mathrm{SAT}_i$ with odd i and $\#\mathrm{Q}_\forall\mathrm{SAT}_i$ with even i) or DNF (for $\#\mathrm{Q}_\exists\mathrm{SAT}_i$ with even i and $\#\mathrm{Q}_\forall\mathrm{SAT}_i$ with odd i) [DHK00]. Similar to Definition 4.3.1 we now define the central problem of this section.

Definition 4.5.1 *Let S be a constraint language and let $i \geq 1$.*

- *For i odd, we define a $\#\mathrm{QCSP}_i(S)$-formula as a formula $\varphi(Y) = \exists X_1 \forall X_2 \dots \exists X_i$ $\psi(Y, X_1, \dots, X_i)$, where ψ is a quantifier-free formula, and $\#\mathrm{QCSP}_i(S)$ as the problem to determine $\#\mathrm{sat}(\varphi)$ for a given $\#\mathrm{QCSP}_i(S)$-formula φ.*

- *For i even, we define a* $\#\mathrm{QCSP}_i(S)$*-formula as a formula* $\varphi(Y) = \forall X_1 \exists X_2 \ldots \exists X_i\ \psi(Y, X_1, \ldots, X_i)$*, where* ψ *is a quantifier-free formula, and* $\#\mathrm{QCSP}_i(S)$ *as the problem to determine* $\#\mathrm{unsat}(\varphi)$ *for a given* $\#\mathrm{QCSP}_i(S)$*-formula* φ*.*

The reason for counting the satisfying assignments in the odd cases and the unsatisfying assignments in the even cases is the same as the reason for deciding whether a $\mathrm{QCSP}_i(S)$-formula is true respectively false in Definition 4.3.1; namely, that we are dealing with constraints only. A special case of the quantified constraints are the conjunctive queries. In this case $i = 1$ and therefore we do not have any universally quantified variables, that is, a conjunctive query over a constraint language S is a $\mathrm{QCSP}_1(S)$-formula.

Definition 4.5.2 *Let S be a constraint language. We define a* $\#\text{SAT-COQ}(S)$*-formula as a* $\#\mathrm{QCSP}_1(S)$*-formula and* $\#\text{SAT-COQ}(S)$ *as the problem to determine* $\#\mathrm{sat}(\varphi)$ *for a given* $\#\text{SAT-COQ}(S)$*-formula* φ*, that is,* $\#\mathrm{QCSP}_1(S)$*.*

The results of this section are summarized in the following main theorem and an incidental corollary.

Theorem 4.5.3 *Let S be a constraint language and let $i \geq 1$.*

- *If S is affine, then* $\#\mathrm{QCSP}_i(S)$ *is in* $\mathbf{FP}$*.*
- *Otherwise, if S is Schaefer, then* $\#\mathrm{QCSP}_i(S)$ *is* $\#\mathbf{P}$*-complete under counting reductions.*
- *If S is not Schaefer, then* $\#\mathrm{QCSP}_i(S)$ *is* $\#\cdot\mathbf{\Sigma_i^P}$*-complete under complementive reductions.*

As an easy consequence we can state the following result about conjunctive queries. The corollary follows immediately from Theorem 4.5.3 by letting $i = 1$.

Corollary 4.5.4 *Let S be a constraint language.*

- *If S is affine, then* $\#\text{SAT-COQ}(S)$ *is in* $\mathbf{FP}$*.*
- *Otherwise, if S is Schaefer, then* $\#\text{SAT-COQ}(S)$ *is* $\#\mathbf{P}$*-complete under counting reductions.*
- *If S is not Schaefer, then* $\#\text{SAT-COQ}(S)$ *is* $\#\cdot\mathbf{NP}$*-complete under complementive reductions.*

A graphical representation of Theorem 4.5.3 can be seen in Figure 4.1. The proof of this theorem follows from the lemmas in the remainder of this chapter. First, we state an easy consequence of Proposition 4.3.2: We can show that the Galois connection also applies to the counting version of quantified constraints, which again enables us to utilize short and elegant proofs.

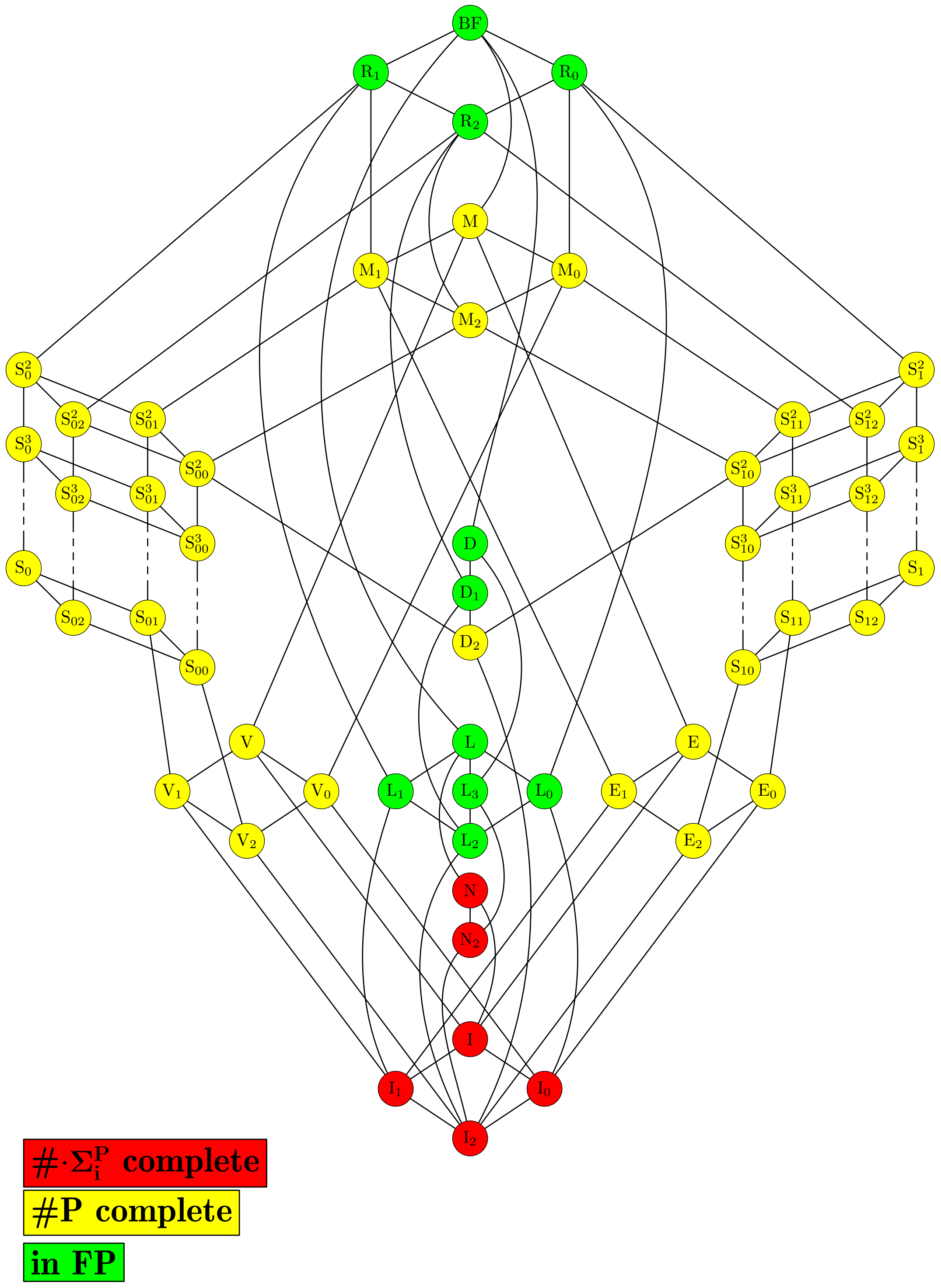

Figure 4.1: The complexity of $\#\mathrm{QCSP}_i(S)$ for $i \geq 1$

Proposition 4.5.5 *Let S_1 and S_2 be two constraint languages such that S_2 does not only contain the full relation, and let $i \geq 1$. If $\mathrm{Pol}(S_2) \subseteq \mathrm{Pol}(S_1)$, then $\#\mathrm{QCSP}_i(S_1) \leq_! \#\mathrm{QCSP}_i(S_2)$.*

Proof. Since the last quantifier of any $\#\mathrm{QCSP}_i(S)$-formula is always an existential quantifier, by Theorem 2.4.6 we can just locally replace every relation in S_1 by conjunctive queries over S_2. These replacements are preserving the number of solutions and thus the reduction is parsimonious. □

We will start the proof of our main theorem with the affine case.

Lemma 4.5.6 *Let S be a constraint language and let $i \geq 1$. If S is affine, then $\#\mathrm{QCSP}_i(S)$ is in* **FP**.

Proof. Let S be affine. As mentioned in Schaefer's work [Sch78] and similarly to the problem $\#\mathrm{CSP}(S)$ in [CH96], a set of affine constraints can be considered as a system of linear equations over $\{0, 1\}$. However, before we can use the Gaussian elimination algorithm for $\#\mathrm{QCSP}_i(S)$, we first have to eliminate the quantifiers. Let

$$\varphi(Y) \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} C_j$$

be a $\#\mathrm{QCSP}_i(S)$-formula, such that each C_j is an $\oplus$-formula over the variables $Y \cup X_1 \cup \cdots \cup X_i$. Then each $C_j = x_{j_1} \oplus \ldots \oplus x_{j_{k_j}}$ corresponds to a linear equation $e_j \colon x_{j_1} + \cdots + x_{j_{k_j}} = 1$. Let $E = \{e_1, \ldots, e_s\}$ be the equation system defined by $\bigwedge_{j=1}^{s} C_j$. The algorithm in Figure 4.2 then either returns an equivalent system of linear equations or rejects the input if φ is not satisfiable due to the quantified variables.

We will now prove the correctness of this algorithm. After the execution of step 1 we are given a system of equations E that is equivalent to the formula φ. We will show that every step of the algorithm preserves the set of solutions of E, but removes all quantified variables; that is, after the execution of the for-loop in step 2 all equations with index $\leq j$ have either been deleted or contain only unquantified variables. For this we examine each step in the for-loop and ensure that the set of solutions is preserved and the current equation e_j does not contain quantified variables at the end of the for-loop. Since $x \oplus x$ is equivalent to 0 for any x, we can just delete two occurrences of a variable in the same equation in step 3. Trivially, if a equation $0 = 1$ occurs, the system of equations has no solution and thus the input is rejected in step 5. If in step 7 no variable is quantified in the current equation, that equation is fine and we carry on with the next one. The key point of the algorithm appears in the instance that at least one variable is quantified in an equation e_j. In that case we look for the variable x, which is quantified last. Then there are two possibilities to consider. First, if x is universally quantified, then E has no solution. This is because in e_j all variables are assigned a value before x, resulting in a formula $\forall x(x = 1)$ or $\forall x(x = 0)$, both of which are obviously false. If on the other hand x is existentially quantified, the equation e_j can be satisfied, but the variable x is

INPUT: quantified formula $\varphi(Y) = Q_1X_1 \dots \forall X_{i-1}\exists X_i \bigwedge_{j=1}^{s} C_j$

```
1: Let $E = \{e_1, \dots, e_s\}$ be the equation system induced by $\varphi$.
2: for $j = 1$ to $s$ do
3:    Replace in $e_j$ every double occurrence of a variable by 0.
4:    if $e_j$ is contradictory (i. e., $0 = 1$) then
5:       reject
6:    end if
7:    if at least one variable in $e_j$ is quantified then
8:       Let $x$ be a quantified variable, such that no other variable is quantified after $x$ is
         quantified.
9:       if $x$ is universally quantified then
10:         reject
11:      else
12:         Solve $e_j$ for $x$.
13:         Substitute the solution for $x$ in every other equation containing $x$.
14:         Delete $e_j$ in $E$.
15:      end if
16:   end if
17: end for
```

Figure 4.2: Algorithm for obtaining a linear equation system from a quantified formula

fixed afterwards. Therefore, the value of x, which is the equation e_j solved for x, is then substituted for any other occurrence of x (obviously this can only happen in equations e_k with $k > j$ since in the other ones are no quantified variables). The equation e_j can then be deleted in step 14, since by Gaussian elimination the system of equations stays consistent, when one variable together with its equation is eliminated by transferring the information of that equation to all other equations containing that variable. In the special case that there is no other occurrence of x, the equation can just be deleted anyway, because it does not matter to which value x is quantified (this is also the case for the last equation). Hence, if the algorithm does not reject the input, we have an equivalent system of linear equations, such that no occurring variable is quantified. Note that since in step 3 free variables might be deleted, it is important to keep the set Y of free variables.

For the time complexity note that step 1 needs linear time. The for-loop 2 through 17 is executed once for every C in φ. Step 3 is also computable in polynomial time, as is searching for the last quantified variable in an equation (step 8). Solving a linear equation for a variable is just a simple transformation, which is possible in polynomial time. Finally, the substitutions of a variable by a formula in step 13 is computable in polynomial time, since there are only linearly many equations to look at and the size of an equation increases at most by the number of variables, because at the beginning of the for-loop all double variables are deleted. Hence, the total computation time is polynomial in the input size.

Thus, after the execution of the algorithm we have a system of equations that is equivalent to the original formula φ in the sense that the number of solutions respectively satisfying assignments is the same. Now we can simply determine that number with the help of the Gaussian elimination method, which is also computable in polynomial time. Therefore, we have a polynomial-time computable function and $\#\mathrm{QCSP}_i(S)$ is in **FP**. □

The other Schaefer cases follow almost immediately from known results.

Lemma 4.5.7 *Let S be a constraint language and let $i \geq 1$. If S is not affine, but Horn, anti-Horn, or bijunctive, then $\#\mathrm{QCSP}_i(S)$ is $\#\mathbf{P}$-complete under counting reductions.*

Proof. In the cases that S is Horn, anti-Horn, or bijunctive, we know from Theorem 4.3.3 that $\mathrm{QCSP}_i(S)$ is in $\mathbf{P}$. Therefore, $\#\mathrm{QCSP}_i(S)$ is in $\#\mathbf{P}$. Creignou and Hermann proved that if S is not affine, $\#\mathrm{CSP}(S)$ in these three cases is already $\#\mathbf{P}$-complete. By the trivial counting reduction from $\#\mathrm{CSP}(S)$ to $\#\mathrm{QCSP}_i(S)$ we have that $\#\mathrm{QCSP}_i(S)$ is also $\#\mathbf{P}$-complete. □

Before we can prove $\#\cdot\mathbf{\Sigma_i^P}$-completeness for all non-Schaefer cases, we need an additional proposition.

Proposition 4.5.8 *Let $i \geq 1$. Then $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$ is $\#\cdot\mathbf{\Sigma_i^P}$-complete under complementive reductions.*

Proof. Since Durand, Hermann, and Kolaitis showed in [DHK00] that the counting problem for arbitrary formulae starting with an existential quantifier and having $i-1$ quantifier alternations is $\#\cdot\mathbf{\Sigma_i^P}$-complete under parsimonious reduction, membership in $\#\cdot\mathbf{\Sigma_i^P}$ for $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$ with i odd follows directly. For i even note that $\#\mathrm{Q}_\forall\mathrm{SAT}_i$ is $\#\cdot\mathbf{\Pi_i^P}$-complete under parsimonious reductions by [DHK00]. Since we are looking for unsatisfiability in this case, $\#\mathrm{QCSP}_i(S)$ is the complement problem of $\#\mathrm{Q}_\forall\mathrm{SAT}_i$ and thus $\#\cdot\mathbf{\Sigma_i^P}$-complete under parsimonious reductions for arbitrary S. Hence, $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$ is also in $\#\cdot\mathbf{\Sigma_i^P}$ for i even.

As in the proof of Proposition 4.3.4, we first prove that the problem $\#\mathrm{QCSP}_i(R_{1\mathrm{in}3})$ is $\#\cdot\mathbf{\Sigma_i^P}$-complete under parsimonious reductions. Due to [DHK00] we already know that $\#\mathrm{QCSP}_i(S_{3\text{-SAT}})$ is $\#\cdot\mathbf{\Sigma_i^P}$-complete under parsimonious reductions. Since $\mathrm{Pol}(R_{1\mathrm{in}3}) = \mathrm{I}_2$, we have that $\#\mathrm{QCSP}_i(S_{3\text{-SAT}}) \leq_! \#\mathrm{QCSP}_i(R_{1\mathrm{in}3})$ by Proposition 4.5.5. We now state a strong complementive reduction from $\#\mathrm{QCSP}_i(R_{1\mathrm{in}3})$ to $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$. Since complementive reductions are the closure of parsimonious and strong complementive reductions, the proposition follows immediately. Thus, let

$$\varphi(Y) \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R_{1\mathrm{in}3}(x_{j_1}, x_{j_2}, x_{j_3})$$

be a $\#\mathrm{QCSP}_i(R_{1\mathrm{in}3})$-formula with $x_{j_1}, x_{j_2}, x_{j_3} \in Y \cup X_1 \cup \cdots \cup X_i$, where $Y = \{y_1, \ldots, y_n\}$ is the set of free variables. Let u and v be new free variables and define

$$\varphi_1(Y, u, v) \quad = \quad \varphi(Y) \wedge R_{1\mathrm{in}3}(u, u, v).$$

Then obviously u is fixed to 0 and v is fixed to 1. Therefore, $\#\mathrm{sat}(\varphi_1) = \#\mathrm{sat}(\varphi)$ and $\#\mathrm{unsat}(\varphi_1) = 2^{n+2} - \#\mathrm{sat}(\varphi_1)$, since there are now $n+2$ free variables. Let $R_{2\mathrm{in}4}$ be the relation defined as a conjunction of R_{nae}-relations as in the proof of Proposition 4.3.4; that is, $R_{2\mathrm{in}4}$ is true if and only if exactly two of the four variables are set to 1. Now we can define the formulae necessary for our reduction. Let

$$\varphi_2(Y,v,u) \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \exists t \bigwedge_{j=1}^{s} R_{2\mathrm{in}4}(x_{j_1}, x_{j_2}, x_{j_3}, t) \wedge R_{2\mathrm{in}4}(u,u,v,t),$$

$$\varphi_3(Y,v,u) \quad = \quad \bigwedge_{j=1}^{n} R_{\mathrm{nae}}(u,v,y_j) \wedge R_{\mathrm{nae}}(u,u,v),$$

and

$$\varphi_4(Y,v,u) \quad = \quad R_{\mathrm{nae}}(u,u,u).$$

The number of satisfying assignments of φ_2 is exactly twice the number of satisfying assignments of φ_1. This is the case, since for every satisfying assignment $(\alpha_1, \alpha_2, \alpha_3)$ of an $R_{1\mathrm{in}3}$-relation, the two assignments $(\alpha_1, \alpha_2, \alpha_3, 1)$ and $(\overline{\alpha_1}, \overline{\alpha_2}, \overline{\alpha_3}, 0)$ satisfy the corresponding $R_{2\mathrm{in}4}$-relation. Thus, $\#\mathrm{unsat}(\varphi_2) = 2^{n+2} - 2 \cdot \#\mathrm{sat}(\varphi)$. For φ_3 we know due to the last conjunct, that u and v must take different values. Then follows that each y_j can take any value and therefore $\#\mathrm{sat}(\varphi_3) = \#\mathrm{unsat}(\varphi_3) = 2^{n+1}$. The formula φ_4 is obviously unsatisfiable and thus $\#\mathrm{sat}(\varphi_4) = 0$ and $\#\mathrm{unsat}(\varphi_4) = 2^{n+2}$. Since we are distinguishing between the number of satisfying solutions for a formula with odd i and unsatisfying solutions for i even, we have to define two reductions:

- For i odd, let $f(\varphi) = \varphi_2$ and $g(\varphi) = \varphi_4$. Surely $\mathrm{sat}(\varphi_4) \subseteq \mathrm{sat}(\varphi_2)$ and $2 \cdot \#\mathrm{sat}(\varphi) = \#\mathrm{sat}(\varphi_2) - \#\mathrm{sat}(\varphi_4)$.
- For i even, we take $f(\varphi) = \varphi_2$ and $g(\varphi) = \varphi_3$. Then $\mathrm{unsat}(\varphi_3) \subseteq \mathrm{unsat}(\varphi_2)$ and $2 \cdot \#\mathrm{unsat}(\varphi) = \#\mathrm{unsat}(\varphi_2) - \#\mathrm{unsat}(\varphi_3)$.

Obviously all constructed formulae are computable in polynomial time. Hence, we have a complementive reduction in both cases. □

Lemma 4.5.9 *Let S be a constraint language. If S is not Schaefer, then $\#\mathrm{QCSP}_i(S)$ is $\#\cdot\mathbf{\Sigma_i^P}$-complete under complementive reductions.*

Proof. Let S be a constraint language such that S is not Schaefer. According to Proposition 2.4.3 we know that $\mathrm{Pol}(S) \subseteq \mathrm{N}$. Since the general problem $\#\mathrm{QCSP}_i(T)$ for arbitrary constraint languages T is already $\#\cdot\mathbf{\Sigma_i^P}$-complete it remains to show hardness for a constraint language S' with $\mathrm{Pol}(S') = \mathrm{N}$ under complementive reductions. Let

$$\varphi(Y) \quad = \quad Q_1X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R_{\mathrm{nae}}(x_{j_1}, x_{j_2}, x_{j_3})$$

be a $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$-formula with $x_{j_1}, x_{j_2}, x_{j_3} \in Y \cup X_1 \cup \cdots \cup X_i$, where $Y = \{y_1, \ldots, y_n\}$ is the set of free variables. First, assume that $i \geq 2$; that is, we have universal and existential quantifiers in our formula. Let

$$R \quad = \quad \{t_1, t_2, x_1, x_2, x_3 \mid t_1 = t_2 \text{ or } R_{\mathrm{nae}}(x_1, x_2, x_3)\}$$

be the relation originating from the proof of Lemma 4.3.5. Let further $S' = \{R\}$. We already know that $\mathrm{Pol}(S') \supseteq \mathrm{N}$. We will now state a parsimonious reduction from $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$ to $\#\mathrm{QCSP}_i(S')$. Let

$$\varphi'(Y) \quad = \quad Q_1 X_1 \ldots \forall X_{i-1} \forall t_1 \forall t_2 \exists X_i \bigwedge_{j=1}^{s} R(t_1, t_2, x_{j_1}, x_{j_2}, x_{j_3}),$$

where $x_{j_1}, x_{j_2}, x_{j_3} \in Y \cup X_1 \cup \cdots \cup X_i$. Corresponding to the proof of Lemma 4.3.5, it is obvious that $\#\mathrm{sat}(\varphi) = \#\mathrm{sat}(\varphi')$; thus, yielding a parsimonious reduction. Note, that for this reduction to work, we require universally quantified variables t_1 and t_2. Thus, for the case $i = 1$ we need a different reduction. Let therefore

$$\begin{aligned} R'(u, v, x, y, z) &= (\overline{u} \wedge \overline{v} \wedge \overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (u \wedge v \wedge x \wedge y \wedge z) \quad \text{and} \\ R''(u, v, x, y, z) &= R'(u, v, x, y, z) \\ &\quad \vee (u \wedge \overline{v} \wedge R_{\mathrm{nae}}(x, y, z)) \ \vee \ (\overline{u} \wedge v \wedge R_{\mathrm{nae}}(x, y, z)) \end{aligned}$$

be two relations and let $S' = \{R', R''\}$. It is easy to see that both R' and R'' are 0-valid, 1-valid, and complementive. Hence, S' is also 0-valid, 1-valid, and complementive and therefore $\mathrm{Pol}(S') \supseteq \mathrm{N}$. Now construct the formulae

$$\varphi_1'(Y, u, v) \quad = \quad Q_1 X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R'(u, v, x_{j_1}, x_{j_2}, x_{j_3})$$

and

$$\varphi_1''(Y, u, v) \quad = \quad Q_1 X_1 \ldots \forall X_{i-1} \exists X_i \bigwedge_{j=1}^{s} R''(u, v, x_{j_1}, x_{j_2}, x_{j_3}),$$

where u and v are new free variables. It is obvious, that $\#\mathrm{sat}(\varphi_1') = 2$. By construction of R'' it is easy to see, that for φ_1'' the number of satisfying assignments is twice that of φ plus 2 and that $\mathrm{sat}(\varphi_1') \subseteq \mathrm{sat}(\varphi_1'')$. Thus, we have a strong complementive reduction with $2 \cdot \#\mathrm{sat}(\varphi) = \#\mathrm{sat}(\varphi_1'') - \#\mathrm{sat}(\varphi_1')$.

Since complementive reductions are the transitive closure of parsimonious and strong complementive reductions, we have in both cases a complementive reduction from the problem $\#\mathrm{QCSP}_i(R_{\mathrm{nae}})$ to $\#\mathrm{QCSP}_i(S')$, such that $\mathrm{Pol}(S') \supseteq \mathrm{N}$. $\square$

4.6 Conclusion

We have been able to show that the Galois connection is applicable in the case of quantified constraint. This knowledge allowed us to re-prove a theorem first obtained by Edith

Hemaspaandra with a much shorter proof. This lead the way to getting a complete classification also for the counting version of quantified constraints. Again, we were able to utilize the Galois connection. In order to get completeness results for the different levels of the counting hierarchy, however, we needed to introduce a new kind of reduction, the complementive reduction. Unfortunately only the $\#{\cdot}\mathbf{\Pi_k^P}$ classes are closed under this reduction, but provably this does not hold for the $\#{\cdot}\mathbf{\Sigma_k^P}$classes. Nevertheless, this reduction was sensible to use in the context of problems exhibiting a complementive nature, because it enables us to separate between the different levels of the counting hierarchy. We think that this reduction will probably also turn out to be useful for problems with similar properties in the counting context. An open question thus is, whether it is possible to find a reduction that closes the gaps between the $\#{\cdot}\mathbf{\Sigma_k^P}$ and $\#{\cdot}\mathbf{\Pi_k^P}$ complexity classes.

5 Graph Related Constraint Problems

5.1 Introduction

The graph isomorphism problem is a very interesting and well-studied problem in computer science. Given two graphs, the task is to determine, whether they are isomorphic, that is, whether there exists a permutation of the vertices in one graph such that both are equivalent. The reason why this problem is so famous is that it is one of the very few natural problems in the class **NP** that is not not known to be in **P** and unlikely to be **NP**-complete [KST93]. As we have already argued in the previous chapters, there are infinitely many complexity classes between **P** and **NP**, under the assumption that both are not equal. Graph isomorphism might be a candidate for a member of one of those intermediate classes.

In this chapter we will take a look at several constraint problems that are in some way related to graph isomorphism. We will first broaden a result obtained by Böhler et al. [BHRV02]. They classified the equivalence problem for constraints. This is the problem, given two constraint formulae, to decide, whether they evaluate the same for all possible assignments of their variables. However, Böhler et al. just looked at the case in which both formulae have similar properties; that is, they are both S-formulae for the same constraint language S. We allow the formulae to originate from different constraint languages. Thereafter we obtain an akin result of constraint implication; that is, the question whether one constraint formula implies a second one.

Another graph theoretical problem is the subgraph isomorphism problem. This is a generalization of graph isomorphism in the way that we are again given two graphs, but now look, whether there is a subgraph of the first graph such that a permutation of its vertices yields a graph equivalent to the second one. This problem is known to be **NP**-complete. In the main part of this chapter we will introduce the isomorphic implication problem. As we will later on prove, this problem is an extension of the subgraph isomorphism in the same way the constraint isomorphism is an extension of graph isomorphism. At the same time the correlation of constraint isomorphism and isomorphic implication is analogous to the correlation of graph isomorphism and subgraph isomorphism. We will show that isomorphic implication is, depending on the constraint languages allowed, either in **P**; **NP**-complete; or **NP**-hard, **coNP**-hard, and in $\mathbf{\Theta_2^P}$. We believe that isomorphic implication is $\mathbf{\Theta_2^P}$-complete in all those cases where it is **NP**-hard and **coNP**-hard. In Section 5.5 we show that we are able to prove this hardness for some cases.

Finally, the close connections between isomorphic implication, isomorphism, subgraph isomorphism, and graph isomorphism allow us to state an interesting and new idea, which could result in a proof that graph isomorphism is solvable in polynomial time.

In order to prove the results of this chapter we have to take a different path from the one we took in the previous chapters. The obstacle is caused by existentially quantified variables. These are needed in order to reduce one problem to another one by expressing the relations over conjunctive queries as in Theorem 2.4.6. For the problem of satisfiability of a constraint formula, it does not matter, whether some of the variables are existentially quantified or not. The problem stays the same. In the context of quantified constraints we already have implicit existentially quantified constraints, so we were able to fit additional quantified variables into the existing quantifiers. In the definition of the equivalence and similar problems, there are no quantifiers allowed and contrary to satisfiability, omitting quantifiers for a variable makes a difference. Take, for example, the formulae $\varphi(x, y) = x \vee y$ and $\psi(x) = \exists y(x \vee y)$. Obviously $\varphi(x, y)$ and $\psi(x)$ are not equivalent, since $\varphi(0, 0)$ is false, whereas $\psi(x)$ is true for any assignment. However, omitting the quantifiers results in two equivalent formulae (they are indeed the same). Therefore, we cannot avail ourselves of the structure exhibited by Post's lattice, but have to cover "manually" all possible cases by our proofs. This makes them much more elaborate and involved, which will be especially apparent in the case of isomorphic implication.

5.2 Prerequisites

Before we start with our main contribution for this chapter, we will refresh some basic notations of graph theory and state some results of previous papers. Note, that all reductions, unless specified differently, are polynomial-time many-one reductions; that is, all hardness and completeness results should be seen as being $\leq_m^{\mathbf{P}}$-hard and $\leq_m^{\mathbf{P}}$-complete, respectively, for their complexity classes.

Definition 5.2.1 *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. The* graph isomorphism problem *(abbreviated by* GI*) is the problem to decide, whether G and H are isomorphic (denoted by $G \cong H$), that is, whether there exists a bijection π from V_G to V_H such that for all $u, v \in V_G$ we have that $(u, v) \in E_G$ if and only if $(\pi(u), \pi(v)) \in E_H$.*

The exact complexity of GI is not known. It is only known that it is in **NP**; it is not known to be in **P** and it is considered unlikely to be **NP**-complete. Therefore, we will denote from now on the complexity of the graph isomorphism problem by **GI**; that is, GI $\in$ **GI**. A generalization of the GI is the subgraph isomorphism problem. Its complexity is, in contrast to GI, well-known.

Definition 5.2.2 *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. The* subgraph isomorphism problem *(abbreviated by* SGI*) is the problem to decide, whether G contains a subgraph isomorphic to H (denoted by $G \widetilde{\leq} H$), that is, whether there exists a subgraph $G' = (V_{G'}, E_{G'})$ of G with $V_{G'} \subseteq V_G$ and $E_{G'} \subseteq E_G$ such that G' and H are isomorphic.*

Theorem 5.2.3 ([GJ79]) *The subgraph isomorphism problem is* **NP***-complete.*

The subgraph isomorphism problem can be restricted to having no isolated vertices and is still **NP**-complete, because isolated vertices are trivial for determining the isomorphism.

The following two problems of equivalence and isomorphism of constraint formulae have been defined and classified by Böhler et al. [BHRV02, BHRV04].

Definition 5.2.4 ([BHRV02]) *Let S be a constraint language. Then* EQUIV(S) *is the problem, given two S-formulae φ and ψ, to decide, whether φ and ψ are equivalent (denoted by $\varphi \equiv \psi$), that is, whether for every assignment to the variables, φ is satisfied if and only if ψ is satisfied.*

Theorem 5.2.5 ([BHRV02]) *Let S be a constraint language.*

- *If S is Schaefer, then* EQUIV(S) *is in* **P**.
- *Otherwise* EQUIV(S) *is* **coNP***-complete.*

Definition 5.2.6 ([BHRV04]) *Let S be a constraint language. Then* ISO(S) *is the problem, given two S-formulae φ and ψ over variables X, to decide, whether φ is isomorphic to ψ (denoted by $\varphi \cong \psi$), that is, whether there exists a permutation π of X such that $\pi(\varphi) \equiv \psi$.*

Theorem 5.2.7 ([BHRV04]) *Let S be a constraint language.*

- *If S is 2-affine, then* ISO(S) *is in* **P**.
- *If S is Schaefer and not 2-affine, then* ISO(S) *is polynomial-time many-one equivalent to* GI.
- *Otherwise, S is not Schaefer and* ISO(S) *is* **coNP***-hard and* **GI***-hard.*

Finally, we need the notion of validity of constraint formulae, combining the properties of 0-valid and 1-valid in two formulae.

Definition 5.2.8 *Let S be a constraint language. Two S-formulae φ and ψ are said to have the same* validity *if and only if they are both 0-valid or both not 0-valid and additionally both are 1-valid or both are not 1-valid.*

5.3 Results for Two Constraint Languages

5.3.1 Equivalence

First, we give the definition for the equivalence problem for two (possibly different) constraint languages.

Definition 5.3.1 *Let S_1 and S_2 be two constraint languages. Then* EQUIV(S_1, S_2) *is the problem, given an S_1-formula φ and an S_2-formula ψ, to decide, whether φ and ψ are equivalent, that is, whether for every assignment to the variables, φ is satisfied if and only if ψ is satisfied.*

This definition is very similar to the equivalence problem with just one constraint language in Definition 5.2.4, since $\mathrm{EQUIV}(S) = \mathrm{EQUIV}(S, S)$. We only allow the second language to be different from the first one, which results in some more possibilities as we will see in the following. Note that it is possible, that S_1 and S_2 are not Schaefer, but yet $\mathrm{EQUIV}(S_1, S_2)$ is in $\mathbf{P}$. This happens, for example, if S_1 is 0-valid, and every constraint in S_2 is not 0-valid. We will show that problems caused by the validity of the sets of constraints are the only cases that break Theorem 5.2.5.

Before we can state our main theorem, we need to define the function *equal-valid*, which has two constraint languages as input and the output is also a pair of constraint languages. The intention of this definition is to take those constraints out of the two languages, which cannot ever be equivalent, because they have a different validity.

Definition 5.3.2 *Let S_1 and S_2 be two constraint languages. We then define the function equal-valid as equal-valid$(S_1, S_2) = (S_1', S_2')$ with $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$ being maximal subsets, such that both have the same validity.*

Now we can completely classify the equivalence problem for two constraint languages.

Theorem 5.3.3 *Let S_1 and S_2 be two constraint languages and let further be $(S_1', S_2') =$ equal-valid(S_1, S_2).*

- *If S_1 and S_2 are Schaefer, then $\mathrm{EQUIV}(S_1, S_2)$ is in $\mathbf{P}$.*
- *Otherwise, if S_1' and S_2' are Schaefer, or $S_1' = \emptyset$, or $S_2' = \emptyset$, or S_1' or S_2' contain only the full relation, then $\mathrm{EQUIV}(S_1, S_2)$ is also in $\mathbf{P}$.*
- *Otherwise S_1' or S_2' are not Schaefer and $\mathrm{EQUIV}(S_1, S_2)$ is $\mathbf{coNP}$-complete.*

The problem with just one constraint language $\mathrm{EQUIV}(S)$ has already been handled in [BHRV02]. The easy case of their dichotomy carries over.

Lemma 5.3.4 *Let S_1 and S_2 be two constraint languages such that both are Schaefer; then $\mathrm{EQUIV}(S_1, S_2)$ is in $\mathbf{P}$.*

Proof. Let $A \uplus B$ be the marked product, defined as $A \uplus B = \{0x \mid x \in A\} \cup \{1x \mid x \in B\}$. If S_1 and S_2 are Schaefer, then $\overline{\mathrm{CSP}(S_1)}$ and $\overline{\mathrm{CSP}(S_2)}$ are in $\mathbf{P}$. Obviously, also the marked product $\overline{\mathrm{CSP}(S_1)} \uplus \overline{\mathrm{CSP}(S_2)}$ is in $\mathbf{P}$, since we only have to check the first digit and then decide, whether to check for $\overline{\mathrm{CSP}(S_1)}$ or $\overline{\mathrm{CSP}(S_2)}$. Following the proof of Lemma 7 [BHRV02], $\mathrm{EQUIV}(S_1, S_2)$ is also polynomial-time conjunctive truth-table reducible to $\overline{\mathrm{CSP}(S_1)} \uplus \overline{\mathrm{CSP}(S_2)}$: If k is the maximal arity of the constraints in S_1 and S_2, then it is easy to check with 2^k conjunctive truth-table queries, whether a constraint formula implies a given constraint by simply trying out all 2^k different assignments to the variables. We thus merely have to test, whether an S_1-formula φ implies all clauses of an S_2-formula ψ and vice versa. □

Before we can prove the next theorem, we need the following proposition.

Proposition 5.3.5 *Let S_1 and S_2 be two constraint languages. If S_1 is not Schaefer and S_2 contains only the full relation, then* $\mathrm{EQUIV}(S_1, S_2)$ *is in* $\mathbf{P}$.

Proof. Let φ be an S_1-formula and let ψ be an S_2-formula. Further let k be the maximal arity of constraints in S_1. Since S_2 contains only the full relation, the formulae φ and ψ are only equivalent, if $\varphi \equiv 1$. Thus, we need to check for every clause C of φ, whether every assignment to the variables satisfies C. Since there are at most 2^k assignments to consider, this is clearly in $\mathbf{P}$. □

For the lower bound, the following four base cases (i. e., all sets have the same validity) yield also the same results as in [BHRV02].

Lemma 5.3.6 *Let S_1 and S_2 be two constraint languages. Let further S_1 be not Schaefer.*

1. *If S_1 and S_2 are not 0-valid and not 1-valid, then* $\mathrm{EQUIV}(S_1, S_2)$ *is* **coNP***-hard.*
2. *If S_1 and S_2 are 0-valid, but not 1-valid, then* $\mathrm{EQUIV}(S_1, S_2)$ *is* **coNP***-hard.*
3. *If S_1 and S_2 are not 0-valid, but 1-valid, then* $\mathrm{EQUIV}(S_1, S_2)$ *is* **coNP***-hard.*
4. *If S_1 and S_2 are 0-valid and 1-valid, but S_2 does not only contain the full relation, then* $\mathrm{EQUIV}(S_1, S_2)$ *is* **coNP***-hard.*

Proof. Cases 1 through 3 follow directly from the proofs in [BHRV02] (see Claims 9, 12.1, and 12.2, resp.).

For case 4 we will do a reduction from $\overline{\mathrm{CSP}_{\neq 0,1}(S_1)}$ (where $\mathrm{CSP}_{\neq 0,1}(S)$ is the problem to decide, whether an S-formula has a satisfying assignment different from the all-0 and all-1 vector) to $\mathrm{EQUIV}(S_1, S_2)$. This will give us the result, since by Proposition 10 [BHRV02], $\overline{\mathrm{CSP}_{\neq 0,1}(S_1)}$ is **coNP**-complete. Let φ be an S_1-formula. Note that the all-0 vector and the all-1 vector satisfy φ trivially. Therefore, $\varphi \notin \mathrm{CSP}_{\neq 0,1}(S_1)$ if and only if φ is equivalent to $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$.

We will first assume S_2 is not complementive. By the proof of Claim 14 [BHRV02], there exists an S_2-formula ψ that is equivalent to $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$, since ψ is 0-valid and 1-valid. Subsequently the above reduction can be carried out.

Next assume that S_2 is 0-valid, 1-valid, and complementive. Furthermore suppose that S_2 contains a constraint C that is not always 1 (otherwise, according to Proposition 5.3.5, $\mathrm{EQUIV}(S_1, S_2)$ would be in $\mathbf{P}$). Rearrange the order of the variables such that $C(0, \ldots, 0, 1, \ldots, 1) = 0$. Let $A(x, y) = C(x, \ldots, x, y, \ldots, y)$. Then $A(0, 0) = A(1, 1) = 1$, because C is 0-valid and 1-valid, and $A(0, 1) = A(1, 0) = 0$, because C is complementive. This implies that $A(x, y) \equiv (x \leftrightarrow y)$. Thus,

$$\bigwedge_{1 \le i \le j \le n} A(x_i, x_j) \quad \equiv \quad \bigwedge_{i=1}^{n} x_i \;\vee\; \bigwedge_{i=1}^{n} \overline{x_i}.$$

Likewise we can perform the above reduction here, too. □

The only cases, whose classification has not been proved so far, are the ones, when both constraint languages are not Schaefer and do not have the same validity. In these cases the following proposition tells us, that we only need to look at certain subsets of the constraint languages to determine the complexity of the equivalence problem.

Proposition 5.3.7 *Let S_1 and S_2 be two constraint languages. If φ is an S_1-formula of which at least one of the clauses is not 0-valid (not 1-valid), and ψ is a formula with only 0-valid (1-valid, resp.) S_2-clauses, then $\varphi \not\equiv \psi$.*

Proof. This trivially holds, since $\varphi(0,\dots,0) = 0$, but $\psi(0,\dots,0) = 1$. The 1-valid part is analogous. □

Thus, if a constraint language S_1 is not 0-valid, but another constraint language S_2 is 0-valid, then it suffices to look at $\text{EQUIV}(S_1', S_2)$ with S_1' being a maximal subset of S_1, that is 0-valid (i. e., the set of all 0-valid constraints of S_1).

For a complete classification we therefore need the restricted constraint languages obtained from the original ones as stated in Definition 5.3.2. Thus, the following corollary is obvious, because only trivial cases are left out.

Corollary 5.3.8 *Let S_1 and S_2 be two constraint languages and let S_1' and S_2' be the results of equal-valid(S_1, S_2). Then the two problems* $\text{EQUIV}(S_1, S_2)$ *and* $\text{EQUIV}(S_1', S_2')$ *have the same complexity.*

Lemma 5.3.9 *Let S_1 and S_2 be two constraint languages and let $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$ be sets as in Definition 5.3.2. If S_1' and S_2' are Schaefer or $S_1' = \emptyset$ or $S_2' = \emptyset$ or S_1' contains only the full relation or S_2' contains only the full relation, then* $\text{EQUIV}(S_1, S_2)$ *is in* **P**; *otherwise* $\text{EQUIV}(S_1, S_2)$ *is* **coNP**-*hard.*

Proof. According to Corollary 5.3.8, instead of looking at the problem $\text{EQUIV}(S_1, S_2)$, we can prove the theorem for $\text{EQUIV}(S_1', S_2')$, where S_1' and S_2' are obtained from S_1 and S_2 by the function *equal-valid*. Now for the case that S_1' and S_2' are Schaefer we have already shown in Lemma 5.3.4 that $\text{EQUIV}(S_1', S_2')$ is in **P**.

The case that at least one of the sets S_1' and S_2' are empty is also in **P**. For this case, an S_1-formula will never be equivalent to an S_2-formula, since there is no constraint in S_1 that has the same validity as any constraint in S_2.

And finally the case that either of the sets contains only the full relation is already covered by Proposition 5.3.5. □

Finally, it needs to be shown, that all equivalence problems are always in **coNP**, making all **coNP**-hardness results **coNP**-complete. This however also carries over directly from [BHRV02].

5.3.2 Implication

First, we give the definition for the implication problem for two (possibly different) constraint languages.

Definition 5.3.10 *Let S_1 and S_2 be two constraint languages. Then* $\mathrm{IMP}(S_1, S_2)$ *is the problem, given an S_1-formula φ and an S_2-formula ψ, to decide, whether φ implies ψ (denoted by $\varphi \Rightarrow \psi$), that is, whether every assignment that satisfies φ also satisfies ψ.*

The following main theorem gives a complete classification for the implication problem with two constraint languages. Similar to the previous section, it is necessary to use the function *equal-valid* from Definition 5.3.2. Here, however, only the second part of the output is used.

Theorem 5.3.11 *Let S_1 and S_2 be two constraint languages and further let $S_2' \subseteq S_2$ be a maximal subset as in Definition 5.3.2.*

- *If S_1 is Schaefer, then* $\mathrm{IMP}(S_1, S_2)$ *is in* $\mathbf{P}$.
- *If S_2' is empty or S_2' only contains the full relation, then* $\mathrm{IMP}(S_1, S_2)$ *is also in* $\mathbf{P}$.
- *Otherwise* $\mathrm{IMP}(S_1, S_2)$ *is* **coNP***-complete.*

We will first look at the easy cases again. There are two properties, which make the implication problem easy. One of them is, if the first constraint language is Schaefer.

Lemma 5.3.12 *Let S_1 and S_2 be two constraint languages. If S_1 is Schaefer, then* $\mathrm{IMP}(S_1, S_2)$ *is in* $\mathbf{P}$.

Proof. Let φ be an S_1-formula and ψ be an S_2-formula. Then φ implies ψ if and only if $\varphi \Rightarrow C$ for every clause C in ψ.

Following the proof of Lemma 7 in [BHRV02], this can be checked with 2^k conjunctive truth-table queries to $\mathrm{CSP}(S_1)$, where k is the maximal arity of S_2. Since S_1 is Schaefer, $\mathrm{CSP}(S_1)$ is in $\mathbf{P}$ and so is $\mathrm{IMP}(S_1, S_2)$. □

The other case, such that the implication problem is also in $\mathbf{P}$, occurs when the two constraint languages are too different.

Lemma 5.3.13 *Let S_1 and S_2 be two constraint languages and further let $(S_1', S_2') =$ equal-valid(S_1, S_2). If S_2' is empty or only contains the full relation, then* $\mathrm{IMP}(S_1, S_2)$ *is in* $\mathbf{P}$.

Proof. Since implication is a one-sided version of equivalence, we are only looking at S_2' and not S_1'. Obviously if S_2' is empty, it means that S_1 is 0-valid or 1-valid (or both) and S_2 contains only not 0-valid or not 1-valid (or both) constraints. Thus, for any S_1-formula φ and any S_2-formula ψ, it is never possible, that $\varphi \Rightarrow \psi$. The all-0 vector or the all-1 vector (or both) will always satisfy φ, but never ψ.

On the other hand, if S_2' only contains the full relation, all implications are trivially true, since any S_2'-formula is equivalent to 1. □

As in the equivalence case, we have again four base cases for the lower bound. However, they are slightly different this time. If the second constraint language is 0-valid (1-valid), it is not important, whether the first one is 0-valid or not 0-valid (1-valid or not 1-valid, resp.).

Theorem 5.3.14 *Let S_1 and S_2 be two constraint languages, such that S_1 is not Schaefer.*

1. *If S_1 and S_2 are not 0-valid and not 1-valid, then* $\mathrm{IMP}(S_1, S_2)$ *is* **coNP***-hard.*
2. *If S_1 and S_2 are not 0-valid and S_2 is 1-valid, then* $\mathrm{IMP}(S_1, S_2)$ *is* **coNP***-hard.*
3. *If S_1 and S_2 are not 1-valid and S_2 is 0-valid, then* $\mathrm{IMP}(S_1, S_2)$ *is* **coNP***-hard.*
4. *If S_2 is 0-valid and 1-valid, then* $\mathrm{IMP}(S_1, S_2)$ *is* **coNP***-hard.*

Proof. The proofs are analogous to the proofs of Theorem 5.3.6 in the previous section. Just instead of reductions to $\mathrm{EQUIV}(S_1, S_2)$, we have reductions to $\mathrm{IMP}(S_1, S_2)$. Thus, an S_1-formula φ is not in $\mathrm{CSP}_{\neq 0,1}(S_1)$ if and only if

$$\varphi \quad \Rightarrow \quad \bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}.$$

□

This concludes the lower bounds. In order to obtain **coNP**-hardness we still need to show membership in **coNP**, which is obvious.

Lemma 5.3.15 *Let S_1 and S_2 be two constraint languages; then,* $\mathrm{IMP}(S_1, S_2)$ *is in* **coNP**.

Proof. Let φ be an S_1-formula and ψ be an S_2-formula. Then, $\varphi \not\Rightarrow \psi$ if and only if there exists an assignment that satisfies φ, but does not satisfy ψ. Such an assignment can simply be guessed by a nondeterministic TM. Hence, the question whether $\varphi \Rightarrow \psi$ is in **coNP**. □

5.4 Isomorphic Implication

The isomorphic implication problem is a combination of the isomorphism problem from Definition 5.2.6 and the implication problem from Definition 5.3.10.

Definition 5.4.1 *Let S be a constraint language. Then* ISO-IMP(S) *is the problem, given two S-formulae φ and ψ over variables X, to decide, whether φ isomorphically implies ψ (denoted by $\varphi \overset{\cong}{\Rightarrow} \psi$), that is, whether there exists a permutation π of X such that $\pi(\varphi) \Rightarrow \psi$.*

We will now show, as mentioned in the introduction, that our definition of ISO-IMP is sensible in the way that on the one hand it extends the constraint isomorphism the same way SGI extends GI, and on the other hand it extends SGI in the same way that constraint isomorphism is an extension of GI. However, first we need a way to translate graphs to constraint formulae. The following definition describes the standard technique to transform graphs into 2-CNF with only positive literals, such that every graph without isolated vertices corresponds to a unique constraint formula.

Definition 5.4.2 *Let $G = (V, E)$ be a graph without isolated vertices. The* standard translation *from graphs to constraint formulae is*

$$t_{g\to c}(G) \quad = \bigwedge_{(i,j)\in E} x_i \vee x_j.$$

Lemma 5.4.3 *1. Let S be a constraint language and let φ and ψ be S-formulae. Then $\varphi \cong \psi$ if and only if $\varphi \overset{\cong}{\Rightarrow} \psi$ and $\psi \overset{\cong}{\Rightarrow} \varphi$.*

2. Let G and H be graphs without isolated vertices. Then $G = (V_G, E_G)$ contains a subgraph isomorphic to $H = (V_H, E_H)$ if and only if $t_{g\to c}(G) \overset{\cong}{\Rightarrow} t_{g\to c}(H)$.

Proof.

1. The left-to-right direction is obvious. For the other direction assume that $\varphi \overset{\cong}{\Rightarrow} \psi$ and $\psi \overset{\cong}{\Rightarrow} \varphi$ and for a contradiction $\varphi \not\cong \psi$. Let X be the set of variables occurring in $\varphi \wedge \psi$ and let π and ρ be permutations on X such that $\pi(\varphi) \Rightarrow \psi$ and $\rho(\psi) \Rightarrow \varphi$. Since $\varphi \not\cong \psi$, we have that $\pi(\varphi) \not\equiv \psi$. Thus, there exists an assignment that satisfies ψ, but does not satisfy $\pi(\varphi)$ or there exists an assignment that satisfies $\pi(\varphi)$, but does not satisfy ψ. Since $\rho(\psi) \Rightarrow \varphi$, there have to be at least as many satisfying assignments for φ as there are for ψ, since the permutations do not alter the number of satisfying assignments of a formula. And similarly since $\pi(\varphi) \Rightarrow \psi$, there have to be at least as many satisfying assignments for ψ as there are for φ. Thus, $\pi(\varphi)$ and ψ have the same number of satisfying assignments, but that is a contradiction to $\pi(\varphi) \not\equiv \psi$.

2. For the left-to-right direction assume that $G' = (V_{G'}, E_{G'})$ is a subgraph of G such that G' is isomorphic to H. Then there exists a bijection π from $V_{G'}$ to V_H such that $\pi(G') = H$. We define a permutation ρ of the variables occurring in $t_{g\to c}(G) \wedge t_{g\to c}(H)$ such that $\rho(x_i) = x_{\pi(i)}$ for all $i \in V_{G'}$. It is easy to see that $\rho(t_{g\to c}(G')) = t_{g\to c}(\pi(G')) = t_{g\to c}(H)$. Since G' is a subgraph of G, we have that $t_{g\to c}(G) \Rightarrow t_{g\to c}(G')$ and therefore $\rho(t_{g\to c}(G)) \Rightarrow t_{g\to c}(H)$.

For the other direction, assume that $t_{g\to c}(G) \overset{\cong}{\Rightarrow} t_{g\to c}(H)$. Then there exists a permutation π on the variables occurring in $t_{g\to c}(G) \wedge t_{g\to c}(H)$ such that $\pi(t_{g\to c}(G)) \Rightarrow t_{g\to c}(H)$. Now let G' be a graph such that $t_{g\to c}(H) = \pi(t_{g\to c}(G'))$ and G' does not have isolated vertices. Then obviously $\pi(t_{g\to c}(G)) \Rightarrow \pi(t_{g\to c}(G'))$ and hence $t_{g\to c}(G) \Rightarrow t_{g\to c}(G')$. Since $t_{g\to c}(G)$ is maximal in the sense that all clauses $x_i \vee x_j$ that are implied by $t_{g\to c}(G)$ are already contained in $t_{g\to c}(G)$, it follows that G' is a subgraph of G. Because $t_{g\to c}(H) = \pi(t_{g\to c}(G'))$ and H and G' do not contain isolated vertices, G' is isomorphic to H.

□

The following is our main theorem and gives a trichotomy-like classification for the isomorphic implication problem.

Theorem 5.4.4 *Let S be a constraint language.*

- *If S contains only constants, the identity function, or negation, then* ISO-IMP(S) *is in* **P**.
- *Otherwise, if S is Schaefer, then* ISO-IMP(S) *is* **NP***-complete.*
- *If S is not Schaefer, then* ISO-IMP(S) *is* **NP***-hard,* **coNP***-hard, and in* $\mathbf{\Theta_2^P}$.

The proof of this theorem will follow by the lemmas of the next sections. We will first prove the upper bounds and then the lower bounds.

5.4.1 Upper Bounds

We will start with the case, when isomorphic implication is solvable in polynomial time.

Lemma 5.4.5 *Let S be a constraint language. If S contains only constants, the identity function, or negation, then* ISO-IMP(S) *is in* **P**.

Proof. Let φ and ψ be two S-formulae. Since each clause is either a constant or a literal (positive or negative), we will distinguish two cases: First, assume that φ or ψ is equivalent to a constant. Note, that the problem, whether an S-formula is equivalent to a constant, is in **P**, since an S-formula is equivalent to 1 if and only if all clauses are 1 and it is equivalent to 0 if at least one clause is 0 or if one variable occurs positive and negative in the formula. Then there are three possibilities to look at and all of them lead to a polynomial-time algorithm for deciding ISO-IMP(S):

- If φ is equivalent to 1, then $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if ψ is also equivalent to 1.
- If φ is equivalent to 0 or ψ is equivalent to 1, then always $\varphi \overset{\cong}{\Rightarrow} \psi$.
- If ψ is equivalent to 0, then $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if φ is also equivalent to 0.

If neither φ nor ψ is equivalent to a constant, we claim that $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if the number of positive literals in φ is greater or equal to the number of positive literals in ψ and the number of negative literals in φ is greater or equal to the number of negative literals in ψ.

For the left-to-right direction assume that π is a permutation of the variables occurring in $\varphi \wedge \psi$ such that $\pi(\varphi) \Rightarrow \psi$. Let φ' be the formula φ without constants and let ψ' be the formula ψ without constants. Since φ and ψ are neither 1 nor 0, only clauses with the constant 1 are being deleted; thus φ' and ψ' are satisfiable. Hence, for all literals l such that $\pi(\varphi') \Rightarrow l$ we have that l is a clause in $\pi(\varphi')$. Then all literals occurring in ψ also occur in $\pi(\varphi)$, because $\pi(\varphi) \Rightarrow \psi$. Hence, the number of positive (negative) literals in φ is greater or equal than the number of positive (negative) literals in ψ.

For the other direction, suppose that the number of positive literals in φ is greater or equal than the number of positive literals in ψ and the number of negative literals in φ is greater or equal than the number of negative literals in ψ. Since both φ and ψ are not equivalent to 0, no variable can occur positively and negatively in one of the formulae. Therefore, one can easily define a permutation π that maps every positive literal in φ to a positive literal in ψ and every negative literal in φ to a negative literal in ψ. Obviously, $\pi(\varphi) \Rightarrow \psi$ and thus $\varphi \overset{\cong}{\Rightarrow} \psi$, which proves the claim.

Since counting the literals in a formula is trivially in **P**, this concludes the proof. □

It is easy to see that isomorphic implication is in **NP**, if the constraint language is Schaefer.

Lemma 5.4.6 *Let S be a constraint language such that S is Schaefer. Then* ISO-IMP(S) *is in* **NP**.

Proof. Let φ and ψ be two S-formulae over variables X. Then $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if there exists a permutation π of X such that $\pi(\varphi) \Rightarrow \psi$. And further $\pi(\varphi) \Rightarrow \psi$ if and only if $\pi(\varphi) \wedge \psi \equiv \pi(\varphi)$. Since S is Schaefer, it can be determined in polynomial time, whether two S-formulae are equivalent (see [BHRV02]). □

Finally, we show that isomorphic implication is always in $\mathbf{\Theta_2^P}$.

Lemma 5.4.7 *Let S be an arbitrary constraint language. Then* ISO-IMP(S) *is in* $\mathbf{\Theta_2^P}$.

Proof. Let φ and ψ be two S-formulae over variables X. Let φ' be the conjunction of all constraint applications over X that are implied by S. Such a formula can be computed in polynomial time with parallel access to an **NP**-oracle; that is, it is computable in $\mathbf{\Theta_2^P}$ (see the proof of Claim 22 in [BHRV02]). Thus, to find out, whether $\varphi \overset{\cong}{\Rightarrow} \psi$, we first have to determine, whether there exists a permutation π on X such that $\pi(\varphi) \Rightarrow \psi$ (this can be done by one query to an **NP**-oracle) and then we have to determine, whether all clauses of ψ are contained in φ'. By [BH91] two rounds of queries to **NP** are the same as one round of queries to **NP** and hence ISO-IMP(S) is in $\mathbf{\Theta_2^P}$. □

5.4.2 Lower Bounds

As we argued in the introduction, the Galois connection does not work for isomorphic implication. Therefore, it is not possible to prove hardness for a few selected cases and implicitly cover all cases. We have to take a much closer look and analyze exactly what cases can occur. In the case of hardness for **NP** we show that there are exactly ten cases to consider.

Lemma 5.4.8 *Let C be a k-ary constraint such that $C(x_1, \ldots, x_k)$ is not equivalent to a conjunction of literals. And let S be the constraint language consisting of C; that is, $S = \{C\}$. Then there exists an S-formula that is equivalent to one of the following formulae:*

1. $t \wedge (x \vee y)$,	*2.* $\overline{f} \wedge t \wedge (x \vee y)$,
3. $\overline{f} \wedge (\overline{x} \vee \overline{y})$,	*4.* $\overline{f} \wedge t \wedge (\overline{x} \vee \overline{y})$,
5. $x \leftrightarrow y$,	*6.* $t \wedge (x \leftrightarrow y)$,
7. $\overline{f} \wedge (x \leftrightarrow y)$,	*8.* $\overline{f} \wedge t \wedge (x \leftrightarrow y)$,
9. $x \oplus y$, *or*	*10.* $\overline{f} \wedge t \wedge (x \oplus y)$.

Proof. First, suppose that C (and thus S) is not 2-affine. According to [BHRV04, Lemma 24], there exists an S-formula φ such that φ is equivalent to either of the following ten formulae. We will show that in each of those cases the formula is either of the form we need or we can create a different S-formula equivalent to one of the ten cases we need.

- $\varphi(x, y) = \overline{x} \wedge y$: We know from the proofs of Theorems 15 and 17 of [BHRV04] that since S is not 2-affine, there exists an S-formula $\psi(0, 1, x, y)$ that is equivalent to $x \vee y$, $\overline{x} \vee \overline{y}$, $\overline{x} \vee y$, or $x \oplus y$. Then $\varphi(f, t) \wedge \psi(f, t, x, y) \wedge \psi(f, t, y, x)$ is equivalent to $\overline{f} \wedge t \wedge (x \vee y)$ (i. e., formula 2), $\overline{f} \wedge t \wedge (\overline{x} \vee \overline{y})$ (i. e., formula 4), $\overline{f} \wedge t \wedge (x \leftrightarrow y)$ (i. e., formula 8), or $\overline{f} \wedge t \wedge (x \oplus y)$ (i. e., formula 10).
- $\varphi(x, y) = \overline{x} \vee y$: Then $\varphi(x, y) \wedge \varphi(y, x) = x \leftrightarrow y$ (i. e., formula 5).
- $\varphi(x, y) = x \oplus y$: This is formula 9.
- $\varphi(x, y) = x \leftrightarrow y$: This is formula 5.
- $\varphi(t, x, y) = t \wedge (\overline{x} \vee y)$: Then $\varphi(t, x, y) \wedge \varphi(t, y, x) = t \wedge (x \leftrightarrow y)$ (i. e., formula 6).
- $\varphi(t, x, y) = t \wedge (x \leftrightarrow y)$: This is formula 6.
- $\varphi(t, x, y) = t \wedge (x \vee y)$: This is formula 1.
- $\varphi(f, x, y) = \overline{f} \wedge (\overline{x} \vee y)$: Then $\varphi(f, x, y) \wedge \varphi(f, y, x) = \overline{f} \wedge (x \leftrightarrow y)$ (i. e., formula 7).

- $\varphi(f,x,y) = \overline{f} \wedge (x \leftrightarrow y)$: This is formula 7.
- $\varphi(f,x,y) = \overline{f} \wedge (\overline{x} \vee \overline{y})$: This is formula 3.

What remains to show is the case that S is 2-affine. In this case we know from Lemma 9 [BHRV04] that there exists a polynomial-time computable normal form function that maps every S-formula φ to an equivalent formula ψ of the form

$$\psi \quad = \quad \bigwedge_{x \in Z} \overline{x} \ \wedge \ \bigwedge_{x \in O} x \ \wedge \ \bigwedge_{i=1}^{\ell} \left(\left(\bigwedge_{x \in X_i} x \ \wedge \ \bigwedge_{y \in Y_i} \overline{y} \right) \vee \left(\bigwedge_{x \in X_i} \overline{x} \ \wedge \bigwedge_{y \in Y_i} y \right) \right),$$

where Z, O, X_1, Y_1, $\ldots$, X_ℓ, Y_ℓ are pairwise disjoint subsets of $\{x_1, \ldots, x_k\}$ such that $X_i \cup Y_i \neq \emptyset$ for all $1 \leq i \leq \ell$. Note that Z contains all the variables that have to be set to zero and O contains all the variables that have to be set to one. Thus, we replace in φ every variable in Z by f and every variable in O by t. Furthermore, for each i the variables in X_i have to take a different value from those in Y_i (and accordingly all variables in a set X_i respectively Y_i have to take the same value). Since φ and therefore also ψ is not equivalent to a conjunction of literals, there exists an i such that $|X_i \cup Y_i| \geq 2$.

There are now two cases to consider: First, if there exists an i such that $X_i \neq \emptyset$ and $Y_i \neq \emptyset$; then, replace in φ every variable in $\bigcup_j X_j$ by x and every variable in $\bigcup_j Y_j$ by y. The resulting formula is equivalent to $x \oplus y$ (i. e., formula 9), $t \wedge (x \oplus y)$, $\overline{f} \wedge (x \oplus y)$, or $\overline{f} \wedge t \wedge (x \oplus y)$ (i. e., formula 10). In the second case the conjunction $t \wedge (x \oplus y) \wedge t \wedge (t \oplus f)$ is equivalent to formula 10, and in the third case the conjunction $\overline{f} \wedge (x \oplus y) \wedge \overline{f} \wedge (t \oplus f)$ is also equivalent to formula 10.

On the other hand, if for all i one of the sets X_i or Y_i is empty, then let i be such that either $|X_i| \geq 2$ or $|Y_i| \geq 2$. If we now replace in φ one of the variables in $X_i \cup Y_i$ by x and all other variables in $\bigcup_j X_j \cup Y_j$ by y, the resulting formula is equivalent to $x \leftrightarrow y$ (i. e., formula 5), $t \wedge (x \leftrightarrow y)$ (i. e., formula 6), $\overline{f} \wedge (x \leftrightarrow y)$ (i. e., formula 7), or $\overline{f} \wedge t \wedge (x \leftrightarrow y)$ (i. e., formula 8). □

With the help of Lemma 5.4.8 we only have to show **NP**-hardness for ten specific cases. We can even ease our work further by noticing that the isomorphic implication problem has a kind of duality property. It is computationally equivalent to the problem where all constraints are replaced by their complementive version, which will be formalized in the following definition. This originates from [Hem04], where a similar property has been proved for the satisfiability of quantified constraints (which we did not need in Chapter 4, because it is implicit in Post's classes).

Definition 5.4.9 ([Hem04]) *1. Let R be a k-ary constraint; then, R^C is the k-ary constraint such that for all $s \in \{0,1\}^k$, we have that $R^C(s) = R(\overline{s})$, where $\overline{s} = \neg s_1, \ldots, \neg s_k$ for $s = s_1, \ldots, s_k$.*

2. Let S be a constraint language; then, $S^C = \{R^C \mid R \in S\}$.

3. Let S be a constraint language and let $\varphi = \bigwedge_{i=1}^n R_i$ be an S-formula; then, $\varphi^C = \bigwedge_{i=1}^n R_i^C$.

Proposition 5.4.10 *Let S be a constraint language; then,* ISO-IMP(S) *is polynomial-time many-one equivalent to* ISO-IMP(S^C).

Proof. Let φ and ψ be two S-formulae and let φ^C and ψ^C be the corresponding S^C-formulae. Then $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if $\varphi^C \overset{\cong}{\Rightarrow} \psi^C$, because any permutation π with $\pi(\varphi) \Rightarrow \psi$ is also a permutation such that $\pi(\varphi^C) \Rightarrow \psi^C$. □

As we have already proved in Lemma 5.4.3, there is a close connection between the subgraph isomorphism problem and the isomorphic implication problem for positive 2-CNF; that is, a graph G contains a subgraph isomorphic to another graph H (both without isolated vertices) if and only if $t_{g\to c}(G) \overset{\cong}{\Rightarrow} t_{g\to c}(H)$. The same correspondence also leads to the **GI**-hardness for the isomorphism problem with the constraint language $\{x \lor y\}$.

We will utilize this connection in order to prove **NP**-hardness for the first four cases of Lemma 5.4.8, because they are very similar to the relation $x \lor y$, which is used for the standard translation from graphs to constraint formulae.

Lemma 5.4.11 *1.* ISO-IMP($\{t \land (x \lor y)\}$) *is* **NP**-*hard.*

2. ISO-IMP($\{\overline{f} \land t \land (x \lor y)\}$) *is* **NP**-*hard.*

3. ISO-IMP($\{\overline{f} \land (\overline{x} \lor \overline{y})\}$) *is* **NP**-*hard.*

4. ISO-IMP($\{\overline{f} \land t \land (\overline{x} \lor \overline{y})\}$) *is* **NP**-*hard.*

Proof. Similar to the standard translation from graphs to formulae from Definition 5.4.2, we will define two different translations $t'_{g\to c}$ and $t''_{g\to c}$ from graphs to constraint formulae that have the same property and that use the relations above.

1. For the first case let
$$t'_{g\to c}(G) \quad = \quad \bigwedge_{(i,j)\in E} t \land (x_i \lor x_j)$$
be a translation from graphs to constraint formulae, where $G = (V, E)$ is a graph without isolated vertices. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs without isolated vertices. We claim that G has a subgraph isomorphic to H if and only if $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$. Since the subgraph isomorphism problem is **NP**-complete, it follows that ISO-IMP($\{t \land (x \lor y)\}$) is **NP**-hard.

   To prove the claim, first assume that $G' = (V_{G'}, E_{G'})$ is a subgraph of G such that G' is isomorphic to H. Let π be the isomorphism from $V_{G'}$ to V_H such that $\pi(G') = H$. Then $t'_{g\to c}(\pi(G')) = t'_{g\to c}(H)$. We can now define a permutation ρ on the variables $\{t\} \cup \{x_i \mid i \in V_G \cup V_H\}$ such that $\rho(t) = t$ and $\rho(x_i) = x_{\pi(i)}$ for all $i \in V_{G'}$. Then $\rho(t'_{g\to c}(G')) = t'_{g\to c}(\pi(G')) = t'_{g\to c}(H)$. Since G' is a subgraph of G, we have that $t'_{g\to c}(G) \Rightarrow t'_{g\to c}(G')$ and hence $\rho(t'_{g\to c}(G)) \Rightarrow \rho(t'_{g\to c}(G'))$. Thus, $\rho(t'_{g\to c}(G)) \Rightarrow t'_{g\to c}(H)$ and therefore $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$.

For the other direction assume that $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$. Then there exists a permutation ρ on the variables occurring in $t'_{g\to c}(G) \wedge t'_{g\to c}(H)$ such that $\rho(t'_{g\to c}(G)) \Rightarrow t'_{g\to c}(H)$. It is easy to see that ρ has to map the variable t to itself. This is because t is the only variable that is implied by the constraint formula $t'_{g\to c}(\widehat{G})$ for any graph $\widehat{G}$. Also note that, if $t'_{g\to c}(\widehat{G}) \Rightarrow t \wedge (x \vee y)$, then $t \wedge (x \vee y)$ has to be a clause in $t'_{g\to c}(\widehat{G})$. If this were not the case, we could simply set t to 1, x and y to 0, and all other variables to 1. Then this assignment would satisfy $t'_{g\to c}(\widehat{G})$, but it would not satisfy $t \wedge (x \vee y)$. For the same reason we know that if $\rho(t'_{g\to c}(\widehat{G})) \Rightarrow t \wedge (x \vee y)$, then also $t \wedge (x \vee y)$ is a clause is $t'_{g\to c}(\widehat{G})$. It follows that all clauses in $t'_{g\to c}(H)$ are also clauses in $\rho(t'_{g\to c}(G))$. If we now let G' be a graph isomorphic to H such that $t'_{g\to c}(H) = \rho(t'_{g\to c}(G'))$, then obviously all clauses of $\rho(t'_{g\to c}(G'))$ are also clauses in $\rho(t'_{g\to c}(G))$ and therefore G' is a subgraph of G.

2. For the second case let

$$t''_{g\to c}(G) \quad = \quad \bigwedge_{(i,j)\in E} \overline{f} \wedge t \wedge (x_i \vee x_j)$$

be a translation from graphs to constraint formulae, where $G = (V, E)$ is a graph without isolated vertices. We now claim that for any graphs G and H without isolated vertices, $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$ if and only if $t''_{g\to c}(G) \overset{\cong}{\Rightarrow} t''_{g\to c}(H)$. Since we already showed that ISO-IMP($\{t \wedge (x \vee y)\}$) is **NP**-hard in case one, it follows that also ISO-IMP($\{\overline{f} \wedge t \wedge (x \vee y)\}$) is **NP**-hard.

First, assume that $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$. Then there exists a permutation ρ such that $\rho(t'_{g\to c}(G)) \Rightarrow t'_{g\to c}(H)$. By extending ρ to f such that $\rho(f) = f$, we have that $\rho(t''_{g\to c}(G)) \Rightarrow t''_{g\to c}(H)$ and thus $t''_{g\to c}(G) \overset{\cong}{\Rightarrow} t''_{g\to c}(H)$.

For the other direction assume that $t''_{g\to c}(G) \overset{\cong}{\Rightarrow} t''_{g\to c}(H)$. Then there exists a permutation ρ on the variables occurring in $t''_{g\to c}(G) \wedge t''_{g\to c}(H)$ such that $\rho(t''_{g\to c}(G)) \Rightarrow t''_{g\to c}(H)$. As in the previous case, ρ has to map f to itself, since f is the only variable whose negation is implied by the constraint formula originating from $t''_{g\to c}(\widehat{G})$ for any graph $\widehat{G}$. Thus, for any graph $\widehat{G}$, the formula $t''_{g\to c}(\widehat{G})$ is equivalent to $\overline{f} \wedge t'_{g\to c}(\widehat{G})$. Hence, $\rho(t'_{g\to c}(G)) \Rightarrow t'_{g\to c}(H)$ and therefore $t'_{g\to c}(G) \overset{\cong}{\Rightarrow} t'_{g\to c}(H)$.

3. Since $(\overline{f} \wedge (\overline{x} \vee \overline{y}))^C = (f \wedge (x \vee y))$, the result follows directly from Proposition 5.4.10 and case 1 of this lemma.

4. Similar to the previous case, we have that $(\overline{f} \wedge t \wedge (\overline{x} \vee \overline{y}))^C = (\overline{t} \wedge f \wedge (x \vee y))$. Then the result follows directly from Proposition 5.4.10 and case 2 of this lemma.

This covers all four cases. $\square$

The remaining six cases of Lemma 5.4.8 cannot be dealt with in the same way. The problem is that the isomorphism problem for these cases is in **P** (see Theorem 5.2.7).

So, unless GI is also in **P**, there is no reduction from graph isomorphism to constraint isomorphism in these cases. And similarly there do not seem to be simple reductions from SGI to isomorphic implication. Therefore, we will reduce from a different problem, the Unary-3-Partition.

Definition 5.4.12 *Let A be a set of $3m$ elements, $B \in \mathbb{N}$ a bound (in unary), and for each $a \in A$ let $s(a) \in \mathbb{N}$ be a size (in unary) such that $\sum_{a\in A} s(a) = mB$. Then* Unary-3-Partition *is the problem to decide, whether A can be partitioned into m disjoint sets $A_1, \dots, A_m$ such that $\sum_{a\in A_i} s(a) = B$ for $1 \leq i \leq m$.*

Theorem 5.4.13 ([GJ79]) Unary-3-Partition *is* **NP***-complete.*

Lemma 5.4.14 *1.* ISO-IMP($\{x \leftrightarrow y\}$) *is* **NP***-hard.*

2. ISO-IMP($\{t \wedge x \leftrightarrow y\}$) *is* **NP***-hard.*

3. ISO-IMP($\{\overline{f} \wedge x \leftrightarrow y\}$) *is* **NP***-hard.*

4. ISO-IMP($\{\overline{f} \wedge t \wedge x \leftrightarrow y\}$) *is* **NP***-hard.*

Proof. We will show that in the first case Unary-3-Partition is reducible to the corresponding isomorphic implication problem. And the other three cases follow from that result.

1. Let A be a set with $3m$ elements, $B \in \mathbb{N}$ a bound (in unary), and for each $a \in A$ let $s(a) \in \mathbb{N}$ be a size (in unary) such that $\sum_{a\in A} s(a) = mB$ as in the definition of Unary-3-Partition. Further let $X_1, \dots, X_m$ be m pairwise disjoint sets of B variables each. We define

   $$\varphi \quad = \quad \bigwedge_{i=1}^{m} \left(\bigwedge_{x,x' \in X_i} x \leftrightarrow x' \right).$$

   Further let $\{\widehat{X}_a \mid a \in A\}$ be a collection of $3m$ pairwise disjoint sets of variables such that $|\widehat{X}_a| = s(a)$ for all $a \in A$ and such that $\bigcup_{a\in A} \widehat{X}_a = \bigcup_{i=1}^m X_i$. We define

   $$\psi \quad = \quad \bigwedge_{a \in A} \left(\bigwedge_{x,x' \in \widehat{X}_a} x \leftrightarrow x' \right).$$

   Since the values of B and the $s(a)$'s are given in unary, we can compute both formulae φ and ψ in polynomial time. We now claim that A can be partitioned into m disjoint sets $A_1, \dots, A_m$ such that $\sum_{a\in A_i} s(a) = B$ for $1 \leq i \leq m$ if and only if $\varphi \overset{\cong}{\Rightarrow} \psi$.

   For the left-to-right direction assume that $A_1, \dots, A_m$ is a partition of A such that $\sum_{a\in A_i} s(a) = B$ for $1 \leq i \leq m$. We define a permutation π of $\bigcup_{i=1}^m X_i$ such that

$\pi(X_i) = \bigcup_{a \in A_i} \widehat{X}_a$ for $1 \le i \le m$. We now show for every clause $x \leftrightarrow x'$ in ψ that it is also a clause in $\pi(\varphi)$. Therefore, let $x \leftrightarrow x'$ be a clause in ψ. By definition of ψ there exists an $a \in A$ such that $x, x' \in \widehat{X}_a$. Let y and y' be variables such that $\pi(y) = x$ and $\pi(y') = x'$. Then by definition of π there has to exist an i such that both y and y' must be in the same set X_i. By definition of φ it follows, that $y \leftrightarrow y'$ has to be a clause in φ and thus $\pi(y) \leftrightarrow \pi(y') = x \leftrightarrow x'$ is a clause in $\pi(\varphi)$. This implies that $\varphi \overset{\cong}{\Rightarrow} \psi$ via π.

For the other direction assume that $\varphi \overset{\cong}{\Rightarrow} \psi$. Then there exists a permutation π of $\bigcup_{i=1}^{m} X_i$ such that $\pi(\varphi) \Rightarrow \psi$. Let $A_i = \{a \mid \pi(X_i) \cap \widehat{X}_a \neq \emptyset\}$ for $1 \le i \le m$. We claim that $A_1, \ldots, A_m$ is a desired partition of A. Since $\bigcup_{a \in A} \widehat{X}_a = \bigcup_{i=1}^{m} X_i$, it is obvious that all elements from the $\widehat{X}_a$ sets are in some X_i sets and thus also in the $\pi(X_i)$ sets. By definition of the A_i it is immediate that $\bigcup_{i=1}^{m} A_i = A$. Next we will show that the A_i are pairwise disjunct. Therefore, suppose for a contradiction that $A_i \cap A_j \neq \emptyset$ for some $i \neq j$. Then, there are some $z \in X_i$ and $z' \in X_j$ such that there exists an $a \in A$ with $\pi(z), \pi(z') \in \widehat{X}_a$. It follows that $\pi(z) \leftrightarrow \pi(z')$ is a clause in ψ. Since z and z' are from different sets X_i and X_j, there exists a satisfying assignment for φ with z set to 1 and z' set to 0 and therefore $\varphi \not\Rightarrow (z \leftrightarrow z')$. This implies that $\pi(\varphi) \not\Rightarrow (\pi(z) \leftrightarrow \pi(z'))$, but that is a contradiction to $\pi(\varphi) \Rightarrow \psi$. Hence, for all $i \neq j$ it follows that $A_i \cap A_j = \emptyset$ and therefore by definition of the A_i's and because there cannot be two elements in any set $\widehat{X}_a$ that are in different sets $\pi(X_i)$, we have that $\pi(X_i) = \bigcup_{a \in A_i} \widehat{X}_a$. Since π is an injection, $|X_i| = |\bigcup_{a \in A_i} \widehat{X}_a|$, and since the $\widehat{X}_a$'s are pairwise disjoint, $|\bigcup_{a \in A_i} \widehat{X}_a| = \sum_{a \in A_i} s(a)$. Since $|X_i| = B$ for $1 \le i \le m$, we have that $\sum_{a \in A_i} s(a) = B$, which concludes this reduction.

2. In the second case the constraint is $t \wedge (x \leftrightarrow y)$ and we reduce from ISO-IMP($\{x \leftrightarrow y\}$), which we just showed to be **NP**-hard. Therefore, we define two formulae φ' and ψ' analogously to φ and ψ in the previous case. Let

$$\varphi' = \bigwedge_{i=1}^{m} \left(\bigwedge_{x,x' \in X_i} t \wedge (x \leftrightarrow x') \right)$$

and

$$\psi' = \bigwedge_{a \in A} \left(\bigwedge_{x,x' \in \widehat{X}_a} t \wedge (x \leftrightarrow x') \right).$$

We now show that $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if $\varphi' \overset{\cong}{\Rightarrow} \psi'$.

First, assume that $\varphi \overset{\cong}{\Rightarrow} \psi$. Then there exists a permutation π of the variables occurring in $\varphi \wedge \psi$ such that $\pi(\varphi) \Rightarrow \psi$. If we extend π by letting $\pi(t) = t$, then $\pi(\varphi') \Rightarrow \psi'$ and thus $\varphi' \overset{\cong}{\Rightarrow} \psi'$.

For the other direction assume that $\varphi' \overset{\cong}{\Rightarrow} \psi'$. Then there exists a permutation π of the variables occurring in $\varphi' \wedge \psi'$ such that $\pi(\varphi') \Rightarrow \psi'$. By construction φ' is equivalent to $\varphi \wedge t$ and ψ' is equivalent to $\psi \wedge t$. Since t is the only variable such that $\varphi' \Rightarrow t$ for any formula φ', it follows that $\pi(t) = t$. Since t neither occurs in φ nor in ψ, obviously $\pi(\varphi) \Rightarrow \psi$ and thus $\varphi \overset{\sim}{\Rightarrow} \psi$.

3. Since $(\overline{f} \wedge (x \leftrightarrow y))^C$ is equivalent to $(t \wedge (x \leftrightarrow y))$, the result follows directly from Proposition 5.4.10 and case 2 of this lemma.

4. In the last case we again reduce from ISO-IMP($\{x \leftrightarrow y\}$). Let

$$\varphi'' \quad = \quad \bigwedge_{i=1}^{m} \left(\bigwedge_{x,x' \in X_i} t \wedge \overline{f} \wedge (x \leftrightarrow x') \right)$$

and

$$\psi'' \quad = \quad \bigwedge_{a \in A} \left(\bigwedge_{x,x' \in \widehat{X}_a} t \wedge \overline{f} \wedge (x \leftrightarrow x') \right).$$

The proof that $\varphi \overset{\sim}{\Rightarrow} \psi$ if and only if $\varphi'' \overset{\sim}{\Rightarrow} \psi''$ is analogous to that of case 2. For the one direction we can extend the permutation to $\pi(t) = t$ and $\pi(f) = f$, and for the other direction it also holds that φ'' is equivalent to $\varphi \wedge t \wedge \overline{f}$ and ψ'' is equivalent to $\psi \wedge t \wedge \overline{f}$.

□

For the remaining two cases of Lemma 5.4.8, we also reduce from Unary-3-Partition, but since we are dealing with *xor* instead of the equivalence constraint, we need to alter our proofs slightly.

Lemma 5.4.15 *1.* ISO-IMP($\{x \oplus y\}$) *is* **NP***-hard.*

2. ISO-IMP($\{\overline{f} \wedge t \wedge (x \oplus y)\}$) *is* **NP***-hard.*

Proof. We will again prove the first case via a reduction from Unary-3-Partition, and the second case by reducing from the first case.

1. Let A be a set with $3m$ elements, $B \in \mathbb{N}$ a bound (in unary), and for each $a \in A$ let $s(a) \in \mathbb{N}$ be a size (in unary) such that $\sum_{a \in A} s(a) = mB$ as in the definition of Unary-3-Partition. Further let $X_1, \ldots, X_m, Y_1, \ldots, Y_m$ be $2m$ pairwise disjoint sets of B variables each. We define

$$\varphi \quad = \quad \bigwedge_{i=1}^{m} \left(\bigwedge_{x \in X_i, y \in Y_i} x \oplus y \right).$$

Further let $\{\widehat{X}_a, \widehat{Y}_a \mid a \in A\}$ be a collection of $6m$ pairwise disjoint sets of variables such that $|\widehat{X}_a| = |\widehat{Y}_a| = s(a)$ for all $a \in A$ and such that $\bigcup_{a \in A}(\widehat{X}_a \cup \widehat{X}_a) = \bigcup_{i=1}^{m}(X_i \cup Y_i)$. We define

$$\psi \quad = \quad \bigwedge_{a \in A} \left(\bigwedge_{x \in \widehat{X}_a, y \in \widehat{Y}_a} x \oplus y \right).$$

Since the values of B and the $s(a)$'s are given in unary, we can compute both formulae φ and ψ in polynomial time. We now claim that A can be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that $\sum_{a \in A_i} s(a) = B$ for $1 \leq i \leq m$ if and only if $\varphi \overset{\cong}{\Rightarrow} \psi$.

For the left-to-right direction assume that $A_1, \ldots, A_m$ is a partition of A such that $\sum_{a \in A_i} s(a) = B$ for $1 \leq i \leq m$. We define a permutation π of $\bigcup_{i=1}^{m} (X_i \cup Y_i)$ such that $\pi(X_i) = \bigcup_{a \in A_i} \widehat{X}_a$ and $\pi(Y_i) = \bigcup_{a \in A_i} \widehat{Y}_a$ for $1 \leq i \leq m$. We now show for every clause $x \oplus y$ in ψ, that it is also a clause in $\pi(\varphi)$. Therefore, let $x \oplus y$ be an arbitrary clause in ψ. By definition of ψ there exists an $a \in A$ such that $x \in \widehat{X}_a$ and $y \in \widehat{Y}_a$. Let x' and y' be variables such that $\pi(x') = x$ and $\pi(y') = y$. Then by definition of π either $x' \in X_i$ and $y' \in Y_i$ or vice versa. It then follows from the definition of φ, that $x' \oplus y'$ has to be a clause in φ and thus $\pi(x') \oplus \pi(y') = x \oplus y$ is a clause in $\pi(\varphi)$. This implies that $\varphi \overset{\cong}{\Rightarrow} \psi$ via π.

For the other direction assume that $\varphi \overset{\cong}{\Rightarrow} \psi$. Then there exists a permutation π of $\bigcup_{i=1}^{m} (X_i \cup Y_i)$ such that $\pi(\varphi) \Rightarrow \psi$. Let $A_i = \{a \mid \pi(X_i \cup Y_i) \cap (\widehat{X}_a \cup \widehat{Y}_a) \neq \emptyset\}$ for $1 \leq i \leq m$. We claim that $A_1, \ldots, A_m$ is a desired partition of A. With the same argument as in case 1 of the proof of Lemma 5.4.14 it is immediate that $\bigcup_{i=1}^{m} A_i = A$. Next we will show that the A_i are pairwise disjunct. Therefore, suppose for a contradiction that $A_i \cap A_j \neq \emptyset$ for some $i \neq j$. Then there are some $z \in (X_i \cup Y_i)$ and $z' \in (X_j \cup Y_j)$ such there exists an $a \in A$ with $\pi(z), \pi(z') \in (\widehat{X}_a \cup \widehat{Y}_a)$. It follows that either $\psi \Rightarrow (\pi(z) \oplus \pi(z'))$ or $\psi \Rightarrow (\pi(z) \leftrightarrow \pi(z'))$. Since z and z' are from different sets $(X_i \cup Y_i)$ respectively $(X_j \cup Y_j)$, there exists a satisfying assignment for φ with z set to 1 and z' set to 0 and another satisfying assignment with z set to 1 and z' set to 1. Therefore, $\varphi \not\Rightarrow (z \oplus z')$ and $\varphi \not\Rightarrow (z \leftrightarrow z')$. This implies that $\pi(\varphi) \not\Rightarrow (\pi(z) \oplus \pi(z'))$ and $\pi(\varphi) \not\Rightarrow (\pi(z) \leftrightarrow \pi(z'))$, but that is a contradiction to $\pi(\varphi) \Rightarrow \psi$. Hence, for all $i \neq j$, it follows that $A_i \cap A_j = \emptyset$ and therefore by definition of the A_i's $\pi(X_i \cup Y_i) = \bigcup_{a \in A_i} (\widehat{X}_a \cup \widehat{Y}_a)$. Since π is an injection, $|X_i \cup Y_i| = |\bigcup_{a \in A_i} (\widehat{X}_a \cup \widehat{Y}_a)|$, and since the $\widehat{X}_a$'s and the $\widehat{Y}_a$'s are pairwise disjoint, $|\bigcup_{a \in A_i} (\widehat{X}_a \cup \widehat{Y}_a)| = \sum_{a \in A_i} 2s(a)$. Since $|X_i \cup Y_i| = 2B$ for $1 \leq i \leq m$, we have that $\sum_{a \in A_i} s(a) = B$, which concludes this reduction.

2. In the second case the constraint is $\overline{f} \wedge t \wedge (x \oplus y)$ and we reduce from ISO-IMP($\{x \oplus y\}$), which we just showed to be **NP**-hard. Therefore, we define two formulae φ' and ψ' analogously to φ and ψ in the previous case. Let

$$\varphi \quad = \quad \bigwedge_{i=1}^{m} \left(\bigwedge_{x \in X_i, y \in Y_i} \overline{f} \wedge t \wedge (x \oplus y) \right)$$

and

$$\psi \quad = \quad \bigwedge_{a \in A} \left(\bigwedge_{x \in \widehat{X}_a, y \in \widehat{Y}_a} \overline{f} \wedge t \wedge (x \oplus y) \right).$$

The proof that $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if $\varphi' \overset{\cong}{\Rightarrow} \psi'$ is exactly the same as the proof for case 4 of Lemma 5.4.14.

□

With this we have covered all of the ten cases from Lemma 5.4.8. Thus, we have proved that ISO-IMP(S) is **NP**-hard if the constraint language S does not only contain constants, the identity function, or negation. We still have to show the **coNP** lower bound for constraint languages that are not Schaefer.

Lemma 5.4.16 *Let S be a constraint language. If S is not Schaefer, then* ISO-IMP(S) *is* **coNP***-hard.*

Proof. A closer look at the proof of Claim 19 [BHRV02] shows that it can be used here, too. In that proof Böhler et al. prove **coNP**-hardness for ISO(S) if S is not Schaefer. Their reduction has the property that for all S-formulae φ and ψ, to which the reduction maps, the property $\psi \overset{\cong}{\Rightarrow} \varphi$ always holds. Thus, we have $\varphi \overset{\cong}{\Rightarrow} \psi$ if and only if $\varphi \cong \psi$. □

This concludes the proof of Theorem 5.4.4.

5.4.3 Connection to GI

As we have already seen in Lemma 5.4.3, the isomorphic implication problem is closely related to the subgraph isomorphism problem and also to the constraint isomorphism problem. Both of them in turn are an extension of graph isomorphism. These connections provide an interesting and new way of approaching the unknown complexity of graph isomorphism. We have seen that if a constraint language is Schaefer, then in all but the trivial cases ISO-IMP is equivalent to SGI, because both are **NP**-complete in these cases. Similarly in almost all of these cases, ISO is also equivalent to GI. The only case that does not "fit" is the one, if the constraint language is 2-affine. Then ISO is in **P**, and thus it could only be equivalent to GI, if GI were also in **P**, which is not known.

Let us thus take a closer look at the 2-affine cases.

Proposition 5.4.17 *Let S be a 2-affine constraint language and let G and H be two graphs without isolated vertices. Then there exists a polynomial-time computable reduction function f such that*

$$\langle G, H\rangle \in \text{SGI} \quad \textit{if and only if} \quad f(\langle G, H\rangle) \in \text{ISO-IMP}(S).$$

Proof. This is a simple observation of the fact that ISO-IMP(S) for a constraint language S that is 2-affine and SGI are both **NP**-complete. □

Hence, we know that there exists a function which maps two graphs to two 2-affine constraint formulae. The following theorem states that if such a function exhibits an additional symmetric property, GI is in **P**.

Theorem 5.4.18 *Let S be a 2-affine constraint language and let f be a reduction function from* SGI *to* ISO-IMP(S)*. If f has the property that for all graphs G and H, $f(\langle G, H\rangle) = \langle \varphi, \psi\rangle$ and $f(\langle H, G\rangle) = \langle \psi, \varphi\rangle$, for some constraint formulae φ and ψ, then* GI *is in* **P**.

Proof. Assume such a reduction function f exists. Then the following are equivalent:

- $\langle G, H\rangle \in \mathrm{GI}$
- $\left\{\begin{array}{c} \langle G, H\rangle \in \mathrm{SGI} \\ \text{and} \\ \langle H, G\rangle \in \mathrm{SGI} \end{array}\right\}$
- $\left\{\begin{array}{c} f(\langle G, H\rangle) = \langle \varphi, \psi\rangle \in \text{ISO-IMP}(S) \\ \text{and} \\ f(\langle H, G\rangle) = \langle \psi, \varphi\rangle \in \text{ISO-IMP}(S) \end{array}\right\}$
- $\langle \varphi, \psi\rangle \in \mathrm{ISO}(S)$

Hence, we have a reduction from GI to ISO(S) and since the latter is in **P**, so is GI. □

5.5 Isomorphic Implication Revisited

The main theorem of the previous section, which classifies the isomorphic implication problem depending on the constraint language allowed, is somewhat unsatisfactory, since for the non-Schaefer cases we have a large gap between the upper and lower bounds. We only showed that in these cases isomorphic implication is in $\mathbf{\Theta_2^P}$ as well as **NP**-hard and **coNP**-hard. In this section we therefore try to establish a real trichotomy by closing this gap. We believe that isomorphic implication is $\mathbf{\Theta_2^P}$-hard in all these cases.

Conjecture 5.5.1 *Let S be a constraint language.*

- *If S contains only constants, the identity function, or negation, then* ISO-IMP(S) *is in* **P**.
- *Otherwise, if S is Schaefer, then* ISO-IMP(S) *is* **NP***-complete.*
- *If S is not Schaefer, then* ISO-IMP(S) *is* $\mathbf{\Theta_2^P}$*-complete.*

There are two arguments that support our conjecture. On the one hand it is not uncommon for problems that are **NP**-hard, **coNP**-hard, and in $\mathbf{\Theta_2^P}$ to also be $\mathbf{\Theta_2^P}$-hard and thus $\mathbf{\Theta_2^P}$-complete (see, e. g., [HHR97]). On the other hand we will actually prove $\mathbf{\Theta_2^P}$-hardness in some of the non-Schaefer cases in Theorem 5.5.7 and Theorem 5.5.8.

The following theorem by Wagner provides a basis for our $\mathbf{\Theta_2^P}$-hardness proofs. It shows a way how an **NP** and **coNP** lower bound can be raised to a $\mathbf{\Theta_2^P}$ lower bound.

Theorem 5.5.2 *Let L be a language. If there exists a polynomial-time computable function h such that*

$$|\{i \mid \chi_i \in \mathrm{SAT}\}| \textit{ is odd} \quad \textit{if and only if} \quad h(\chi_1, \ldots, \chi_{2k}) \in L$$

for all $k \geq 1$ and all Boolean formulae $\chi_1, \ldots, \chi_{2k}$ such that $\chi_i \in \mathrm{SAT} \Rightarrow \chi_{i+1} \in \mathrm{SAT}$, then L is $\mathbf{\Theta_2^P}$-hard.

The basic idea behind applying Wagner's theorem can be best seen in the context of *and*-functions, *or*-functions and their ω versions. The following definition originates from [KST93].

Definition 5.5.3 *Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a language. An or-*function *(and-*function*) for L is a function f such that for all $x, y \in \Sigma^*$, we have that $f(x, y) \in L$ if and only if $x \in L$ or $y \in L$ ($x \in L$ and $y \in L$, resp.). An ω-or-function (ω-and-function) is a function f such that for all $x_1, \ldots, x_n \in \Sigma^*$, we have that $f(x_1, \ldots, x_n) \in L$ if and only if $x_i \in L$ for some i ($x_i \in L$ for all i, resp.), $1 \leq i \leq n$.*

Proposition 5.5.4 *Let L be a language. If L is* **NP***-hard,* **coNP***-hard, and one of the following two conditions holds*

- *L has polynomial-time computable and-functions and ω-or-functions or*
- *L has polynomial-time computable or-functions and ω-and-functions,*

then L is $\mathbf{\Theta_2^P}$-hard.

Proof. Since L is **NP**-hard and **coNP**-hard, there exist two functions f and g such that f is a reduction from SAT to L and g is a reduction from $\overline{\mathrm{SAT}}$ to L. For the first case assume that L has a polynomial-time computable *or*-function *or*, and a polynomial-time computable ω-*and*-function *and*. For $k \geq 1$ let $\chi_1, \ldots, \chi_{2k}$ be Boolean formulae such that $\chi_i \in \mathrm{SAT} \Rightarrow \chi_{i+1} \in \mathrm{SAT}$. It is obvious that $|\{i \mid \chi_i \in \mathrm{SAT}\}|$ is odd if and only if $\chi_{2i-1} \notin \mathrm{SAT}$ and $\chi_{2i} \in \mathrm{SAT}$ for some i with $1 \leq i \leq k$. Then define $h(\chi_1, \ldots, \chi_{2k})$ as

$$\mathit{or}(\mathit{and}(f(\chi_1), g(\chi_2)), \mathit{and}(f(\chi_3), g(\chi_4)), \ldots, \mathit{and}(f(\chi_{2k-1}), g(\chi_{2k}))).$$

It is easy to see that h is computable in polynomial time. Surely it is the case that $h(\chi_1, \ldots, \chi_{2k}) \in L$ if and only if there exists an i, $1 \leq i \leq n$, such that $\chi_{2i-1} \notin \mathrm{SAT}$ and $\chi_{2i} \in \mathrm{SAT}$. Then by Theorem 5.5.2 L is $\mathbf{\Theta_2^P}$-hard.

For the other case assume that L has a polynomial-time computable *and*-function and a polynomial-time computable ω-*or*-function. Then $\overline{L}$ has a polynomial-time computable *or*-function and a polynomial-time computable ω-*and*-function. By the argument above, $\overline{L}$ is $\mathbf{\Theta_2^P}$-hard and since $\mathbf{\Theta_2^P}$ is closed under complement, L is also $\mathbf{\Theta_2^P}$-hard. □

Agrawal and Thierauf [AT00] showed that the Boolean isomorphism problem has polynomial-time computable *and*-functions and *or*-functions. Looking into their proof in detail, it is easy to see that the *and*-function are even computable in linear time. It is easy to see that if there exists an *and*-function f_2 computable in linear time, that one can construct an *and*-function f_n of arbitrary arity n by simply repeatedly applying f_2 to the arguments (e. g., $f_3(x, y, z) = f_2(f_2(x, y), z)$ for a ternary *and*-function f_3) such that f_n is computable in polynomial time. This implies that the Boolean isomorphism problem also has ω-*and*-functions. Together with the previous result, this yields an easy corollary. Note, that **coNP**-hardness for the Boolean isomorphism problem follows immediately, since one only has to guess a permutation and an assignment such that two formulae are not equivalent.

Corollary 5.5.5 *If the Boolean isomorphism problem is* **NP***-hard, it is also* $\mathbf{\Theta_2^P}$*-hard.*

Although we have **coNP**-hardness and in contrast to Boolean isomorphism also **NP**-hardness in the case of isomorphic implication, this approach does not seem to work here. There even exists an ω-*and*-function, but the problem is that we are dealing with constraints and we need to connect two formulae with an *or*-function. This unfortunately does not work in the constraint context. Therefore, we need to take a closer look at Wagner's theorem and construct such a reduction function ourselves. Before we can start, we need an additional proposition.

Proposition 5.5.6 *Let S be a constraint language and let φ and ψ two S-formulae. Let further X be the set of variables occurring in φ and let Y be the set of variables occurring in ψ. If $\varphi \overset{\cong}{\Rightarrow} \psi$, $|X| \geq |Y|$, and $X \cap Y = \emptyset$, then there exists a permutation π of $X \cup Y$ such that $\pi(\varphi) \Rightarrow \psi$ and $\pi(Y) \cap Y = \emptyset$.*

Proof. Assume that $\varphi \overset{\cong}{\Rightarrow} \psi$, $|X| \geq |Y|$, and $X \cap Y = \emptyset$. Then there exists a permutation π' of $X \cup Y$ such that $\pi'(\varphi) \Rightarrow \psi$. Further assume that $\pi'(Y) \cap Y \neq \emptyset$ (otherwise we are done). We will describe an iterative method to obtain a permutation π from π' such that $\pi(Y) \cap Y = \emptyset$.

Since $\pi'(Y) \cap Y \neq \emptyset$, there exist $y, y' \in Y$ such that $\pi'(y) = y'$. Since $|X| \geq |Y|$, there also exist $x, x' \in X$ such that $\pi'(x) = x'$. Now we define a permutation ρ such that $\rho(x) = y'$ and $\rho(y) = x'$ and $\rho(z) = \pi'(z)$ for all other $z \in X \cup Y$. We claim that $\rho(\varphi) \Rightarrow \psi$. Since $X \cap Y = \emptyset$, repeated application of this method then yields the desired permutation.

In order to prove the claim, assume for a contradiction that $\rho(\varphi) \not\Rightarrow \psi$. Let Z be the list of variables occurring in $\varphi \wedge \psi$ without x' and y'. Then there exists an assignment $(\alpha, \beta_1, \beta_2)$ to the variables (Z, x', y') with $\alpha \in \{0,1\}^{|Z|}$ and $\beta_1, \beta_2 \in \{0,1\}$ such that $\rho(\varphi)(\alpha, \beta_1, \beta_2) \not\Rightarrow \psi(\alpha, \beta_1, \beta_2)$, that is, $\rho(\varphi)(\alpha, \beta_1, \beta_2) = 1$ and $\psi(\alpha, \beta_1, \beta_2) = 0$. We know that y' does not occur in φ and x' does not occur in ψ. Thus, we can negate their assignments without changing the result: We have $\rho(\varphi)(\alpha, \beta_1, \overline{\beta_2}) = 1$ and $\psi(\alpha, \overline{\beta_1}, \beta_2) = 0$. Since π' is a permutation such that $\pi'(\varphi)(Z, x', y') \Rightarrow \psi(Z, x', y')$ for all assignments to (Z, x', y'), we have that $\pi'(\varphi)(\alpha, \beta_1, \beta_2) = \pi'(\varphi)(\alpha, \overline{\beta_1}, \beta_2) = 0$, because ψ also evaluates to 0 via both assignments. Now y' does not occur in φ and we can again swap the assignment at that position without changing the result and we have that $\pi'(\varphi)(\alpha, \beta_1, \overline{\beta_2}) = \pi'(\varphi)(\alpha, \overline{\beta_1}, \overline{\beta_2}) = 0$. Thus, obviously $\pi'(\varphi)(\alpha, x', y') \equiv 0$. But, by definition of ρ it follows that $\pi'(\varphi)(\alpha, \beta_2, \beta_1) = \rho(\varphi)(\alpha, \beta_1, \beta_2) = 1$, which is a contradiction. □

Now we can state and prove our two main theorems that yield $\mathbf{\Theta_2^P}$-hardness for isomorphic implication with some non-Schaefer constraint languages.

Theorem 5.5.7 *Let S' be a constraint language that is 0-valid, 1-valid, not complementive, and not Schaefer. Let $S = S' \cup \{(x \vee y)\}$. Then* ISO-IMP($S$) *is $\mathbf{\Theta_2^P}$-complete.*

Proof. In the previous section we already showed that ISO-IMP(S) is in $\mathbf{\Theta_2^P}$ (see Lemma 5.4.7); thus we only need to show hardness. Therefore, let $k \geq 1$ and let $\chi_1, \dots, \chi_{2k}$ be formulae such that $\chi_i \in \text{SAT} \Rightarrow \chi_{i+1} \in \text{SAT}$. We will now construct a reduction function h such that

$$|\{i \mid \chi_i \in \text{SAT}\}| \text{ is odd} \quad \text{if and only if} \quad h(\chi_1, \dots, \chi_{2k}) \in \text{ISO-IMP}(S).$$

It is easy to see that $|\{i \mid \chi_i \in \text{SAT}\}|$ is odd if and only if there exists an i, $1 \leq i \leq k$, such that $\psi_{2i-1} \notin \text{SAT}$ and $\psi_{2i} \in \text{SAT}$. We will now prove that such an i exists if and only if $h(\chi_1, \dots, \chi_{2k}) \in \text{ISO-IMP}(S)$.

Since S is not Schaefer, we know from Theorem 5.4.4 that ISO-IMP(S) is **NP**-hard and **coNP**-hard. Thus, there exist polynomial-time many-one reductions from SAT to ISO-IMP(S) and from $\overline{\text{SAT}}$ to ISO-IMP(S). The existence of such reductions will be used to create the function h we are looking for.

Therefore, let f be a polynomial-time computable function such that for all Boolean formulae χ, $f(\chi)$ is an S'-formula with

$$\chi \in \overline{\text{SAT}} \quad \text{if and only if} \quad f(\chi) \overset{\cong}{\Rightarrow} \bigwedge_{1 \leq j, \ell \leq n} x_j \rightarrow x_\ell,$$

where $\{x_1, \dots, x_n\}$ is the set of variables occurring in $f(\chi)$. The function $f(\chi)$ exists, because $\overline{\text{SAT}}$ is reducible to $\overline{\text{CSP}_{\neq 0,1}(S')}$ and that in turn is reducible to ISO-IMP(S'). In the proof of Claim 14 in [BHRV02] it is shown that there exists an S'-formula equivalent to $x \rightarrow y$ and from the proof of Claim 19 in [BHRV02] we know (with similar arguments as in Lemma 5.4.16) that $\overline{\text{CSP}_{\neq 0,1}(S')}$ is reducible to ISO-IMP(S').

Let g be a polynomial-time computable function such that for all Boolean formulae χ, $g(\chi)$ is an $\{x \vee y\}$-formula without duplicates (i. e., there are no clauses $z \vee z'$ with $z = z'$ in $g(\chi)$), and

$$\chi \in \text{SAT} \quad \text{if and only if} \quad g(\chi) \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n-1} y_j \vee y_{j+1},$$

where $\{y_1, \dots, y_n\}$ is the set of variables occurring in $g(\chi)$. The function $g(\chi)$ exists, because SAT is reducible to HAMILTONIAN PATH, which is reducible to ISO-IMP($\{x \vee$

$y\}$) as follows. The input of HAMILTONIAN PATH is a graph G, which we can translate to a constraint formula φ as in Definition 5.4.2. It is then clear that

$$\varphi \quad \overset{\cong}{\Rightarrow} \quad \bigwedge_{j=1}^{n-1} y_j \vee y_{j+1}.$$

Since we need the property that there is an odd i with $\chi_i \notin \text{SAT}$ and $\chi_{i+1} \in \text{SAT}$ if and only if $h(\chi_1, \ldots, \chi_{2k}) \in \text{ISO-IMP}(S)$, we will use the function f from the **coNP**-hardness reduction for formulae with odd index and the function g from the **NP**-hardness reduction for formulae with even index. It is also important that all formulae will have a distinct set of variables. For this we define the following constraint formulae.

For every i, $1 \leq i \leq k$, define ω_i to be the constraint formula $f(\chi_{2i-1})$ with each variable x_j replaced by $x_{i,j}$. Then

$$\chi_{2i-1} \notin \text{SAT} \quad \text{if and only if} \quad \omega_i \overset{\cong}{\Rightarrow} \bigwedge_{1 \leq j,\ell \leq n_i} x_{i,j} \rightarrow x_{i,\ell},$$

where n_i is the n from $f(\chi_{2i-1})$.

Similarly we define for every i, $1 \leq i \leq k$, the constraint formula υ_i to be $g(\chi_{2i})$ with each variable y_j replaced by $y_{i,j}$. Then

$$\chi_{2i} \in \text{SAT} \quad \text{if and only if} \quad \upsilon_i \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n'_i-1} y_{i,j} \vee y_{i,j+1},$$

where n'_i is the n from $g(\chi_{2i})$.

One can easily verify that all formulae

$$\bigwedge_{1 \leq j,\ell \leq n_i} x_{i,j} \rightarrow x_{i,\ell}$$

for $1 \leq i \leq k$ are almost isomorphic. The same holds for the formulae to the right of $\upsilon_i \overset{\cong}{\Rightarrow}$. Only the number n_i respectively n'_i of variables differs. Since we need those formulae to be exactly isomorphic, we pad them with additional clauses. Therefore, let $n = \max\{n_i, n'_i + 2 \mid 1 \leq i \leq k\}$ and define for $1 \leq i \leq k$

$$\widehat{\omega}_i \quad = \quad \omega_i \ \wedge \ (x_{i,j} \rightarrow x_{i,1}) \ \wedge \ \bigwedge_{j=n_i+1}^{n} (x_{i,1} \rightarrow x_{i,j}).$$

Note that $\widehat{\omega}_i$ is an S'-formula, because by the proof of Claim 14 in [BHRV02] there exist S'-formulae equivalent to $x \rightarrow y$. It is immediate that

$$\widehat{\omega}_i \ \overset{\cong}{\Rightarrow} \bigwedge_{1 \leq j,\ell \leq n} x_{i,j} \rightarrow x_{i,\ell} \quad \text{if and only if} \quad \omega_i \ \overset{\cong}{\Rightarrow} \bigwedge_{1 \leq j,\ell \leq n_i} x_{i,j} \rightarrow x_{i,\ell}.$$

Analogously we pad the formulae υ_i for $1 \le i \le k$:

$$\widehat{\upsilon}_i \quad = \quad \upsilon_i \wedge \left(\bigwedge_{j=1}^{n'_i} y_{i,j} \vee y_{i,n'_i+1} \right) \wedge \left(\bigwedge_{j=n'_i+1}^{n-1} y_{i,j} \vee y_{i,j+1} \right).$$

We need to show that the padding did not change the reduction, that is,

$$\widehat{\upsilon}_i \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n-1} (y_{i,j} \vee y_{i,j+1}) \quad \text{if and only if} \quad \upsilon_i \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n'_i-1} (y_{i,j} \vee y_{i,j+1}).$$

For the left-to-right direction we can think about this as the constraint representation of graphs. By construction of $\widehat{\upsilon}_i$ and because $n \ge n'_i + 2$, the vertex n is an endpoint in the graph and the only way to reach n is via the path $n'_{i+1}, n'_{n+2}, \ldots, n$. Thus, any Hamiltonian path in $\widehat{\upsilon}_i$ has to contain that subpath. Since n'_{i+1} is connected with any edge $1, \ldots, n'_i$, there has to be a Hamiltonian path in the subgraph restricted to $\{1, \ldots, n'_i\}$ and therefore in υ_i. The converse follows immediately, since all constraints added to the right of $\upsilon_i \overset{\cong}{\Rightarrow}$ have also been added to υ_i.

Thus, we have the following situation. For all i, $1 \le i \le k$, $\widehat{\omega}_i$ is an S'-formula such that

$$\chi_{2i-1} \notin \mathrm{SAT} \quad \text{if and only if} \quad \widehat{\omega}_i \overset{\cong}{\Rightarrow} \bigwedge_{1 \le j,\ell \le n} (x_{i,j} \rightarrow x_{i,\ell})$$

and $\widehat{\upsilon}_i$ is a $\{x \vee y\}$-formula without duplicates such that

$$\chi_{2i} \in \mathrm{SAT} \quad \text{if and only if} \quad \widehat{\upsilon}_i \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n-1} (y_{i,j} \vee y_{i,j+1}).$$

Now we are finally in a position to define our reduction function

$$h(\chi_1, \ldots, \chi_{2k}) \quad = \quad \langle \varphi, \psi \rangle,$$

where

$$\varphi \quad = \quad \bigwedge_{i=1}^{k} \left(\widehat{\omega}_i \wedge \widehat{\upsilon}_i \wedge \bigwedge_{1 \le j,\ell \le n} x_{i,j} \rightarrow y_{i,\ell} \right)$$

and

$$\psi \quad = \quad \bigwedge_{1 \le j,\ell \le n} (x_j \rightarrow x_\ell) \wedge \bigwedge_{j=1}^{n-1} (y_j \vee y_{j+1}) \wedge \bigwedge_{1 \le j,\ell \le n} (x_j \rightarrow y_\ell).$$

It is obvious that h is computable in polynomial time and also that φ and ψ are S-formulae, because $\widehat{\omega}_i$ and $\widehat{\upsilon}_i$ are S-formulae and we can express the formulae $x \vee y$ and $x \rightarrow y$.

We still need to show that there exists an i such that $\chi_{2i-1} \notin \mathrm{SAT}$ and $\chi_{2i} \in \mathrm{SAT}$ if and only if $h(\chi_1, \ldots, \chi_{2k}) \in \text{ISO-IMP}(S)$. By construction of the formulae $\widehat{\omega}_i$ and $\widehat{\upsilon}_i$ and the function h, this is equivalent to showing that there exists an i such that

$$\widehat{\omega}_i \overset{\cong}{\Rightarrow} \bigwedge_{1\le j,\ell\le n} (x_{i,j} \rightarrow x_{i,\ell}) \text{ and } \widehat{\upsilon}_i \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n-1} (y_{i,j} \vee y_{i,j+1}) \quad \text{if and only if} \quad \varphi \overset{\cong}{\Rightarrow} \psi.$$

For the left-to-right direction let i_0 be such that

$$\widehat{\omega}_{i_0} \overset{\cong}{\Rightarrow} \bigwedge_{1\le j,\ell\le n} (x_{i_0,j} \rightarrow x_{i_0,\ell}) \quad \text{and} \quad \widehat{\upsilon}_{i_0} \overset{\cong}{\Rightarrow} \bigwedge_{j=1}^{n-1} (y_{i_0,j} \vee y_{i_0,j+1}).$$

Further let π_x and π_y be two permutations on $\{x_{i_0,1}, \ldots, x_{i_0,n}\}$ and $\{y_{i_0,1}, \ldots, y_{i_0,n}\}$, respectively, such that

$$\pi_x(\widehat{\omega}_{i_0}) \Rightarrow \bigwedge_{1\le j,\ell\le n} (x_{i_0,j} \rightarrow x_{i_0,\ell}) \quad \text{and} \quad \pi_y(\widehat{\upsilon}_{i_0}) \Rightarrow \bigwedge_{j=1}^{n-1} (y_{i_0,j} \vee y_{i_0,j+1}).$$

We now define a permutation π on the variables occurring in $\varphi \wedge \psi$ such that $\pi(x_{i_0,j}) = x_\ell$ if $\pi_x(x_{i_0,j}) = x_{i_0,\ell}$ and $\pi(y_{i_0,j}) = y_\ell$ if $\pi_y(y_{i_0,j}) = y_{i_0,\ell}$, for all $1 \le j, \ell \le n$. It is then immediate that

$$\pi(\widehat{\omega}_{i_0}) \Rightarrow \bigwedge_{1\le j,\ell\le n} (x_j \rightarrow x_\ell),$$

$$\pi(\widehat{\upsilon}_{i_0}) \Rightarrow \bigwedge_{j=1}^{n-1} (y_j \vee y_{j+1}),$$

and

$$\pi\left(\bigwedge_{1\le j,\ell\le n} (x_{i_0,j} \rightarrow y_{i_0,\ell})\right) \Rightarrow \bigwedge_{1\le j,\ell\le n} (x_j \rightarrow y_\ell).$$

This implies directly that $\varphi \overset{\cong}{\Rightarrow} \psi$ via π.

For the other direction assume that $\varphi \overset{\cong}{\Rightarrow} \psi$. Then by Proposition 5.5.6 there exists a permutation π of the variables occurring in $\varphi \wedge \psi$ such that $\pi(\varphi) \Rightarrow \psi$ and for all j with $1 \le j \le n$, we have that $\pi(x_j)$ and $\pi(y_j)$ do not occur in ψ, because x_j and y_j occur in ψ.

In the following we will show that π is a reasonable permutation in that it maps all variables to their respective counterparts. First, we will show that π cannot map a y-variable to x_j for all $1 \le j \le n$. For a contradiction assume that $\pi(y_{i,\ell}) = x_j$. It is clear that φ is satisfied by the assignment that sets all y-variables to 1 and all x-variables to 0. It remains satisfied when we swap the value of just one y-variable in φ, because the *or*-clauses in $\widehat{\upsilon}_i$ are without duplicates and any $x \rightarrow y$ clause is always satisfied if x is set to 0. Then $\pi(\varphi)$ is satisfied by the assignment that sets $\pi(y)$ to 1 and $\pi(x)$ to 0 for all y-

and x-variables. Similarly $\pi(\varphi)$ remains satisfied when changing the value of $\pi(y_{i,\ell})$ from 1 to 0. But, this is a contradiction, because $\pi(y_{i,\ell}) = x_j$ and because of

$$\bigwedge_{1\le j,\ell\le n} (x_j \rightarrow x_\ell)$$

all x-variables have to take the same value in any satisfying assignment. Thus, changing the value of x_j in a satisfying assignment for ψ will always make ψ false. Hence, only x-variables can be mapped to x-variables. Now let $x_1 = \pi(x_{i_0,j_0})$ for some i_0 with $1 \le i_0 \le k$ and some j_0 with $1 \le j_0 \le n$. Assume that $x_j = \pi(x_{i,\ell})$ for some $x_{i,\ell}$. Because $\pi(\varphi) \Rightarrow (x_1 \leftrightarrow x_j)$, also $\varphi \Rightarrow (x_{i_0,j_0} \leftrightarrow x_{i,\ell})$. Thus, $i = i_0$ and therefore $\pi(\{x_{i_0,\ell} \mid 1 \le \ell \le n\}) = \{x_\ell \mid 1 \le \ell \le n\}$.

Next, we will see which variables can be mapped to y-variables. Assume therefor that $\pi(z) = y_j$ for some variable z. Because $\pi(\varphi) \Rightarrow (x_1 \rightarrow y_j)$, we have that $\varphi \Rightarrow (x_{i_0,j_0} \rightarrow z)$. Thus, z can be either an x-variable of the form $x_{i_0,\ell}$ or a y-variable of the form $y_{i_0,\ell}$. Since we showed above that all $x_{i_0,\ell}$-variables are mapped to x_ℓ-variables, only $z = y_{i_0,\ell}$ remains possible. Hence, $\pi(\{y_{i_0,\ell} \mid 1 \le \ell \le n\}) = \{y_\ell \mid 1 \le \ell \le n\}$ and thus π is a reasonable permutation.

Let α be the partial assignment that sets all y-variables to 1 and all x-variables except those in $\{x_\ell \mid 1 \le \ell \le n\}$ to 0. Then it is easy to see that $\pi(\varphi)[\alpha]$ is equivalent to $\pi(\widehat{\omega}_{i_0})$ and $\psi[\alpha]$ is equivalent to

$$\bigwedge_{1\le j,\ell\le n} (x_j \rightarrow x_\ell).$$

Since $\pi(\varphi) \Rightarrow \psi$, we also have that $\pi(\varphi)[\alpha] \Rightarrow \psi[\alpha]$, that is,

$$\pi(\widehat{\omega}_{i_0}) \quad \Rightarrow \quad \bigwedge_{1\le j,\ell\le n} (x_j \rightarrow x_\ell)$$

and thus

$$\widehat{\omega}_{i_0} \quad \overset{\cong}{\Rightarrow} \quad \bigwedge_{1\le j,\ell\le n} (x_{i_0,j} \rightarrow x_{i_0,\ell}).$$

Similarly let β be the partial assignment that sets all x-variables to 0 and all y-variables except those in $\{y_\ell \mid 1 \le \ell \le n\}$ to 1. Then $\pi(\varphi)[\beta]$ is equivalent to $\pi(\widehat{\upsilon}_{i_0})$ and $\psi[\beta]$ is equivalent to

$$\bigwedge_{j=1}^{n-1} (y_j \vee y_{j+1}).$$

Since $\pi(\varphi) \Rightarrow \psi$, we also have that $\pi(\varphi)[\beta] \Rightarrow \psi[\beta]$, that is,

$$\pi(\widehat{\upsilon}_{i_0}) \quad \Rightarrow \quad \bigwedge_{j=1}^{n-1} (y_j \vee y_{j+1})$$

and thus

$$\widehat{\upsilon}_{i_0} \quad \overset{\cong}{\Rightarrow} \quad \bigwedge_{j=1}^{n-1} (y_{i_0,j} \vee y_{i_0,j+1}).$$

This concludes the proof of this theorem. □

A similar construction yields $\mathbf{\Theta_2^P}$-hardness also for the following case.

Theorem 5.5.8 *Let S' be a constraint language that is 0-valid, 1-valid, not complementive, and not Schaefer. Let $S = S' \cup \{(x \oplus y \oplus z)\}$. Then* ISO-IMP($S$) *is $\mathbf{\Theta_2^P}$-complete.*

Proof. This can be proved analogously to the previous theorem. We will again construct a polynomial-time computable function h such that

$$|\{i \mid \chi_i \in \mathrm{SAT}\}| \text{ is odd} \quad \text{if and only if} \quad h(\chi_1, \ldots, \chi_{2k}) \in \text{ISO-IMP}(S),$$

where $\chi_1, \ldots, \chi_{2k}$ are formulae, for $k \geq 1$, such that $\chi_i \in \mathrm{SAT} \Rightarrow \chi_{i+1} \in \mathrm{SAT}$.

Let f be a polynomial-time computable function defined as in the proof of the previous theorem.

Let g be a polynomial-time computable function such that for all χ, $g(\chi)$ is an $\{x \oplus y \oplus z\}$-formula and

$$\chi \in \mathrm{SAT} \quad \text{if and only if} \quad g(\chi) \overset{\cong}{\Rightarrow} (y_1 \oplus y_n \oplus z_n) \wedge \bigwedge_{j=1}^{n-1} (y_j \oplus y_{j+1} \oplus z_j),$$

where $\{y_1, \ldots, y_n\}$ is the set of variables occurring at least twice in $g(\chi)$. The function $g(\chi)$ exists, because SAT is reducible to VERTEX COVER, which is reducible to HAMILTONIAN CYCLE, which in turn is reducible to ISO-IMP($\{x \oplus y \oplus z\}$) as we will show below. However, we take a different version of HAMILTONIAN CYCLE, where there is one distinguished edge, which always belongs to a Hamiltonian cycle and there are no direct triangles in the graph, that is, there is no clique of size three. Since in the reduction from VERTEX COVER to HAMILTONIAN CYCLE no direct triangle appears and there is such a distinguished edge (see [GJ79]), this is no restriction. To serve as the translation from graphs to formulae we take the one used in the proof of Theorem 25 in [BHRV02]: For $G = (V, E)$ a connected graph on Vertices $V = \{1, \ldots, n\}$, let $t(G)$ be the formula consisting of the clauses from $\{y_i \oplus y_j \oplus z_k \mid e_k = \{i, j\} \in E\} \cup \{y_i \oplus v_i \oplus v_i' \mid i \in V\} \cup \{z_i \oplus z_j \oplus z_k \mid e_i, e_j, \text{ and } e_k \text{ form a triangle in } G\}$. For future references, we call the clauses from $\{y_i \oplus y_j \oplus z_k \mid e_k = \{i, j\} \in E\}$ y-clauses, the ones from $\{y_i \oplus v_i \oplus v_i' \mid i \in V\}$ will be called v-clauses, and the others are the z-clauses. The z-clauses are already implied by the y-clauses. They are added to make sure the obtained constraint formula is maximal in its clauses, that is, every constraint implied by it is already a clause of the formula. The maximality has also been proved in [BHRV02]. Let

$$\xi \quad = \quad (y_1 \oplus y_n \oplus z_n) \wedge \bigwedge_{j=1}^{n-1} (y_j \oplus y_{j+1} \oplus z_j).$$

We still need to show that a graph G has a HAMILTONIAN CYCLE if and only if $t(G) \overset{\cong}{\Rightarrow} \xi$. For the one direction it is clear that if G has a HAMILTONIAN CYCLE, there has to be a conjunction of y-clauses isomorphic to ξ. For the converse assume that $t(G) \overset{\cong}{\Rightarrow} \xi$, but G does not have a HAMILTONIAN CYCLE. Since $t(G)$ is maximal, there has to be a permutation π such that all clauses of ξ are also clauses of $\pi(t(G))$. In this case, for all v- and v'-variables, $\pi(v)$ and $\pi(v')$ cannot map to a y-variable, because each of those occurs only once in $t(G)$. Since we further know that there are no direct triangles in G, $t(G)$ does not contain any z-clauses. So, $\pi(z)$ also cannot map to a y-variable for any z-variable for the same reason. Thus, only y-variables can map to y-variables and accordingly only z-variables are mapped to z-variables by π (these are the only other variables in the y-clauses). Therefore, there exists a permutation π' on the vertices of G obtained from π in the obvious way (i. e., $\pi'(i) = j$ if and only if $\pi(x_i) = x_j$) such that $\pi'(G)$ has a HAMILTONIAN CYCLE. But then also, G has a HAMILTONIAN CYCLE and that is a contradiction to our assumption. Thus, G has a HAMILTONIAN CYCLE if and only if $t(G) \overset{\cong}{\Rightarrow} \xi$.

The formulae ω_i and υ_i are again defined analogously. For the padding of ω_i and υ_i we define n as in the previous proof. Then the formula $\widehat{\omega}_i$ is defined exactly the same way. For $\widehat{\upsilon}_i$ let Υ_i be the set of all clauses of υ_i, that is, $\upsilon_i = \bigwedge_{C\in\Upsilon_i} C$, and let $C'_i \in \Upsilon_i$ with $C'_i = y_{i,j_0} \oplus y_{i,j_1} \oplus z_{i,j_2}$ be the clause representing the distinguished edge, which is known to be in any Hamiltonian cycle. Then we will define a new formula, where we delete the distinguished edge and introduce at its place a new path, thereby increasing the number of edges respectively variables. Formally, we have for every i with $1 \le i \le k$

$$\begin{aligned}\widehat{\upsilon}_i \quad = \quad & \bigwedge_{C\in\Upsilon_i\setminus\{C'_i\}} C \;\wedge \bigwedge_{j=n'_i+1}^{n-1} (y_{i,j} \oplus y_{i,j+1} \oplus z_{i,j}) \\ & \qquad \wedge\; (y_{i,j_0} \oplus y_{i,n'_i+1} \oplus z_{i,j_2}) \;\wedge\; (y_{i,n} \oplus y_{i,j_1} \oplus z_{i,n}),\end{aligned}$$

where n'_i is the n from $g(\chi_{2i})$. The formula $\widehat{\upsilon}$ is already maximal: Since in its construction no new triangles are introduced, we do not need to add any z-clauses to $\widehat{\upsilon}_i$. Furthermore we know that any Hamiltonian cycle in υ_i has to go through the edge represented by C'_i, and hence any Hamiltonian cycle in $\widehat{\upsilon}_i$ has to go through the newly introduced edges. Then it is immediate, that

$$\widehat{\upsilon}_i \quad \overset{\cong}{\Rightarrow} \quad (y_{i,n} \oplus y_{i,1} \oplus z_{i,n}) \wedge \bigwedge_{j=1}^{n-1} (y_{i,j} \oplus y_{i,j+1} \oplus z_{i,j})$$

if and only if

$$\upsilon_i \quad \overset{\cong}{\Rightarrow} \quad (y_{i,n'_i} \oplus y_{i,1} \oplus z_{i,n'_i}) \wedge \bigwedge_{j=1}^{n'_i-1} (y_{i,j} \oplus y_{i,j+1} \oplus z_{i,j}).$$

Now we can define the reduction function h in an analogous way to the previous proof:

$$h(\chi_1, \ldots, \chi_{2k}) \quad = \quad \langle \varphi, \psi \rangle,$$

where

$$\varphi \quad = \quad \bigwedge_{i=1}^{k} \left(\widehat{\omega}_i \wedge \widehat{\upsilon}_i \wedge \bigwedge_{1 \le j,\ell \le n} x_{i,j} \rightarrow y_{i,\ell} \right)$$

and

$$\psi \quad = \quad \bigwedge_{1 \le j,\ell \le n} (x_j \rightarrow x_\ell) \;\wedge\; (y_n \oplus y_1 \oplus z_n) \wedge \bigwedge_{j=1}^{n-1} (y_j \oplus y_{j+1} \oplus z_j) \;\wedge\; \bigwedge_{1 \le j,\ell \le n} (x_j \rightarrow y_\ell).$$

As in the previous proof, h is clearly computable in polynomial time and φ and ψ are S-formulae. It remains to show that there exists an i such that

$$\widehat{\omega}_i \quad \overset{\cong}{\Rightarrow} \quad \bigwedge_{1 \le i,\ell n} (x_{i,j} \rightarrow x_{i,\ell})$$

and

$$\widehat{\upsilon}_i \quad \overset{\cong}{\Rightarrow} \quad (y_{i,n} \oplus y_{i,1} \oplus z_{i,n}) \wedge \bigwedge_{j=1}^{n-1} (y_{i,j} \oplus y_{i,j+1} \oplus z_{i,j})$$

if and only if

$$\varphi \quad \overset{\cong}{\Rightarrow} \quad \psi.$$

The left-to-right direction carries over directly. For the converse, we need to change some arguments. Suppose that $\varphi \overset{\cong}{\Rightarrow} \psi$. By Proposition 5.5.6 we know that there exists a permutation π of the variables occurring in $\varphi \wedge \psi$ such that $\pi(\varphi) \Rightarrow \psi$ and such that for all j with $1 \le j \le n$, we have that $\pi(x_j)$, $\pi(y_j)$, and $\pi(z_j)$ do not occur in ψ.

In the following we will show that π is a reasonable permutation. First, we will show that for all $1 \le j \le n$, the permutation π cannot map a v-variable to x_j. For a contradiction assume that $\pi(v_{i',\ell}) = x_j$. Then φ is satisfied by an assignment that sets all x-variables to 0 and all other variables to 1. If we now change the value of $v_{i',\ell}$ and the corresponding $v'_{i',\ell}$ to 0, then φ remains satisfied, because $v_{i',\ell}$ occurs only once in φ, namely in a v-clause together with $v'_{i',\ell}$. Accordingly also $\pi(\varphi)$ is satisfied by the assignment that sets $\pi(x)$ to 0 for all x-variables and $\pi(y)$, $\pi(z)$, $\pi(v)$, and $\pi(v')$ to 1 for all other variables. Also $\pi(\varphi)$ remains satisfied if in this assignment we change the value of $\pi(v_{i',\ell})$ and $\pi(v'_{i',\ell})$ from 1 to 0. But, this is a contradiction, since $\pi(v_{i',\ell}) = x_j$ and changing the value of x_j in a satisfying assignment for ψ will always make ψ false, even if one additional variable $\pi(v'_{i',\ell})$ is swapped. Exactly the same argument can be used to show that for all $1 \le j \le n$, the permutation π cannot map a v'-variable to x_j. With a similar argument we will also show that for all $1 \le j \le n$, the permutation π cannot map a z-variable to x_j. For a contradiction suppose that $\pi(z_{i',\ell}) = x_j$. Then $z_{i',\ell}$ occurs by the construction of the $\widehat{\upsilon}_i$ formulae exactly once in φ, namely in a y-clause. In the same clause are also two y-variables. Let one of them be $y_{i',k}$. Then $y_{i',k}$ appears exactly once in a v-clause, together with $v_{i',k}$, and m times in a y-clause, with $1 \le m \le n-2$, for the following reason. The y-clauses represent the edges in the graph and since we argued that there are no direct triangles, no vertex is directly connected to all other vertices. Thus, there can be at most $n-2$ different y-clauses containing $y_{i',k}$. Now let $z_{i',\ell_1},\dots,z_{i',\ell_m}$ be the z-variables from those y-clauses. Obviously $z_{i',\ell}\in\{z_{i',\ell_1},\dots,z_{i',\ell_m}\}$. Now φ is satisfied by an assignment that sets all x-variables to 0 and all other variables to 1. If we change the value of all variables $z_{i',\ell_1},\dots,z_{i',\ell_m}$ and $y_{i',k}$ from 1 to 0, φ remains satisfied. Accordingly also $\pi(\varphi)$ is satisfied by the assignment that sets $\pi(x)$ to 0 for all x-variables and $\pi(y)$, $\pi(z)$, $\pi(v)$, and $\pi(v')$ to 1 for all other variables. The formula $\pi(\varphi)$ remains satisfied if in this assignment we change the value of $\pi(z_{i',\ell_1}),\dots,\pi(z_{i',\ell_m})$ and $\pi(y_{i',k})$ from 1 to 0. Since $\pi(z_{i',\ell})=x_j$ and all x-variables have to take the same value, this implies that $\{\pi(z_{i',\ell_1}),\dots,\pi(z_{i',\ell_m}),\pi(y_{i',k})\}\supseteq\{x_1,\dots,x_n\}$. However, this is a contradiction, since $m+1<n$ and all variables are distinct. Finally, for $1\le j\le n$, the permutation π cannot map a y-variable to x_j. Since every y-variable always occurs with at least one v-, v'-, or z-variable in the same clause, we can use the above argument. Change the value of the y-variables and exactly one of the v-, v'-, or z-variables in every clause, where the y-variable occurs. This keeps the truth value of $\pi(\varphi)$, but makes ψ false, since only the y-variable is mapped to an x-variable.

Let i_0 and j_0 be such that $\pi(x_{i_0,j_0})=x_1$. Now suppose that $\pi(x_{i',\ell})=x_j$. Since $\pi(\varphi)\Rightarrow(x_1\leftrightarrow x_j)$, we have that $\varphi\Rightarrow(x_{i_0,j_0}\leftrightarrow x_{i',\ell})$. It follows that $i'=i_0$ and thus $\pi(\{x_{i_0,\ell}\mid 1\le\ell\le n\})=\{x_\ell\mid 1\le\ell\le n\}$.

Next suppose that $\pi(z)=y_j$ for some variable z. Since $\pi(\varphi)\Rightarrow(x_1\rightarrow y_j)$, we have that $\varphi\Rightarrow(x_{i_0,j_0}\rightarrow z)$. It follows that $z=x_{i_0,\ell}$ or $z=y_{i_0,\ell}$. Since $\pi(\{x_{i_0,\ell}\mid 1\le \ell\le n\})=\{x_\ell\mid 1\le\ell\le n\}$, the only possibility is $z=y_{i_0,\ell}$ and it follows that $\pi(\{y_{i_0,\ell}\mid 1\le\ell\le n\})=\{y_\ell\mid 1\le\ell\le n\}$.

Finally, let j_1 and j_2 be such that $\pi(y_{i_0,j_1})=y_j$ and $\pi(y_{i_0,j_2})=y_{j+1}$. Suppose that $\pi(z)=z_j$ for some variable z. Since $\pi(\varphi)\Rightarrow(y_j\oplus y_{j+1}\oplus z_j)$, we have that $\varphi\Rightarrow(y_{i_0,j_1}\oplus y_{i_0,j_2}\oplus z)$. It follows that $z=z_{i_0,\ell}$ and thus $\pi(\{z_{i_0,\ell}\mid \ell\in\mathbb{N}\})\supseteq\{z_\ell\mid 1\le\ell\le n\}$. Note that we only consider variables $z_{i_0,\ell}$ that actually occur in the formula.

Let α be the partial assignment that sets all y-variables, all z-variables, all v-variables, and all v'-variables to 1, and all x-variables except those in $\{x_\ell\mid 1\le\ell\le n\}$ to 0. Then $\pi(\varphi)[\alpha]$ is equivalent to $\pi(\widehat{\omega}_{i_0})$ and $\psi[\alpha]$ is equivalent to

$$\bigwedge_{1\le j,\ell\le n}(x_j\rightarrow x_\ell).$$

Since $\pi(\varphi)\Rightarrow\psi$, also $\pi(\varphi)[\alpha]\Rightarrow\psi[\alpha]$, that is,

$$\pi(\widehat{\omega}_{i_0})\quad\Rightarrow\quad\bigwedge_{1\le j,\ell\le n}(x_j\rightarrow x_\ell),$$

and thus

$$\widehat{\omega}_{i_0}\quad\overset{\cong}{\Rightarrow}\quad\bigwedge_{1\le j,\ell\le n}(x_{i_0,j}\rightarrow x_{i_0,\ell}).$$

Let β be the partial assignment that sets all x-variables and all v-variables to 0, all v'-variables to 1, and all y-variables and all z-variables except those in $\{y_\ell,z_\ell\mid 1\le\ell\le n\}$

to 1. Then $\pi(\varphi)[\beta]$ is equivalent to $\pi(\widehat{\upsilon}_{i_0})$ and $\psi[\beta]$ is equivalent to

$$\bigwedge_{j=1}^{n-1}(y_j \oplus y_{j+1} \oplus z_j).$$

Since $\pi(\varphi) \Rightarrow \psi$, also $\pi(\varphi)[\beta] \Rightarrow \psi[\beta]$. If follows that

$$\pi(\widehat{\upsilon}_{i_0}) \quad \Rightarrow \quad (y_n \oplus y_1 \oplus z_n) \wedge \bigwedge_{j=1}^{n-1}(y_j \oplus y_{j+1} \oplus z_j),$$

and thus

$$\widehat{\upsilon}_{i_0} \quad \overset{\cong}{\Rightarrow} \quad (y_{i_0,n} \oplus y_{i_0,1} \oplus z_{i_0,n}) \wedge \bigwedge_{j=1}^{n-1}(y_{i_0,j} \oplus y_{i_0,j+1} \oplus z_{i_0,j}).$$

This completes the proof of this theorem. □

5.6 Conclusion

In this chapter we have been able to extend several already known results. On the one hand we classified the equivalence problem for arbitrary constraints. This is either in **P** or **coNP**-complete. As a one-sided version of equivalence, we also obtained a similar classification for the implication problem for arbitrary constraints, which is also either in **P** or **coNP**-complete. In the main part of this chapter we were able to get a classification for the isomorphic implication problem. The Schaefer cases are either in **P** or **NP**-complete. The exact complexity for the non-Schaefer cases is not known, yet. We were able to prove $\boldsymbol{\Theta}_{\mathbf{2}}^{\mathbf{P}}$-completeness for some cases, but there are still open cases, where we were only able to prove an **NP** and **coNP** lower bound.

The most important open question is thus, whether our conjecture holds for all non-Schaefer cases. Another open question is the classification of isomorphic implication for two constraint languages. Although we have some results in this direction (e. g., the problem is in **P**, if one of the constraint languages contains only constant relations or both of them contain only the identity and negation; it is in **NP**, if the first constraint language is Schaefer; it is **coNP**-hard, if the first constraint language is not Schaefer and the second one contains only constant relations), we are still far away from a complete classification.

Finally, there is of course the open question, whether graph isomorphism is in **P**. Though we were not able to answer this question, we hope that our new approach allows for some new insights into this famous open problem.

Bibliography

[ABI$^+$05] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 71–82, 2005.

[AG00] C. Alvarez and R. Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Computational Complexity*, 9(2):123–145, 2000.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995.

[AT00] M. Agrawal and T. Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, 2000.

[BBC$^+$05] M. Bauland, E. Böhler, N. Creignou, S. Reith, H. Schnoor, and H. Vollmer. Quantified constraints: The complexity of decision and counting for bounded alternation. Technical Report TR05-024, ECCC Reports, 2005.

[BCC$^+$04] M. Bauland, P. Chapdelaine, N. Creignou, M. Hermann, and H. Vollmer. An algebraic approach to the complexity of generalized conjunctive queries. In *Proceedings 7th International Conference on Theory and Applications of Satisfiability Testing*, volume 3542 of *Lecture Notes in Computer Science*, pages 30–45, Berlin Heidelberg, 2004. Springer Verlag.

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *SIGACT News*, 34(4):38–52, 2003.

[BCRV04] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM-SIGACT Newsletter*, 35(1):22–35, 2004.

[BH91] S. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.

[BH05] M. Bauland and E. Hemaspaandra. Isomorphic implication. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 119–130, 2005.

[BHRV02] E. Böhler, E. Hemaspaandra, S. Reith, and H. Vollmer. Equivalence and isomorphism for Boolean constraint satisfaction. In *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*, pages 412–426, Berlin Heidelberg, 2002. Springer Verlag.

[BHRV04] E. Böhler, E. Hemaspaandra, S. Reith, and H. Vollmer. The complexity of Boolean constraint isomorphism. In *Proceedings 21st Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 164–175, Berlin Heidelberg, 2004. Springer Verlag.

[BRSV05] E. Böhler, S. Reith, H. Schnoor, and H. Vollmer. Bases for Boolean co-clones. *Information Processing Letters*, 96:59–66, 2005.

[CH96] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125:1–12, 1996.

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* Monographs on Discrete Applied Mathematics. SIAM, 2001.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings 3rd Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.

[Dal97] V. Dalmau. Some dichotomy theorems on constant-free quantified boolean formulas, 1997.

[Dal00] V. Dalmau. *Computational Complexity of Problems over Generalized Formulas.* PhD thesis, Department de Llenguatges i Sistemes Informàtica, Universitat Politécnica de Catalunya, 2000.

[DHK00] A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. In *25th International Symposium on Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 323–332. Springer-Verlag, 2000.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, New York, 1995.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Hem04] E. Hemaspaandra. Dichotomy theorems for alternation-bounded quantified boolean formulas. *CoRR*, cs.CC/0406006, 2004.

[HHR97] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Raising NP lower bounds to parallel NP lower bounds. *SIGACT News*, 28(2):2–13, 1997.

[HO02] L. A. Hemaspaandra and M. Ogihara. *The complexity theory companion.* Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[Imm99] N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer Verlag, New York, 1999.

[JCG97] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

[JGK70] S. W. Jablonski, G. P. Gawrilow, and W. B. Kudrjawzew. *Boolesche Funktionen und Postsche Klassen*, volume 6 of *Logik und Grundlagen der Mathematik.* Friedr. Vieweg & Sohn and C. F. Winter'sche Verlagsbuchhandlung, Braunschweig and Basel, 1970.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: its Structural Complexity.* Progress in Theoretical Computer Science. Birkhäuser, 1993.

[Lad75] R. Ladner. On the structure of polynomial-time reducibility. *Journal of the ACM*, 22:155–171, 1975.

[Len02] M. Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[Lev73] L. A. Levin. Universal sorting problems. *Problemi Peredachi Informatsii*, 9(3):115–116, 1973. English translation: *Problems of Information Transmission*, 9(3):265–266.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings 13th Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society Press, 1972.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[Pip97] N. Pippenger. *Theories of computability.* Cambridge University Press, New York, NY, USA, 1997.

[PK79] R. Pöschel and L. A. Kalužnin. *Funktionen- und Relationenalgebren.* Deutscher Verlag der Wissenschaften, Berlin, 1979.

[Pos41] E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

[Pös01] R. Pöschel. Galois connection for operations and relations. Technical Report MATH-LA-8-2001, Technische Universität Dresden, 2001.

[Rei01] S. Reith. *Generalized Satisfiability Problems.* PhD thesis, Fachbereich Mathematik und Informatik, Universität Würzburg, 2001.

[Rei05] O. Reingold. Undirected st-connectivity in log-space. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 376–385, New York, NY, USA, 2005. ACM Press.

[Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978.

[Sip97] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, Boston, 1997.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tod91] S. Toda. *Computational Complexity of Counting Complexity Classes.* PhD thesis, Tokyo Institute of Technology, Department of Computer Science, Tokyo, 1991.

[TW92] S. Toda and O. Watanabe. Polynomial time 1-Turing reductions from #PH to #P. *Theoretical Computer Science*, 100:205–221, 1992.

[Val79a] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val79b] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):411–421, 1979.

[Vol94] H. Vollmer. *Komplexitätsklassen von Funktionen.* PhD thesis, Universität Würzburg, Institut für Informatik, Germany, 1994.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

[Wid95] J. Widom. Research problems in data warehousing. In *4th International Conference on Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, 1995.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

Index