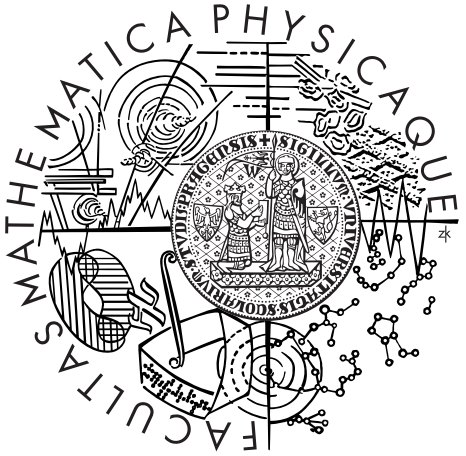
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Štefan Gurský

Special Classes of Boolean Functions with Respect to the Complexity of their Minimization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the doctoral thesis:	doc. RNDr. Ondřej Čepek, Ph.D.
Study programme:	Computer Science
Specialization:	4I1 Theoretical Computer Science

Prague 2014

I would like to thank my supervisor Ondřej Čepek for guiding me in writing this work and also to other attendants of the Seminar on Boolean functions, namely Petr Kučera, Martin Babka, Tomáš Balyo and Václav Vlček for their participation on joint research.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, June 11, 2014

Název práce: Speciální třídy Booleovských funkcí s ohledem na složitost jejich minimalizace.
Autor: Štefan Gurský
Katedra: Katedra teoretické informatiky a matematické logiky
Vedoucí disertační práce: doc. RNDr. Ondřej Čepek, Ph.D., KTIML
Abstrakt: V práci zkoumáme boolovské funkce ze tří různých hledisek. Zaprvé zkoumáme složitost minimalizace formulí z několika tříd s polynomiálně řešitelným SATem a uvádíme postačující podmínky pro třídy CNF aby se jejich minimalizace přesunula níž alespoň o jeden stupeň v polynomiální hierarchii. Zadruhé zkoumáme třídu formulí zvaných matched (párované) pro které je SAT triviální, ale minimalizace zůstává Σ_2^p úplná. Dokazujeme, že pro každou párovanou CNF existuje alespoň jedna primární a iredundantní CNF s ní ekvivalentní, která je také párovaná. Použitím tohoto tvrzení ukazujeme hlavní výsledek této části a to, že pro každou párovanou CNF všechny s ní ekvivalentní CNF mající minimální možný počet klauzulí jsou taky párované. Zatřetí se věnujeme vlastnosti propagation completeness (úplnosti pro propagaci) – CNF je úplná pro propagaci, pokud pro každé částečné dosazení jsou všechny vynucené literály odvoditelné jednotkovou propagací. Každá CNF sa dá rozšířit na úplnou pro propagaci přidáváním empowering (zesilujících) implikátů. Hlavním výsledkem této části je důkaz coNP úplnosti rozpoznávání formulí úplných pro propagaci. Dále ukazujeme, že existují formule, ke kterým je nutno přidat exponenciálně mnoho zesilujících implikátů, aby se staly úplnými pro propagaci.
Klíčová slova: Booleovské funkce, Splnitelnost (SAT), Booleovská minimalizace, Matched formule, Propagation completeness, Empowering implikáty

Title: Special Classes of Boolean Functions with Respect to the Complexity of their Minimization.
Author: Štefan Gurský
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: doc. RNDr. Ondřej Čepek, Ph.D.
Abstract: In this thesis we study Boolean functions from three different perspectives. First, we study the complexity of Boolean minimization for several classes of formulas with polynomially solvable SAT, and formulate sufficient conditions for a class which cause the minimization problem to drop at least one level in the polynomial hierarchy. Second, we study a class of matched CNFs for which SAT is trivial but minimization remains Σ_2^p complete. We prove that every matched CNF has at least one equivalent prime and irredundant CNF that is also matched. We use this fact to prove the main result of this part, namely that for every matched CNF all clause minimal equivalent CNFs are also matched. Third, we look at propagation completeness – the property of a CNF that says that for every partial assignment all entailed literals can be discovered by unit propagation. We can extend every CNF to be propagation complete by adding empowering implicates to it. The main result of this section is a the proof of coNP completeness of the recognition problem for propagation complete CNFs. We also show that there exist CNFs to which an exponential number of empowering implicates have to be added to make them propagation complete.
Keywords: Boolean functions, Satisfiability (SAT), Boolean minimization, Matched formulas, Propagation completeness, Empowering implicates

Contents

Chapter 1

Introduction

Boolean function is a mapping with n binary inputs and one binary output. It is a very general concept that appears frequently in computer science and related fields. The interest in Boolean functions first originated from propositional logic, but today they appear in a broad spectrum of areas. Boolean functions provide canonical examples for many classes in complexity theory. They appear in design of computer chips and circuits. They model problems in the area of artificial intelligence. They are used in software verification, and we can also find them in dependency resolution in various software package management systems.

In this dissertation we will deal with Boolean functions represented in their normal forms. We will study these normal forms from several angles and deal with many questions related to them.

In Chapter 2 we will define Boolean functions and formulas and their conjunctive and disjunctive normal forms. We will show the duality that exists between these forms that allows us to translate results between them. We will also present two basic Boolean problems – satisfiability and minimization – and establish their computational complexity (NP-completeness, i.e. Σ_1^p completeness, for satisfiablity, and Σ_2^p completeness for minimization).

In Chapter 3 we will take a look at several classes of Boolean formulas in conjunctive normal form for which the satisfiability problem is solvable in polynomial time (its complexity drops down one level in the polynomial hierarchy). We will examine what happens with the complexity of Boolean minimization for formulas in these classes. We will provide examples of classes where the minimization problem is solvable in polynomial time (its complexity drops down two levels in the polynomial hierarchy) and also of classes where it is NP-complete (its complexity drops down one level in the polynomial hierarchy). We will defer an example of a class with polynomial time satisfiablity for which the complexity of minimization stays the same as for the general case (Σ_2^p complete) to the Chapter 4. Then we will generalize the observations made above for particular classes and specify sufficient conditions that ensure that the complexity of minimization for a given class drops down at least one level in the polynomial hierarchy. We will end this chapter with a short note on the complexity of recognition of classes of Boolean formulas.

In Chapter 4 we will focus on one specific class of formulas in conjunctive normal form – matched formulas. This class is an example of class that does not satisfy the sufficient conditions from Chapter 3. We will provide a new and simple

proof of Σ_2^p completeness of the minimization problem for this class. Then we will focus on the properties of minimum representations of matched formulas. We will show that for any matched formula there exists at least one equivalent prime and irredundant formula that is matched. We will conclude this chapter by showing that every minimum representation of a matched formula is also matched.

Chapter 5 deals with a property called propagation completeness for formulas in conjunctive normal form. Any conjunctive normal form can be expanded to a propagation complete one by adding so called empowering implicates. Finding and adding empowering implicates is a type of knowledge compilation – a process that changes a knowledge base in such a way that it does not allow inference of new knowledge, but makes queries to the knowledge base faster. We will also provide connection between propagation completeness and arc consistency in constraint programming. The main results of Chapter 5 are linked to the complexity issues connected to propagation completeness and empowering implicates, such as what is the complexity of recognition that a given clause is an empowering implicate of a given formula, and of recognition that a given formula is propagation complete. We will look at the hardness of a problem of generating some empowering implicate for a given formula, and also show that it may be necessary to add exponentially many empowering implicates to a given formula to make it propagation complete.

Chapter 3 is based on the conference paper "On the Gap between the Complexity of SAT and Minimization for Certain Classes of Boolean Formulas" by Ondřej Čepek and Štefan Gurský presented at ISAIM 2014 [20]. Chapter 4 is based on a paper in preparation "On minimum representations of Matched Formulas" by Ondřej Čepek, Štefan Gurský and Petr Kučera. Chapter 5 is based on and uses parts of the article "Complexity issues related to propagation completeness" by Martin Babka, Tomáš Balyo, Ondřej Čepek, Štefan Gurský, Petr Kučera and Václav Vlček that appeared in Volume 203 of Artificial Intelligence in October 2013 [5].

Chapter 2

Boolean functions

2.1 Boolean functions and formulas

Boolean function on n variables is a mapping from $\{0,1\}^n$ to $\{0,1\}$. Such a function can be represented in many ways. Since it has only finitely many possible inputs, one can list values for every input. This representation of a Boolean function is called a truth table representation. The name truth table comes from interpretation of one and zero as *true* and *false* respectively. Also, the input vectors where function value is zero are called *falsepoints* and input vectors where it is one are called *truepoints*.

Since each of 2^n possible inputs can have one of two outputs, it is easy to see that there are $2^{(2^n)}$ possible Boolean functions on n variables. The advantage of truth table is that every function has exactly one truth table representation so one can find out if two Boolean functions given in a different representation are identical by constructing truth table representation and comparing those.

The main disadvantage of table representation of Boolean function is that it is always exponential in the number of input variables. There are other representations of Boolean functions that, for some functions, do not take space exponential in number of variables. However, since there are $2^{(2^n)}$ Boolean functions on n variables, simple counting argument shows us that for any representation, if we were to write it down using finite alphabet, there is a function that will need exponentially many symbols.

The most common representations of Boolean functions, besides truth table, are Boolean circuits, decision diagrams and Boolean formulas. In this work we consider only Boolean formulas and only certain kinds of Boolean formulas.

Boolean formula is a representation of Boolean function that uses Boolean variables and Boolean connectives (usually in infix notation). Boolean connectives are certain named Boolean functions with at most two variables.

Boolean variable is a variable that can get value either one or zero. We will use lowercase letters sometimes with indices to represent Boolean variables. Negation is a Boolean function with one input written as $\neg x$ or as $\overline{x}$ that returns $1 - x$. Negation is a Boolean connective. A variable or negation of variable is called a *literal*. Conjunction of n Boolean inputs is a Boolean function that returns one when all inputs are set to one and zero otherwise. For zero inputs the value of a conjunction is one. Conjunction on 2 variables is a logical connective called *and* and is written in an infix notation as $x \wedge y$, sometimes only as xy. We will call

a conjunction of literals a *term*. We will consider term to be a set of literals and use set operations on them. We will use $\top$ as a symbol for an empty term.

A disjunction of n Boolean inputs is a Boolean function that returns zero when all its inputs are zero and one otherwise. For zero inputs the value of a disjunction is zero. Disjunction on 2 variables is a logical connective called *or* and is written in an infix notation as $x \vee y$, sometimes as $x + y$. We will call a disjunction of literals a *clause*. We will also use set operations on clauses. We will use symbol $\bot$ for an empty clause.

We do not allow both literal l an $\bar{l}$ in clause or in term. For any input vector $\vec{x}$ the value of such a clause would be 1 and value of such a term would be 0, so such clauses or terms can be always replaced by the appropriate constant.

We can define Boolean formula inductively. A Boolean variable is a Boolean formula. Boolean constants 1 and 0 are Boolean formulas. If φ is a Boolean formula, then $\neg\varphi$ is a Boolean formula. If φ and ψ are Boolean formulas and $\circ$ is a Boolean connective, then $\varphi \circ \psi$ is a Boolean formula. We will represent Boolean formulas using lowercase Greek letters. For a formula φ representing function f we will use notation $\varphi(\vec{x})$ interchangeably with $f(\vec{x})$.

Definition. *We say that two Boolean formulas are* equivalent *if they represent the same Boolean function.*

While there are more Boolean connectives, we will use only connectives *and*, *or* and *not*.

There is relation between clauses and terms given by *De Morgan's laws* that say that negation of clause is term with negated literals and negation of term is a clause with negated literals. More formally:

$$\neg \bigvee_{i=1}^{n} l_i = \bigwedge_{i=1}^{n} \neg l_i$$

and

$$\neg \bigwedge_{i=1}^{n} l_i = \bigvee_{i=1}^{n} \neg l_i$$

We will look only at two special kinds of formulas, those that are in one of the normal forms.

Definition. *A Boolean formula that is a disjunction of terms is said to be in a* disjunctive normal form *(DNF).*

Definition. *Boolean formula that is a conjunction of clauses is in* conjunctive normal form *(CNF).*

We will often say " DNF φ" instead of "φ in disjunctive normal form" and "CNF φ" instead of "φ in conjunctive normal form".

We will also often treat CNF as set of clauses and DNF as set of terms and use set notations when working with them.

Fact ([22])**.** *Any Boolean function can be represented in both normal forms.*

This representation is usually not unique. There are also functions that have one representation short but the other is exponentially longer and cannot be shortened.

There is a duality between CNFs and DNFs that allows us to adapt results about one of the normal forms to the other. This is because it is straightforward to produce DNF of function $\neg f$ if one has CNF of function f and also other way round. One just swaps *ands* and *ors* and replaces literals with their complements. This duality means there are similar concepts, problems and terminology for CNFs and DNFs and many of the following facts have both CNF and DNF version.

Definition. *An* implicate *of a function f is such a clause C for which $f \implies C$. That is for any vector $\vec{x}$ such that $f(\vec{x}) = 1$ also $C(\vec{x}) = 1$.*

While definition of implicate does not depend on representation of f, it is common to talk about implicates when working with CNFs, since every clause in CNF φ is an implicate of a function represented by φ. Also for any implicate C of function f represented by CNF φ, $\varphi \cup \{C\}$ is again CNF representation of f.

In DNF world we have a notion of implicant.

Definition. *And* implicant *of a function f is such a term t for which $t \implies f$. That is for any vector $\vec{x}$ such that $t(\vec{x}) = 1$ also $f(\vec{x}) = 1$.*

Again, while this definition does not require special representation of f, it is connected to DNF in the same way as implicates are with CNFs. All terms of DNF are its implicants and adding an implicant to DNF does not change the represented function.

Definition. *We say that an implicate C' of f* subsumes *an implicate C of f if $C' \subseteq C$.*

Implicants have a similar notion.

Definition. *We say that an implicant t' of f* absorbs *an implicant t of f if $t' \subseteq t$.*

Fact (CNF version, [22])**.** *If CNF φ contains two clauses C and C' such that $C' \subseteq C$ then $\varphi \setminus C$ represents the same function as φ.*

Fact (DNF version, [22])**.** *If DNF φ contains two terms t and t' such that $t' \subseteq t$ then $\varphi \setminus t$ represents the same function as φ.*

Definition. *An implicate C is called* prime *if there is no implicate $C' \subset C$.*

An implicant t is called prime *if there is no implicant $t' \subset t$.*

Since adding an implicate to CNF and removing subsumed implicate does not change the represented function, we can replace every clause in CNF by a prime implicate that subsumes it. We call CNF that contains only prime implicates *prime CNF*. In the same way we define *prime DNF*.

Definition. *We call CNF that contains all prime implicates of a function f and does not contain any non-prime implicate a* canonical *CNF representation of f.*

Definition. *We call DNF that contains all prime implicants of a function f and does not contain any non-prime implicants a* canonical *DNF representation of f.*

Here we should note that the only prime implicate of a function without truepoints is an empty clause $\bot$ and the only prime implicant of a function without falsepoints is an empty term $\top$.

Fact ([22]). *For any function there is only one canonical DNF and only one canonical CNF.*

There is an algorithm that from any given CNF produces an equivalent canonical CNF. Virtually identical algorithm produces canonical DNF from a DNF input.

Definition. *We say that two clauses (terms) A and B have* conflict *in literal l if $l \in A$ and $\bar{l} \in B$.*

Fact ([22]). *Let A and B be two implicates (implicants) of f. If they have conflict in only one literal l then $A \cup B \setminus \{l, \bar{l}\}$ is also an implicate (implicant) of f.*

Applied to clauses (implicates), this fact is called the *resolution rule* or just resolution. Clause $A \cup B \setminus \{l, \bar{l}\}$ is called a *resolvent* of parent clauses A and B. In DNF world, where A and B are terms, the term $A \cup B \setminus \{l, \bar{l}\}$ is called the *consensus* of terms A and B.

Definition. *A resolution in which the parent clauses have no common literal ($A \cap B = \emptyset$) is called a* non-merge resolution*, otherwise it is a* merge resolution*.*

A resolution method is a repeated application of the resolution rule. Its DNF counterpart is called a consensus method (or consensus procedure) These methods are complete in the following sense.

Fact (CNF version). *Starting from any CNF φ representing function f, any prime implicate of f can be produced by repeated application of the resolution rule.*

Fact (DNF version, [45]). *Starting from any DNF φ representing function f, any prime implicant of f can be produced by repeated application of the consensus rule.*

The algorithm for producing canonical CNF uses resolution rule and subsuming of implicates. It takes CNF φ and uses resolution rule on all pairs of clauses where it can be applied. Then it removes all clauses that are subsumed by any other clause and calls this CNF ψ. If φ and ψ are identical (as sets of clauses), algorithm stops and returns ψ. Otherwise it repeats both steps with ψ as φ.

Returned ψ is a canonical CNF for the function represented by φ. The running time of the algorithm can be exponential in number of variables. There are CNFs where exponentially many implicates have to be added to receive the canonical CNF.

By now it should be obvious that same can be done for DNF and resulting DNF is the canonical DNF for a given function.

For a prime implicate C of φ we can define *resolution proof.*

Definition. Resolution proof *(sometimes called resolution derivation) of a (prime) implicate C of φ is a sequence of clauses $C_1, C_2, \ldots C_n$ such that every clause C_i is either in φ or is a resolvent of some clauses C_j and C_k for $j, k < i$ and $C_n = C$. We call resolution proof of an empty clause $\bot$ a* resolution refutation*.*

Resolution proof can be represented as a directed acyclic graph with clauses as vertices with edges going from parent clauses to their resolvent.

Definition. *We call resolution proof* tree-like *if every occurrence of a clause is used at most once as parent clause.*

Any resolution proof can be converted to tree-like resolution proof by repeating clauses if necessary. The resolution graph for treelike resolution is a tree and depth of tree-like resolution proof is the length of the longest path from leaf to root in its resolution graph.

Definition. *Resolution is* regular *if in any path from leaf to root no variable is resolved more than once.*

It is known that any prime implicate of given CNF can be derived by a regular tree-like resolution [53].

While there is only one canonical CNF (and canonical DNF) for a given function, it is often the case that one could remove some (even most) of clauses (terms) from it and not change the represented function.

Definition. *CNF (DNF) φ is called* irredundant *if for any CNF (DNF) $\psi \subset \varphi$ function represented by ψ is different from function represented by φ.*

There is often more than one irredundant prime CNF and more than one irredundant prime DNF for a function f.

2.2 Satisfiability and falsifiability

From the point of view of computational complexity theory, Boolean functions are objects that provide canonical examples of problems for many complexity classes. We will again see the duality between CNFs and DNFs. Let us start with the most famous of the problems.

SAT
Instance: Boolean formula φ in CNF
Question: Is there a vector $\vec{x}$ such that $\varphi(\vec{x}) = 1$?

SAT was the first problem shown to be NP complete [21]. While there are many others known NP complete problems, SAT is still probably the most studied one since solving it is useful in practical applications such as software verification or constraint solving. In practical applications we are usually not interested only in Yes/No answer but for Yes answer also in the truepoint $\vec{x}$. For answer No we may get resolution refutation of the input CNF. However, this resolution refutation is sometimes of exponential size (if it could be always polynomial we would have NP=coNP).

The DNF version of satisfiability is trivial. It suffices to satisfy any term in DNF, which means that unless every term of input contains some complementary pair of literals (literal and its negation), it is satisfiable.

The DNF equivalent of SAT is FALS.

FALS

Instance: Boolean formula φ in DNF

Question: Is there a vector $\vec{x}$ such that $\varphi(\vec{x}) = 0$

The conversion between these two problems is straightforward and is based on the duality mentioned above. Given an instance of one of these problems, doing the negation of input formula creates an instance of the other.

The SAT problem is not only of theoretical interest. There is a real need to solve its instances and a lot of research goes to creating SAT solvers – software that can solve SAT instances. There are even competitions where people compete with their SAT Solvers against each other.

Most SAT solvers today assign values to variables one by one and keep track of this assignment, backtracking when necessary.

Definition. *A* partial assignment *is a mapping from Boolean variables of given Boolean function to* $\{0, 1, *\}$.

A partial assignment that does not use $*$ is equivalent to an input vector (for a given function). Variables that are mapped to $*$ are said to be unassigned.

One can find natural correspondence between partial assignments and clauses and between partial assignments and terms. To a clause C we attach partial assignment α that assigns only variables from C and in such a way that it does not satisfy C. To a term t we attach partial assignment α that assigns only variables in t in such a way that it satisfies t.

Definition. *We say that partial assignment α extends partial assignment β if there is no variable x such that $\alpha(x) = *$ but $\beta(x) \neq *$ and for all variables x where $\beta(x) \neq * \; \alpha(x) = \beta(x)$.*

Note, that α extends β if and only if clause (term) corresponding to β is a subclause (subterm) of the one corresponding to α.

Since clause is true (its value is one) when any of its literals is true, it is possible to satisfy a clause without assigning value to all variables in it. Also one does not need to assign all variables to satisfy CNF. That is why one can say that SAT Solvers are searching for a partial assignment that satisfies its input CNF. Once it is found, any assignment that extends it is also satisfying.

When SAT Solver has a partial assignment it applies it to formula and simplifies the formula. If any literal in clause is set to one, the clause can be dropped as satisfied. If that results in no more clauses, all clauses are satisfied and satisfying partial assignment is found. If literal in clause is set to zero, it can be dropped from that clause. When that results in empty clause, that clause is falsified and that partial assignment cannot be extended to a satisfying one. If this happens, SAT solver backtracks. The algorithm Conflict Driven Clause Learning (CDCL) that is used in modern SAT solvers analyzes the partial assignment that falsified the input CNF and adds a new clause (implicate) to make following search faster. It also uses this analysis to backjump to appropriate level.

During search for a satisfying assignment, most SAT Solvers use these two techniques.

- Pure literal elimination
- Unit propagation

Definition. Pure literal *in CNF is such a literal that its complementary literal is not present in CNF.*

Assigning a pure literal to one (that is assigning its variable one if the literal is this variable and to zero if literal is negation of this variable) does not only satisfy every clause with this literal (this is true for any literal) but it cannot falsify any other clause (since there is no clause with complementary literal). Therefore it is reasonable to scan for pure literals and assign them value one.

Unit propagation looks at unit clauses. Unit clauses are clauses of size (length) one. To satisfy unit clause it is necessary to set its only literal to one. Then one again simplifies CNF. During this simplification it is possible that another clause becomes unit and it is processed in the same way. Unit propagation stops only when no more unit clauses are present in CNF. It is known that unit propagation can be done in linear time in the length of the input CNF.

2.3 Boolean minimization

The other problem we will consider in this dissertation is Boolean minimization. Here we ask for a CNF equivalent to an input that is short in some sense. There are several measures that one can use for Boolean minimization. Two most common are number of clauses for CNF (or terms for DNF) and number of occurrences of literals – sum of sizes of clauses (terms). Two normal forms and two measures give us four variants of Boolean minimization.

MIN TERM DNF

Instance: DNF φ and an integer k.

Question: Is there a DNF ψ equivalent to φ that has at most k terms?

MIN LIT DNF

Instance: DNF φ and an integer k.

Question: Is there a DNF ψ equivalent to φ that has at most k occurrences of literals?

For CNF versions MIN CLAUSE CNF and MIN LIT CNF it suffices to replace DNF in problem statements with CNF.

For some reason, most of the results about Boolean minimization are done for DNFs. It is known that MIN TERM DNF and MIN LIT DNF are both Σ_2^p complete [51]. However, the duality of CNF and DNF shows us immediately that CNF versions are equivalent to DNF versions and are therefore also Σ_2^p complete.

While there are algorithms and software packages that perform Boolean minimization they are mostly concerned with minimizing circuits and not formulas.

We should note that all these problems ask for equivalent formula. Again we get two versions of this problem, one for CNF and one for DNF. Let us look at CNF version, since DNF is again equivalent to CNF one.

EQUIV CNF

Instance: CNF φ and CNF ψ

Question: Are φ and ψ equivalent?

The problem EQUIV CNF is easily shown to be coNP complete, since one can test equivalence with $\perp$ to see if a formula is unsatisfiable. However, in next chapter we will see several classes of CNFs where easier equivalence testing brings the complexity of Boolean minimization down in polynomial hierarchy.

Chapter 3

Complexity of SAT and minimization for certain CNF classes

While SAT is NP complete for general CNFs, there are classes of CNFs that are interesting and where SAT is solvable in polynomial time. Thanks to the duality of CNF and DNF each of these classes has a DNF counterpart that has polynomially solvable FALS. One may wonder, what happens with the complexity of Boolean minimization for these classes. In the general case, there is a gap of one level in the polynomial hierarchy between them. For a class with polynomially solvable SAT there are three possibilities.

- Minimization drops down two levels. There is no gap between satisfiability and minimization.
- Minimization drops down one level. The gap stays one level.
- Minimization stays as hard as for general CNFs. The gap between minimization and satisfiability increases to two levels.

In this chapter we will show examples for the first two cases and defer the third case to the next chapter. We will also provide sufficient conditions for a class of formulas ensuring that the minimization problem drops down at least one level in the polynomial hierarchy.

We should note that for no class of CNFs can minimization be easier than satisfiability. To see if formula φ is satisfiable if we can solve Min Lit CNF we ask if this formula has equivalent CNF with zero occurrences of literals. If not, then it is satisfiable, since unsatisfiable formula has such a CNF. If yes, then it is equivalent to either constant zero (CNF with an empty clause) or constant one (an empty CNF). We can distinguish these two cases we just evaluate φ on any input vector (for example on constant ones). If we can solve Min Clause CNF, we ask whether given CNF has equivalent CNF with at most one clause. If not, then it is satisfiable (unsatisfiable CNF has equivalent CNF with one empty clause). If yes then, if this clause is not empty, this clause is satisfied by one of the input vectors $\vec{1}$ (all ones) or $\vec{0}$ (all zeros). If neither of these two vectors satisfy it, the CNF is equivalent to an empty clause and is not satisfiable. In formal reduction one would do evaluation on input vectors first.

3.1 Monotone CNFs

An example of a class of CNFs where minimization drops down two levels to class P are monotone CNFs.

Definition. *A* monotone CNF *is a CNF without negations, i.e. every literal in a monotone CNF is positive.*

The CNF where every literal is negative is called anti-monotone. The name comes from monotone functions. A Boolean function is monotone if for any two vectors $\vec{x}$ and $\vec{y}$ such that $\vec{x} \leq \vec{y}$ (compared componentwise) we get $f(\vec{x}) \leq f(\vec{y})$. Every monotone function can be represented by formula without negations (which can be then converted to monotone CNF) and every monotone CNF represents monotone function.

Sometimes monotone functions are called positive and antimonotone are called negative, with the word monotone covering both.

The DNF counterpart to monotone CNFs are antimonotone DNFs – those with only negative literals – and DNF counterpart of antimonotone CNFs are monotone DNFs. Again, every monotone function has monotone DNF representation and vice versa.

Satisfiability for monotone CNFs is trivial. They are satisfiable if an only if they are satisfied with vector $\vec{1}$ (that is iff they do not contain an empty clause). Antimonotone are analogous. They are satisfied with vector $\vec{0}$, again if and only if they do not contain an empty clause. Monotone DNF can be falsified by $\vec{0}$ or not at all (when it contains only empty term) and antimonotone by $\vec{1}$.

Minimization of monotone CNFs is also in P, both for number of clauses and number of occurrences of literals. Monotone CNF does not allow resolutions, so all prime implicates must be contained in it. Given an instance of Boolean minimization (either case), it suffices to do absorption of non-prime implicates and test the resulting (canonical) CNF whether it is "smaller than k" for given measure. Same argument goes for antimonotone CNFs and for monotone and antimonotone DNF counterparts.

3.2 Horn CNF

Horn CNFs are a superclass of antimonotone CNFs. While antimonotone CNFs do not have any positive literal, Horn CNFs allow at most one positive literal in each clause.

Definition. Horn CNF *is a CNF where every clause has at most one positive literal.*

Horn CNFs are prominent in the field of logic programming and are named after Alfred Horn. A clause in the form $h \vee \overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_n}$ is equivalent to implication $x_1 \wedge x_2 \wedge \ldots \wedge x_n \implies h$. The negative literals on x-variables are called subgoals and literal h is called head.

DNFs also have their Horn subclass.

Definition. Horn DNF *is a DNF where every clause has at most one negative literal.*

We call clauses with positive literal *pure Horn* and clauses without positive literal *negative*.

Horn DNFs are exactly negations of Horn CNFs. Results about one normal form translates directly to results about the other.

Satisfiability for Horn CNFs is well known to be in P [24, 36, 41]. Antimonotone CNFs were satisfied by vector of zeros. The clauses in Horn CNFs that are not satisfied by this vector are unit clauses with only one positive literal. However, these clauses can be dealt with using unit propagation. If unit propagation did not create an empty clause then the remaining clauses can be satisfied by assigning every variable value zero.

Both Boolean minimization problems for Horn formulas drop down one level to NP and both are known to be NP complete [4, 33, 40, 16].

There is, however, one fact that is important about Horn CNFs. Every prime irredundant Horn CNF of a given function has the same number of negative clauses [32].

3.3 Renamable Horn

Horn clauses arise in practice and in that case variables in them have real world meaning. If a variable represents two possible states of world one is free to choose which value represents which state. However, this choice can affect whether the CNF representing the problem is Horn. Renamable Horn CNFs are generalization of Horn CNFs for which there exists "renaming" of variables that produces Horn CNF.

Definition. *A formula φ is a* renamable Horn formula *if there exists V, subset of its variables, such that replacing literals x with $\overline{x}$ and $\overline{x}$ with x in φ for every variable $x \in V$ produces Horn CNF.*

The important fact is that renamable Horn CNFs can be recognized and subset V of variables that has to be flipped to produce Horn CNF can be constructed in polynomial time using the following method from [39]. Given CNF $F = \bigvee_i = 1^m C_I$ with $C_i = \bigwedge_{j=1}^{|C_i|} l_{i,j}$, create

$$F_h = \bigwedge_{i=1}^{m} \bigwedge_{1 \le j < k \le |C_i|} (l_{i,j} \vee l_{i,k}$$

Then F is renamable Horn if and only if F_h is satisfiable and those variables that are set to one in satisfying assignment are those that constitute set V of variables whose switching makes F Horn. Finding satisfying assignment of F_h can be done in polynomial time, since F_h is quadratic (see next section).

Results about Horn CNFs translate to renamable Horns straightforwardly. Satisfiability is in P, since we just rename renamable Horn to Horn CNF and solve it. Minimization for renamable Horn is also in the same level of polynomial hierarchy as for Horn CNFs, that is it is NP complete for both measures.

Negations of renamable Horn CNFs are renamable Horn DNFs and conversion between CNF and DNF versions of problems works as expected.

3.4 Quadratic CNF

Definition. *Quadratic CNF is a CNF where every clause has at most two literals.*

Quadratic CNFs are sometimes called 2CNFs and satisfiability problem for them is called 2-SAT.

Since resolvent of two clauses of quadratic CNF can have at most two literals creating canonical CNF for a given quadratic CNF takes only polynomial time. Since canonical CNF for non satisfiable function is one that contains only empty clause, one can solve 2-SAT using resolution only.

Another way to solve 2-SAT is first doing unit propagation and then looking at the remaining quadratic clauses $a \vee b$ as pair of implications $\neg a \implies b$ and $\neg b \implies a$. If one can chain implications in such a way that one gets from a to $\neg a$ and from $\neg a$ to a, then the formula is unsatisfiable. Aspvall et al. in [2] showed that this is necessary and sufficient condition. Their algorithm creates an oriented graph on literals using " $\implies$ " as edges and checks if any complementary pair of literals is in the same strongly connected component. It runs in linear time.

For a quadratic CNF it is possible to find shortest equivalent CNF (it is the same CNF for both measures) in polynomial time, so the decision version of Boolean minimization of quadratic CNFs is also in P – one just checks whether constructed CNF is smaller than given k. Same holds for DNFs, since negation of quadratic CNF is quadratic DNF.

The algorithm for minimization of quadratic CNFs can be considered a folklore but for completeness we provide the method here.

The first fact that is used is that quadratic prime implicates share no variables with unit prime implicates. If there were two implicates, one unit and one quadratic, the quadratic one would be absorbed either by given unit implicate or by their resolvent.

The second fact used in this algorithm is that quadratic CNF is satisfiable if and only if it is hidden Horn. This can be seen from method of recognition of hidden Horn CNFs. CNF F is hidden Horn if and only if F_h is satisfiable and for quadratic CNF F the corresponding F_h is identical to F.

The third fact used is that in any prime irredundant Horn CNF has same number of negative clauses.

So to minimize quadratic CNF φ, we can do that by the following algorithm.

1. Create canonical CNF φ_c equivalent to φ.

2. Put aside unit clauses since they all have to be in any prime irredundant equivalent CNF. Name the remaining pure quadratic CNF φ_q.

3. Find V, the set of variables, renaming of which, makes the CNF Horn. Call this Horn CNF φ_h.

4. Try removing negative clauses one by one, checking after each removal whether represented function changed.

5. Put remaining negative clauses aside. They all will be in the constructed minimum CNF.

6. Create graph G with variables as vertices and an directed edge from a to b for every clause $\neg a \vee b$.
7. Do transitive reduction of the condensation of G and call it G_c.
8. Remove all edges between strongly connected components in G.
9. Replace edges in every strongly connected component by a directed circle.
10. For every edge XY in G_c put an edge xy to G where x is any vertex in component X and y is any vertex in component Y.
11. Convert G back to CNF.
12. Put back negative clauses.
13. Undo renaming.
14. Put back unit clauses

The result is a minimum CNF that represents the same function as input φ. All steps can be done in polynomial time.

3.5 qHorn

qHorn CNFs are generalization of both Horn and quadratic CNFs.

Definition. *Formula φ is* qHorn *if there exists clause P such that for every clause $C \in \varphi$ either $|C \setminus P| \leq 1$ or $|C \setminus P| = 2$ and there is no literal l such that $l \in C$ and $\bar{l} \in P$.*

There is another possible definition that uses *valuation*.

Definition. Valuation *u is a mapping from literals to interval $[0, 1]$ such that for every literal l $u(l) = 1 - u(\bar{l})$.*

Definition. *A formula φ is* qHorn *if there is a valuation u such that for every $C \in \varphi$ $\sum_{l \in C} u(l) \leq 1$.*

The class of qHorn CNFs stays the same if we restrict valuation function only to values $\{0, \frac{1}{2}, 1\}$. If a given CF φ is qHorn with valuation u, then we define valuation u' such that

$$u'(l) = \begin{cases} 0 & \text{if} u(l) < \frac{1}{2} \\ 1 & \text{if} u(l) > \frac{1}{2} \\ \frac{1}{2} & \text{if} u(l) = \frac{1}{2} \end{cases}$$

and u' will still be valuation that confirms that CNF is qHorn.

The definition with valuation is equivalent to the definition that uses clause P. Literals in P will get value zero, their complements have to get value one and those not in P get value $\frac{1}{2}$. The opposite direction is analogous.

Quadratic CNF is qHorn CNF with empty clause P, Horn CNF is qHorn CNF with P containing all positive literals, renamable Horn has P containing all variables.

The qHorn DNFs are exactly negations of qHorn CNFs, so the duality works again and results about satisfiability and Boolean minimization can be translated straightforwardly.

The satisfiability for qHorn CNFs is in P. The Boolean minimization of both kinds is NP complete for qHorn formulas. It cannot drop to P, because otherwise minimization of Horn formulas, that are subclass of qHorn, would also be in P.

3.6 Restrictions of Horn CNFs

While we know that Horn CNFs have NP complete minimization, there are some subclasses of Horn CNFs that have both minimization problems in P. Their definitions rest on the notion of the CNF digraph.

Definition. *For Horn CNF φ, the* CNF digraph G_φ *is a directed graph, where vertices are the Boolean variables and every clause C in φ with exactly one positive literal (head) generates directed edges from variables corresponding to negative literals (subgoals) in C to the head of C. Note that clauses having only negative literals generate no edges in G_φ.*

Definition. *Horn CNF is* acyclic *if its CNF digraph is acyclic.*

Definition. *Horn CNF is called* quasi-acyclic *if every directed edge inside a strongly connected component of its CNF digraph comes from a quadratic prime implicate.*

As we saw in section about quadratic CNFs, such edges can be seen as implications between variables and if they are in a strongly connected component they give us equivalence between variables. We can view quasi-acyclic Horn CNFs as acyclic CNFs where each variable may be given several names. Quasi-acyclic Horn CNFs are a proper superclass of acyclic Horn CNFs.

Definition. *Horn CNF φ is* CQ-Horn *(component-wise quadratic Horn) if every prime implicate C of φ has at most one subgoal of C in the same strong component of G_φ as the head of C.*

CQ-Horn CNFs are a proper superclass of quasi-acyclic Horn CNFs. The polynomial algorithm that minimizes the number of clauses and the number of occurrences of literals for acyclic Horn CNFs appeared in [32] and for quasi-acyclic in [34]. The polynomial algorithm for minimization of CQ-Horn CNFs was presented in [14] and works for both the number of clauses and for the number of occurrences of literals. Since acyclic and quasi-acyclic CNFs are proper subclasses of CQ-Horn CNFs, the algorithm works for them too.

3.7 When the complexity of minimization drops

Boolean minimization problem has a typical Σ_2^p structure. It asks whether there exists an object (in this case CNF not larger than k) such that for all objects (in this case assignments) something testable in polynomial time holds (in this case same value for both CNFs). The second part (same value of function for all inputs

– that is equivalence testing) is coNP complete for general formulas. However, if for some class it drops to P, then Boolean minimization gets NP structure and drops down to NP.

Definition. *We will call class X of CNFs* tractable *if it satisfies the following three conditions:*

- *Satisfiability: Given any CNF φ from X it is possible to decide whether it is satisfiable in time polynomial in size of φ.*
- *Partial assignment: Given CNF φ from X and any partial assignment α, the CNF $\varphi(\alpha)$ is again in X.*
- *Prime representations: Given any CNF φ from X, any prime CNF ψ equivalent to φ must also be in X.*

Often also polynomial time recognition whether φ belongs to X is required to call X tractable.

Lemma 3.7.1. *For two CNFs φ and ψ that both belong to some tractable class (not necessarily both to the same) it is possible to test in polynomial time whether they are equivalent.*

Proof. To see if two CNFs φ and ψ are equivalent it suffices to check whether every clause of φ is an implicate of ψ and if every clause of ψ is an implicate of φ. Checking whether clause C is an implicate of CNF φ can be done by applying partial assignment corresponding to C on φ and checking whether the resulting CNF is satisfiable. □

Lemma 3.7.2. *Let X be a tractable class. Then both Boolean minimization problems for CNFs from this class belong to NP.*

Proof. It suffices to show that there exists a certificate that confirms positive instances of Boolean minimization. For positive instance (φ, k) of Boolean minimization (either Min Clause CNF or Min Lit CNF)) this certificate is a prime CNF ψ that is „at most of size k“ (has at most k clauses or has at most k occurrences of literals respectively). If there exists any CNF that is at most of size k then there exists prime CNF that is not larger, since replacing every clause by its prime subset does not make the CNF larger in any of these two measures. Since φ is from tractable class, it means that ψ is also from tractable class (third property) and their equivalence can be tested in polynomial time. □

All of the classes presented in this chapter so far have been tractable and therefore Boolean minimization dropped to NP (sometimes even to P). To find a class where minimization does not drop down to NP, it must avoid one of the tractability conditions. Since we are interested only in classes that have polynomially solvable SAT, the class must break the second or the third tractability condition. The class of *matched CNFs* from the next chapter is such a class that breaks both the second and the third condition. It is not closed under partial assignment and there are prime CNFs that are not matched but are equivalent to a matched CNF.

3.7.1 Note on recognition

The fourth condition for tractable classes – polynomial algorithm for testing if given CNF belongs to given class – is very useful in practice. However, there seems to be very little connection between complexity of recognizing the class and complexity of SAT and complexity of its minimization. As an example of a class that has easy SAT, easy minimization, but is hard to recognize we can take class of all unsatisfiable CNFs. This class has SAT in P – any formula from this class is unsatisfiable – it has both minimization problems in P – the shortest equivalent CNF for any CNF in this class is CNF with one empty clause – but the question whether given CNF belongs to this class is coNP complete. This class even satisfies all requirements in definition of tractable classes.

While for no class of CNFs can satisfiability get harder than NP and minimization above Σ_2^p, the recognition problem has no bounds. To see an example of that, let us define for any given set S of integers, a class of CNFs such that CNF belongs to this class if the number of its variables is in S. Then complexity of recognition of this class depends on complexity of the set S. Yet, even for this class, the satisfiability would be in NP and minimization in Σ_2^p.

Chapter 4

Matched CNFs

In the previous chapter we presented examples of classes of CNFs with polynomially solvable SAT where complexity of minimization dropped either one or two levels down from the second level of the polynomial hierarchy. In this chapter we will take a look at the class of matched CNFs for which, while having polynomially solvable SAT, the minimization problem stays Σ_2^p complete.

Σ_2^p completeness of minimization of matched CNFs implies that matched CNFs are not tractable. We will show that they do not satisfy two of the three conditions for tractable classes. The class of matched CNFs is not closed under partial assignment, and also there are prime irredundant CNFs that are equivalent to matched CNFs but that are not matched themselves. However, we will show in this chapter, that for any matched CNF there is always some equivalent prime and irredundant CNF that is matched. We will also show that for a matched CNF, every equivalent CNF that has the least number of clauses is again matched.

The concept of matched CNFs is based on graph properties, so to this end we shall use standard graph terminology, see e.g. [12]. Given an undirected graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a *matching* in G if the edges in M are pairwise disjoint. A *bipartite graph* $G = (A, B, E)$ is an undirected graph with disjoint sets of vertices A and B, and the set of edges E satisfying $E \subseteq A \times B$. For a set W of vertices of G, let $\Gamma(W)$ denote the *neighbourhood* of W in G, i.e. the set of all vertices adjacent to some element of W. We shall use the following well-known result on matchings in bipartite graphs:

Theorem 4.0.3 (Hall's Theorem [31])**.** *Let $G = (A, B, E)$ be a bipartite graph. A matching M of size $|M| = |A|$ exists if and only if for every subset S of A we have that $|S| \leq |\Gamma(S)|$.*

Now we are ready to define matched formulas.

Definition. *Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a CNF on n variables $X = \{x_1, \ldots, x_n\}$. We shall associate a bipartite graph $G_\varphi = (\varphi, X, E)$ with φ (also called the incidence graph of φ), where the vertices correspond to clauses in CNF φ and the variables X. A clause C_i is connected to a variable x_j (i.e. $\{C_i, x_j\} \in E$) if C_i contains x_j or $\overline{x_j}$. A CNF φ is* matched *if G_φ has a matching of size m, i.e. if there is a matching which pairs each clause with a unique variable.*

The fact that a clause C_i is matched to a variable x_j in a matching M will be denoted as $[C_i, x_j] \in M$. Note, that a matched CNF is trivially satisfiable. If a

clause C_i is matched to variable x_j, then we can simply assign x_j a value which will satisfy C_i. The name "matched" was given to these formulas in [26], although they appeared already in [1, 50]. A variable which is matched to some clause in matching M is called *matched* in M, it is *free* in M otherwise.

Matched DNFs are analogous to matched CNFs and again we have that negation of a matched CNF is a matched DNF and vice versa. In this chapter we will consider only CNFs, the results translate to DNFs straightforwardly.

4.1 Minimization of matched formulas

As we already hinted, both types of Boolean minimization for matched CNFs are Σ_2^p complete. Proofs for these claims can be found in full in [28].

In the case of number of occurrences of literals one can straightforwardly apply the original proof of complexity of Min Lit DNF from [51]. The formula that comes out of reduction is matched and usual translation between CNF and DNF applies.

The proof of Σ_2^p completeness of general Min Clause CNF from [51], which again presents DNF version, cannot be used as is, since resulting formula is not matched. It has to be amended a bit and this can also be found in [28]. However, here we present a much shorter and simpler proof that the minimization of the number of clauses is Σ_2^p complete for matched CNFs.

Min Matched Clause CNF

Instance: A matched CNF φ and an integer k

Question: Is there a CNF ψ equivalent to φ with at most k clauses?

Theorem 4.1.1. *The problem* Min Matched Clause CNF *is* Σ_2^p *complete.*

Proof. Since Min Matched Clause CNF is a special case of Boolean minimization, and it is known that Boolean minimization is Σ_2^p complete, Min Matched Clause CNF must be in Σ_2^p.

To see that this problem is Σ_2^p hard, we reduce the Σ_2^p complete Boolean minimization to it.

Let (φ, k) be an instance of Boolean minimization where $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$. Let $a_1, a_2, \ldots, a_m$ be new variables that do not occur in φ. Let then φ' be a CNF of $\varphi \vee (a_1 \vee a_2 \vee \ldots \vee a_m)$, that is $\varphi = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_m$ where $C'_i = C_i \vee a_1 \vee a_2 \vee \ldots \vee a_m$ for $1 \leq i \leq m$. φ' is matched, since its clause C'_i can be matched to a literal a_i. The instance of Min Matched Clause CNF is (φ', k).

Let (φ, k) be a positive instance of Boolean minimization. Then there is CNF $\psi = D_1 \wedge D_2 \wedge \ldots \wedge D_{k'}$ (with $k' \leq k$) that is equivalent to φ. Let ψ' be CNF of function $\psi \vee (a_1 \vee a_2 \vee \ldots \vee a_m)$ that is $\psi' = D'_1 \wedge D'_2 \wedge \ldots \wedge D'_{k'}$ where $D'_i = D_i \vee a_1 \vee a_2 \vee \ldots \vee a_m$ for $1 \leq i \leq k'$. Clearly ψ' is equivalent to φ' and has at most k clauses. Therefore (φ', k) is a positive instance of Min Matched Clause CNF.

To see the other direction let (φ', k) be a positive instance of Min Matched Clause CNF and let ψ' be a CNF equivalent to φ' with at most k clauses. Let α

be a partial assignment that sets all a-variables to zero and sets no other variable. Since ψ' is equivalent to φ' and φ' is equivalent to $\varphi \vee (a_1 \vee a_2 \vee \ldots \vee a_m)$, we have $\psi = \psi'(\alpha)$ equivalent to φ. Since ψ comes from ψ' after applying a partial assignment, it cannot have more clauses than ψ'. Therefore ψ has at most k clauses and since it is equivalent to φ we conclude that (φ, k) is a positive instance of Boolean minimization. □

Remark 4.1.2. *Since every clause of φ' in the previous proof had all a-variables appear positively, every resolution from φ' keeps all a-variables in every derived clause. We can assume that ψ' is prime and therefore every clause of ψ' also contains all a-variables positively and from this follows that $\psi = \psi'(\alpha)$ has the same number of clauses as ψ'.*

While Σ_2^p completeness of Min Matched Clause CNF is not a formal proof of coNP completeness of equivalence testing for matched CNFs, it strongly suggests that it is the case. We provide a formal proof of this fact below.

Matched Equiv CNF

Instance: Matched CNFs φ and ψ

Question: Are φ and ψ equivalent?

Observation 4.1.3. Matched Equiv CNF *is coNP complete.*

Proof. Since Matched Equiv CNF is a special case of coNP complete problem Equiv CNF, it is a coNP problem. To see coNP hardness we reduce Equiv CNF to Matched Equiv CNF using the same method as in previous proof. Let (φ, ψ) be an instance of Equiv CNF where $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ and $\psi = D_1 \wedge D_2 \wedge \ldots \wedge D_m$. Without loss of generality let $n \geq m$. Let $a_1, a_2, \ldots, a_n$ be new variables. Let φ' be $\varphi \vee (a_1 \vee a_2 \vee \ldots \vee a_n)$ converted to CNF (that is we add all a-variables to each clause from φ) and ψ' be CNF of $\psi \vee (a_1 \vee a_2 \vee \ldots \vee a_n)$. Both φ' and ψ' are matched, since clause C_i (D_i) can be matched with variable a_i. Then (φ', ψ') is an instance of Matched Equiv CNF. Trivially φ is equivalent ψ if and only if φ' is equivalent to ψ' and the reduction is complete. □

4.2 More definitions

To show the main results of this chapter we will need several concepts which we have not defined yet.

4.2.1 Exclusive sets of implicates of a Boolean function

In this section we recall definition of exclusive sets of implicates of a Boolean function and we state some of their properties, which were shown in [15].

Let us first introduce the following notation. By $\mathcal{I}^p(f)$ we shall denote the set of all prime implicates of a function f. By $\mathcal{I}(f)$ we shall denote the resolution closure of $\mathcal{I}^p(f)$, i.e. an implicate C of f belongs to f if it can be derived by a series of resolutions from $\mathcal{I}^p(f)$.

Definition ([15]). *Let f be a Boolean function and let $\mathcal{X} \subseteq \mathcal{I}(f)$ be a set of clauses. We shall say, that $\mathcal{X}$ is an* exclusive set of clauses of f *if for every pair of resolvable clauses $C_1, C_2 \in \mathcal{I}(f)$ the following implication holds:*

$$R(C_1, C_2) \in \mathcal{X} \Longrightarrow C_1 \in \mathcal{X} \text{ and } C_2 \in \mathcal{X},$$

i.e. the resolvent belongs to $\mathcal{X}$ only if both parent clauses are in $\mathcal{X}$. If function f is clear from the context, we shall simply say that $\mathcal{X}$ is an exclusive set.

We shall recall some of the properties of exclusive sets, which were proved in [15] and which we will use in this dissertation.

Lemma 4.2.1 ([15]). *Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{I}(f)$ be exclusive subsets of f, then both $\mathcal{A} \cup \mathcal{B}$ and $\mathcal{A} \cap \mathcal{B}$ are also exclusive subsets of f.*

Theorem 4.2.2 ([15]). *Let f be an arbitrary Boolean function, let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{I}(f)$ be two distinct sets of clauses which both represent f, and let $\mathcal{X} \subseteq \mathcal{I}(f)$ be an exclusive set of clauses. Then $\mathcal{C}_1 \cap \mathcal{X} \equiv \mathcal{C}_2 \cap \mathcal{X}$, i.e. both represent the same function.*

Based on this proposition we define an exclusive component of a Boolean function.

Definition ([15]). *Let f be an arbitrary Boolean function, $\mathcal{X} \subseteq \mathcal{I}(f)$ be an exclusive set of clauses of f, and $\mathcal{C} \subseteq \mathcal{I}(f)$ be a set of clauses which represents f. The Boolean function $f_{\mathcal{X}}$ represented by the set $\mathcal{C} \cap \mathcal{X}$ is called the* $\mathcal{X}$-component *of the function f. We shall simply call a function g an* exclusive component of *f, if $g = f_{\mathcal{X}}$ for some exclusive subset $\mathcal{X} \subseteq \mathcal{I}(f)$.*

Theorem 4.2.2 guarantees that the $\mathcal{X}$-component $f_{\mathcal{X}}$ is well defined for every exclusive set $\mathcal{X} \subseteq \mathcal{I}(f)$. Theorem 4.2.2 has the following corollary.

Corollary 4.2.3 ([15]). *Let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{I}(f)$ be two distinct sets of clauses such that $\mathcal{C}_1 \equiv \mathcal{C}_2 \equiv f$, i.e. such that both sets represent f, and let $\mathcal{X} \subseteq \mathcal{I}(f)$ be an exclusive set of clauses. Then $(\mathcal{C}_1 \setminus \mathcal{X}) \cup (\mathcal{C}_2 \cap \mathcal{X})$ also represents f.*

4.2.2 Autarkies

Definition. *Let ψ be a CNF on the set V of variables, let $Y \subseteq V$ be a subset of variables, let $L = \{x \mid x \in L\} \cup \{\overline{x} \mid x \in L\}$ be the corresponding set of literals, and let $t : Y \longrightarrow \{0, 1\}$ be a partial assignment on ψ. Then t is an* autarky *on ψ if for every clause $C \in \psi$ either $C \cap L = \emptyset$ or C is satisfied by t.*

Autarky is a special type of partial assignment which satisfies every clause in which it substitutes a value for some literal. We shall prove two simple lemmas about autarkies which will be needed later in this chapter. The first lemma is (in a different notation) contained in [38] as Lemma 3.13, but we will give a short proof here as well to make this chapter self-contained.

Lemma 4.2.4. *Let ψ be a CNF on the set V of variables, let $Y \subseteq V$ be a subset of variables, let $L = \{x \mid x \in L\} \cup \{\overline{x} \mid x \in L\}$ be the corresponding set of literals, and let $t : Y \longrightarrow \{0, 1\}$ be an autarky on ψ. Then t is an autarky on $\mathcal{I}(f)$.*

Proof. Since all clauses in $\mathcal{I}(f)$ can be derived from ψ by resolution, it will suffice to show that resolution preserves the autarky properties, namely that if parent clauses are satisfied by t whenever they contain a literal from L, then so does the resolvent. Let $C, C_1, C_2 \in \mathcal{I}(f)$ be clauses such that $C = R(C_1, C_2)$. Let $\ell \in L$ be a literal in C. If ℓ is satisfied by t we are done, so let us assume that ℓ is not satisfied by t. Clause C inherited ℓ from one of its parent clauses so let us assume without loss of generality $\ell \in C_1$. By the autarky property, C_1 must be now satisfied by t, so it must contain another literal $\ell' \in L$ satisfied by t. Now there are two possibilities: either $\ell' \in C$ (clause C inherited both ℓ and ℓ' from C_1) in which case C is satisfied by t and we are done, or $\ell' \notin C$ which means that $R(C_1, C_2)$ resolves over ℓ'. That implies $\overline{\ell'} \in C_2$, i.e. C_2 contains a literal from L not satisfied by t. Thus, by the autarky property, C_2 must be satisfied by t, so it must contain another literal $\ell'' \in L$ satisfied by t. However, in this case C inherits ℓ'' form C_2, which finishes the proof. □

Corollary 4.2.5. *Let ψ be a CNF on the set V of variables, let $Y \subseteq V$ be a subset of variables, and let $t : Y \longrightarrow \{0,1\}$ be an autarky on ψ. Let $\varphi \subseteq \mathcal{I}(f)$ be an arbitrary representation of f. Then t is an autarky on φ.*

Lemma 4.2.6. *Let ψ be a CNF on the set V of variables, let $Y \subseteq V$ be a subset of variables, let $L = \{x \mid x \in L\} \cup \{\overline{x} \mid x \in L\}$ be the corresponding set of literals, and let $t : Y \longrightarrow \{0,1\}$ be an autarky on ψ. Then $\psi[t]$ represents an exclusive component $f_{\mathcal{X}}$ of f defined by the exclusive set of clauses*

$$\mathcal{X} = \{C \in \mathcal{I}(f) \mid C \cap L = \emptyset\}.$$

Proof. It suffices to show that $\mathcal{X}$ is an exclusive subset of $\mathcal{I}(f)$. Let $C, C_1, C_2 \in \mathcal{I}(f)$ be clauses such that $C = R(C_1, C_2)$ and $C \in \mathcal{X}$. Let us assume by contradiction that one of the parent clauses, say C_1, does not belong to $\mathcal{X}$, which means that there exists $\ell \in C_1 \cap L$. Since $\ell \notin C$, $R(C_1, C_2)$ must resolve over ℓ, which implies $\overline{\ell} \in C_2$. However, one of $\ell, \overline{\ell}$ is not satisfied by t, so the corresponding clause (one of C_1, C_2) must contain some other literal $\ell' \in L$ which is satisfied by t because t is an autarky. But now we get $\ell' \in C$ contradicting the assumption $C \in \mathcal{X}$. □

4.3 Prime representations of matched formulas

We have already mentioned that the class of matched CNFs is not tractable and that it does not satisfy two of the conditions for tractable classes. It is not closed under partial assignment as can be seen from the following example.

Example 4.3.1. *A CNF $(x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee \overline{z})$ is clearly matched, but partial assignment $x = 0$ leaves formula $(y \vee z) \wedge (\overline{y} \vee z) \wedge (y \vee \overline{z})$ that is not matched.*

It is also not difficult to find an example of matched CNF for which there is an equivalent prime and irredundant CNFs that is not matched.

Example 4.3.2. *Consider the CNF*

$$(\overline{a} \vee b) \wedge (\overline{b} \vee c) \wedge (\overline{c} \vee a)$$

which is matched and a logically equivalent CNF

$$(\overline{a} \vee b) \wedge (\overline{b} \vee a) \wedge (\overline{c} \vee b) \wedge (\overline{b} \vee c)$$

which is not matched despite being prime and irredundant.

Thus it is a legitimate question, whether given a (nonprime) matched CNF, there exists at least one logically equivalent prime and irredundant CNF which is also matched. In the rest of this section we will prove an affirmative answer to this question. Let us start with a simple but useful observation.

Claim 4.3.3. *Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a matched CNF and let C be a clause derived by a regular tree-like resolution derivation T from φ. Let $C = R(D_1, D_2)$, where $D_1 = (A_1 \vee z)$ and $(D_2 \vee \overline{z})$, i.e. the resolution is over z. Let T_1 denote the subtree of T rooted at D_1, and let T_2 denote the subtree of T rooted at D_2. If $C_i \in \varphi$ is a leaf clause in both T_1 and T_2, then C_i contains neither z nor $\overline{z}$ and thus it cannot be matched to z in any matching for φ.*

Proof. Since T is regular, the only resolution over z in T is $C = R(D_1, D_2)$. Thus there can be no $\overline{z}$ in T_1 and no z in T_2, since then they would be in D_1 and D_2 respectively as well. Thus C_i which is in both T_1 and T_2 cannot contain z at all. □

Lemma 4.3.4. *Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a matched CNF and let M be a matching for φ. Let D be a clause derived by a regular tree-like resolution T from φ. Let $x \in D$ be a variable which is free in M, then there is a variable $y \in D$ and a matching M' for φ which satisfy the following property: If X denotes the set of variables which are matched in M, and X' denotes the set of variables matched in M', then $X' = (X \setminus \{y\}) \cup \{x\}$.*

Proof. We shall proceed by induction on the depth of the regular tree-like resolution proof T of D. If D is in φ, then we set y to be the variable matched with D in M and

$$M' := (M \setminus \{[D, y]\}) \cup \{[D, x]\}.$$

Now let us assume that $D = R(D_1, D_2)$, where D_1 and D_2 are either clauses in φ, or they are derived by a regular tree-like resolution from φ. Suppose the resolution is over a variable z and let us denote $D_1 = (A_1 \vee z)$, $D_2 = (A_2 \vee \overline{z})$. For $i = 1, 2$ let T_i denote the subtree rooted at D_i, L_i the set of leaf clauses in T_i, φ_i the sub-CNF of φ formed by clauses in L_i, M_i the sub-matching of M on clauses from φ_i, and X_i the set of variables matched in M_i.

First let us assume that variable $x \in A_1 \setminus A_2$. Let us without loss of generality assume that x appears positively in D_1 (otherwise we can switch the polarity of x in φ), thus $D_1 = (A_1' \vee x \vee z)$, where $A_1' = A_1 \setminus \{x\}$. Now we can use induction hypothesis on subtree T_1 rooted at D_1, and matching M_1 on φ_1. It follows that there is a matching M_1' on φ_1 in which x is a matched variable and there is a variable y_1 in D_1 which is matched in M_1 but free in M_1'. Moreover if X_1' denotes the set of variables matched in M_1', then $X_1' = (X_1 \setminus \{y_1\}) \cup \{x\}$. We can extend the matching M_1' to whole φ by adding pairs matching clauses in $L_2 \setminus L_1$ to variables in $X_2 \setminus X_1$ as in M, let the new matching be denoted M''. If $y_1 \neq z$, then y_1 occurs in D and we can set $M' = M''$.

If $y_1 = z$, then it is a free variable in M''. Let M''_2 denote the sub-matching of M'' on clauses of φ_2. Now we can use induction hypothesis on φ_2, its resolvent D_2, and variable $y_1 = z$ (playing role of the free variable). By this we find a matching M^*_2 for φ_2 and a variable y in D_2 such that y_1 is matched in M^*_2, y is free in M^*_2 while it was matched in M''_2. In particular if X^*_2 denotes the set of matched variables in M^*_2 and X''_2 denotes the set of variables matched in M''_2, then $X^*_2 = (X''_2 \setminus \{y\}) \cup \{y_1\}$. We can now extend matching M^*_2 to the whole formula φ by adding pairs matching clauses in $L_1 \setminus L_2$ to variables $X''_1 \setminus X''_2$ as in M'' (here X''_1 denotes the set of variables matched in M'' to leaves of T_1). It is clear that $y \neq z$ and thus $y \in D$. Moreover $X' = (X \setminus \{y\}) \cup \{x\}$.

Now, let us assume $x \in A_1 \cap A_2$. By Claim 4.3.3 we have that $z \notin X_1 \cap X_2$. Let $i \in \{1, 2\}$ be such that $z \notin X_i$. Now let us use induction hypothesis on φ_i and variable x. We get a matching M'_i for formula φ_i which can be extended to the whole φ by pairs matching clauses in $L_{3-i} \setminus L_i$ to variables in $X_{3-i} \setminus X_i$ as in M. We also get a variable y which is matched in M_i, but which is free in M'_i. Since $y \in X_i$, we get that $y \neq z$ and thus y is in D. Clearly $X' = (X \setminus \{y\}) \cup \{x\}$. □

Lemma 4.3.5. *Let $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a matched CNF and let us assume that an implicate D is derived by a resolution derivation from φ in which C_1 is used , then $\varphi' = D \wedge C_2 \wedge \cdots \wedge C_m$ is a matched CNF.*

Proof. Let us assume that T is a regular tree-like resolution derivation of D from φ. The fact that C_1 is used in this resolution derivation implies that C_1 is a leaf clause in T. We shall proceed by induction on the structure of T. Let M be a matching for φ and let X denote the set of variables, which are matched in M. We shall preserve the following invariant:

(*) If M' is a matching for φ constructed by the proof and X' denotes the matched variables in M', then $X' = X$.

Let us at first assume that $D = C_1$, then the proposition trivially follows and invariant (*) is satisfied. Now let us suppose that $D = R(C_1, C_j)$, where $j \in \{2, \ldots, m\}$. Let us assume that $[C_1, y] \in M$. If $y \in D$, then we set $M' = (M \setminus \{[C_1, y]\}) \cup \{[D, y]\}$. If $y \notin D$, then it follows that C_1 and C_j resolve over y. Let us without loss of generality assume that y appears positively in C_1 and let us denote $C_1 = (A_1 \vee y)$ and $C_j = (A_j \vee \overline{y})$, hence $D = A_1 \vee A_j$. Then C_j is matched with another variable $z \in A_j$ in M, thus we can set $M' = (M \setminus \{[C_1, y], [C_j, z]\}) \cup \{[C_j, y], [D, z]\}$. Then M' is a matching for φ with $X' = X$.

Now let us assume that $D = R(D_1, D_2)$ where D_1 and D_2 are themselves derived by resolution derivation from φ, or they belong to φ. Suppose the resolution is over a variable z and let us denote $D_1 = (A_1 \vee z)$, $D_2 = (A_2 \vee \overline{z})$. For $i = 1, 2$ let T_i denote the subtree of T rooted at D_i, L_i the set of leaf clauses in T_i, φ_i the sub-CNF of φ formed by clauses in L_i, let M_i be the sub-matching of M on clauses from φ_i, and X_i the set of variables matched in M_i.

Let us at first assume that $C_1 \in L_1 \cap L_2$. By Claim 4.3.3 we get that $z \notin X_1 \cap X_2$. Let $i \in \{1, 2\}$ be such that $z \notin X_i$. Let us use induction hypothesis on T_i and D_i to find a matching for a formula φ'_i which is a CNF formed by clauses $(L_i \setminus \{C_1\}) \cup \{D_i\}$. By induction hypothesis φ'_i is a matched formula, let M'_i be a matching constructed for φ'_i which satisfies invariant (*), i.e. the set of matched variables in M'_i is X_i. Let x be the variable matched with D_i in M'_i. Now since

$z \notin X_i$ we have that $x \in A_i$, thus x belongs to D as well. We can now construct a matching M' for φ' by extending M'_i with pairs matching variables in $X_{3-i} \setminus X_i$ to clauses in $L_{3-i} \setminus L_i$ as in M. Moreover we replace the pair $[D_i, x]$ with the pair $[D, x]$. The result M' is a matching for φ' in which exactly the variables in X are matched and thus M' satisfies invariant (*).

In the rest of the proof we shall assume that $C_1 \in L_1 \setminus L_2$. We can use induction hypothesis on T_1 and D_1 to find a matching M'_1 for a formula φ'_1 which is again a CNF formed by clauses $(L_1 \setminus \{C_1\} \cup \{D_1\}$. By induction hypothesis M'_1 satisfies invariant (*) and thus the matched variables in M'_1 are exactly the variables in set X_1. Let x be the variable to which clause D_1 is matched in M'_1. Now there are two cases to consider .

1. If $x \in A_1$, then we can construct a matching M' for the whole formula φ by extending M'_1 with pairs matching clauses in $L_2 \setminus L_1$ to variables in $X_2 \setminus X_1$ as in M, and we replace pair $[D_1, x]$ with pair $[D, x]$. Then M' is a matching which again ties variables in X and thus it also satisfies invariant (*).

2. If $x = z$, the situation is more complicated. In this case we can observe that $z \in X_1 \setminus X_2$ ($z \notin X_1 \cap X_2$ by Claim 4.3.3, on the other hand z is matched in M_1 and M'_1). Let M'_2 be a matching for the sub-CNF φ_2 formed by clauses in L_2 which is constructed as follows. The clauses in $L_1 \cap L_2$ are matched to the same variables as in M'_1, the clauses in $L_2 \setminus L_1$ are matched to the same clauses as in M. Note that M'_2 formed in this way is really a matching, in particular C_1 does not belong to L_2 and thus it does not matter that it is not matched to any variable in M'_1. Moreover, if X'_2 denotes the set of variables matched in M'_2, then each variable in X'_2 is matched to exactly one clause. This is because M'_1 did not change anything on clauses in $L_2 \setminus L_1$ and it did not match any of the variables in $X_2 \setminus X_1$. Note, that X'_2 is not necessarily equal to X_2, because M'_1 is allowed to use variables from $X_1 \setminus X_2$ for the clauses in $L_1 \cap L_2$, but we have that $X = X'_2 \cup X_1$. We also have that z is a free variable in M'_2, that is because $z \in X_1 \setminus X_2$, thus it is not matched to any clause in L_2 in M, and because z is matched to D_1 in M'_1, thus z is not matched to any clause in $L_1 \cap L_2$ in M'_1.

   Now we have the following situation: We have a formula φ_2, we have a clause D_2 which is derived by resolution from φ_2. We have a matching M'_2 for φ_2 which ties the variables in a set X'_2. We have a variable z which is free in M'_2, thus we can use Lemma 4.3.4 to find another matching M''_2 for φ_2 and a variable y in D_2 such that z is matched to some clause in M''_2, and y is now free in M''_2 while it was matched in M'_2. Moreover for any variable $u \in X'_2 \setminus \{x, y\}$, u remains matched and any free variable except z remains free. In particular, if X''_2 denotes the set of variables matched in M''_2, then $X''_2 = (X'_2 \setminus \{z\}) \cup \{y\}$. Necessarily $z \neq y$, and thus $y \in A_2$. Now we are ready to form desired matching M' for φ.

   (a) A clause C from $L_1 \setminus (L_2 \cup \{C_1\})$ is matched to variable a in M' such that $[C, a] \in M'_1$.

   (b) A clause C from L_2 is matched to a variable a in M' such that $[C, a] \in M''_2$.

   (c) A clause D is matched to the variable y in M'.

M' defined in this way is indeed a matching, in particular if $C \in L_1 \setminus (L_2 \cup \{C_1\})$, then the matched variable a belongs to $X_1 \setminus (X_2' \cup \{z\})$, and thus a is free in both M_2' and M_2'', and it still can be used for C. Let us also observe that M' preserves invariant (*), in particular if X' denotes the set of variables matched in M', then $X' = X$. Let a be a variable from X', we shall show that $a \in X$ as well, because necessarily $|X'| = |X|$, it follows that $X' = X$.

(a) If a is matched to a clause $C \in L_1 \setminus (L_2 \cup \{C_1\})$, then this is because $[C, a] \in M_1'$ since by induction hypothesis invariant (*) is preserved for M_1' we have that $a \in X_1$ and thus $a \in X$.

(b) If a is matched to a clause $C \in L_2$, then this is because $[C, a] \in M_2''$, then $a \in X'' = (X_2' \setminus \{y\}) \cup \{z\}$. We know that $z \in X$ and that $X_2' \subseteq X$.

(c) If a is matched to D, then $a = y \in X_2' \subseteq X$.

Together we have that invariant (*) is satisfied for M'.

In any case we found a matching M' for CNF φ' which satisfies invariant (*), and the proof is finished. □

Theorem 4.3.6. *Let φ be a matched CNF representing function f, then there is a prime and irredundant representation of f which is also matched.*

Proof. This follows from Lemma 4.3.5. Firstly, we can drop any redundant clauses from φ without spoiling its matched property. If φ' is an irredundant representation of f originated from φ by dropping these redundant clauses, then we can turn it into a prime representation by using Lemma 4.3.5 as follows. If $\varphi' = C_1 \wedge \cdots \wedge C_m$, and $C_1' \subsetneq C_1$ is a prime implicate, then C_1' can be derived by a resolution derivation from φ'. Since φ' is already irredundant, in every resolution derivation of C_1' from φ' we have to use C_1. Thus by Lemma 4.3.5, formula $\varphi'' = C_1' \wedge C_2 \wedge \cdots \wedge C_m$ is also matched. In this way we can replace every clause in φ' by a prime subimplicate. Thus we obtain a prime and irredundant representation of f. □

4.4 Minimum representations of matched formulas

In the previous sections we have seen that for a matched CNF there may be some logically equivalent prime and irredundant CNFs which are not matched but always at least one such CNF which is matched. In this section we shall show that a stronger statement holds for CNFs which are not only prime and irredundant but also clause minimum. We shall prove that every clause minimum CNF representation of a matched CNF is a matched CNF.

Theorem 4.4.1. *Let φ be a matched CNF representing function f on set of variables V and let ψ be a clause minimum CNF representation of f. Then ψ is a matched CNF.*

Proof. Due to Theorem 4.3.6 we may assume that φ is prime and irredundant and thus $\varphi \subseteq \mathcal{I}(f)$. Let us assume by contradiction that ψ is not matched and that $\psi_X \subseteq \psi$ is a maximal (under inclusion) subCNF violating Hall's condition (such a subset must exist due to Theorem 4.0.3). Let us denote X the set of variables in the subCNF ψ_X, $Y = V \setminus X$ the set of remaining variables in ψ, and $\psi_Y = \psi \setminus \psi_X$ the remaining clauses in ψ (note that clauses in ψ_Y may contain variables not only form Y but also from X). Now the following holds:

- By the violation of Hall's condition we have $|\psi_X| > |X|$.
- By the maximality of ψ_X there exists a matching M of all clauses in ψ_Y to variables in Y, i.e. ψ_Y is a matched CNF even if we drop all variables in X from its clauses. This follows from the fact that every subset of ψ_Y must satisfy Hall's condition even with respect to the variables in Y, since otherwise any such violating subset could be added to ψ_X contradicting its maximality.

The existence of matching M implies that ψ_Y can be satisfied using only variables from Y (each clause can be satisfied by its matched variable). So let $t : Y \longrightarrow \{0, 1\}$ be some partial assignment satisfying all clauses in ψ_Y (t is not necessarily unique). Clearly, t is an autarky on ψ as it satisfies every clause containing an assigned literal.

It follows from Lemma 4.2.6 that $\psi[t] = \psi_X$ represents an exclusive component $f_{\mathcal{X}}$ of f defined by the exclusive set $\mathcal{X} \subseteq \mathcal{I}(f)$, which contains all clauses consisting only of variables from X. Since φ also represents f and we assumed $\varphi \subseteq \mathcal{I}(f)$, it follows from Corollary 4.2.5 that t is an autarky also on φ. Thus, similarly as for $\psi[t]$ above, we can conclude that $\varphi[t]$ is a subCNF of φ which represents the exclusive component $f_{\mathcal{X}}$ of f, i.e. $\psi[t] \equiv \varphi[t]$. However, φ is matched, so every its sub CNF (and in particular $\varphi[t]$) is matched, and thus $|\varphi[t]| \leq |X|$ while $|\psi[t]| = |\psi_X| > |X|$. But now, since both $\varphi[t]$ and $\psi[t] = \psi_X$ represent an exclusive component of f, also CNF $\psi' = (\psi \setminus \psi[t]) \cup \varphi[t]$ represents f by Corollary 4.2.3. However, we get $|\psi'| < |\psi|$ contradicting the assumed minimality of ψ. □

Chapter 5

Propagation completeness

This dissertation deals primarily with CNF minimization which is a special type of knowledge compression. In this chapter we will look at CNF compilation, a special type of knowledge compilation, which is in some sense a reverse process to compression, as it makes the input CNF longer not shorter. Knowledge compilation is a process of adding more information to a knowledge base in order to make it easier to deduce facts from the compiled knowledge base than the original one. The knowledge base after knowledge compilation does not allow deducing any more facts than before, however it can speed up deduction considerably. In our case a CNF formula can be considered to represent a knowledge base and knowledge compilation means adding clauses so that the new CNF with the added clauses is propagation complete, that is after any partial assignment all entailed literals can be inferred by unit propagation. This chapter is based on article [5] and uses large portions of it verbatim.

Let us recall bijection between partial assignments and clauses, since it will be needed. Partial assignment α that sets literals $l_2, l_2, \ldots l_n$ to true corresponds to a clause $C = \neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_n$.

Definition. *We say that literal l is logically* entailed *from formula φ ($\varphi \models l$) if any satisfying assignment of φ sets l to one. We say that l is logically entailed from φ by partial assignment α if it is entailed by $\varphi[\alpha]$.*

Note that for partial assignment α and its corresponding clause C $\varphi[\alpha] \models l$ is equivalent to $\varphi \wedge \neg C \models l$. We should note that for $C = \neg l_1 \vee \neg l_2 \vee \ldots \neg l_n$, $\varphi \wedge \neg C$ is equivalent to $\varphi \wedge l_1 \wedge l_2 \ldots \wedge l_n$.

We will use resolution where at least one of the parent clauses is a unit clause often in this chapter. Such a resolution is called unit resolution. Resolution proof where every resolution step is unit is called unit resolution proof. We will use notation $\varphi \vdash_1 C$ to mean that clause C is derived from φ by unit resolution. Unit resolution of empty clause from φ ($\varphi \vdash_1 \bot$) is called resolution refutation of φ. There is a well-known and simple connection between unit propagation and unit resolution – if $\varphi \vdash_1 l$ for any literal l (that is for unit clause consisting only of literal l) then unit propagation on φ satisfies l or derives a contradiction.

Definition (Propagation Completeness (PC) [13])**.** *A CNF φ is propagation complete if for any partial assignment α and its corresponding clause C and any literal d the following holds: If d is logically entailed from φ by α then d can be derived*

from $\varphi[\alpha]$ by unit propagation. That is if $\varphi \wedge \neg C \models d$ then $\varphi \wedge \neg C \vdash_1 d$. The PC class is a class of all propagation complete formulas.

Definition (Empowering implicate [43, 13], Absorbtion [3, 13])**.** *A clause $C = l_1 \vee l_2 \vee l_3 \vee \ldots \vee l_k$ is called an* empowering implicate *for a formula φ if it contains a literal l_i called an* empowered literal *such that $\varphi \wedge \bigwedge_{j \in 1 \ldots k, j \neq i} \neg l_j \nvdash_1 \bot$ and $\varphi \wedge \bigwedge_{j \in 1 \ldots k, j \neq i} \neg l_j \models l_i$ but $\varphi \wedge \bigwedge_{j \in 1 \ldots k, j \neq i} \neg l_j \nvdash_1 l_i$. An implicate C is called* absorbed *by φ if it has no empowered literal.*

We will also need 1-provability and unit refutation completeness.

Definition. *We will say that clause C is* 1-provable *with respect to CNF φ if $\varphi \wedge \neg C \vdash_1 \bot$.*

That C is 1-provable means that proving the fact that C is an implicate of φ can be done using unit propagation.

Definition. *A CNF φ is* unit refutation complete *if every implicate of φ is 1-provable.*

It has been shown in [13], along with other properties of PC formulas, that a formula φ is PC if and only if there is no empowering implicate for φ. However, several complexity issues directly connected to propagation completeness and empowering implicates are left open in [13]. A short list of such questions is the following:

1. Given a CNF formula φ and a clause C, what is the complexity of deciding whether C is an empowering implicate for φ?

2. Given a CNF formula φ that is not PC, how difficult is it to generate an empowering implicate for φ by resolution, where the "level of difficulty" is measured by the length of the resolution proof?

3. Given a CNF formula φ, what is the complexity of deciding whether there exists an empowering implicate for φ?

4. Given a CNF formula φ that is not PC, how many empowering implicates is it necessary to add to φ in order to make it PC?

In this chapter we tackle all of the above listed problems. In Section 5.2 we derive several simple properties of empowering implicates and in the following sections we address the following four questions as follows:

1. In Section 5.2 we show that the first problem is co-NP complete. This is not a very difficult result, however, to the best of our knowledge, it was not stated in the related literature yet.

2. In Section 5.3 we tackle the second problem. We prove that for a non-PC CNF formula with s occurences of literals there always exists a resolution proof of length $O(s)$ of some empowering implicate. On the other hand, we construct examples of CNF formulas where a resolution proof of length $\Omega(s)$ is needed for any empowering implicate, which means that $\Theta(s)$ is an asymptotically tight bound for this problem. It is important to note that the upper bound result does not require the derived empowering implicate

to be prime. We show (by a simple modification of results concerning refutation proofs [30, 52]) that there exist CNF formulas such that in order to derive any prime empowering implicate of such CNF a resolution proof of an exponential length is needed.

3. Section 5.4 contains the main results of this chapter which are connected to the third problem. It was proved in [13] that deciding about an existence of an empowering implicate is in Σ_2^p. Using the results from Section 4 we strengthen this result by showing that the problem belongs to $\Sigma_1^p = \text{NP}$. Given the equivalence between propagation completeness and non-existence of empowering implicates proved in [13], this immediately implies that testing propagation completeness belongs to co-NP. Then we proceed with the hardness proof for this problem. We present a reduction from a well known NP-complete 3-dimensional matching problem which proves that deciding for a CNF formula whether there exists an empowering implicate for it is NP-hard (and thus testing propagation completeness is co-NP-hard).

4. The fourth question is answered in Section 5.4 as well by showing that there exist CNF formulas where an exponential number (both with respect to the number of variables and the number of clauses) of empowering implicates must be added in order to arrive at a PC formula. This strengthens the superpolynomial bound which follows from a combination of results in [13] and [11] using a superpolynomial lower bound for certain monotone circuits from [46]. The connection is discussed in detail in Section 5.1.

5.1 Relation of propagation cCompleteness to constraint propagation

Propagation complete formulas play an important role in constraint propagation (for general reading on constraint propagation see Chapter 3 of [48]), where the notion appeared implicitly in earlier literature [6, 35, 10, 17, 44, 9] concentrated on CNF encodings of constraints both in general [6, 35], and for particular constraints (the sequence constraint [17], the grammar constraint [44], the REGULAR, the AMONG, and the GENERALIZED SEQUENCE constraints [35], or the ALLDIFFERENT constraint [10]). A general idea which appeared in all these papers is to take a constraint C, encode it using a CNF and then use unit propagation to maintain some kind of domain consistency (e.g. generalized arc consistency (GAC) in [6], or (relational) (i, j)-consistency in [9]). In general, these encodings are polynomial in table representation of a constraint, however in the case of a particular global constraint we can sometimes use its special properties to get a CNF representation of polynomial size with respect to the arity of a constraint with suitable properties, such as in [35] in case of the REGULAR constraint, the AMONG constraint, or the GENERALIZED SEQUENCE constraint. On the other hand in some cases this is not possible as it is shown in [10] for the ALLDIFFERENT constraint.

The idea of using unit propagation to maintain domain consistencies was formalized in [10], let us recall some of the results and the notation from [10] here. A *constraint* C is defined over a set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, each of

which has a finite domain $D(X_i)$. An *assignment* to a variable X_i is a mapping of X_i to a value $j \in D(X_i)$, called a literal and written $X_i = j$. $\mathbf{D}(\mathbf{X})$ denotes the set of literals, i.e. $\mathbf{D}(\mathbf{X}) = \{X_i = j \mid X_i \in \mathbf{X} \wedge j \in D(X_i)\}$. $\mathcal{P}(\mathbf{D}(\mathbf{X}))$ then denotes the set of all possible sets of literals. A constraint C is defined over a set of variables denoted as $scope(C) \subseteq \mathbf{X}$ (*scope* of C) and it allows a subset of the possible assignments to the variables in $scope(C)$.

Following [10, 49, 48] a propagator for a constraint C is an algorithm which takes as input the domains of variables in $scope(C)$ and returns restrictions of these domains. Formally a propagator for a constraint C can be defined as follows (definition is taken from [10], where they followed [49], see also Chapter 3 of [48]).

Definition (Definition 1 in [10])**.** *A* propagator f *for a constraint* C *is a polynomial time computable function* $f : \mathcal{P}(\mathbf{D}(\mathbf{X})) \mapsto \mathcal{P}(\mathbf{D}(\mathbf{X}))$*, such that* f *is* monotone*, i.e.* $\mathbf{D}'(\mathbf{X}) \subseteq \mathbf{D}(\mathbf{X}) \Rightarrow f(\mathbf{D}'(\mathbf{X})) \subseteq f(\mathbf{D}(\mathbf{X}))$*,* contracting*, i.e.* $f(\mathbf{D}(\mathbf{X})) \subseteq \mathbf{D}(\mathbf{X})$*, and* idempotent*, i.e.* $f(\mathbf{D}(\mathbf{X})) = f(f(\mathbf{D}(\mathbf{X})))$*. If* f *detects that* C *has no solutions under* $\mathbf{D}(\mathbf{X})$ *then* $f(\mathbf{D}(\mathbf{X})) = \emptyset$*.*

We say that a propagator *detects dis-entailment* if $f(\mathbf{D}(\mathbf{X})) = \emptyset$ whenever C has no solution. A propagator enforces *domain consistency (DC)* when $X_i = j \in f(\mathbf{D}(\mathbf{X}))$ implies that there exists a solution of C that contains $X_i = j$. Other consistencies (such as GAC [6] which is equivalent to DC, or (i, j)-consistency [9]) can be considered as well, we mention domain consistency here because later on we shall recall the results of [10] on domain consistency propagators.

Let us recall a definition of a CNF decomposition of a propagator as it appeared in [10]. Before that let us recall a general way how a constraint satisfaction program (CSP) variables with multiple valued domains are usually encoded within a CNF. Given a variable $X_i \in \mathbf{X}$ with domain $D(X_i)$ we encode it with a set of Boolean variables $x_{i,j}$, $X_i \in \mathbf{X}$, $j \in D(X_i)$ such that $X_i \neq j \Leftrightarrow \overline{x}_{i,j}$. The property that a CSP variable X_i has at most one value is enforced by the set of *AMO* (i.e. *at most one*) clauses $(\overline{x}_{i,j} \vee \overline{x}_{i,k})$ for all $j, k \in D(X_i)$, $k \neq j$ and the property that it has at least one value is enforced by the *ALO* (i.e. *at least one*) clause $\bigvee_{j \in D(X_i)} x_{i,j}$. Following [10] we call the above described propositional representation of $\mathbf{D}((X))$ (i.e. set of all AMO and ALO clauses over all CSP variables) a *direct encoding* and denote it as $\mathbf{D}^{sat}(\mathbf{X})$.

Definition (Definition 4 in [10])**.** *A CNF decomposition of a propagation algorithm (propagator)* f_P *is a formula in CNF* φ_P *over variables* $\mathbf{x} \cup \mathbf{y}$ *such that*

- *The* input variables $\mathbf{x}$ *are the propositional representation* $\mathbf{D}^{sat}(\mathbf{X})$ *of* $\mathbf{D}(\mathbf{X})$ *and* $\mathbf{y}$ *is a set of* auxiliary variables *whose size is polynomial in* $|x|$*.*
- $x_{i,j}$ *is set to 0 by a unit propagation if and only if* $X_i = j \notin f_P(\mathbf{D}(\mathbf{X}))$*.*
- *Unit propagation on* φ_P *produces the empty clause when* $f_P(\mathbf{D}(\mathbf{X})) = \emptyset$*.*

Before further discussion, let us look at an example from [10].

Example 5.1.1 (Example 1 in [10])**.** *Consider a* Table *constraint over the variables* X_1, X_2 *with* $D(X_1) = D(X_2) = \{a, b\}$ *and the satisfying assignments:* $\{\langle a, a\rangle, \langle b, b\rangle, \langle a, b\rangle\}$*. Using encoding introduced in [6] we can decompose a* Table

constraint into the following CNF φ_T:

$$\begin{aligned}\varphi_T \;=\; & (\overline{x}_{1a} \vee y_1 \vee y_3) \wedge (\overline{x}_{2a} \vee y_1) \wedge (\overline{y}_1 \vee x_{1a}) \wedge (\overline{y}_1 \vee x_{2a}) \\ & \wedge (\overline{x}_{1b} \vee y_2) \wedge (\overline{x}_{2b} \vee y_2 \vee y_3) \wedge (\overline{y}_2 \vee x_{1b}) \wedge (\overline{y}_2 \vee x_{2b}) \\ & \wedge (\overline{y}_3 \vee x_{1a}) \wedge (\overline{y}_3 \vee x_{2b}) \wedge (y_1 \vee y_2 \vee y_3)\end{aligned}$$

Here $\mathbf{y} = \{y_1, y_2, y_3\}$ consists of auxiliary variables corresponding to the three possible solutions to the Table *constraint (y_1 corresponds to $\langle a, a\rangle$, y_2 to $\langle b, b\rangle$, and y_3 to $\langle a, b\rangle$). Suppose the value a is removed from the domain of X_1. The assignment $x_{1a} = 0$ forces the variable y_1 to 0, which in turn causes the variable x_{2a} to 0, removing the value a from the domain of X_2 as well.*

Now let us generalize the ideas presented in the above example. Let us consider a constraint C, its consistency propagator f_P, and a CNF decomposition of f_P via a CNF φ_P.

- By definition, if f_P detects dis-entailment, then $f_P(\mathbf{D}(\mathbf{X})) = \emptyset$ whenever C restricted to $\mathbf{D}(\mathbf{X})$ admits no solution. Passing $\mathbf{D}(\mathbf{X})$ to f_P corresponds to setting all values $x_{i,j} = 0$ for $X_i = j \notin \mathbf{D}(\mathbf{X})$. The third condition of Definition 5.1 thus requires that after this partial assignment φ_P is not only unsatisfiable but this fact can be detected by unit propagation. In particular we require that for any $\mathbf{D}(\mathbf{X})$:

  $$\varphi_P \wedge \bigwedge_{i,j:X_i=j\notin\mathbf{D}(\mathbf{X})} \neg x_{i,j} \models \bot \Leftrightarrow \varphi_P \wedge \bigwedge_{i,j:X_i=j\notin\mathbf{D}(\mathbf{X})} \neg x_{i,j} \vdash_1 \bot$$

  What we in fact require here is that φ_P is unit refutation complete with respect to the partial assignments to the input variables $\mathbf{x}$. Although we admit here only assignments to 0, the direct encoding clauses $\mathbf{D}^{sat}(\mathbf{X})$ allow us to use 1 as well (assigning $x_{i,j} = 1$ forces $x_{i,k} = 0$ for any $k \neq j$, $k \in D(X_i)$ by unit propagation on $\mathbf{D}^{sat}(\mathbf{X})$).

- If f_P is a domain consistency propagator then there is no solution to C containing $X_i = j$ if and only if $X_i = j \notin f(\mathbf{D}(\mathbf{X}))$. Using the second condition from Definition 5.1 it corresponds to the fact that $\overline{x}_{i,j}$ is a unit implicate of φ_P under partial assignment given by $\mathbf{D}(\mathbf{X})$ if and only if $x_{i,j}$ is forced to 0 by unit propagation on φ_P under this partial assignment. In particular we require that for any $\mathbf{D}(\mathbf{X})$ and any $x_{i,j}$:

  $$\varphi_P \wedge \bigwedge_{i',j':X_{i'}=j'\notin\mathbf{D}(\mathbf{X})} \neg x_{i',j'} \models \neg x_{i,j} \Leftrightarrow \varphi_P \wedge \bigwedge_{i',j':X_{i'}=j'\notin\mathbf{D}(\mathbf{X})} \neg x_{i',j'} \vdash_1 \neg x_{i,j}$$

  What we in fact require here is the fact that φ_P is propagation complete with respect to partial assignments and literals on the input variables in $\mathbf{x}$. In the above equation it is enough to consider only negative literals $x_{i,j}$ due to presence of clauses of direct encoding $\mathbf{D}^{sat}(\mathbf{X})$.

The author of [6] suggests that a canonical CNF decomposition (i.e. CNF consisting of all prime implicates) is sufficient to encode both dis-entailment detecting propagator and domain consistency propagator. The above discussion shows that a shorter CNF is sufficient, in particular unit refutation completeness

is sufficient for detecting dis-entailment and propagation completeness is sufficient for domain consistency propagator. This can be naturally generalized to other consistencies such as (i, j)-consistency [9].

It was shown in [11] that a polynomial sized decomposition of a consistency propagator f_P exists if and only if it can by computed by a monotone circuit of polynomial size. This result was used to derive the following corollary:

Corollary 5.1.2 (Corollary 4 in [11])**.** *There is no polynomial sized CNF decomposition of any* AllDifferent *domain consistency propagator.*

On the other hand we can get a polynomial sized CNF encoding of the AllDifferent constraint. Using such an encoding and Corollary 5.1.2 it follows that no propagation complete encoding of the AllDifferent constraint can have polynomial size. There are thus formulas such that any equivalent propagation complete formula has superpolynomial size. Let us have a more detailed look at the proof of Corollary 5.1.2 in [11]. It was shown in in [47] that AllDifferent constraint has a solution if and only if the corresponding bipartite value graph has a perfect matching. From the aforementioned connection to monotone circuits proved in [11] it follows that based on an AllDifferent domain consistency propagator we can construct a monotone circuit that computes whether a bipartite graph has a perfect matching and such a circuit has a polynomial size with respect to the domain consistency propagator we start with. The proof is finished using an older result of Razborov [46] according to which the size of monotone circuit computing whether there is a perfect matching in a bipartite graph G on n vertices has size at least $n^{\Omega(\log n)}$, which is a superpolynomial (in some literature called quasi-polynomial) but not an exponential bound. In Section 5.4 we strengthen the result of Corollary 5.1.2 for propagation complete formulas by showing that there are in fact formulas to which an exponential number of implicates have to be added in order to make them propagation complete.

5.2 Properties of empowering implicates

Let us start this section with a discussion on the connection between unit refutation completeness introduced in [23] and propagation completeness introduced in [13]. First let us recall that a formula φ is unit refutation complete if for every implicate C of φ we have that $\varphi \wedge \neg C \vdash_1 \bot$. This means that the fact that C is an implicate of φ can be proved using just unit resolution. It is not hard to see that if a formula φ is propagation complete, then it is unit refutation complete as well. Indeed, if $C = (l_1 \vee \cdots \vee l_k)$ is an implicate of φ, then either $\varphi \wedge \bigwedge_{i=1}^{k-1} \neg l_i \vdash_1 \bot$, or $\varphi \wedge \bigwedge_{i=1}^{k-1} \neg l_i \vdash_1 l_k$ which implies $\varphi \wedge \bigwedge_{i=1}^{k} \neg l_i \vdash_1 \bot$. On the other hand, it is not true that every unit refutation complete formula is also propagation complete. Consider the following formula φ:

$$\varphi = (\overline{x} \vee a \vee b \vee c) \wedge (x \vee a \vee b \vee d).$$

This is a prime CNF which has only one more prime implicate $C = (a \vee b \vee c \vee d)$ produced by resolving the two clauses in φ. We can observe that φ is unit refutation complete. If we add negation of C to φ, then unit resolution of $\overline{x}$ and x gives us the empty clause. However φ is not propagation complete and C is an

empowering implicate of φ with empowered literal a (or b). Indeed, if we set b, c, and d to false in φ, we get CNF $(\overline{x} \vee a) \wedge (x \vee a)$ from which it is not possible to derive a just using unit resolution.

It follows from the above discussion that the set of propagation complete CNFs is a proper subset of unit refutation complete CNFs. The aim of this section is to recall several results about the class of unit refutation complete CNFs from [23] and observe that some of these results are in fact true already for the set of propagation complete formulas.

We shall start with showing that adding a literal to a non-empowering implicate cannot make it empowering with respect to any of the original literals.

Lemma 5.2.1. *Let φ be a CNF formula and let $C = l_1 \vee \cdots \vee l_k$ be an implicate of φ which is not empowering. Let A be a clause and let $l_i \in C$ be an arbitrary literal. Then $C \vee A$ is not empowering implicate of φ with empowered literal l_i.*

Proof. Let us assume without loss of generality that $i = k$ (if not, then we can achieve this by renaming the variables). By definition, the fact that C is not empowering with empowered literal l_k implies that either $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \vdash_1 \bot$, or $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \vdash_1 l_k$ because C is an implicate of φ and thus $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \models l_k$. Since $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j$ is a subformula of $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \wedge \bigwedge_{a \in A} \neg a$, what we can derive by unit propagation from the former formula, can be derived from latter one as well. Thus we have that $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \wedge \bigwedge_{a \in A} \neg a \vdash_1 \bot$ or $\varphi \wedge \bigwedge_{j=1}^{k-1} \neg l_j \wedge \bigwedge_{a \in A} \neg a \vdash_1 l_k$. This means that by definition, $C \vee A$ is not empowering implicate with empowered literal l_k. □

Note that in the previous proposition we cannot argue that $A \vee C$ is not empowering because it could be empowering with an empowered literal from A, e.g. if A would itself be an empowering clause. That is why we consider only literals in C. Using Lemma 5.2.1, we can easily show that among empowering implicates the prime implicates are the only ones we need to consider.

Lemma 5.2.2. *Let φ be a nonempty satisfiable CNF formula (i.e. it has at least one nonempty clause) and let C be an empowering implicate for φ. Then any implicate C' of φ subsuming C is empowering for φ (this in particular includes the case when C' is prime).*

Proof. Let C' be an arbitrary implicate subsuming C and let us assume that $C' \neq C$. If C' is not empowering, then $\varphi \wedge \neg C' \vdash_1 \bot$ and thus C cannot be empowering with respect to a literal a, which is not in C'. On the other hand, C cannot be empowering with respect to a literal in C' as well due to Lemma 5.2.1. □

As an easy corollary we now get that a canonical CNF formula is always propagation complete, although this can be easily deduced from properties of canonical CNFs as well.

Now let us consider the problem of generating an empowering implicate for a given CNF formula. A natural method to consider is to generate an empowering implicate by the resolution procedure. In [23] it is shown that non-merge resolution cannot produce an empowering implicate with respect to unit refutation completeness. In case of propagation completeness the same is true. It follows from discussion at the beginning of Section 5 in [13]. We shall formulate this proposition as a lemma to be able to reference to it later.

Lemma 5.2.3. *Let φ be a CNF formula and let C be produced from φ by a series of non-merge resolutions. Then C is not an empowering implicate of φ.*

Keeping in mind that a canonical formula is propagation complete, it follows that a CNF formula φ satisfying that every prime implicate of φ is either present in it or it can be derived from φ by a series of non-merge resolutions is always propagation complete. This property was already shown in form of Theorem 1 in [23] for unit refutation completeness and it is an easy corollary of arguments about non-merge resolution in [13] (here stated as Lemma 5.2.3) that it is true also for PC formulas.

Since PC formulas allow easy inference, it is interesting to investigate classes which are contained in the class of PC CNF formulas. One such example is given by the following theorem which shows the desired property for the class of prime quadratic CNF formulas.

Theorem 5.2.4. *If φ is a prime quadratic CNF formula, then it is propagation complete.*

Proof. If φ is not satisfiable, then the proposition of the theorem is trivial. Let us assume that φ is a satisfiable prime quadratic CNF formula.

By Lemma 5.2.2 it is enough to consider prime implicates as candidates for empowering implicates. Because φ is a prime CNF formula, it must contain all the unit prime implicates. Hence, if any other prime implicate should be added to φ to make it propagation complete, it must be a quadratic clause which is produced by resolving two other quadratic clauses. It is a simple observation that these resolutions have to be non-merge. Thus by Lemma 5.2.3 we have that φ must already be propagation complete. □

The following Lemma shows that the primeness assumption in the statement of Theorem 5.2.4 is necessary.

Lemma 5.2.5. *There is a (nonprime) quadratic CNF formula which is not PC.*

Proof. Let us consider the following CNF formula:

$$\varphi = (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b})$$

φ is clearly a quadratic CNF formula. On the other hand, φ is not PC because $\varphi \models \overline{a}$, but $\varphi \not\vdash_1 \overline{a}$. □

Now let us turn our attention to the complexity of testing if a given clause C is an empowering implicate of a CNF formula φ. We shall denote this problem as IsEmpowering(φ, C). Note that co-NP completeness of IsEmpowering(φ, C) comes as no surprise since it is in essence very similar to another co-NP complete problem IsImplicate(φ, C). The hard part of checking whether given clause C is an empowering implicate of φ is in fact checking whether C is an implicate of φ at all. Thus the co-NP completeness of IsEmpowering(φ, C) is a direct consequence of co-NP completeness of IsImplicate(φ, C). The proof of the following theorem only formalizes this idea.

Theorem 5.2.6. *The problem* IsEmpowering(φ, C) *is co-NP complete.*

Proof. To show that a problem is in co-NP it suffices to have for every negative instance of the problem a polynomially verifiable certificate which allows to verify that the answer is no. We can distinguish two cases when a pair (φ, C) forms a negative instance of IsEmpowering. In the first case C is not even an implicate of φ and the desired certificate is then an assignment of truth values to the variables which satisfies φ and falsifies C. The second case is when C is an implicate of φ but not an empowering one. In this case an empty certificate is good enough because one needs no additional information to be able to check in polynomial time that no literal in C is empowered. This can be done by running unit propagation and checking for every literal ℓ in C that $\varphi \wedge \bigwedge_{\ell' \in C, \ell' \neq \ell} \neg \ell' \vdash_1 \bot$ or $\varphi \wedge \bigwedge_{\ell' \in C, \ell' \neq \ell} \neg \ell' \vdash_1 \ell$.

For the co-NP hardness we reduce the co-NP complete problem IsImplicate to IsEmpowering. Let (φ, C) be an arbitrary instance of IsImplicate. We start by a simple preprocessing step in which we run unit propagation to test whether $\varphi \wedge \bigwedge_{\ell \in C} \neg \ell \vdash_1 \bot$. If yes, then C is an implicate of φ (in fact a 1-provable implicate), i.e. (φ, C) is a positive instance of IsImplicate, and the reduction algorithm can terminate by answering yes. Note that this case includes the situation when C is a clause in φ. If no, i.e. if $\varphi \wedge \bigwedge_{\ell \in C} \neg \ell \nvdash_1 \bot$, we define an instance (φ', C') of IsEmpowering by $\varphi' = \varphi \wedge (x \vee y) \wedge (x \vee \overline{y})$ and $C' = C \vee \overline{x}$, where x and y are two new variables not appearing in (φ, C). We shall show that C is an implicate of φ if and only if C' is an empowering implicate of φ' with $\overline{x}$ as the empowered literal.

Let C be an implicate of φ. C is clearly an implicate of φ', and hence also C' is an implicate of φ'. To see that C' is empowering with $\overline{x}$ as the empowered literal recall that $\varphi \wedge \bigwedge_{\ell \in C} \neg \ell \nvdash_1 \bot$ and thus $\varphi' \wedge \bigwedge_{\ell \in C} \neg \ell \nvdash_1 \overline{x}$ because unit propagation does nothing on $(x \vee y) \wedge (x \vee \overline{y})$ and neither x nor y appear in φ and C.

Let C' be an empowering implicate of φ' with $\overline{x}$ as the empowered literal. The fact that C' is an implicate of φ' means that any assignment that falsifies C' must also falsify φ'. However, any assignment which falsifies C' sets x to 1, falsifies C and satisfies $(x \vee y) \wedge (x \vee \overline{y})$ regardless of the value of y. Thus, to falsify φ' it must falsify φ. Therefore any assignment which falsifies C must also falsify φ, which means that C is an implicate of φ. □

Let us recall the notion of a tied chain in a CNF formula used in [23] in case of unit refutation completeness.

Definition ([23], introduced in [25])**.** *A* tied chain *in a CNF formula φ is a sequence of triples $(x_1, C_1, y_1), (x_2, C_2, y_2), \ldots, (x_n, C_n, y_n)$ such that:*

- *For $1 \leq i \leq n$, C_i is a clause in φ and x_i, y_i are two different literals in C_i (i.e. $x_i \neq y_i$).*
- *For $1 \leq i \leq n-1$, we have that y_i and x_{i+1} are complementary literal (called* link literals *of the chain).*
- *$x_1 = y_n$ is called the* tied literal *of the chain.*

For example CNF formula $\varphi = (p \vee q \vee r) \wedge (\overline{r} \vee s) \wedge (\overline{s} \vee p)$ contains a tied chain with p as tied literal. In [25] it is shown that the absence of tied chains is a sufficient condition for unit refutation completeness. The following lemma was shown in [23] as Lemma 6.

Lemma 5.2.7 (Lemma 6 of [23]). *Let C be an implicate of a CNF formula φ which is produced from φ by a resolution proof D in which the last resolution made is a merge resolution. Let us assume that the parent clauses of C in D are C_1 and C_2, where M denotes the set of common literals in C_1 and C_2. $M \neq \emptyset$ since the last step in D is a merge resolution and every literal of M is contained in C. Then φ contains, for each literal $\ell \in M$, a tied chain T_ℓ with ℓ as its tied literal. Furthermore, each link literal in T_ℓ has a clause in D which is produced from its parent clauses by resolution upon ℓ.*

It is argued in [23] that if there are no tied chains in a CNF formula φ, there can be no merge resolutions and thus the formula φ has to be unit refutation complete. The same argument can be used for propagation completeness. We can argue in the same way using Lemma 5.2.3 that absence of tied chains in a CNF formula φ implies that φ is propagation complete. We shall use this property in proofs in the text and thus we shall formulate it as a lemma to be able to reference to it.

Lemma 5.2.8. *If a CNF formula φ does not contain tied chains, then it is propagation complete.*

5.3 Resolution derivations of empowering implicates

We have seen in Theorem 5.2.6 that it is hard to check whether a given clause C is an empowering implicate of a given formula φ. The hard part of this test is to check whether C is in the set S of all implicates of φ. The core of the proof of Theorem 5.2.6 shows that considering a smaller set $S' \subseteq S$ of all empowering implicates of φ does not make this test easier.

We shall show now that the hard part of the test can be in some sense avoided by considering a suitable enlargement of the tested clause. In particular, we shall show that if a clause C is an empowering implicate of a CNF formula φ, we can always extend C by adding suitable literals to obtain clause C' which is still empowering and moreover we can check that C' is an implicate of φ simply by unit resolution. Let us recall that such a clause is called 1-provable (which is a notion introduced in [43]).

Proof of Theorem 5.3.2 is significantly based on Proposition 5.3.1. The proposition is given in [43] and may be restated as follows.

Proposition 5.3.1 (Proposition 2 of [43]). *Let ψ be an unsatisfiable CNF such that $\psi \nvdash_1 \bot$ and Π be its resolution refutation. Then there exists a clause $C_\psi \in \Pi$ which is both empowering and 1-provable.*

Let C be an empowering implicate of a formula φ which is not 1-provable. This means that if we falsify all the literals in C and add them to φ, thus producing $\psi = \varphi \wedge \neg C$, then we get a contradiction which is not provable by unit resolution. By using the previous proposition we show that we may add the clause C to the obtained clause C_ψ to obtain a clause which is both 1-provable and empowering.

The following Theorem 5.3.2 is a consequence of Proposition 5.3.1. For readers familiar with CDCL SAT solvers, the idea remains the same as in the previous

paragraph. By falsifying the literals in C any CDCL SAT solver must derive a contradiction. We add the solver's decisions to the input clause C, i.e. we add $\neg C$ and the conjunction of literals corresponding to each assigned variable to the input formula. After such an addition, we obtain the desired 1-provable and empowering clause. As we shall show later in Proposition 5.3.3 we can even derive such a clause C using only linear number of resolution steps with respect to the number of literal occurrences in φ.

Theorem 5.3.2. *Assume that C is an empowering implicate of a formula φ which is not 1-provable. Then there is an implicate C' of φ such that $C \subset C'$ and C is both 1-provable and empowering.*

Proof. Let $\psi = \varphi \wedge \neg C$ and Π be a resolution refutation of ψ. Because C is not 1-provable for φ it holds that $\psi \nvdash_1 \bot$. Then according to Proposition 5.3.1 there is a clause $C_\psi \in \Pi$ which is both empowering and 1-provable with respect to ψ. We consider a clause $C_\varphi = C_\psi \vee C$.

First, C_φ is 1-provable for φ because C_ψ is 1-provable for ψ and all the possibly required literals were added to C_φ. It follows from the following obvious chain of equivalence:

$$\begin{aligned} &C_\psi \text{ is 1-provable for } \psi \Leftrightarrow \psi \wedge \neg C_\psi \vdash_1 \bot \Leftrightarrow \varphi \wedge \neg C \wedge \neg C_\psi \vdash_1 \bot \Leftrightarrow \\ &\varphi \wedge \neg C_\varphi \vdash_1 \bot \Leftrightarrow C_\varphi \text{ is 1-provable for } \varphi. \end{aligned} \tag{5.1}$$

Let ℓ be an empowering literal of C_ψ for ψ. Then ℓ is trivially also an empowered literal of C_φ for φ. The required properties for unit resolution and entailment come from the definitions of C_φ and ψ using similar chain of equivalences as in 5.1.

Thus C_φ is the desired clause since it is both empowering and 1-provable with respect to φ. □

The following proposition shows that not only can we find a 1-provable and empowering implicate as in Theorem 5.3.2 but we can also derive some empowering implicate by a resolution derivation of linear length with respect to the number of literals occurring in given formula. The proof is based on ideas presented in [43] and [42, 7, 54] in the context of CDCL SAT solvers. More detailed discussion about this connection is presented just after the proof of the proposition.

Proposition 5.3.3. *Let φ be a formula on n variables which is not propagation complete and s be the size of the CNF representation of φ (i.e. s is the total number of occurences of literals in φ). Then there is an empowering implicate C of φ which can be derived by a series of resolutions of length at most s from φ*

Proof. From Theorem 5.3.2 it follows that there is an empowering 1-provable implicate C_φ of φ with an empowered literal ℓ. Let φ' denote the formula which originates from φ after adding unit clauses formed by negated literals from C_φ, i.e. $\varphi' \equiv \varphi \wedge \neg C_\varphi$. Since C_φ is 1-provable it follows that $\varphi' \vdash_1 \bot$, i.e. we can derive contradiction from φ' by using only unit resolution.

Due to the nature of unit resolutions, we can assume that the unit refutation proceeds in two phases.

1. In the first phase we take all unit clauses from $\neg C_\varphi$ and perform unit resolutions only over the variables from C_φ.

2. In the second phase we continue with refutation proof without using unit clauses from $\neg C_\varphi$ and we do not resolve over variables from C_φ at all.

In case of unit resolutions if we already have some unit clauses it does not matter in which order we make unit resolutions over them. Thus we may assume without loss of generality that unit clauses from $\neg C_\varphi$ are used first. The first phase thus corresponds to performing partial assignment to variables of φ which falsifies literals in C_φ.

Let us now assume that $D'_1, \ldots, D'_m = \bot$ is a unit refutation proof which proceeds in the above two phases. Since it is a unit refutation proof, it can be observed that $m \leq s$. Let us assume that the first phase is formed by clauses $D'_1, \ldots, D'_{k'-1}$ and the second phase is formed by clauses $D'_{k'}, \ldots, D'_m$. Each clause among $D'_1, \ldots, D'_{k'-1}$ is therefore either a clause in φ' or it originates from two preceding clauses by unit resolution over variable from C_φ. Similarly each clause among $D'_{k'}, \ldots, D'_m$ is either an original clause from φ or it originates from two preceding clauses by unit resolution over a variable which is not in C_φ. Observe that if the resolution proof is irredundant, i.e. no clause can be dropped from it, no clause among $D'_{k'}, \ldots, D'_m$ contains a variable from C_φ. This is because in the end we arrive at an empty clause and there is no way to remove a variable from C_φ in the second phase.

Let us now consider the situation in which we do not proceed with the first phase, in particular if we replace the first phase only with a list of corresponding clauses from φ. In this case the second phase can proceed as before (except that unit resolution steps are replaced by general resolution steps) only now the input clauses of phase 2 may contain some literals from C_φ as these were not removed in missing phase 1. These literals propagate down to the end of the proof and $\bot$ now becomes a subclause of C_φ. In this way we shall obtain a resolution proof of a clause $C' \subseteq C_\varphi$. Note that ℓ will be present in C' because otherwise the original unit refutation would actually prove that $\varphi \wedge \neg(C_\varphi \setminus \{\ell\}) \vdash_1 \bot$ which would be in contradiction with the fact that C_φ is an empowering implicate with empowered literal ℓ. Thus C' will be empowering by Lemma 5.2.1.

Let us now formalize the above idea. If D'_j is a clause among the clauses $D'_1, \ldots, D'_{k'-1}$, then it is either a unit literal from $\neg C_\varphi$ or there is a clause $D_j \in \varphi$, such that D'_j originates from D_j by falsifying some literals from C_φ. If D'_j is later used in the second phase of the unit refutation proof, the latter is the case and then D'_j originates from some clause $D_j \in \varphi$ by falsifying all literals of C_φ which appear in D_j.

If D'_j is among $D'_{k'}, \ldots, D'_m$ then we shall define clause D_j as follows. If D'_j is an original clause from φ, then $D_j = D'_j$, if $D'_j = R(D'_a, D'_b)$ where $1 \leq a, b < j$, then we define $D_j = R(D_a, D_b)$. Note that if D'_a and D'_b were two resolvable clauses, then the same is true about D_a and D_b. This is because if D_a contains more literals than D'_a these literals are from C_φ and the same is true for D_b and D'_b.

In the end we get a resolution derivation of length at most $m \leq s$ of clause $C = D_m$ which is a subclause of C_φ. As we have already mentioned ℓ must be present in C. Otherwise we would get that it was not necessary to use ℓ to derive $\bot$ from φ' which would be in contradiction with the fact that C_φ is an empowering implicate. According to Lemma 5.2.1 C is an empowering implicate. In particular, if C would not be empowering, then by Lemma 5.2.1 C_φ would not

be empowering with empowered literal ℓ. □

It is also possible to prove Proposition 5.3.3 by analysing a run of a CDCL SAT solver. Let us describe the sketch of such proof, the precise definitions of the below mentioned properties can be found in [42, 7, 54]. Consider the situation during the run of a CDCL SAT solver solving φ when the partial assignment satisfies $\neg C_\varphi$. Clearly then unit propagation is able to derive a conflict. By Proposition 3 of [7] each conflict clause can be derived by a trivial resolution derivation of length at most s. From Proposition 2 of [42] it follows that each asserting clause is also empowering. Since for each conflict there is at least one asserting clause, e.g. 1-UIP [54], then we have found an empowering clause with a short resolution proof from φ.

As the example in the following proposition shows, the linear upper bound on the length of resolution derivation of an empowering implicate is tight up to a multiplicative constant.

Theorem 5.3.4. *For each n there is a formula on $2n+1$ variables with size $O(n)$ such that it is not propagation complete, but resolution derivation of length n is needed to find an empowering implicate.*

Proof. This is actually a very simple corollary to Lemma 5.2.8. It is enough to construct a tied chain of length n. The following formula (for given n) contains such a chain:

$$\begin{aligned}\varphi_n \;=\; & (z \vee A_1 \vee a_1) \wedge (\overline{a_1} \vee A_2 \vee a_2) \wedge \ldots \wedge (\overline{a_{n-3}} \vee A_{n-2} \vee a_{n-2}) \\ & \wedge (\overline{a_{n-2}} \vee A_{n-1} \vee a_{n-1}) \wedge (\overline{a_{n-1}} \vee A_n \vee z).\end{aligned}$$

Clearly both the number of variables and the number of clauses are linear in n. First, let us observe that $(A_1 \vee A_2 \vee \cdots \vee A_n \vee z)$ is an empowering implicate with empowered literal z. This is because by falsifying all literals $A_1, \ldots, A_n$ we get a quadratic formula which has z as a unit implicate, but unit resolution cannot derive this fact. Since the only tied chain in φ_n is composed by all the n clauses in φ_n, by Lemma 5.2.8 we need all these clauses in order to derive an empowering implicate. □

Note also that formula φ_n is anti-Horn (i.e. every clause contains at most one negative literal) and thus the Theorem 5.3.4 holds also when restricted to anti-Horn or Horn formulas as we observe in the following corollary.

Corollary 5.3.5. *For each n there is a Horn formula on $2n+1$ variables with n clauses that it is not propagation complete but resolution derivation of length n is needed to find an empowering implicate.*

Proof. The formula φ_n in the previous theorem had at most one negative literal in each clause. Therefore, switching all literals to their complement in φ_n creates a Horn CNF formula with the desired property. □

The first idea which comes to mind when trying to find an empowering implicate of a CNF formula φ is to run resolution until one such implicate is generated. Of course, if φ already is propagation complete, it might be necessary to find all prime implicates before we can claim φ is propagation complete. On the other hand, Proposition 5.3.3 suggests that in some cases, we can find an empowering implicate relatively quickly in this way. However, the obtained empowering implicate is not guaranteed to be prime. So it is natural to ask what is the necessary

length of resolution derivations of prime empowering implicates. As the following observation shows, when seeking to find a prime empowering implicate all the hardness results about resolution refutations apply in this case.

Lemma 5.3.6. *Let φ be a CNF formula and let x be a new variable not appearing in φ. Then $\varphi \models \bot$ if and only if $\varphi \vdash_1 \bot$ or x is the only prime empowering implicate of $(\varphi \vee x)$. Moreover, if $\varphi \nvdash_1 \bot$, then there is a one to one correspondence between the resolution refutations of φ and resolution derivations of x from $(\varphi \vee x)$.*

Proof. Let us denote φ' the CNF formula equivalent to $\varphi \vee x$. Note that φ' can be obtained from φ by adding literal x to every clause.

Let us at first suppose that $\varphi \models \bot$. In this case $\varphi' \equiv x$ and thus x is the only prime implicate of φ'. Let us assume that x is not an empowering implicate of φ', thus $\varphi' \vdash_1 x$. Let $D'_1, \ldots, D'_k$ be a unit resolution derivation of x from φ'. Let us now denote D_i the clause D'_i with literal x removed. It is clear that $D_1, \ldots, D_k$ is now a unit resolution refutation of φ and thus $\varphi \vdash_1 \bot$.

Now let us assume that $\varphi \vdash_1 \bot$ or x is the only prime empowering implicate of φ'. In the former case trivially $\varphi \models \bot$. In the latter case observe that φ is equivalent to φ' with x assigned to 0. Since x is an implicate of φ' it implies that $\varphi \models \bot$. Moreover, the resolution proof of x from φ' immediately gives resolution proof of $\bot$ from φ.

The one to one correspondence is immediately seen from the above arguments. □

Lemma 5.3.6 is an easy observation which shows that all results about complexity of resolution refutations of CNF formulas can be repeated for resolution derivations of prime empowering implicates as well. There are many results that can be used in this context, let us mention at least some of them. In [30] (and in many papers and books that followed) it was shown that pigeon hole principle formulas on $n(n+1)$ variables and $O(n^3)$ clauses (PHP_n) have minimal resolution refutation of size c^n for some $c > 0$. If a formula PHP_n is used with Lemma 5.3.6, we get immediately that every resolution derivation of the single prime empowering implicate x from $(PHP_n \vee x)$ must have superpolynomial length as well. This is in contrast with Proposition 5.3.3 in which we showed that for a formula which is not propagation complete there always exists some empowering implicate that can be generated using only number of resolutions that is linear in the length of the formula i.e. $O(n^4)$ for $(PHP_n \vee x)$. Of course, in case of $(PHP_n \vee x)$ such an implicate would not be prime.

Note that in [13] the formula $(PHP_n \vee x)$ was used in Section 4.2 as an example of a formula in which we can generate superpolynomially many empowering implicates while the only meaningful empowering implicate is x. In this sense Lemma 5.3.6 can be viewed as a simple generalization of their example, where by meaningful implicates we now consider prime implicates.

The lower bound on the length of a resolution refutation of a PHP_n formula is only superpolynomial with respect to the length of the formula. Examples of formulas on $\Theta(n)$ variables consisting of $\Theta(n)$ clauses were given in [52] for which the lower bound on the length of a minimal resolution refutation is truly exponential.

Although we used Lemma 5.3.6 for general resolution in our example, it is more general than that, it shows that in fact any hardness result about resolution refutations can be used for similar results about resolution derivations of a prime empowering implicate. Thus we can consider formulas which require exponential tree resolution refutations though shortest general resolution refutations have only polynomial number of steps [8]. There are other resolution refinements inbetween tree and general resolution we can take into account as well [18]. Similarly, we can use results of [53] which show formulas with resolution refutations requiring almost linear depth, it can be observed that these formulas are even Horn. There are many other results about resolution refutations which we have omitted but all of them could be used for claims about resolution derivations of prime empowering implicates as well.

5.4 Hardness of generating an empowering implicate

In this section we prove that testing whether a given CNF formula has an empowering implicate is an NP-complete problem. We start by showing that it is in NP.

Lemma 5.4.1. *The problem of testing whether a given CNF formula φ has an empowering implicate is in NP.*

Proof. It follows from Theorem 5.3.2 that φ has an empowering implicate if and only if it has an empowering and 1-provable implicate. Thus the certificate for φ having an empowering implicate is a clause C which is both empowering and 1-provable. These properties can be checked in polynomial time using unit propagation. It follows that the problem is in NP. □

Now we shall show that the problem is NP-hard by a reduction from 3D Matching (3DM), which is a well-known NP-complete problem [37, 27]. In 3DM we are given three pairwise disjoint sets X, Y, Z of the same size $|X| = |Y| = |Z| = q$ and a set of triples $W \subseteq X \times Y \times Z$. The question we seek to answer is whether W contains a matching of size q, i.e. whether there is a subset $M \subseteq W$ of size $|M| = q$ such that each element of X, Y, and Z is contained in exactly one triple in M (i.e. the triples in M are pairwise disjoint).

Next, we present a reduction of a 3DM problem into the problem of testing the existence of an empowering implicate. The reduction is a slight modification of the proof of coNP-hardness of recognizing whether a given CNF formula is a SLUR formula [19]. Unfortunately, the reduction from [19] cannot be used directly and we have to modify it. This is because the SLUR class coincides with the class of unit refutation complete formulas (see [29]), and the class of propagation complete formulas forms a strict subclass of unit refutation formulas as we have argued at the beginning of Section 5.2. In [19] we have associated a formula φ_W to an instance of 3DM for which it was true that it was SLUR (or unit refutation complete) if and only if W contained a perfect matching. In case W does not contain a perfect matching φ_W is not unit refutation complete and thus it is not propagation complete as well. Unfortunately, the opposite implication was not true for formula φ_W constructed in [19], i.e. if W contains a perfect matching,

then φ_W is unit refutation complete, but it still not propagation complete. Thus we have to modify the original reduction in order to get the opposite implication as well.

Definition. *With every instance X, Y, Z, W of 3DM we associate a CNF formula φ_W as follows. We assume that $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$, $Z = \{z_1, \ldots, z_q\}$, and $W = \{E_1, \ldots, E_w\}$ where $w = |W|$. We also assume that $E_j = (x_{f(j)}, y_{g(j)}, z_{h(j)})$ where f, g, and h are functions determining which elements of X, Y, and Z belong to E_j (i.e. given j with $E_j = [x_{i_1}, y_{i_2}, z_{i_3}]$, function f returns the index of the x member of triple E_j, thus $f(j) = i_1$, similarly $g(j) = i_2$, and $h(j) = i_3$).*

- *For every $i \in \{1, \ldots, q-1\}$ let us denote $A_i = (a_i \vee \overline{a_{i+1}})$ where $a_1, \ldots, a_q$ are new variables, and let $A_q = (a_q \vee a_1)$.*
- *For every $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, w\}$ let us denote $B_i^j = (\overline{b_i^1} \vee \cdots \vee \overline{b_i^{j-1}} \vee b_i^j \vee \overline{b_i^{j+1}} \vee \cdots \vee \overline{b_i^w})$, i.e. B_i^j denotes a clause on variables $b_i^1, \ldots, b_i^w$ in which every literal is negative except b_i^j.*
- *For every $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, w\}$ let us denote $C_i^j = (\overline{c_i^1} \vee \cdots \vee \overline{c_i^{j-1}} \vee c_i^j \vee \overline{c_i^{j+1}} \vee \cdots \vee \overline{c_i^w})$, i.e. C_i^j denotes a clause on variables $c_i^1, \ldots, c_i^w$ in which every literal is negative except c_i^j.*
- *Given a triple $E_j \in W$, let $D_j = (A_{f(j)} \vee B_{g(j)}^j \vee C_{h(j)}^j)$.*
- *Finally, let $\varphi_W = \bigwedge_{j=1}^w D_j$.*

Example 5.4.2. *Let $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$, $Z = \{z_1, z_2, z_3\}$, and $W = \{[x_1, y_1, z_1], [x_2, y_3, z_2], [x_3, y_2, z_3], [x_1, y_2, z_3], [x_3, y_1, z_1]\}$. Then there are two possible matchings $M_1 = \{[x_1, y_1, z_1], [x_2, y_3, z_2], [x_3, y_2, z_3]\}$ and $M_2 = \{[x_1, y_2, z_3], [x_2, y_3, z_2], [x_3$*

The formula φ_W for this 3DM instance would be

$$\begin{aligned}\varphi_W = {} & (a_1 \vee \overline{a_2} \vee b_1^1 \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee \overline{b_1^5} \vee c_1^1 \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee \overline{c_1^5}) \wedge \\ & (a_2 \vee \overline{a_3} \vee \overline{b_3^1} \vee b_3^2 \vee \overline{b_3^3} \vee \overline{b_3^4} \vee \overline{b_3^5} \vee \overline{c_2^1} \vee c_2^2 \vee \overline{c_2^3} \vee \overline{c_2^4} \vee \overline{c_2^5}) \wedge \\ & (a_3 \vee a_1 \vee \overline{b_2^1} \vee \overline{b_2^2} \vee b_2^3 \vee \overline{b_2^4} \vee \overline{b_2^5} \vee \overline{c_3^1} \vee \overline{c_3^2} \vee c_3^3 \vee \overline{c_3^4} \vee \overline{c_3^5}) \wedge \\ & (a_1 \vee \overline{a_2} \vee \overline{b_2^1} \vee \overline{b_2^2} \vee \overline{b_2^3} \vee b_2^4 \vee \overline{b_2^5} \vee \overline{c_3^1} \vee \overline{c_3^2} \vee \overline{c_3^3} \vee c_3^4 \vee \overline{c_3^5}) \wedge \\ & (a_3 \vee a_1 \vee \overline{b_1^1} \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee b_1^5 \vee \overline{c_1^1} \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee c_1^5).\end{aligned}$$

In the following, we shall show that the empowering implicates of φ_W correspond to perfect matchings of W. In particular, given φ_W in this example we shall have two possible empowering implicates corresponding to matchings M_1 and M_2 respectively.

$$\begin{aligned}H_1 = (a_1 \vee {} & b_1^1 \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee \overline{b_1^5} \vee c_1^1 \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee \overline{c_1^5} \vee \\ & \overline{b_3^1} \vee b_3^2 \vee \overline{b_3^3} \vee \overline{b_3^4} \vee \overline{b_3^5} \vee \overline{c_2^1} \vee c_2^2 \vee \overline{c_2^3} \vee \overline{c_2^4} \vee \overline{c_2^5} \vee \\ & \overline{b_2^1} \vee \overline{b_2^2} \vee b_2^3 \vee \overline{b_2^4} \vee \overline{b_2^5} \vee \overline{c_3^1} \vee \overline{c_3^2} \vee c_3^3 \vee \overline{c_3^4} \vee \overline{c_3^5})\end{aligned}$$

and

$$H_2 = (a_1 \vee \overline{b_2^1} \vee \overline{b_2^2} \vee \overline{b_2^3} \vee b_2^4 \vee \overline{b_2^5} \vee \overline{c_3^1} \vee \overline{c_3^2} \vee \overline{c_3^3} \vee c_3^4 \vee \overline{c_3^5} \vee$$
$$\overline{b_3^1} \vee b_3^2 \vee \overline{b_3^3} \vee b_3^4 \vee \overline{b_3^5} \vee \overline{c_2^1} \vee c_2^2 \vee \overline{c_2^3} \vee c_2^4 \vee \overline{c_2^5} \vee$$
$$\overline{b_1^1} \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee b_1^5 \vee \overline{c_1^1} \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee c_1^5).$$

We claim that φ_W admits an empowering implicate if and only if the input 3DM instance has a perfect matching. The proof of this claim is split into the following two lemmas.

Lemma 5.4.3. *Using the notation from Definition 5.4 let $M \subseteq \{1, \ldots, w\}$ be a perfect matching and let $G = \bigvee_{j \in M} B^j_{g(j)} \vee C^j_{h(j)} \vee a_1$ be a clause. Then G is an empowering implicate of φ_W with a_1 being the empowered literal.*

Proof. Let us renumber the triples in W so that $M = \{1, \ldots, q\}$ and that $f(i) = i$, i.e. $x_i \in E_i$, for $i = 1, \ldots, q$. Now, let us consider the following chain of resolutions:

$$G_1 = D_1, G_2 = R(G_1, D_2), \ldots, G_i = R(G_{i-1}, D_i), \ldots, G_q = R(G_{q-1}, D_q).$$

One can check that $G_q = G$ and thus G is an implicate of φ_W. All clauses in this chain are resolvable because the triples in the matching are disjoint.

It remains to prove that G is empowering. Let φ' denote the formula originating from φ_W by falsifying (substituting the value $false$ for) all literals in G except a_1. We are going to show that $\varphi' \equiv A_1 \wedge \cdots \wedge A_q$. To this end, let $D_j = (A_{f(j)} \vee B^j_{g(j)} \vee C^j_{h(j)})$ be an arbitrary clause in φ_W and let us have a look on what happens with D_j after falsifying the aformentioned literals. If $j \in M$, then we have falsified all literals in D_j except $A_{f(j)}$. If $j \notin M$, then let $j' \in M$ be an index for which $g(j') = g(j)$. Such an index exists because M as a matching covers element $y_{g(j)}$. It follows that $B^{j'}_{g(j')}$ has two conflict variables with $B^j_{g(j)}$ (note that $j \neq j'$) and thus by falsifying all literals in $B^{j'}_{g(j')}$ we necessarily satisfy the clause D_j. By these considerations only clauses $A_1, \ldots, A_q$ remain in φ'. Note that none of these clauses is missing since M is a matching. The CNF formula $\varphi' = A_1 \wedge \cdots \wedge A_q$ has no unit clauses and thus unit propagation will not derive anything from it. It follows that G is an empowering implicate. □

Example 5.4.4. *Let us consider implicate H_1 from example 5.4.2 and let us show how it can be derived by resolutions from formula φ_W. Implicate H_1 corresponds to matching $M_1 = \{[x_1, y_1, z_1], [x_2, y_3, z_2], [x_3, y_2, z_3]\}$. The clauses of φ_W corresponding to triples in M_1 are:*

$$\begin{aligned}
D_1 &= (a_1 \vee \overline{a_2} \vee b_1^1 \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee \overline{b_1^5} \vee c_1^1 \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee \overline{c_1^5}) \\
D_2 &= (a_2 \vee \overline{a_3} \vee \overline{b_3^1} \vee b_3^2 \vee \overline{b_3^3} \vee \overline{b_3^4} \vee \overline{b_3^5} \vee \overline{c_2^1} \vee c_2^2 \vee \overline{c_2^3} \vee \overline{c_2^4} \vee \overline{c_2^5}) \\
D_3 &= (a_3 \vee a_1 \vee \overline{b_2^1} \vee \overline{b_2^2} \vee b_2^3 \vee \overline{b_2^4} \vee \overline{b_2^5} \vee \overline{c_3^1} \vee \overline{c_3^2} \vee c_3^3 \vee \overline{c_3^4} \vee \overline{c_3^5}).
\end{aligned}$$

By resolving D_1 with D_2 we get clause

$$G_1 = (a_1 \vee \overline{a_3} \vee b_1^1 \vee \overline{b_1^2} \vee \overline{b_1^3} \vee \overline{b_1^4} \vee \overline{b_1^5} \vee c_1^1 \vee \overline{c_1^2} \vee \overline{c_1^3} \vee \overline{c_1^4} \vee \overline{c_1^5} \vee$$
$$\overline{b_3^1} \vee b_3^2 \vee \overline{b_3^3} \vee \overline{b_3^4} \vee \overline{b_3^5} \vee \overline{c_2^1} \vee c_2^2 \vee \overline{c_2^3} \vee \overline{c_2^4} \vee \overline{c_2^5}).$$

By further resolving G_1 with D_3 we get desired implicate H_1. Observe that G_1 was produced by non-merge resolution and thus by Lemma 5.2.3 it is not an empowering implicate. On the other hand $H_1 = R(G_1, D3)$ is produced using a merge resolution and it is hence a good candidate for an empowering implicate of φ_W. If we falsify all literals in H_1 except a_1, we get clauses $D'_1 = (a_1 \vee \overline{a_2})$, $D'_2 = (a_2 \vee \overline{a_3})$, and $D'_3 = (a_3 \vee a_1)$ from clauses D_1, D_2, and D_3 respectively. The remaining two clauses in φ_W are satisfied by this assignment. Thus H_1 is indeed an empowering implicate of φ_W.

Lemma 5.4.5. *Let $G = G' \vee u$ be a prime empowering implicate of φ_W with u being the empowered literal. Then u must be a_1 and G' determines perfect matching in the same way as G in Lemma 5.4.3.*

Proof. The proof heavily relies on Lemma 5.2.7 and Lemma 5.2.8. Let us consider a resolution derivation of $G' \vee u$. By Lemma 5.2.7 and Lemma 5.2.8 this derivation contains a tied chain in which every link literal is resolved upon. Consider how such a tied chain in φ_W may look like. The only possible tied literal is a_1, and the link literals are among $a_2, \ldots, a_q$. This is because, as we can observe, the c and b variables cannot be resolved upon (if two clauses have a conflict in a c or b variable, they are not resolvable as they have at least two conflicts). Thus a tied chain in φ_W has to look as follows: $D_{i_1}, \ldots, D_{i_q}$, where $f(i_j) = j$, i.e. D_{i_1} contains $(a_1 \vee \overline{a_2})$, D_{i_q} contains $(a_q \vee a_1)$ or vice versa if we look at the tied chain in the opposite direction. We shall assume without loss generality that the former is the case (i.e. D_{i_1} contains $(a_1 \vee \overline{a_2})$ and D_{i_q} contains $(a_q \vee a_1)$). For $1 < j < q$ clause D_{i_j} contains $(a_j \vee \overline{a_{j+1}})$. The chain has to have the length q as this is necessary to get from a_1 back to a_1 through link literals among $a_2, \ldots, a_q$. Now, if we look at how clauses $D_{i_1}, \ldots, D_{i_q}$ can be resolved upon, we get that triples $E_{i_1}, \ldots, E_{i_q}$ must be disjoint to have each consecutive pair in the sequence $D_{i_1}, \ldots, D_{i_q}$ resolvable. This is because if D_{i_j}, $D_{i_{j+1}}$ are resolvable, then they cannot have a conflict in any b or c variable, which implies that E_{i_j} and $E_{i_{j+1}}$ are disjoint. We can also observe that the b and c variables cannot be cancelled out by resolution and they are therefore all present in the resolvent. Thus, the only possibility is that G' itself contains all the B and C parts from $D_{i_1}, \ldots, D_{i_q}$ and thus G' itself determines the matching. □

Theorem 5.4.6. *The problem of testing whether a given CNF formula has an empowering implicate is an NP-complete problem.*

Proof. The NP membership is proved in Lemma 5.4.1. The NP hardness follows from the reduction defined by Definition 5.4 using Lemma 5.4.3 and Lemma 5.4.5. Note that by Lemma 5.2.2 it is enough to consider prime empowering implicates and thus the assumption that G is a prime implicate in Lemma 5.4.5 is not restrictive. □

Using the reduction from Definition 5.4, we can also show that when trying to make a CNF formula propagation complete, we can in general observe an exponential blow up of the number of clauses. In particular we can show that there is a uniform and size increasing family of CNF formulas in which an exponential number of implicates has to be added to a member of this family in order to make it propagation complete. Here by "uniform" we mean that we have a uniform

construction which constructs a formula on n variables from this family based only on a parameter n.

Theorem 5.4.7. *There is a uniform and size increasing family of CNF formulas parameterized with the number of variables n and where the number of clauses is $O(n)$, such that the number of implicates that needs to be added to a CNF formula φ on n variables from this family to make it propagation complete is exponential in n (and thus in the size of the formula, too).*

Proof. Let us consider an instance of 3DM where $|X| = |Y| = |Z| = q$ and $W = X \times Y \times Z$. We claim that at least $(q!)^2$ implicates have to be added to φ_W to make it propagation complete. First, let us observe that we can find $(q!)^2$ pairwise different perfect matchings in W. This is because each perfect matching in W can be viewed as a pair of perfect matchings in a complete bipartite graph with partities X and Y and a perfect matching in a complete bipartite graph with partities Y and Z. We have $q!$ perfect matchings in each of these bipartite graphs and thus we have $(q!)^2$ possible pairs of them. Note, on the other hand, that $w = |W| = q^3$.

Let $M = \{m_{r_1}, m_{r_2}, \ldots, m_{r_q}\}$, $M \subseteq W$ be a perfect matching. Let us denote the clause

$$H_M = a_1 \vee \bigvee_{i=1}^{q} \left(B^{r_i}_{g(r_i)} \vee C^{r_i}_{h(r_i)} \right).$$

From Lemma 5.4.3 it follows that H_M is an empowering implicate. We claim that if we add H_M to φ_W, then $H_{M'}$ remains empowering provided M' is a different matching than M. Since M and M' are different and $|M| = |M'|$, we have that $M \setminus M' \neq \emptyset$ and $M' \setminus M \neq \emptyset$. Let r_ℓ be an index of a triple such that $m_{r_\ell} \in M \setminus M'$. Thus $B^{r_\ell}_{g(r_\ell)}$ forms a subclause of H_M. Since m_{r_ℓ} does not belong to M', we have that $g(r_\ell)$ is covered by a different triple r'_ℓ in M'. Thus $B^{r'_\ell}_{g(r'_\ell)}$, where $g(r'_\ell) = g(r_\ell)$, forms a subclause of $H_{M'}$. If we falsify all literals in $H_{M'}$ except a_1, we get that H_M is satisfied, because $B^{r_\ell}_{g(r_\ell)}$ and $B^{r'_\ell}_{g(r'_\ell)}$ have a conflict variable. Hence H_M plays no role in unit propagation used to possibly derive a_1. Whether a_1 can or cannot be derived by unit resolution from $\varphi_W \wedge H_M$ after falsifying the literals in $H_{M'}$ except a_1 is thus equivalent to whether a_1 can be derived by unit resolution from φ_W after falsifying the literals in $H_{M'}$ except a_1. Note that the above observation can be generalized to the case when we add more than one of these matching clauses to φ_W and thus it is necessary to add all the clauses corresponding to perfect matchings to φ_W to make it propagation complete.

It follows that the number of implicates needed to be added to φ_W to make it propagation complete is at least $(q!)^2$ which is exponential in the size of formula φ_W consisting of q^3 clauses build on $q + qw + qw = \Theta(q^4)$ variables. Family of the formulas defined in this, with $n = \Theta(q^4)$, satisfies the proposition of the theorem. □

As we have already discussed in Section 5.1, the result contained in Theorem 5.4.7 is in tight connection to the results in [11] where the authors show in Corollary 4 that there is no polynomial sized CNF decomposition of any AllDifferent domain consistency propagator. As we have mentioned in Section 5.1, the superpolynomial bound shown in [10] is based on a superpolynomial lower

bound of Razborov [46] on the size of a monotone circuit computing whether there is a perfect matching in a given graph. Quite interestingly, to the best of our knowledge, there is no stronger lower bound on the size of monotone circuit computing whether a bipartite graph contains perfect matching than the super-polynomial shown in [46]. It is thus an interesting question whether a stronger lower bound could not be shown using the connection between domain consistency propagators and monotone circuits shown in [11]. Note that in the proof of Theorem 5.4.7 we could use a bipartite matching instead of 3DM for constructing the family of CNF formulas with required properties (we used 3DM mainly to take advantage of Lemma 5.4.3). Thus, it might be possible to construct a domain consistency propagator for the AllDifferent constraint which contains such a modified family of CNF formulas. On the other hand, it would still be only one example of a domain consistency propagator which is not enough to argue about any domain consistency propagator for the AllDifferent constraint.

Chapter 6

Conclusion

Almost every area of theoretical computer science touches in one way or the other the main topic of this dissertation – Boolean functions and especially their CNF representations. We have looked at three areas of this broad topic.

First we have looked at Boolean minimization and its connection to satisfiability in certain classes of CNFs. In particular, we have looked at how the complexity of minimization changes when the satisfiability problem becomes solvable in polynomial time – that is when it drops down one level in the polynomial hierarchy. Most of the facts presented in this chapter are not surprising or new, however, we found it interesting that there are classes that have a gap of two levels in the polynomial hierarchy between minimization and satisfiability, and tried to provide a overview of common classes of CNF from this perspective.

We have shown that when a given class satisfies certain conditions, the minimization problem is guaranteed to drop down at least one level and in some cases even two levels. We have provided examples of classes for both of these possibilities. However, if the class does not satisfy the conditions (i.e. the class is not tractable), its minimization can stay as hard as in the general case.

An example of such a class, matched CNFs, were the topic of Chapter 4. Both minimization problems for this class of CNFs are Σ_2^p complete. The original proofs for these statements were modifications of proofs of Σ_2^p completeness of Boolean minimization for the general case. We provided here a simpler proof for clause minimization.

The main results of this chapter states that for any matched CNF there exists an equivalent prime and irredundant CNF that is also matched, and that every clause minimum CNF of a matched formula is matched. While the second claim implies the first, the proof of the second one requires the first one.

Chapter 5 looks at CNFs from a different perspective. Instead of minimization, i.e. CNF compression, we looked at CNF compilation, a special type of knowledge compilation, which is in some sense a reverse process to compression, as it makes the input CNF longer not shorter. The compilation process rests in making a CNF propagation complete, that is it tries to expand the given CNF by adding empowering implicates in such a way, that queries against this CNF can be processed fast using only unit propagation. We looked at the complexity issues that are connected to propagation completeness.

We have shown that the recognition of an empowering implicate is coNP complete where the hard part is showing that the given clause is indeed an implicate.

However, if there is an empowering implicate for a given CNF, there must be always at least one that is 1-provable. The main result of this chapter is that testing the existence of an empowering implicate for a given CNF is NP complete and therefore the recognition of propagation complete CNFs is coNP complete. Furthermore, we have shown that there are CNFs which cannot be made propagation complete without adding exponentially many implicates.

Bibliography

[1] Ron Aharoni and Nathan Linial. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *Journal of Combinatorial Theory, Series A*, 43(2):196 – 204, 1986.

[2] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121 – 123, 1979.

[3] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 114–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] G. Ausiello, A. D'Atri, and D. Sacca. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, 15(2):418–431, May 1986.

[5] Martin Babka, Tomáš Balyo, Ondřej Čepek, Štefan Gurský, Petr Kučera, and Václav Vlček. Complexity issues related to propagation completeness. *Artificial Intelligence*, 203(0):19 – 34, 2013.

[6] Fahiem Bacchus. GAC via unit propagation. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, Berlin, Heidelberg, 2007.

[7] Paul Beanie, Henry Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, pages 1194–1201, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[8] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

[9] Christian Bessière, Emmanuel Hebrard, and Toby Walsh. Local consistencies in sat. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 299–314. Springer Berlin Heidelberg, 2004.

[10] Christian Bessiere, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decompositions of all different, global cardinality and

related constraints. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 419–424, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[11] Christian Bessiere, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 412–418, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[12] Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, 1998.

[13] Lucas Bordeaux and Joao Marques-Silva. Knowledge compilation with empowerment. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 612–624. Springer-Verlag, Berlin, Heidelberg, 2012.

[14] Endre Boros, Ondřej Čepek, Alexander Kogan, and Petr Kučera. A subclass of horn cnfs optimally compressible in polynomial time. *Annals of Mathematics and Artificial Intelligence*, 57:249–291, 2009. 10.1007/s10472-010-9197-7.

[15] Endre Boros, Ondřej Čepek, Alexander Kogan, and Petr Kučera. Exclusive and essential sets of implicates of boolean functions. *Discrete Applied Mathematics*, 158(2):81 – 96, 2010.

[16] Endre Boros, Ondřej Čepek, and Petr Kučera. A decomposition method for CNF minimality proofs. *Theoretical Computer Science*, 2013. Available online 23 September 2013.

[17] Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter Stuckey, and Toby Walsh. Encodings of the sequence constraint. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, pages 210–224, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] J. Buresh-Oppenheim and T. Pitassi. The complexity of resolution refinements. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, pages 138–147, 2003.

[19] Ondřej Čepek, Petr Kučera, and Václav Vlček. Properties of SLUR formulae. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 177–189. Springer-Verlag, Berlin, Heidelberg, 2012.

[20] Ondřej Čepek and Štefan Gurský. On the gap between the complexity of sat and minimization for certain classes of boolean formulas. In *Proceedings of International Symposium on AI and Mathematics (ISAIM)*, 2014.

[21] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[22] Y. Crama and P.L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 2011.

[23] Alvaro del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In J. Doyle, E. Sandewall, and P. Torassi, editors, *KR'94, Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 551–561. Morgan Kaufmann, 1994.

[24] W.F. Dowling and J.H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267 – 284, 1984.

[25] Kave Eshghi. A tractable class of abduction problems. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1*, IJCAI'93, pages 3–8, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[26] John Franco and Allen Van Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Appl. Math.*, 125(2-3):177–214, 2003.

[27] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[28] Štefan Gurský. Časová složitost minimalizace booleovských funkcí. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2010.

[29] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. *CoRR*, abs/1204.6529, 2012.

[30] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(0):297 – 308, 1985.

[31] Philip Hall. On Representatives of Subsets. *Journal of The London Mathematical Society-second Series*, s1-10:26–30, 1935.

[32] P.L. Hammer and A. Kogan. Horn functions and their dnfs. *Information Processing Letters*, 44:23 – 29, 1992.

[33] P.L. Hammer and A. Kogan. Optimal compression of propositional horn knowledge bases: Complexity and approximation. *Artificial Intelligence*, 64:131 – 145, 1993.

[34] P.L. Hammer and A. Kogan. Quasi-acyclic propositional horn knowledge bases: Optimal compression. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):751 – 762, 1995.

[35] Jinbo Huang. Universal booleanization of constraint models. In Peter J. Stuckey, editor, *Principles and Practice of Constraint Programming*, volume 5202 of *Lecture Notes in Computer Science*, pages 144–158. Springer-Verlag, Berlin, Heidelberg, 2008.

[36] A. Itai and J.A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4:105 – 117, 1987.

[37] 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.

[38] Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107(1–3):99 – 137, 2000.

[39] Harry R. Lewis. Renaming a set of clauses as a horn set. *J. ACM*, 25(1):134–135, January 1978.

[40] D. Maier. Minimal covers in the relational database model. *Journal of the ACM*, 27:664 – 674, 1980.

[41] M. Minoux. Ltur: A simplified linear time unit resolution algorithm for horn formulae and computer implementation. *Information Processing Letters*, 29:1 – 12, 1988.

[42] Knot Pipatsrisawat and Adnan Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1481–1484. AAAI Press, 2008.

[43] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, February 2011.

[44] Claude-Guy Quimper and Toby Walsh. Decompositions of grammar constraints. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1567–1570. AAAI Press, 2008.

[45] Willard V. Quine. A way to simplify truth functions. *The American Mathematical Monthly*, 62(9):627–631, 1955.

[46] A. A. Razborov. Lower bounds on monotone complexity of some boolean functions. *Doklady Akademii Nauk SSSR*, 281(4):789–801, 1985.

[47] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[48] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[49] Christian Schulte and PeterJ. Stuckey. Speeding up constraint propagation. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 619–633. Springer Berlin Heidelberg, 2004.

[50] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984.

[51] Christopher M. Umans. *Approximability and completeness in the polynomial hierarchy*. PhD thesis, University of California, Berkeley, 2000.

[52] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, January 1987.

[53] Alasdair Urquhart. The depth of resolution proofs. *Stud. Log.*, 99(1-3):349–364, October 2011.

[54] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, ICCAD '01, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press.