Hierarchies for efficient clausal entailment checking: With applications to satisfiability and knowledge compilation.

Gwynne, Matthew

Hierarchies for efficient clausal entailment checking: with applications to satisfiability and knowledge compilation

Matthew Gwynne

Submitted to Swansea University in
fulfilment of the requirements for the Degree of
Doctor of Philosophy.

Swansea University
Prifysgol Abertawe

Swansea University

July 2013

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)
Date .....13/02/2014.....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed . ..... (candidate)
Date .....13/02/2014.....

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)
Date .....13/02/2014.....

Abstract

Given the NP-completeness of the satisfiability problem, the search for classes of CNF clause-set with poly-time satisfiability checking is of particular interest and applicability (e.g., Horn, 2-$\mathcal{CLS}$, $\mathcal{SLUR}$). In areas such as knowledge compilation (KC) and constraint translation (to CNF), even stronger properties such as clausal entailment (i.e., a CNF F logically implies a clause C) need to be efficiently decidable. This thesis introduces several new hierarchies of clause-sets with efficient clausal-entailment checking, which extend and subsume existing hierarchies.

The main object of study is the $\mathcal{UC}_k$ hierarchy, generalising the $\mathcal{UC}$ class of *unit-refutation complete clause-sets*, introduced in [51] as a target class for knowledge compilation. Via the theory of "hardness" of clause-sets as developed in [104, 110, 2] the class $\mathcal{UC}$ is naturally generalised to $\mathcal{UC}_k$, containing those clause-sets which are "unit-refutation complete of level k", which is the same as having hardness at most k. Relating $\mathcal{UC}_k$ to poly-time satisfiability the class $\mathcal{SLUR}$ (*Single Lookahead Unit Resolution*) is considered, which was introduced in [141] as an umbrella class for efficient (poly-time) SAT solving. $\mathcal{SLUR}$ generalises to $\mathcal{SLUR}_k$ in the same way as $\mathcal{UC}$ and one of the core insights of this thesis is that $\mathcal{SLUR}_k = \mathcal{UC}_k$ holds for all k (including $\mathcal{SLUR} = \mathcal{UC}$!). One can thus exploit both SAT and KC streams of intuitions and methods for investigations of these hierarchies. As applications it is shown that membership decision for fixed levels of these hierarchies is coNP-complete and that $\mathcal{SLUR}_k$ ($= \mathcal{UC}_k$) strongly subsumes hierarchies from [36, 10], as well as other hierarchies for efficient SAT solving.

A natural question is whether each level allows more succinct representations. Another core result of this thesis is an answer in the positive sense. Without new variables, each level of these hierarchies offer exponentially more succinct representations. This means that there are boolean functions with only exponential-size equivalent clause-sets at level k, but with poly-size equivalent clause-sets at level $k+1$. This shows that $\mathcal{UC}_k$ forms a knowledge compilation hierarchy inbetween the CNF and PI classes from [46], with query efficiency similar to the PI class (i.e., sets of prime implicates), allowing one to trade query time for succinctness.

In the outlook, representations allowing new variables are considered. It is envisioned, due to the strong connections between the "hardness" notion from [104, 110, 2] and resolution complexity, that the hierarchies defined might act as a SAT-orientated alternatives to constraint-based notions of "good" SAT representation such as "maintaining arc-consistency via unit-clause-propagation". In this direction it is shown that several common CNF representations already fit into the $\mathcal{UC}_k$ scheme, and experiments are provided demonstrating that modern DPLL and CDCL solvers *are able* to solve CNF clause-sets at higher levels of $\mathcal{UC}_k$ and $\mathcal{WC}_k$, while solvers can perform poorly when using much larger representations in $\mathcal{UC}_1$. This provides motivation for considering higher levels of $\mathcal{UC}_k$ for representations in satisfiability solving, and adds an additional dimension to the question of "good" SAT representations.

Acknowledgements

Firstly I'd like to thank my examiners, Arnold Beckmann and Ian Gent. Their comments and questions helped shape this thesis into a better and more well presented whole and for that I am very grateful.

My supervisor, Oliver Kullmann, needs special mention for his tireless effort in guiding me not just academically in my work but also in terms of being a better, more productive person. I think it will still be many years to come before I fully appreciate all of the lessons I have learnt, and am still learning, from working with Oliver. I can not thank him enough.

I am eternally indebted to the generousity and encouragement of Swansea University Computer Science Department. They have both supported me financially throughout my time there and offered so many opportunities to excel. I am particularly grateful to members of the Theory group, including, in no particular order, Ulrich, Monika, John, Arnold, Jens, Markus, Anton and so many others, who all helped to make me feel welcome and enlighten me with a constant supply of new ideas.

In the same way, I must mention the companionship of those with whom I shared the same journey. Without friends like Phillip James, Emma Thom, Fredrik Forsberg, Liam O'Reilly, Jennifer Pearson, Tom Owen and many more, I think my sanity would have slowly ebbed away. My thanks also extend to Stephen Richards, who stoically and patiently suffered living with me throughout my PhD, with barely a word of complaint, no matter how much deserved.

My longer suffering family deserve so many thanks. I cherish your love and support throughout my life and appreciate how much such things matter in shaping one's path in life.

Lastly, but most importantly, I am grateful for the tireless, absolute and loving support of my partner Emma. Thank you for sticking with me throughout the entire journey and for making everything worth it.

Contents

Chapter 1

Introduction

The (boolean) satisfiability problem (SAT) has been a much studied research area, especially in the last 20 years, seeing applications in verification (see [102]) and bounded model checking (see [22]), scheduling and planning (see [137]), combinatorial mathematics (see [164]), cryptanalysis (see [122, 41, 11, 150]), and also further afield in areas such as bioinformatics (see [72]); for an overview see [23]. A large part of the success and popularity of SAT and modern SAT solvers over other methods comes from the simplicity and rich underlying structure of Conjunctive Normal Form (CNF) clause-sets (see Chapter 2), used as the standardised representation format. Given the NP-completeness of the SAT problem (as proven by Cook in [38]), an important theme is the search for relevant classes $\mathcal{C}$ of clause-sets F for which one can (at least) decide satisfiability in *polynomial time* (that is, deciding whether F logically implies the empty clause); see Section 1.19 in [63] for some basic information.

However, what if one wants a stronger property than mere poly-time satisfiability checking? What if one wants to be able to check satisfiability of F under any partial instantiation? Then the (known and fixed) satisfiability of F does not necessarily help. Such properties are important in *knowledge compilation*, where one compiles (offline) a knowledge base into some representation (e.g., a CNF clause-set) and then wants that to be able to perform many (online) queries efficiently (e.g., checking satisfiability under a partial assignment). For this task one requires that clausal entailment queries (deciding whether F logically implies some given clause, rather than just the empty clause) can be decided in polynomial time; see [46] for an overview.

Such properties are also important when one is interested in translating constraints or boolean functions to some CNF F which will form only *part* of a larger SAT problem. Most SAT solvers search for a satisfying assignment by building up partial assignments, piece by piece. Search times can be exponentially reduced if the solver can (efficiently) detect the unsatisfiability of F under this *partial* assignment, and so halt search "in the wrong direction" as soon as possible. This is the underlying idea behind "good" representations in satisfiability by "maintaining arc-consistency" via unit-clause propagation, i.e., detect assignments which must be made (to satisfy the CNF) as early as possible, using linear time unit-clause propagation.

This thesis brings together these two areas to form hierarchies, based on hardness notions from [104, 110], for *SAT knowledge compilation*: compiling all of the "knowledge" of a constraint or boolean function, (i.e., whether it is satisfiable under each partial assignment), into the CNF representation, such that partial assignments made by modern SAT solvers (the "queries") are efficient. Three hierarchies are constructed: $\mathcal{UC}_k$, based on the $\mathcal{UC}$ class introduced by Del Val in [51] for knowledge compilation and strongly connected to tree-resolution space complexity; $\mathcal{PC}_k$, based on the $\mathcal{PC}$ class introduced in [26] and intimately connected with well-known concept of

"maintaining arc-consistency via unit-clause-propagation" (see Section 2.8 of Chapter 2); and the $\mathcal{WC}_k$ hierarchy, extending $\mathcal{WC} = \mathcal{UC}$ but now using (improved) *asymmetric* based notions of *full*-resolution width complexity from [104, 110]. Each fixed level of each of these hierarchies offer poly-time clausal entailment (as well as other queries) for knowledge compilation, while subsuming existing classes, and in Chapter 5 it is shown that each level allows for exponentially more compression than the level below (i.e., the entire hierarchies themselves are useful for knowledge compilation). Furthermore, in Chapter 7 it is shown SAT solvers are able to utilise the strong connections of these hierarchies to resolution to quickly find short proofs for small compressed clause-sets in $\mathcal{UC}_k$ for low enough k, while demonstrating poor performance on "easier" but much larger instances in $\mathcal{UC}_1$. Together, the results from Chapter 5 and Chapter 7 demonstrate both that these hierarchies as a whole are useful, not just $\mathcal{UC}_1$, and also that finding "good" SAT representations is a multi-dimensional problem, with at least "hardness" and size dimensions to consider.

1.1 Clausal entailment in knowledge compilation and SAT

As an illustrative example, consider the boolean function f given by the truth table in Figure 1.1. While in reality this function has been chosen for its ability to concisely convey the key concepts of this thesis, for the sake of argument, consider two scenarios and interpretations of f:

1. **Scenario 1**: f encodes the following product configuration problem (see [146] for work on using $\mathcal{UC}$ for product configuration problems):
   (a) there is a product (e.g., a computer, car etc) which can be configured with 4 different parts $i_1, \ldots, i_4$;
   (b) the valid configurations for p are given by those sets $S \subseteq \{i_1, \ldots, i_4\}$ where $f(\varphi_S) = 1$ and the partial assignment $\varphi_S : \{v_j : i_j \in S\} \times \{0,1\}$ is defined by
   $$\varphi_S(v_j) := \begin{cases} 1 & i_j \in S \\ 0 & \text{otherwise.} \end{cases}$$
2. **Scenario 2**: f encodes scheduling constraints at time t in the following scheduling problem:
   (a) there are 4 employees $e_1, \ldots, e_4$ working at a factory;
   (b) a valid scheduling of employees to time slot t must obey the constraints in f, i.e., the valid sets of employees $S \subseteq \{e_1, \ldots, e_4\}$ which can be scheduled together at time t is given by those sets for which $f(\varphi_S) = 1$ and the partial assignment $\varphi_S : \{v_i : e_i \in S\} \times \{0,1\}$ is defined by
   $$\varphi_S(v_i) := \begin{cases} 1 & e_i \in S \\ 0 & \text{otherwise.} \end{cases}$$
   In this case, f is just *one* constraint in the larger scheduling problem. Each time slot $t' \in T$ (for a set of times T) will have its own scheduling constraints for $e_1, \ldots, e_4$. In general these constraints might even overlap (e.g., employees might not want to work back to back). So overall there is a boolean function $\hat{f} := f \wedge f'$ where f' encodes together all scheduling constraints at times other than t.

v_1	v_2	v_3	v_4	$f(v_1, v_2, v_3, v_4)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

v_1	v_2	v_3	v_4	$f(v_1, v_2, v_3, v_4)$
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Figure 1.1: Truth for boolean function $f : \{0,1\}^{\{v_1,v_2,v_3,v_4\}} \to \{0,1\}$.

In the above scenarios we might want to consider two different tasks:

1. In Scenario 1, users will attempt to build different configurations of product p and the system must be able to quickly return whether these configurations are valid. That is, we want to represent f in such a way as to be able to *efficiently answer queries* regarding its satisfiability under certain assignments.

2. In Scenario 2, the larger scheduling problem $\hat{f}$ is given only implicitly as the conjunction of the smaller scheduling problems for each timeslot, and so the task here is find representations for f (and for f') such that it is efficient to decide whether there exists a valid scheduling of the employees across timeslots (i.e., *deciding satisfiability* for $\hat{f}$).

To answer these two tasks we will examine the following 3 Conjunctive Normal Form (CNF) *clause-set* representations of f where for each CNF the order and operations have been extracted away (recall that boolean conjunction ($\wedge$) and disjunction ($\vee$) are both associative and commutative).

$$\boldsymbol{F_0} := \{ \underbrace{\{v_1, \overline{v_3}, \overline{v_4}\}}_{C_1}, \underbrace{\{v_2, v_3, \overline{v_4}\}}_{C_2}, \underbrace{\{v_2, \overline{v_3}, v_4\}}_{C_3}, \underbrace{\{\overline{v_2}, v_3, v_4\}}_{C_4}, \underbrace{\{v_1, v_3, v_4\}}_{C_5}, \underbrace{\{v_1, v_2\}}_{C_6} \}.$$

$$\boldsymbol{F_1} := \{ \underbrace{\{v_1, \overline{v_3}, \overline{v_4}\}}_{C_1}, \underbrace{\{v_2, v_3, \overline{v_4}\}}_{C_2}, \underbrace{\{v_2, \overline{v_3}, v_4\}}_{C_3}, \underbrace{\{\overline{v_2}, v_3, v_4\}}_{C_4}, \underbrace{\{v_1, v_2\}}_{C_6} \}.$$

$$\boldsymbol{F_2} := \{ \underbrace{\{v_1, \overline{v_3}, \overline{v_4}\}}_{C_1}, \underbrace{\{v_2, v_3, \overline{v_4}\}}_{C_2}, \underbrace{\{v_2, \overline{v_3}, v_4\}}_{C_3}, \underbrace{\{\overline{v_2}, v_3, v_4\}}_{C_4}, \underbrace{\{v_1, v_3, v_4\}}_{C_5} \}.$$

For each of the above *clause-sets* the CNF interpretation is applied, i.e.,

$$f = \bigwedge_{C \in F_0} \bigvee_{x \in C} x = \bigwedge_{C \in F_1} \bigvee_{x \in C} x = \bigwedge_{C \in F_2} \bigvee_{x \in C} x.$$

So, for example, interpretting F_0 as a CNF, we have

$$f = (v_1 \vee \neg v_3 \vee \neg v_4) \wedge (v_2 \vee v_3 \vee v_4) \wedge (v_2 \vee \neg v_3 \vee v_4) \wedge (\neg v_2 \vee v_3 \vee v_4) \wedge (v_1 \vee v_3 \vee v_4) \wedge (v_1 \vee v_2)$$

Task 1: F_0, F_1 and F_2 are expected to be able to "efficiently" answer queries in the form of assignments to the variables to check the consistency of potential product configurations. To illustrate the different possibilities and an initial insight into the $\mathcal{UC}_k$ hierarchy (introduced in Chapter 2), consider these representations of F under the following queries.

1. Consider F_0 and the partial assignment $\varphi_{C_6} := \langle v_1 \to 0, v_2 \to 0 \rangle$ (i.e., v_1 is set to 0 and v_2 is set to 0). By applying standard simplifications (i.e., removal of satisfied clauses and falsified literals) on the CNF we see immediately that $\varphi_{C_6} * f$ (i.e., f after applying φ_{C_6}) is unsatisfiable, i.e., any product configuration not containing either i_1 or i_2 is invalid:

$$\begin{aligned} \varphi_{C_6} * F_0 &= \{ \{v_1, \overline{v_3}, \overline{v_4}\}, \{v_2, v_3, \overline{v_4}\}, \{v_2, \overline{v_3}, v_4\}, \{\overline{v_2}, v_3, v_4\}, \{v_1, v_3, v_4\}, \{v_1, v_2\} \} \\ &= \{ \{\overline{v_3}, \overline{v_4}\}, \{v_3, \overline{v_4}\}, \{\overline{v_3}, v_4\}, \{v_3, v_4\}, \bot \}. \end{aligned}$$

That $\langle v_1 \to 0, v_2 \to 0 \rangle * f$ is unsatisfiable follows via the observation that the empty clause (i.e., the empty disjunction) is derived.

2. Consider F_1 and the query $\varphi_{C_5} := \langle v_1 \to 0, v_3 \to 0, v_4 \to 0 \rangle$. Now when applying the same simplifications as for the above case, we do not immediately derive the empty clause and so do not "immediately" detect that $\varphi_{C_5} * f = 0$:

$$\begin{aligned} \varphi_{C_5} * F_1 &= \{ \{v_1, \overline{v_3}, \overline{v_4}\}, \{v_2, v_3, \overline{v_4}\}, \{v_2, \overline{v_3}, v_4\}, \{\overline{v_2}, v_3, v_4\}, \{v_1, v_2\} \} \\ &= \{ \{\overline{v_3}, \overline{v_4}\}, \{\overline{v_2}\}, \{v_2\} \}. \end{aligned}$$

However, observe that further setting v_2 to 0 would yield the empty clause (due to $\{v_2\}$) - therefore if one were to satisfy $\varphi_{C_5} * F_1$ then such a satisfying assignment must set v_2 to 1. That is, we must satisfy the unit-clause $\{v_2\}$. Propagating $\langle v_2 \to 1 \rangle$ then yields the empty clause due to the other unit-clause $\{\overline{v_2}\}$. In this way, this simple form of lookahead ("unit-clause propagation" - see Section 2.4 of Chapter 2) determines the unsatisfiability of $\varphi_{C_5} * F_1$ and so the incompatibility of the corresponding product configuration.

3. Finally, consider F_2 and the query $\varphi_{C_6} := \langle v_1 \to 0, v_2 \to 0 \rangle$. This time, applying the same simplifications above yields a CNF clause-set without any unit-clauses and so unit-clause propagation does nothing:

$$\begin{aligned} \varphi_{C_6} * F_0 &= \{ \{v_1, \overline{v_3}, \overline{v_4}\}, \{v_2, v_3, \overline{v_4}\}, \{v_2, \overline{v_3}, v_4\}, \{\overline{v_2}, v_3, v_4\}, \{v_1, v_3, v_4\} \} \\ &= \{ \{\overline{v_3}, \overline{v_4}\}, \{v_3, \overline{v_4}\}, \{\overline{v_3}, v_4\}, \{v_3, v_4\} \}. \end{aligned}$$

Instead, we can apply the same reasoning but one level higher. That is, checking whether a literal is forced by setting it and then checking via unit-clause propagation whether that assignment results in f being unsatisfiable (this is *failed literal elimination*).

More precisely, in this case we have $\langle v_3 \to 0 \rangle * (\varphi_{C_6} * F_0) = \{\{\overline{v_4}\}, \{v_4\}\}$ which then via unit-clause-propagation is obviously unsatisfiable. Therefore, all satisfying assignments to $(\varphi_{C_6} * F_0)$ must set v_3 to 1, however $\langle v_3 \to 1 \rangle * (\varphi_{C_6} * F_0) = \{\{\overline{v_4}\}, \{v_4\}\}$ which again by unit-clause propagation is unsatisfiable. Therefore $(\varphi_{C_6} * F_0)$ is unsatisfiable.

These three representations offer a trade-off between the size of the representation (note F_0 is larger than F_1 and F_2) and the level of lookahead necessary to decide *clausal entailment* (i.e., whether some assignment falsifies f). The above examples are small but in general the trade-offs can be exponential as will be shown in Chapter 5.

Task 2 In this case, we assume a fixed CNF clause-set representation F' of f' and vary the representation of f over F_0, F_1 and F_2. This yields three CNF clause-set representations $\hat{F}_0 := F_0 \cup F'$, $\hat{F}_1 := F_1 \cup F'$ and $\hat{F}_2 := F_2 \cup F'$ of the overall scheduling problem $\hat{f}$. The task is to now

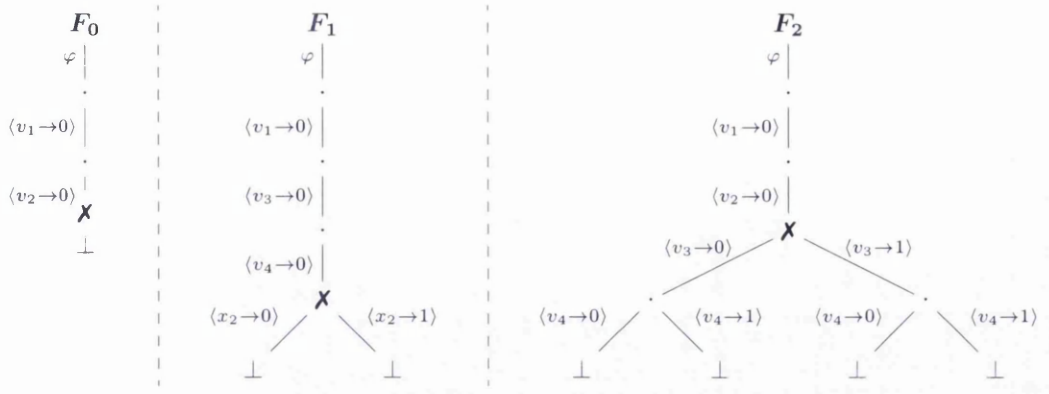


Figure 1.2: (Partial) backtracking trees for F_0', F_1' and F_2' respectively.

determine the satisfiability of $\hat{f}$ (i.e., the satisfiability of $\hat{F}_0$, $\hat{F}_1$ or equivalently $\hat{F}_2$) and hence the existence of a valid schedule in the underlying scheduling problem.

The most fundamental technique in modern SAT solving, underlying both modern DPLL[1)] (see [88]) and CDCL[2)] (see [121]) methods is backtracking (albeit augmented with clause-learning and non-chronological backjumping in CDCL). That is, essentially the solver attempts to find a satisfying assignment for the input CNF by building up a satisfying assignment piece-by-piece, applying simple reductions as the solver proceeds, and backtracking and trying a different assignment when a conflict (i.e., the empty clause) is detected. By considering the (partial) backtracking trees given in Figure 1.2 one can see that in each case, depending on the choices the SAT solver makes, the sub-problem on f can become unsatisfiable (i.e., the current partial assignment falsifies F_0 resp. F_1 resp. F_2) but the solver does not detect this until much later (i.e., the solver continues to backtrack until it detects the empty clause). In particular, we see that

1. When F_0 is made unsatisfiable under the partial assignment $\langle v_1 \to 0, v_2 \to 0 \rangle$, this is immediately detected by the SAT solver because the clause $\{v_1, v_2\}$ yields the empty clause under this assignment; no further backtracking is necessary.

2. When F_1 is made unsatisfiable under the partial assignment $\langle v_1 \to 0, v_3 \to 0, v_4 \to 0 \rangle$ then backtracking must still lookahead one level to determine this (essentially mimicking unit-clause propagation).

3. When F_2 is made unsatisfiable under the partial assignment $\langle v_1 \to 0, v_2 \to 0 \rangle$ then the solver must now backtrack even more, looking ahead two levels rather than one.

Again as in Task 1, we see a trade-off, F_0 is larger but when an inconsistent assignment is made, this is immediately obvious and no further backtracking is necessary. In general, a solver may (unnecessarily) need to search an exponential size backtracking tree to determine unsatisfiability, and so the search for representations where backtracking trees are short, or at least short backtracking trees exist are guaranteed to exist, are very useful in reducing solver time.

The aim of this thesis is to develop natural and elegant CNF "target-classes" to represent boolean functions such that the above queries and unsatisfiability detection are efficient. In

[1)] The Davis-Putnam-Logemann-Loveland (DPLL) algorithm applies backtracking, along with *unit-clause elimination* (see Definition 2.4.4) and *pure literal elimination* (see Definition 2.4.3) applied at every node in the tree.

[2)] The Conflict-Driven Clause Learning (CDCL) algorithm applies backtracking with *clause learning* and *non-chronological backjumping* (see [121]).

both cases, this essentially boils down to requiring that the representation allows efficient *clausal entailment* checking, i.e., checking unsatisfiability is efficient under any partial assignment. The meaning of the naturality and elegance conditions requirement is not just that the classes defined are not "ad-hoc" but also that the classes defined connect well with existing concepts and classes (e.g., resolution and existing poly-time SAT classes - see Chapter 4).

1.2 Classes and hierarchies for SAT knowledge compilation

The concept of "SAT knowledge compilation" comes about by bringing together two previously unconnected streams of research from SAT and knowledge compilation:

SLUR The SLUR algorithm is an incomplete linear-time SAT-decision algorithm, based on look-ahead via unit-clause propagation.

UC The class UC of unit-refutation complete clause-sets enables clausal-entailment decision in linear time via unit-clause propagation.

That both streams are based on unit-clause propagation, which is also the basic tool for efficient SAT solving, is considered here as an essential feature. It means that actually we have some form of "SAT knowledge compilation", where the "knowledge" is compiled in such a way that a SAT solver can "understand" it!

1.2.1 The quest for SLUR hierarchies

In the year 1995 in [141] the SLUR algorithm was introduced, a simple incomplete non-deterministic SAT-decision algorithm, which always succeeded on various classes with polynomial-time SAT decision where previously only rather complicated algorithms were known. The computation is divided into two phases for input-clause-set F: First we check via unit-clause propagation (UCP) for unsatisfiability. If this check fails, then we assume F is satisfiable, and guess a satisfying assignment, using UCP-look-ahead for the guessed assignments to avoid obviously false assignments. The class $\mathcal{SLUR}$ contains those F where this algorithm always succeeds (i.e., determines unsatisfiability in the first phase, or always finds a satisfying assignment in the second phase).

The natural question arises, whether $\mathcal{SLUR}$ can be turned into a hierarchy, covering in the limit all clause-sets. A generalisation of SLUR has been considered in [64] under the name "ISLUR" (improved SLUR), allowing a polynomial number $p(\ell(F))$ of backtracks (for a fixed polynomial p, in the input-size $\ell(F)$), in the unsatisfiability as well as in the satisfiability phase of the SLUR algorithm, before giving up. It is mentioned that ISLUR gives up on every large enough "sparse" clause-set (which are "typical" as random k-CNF clause-sets), when no variable occurs "too often". This was considered to be "disappointing" — but from the point of this thesis the value of the class $\mathcal{SLUR}$ lies not in being a "big" class of clause-sets with polynomial-time SAT solving, but in establishing a basic target class for representations of boolean functions with very strong properties via clause-set. For all fixed k there exists a polynomial p such that the k-th level of the hierarchy, $\mathcal{SLUR}_k$ (introduced in Chapter 4), is contained in the class ISLUR (those clause-sets where the ISLUR algorithm never gives up). So all levels are negligible when considering the above sparse clause-sets, but as we show in Chapter 5, nevertheless this hierarchy is proper regarding good representations of boolean functions, and the parameter k is meaningful and robust (not just a numerical parameter like the polynomial p).

In [36, 10] it was finally proved that membership decision of $\mathcal{SLUR}$ is coNP-complete, and three hierarchies, $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ and CANON(k) were presented. It still seemed that none

of these hierarchies is the final answer, though they all introduce a certain natural intuition. This thesis now presents what seems the natural "limit hierarchy", called here $\mathcal{SLUR}_k$, which unifies the two basic intuitions embodied in $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ on the one hand and CANON(k) on the other hand.

In order to do so a precise analysis of the $\mathcal{SLUR}$-class is needed. A SLUR transition relation $F \xrightarrow{\text{SLUR}} F'$ between clause-sets F, F' is introduced, which makes precise one non-deterministic step of the SLUR-algorithm. This transition from F to F' happens when assigning a (single) literal in such a way that UCP does not create the empty clause. The core idea of the classes $\mathcal{SLUR}(k)$ and $\mathcal{SLUR}*(k)$ is to strengthen the transition relation by requesting that not just one literal is choosable, but actually k literals can be chosen, while the difference between them is that $\mathcal{SLUR}*(k)$ performs UCP inbetween the choices, while the weaker class $\mathcal{SLUR}(k)$ does not.

Before the solution can be described, the $\mathcal{SLUR}_k$-hierarchy, the second source of "SAT knowledge compilation" approach must be discussed, the class $\mathcal{UC}$ of "unit-refutation complete clause-sets", which is related to the stream embodied by CANON(k).

1.2.2 Unit-refutation completeness and "hardness"

In the year 1994 in [51] the class $\mathcal{UC}$ was introduced, containing clause-sets F such that clausal entailment, that is, whether $F \models C$ holds (clause C follows logically from F, i.e., C is an implicate of F), can be decided by unit-clause propagation. The motivation was knowledge compilation, that is, to have a more succinct alternative to the use of the set of all prime implicates (i.e., all minimal CNF clauses entailed by F; see Section 2.6) of a given clause-set F_0 (clausal database), for which one seeks an equivalent F such that clausal entailment can be decided quickly.

A second development is important here, namely the development of the notion of "hardness" in [104, 110, 2]. The first source [104] from 1999 introduced the notion of hardness as a measure $\mathrm{hd}_0 : \mathcal{CLS} \to \mathbb{N}_0$, assigning natural numbers to clause-sets in the following way (using $\mathcal{SAT} \subset \mathcal{CLS}$ for the satisfiable clause-sets, and $\mathcal{USAT} := \mathcal{CLS} \setminus \mathcal{SAT}$):

- $\mathrm{hd}_0(F) := 0$ for the simplest clause-sets $F \in \mathcal{CLS}$ regarding SAT decision, containing the empty clause (i.e., $\bot \in F$) or being empty (i.e., $F = \top$).[3)]
- $\mathrm{hd}_0(F) \le k$ for $k \ge 1$ iff there is a literal x such that for $F' := \langle x \to 0\rangle * F$ (setting x to 0) we have $\mathrm{hd}_0(F') \le k-1$ and either $F' \in \mathcal{USAT}$ and $\mathrm{hd}_0(\langle x \to 1\rangle * F) \le k$, or $F' \in \mathcal{SAT}$. The precise value of $\mathrm{hd}(F)$ is then the minimum k such that $\mathrm{hd}(F) \le k$.

In fact, for unsatisfiable clause-sets F, $\mathrm{hd}_0(F)$ is the minimum "Horton-Strahler" number of any tree-resolution refutation of F and [104] showed that from $\mathrm{hd}_0(F)$ we get upper and lower bounds on the tree-resolution complexity of F (see Section 3.3 of Chapter 3). hd_0 can be computed in time $O(\ell(F) \cdot n(F)^{2k-2})$ (essentially via breath-first search - see [104]).

The second source [110] from 2004 generalised this approach to constraint satisfaction problems (and beyond). The third source [2] from 2008 considered $\mathrm{hd}_0(F)$ on unsatisfiable clause-sets $F \in \mathcal{USAT}$, relating it to backdoors, cycle-cutsets and treewidth, and performing an experimental study on random instances. Also in [2] we find a different extension of $\mathrm{hd}_0 : \mathcal{USAT} \to \mathbb{N}_0$ to a measure $\mathrm{hd} : \mathcal{CLS} \to \mathbb{N}_0$, using for satisfiable instances $F \in \mathcal{SAT}$ the maximisation over all unsatisfiable sub-instances obtained by applying partial assignments. This hardness notion is

[3)] Actually a two-dimensional family $\mathrm{hd}_{\mathcal{U},\mathcal{S}}$ of such measures was introduced, based on oracles $\mathcal{U} \subseteq \mathcal{USAT}$, $\mathcal{S} \subseteq \mathcal{SAT}$ for deciding unsatisfiability resp. satisfiability, and setting $\mathrm{hd}_{\mathcal{U},\mathcal{S}}(F) := 0$ for $F \in \mathcal{U} \cup \mathcal{S}$. In this thesis we consider only the simplest base case $\mathrm{hd}_0 = \mathrm{hd}_{\mathcal{U}_0,\mathcal{S}_0}$, where $\mathcal{U}_0 := \{F \in \mathcal{CLS} : \bot \in F\}$ and $\mathcal{S} := \{\top\}$. Oracle $\mathcal{S}$ does not play a role in the setting of this thesis, which is fully unsatisfiability-based. See Subsection 4.3.2 for more information on these hierarchies, and see Section 6.1 for discussions on relativised hardness.

harder to measure: as is shown in Theorem 4.4.5 of this thesis, determining whether $\mathrm{hd}(F) \leq k$ holds for a fixed $k \geq 1$ is coNP-complete, while $\mathrm{hd}_0(F) \leq k$ can be decided in polynomial time (for fixed k). Nevertheless it is the central measure for this thesis, and is considered as measuring "representation hardness", while hd_0 measures "solver hardness".[4)]

As will be shown in Theorem 4.2.2, $\mathrm{hd}(F) \leq k$ is equivalent to the property of F, that all implicates of F (i.e., all clauses C with $F \models C$) can be derived by k-times nested input resolution from F, a generalisation of input resolution as introduced and studied in [104, 110].[5)] So we obtain that $\mathcal{UC}$ is precisely the class of clause-sets F with $\mathrm{hd}(F) \leq 1$! It is then natural to define the hierarchy $\mathcal{UC}_k$ via the property $\mathrm{hd}(F) \leq k$. The hierarchy CANON(k) is based on resolution trees of height at most k, which is a special case of k-times nested input resolution, and so we have CANON(k) $\subset \mathcal{UC}_k$ (see Theorem 4.4.6).

1.2.3 Bringing SLUR and UC together

In order to get back to SLUR, we need to emphasise the two-sided nature of the hardness measure, as developed in [104, 110]. In Subsection 1.2.2 the *proof-theoretic side* of it was discussed. The *algorithmic side* is given by the reductions $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ (introduced in [104]), which perform certain forced assignments:

1. r_1 is unit-clause propagation (UCP), assigning $x \to 1$ for unit-clauses $\{x\}$ until all are eliminated.

2. r_2 is (complete) failed-literal elimination, assigning, while possible, $x \to 1$ for literals x such that the assignment $x \to 0$ yields a contradiction via r_1; see Section 5.2.1 in [88] for the usage of failed literals in SAT solvers (so-called "look-ahead solvers"), and see Section 7.2.2 in [114] for the general explanation of r_2 being the "look-ahead version" of r_1.

3. In general r_{k+1} is the "look-ahead version" of r_k, assigning, while possible, $x \to 1$ for literals x such that the assignment $x \to 0$ yields a contradiction via r_k.

For unsatisfiable F the hardness $\mathrm{hd}(F)$ is equal to the minimal k such that $\mathrm{r}_k(F)$ detects unsatisfiability of F, i.e., $\mathrm{r}_k(F) = \{\bot\}$. This yields the basic observation $\mathcal{UC} \subseteq \mathcal{SLUR}$ — and actually we have $\mathcal{UC} = \mathcal{SLUR}$!

So by replacing the use of r_1 in the SLUR algorithm by r_k (using the analysis in Chapter 3 via the transition relation) we obtain a natural hierarchy $\mathcal{SLUR}_k$, which includes the previous SLUR-hierarchies $\mathcal{SLUR}(k)$ and $\mathcal{SLUR}*(k)$, as well as various other classes and hierarchies for poly-time SAT, and where we have $\mathcal{SLUR}_k = \mathcal{UC}_k$. This equality of these two hierarchies, and the dual perspective offered (algorithmic and proof theoretic), is the argument that the "limit hierarchy" for SLUR has been found.

1.2.4 PC and arc-consistency

A similar class to $\mathcal{UC}$, but coming initially from a SAT perspective, is the class $\mathcal{PC}$, introduced in [26], of propagation complete clause-sets. Whereas $\mathcal{UC}$ treats r_1 as a consistency-checker, $\mathcal{PC}$ uses r_1 to detect all forced assignments under any partial assignment. As $\mathcal{UC}$ is generalised to $\mathcal{UC}_k$, $\mathcal{PC}$ is now generalised to the hierarchy $\mathcal{PC}_k$ in Chapter 4. $\mathcal{PC}_k$ then provides intermediate

[4)] $\mathrm{hd}(F)$ actually captures tree-like resolution (in a sense). In Section 3.6.2 a width-based measure of hardness is discussed, which captures dag-like resolution. This thesis considers the tree-hardness as the natural starting point.

[5)] Equivalently, as shown in [104, 110], one can say that all implicates C have a tree-resolution proof using space at most $k + 1$.

layers to the $\mathcal{UC}_k$ hierarchy (i.e., we have $\mathcal{PC}_0 \subset \mathcal{UC}_0 \subset \mathcal{PC}_1 \subset \mathcal{UC}_1 \subset \mathcal{PC}_2 \ldots$) and relates more directly to the notion of a clause-set "maintaining arc-consistency via unit-clause propagation" (see Section 2.8 in Chapter 2).

That "arc-consistency" is "maintained" via unit-clause propagation means that for all (partial) assignments to the variables of the constraint, if there is a forced assignment (i.e., some variable which must be set to a particular value to avoid inconsistency), then unit-clause propagation (UCP) is sufficient to find and set this assignment. Maintaining arc-consistency and propagation-completeness may at a glance seem the same concept (both ask for forced literals to be propagated by r_1). However there is an essential difference. When translating a constraint into SAT, typically one does not just use the variables of the constraint, but one adds auxiliary variables to allow for a compact representation. Now when one speaks of maintaining arc-consistency, one only cares about assignments to the *constraint variables*. However propagation-completeness deals only with the representation clause-set, thus can not know about the distinction between original and auxiliary variables, and thus it is a property on the (partial) assignments over *all* variables! A SAT representation, which maintains arc-consistency via UCP, will in general not be propagation-complete, due to assignments over both constraint *and* new variables yielding a forced assignment or even an inconsistency which UCP doesn't detect; see Example 6.2.5 and Lemma 6.2.6. In contrast to this, for the basic concepts of "good" representations investigated in this thesis, considering *all* variables is a fundamental feature.

1.2.5 WC_k: a hierarchy based on full resolution

As already mentioned in Section 1.2.2 (to be further elaborated in Chapter 3), there are strong proof theoretic connections for $\mathcal{UC}_k$ to *tree*-resolution. In [95] the argument is made that tree-resolution complexity can not provide a good measure of hardness of instances for SAT solving, citing the ability of modern Conflict Driven Clause Learning (CDCL) solvers to simulate exponentially more powerful full-resolution (see [4, 129, 130] for evidence that CDCL solvers can simulate full resolution). However, the aim of $\mathcal{UC}_k$ is not to *measure* hardness, but to offer a *target class* for SAT translation. In this respect tree-resolution complexity measures are ideal, because they provide the strongest translations, and upper-bound measures for full resolution.

$\mathcal{UC}_k$ being based on the hd_0, the hardness measure from [104], which has strong connections to tree-resolution, means that if an unsatisfiable (sub-)clause-set has a high tree-resolution complexity (see Definition 2.5.3 in Chapter 2) then it has a high hardness, even if there exists a short resolution proof for it. For tighter target classes in the case of full resolution, the notion of width-hardness as introduced in [77, 78] is also considered, based on the width-based hierarchies of unsatisfiable clause-sets in [104, 110]. That is, a clause-set is in $\mathcal{WC}_k$, the hierarchy of clause-sets of width-hardness k, iff under any partial assignment resulting in an unsatisfiable clause-set there is a "k-resolution" refutation as introduced in [99]. Here, unlike the typical notion of width, resolutions where only *one* parent clause needs to have length at most k are allowed, thus properly generalising unit-resolution (one could speak of "asymmetric width" here, compared to the standard "symmetric width"). This allows the simulation of nested input resolution, and thus we have $\mathcal{UC}_k \subseteq \mathcal{WC}_k$ for all k, whereas otherwise in the standard (symmetrical) sense even Horn clause-sets require unbounded width (recall that $\mathcal{HO} \subset \mathcal{UC}_1$). While $\mathcal{UC}_k$ offers the stronger guarantee that short tree-resolution proofs exist, $\mathcal{WC}_k$ relaxes this condition to allow clause-sets which might have only short *full*-resolution proofs. In this way, one can prove lowerbounds for width-hardness and upper-bounds for (tree-)hardness and hence prove the precise width- and tree- hardness for certain classes of clause-sets (see e.g., Section 1.3 and Chapter 5).

1.3 Strict hierarchies of representations

Considering these hierarchies as target classes for SAT translations and knowledge compilation, we know at least that for every $k \in \mathbb{N}_0$ and every boolean function f there is some CNF representation $F \in \mathcal{UC}_0$ (the prime implicate representation of F). A natural question to ask is whether going from e.g., $\mathcal{UC}_k$ to $\mathcal{UC}_{k+1}$ means that certain boolean functions can be represented in *polynomial size* which only had *exponential-size* representations before. That is, in terms of representation, does each level offer more, or does the hierarchy collapse to $\mathcal{UC}_1$?

In [51] (Example 2) a separation was already shown between $\mathcal{UC}_0$ (clause-sets containing all of their prime implicates) and $\mathcal{UC}_1 = \mathcal{UC}$, and the question was raised of the worst-case growth when compiling from an arbitrary CNF clause-set F to some equivalent $F' \in \mathcal{UC}$. This question was partly answered in [17] (although the connection was not made), where examples are provided of poly-size clause-sets with only super-polynomial size representations in $\mathcal{UC}$, even when allowing new variables (see Subsections 6.1 and 8.2 for more on the connection between [17] and $\mathcal{UC}_k$). This shows a super-polynomial lower-bound on the worst-case growth, but no method or new (larger) target-class for knowledge-compilation. Theorem 5.3.13 in Chapter 5 now answers the question of worst-case growth from [51] in full generality with the hierarchy $\mathcal{UC}_k$. Each level of $\mathcal{UC}_k$ is exponentially more expressive than the previous (i.e., with possible *exponential* blow-up when compiling from some $F \in \mathcal{UC}_{k+1}$ to equivalent $F' \in \mathcal{UC}_k$), and so each level offers its own new, *larger* class for knowledge compilation, at the expense of increased query time (now $O(\ell(F) \cdot n(F)^{2k-2})$ for $\mathcal{UC}_k$ compared to $O(\ell(F))$ for $\mathcal{UC}$). This separation, between $\mathcal{UC}_{k+1}$ and $\mathcal{UC}_k$ for arbitrary k, is more involved than the simple separation in [51], due to the parameterised use of more advanced polynomial-time methods than r_1 (UCP). Especially the separation between $\mathcal{UC}_0$ and $\mathcal{UC}_1$ is rather simple, since $\mathcal{UC}_0$ does not allow any form of compression (i.e., simply checking for the empty clause propagates nothing and assigns to no variables).

1.3.1 Separating the $\mathcal{UC}_k$ and $\mathcal{WC}_k$ hierarchies

A sequence $(f_h)_{h\in\mathbb{N}}$ of boolean functions, which separates $\mathcal{UC}_{k+1}$ from $\mathcal{UC}_k$ w.r.t. clause-sets equivalent to f_h in $\mathcal{UC}_{k+1}$ resp. $\mathcal{UC}_k$, should have the following properties:

1. **A large number of prime implicates**: the number of prime implicates for f_h should at least grow super-polynomially in h, since otherwise already the set of prime implicates is a small clause-set in $\mathcal{UC}_0$ (see Definition 3.4.1) equivalent to f_h.
2. **Easily characterised prime implicates**: the prime implicates of f_h should be easily characterised, since otherwise we can not understand how clause-sets equivalent to f_h look like.
3. **Poly-size representations**: there must exist short clause-sets in $\mathcal{UC}_{k+1}$ equivalent to f_h for all $h \in \mathbb{N}$.

[148] introduced a special type of boolean functions, called there Non-repeating Unate Decision trees (NUD), by adding new variables to each clause of clause-sets in $\mathcal{SMU}_{\delta=1}$, which is the class of unsatisfiable hitting clause-sets of deficiency $\delta = 1$ (see Section 5.2.1 of Chapter 5). These boolean functions have a large number of prime implicates (the maximum regarding the original number of clauses), and thus are natural to consider as candidates to separate the levels of $\mathcal{UC}_k$. Theorem 5.1.18 in Section 5.1 shows that it is actually the underlying $\mathcal{SMU}_{\delta=1}$ clause-sets that contribute the structure. The clause-sets in $\mathcal{SMU}_{\delta=1}$ are exactly those with the maximum number of "minimal premise sets", and then "doping" elements of $\mathcal{SMU}_{\delta=1}$ yields clause-sets

with the maximal number of prime implicates. Theorem 5.3.13 then uses the tree structure of $\mathcal{SMU}_{\delta=1}$ to prove lower bounds on the size of equivalent representations of doped $\mathcal{SMU}_{\delta=1}$ clause-sets in $\mathcal{UC}_k$.

To be able to prove properties about all equivalent representations of some CNF clause-set F, we must be able to understand its combinatorial structure in relation to the set of all its prime implicates. The notion of minimal unsatisfiability (MU) and minimally unsatisfiable subsets (MUS) is important in understanding the combinatorics of unsatisfiable clause-sets (see [[100, 120]). To understand the structure of satisfiable clause-sets and their associated boolean functions, Section 5.1 of Chapter 5 now considers the concept of "minimal premise sets" (MPS) introduced in [116]. The notion of MPS generalises that of MUS by considering clause-sets F which are minimal w.r.t implying *any* clause C rather than just those implying $\bot$ (the empty clause). And accordingly one can consider the minimal-premise subsets (MPSS) of a clause-set F.

Every prime implicate C of a clause-set F has an associated MPSS (just consider the minimal sub-clause-set of F that implies C), but not every MPSS of F yields a prime implicate of F (e.g., consider the MPSS $\{C\}$ for some non-prime clause $C \in F$). By "doping" the clause-set, i.e., adding a new unique variable to every clause, every clause in an MPSS F' makes a unique contribution to its derived clause C. This results in a new clause-set $\mathrm{D}(F)$ which has an exact correspondence between its minimal premise sets (which are (essentially) also those of F) and its prime implicates. In this way, by considering clause-sets F with a very structured set of minimal premise subsets, we can derive clause-sets $\mathrm{D}(F)$ with very structured set of prime implicates.

Section 5.3 of Chapter 5 introduces the basic method (see Theorem 5.3.4) for lower bounding the size of equivalent clause-sets of a given hardness, via the transversal number of "trigger hypergraphs". Using this lower bound method, Theorem 5.3.12 shows a lower bound on the matching number of the trigger hypergraph of doped "extremal" $\mathcal{SMU}_{\delta=1}$-clause-sets. From this follows immediately Theorem 5.3.13, that for every $k \in \mathbb{N}_0$ there are polysize clause-sets in $\mathcal{UC}_{k+1}$, where every equivalent clause-set in $\mathcal{WC}_k$ is of exponential size. Thus the $\mathcal{UC}_k$ as well as the $\mathcal{WC}_k$ hierarchy is strict regarding equivalence of polysize clause-sets.

1.3.2 Strict hierarchies for knowledge compilation

Looking at $\mathcal{UC}_k = \mathcal{SLUR}_k$ again from the $\mathcal{UC}$ perspective, i.e., for knowledge compilation, the question is where does it fit with respect to existing knowledge compilation classes? [34] gives an overview of the CNF-based target languages (prime implicates, $\mathcal{UC}$, 2-$\mathcal{CLS}$, Horn clause-sets). [59] consider disjunctions of simple CNF classes. [46] provides an overview of target compilation languages based on "nested" (graph-based) classes, namely variants of NNF, DNNF and BDDs. In all cases query complexity and succinctness is investigated. This thesis focuses on CNF representations, with the hope in the outlook towards good representations for current resolution-based SAT solvers. All of the CNF classes studied in [34, 46, 59] are included at the first three levels of the hierarchy $\mathcal{UC}_k$, namely, sets of prime implicates in $\mathcal{UC}_0$, (renamable) Horn clause-sets in $\mathcal{UC}_1 = \mathcal{UC}$, and 2-$\mathcal{CLS}$ in $\mathcal{UC}_2$.

We will see in Theorem 4.7.1 in Chapter 4 that $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ have the same poly-time queries as the PI knowledge compilation class (i.e., the class of clause-sets which are sets of prime implicates; see Section 2.6 of Chapter 2). This is shown in comparison to other knowledge compilation classes in Figure 1.4 (with full discussion in Chapter 4). Along with separation result in Theorem 5.3.13, this means that $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ form intermediate layers between the PI and CNF knowledge compilation classes, offering increasing more succinct representations further up the hierarchy, at the cost of polynomially more query effort (illustrated in Figure 1.5). Furthermore, every fixed level of each hierarchy is a complete class with respect to representation

Notation	Query
CO	poly-time consistency (satisfiability) checking
VA	poly-time validity (tautology) checking
CE	poly-time clausal entailment checking
IM	poly-time implicant check
EQ	poly-time equivalence checking
SE	poly-time semantic entailment
CT	poly-time model counting
ME	poly-time model enumeration

Figure 1.3: Notations for knowledge compilation queries. Full details are available in Theorem 4.7.1 in Chapter 4 and in [46].

of boolean functions (as are all in Figure 1.5), unlike classes such as 2-$\mathcal{CLS}$ (CNF clause-sets with clauses of size at most two) or $\mathcal{HO}$ (Horn clause-sets) and other hierarchies for polynomial time satisfiability like $\mathcal{RHO}_k$ (generalised renamable Horn clause-sets; see Section 2.7.2.1); each of which is included at some fixed level of $\mathcal{UC}_k \subseteq \mathcal{WC}_k$.

An important point here is that, although the separation between levels of $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ is only w.r.t to equivalent representations (i.e., without new variables), this perfectly fits into the knowledge compilation framework. Allowing the addition of new variables, i.e., moving to classes such as $\exists\mathcal{UC}$ from [25], means that queries such as SE (semantic entailment, i.e., checking whether one CNF clause-set $F \in \exists\mathcal{UC}$ entails another $F' \in \exists\mathcal{UC}$ on free variables) are no longer poly-time decidable because unlike for CNF clause-sets (without new variables) one can no longer enumerate sets of falsifying assignments on the *free* (original) variables by enumerating clauses of the clause-set (i.e., for a CNF clause-set F we have $F \models F'$ iff $\forall C \in F' : F \models C$ which is not the case for existentially quantified CNFs). In particular, it is shown in [25] that $\exists\mathcal{UC}$, as well as other query classes built by taking the closure of $\mathcal{UC}$ under disjunction, does not allow poly-time VA, IM, EQ or SE queries (as shown in Figure 1.4) while $\mathcal{UC}$, and now more generally $\mathcal{UC}_k$, does.

1.4 "Good" SAT representations

So we have seen that the $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ hierarchies are promising target classes for knowledge compilation (KC) and polynomial time satisfiability, offering intermediate layers between the CNF and PI classes in KC. Looking back at the $\mathcal{SLUR}_k$ perspective and connections to resolution (and hence to modern SAT solvers), in the outlook these classes should also provide good target classes for SAT translation, i.e., the translation to CNFs to be solved by modern SAT solvers. In general, for translations to SAT a typical path is

$$\textbf{Problem} \rightarrow \textbf{Constraints} \rightarrow \underbrace{\textbf{Boolean functions} \rightarrow \textbf{SAT}}_{\textbf{focus of this thesis}}.$$

The $\mathcal{PC}_k, \mathcal{UC}_k$ and $\mathcal{WC}_k$ classes now offer "good" target classes for representing boolean functions (i.e., after a constraint is *encoded* as a boolean function) in the sense that they guarantee the existence of short tree- resp. full-resolution proofs. In [4, 129, 130] we see evidence that modern CDCL SAT solvers are able to find resolution proofs with low complexity in expected polynomial time. Furthermore, in Chapter 7, we see experimental evidence that when the level of $\mathcal{UC}_k$ is low enough modern state-of-the-art SAT solvers are able to solve these problems relatively quickly.

L	CO	VA	CE	IM	EQ	SE	CT	ME
NNF	o	o	o	o	o	o	o	o
DNNF	✔	o	✔	o	o	o	o	✔
d-NNF	o	o	o	o	o	o	o	o
s-NNF	o	o	o	o	o	o	o	o
f-NNF	o	o	o	o	o	o	o	o
d-DNNF	✔	✔	✔	✔	?	o	✔	✔
sd-DNNF	✔	✔	✔	✔	?	o	✔	✔
BDD	o	o	o	o	o	o	o	o
FBDD	✔	✔	✔	✔	?	o	✔	✔
OBDD	✔	✔	✔	✔	✔	o	✔	✔
$\text{OBDD}_{\leq}$	✔	✔	✔	✔	✔	✔	✔	✔
DNF	✔	o	✔	o	o	o	o	✔
CNF	o	✔	o	✔	o	o	o	o
PI	✔	✔	✔	✔	✔	✔	o	✔
IP	✔	✔	✔	✔	✔	✔	o	✔
MODS	✔	✔	✔	✔	✔	✔	✔	✔
$\mathcal{UC}_k$	✔	✔	✔	✔	✔	✔	o	✔
$\mathcal{PC}_k$	✔	✔	✔	✔	✔	✔	o	✔
$\mathcal{WC}_k$	✔	✔	✔	✔	✔	✔	o	✔
$\exists\mathcal{UC}$	✔	o	✔	o	o	o	o	✔
$\mathcal{UC}[\vee, \exists]$	✔	o	✔	o	o	o	o	✔

Figure 1.4: Subsets of the NNF language and their corresponding polytime queries. ✔means "query possible in polytime", o means "not possible in polytime (in general) unless P = NP" and ? that there is no known result either way. Results and definitions for $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ are from Theorem 4.7.1 and Chapter 4 resp., results and definitions for $\exists\mathcal{UC}$ and $\mathcal{UC}[\vee, \exists]$ are from [25] and all other results and definitions are from [46]. See Figure 1.3 for query descriptions.

[An NNF-language represents a boolean function as a rooted DAG (directed acyclic graph) with $\wedge$ and $\vee$ at the nodes and 1, 0, literal X or literal $\neg X$ at the leaves. Subsets of NNF add constraints on the DAG such as decomposability (D), determinism (d-) and smoothness (s-) - see [46] for definitions.]

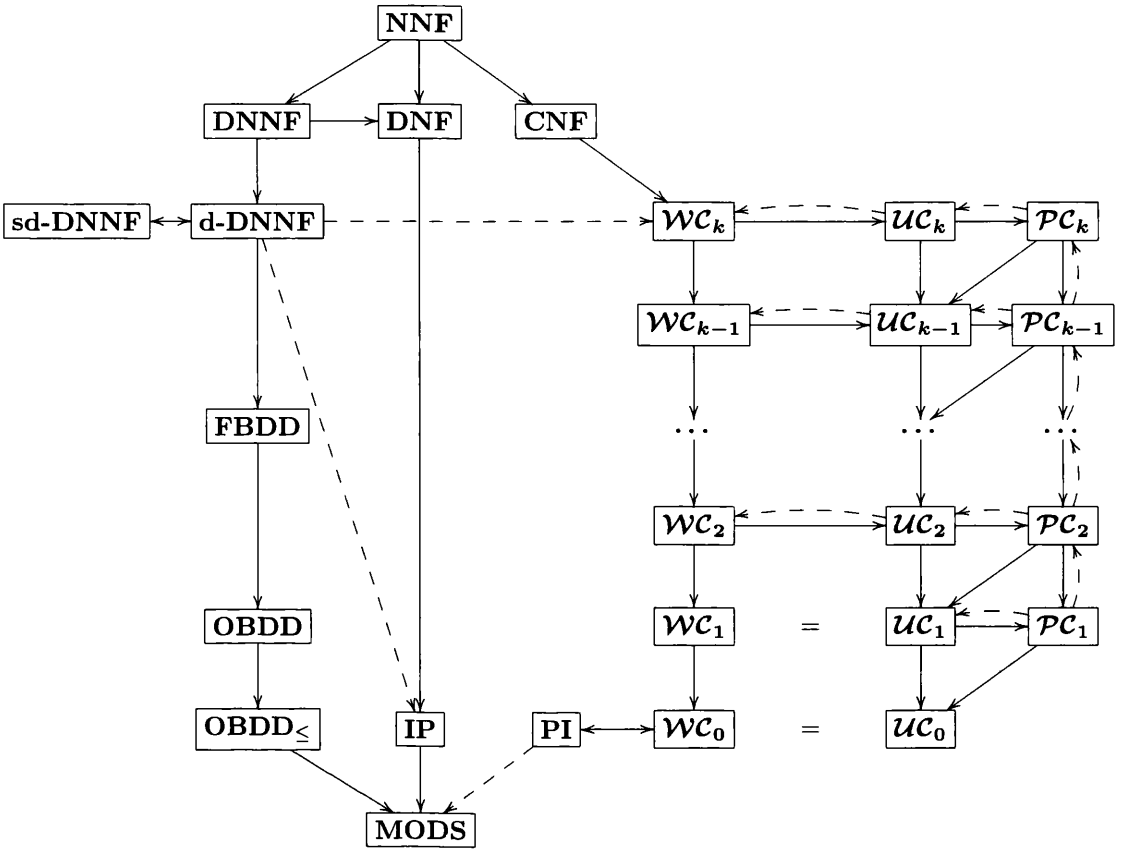


Figure 1.5: Illustration of the succinctness relations between knowledge compilation classes from [46] and those introduced in this thesis. If there is a solid directed path (using only $\rightarrow$) from class L to class L' then every boolean function with poly-size representations in L' has a poly-size representation in L (i.e., L is at least as succinct as L'). If there is no solid directed path from L to L' but there is a directed path (using some $\dashrightarrow$) then it is unknown whether L is at least as succinct as language L'. No directed path from L to L' indicates that there are boolean functions for which L has a succinct (poly-size) representation but L' does not. Results relating to $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ are from Theorem 5.3.13; all other results are from [46]; some of which rely on the non-collapse of the polynomial hierarchy. More details are given in Section 4.7 of Chapter 4.

Furthermore, a more direct relation between SAT solving and $\mathcal{UC}_k$ is possible. The class $\mathcal{UC}_k$ uses generalised UCP, namely the reduction r_k. Especially r_2, which is (complete) failed-literal elimination, is used in look-ahead SAT solvers (see [88] for an overview) such as `OKsolver` ([108]), `march` ([89, 87]) and `Satz` ([119]). Also conflict-driven solvers such as `CryptoMiniSat` ([149]) and `PicoSAT` ([18, 20]) integrate r_2 during search, and solvers such as `Lingeling` ([20, 21]) use r_2 as a preprocessing technique. Furthermore, in general r_k is used, in even stronger versions, in the Stålmarck-solver (see [153, 84, 143], and see Section 3.5 of [104] for a discussion of the connections to r_k), and via breadth-first "branch/merge" rules in `HeerHugo` (see [73]).

1.4.1 New variables: the relative vs absolute condition

Certain results in this thesis apply only to representing boolean functions *without new variables*. On the one hand, fixing the boolean function has certain advantages. We saw for knowledge compilation that considering only equivalent representations had the upside that certain queries (i.e., VA, EQ, and SE) were poly-time decidable, while not being polytime decidable once existential quantification was introduced. When finding "good" SAT representations, avoiding the use of new variables allows the enumeration and optimisation of representations within a target-class, i.e., the space of all representations is finite and the optimum representation can be searched for (e.g., see [74]).

On the other hand, when translating constraints to satisfiability, new variables can play a pivotal role, both in terms of succinctness (for example, the XOR clauses in Section 6.4 of Chapter 6 have no short representation without new variables at all) and in terms of power, e.g., many representations which "maintain arc-consistency via UCP" do this via the introduction of new variables with certain semantic meaning (see e.g., cardinality constraint translations in [7]). When using $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ as target-classes for SAT translation an important question is not just *whether* one uses new variables, but what being in $\mathcal{UC}_k$ etc means in the presence of new variables.

The most prevalent notion of "good" representation in satisfiability (of some higher level constraint) is one that "maintains arc-consistency via unit-clause propagation". This is a *relative* notion that requires that for any partial assignment (instantiation) of the constraint variables (applied to the representation) unit-clause propagation on the CNF representation sets variables which are "forced" at the constraint level (see Section 2.8 for an explanation). As with the $\exists\mathcal{UC}$ knowledge compilation class, there is a distinction between the bound new variables and the free original variables – the properties regarding propagation only apply to partial assignments over the original variables. Translating a constraint "fully" as a clause-set into one of the $\mathcal{UC}_k, \mathcal{PC}_k$, or $\mathcal{WC}_k$ hierarchies is instead an *absolute* condition – the clause-set knows nothing about the constraint and *all* partial assignments must be considered – a modern SAT solver can branch on any variable (including new).

In fact, in [94] it is shown that conflict-driven solvers with branching restricted to input variables have only superpolynomial run-time on EPHP'_n, an Extended Resolution extension to the well-known pigeon-hole formulas, while unrestricted branching determines unsatisfiability quickly (see Section 8.2 for more on this). Also experimentally it is demonstrated in [96] that input-restricted branching can have a detrimental effect on solver times and proof sizes for modern CDCL solvers. This adds motivation to the fundamental choice of considering *all* variables (rather than just input variables), when deciding what properties we want for SAT translations. This is called here the "absolute (representation) condition", taking also the auxiliary variables into account, while the "relative condition" only considers the original variables. Besides avoiding the creation of hard unsatisfiable sub-problems, the absolute condition also enables one to study the target classes, like $\mathcal{PC}$, on their own, without relation to what is represented, i.e., $\mathcal{UC}_k$ is *just* a class of clause-sets.

1.4.2 Tools for good representations

Chapter 6 concludes the theoretical investigations by considering translations based on the Tseitin translation in Chapter 5, and show that interesting classes of boolean function can be polynomially translated to $\mathcal{UC}$ under the absolute condition using new variables. First the notion of "representation" in general is discussed in Subsection 6.1, with special emphasise on the "relative" versus the "absolute condition".

The Tseitin translation for DNFs is called here the "canonical translation", and is investigated in Subsection 6.2. In particular, in Lemma 6.2.7 it is shown that every orthogonal (or "disjoint", or "hitting") DNF is translated to $\mathcal{UC}$, while Lemma 6.3.3 shows that actually every DNF is translated to $\mathcal{UC}$, when using the "reduced" canonical translation, which uses only the necessary part of the equivalences constitutive for the Tseitin translation. Applied to the examples from Chapter 5 yielding the separation of $\mathcal{UC}_{k+1}$ from $\mathcal{WC}_k$ (Theorem 5.3.13, regarding polysize representations without new variables), we obtain a representation in $\mathcal{UC}$ in Theorem 6.2.9 (for the canonical translation), demonstrating the power of using new variables.

It has been noted in the literature at several places (see [131, 93, 55]) that one might use only one of the two directions of the equivalences in the Tseitin-translation. Regarding the canonical translation we have the full translation (Definition 6.2.1) versus the reduced translation (Definition 6.3.1). The full translation yields $\mathcal{UC}$ for special inputs (Lemma 6.2.7), and has relative hardness 1 for general DNF (Lemma 6.2.4), however (absolute) hardness for arbitrary DNF-inputs can be arbitrarily high as shown in Lemma 6.2.6. On the other hand, the reduced translation yields always $\mathcal{UC}$ (Lemma 6.3.3). So we have the following explanations why using either both directions or only one direction in the Tseitin translation, in the context of translating DNFs, can perform better than the other form:

- When using both directions (i.e., the canonical translation), then splitting on the auxiliary variables is powerful, which is an advantage over using only one direction (i.e., the reduced canonical translation), where setting an auxiliary variable to false says nothing.

- On the other hand, the canonical translation, when applied to non-hitting DNFs, can create hard unsatisfiable sub-problems (via partial assignments), which can not happen for the reduced translation.

It seems very interesting to us to turn these arguments into theorems (for concrete examples), and also to experimentally evaluate them. In this way we hope that in the future more precise directions can be given when to use which form of the Tseitin translation.

1.4.3 Experimental results

In Chapter 7 attention is turned to experimentation using the class of boolean functions f used for the lower bound in Chapter 5, comparing short representations in $\mathcal{UC}_k$ (for $k \in \{2, \ldots, 5\}$) to the $\mathcal{UC}_1$ translations introduced in Chapter 6. as a constraint in a general SAT problem. We complement these three constraint-representations in a fixed way to obtain an unsatisfiable clause-set. The experiments show that for state-of-the-art SAT solvers (both DPLL and CDCL) the optimal (smaller) $\mathcal{UC}_k$ representations perform much better in terms of running time. This yields some evidence to the claim that equivalent representations in $\mathcal{UC}_k$ even for higher k (in the experiments here $k \leq 5$ is considered) might outperform representations obtained by introducing new variables, due to using (possibly) far fewer variables and clauses.

1.5 Summary of results

The fundamental (new) definitions and concepts introduced in this thesis, as well as basic associated lemmas, are:

1. Definition 3.4.1 defines the notion of hardness upon which the $\mathcal{UC}_k$ hierarchy is built, generalising the hardness for unsatisfiable clause-sets from [104, 110]. This concept was first mentioned as one of several possibilities for hardness measures in [2]; it is now a central concept.

2. Definition 4.2.1 introduces the $\mathcal{UC}_k$ hierarchy, the emphasis of this thesis. The fundamental change of perspective is that rather than *measuring* hardness as in [104, 110, 2], the $\mathcal{UC}_k$ hierarchy now acts as a target-class for translation and good representations.

3. Definition 3.5.1 and Definition 4.6.1 introduce propagation hardness and the $\mathcal{PC}_k$ hierarchy based on the $\mathcal{PC}$ class from [26]. The $\mathcal{PC}_k$ hierarchy has strictly stronger properties than $\mathcal{UC}_k$ (see Section 3.5 of Chapter 3), however $\mathcal{UC}_k$ is conceptually simpler with (more) direct connections to tree-resolution complexity (see Theorem 4.2.2), and so forms the foundational hierarchy in this thesis, to which $\mathcal{PC}_k$ is then related.

4. Definition 3.6.2 and Definition 4.6.8 introduce the notions of (asymmetric) width-hardness from [104, 110] and the (new) associated $\mathcal{WC}_k$ hierarchy. While the $\mathcal{UC}_k$ hierarchy enforces bounds on tree-resolution complexity (see Lemma 3.2.4), $\mathcal{WC}_k$ places bounds on full-resolution complexity (see Definition 3.6.2), allowing greater scope for smaller translations under the stronger proof system (and SAT solvers that simulate such systems).

5. Definition 4.5.1 defines the notion of a k-base for optimising representations in $\mathcal{UC}_k$. A k-base for a clause-set F is a minimal equivalent representation $F \in \mathcal{UC}_k$ of F. With such optimal representations in mind, results are proven on the complexity of finding such representations (i.e., minimisation) in Section 4.5 of Chapter 4.

The main results on $\mathcal{UC}_k$, the $\mathcal{SLUR}_k$ hierarchy, and their relation to the $\mathcal{SLUR}$ class are:

1. Theorem 4.2.2 shows that the elements of $\mathcal{UC}_k$ are precisely the clause-sets F where every prime implicate of F can be derived by k-times nested input resolution from F.

2. Theorem 4.4.4 shows that $\mathcal{UC}_k = \mathcal{SLUR}_k$ holds. This brings together two key perspectives, the proof theoretic side, $\mathcal{UC}_k$, defined in terms of k-times-nested-input-resolution, and the algorithmic side, $\mathcal{SLUR}_k$, defined in terms of the algorithmic r_k definition. Furthermore, this creates a further intuitive connection between the concepts of knowledge compilation and poly-time SAT representations, where now one can think of "SAT knowledge compilation" – compiling boolean functions into good SAT representations with efficient detection of clausal-entailment.

3. Theorem 4.4.5 (via Theorem 4.4.4) demonstrates the coNP-completeness of membership decision for $\mathcal{UC}_k$ when $k \geq 1$.

4. Theorems 4.4.6, 4.4.7 prove that the previous hierarchies based on $\mathcal{SLUR}$ are (strictly) included in the $\mathcal{SLUR}_k$ hierarchy, which we consider as a kind of "completion", where both approaches, based on SLUR and UC, meet.

5. Regarding optimisation:

(a) Theorem 4.5.4 shows that for F in 2-CNF we can compute optimal equivalent clause-sets (of low hardness) in polynomial time.

(b) Theorem 4.5.5 shows that already for Horn clause-sets F, even when all prime implicates are given as part of the input, the decision whether there is an equivalent clause-set (of low hardness) using at most a given number of clauses is NP-complete.

Chapter 5 starts by introducing a conceptual framework linking prime implicates and essential clauses of "doped clause-sets" with minimally unsatisfiable sub-clause-sets (minimal premise sets from [116]), yielding methods for constructing clause-sets (and associated boolean functions) with well-structured and well-understood sets of prime implicates:

1. Theorem 5.1.18 shows the correlation between prime implicates of doped clause-sets and minimal premise-sets of the original (undoped) clause-sets.
2. Theorem 5.2.11 characterises unsatisfiable clause-sets where every non-empty sub-clause-set is a minimal premise set.
3. Theorem 5.2.18 gives basic characteristics of the special class of doped $\mathcal{SMU}_{\delta=1}$-clause-sets which are then used in the rest of Chapter 5 to prove a separation in terms of poly-size representability between each level of the $\mathcal{UC}_k$ hierarchy.

The main results on size lower bounds for $\mathcal{UC}_k$ are:

1. Theorem 5.3.4 introduces the basic method for lower bounding the size of equivalent clause-sets of a given hardness, via the transversal number of "trigger hypergraphs".
2. Theorem 5.3.12 shows a lower bound on the matching number of the trigger hypergraph of doped "extremal" $\mathcal{SMU}_{\delta=1}$-clause-sets.
3. Theorem 5.3.13 states the core separation result, showing that for every $k \in \mathbb{N}_0$ there are polysize clause-sets in $\mathcal{UC}_{k+1}$, where every equivalent clause-set in $\mathcal{WC}_k$ is of exponential size.

Turning to upper bounds, that is, short representations (with new variables) with low hardness, the following main results are shown:

1. Lemma 6.2.7 shows how the Tseitin translation applied to hitting (orthogonal) DNFs (called here the "canonical translation") can yield results in $\mathcal{UC}$.
2. Lemma 6.3.3 shows that by removing clauses from the Tseitin translation (representing only implication rather than equivalence), one can map any DNF to $\mathcal{UC}_k$ using new variables. This offers a possible explanation for differences in performance between using only one or both directions of the equivalences in the Tseitin-translation, noted in particular in [131, 93, 55].
3. Theorem 6.2.9 states that all doped $\mathcal{SMU}_{\delta=1}$-clause-sets (and in fact all doped unsatisfiable hitting clause-sets) have short CNF-representations in $\mathcal{UC}$ via the canonical translation.
4. Theorem 6.4.7 then shows that in general the Tseitin translation has unbounded hardness, using pairs of unsatisfiable XOR equations as an example.

In Chapter 7, experiments are then presented demonstrating that modern DPLL and CDCL SAT solvers can perform considerably better when one trades time for succinctness. Finally in the conclusion (Chapter 8), the contributions of the thesis are summarised and remaining conjectures and open questions are enumerated. Relations to existing work are discussed at the relevant locations throughout; in particular, comparisons to existing hierarchies and classes for knowledge compilation and polynomial time satisfiability occur in Chapter 4.

1.5.1 Related publications

The publications related to this thesis are:

- The content of Chapter 3 and Chapter 4 was first presented as a conference paper ([76]) at SOFSEM 2013. This paper won Best Paper award at the conference.
- A (longer) journal paper ([78]), featuring the contents of Chapter 3 and Chapter 4, was published in the Journal of Automated Reasoning.
- Results related to the hardness of representing XOR clause-sets in Section 6.4 of Chapter 6 are published as part of a conference paper [81], to appear at LATA 2014.
- The content of Chapters 5, 6 and 7 appear in technical reports ([79, 80]) which will be submitted as two further journal papers after this.

Furthermore, this work has been presented by the author at the following workshops and colloquia: BCTCS 2013, CP Doctoral Program 2011, and BCTCS 2011, and has also been presented (again by the author) at an invited talk at the Boolean Seminar 2013[6)] in Liblice.

[6)] http://ktiml.mff.cuni.cz/booleanseminar2013/index.php

Chapter 2

Preliminaries on clause-sets and boolean functions

The basic existing notions and notations used in this thesis are now introduced. In particular, boolean functions are introduced in Section 2.1, clause-sets are introduced in Section 2.2, resolution proofs are defined in Section 2.5, prime implicates and implicants are discussed in Section 2.6, relevant special classes of clause-sets are introduced in Section 2.7, and finally Constraint Programming and the connections to clause-sets and CNF satisfiability are discussed in Section 2.8. In each case, there are entire theories (along with associated handbooks) and so the definitions and discussions are, for the most part, kept to those that are required and relevant for this thesis.

2.1 Boolean functions

The majority of this thesis will be concerned with the representation of boolean functions and so the basic notion of a boolean function and the essential notations are now defined. For the most part, the notations are essentially standard. A particularly important point is the use of named variables rather than positional arguments (see Definition 2.1.1) which ties in perfectly with the notion of clause-set representations of boolean functions (see Section 2.2). For a full overview of the theory of boolean functions, see [42].

Definition 2.1.1 *A fixed universe $\mathcal{VA}$ of variables with $\mathbb{N} \subseteq \mathcal{VA}$ is assumed. Consider a finite set $V \subset \mathcal{VA}$ of variables.*

- *A **boolean function on** V is a map $f : \{0,1\}^V \to \{0,1\}$, and we set $\mathbf{var}(f) := V$.*
- *The number of variables is $n(f) := |V| \in \mathbb{N}_0$.*
- *The set of all boolean functions is denoted by $\mathcal{BF}$ (or, fully explicit, $\mathcal{BF}(\mathcal{VA})$), and the set of all boolean functions f with $\mathrm{var}(f) = V$ by $\mathcal{BF}^V$.*

Remarks:

1. Note that we take $\mathbb{N}$ to not contain 0, and so 0 is not a variable. This in general avoids confusion with negation often being represented as $-$ while $0 = -0$.
2. So arguments of boolean functions f are total assignments $\varphi : \mathrm{var}(f) \to \{0,1\}$.

3. For the sets of satisfying/falsifying assignments of a boolean function f the standard mathematical notation, namely $f^{-1}(1)$ resp. $f^{-1}(0)$, is used[1)].
4. $\mathbf{0}^V$ and $\mathbf{1}^V$ are used for the boolean functions with domain V which are constant 0 resp. 1, while $0 := 0^\emptyset$ and $1 := 1^\emptyset$ (in this context).
5. A variable $v \in \mathcal{VA}$ considered as boolean function is id_v (the identity on $\{v\}$). And complementation is interpreted as negation, that is, $\overline{v}$ considered as boolean function is $\neg\,\mathrm{id}_v$.
6. The boolean functions $0^V, 1^V$ are referred to as the *constant* 0 resp. *constant* 1 boolean functions.

So a boolean function takes some total assignment to its variables and returns 1 or 0. In this way, the boolean function $f : 0,1^V \to \{0,1\}$ acts as a characteristic function for the relation given by $f^{-1}(1)$. Via encoding non-boolean domains into boolean (for example, see Section 2.8), the boolean function (as a concept) can act as an elementary building block in many modelling tasks. For example, in representation of knowledge bases (see Section 2.7.2); in the design of cryptographic primitives (e.g., substitution boxes; see [44] for an overview on cryptographic boolean functions); in modelling and minimisation of electrical circuits (for example, analysis of prime implicates, prime implicants and "minimal covers" for circuit minimisation in VLSI design in [30]); and analysis in the social sciences (e.g., Qualitative Comparative Analysis (QCA); see [134, 136] for an overview). Particularly relevant to this thesis is the general modelling of constraint problems and problems in the NP complexity class as boolean functions via translation to satisfiability problems (a full overview of the SAT problem and related literature is available in [23] and a brief overview is given in the remainder of this chapter).

Important for the understanding and use of boolean functions in modelling is the notion of a partial assignment, allowing certain values of the function to be fixed and a new boolean function generated. A simple example of the need for this is in the construction of knowledge bases. For example, in modelling medical knowledge bases, the presence of absence of many factors may contribute to a patient being diagnosed with a disease X and this knowledge can be represented as a boolean function. When certain factors are known to be present for a particular patient, the boolean function can then be instantiated with the partial assignment corresponding to those factors, and a new boolean function derived for their specific case (for more examples of representing medical knowledge bases as (partial) boolean functions, see Logical Analysis of Data in [83, 28]). The application of partial assignments is also vital in satisfiability and constraint programming, where backtracking algorithms (see e.g. DPLL as described in [88] and CDCL solvers as described in [121]) rely on the ability to build up partial assignments and evaluate the result.

Definition 2.1.2 *A **partial assignment** is a map $\varphi : V \to \{0,1\}$ for some finite $V \subset \mathcal{VA}$, and we set $\mathbf{var}(\varphi) := V$. The set of all partial assignments is denoted by $\mathcal{PASS}$, while for $V \subseteq \mathcal{VA}$ we define $\mathcal{PASS}(V) := \{\varphi \in \mathcal{PASS} : \mathrm{var}(\varphi) \subseteq V\} \subseteq \mathcal{PASS}$. The empty partial assignment is denoted by $\langle\rangle \in \mathcal{PASS}$. Relative to some finite $V \subset \mathcal{VA}$ one calls a partial assignment φ with $\mathrm{var}(\varphi) = V$ a **total assignment**. For a variable $v \in \mathrm{var}(\varphi)$ we define $\varphi(\overline{v}) := 1 - \varphi(v)$. Two partial assignments φ, ψ are **consistent**, if for all $v \in \mathrm{var}(\varphi) \cap \mathrm{var}(\psi)$ we have $\varphi(v) = \psi(v)$; otherwise they are called **inconsistent**.*

Definition 2.1.3 *The composition $\circ : \mathcal{PASS} \times \mathcal{PASS} \to \mathcal{PASS}$, denoted by $\psi \circ \varphi \in \mathcal{PASS}$ for $\varphi, \psi \in \mathcal{PASS}$, is defined as follows:*

[1)] The standard notation often used is actually $f^{-1}(\{0\})$, but a small liberty is taken for notational convenience.

1. $\mathrm{var}(\psi \circ \varphi) = \mathrm{var}(\varphi) \cup \mathrm{var}(\psi)$

2. $(\psi \circ \varphi)(v) := \varphi(v)$ *if* $v \in \mathrm{var}(\varphi)$, *while otherwise* $(\psi \circ \varphi)(v) := \psi(v)$.

Definition 2.1.4 *The operation* $* : \mathcal{PASS} \times \mathcal{BF} \to \mathcal{BF}$ *of partial assignments on boolean functions, denoted by* $\varphi * f \in \mathcal{BF}$ *for* $\varphi \in \mathcal{PASS}$ *and* $f \in \mathcal{BF}$, *is defined as follows:*

1. $\mathrm{var}(\varphi * f) = \mathrm{var}(f) \setminus \mathrm{var}(\varphi)$

2. *for* $\psi : \mathrm{var}(\varphi * f) \to \{0,1\}$ *let* $(\varphi * f)(\psi) := f(\psi \circ \varphi) = f(\varphi \circ \psi)$.

Semantic entailment is then defined for boolean functions in the standard way:

Definition 2.1.5 *Consider two boolean functions* $f, g \in \mathcal{BF}$. *We say that* $f \models g$ *iff for all* $\varphi \in \{0,1\}^{\mathrm{var}(f) \cup \mathrm{var}(g)}$ *we have that* $f(\varphi) = 1 \Rightarrow g(\varphi) = 1$. *We say that* $f \cong g$ *if* $f \models g$ *and* $g \models f$.

Application of partial assignments is compatible with the standard boolean operations ($\wedge$ for conjunction ("and"), $\vee$ for disjunction ("or"), and $\neg$ for negation ("not")) in the following sense:

Lemma 2.1.6 *For a partial assignment* φ, *a finite set* V *of variables, and boolean functions* f, g *we have:*

1. $\varphi * 0^V = 0^{V \setminus \mathrm{var}(\varphi)}$

2. $\varphi * 1^V = 1^{V \setminus \mathrm{var}(\varphi)}$

3. $\varphi * (\neg f) = \neg(\varphi * f)$

4. *For every truth-functional composition* $\square$ *of boolean functions (i.e.,* $(f \square g)(x)$ *depends only on* $f(x)$ *and* $g(x)$*) we have* $\varphi * (f \square g) = (\varphi * f) \square (\varphi * g)$. *So*

   (a) $\varphi * (f \wedge g) = (\varphi * f) \wedge (\varphi * g)$

   (b) $\varphi * (f \vee g) = (\varphi * f) \vee (\varphi * g)$

   (c) $\varphi * (f \to g) = (\varphi * f) \to (\varphi * g)$

   (d) $\varphi * (f \leftrightarrow g) = (\varphi * f) \leftrightarrow (\varphi * g)$

   (e) $\varphi * (f \oplus g) = (\varphi * f) \oplus (\varphi * g)$.

Taking these operations as a basis, we can form arbitrary propositional formulae (e.g., $(a \vee \neg b) \wedge (\neg a \vee (\neg b \wedge c))$) which then represent the underlying boolean function. Central to the topic of this thesis are Conjunctive and Disjunctive Normal Forms.

Definition 2.1.7 *A propositional formula is in* ***C**onjunctive **N**ormal **F**orm (**CNF**) if it is the conjunction of disjunctions of literals (i.e., variables or their negations). A proposition formula is in* ***D**isjunctive **N**ormal **F**orm (**DNF**) if it is the disjunction of conjunctions of variables or their negations.*

Remarks:

1. So, for example $(a \vee b) \wedge (c \vee d) \wedge (a \vee \neg c)$ is a CNF, and $(a \wedge c \wedge d) \vee (\neg b \wedge \neg c \wedge d) \vee (a \wedge b)$ is a DNF.

It is a well-known fact that every boolean function has a representation as a CNF formula and as a DNF formula in the following sense.

Lemma 2.1.8 *Consider a boolean function $f : \{0,1\}^V \to \{0,1\}$. We have that*

$$f = \bigwedge_{\varphi \in f^{-1}(0)} \left(\bigvee_{\substack{v \in V \\ \varphi(v)=0}} v \vee \bigvee_{\substack{v \in V \\ \varphi(v)=1}} \neg v \right).$$

$$f = \bigvee_{\varphi \in f^{-1}(1)} \left(\bigvee_{\substack{v \in V \\ \varphi(v)=1}} v \wedge \bigvee_{\substack{v \in V \\ \varphi(v)=0}} \neg v \right).$$

In this way, the CNF representation encodes that for *all* falsifying assignments φ (i.e., unfolding $\forall$ to a conjunction) for f that f is not true under φ (note here that via DeMorgan's law we have $\neg(v_1 \wedge \cdots \wedge v_n) = \neg v_1 \vee \ldots \vee \neg v_n$), and the DNF representation encodes that for f to be true *one* of the satisfying assignments must be true (i.e., unfolding $\exists$ to a disjunction).

2.2 Clause-sets

The notion of a clause-set provides a convenient abstraction for representing both Conjunctive and Disjunctive Normal Forms (CNFs and DNFs) by forgetting details of boolean operations and ordering and keeping only the underlying conflict structure (i.e., literals occurring positively or negatively).

Definition 2.2.1 *We consider a fixed universe $\mathcal{VA}$ of variables with $\mathbb{N} \subseteq \mathcal{VA}$.*

- *The set $\mathcal{LIT}$ of literals is structured by complementation, a self-inverse bijection $x \in \mathcal{LIT} \mapsto \overline{x} \in \mathcal{LIT}$, such that for every $x \in \mathcal{LIT}$ we have exactly one of $x \in \mathcal{VA}$ (a positive literal) or $\overline{x} \in \mathcal{VA}$ (a negative literal), and such that for $v \in \mathbb{N}$ we have $\overline{v} = -v$.*
- *A clause is a finite set $C \subset \mathcal{LIT}$ of literals without clashes, that is, $\overline{C} \cap C = \emptyset$ with $\overline{C} := \{\overline{x} : x \in C\}$; the set of all clauses is denoted by $\mathcal{CL}$.*
- *A finite clause-set F is a finite set of clauses, the set of all finite clause-sets is denoted by $\mathcal{CLS}$. A special clause is $\bot := \emptyset \in \mathcal{CL}$, a special clause-set is $\top := \emptyset \in \mathcal{CLS}$.*

The following core measures and notations are used throughout:

1. *Let $\mathbf{var} : \mathcal{LIT} \to \mathcal{VA}$ be defined by $\mathrm{var}(v) := v$ and $\mathrm{var}(\overline{v}) := v$ for $v \in \mathcal{VA}$. For a set L of literals let $\mathrm{var}(L) := \{\mathrm{var}(x) : x \in L\}$, and for a set G of sets of literals let $\mathrm{var}(G) := \bigcup_{L \in G} \mathrm{var}(L)$. And for a set V of variables let $\mathbf{lit}(V) := V \cup \overline{V}$.*
2. *For $F \in \mathcal{CLS}$ let $\boldsymbol{n}(F) := |\mathrm{var}(F)| \in \mathbb{N}_0$ be the number of variables of F, $\boldsymbol{c}(F) := |F| \in \mathbb{N}_0$ be the number of clauses of F, $\boldsymbol{\ell}(F) := \sum_{C \in F} |C| \in \mathbb{N}_0$ be the number of literal occurrences in F.*

3. *For $V \subseteq \mathcal{VA}$ and $F \in \mathcal{CLS}$ we define* $\boldsymbol{V * F} := \{C \setminus (V \cup \overline{V}) : C \in F\}$.

An isomorphism from a clause-set F to a clause-set G is a bijection $\alpha : \mathrm{lit}(F) \to \mathrm{lit}(G)$ for which (a) the map $C \in F \mapsto \{\alpha(x) : x \in C\} \in G$ is also a bijection (i.e., α is a hypergraph isomorphism on the underlying hypergraphs F and G) and (b) α respects literal complementation, i.e., $\alpha(\overline{x}) = \overline{\alpha(x)}$.

Remarks:

1. So essentially $\mathcal{LIT}$ is just $\mathcal{VA}$ extended by a set of "negations" for every element in $\mathcal{VA}$ with the "negation" operation defined such that whatever $\mathcal{VA}$ can't contain both an element an its negation.
2. We do not have (in general) $0 \in \mathcal{VA}$ (recall we do not have $0 \in \mathbb{N}$), since $-0 = 0$ [2)].
3. $\mathrm{var}(x)$ is the *underlying variable* of literal x, and accordingly $\mathrm{var}(C)$ and $\mathrm{var}(F)$ for clauses C and clause-sets F are the sets of underlying variables.
4. $\mathrm{lit}(V)$ is the set of literals created from V, and for the set of all literals we have $\mathcal{LIT} = \mathrm{lit}(\mathcal{VA})$. We set $\mathrm{lit}(C) := \mathrm{lit}(\mathrm{var}(C))$ and $\mathrm{lit}(F) := \mathrm{lit}(\mathrm{var}(F))$ for clauses C and clause-sets F. In some cases, we will wish to directly reference the set of literals occurring in F, in which case we will denote this by $\mathrm{occlit}(F) := \bigcup_{C \in F} C$.
5. We call a clause-set **full** if every clause contains all variables, i.e., for all $C \in F$ we have $\mathrm{var}(C) = \mathrm{var}(F)$.

2.2.1 Partial assignments and the semantics of clause-sets

To understand the semantics of clause-sets and their correspondence to boolean functions, it is necessary to understand the application of partial assignments (as at the boolean function level) on clause-sets.

Definition 2.2.2 *The operation $* : \mathcal{PASS} \times \mathcal{CLS} \to \mathcal{CLS}$ of partial assignments on clause-sets, denoted by $\varphi * \boldsymbol{F} \in \mathcal{CLS}$ for $\varphi \in \mathcal{PASS}$ and $F \in \mathcal{CLS}$, is obtained from F in two steps:*

1. *All clauses $C \in F$ containing a literal $x \in C$ with $\varphi(x) = 1$ are removed.*
2. *From the remaining clauses all literals x with $\varphi(x) = 0$ are removed.*

Remarks:

1. See [100] for more on partial assignments and their operations.
2. A clause-set F is **satisfiable** iff there exists a partial assignment φ (the **satisfying assignment**) with $\varphi * F = \top$. Note that here only the CNF-interpretation of clause-sets is used.

[2)] Of course, one can choose to have $0 \in \mathcal{VA} \setminus \mathbb{N}$ but then one must (counter-intuitively) choose some non-0 negation/"inverse" for 0.

It is now possible to introduce the CNF and DNF semantics of clause-sets:

Definition 2.2.3 *For a clause-set F, we use* $\mathbf{CNF}(F)$ *resp.* $\mathbf{DNF}(F)$ *for the corresponding boolean functions* $\{0,1\}^{\mathrm{var}(F)} \to \{0,1\}$*:*

$$\begin{aligned} \mathrm{DNF}(F) &:= \bigvee_{C \in F} \bigwedge_{x \in C} x \\ \mathrm{CNF}(F) &:= \bigwedge_{C \in F} \bigvee_{x \in C} x. \end{aligned}$$

Remarks:

1. For clause-sets F we have $\mathrm{var}(\mathrm{DNF}(F)) = \mathrm{var}(\mathrm{CNF}(F)) = \mathrm{var}(F)$.
2. Regarding the clause-sets without variables we have:
   (a) $\mathrm{DNF}(\top) = \mathrm{CNF}(\{\bot\}) = 0$
   (b) $\mathrm{DNF}(\{\bot\}) = \mathrm{CNF}(\top) = 1$.

The CNF resp. DNF interpretations of clause-sets yield a mapping between clauses of the clause-set and subsets of $f^{-1}(0)$ resp. $f^{-1}(1)$. This mapping has a certain geometric aspect, in that each clause in the clause-set defines a sub-space in the hypercube $\{0,1\}^V$ (see Section 1.9 of Chapter 1 in [42] for more detail). This map is map explicit in the map from clauses to partial assignments in Definition 2.2.4.

Definition 2.2.4 *For a clause C let the partial assignment φ_C be specified by*

1. $\mathrm{var}(\varphi_C) = \mathrm{var}(C)$
2. $\varphi_C(x) = 0 \Leftrightarrow x \in C$ *for* $x \in \mathrm{lit}(C)$.

More generally for $\varepsilon \in \{0,1\}$:

$$\begin{aligned} \boldsymbol{\varphi}_C^{\varepsilon} &:= \langle x \to \varepsilon : x \in C \rangle \\ \boldsymbol{C}_{\varphi}^{\varepsilon} &:= \{x \in \mathrm{occlit}(\mathrm{var}(\varphi)) : \varphi(x) = \varepsilon\}. \end{aligned}$$

And conversely, for a partial assignment φ let the clause $\boldsymbol{C}_{\varphi}$ be given as $C_\varphi := \{x \in \mathrm{occlit}(\mathrm{var}(\varphi)) : \varphi(x) = 0\}$.

Remarks:

1. $\varphi_\bot = \langle\rangle$.
2. φ, ψ are consistent iff $C_\varphi \cap \overline{C_\psi} = \emptyset$ (that is, the corresponding clauses do not clash).
3. $\varphi_C^0 = \varphi_C$ and $C_\varphi^0 = C_\varphi$.

As with CNF formulae, for each boolean function there are canonical clause-sets associated with the canonical CNF and DNF formulas.

Definition 2.2.5 *For a boolean function f let* $\mathbf{CNF}(f) \in \mathcal{CLS}$ *denote the* ***distinguished conjunctive normal form*** *of f (containing the "max terms"), and let* $\mathbf{DNF}(f) \in \mathcal{CLS}$ *denote the* ***distinguished disjunctive normal form*** *of f (containing the "min terms"):*

$$\begin{aligned} \mathrm{CNF}(f) &:= \{C_\varphi : \varphi \in \mathcal{PASS} \wedge \mathrm{var}(\varphi) = \mathrm{var}(f) \wedge f(\varphi) = 0\} \\ \mathrm{DNF}(f) &:= \{C_\varphi^1 : \varphi \in \mathcal{PASS} \wedge \mathrm{var}(\varphi) = \mathrm{var}(f) \wedge f(\varphi) = 1\}. \end{aligned}$$

Remarks:

1. We have $\mathrm{CNF}(\mathrm{CNF}(f)) = f$ as well as $\mathrm{DNF}(\mathrm{DNF}(f)) = f$.

The default interpretation of a clause-set F is as a CNF, which can be emphasised by speaking of the "CNF-clause-set F", that is, the interpretation as a boolean function is

$$F \rightsquigarrow \bigwedge_{C \in F} \bigvee_{x \in C} x.$$

We might consider F also as a DNF-clause-set, which does not change F itself, but only changes the interpretation of F in considerations regarding the semantics:

$$F \rightsquigarrow \bigvee_{C \in F} \bigwedge_{x \in C} x.$$

Note that by the de Morgan rules from the CNF-formula we obtain the DNF-formula via negating the whole formula together with negating the literals (in other words, the underlying boolean function of a CNF-clause-set F is the "dual" of the underlying boolean function of the DNF-clause-set F; see [42]). Thus the logical negation (as CNF) of a clause-set F (as CNF) is obtained from a DNF-clause-set equivalent to F by negating all literals.

Example 2.2.6 *The clause-set $F = \{\{v\}\}$ has the equivalent DNF-clause-set $F = \{\{v\}\}$ (the underlying boolean function is "self-dual"; see [42]), while the negation is $\{\{\overline{v}\}\}$. And $F = \{\{v, w\}\}$ has the equivalent DNF-clause-set $\{\{v\}, \{w\}\}$, while the negation is $\{\{\overline{v}\}, \{\overline{w}\}\}$.*

While clause-sets and partial assignments themselves are neutral regarding CNF- or DNF-interpretation, the application $\varphi * F$ is based on the CNF-interpretation of F; if we wish to use the DNF-interpretation of F, then we use $\overline{\varphi} * F$, where $\overline{\varphi} := \langle v \to \overline{\varphi(v)} : v \in \mathrm{var}(\varphi) \rangle$. While $\top$ in the CNF-interpretation stands for "true", in the DNF-interpretation it becomes "false".

Example 2.2.7 *Consider $F := \{\{a\}, \{b\}\} \in \mathcal{CLS}$ (with $n(F) = c(F) = \ell(F) = 2$). Then $\mathrm{DNF}(F) = \{\{a, b\}\}$, and for $\varphi := \langle a, b \to 1 \rangle$ we have $\varphi * F = \top$. This corresponds to the CNF-interpretation $a \wedge b$ of F, which has exactly one satisfying assignment φ. If we consider the DNF-interpretation $a \vee b$ of F, then we have three satisfying total assignments for the DNF-clause-set F, and for example the satisfying assignment $\psi := \langle a \to 1 \rangle$ is recognised via $\overline{\psi} * F = \langle a \to 0 \rangle * F = \{\bot, \{b\}\}$, where the result as DNF is a tautology, since $\bot$ as a DNF-clause becomes the constant 1 (as the empty conjunction).*

2.3 Forced literals/assignments

Fundamental is the notion of a "forced literal" of a boolean function resp. a clause-set[3], which are literals which must be set to true in order to satisfy the function resp. clause-set:

Definition 2.3.1 *A literal x is **forced** for a boolean function f if $f \models x$, and the set of forced literals for f is $\mathrm{fl}(f) \subseteq \mathcal{LIT}$. A literal is forced for a clause-set F if it is forced for $\mathrm{CNF}(F)$, and we set $\mathrm{fl}(F) := \mathrm{fl}(\mathrm{CNF}(F))$.*

[3] we prefer this logical (and common) terminology over "backbone literal", which is only used in a special context

Every literal is forced for every 0^V. In fact a boolean function f is constant zero iff $\mathrm{fl}(f) = \mathcal{LIT}$ iff there is a literal x with $x, \overline{x} \in \mathrm{fl}(f)$. No literal is forced for any 1^V (i.e., $\mathrm{fl}(1^V) = \emptyset$). We have for every boolean function f that

$$\mathrm{fl}(f) = \bigcap\nolimits_{\mathcal{LIT}} \mathrm{DNF}(f)$$

(the index "$\mathcal{LIT}$" in the intersection is the "universe" of the sets considered in the intersection, which becomes the result if there are no sets to intersect, that is, if f is unsatisfiable).

Example 2.3.2 *Some basic determinations of* $\mathrm{fl}(F)$ *are:*

1. $\mathrm{fl}(\{\bot\}) = \mathcal{LIT}$.
2. $\mathrm{fl}(\top) = \emptyset$.
3. $\mathrm{fl}(\{\{x_1\}, \ldots, \{x_n\}\}) = \{x_1, \ldots, x_n\}$.
4. $\mathrm{fl}(\{\{x, \overline{y}\}, \{\overline{x}, y\}\}) = \emptyset$.
5. $\mathrm{fl}(\{\{x, y\}, \{x, \overline{y}\}\}) = \{x\}$.

If x is a forced literal for F, then the **forced assignment** $\langle x \to 1 \rangle$ yields the clause-set $\langle x \to 1 \rangle * F$ which is satisfiability-equivalent to F. We denote by $\mathbf{r_\infty(F)} \in \mathcal{CLS}$ the result of applying all forced assignments to F. Note that F is unsatisfiable iff $\mathrm{r}_\infty(F) = \{\bot\}$ (while F is uniquely satisfiable after discarding variables without influence iff $\mathrm{r}_\infty(F) = \top$).

2.4 Reductions

In 1962, Davis, Putnam, Logemann and Loveland introduced the DPLL algorithm in [47], based on earlier work by Davis and Putnam in [48]. This significantly improves on the simple brute-force techniques for solving satisfiability and opens the field for "intelligent backtracking" by introducing two *reduction* techniques *unit-clause propagation* and *pure literal elimination*, used at every node in the backtracking search tree (see [88] for an overview on backtracking in SAT) to avoid unnecessary search.

Definition 2.4.1 *For clause-sets F, F' the relation $\boldsymbol{F \supseteq^\mapsto F'}$ holds if for all $C \in F$ there is $C' \in F'$ with $C' \subseteq C$; we say that F' **strengthens** F. A **reduction** in this context is a map $r : \mathcal{CLS} \to \mathcal{CLS}$ such that for all $F, F' \in \mathcal{CLS}$ we have*

1. *$r(F)$ is satisfiability-equivalent to F;*
2. *if $\bot \in r(F)$ and F' strengthens F then $\bot \in r(F')$.*

*A reduction r **discovers** unsatisfiability of F if $\bot \in r(F)$.*

Definition 2.4.2 *Consider a reduction r. The relation $\boldsymbol{F \models_r C}$ holds for a clause-set F and a clause C, and we say C **is deducible from** F **via** r, if r discovers unsatisfiability of $\varphi_C * F$ (that is, $\bot \in r(\varphi_C * F)$ for $\varphi_C = \langle x \mapsto 0 : x \in C \rangle$).*

Pure literal elimination removes literals (and the clauses containing them) which occur in only one polarity. These literals can be set while maintaining satisfiability, and via the removal of the associated clauses, potentially hard sub-clause-sets are removed, reducing back-tracking.

Definition 2.4.3 *Consider a clause-set F and a literal $x \in \operatorname{occlit}(F)$. x is* ***pure*** *if $\overline{x} \notin \operatorname{occlit}(F)$.* ***Pure literal propagation*** *is defined as follows*

$$\operatorname{rp}(F) := \begin{cases} \operatorname{rp}(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \operatorname{occlit}(F) : x \text{ is pure in } F \\ F & \text{otherwise} \end{cases}.$$

Particularly important to the work in this thesis, **unit-clause propagation (UCP)** propagates the most simple forced literals in a clause-set, namely those which occur in singleton disjunctions. We will see a generalisation of UCP in Definition 3.1.1 of Chapter 3.

Definition 2.4.4 *Consider a clause-set F.* ***Unit-clause propagation*** *$\mathrm{r}_1 : \mathcal{CLS} \to \mathcal{CLS}$ is the reduction defined as follows*

$$\mathrm{r}_1(F) := \begin{cases} \{\bot\} & \text{if } \bot \in F \\ \mathrm{r}_1(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \operatorname{lit}(F) : \bot \in \langle x \to 0\rangle * F \\ F & \text{otherwise} \end{cases}.$$

Remarks:

1. The map $\mathrm{r}_1 : \mathcal{CLS} \to \mathcal{CLS}$ is well-defined (does not depend on the choices)[4].
2. r_1 applies only forced assignments (and so $\mathrm{r}_1(F)$ is satisfiability-equivalent to F).
3. $\mathrm{r}_1(F)$ is computable in time $O(\ell(F))$ and linear space (see [53]) by using clause-variable graphs to avoid visiting "untouched" clauses during propagation. Watched-literal data-structures, introduced in [125], further improve on such graph-based algorithms by watching 2 literals in every clause, and only taking action when one of these is set (implying that a unit-clause might have been created).
4. $\mathrm{r}_1(F) = \{\bot\}$ implies $\mathrm{r}_1(\varphi * F) = \{\bot\}$.
5. $\mathrm{r}_1(\varphi * \mathrm{r}_1(F)) = \mathrm{r}_1(\varphi * F)$.
6. r_1 is used in all modern DPLL and CDCL solvers which dominate the best performing SAT solvers in international SAT competitions (see Section 7.2 for discussion on state of the art solvers and their performance on instances in this thesis).

Example 2.4.5 *Consider the following clause-set and the corresponding applications of r_1:*

$$\{\ \underbrace{\{a\}}_{\textit{unit-clause}}, \{\overline{a}, b\}, \{\overline{b}\}\ \} \xrightarrow{\langle a \to 1\rangle} \{\ \{b\}, \{\overline{b}\}\ \} \xrightarrow{\langle b \to 1\rangle} \{\ \bot\ \}.$$

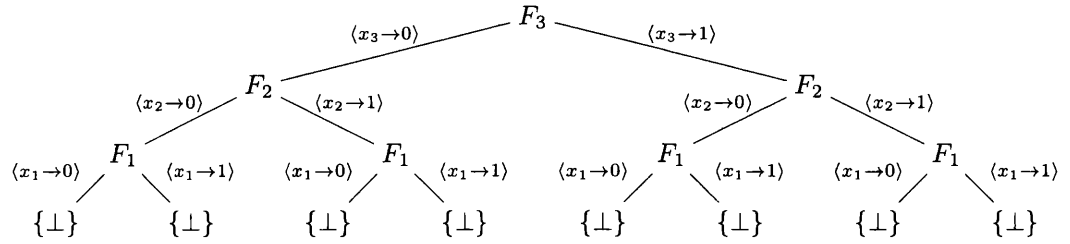
The utility of unit-clause propagation is demonstrated in Example 2.4.6, where a clause-set is shown for which r_1 determines unsatisfiability in linear time, but without r_1 worst case exponential backtracking trees are possible.

[4] This follows essentially via the diamond lemma ([126, 92]) and the fact that any rewrite system which sets only forced literals is locally confluent (setting one forced literal to true can't remove another forced literal) and terminating - see e.g., Lemma 3.13 in [104].

Example 2.4.6 *Consider $n \in \mathbb{N}$ and the unsatisfiable clause-set*

$$F_n := \left(\bigcup_{i \in \{1,\ldots,n-1\}} \{ \{x_1, \ldots, x_{i-1}, \overline{x_i}\} \} \right) \cup \{ \{x_1, \ldots, x_n\}, \{x_1, \ldots, x_{n-1}, \overline{x_n}\} \}$$

We have that $\langle x_n \to 0\rangle * F_n = \langle x_n \to 1\rangle * F_n = F_{n-1}$ and hence by induction the backtracking tree formed by taking the fixed variable-order heuristic $x_n, \ldots, x_1$ (i.e., always choose the "largest" x_i) has size 2^n. However, $\mathrm{r}_1(F_n) = \{\bot\}$ and r_1 runs in linear time. This shows a worst-case separation between backtracking with and without r_1. So for example, with $n = 3$ we have:



compared to the following 3 steps for r_1:

$$\mathrm{r}_1(F_3) = \mathrm{r}_1(\underbrace{\langle x_1 \to 0\rangle * F_3}_{\{\{\overline{x_2}\}, \{x_2, x_3\}, \{x_2, \overline{x_3}\}\}}) = \mathrm{r}_1(\underbrace{\langle x_2 \to 0, x_1 \to 0\rangle * F_3}_{\{\{x_3\}, \{\overline{x_3}\}\}}) = \{\bot\}.$$

2.5 Resolution

The resolution calculus was introduced in 1965 by Robinson in [138]. The resolution rule allows clauses, essentially representing implications such as $A \to b$ and $b \to C$ for arbitrary formulas A and C and literal b, to be resolved to yield the clause representing $A \to C$.

Definition 2.5.1 *Two clauses C, D are **resolvable** if $|C \cap \overline{D}| = 1$, i.e., they clash in exactly one variable:*

- *For two resolvable clauses C and D the **resolvent** $C \diamond D := (C \cup D) \setminus \{x, \overline{x}\}$ for $C \cap \overline{D} = \{x\}$ is the union of the two clauses minus the resolution literals.*
- *x is called the **resolution literal**, while $\mathrm{var}(x)$ is the **resolution variable**.*

Remarks:

1. If x is the resolution literal of C, D, then $x \in C$, and $\overline{x}$ is the resolution literal of D, C.

Every resolvent clause derived by the resolution rule is a logical consequence of the parent clauses, and in [138] Robinson showed that resolution is in fact complete for propositional logic. That is, for any clause-set F if a clause C logically follows from F (i.e., $\mathrm{CNF}(F) \models \mathrm{CNF}(\{C\})$ then C can be derived from F via a series of resolution steps.

Definition 2.5.2 *A **resolution tree** is a pair $R = (T, C)$ such that:*

- *T is an ordered rooted tree, where every inner node has exactly two children, and where the set of nodes is denoted by $\mathrm{nds}(T)$, the root by $\mathrm{rt}(T) \in \mathrm{nds}(T)$, and the set of leaves by $\mathrm{lvs}(T) \subseteq \mathrm{nds}(T)$.*

- *While* $C : \mathrm{nds}(T) \to \mathcal{CLS}$ *labels every node with a clause such that the label of an inner node is the resolvent of the labels of its two parents.*

We use:

- $\boldsymbol{F(R)} := \{C(w) : w \in \mathrm{lvs}(T)\} \in \mathcal{CLS}$ *for the "axioms" (or "premisses") of* R*;*
- $\boldsymbol{C(R)} := C(\mathrm{rt}(T)) \in \mathcal{CL}$ *as the "conclusion";*

A ***resolution proof*** R *of a clause* C *from a clause-set* F*, denoted by* $\boldsymbol{R : F \vdash C}$*, is a resolution tree* $R = (T, C)$ *such that* $F(R) \subseteq F$ *and* $C(R) = C$*. We use* $\boldsymbol{F \vdash C}$ *if there exists a resolution proof* R *of some* $C' \subseteq C$ *from* F *(i.e.,* $R : F \vdash C'$*). A* ***resolution refutation*** *of a clause-set* F *is a resolution proof deriving* $\bot$ *from* F*. A resolution proof* $T : F \vdash C$ *is* ***regular*** *iff along any path*

A resolution tree $T : F \vdash C$ *is* ***regular*** *iff no resolution step reintroduces a literal which has been resolved away earlier* [5)]

While the resolution rule is complete for propositional logic, a fundamental question is whether it is a practical proof system for showing unsatisfiability (for CNFs) or tautology (for DNFs). That is, given a CNF (or dually a DNF) F, how quickly can one find a resolution proof for $F \vdash C$? A *lower bound* on the time for such computation is then the size of the smallest resolution proof that exists. There are two main measures for the complexity of resolution proofs.

Definition 2.5.3 *Consider a resolution proof* R*.*

- *The* ***tree-resolution complexity*** $\boldsymbol{\mathrm{Comp}^*_{\mathrm{R}}(R)} \in \mathbb{N}$ *is the number of nodes in* R *(the "tree size"), that is,* $\mathrm{Comp}^*_{\mathrm{R}}(R) := \#\mathrm{nds}(R) = |\mathrm{nds}(T)|$*.*
- *The* ***resolution complexity*** $\boldsymbol{\mathrm{Comp}_{\mathrm{R}}(R)} \in \mathbb{N}$ *is the number of* distinct *clauses in* R *(the "dag size"), that is* $\mathrm{Comp}_{\mathrm{R}}(R) := c(\widehat{\mathrm{F}}(R))$*.*

Finally, for $F \in \mathcal{USAT}$ *we set*

- $\boldsymbol{\mathrm{Comp}^*_{\mathrm{R}}(F)} := \min\{\mathrm{Comp}^*_{\mathrm{R}}(R) \mid R : F \vdash \bot\} \in \mathbb{N}$
- $\boldsymbol{\mathrm{Comp}_{\mathrm{R}}(F)} := \min\{\mathrm{Comp}_{\mathrm{R}}(R) \mid R : F \vdash \bot\} \in \mathbb{N}$.

Remarks:

1. Typically we identify R with T, while suppressing the labelling C.

Tree-resolution complexity is important because the tree-structure allows for simple inductive lower bound proofs (for example, see Chapter 3). However, in [71], Geordt proved that there are families of CNF clause-sets with poly-size proofs in resolution but only superpolynomial size regular resolution proofs, a superpolynomial separation between regular and full resolution. Later in [1], it was shown that there is an exponential separation between regular and full resolution, hence also showing an exponential separation for tree-resolution and full-resolution (as shown

[5)] If the empty clause is derived, then this is equivalent to the property, that along each path from the root to some leaf no resolution variable is used twice. Every resolution tree $T : F \vdash C$ can be regularised (as first mentioned in [156] (Theorem 3)), obtaining $T' : F \vdash C'$ with $C' \subseteq C$, where the underlying tree of T' is obtained from T by cutting off some branches (thus the number of nodes and the height are not increased). This works by removing such resolution steps and keeping one of the parent clause, where the resolution literal would be reintroduced later; see Subsection 5.2.1 in [110] for details.

in Lemma 5.1 of [157] minimal size tree-resolution refutations are regular). Every backtracking tree produced by a DPLL SAT solver can be directly translated to a tree-resolution proof, and so such lower bounds on tree-resolution complexity guarantee that there are CNFs which are "hard" for DPLL SAT solvers but not necessarily for systems based on full-resolution or stronger proof systems.

In [144] in 1999, Marques-Silva and Sakallah introduce the first Conflict-Driven Clause Learning (CDCL) solver, breaking the direct connection between SAT solvers and tree-resolution by allowing the solver to learn new clauses characterising failed backtracking attempts; for more details see [121]. Furthermore, in 2009 in [4], Atserias, Fichte and Thurley showed that idealised (yet practical) models of CDCL solvers can polynomially simulate full-resolution, hence showing that there are CNF clause-sets for which DPLL solvers must perform worse than CDCL.

One might think that such exponential separations between tree-resolution and full-resolution imply that tree-resolution should be ignored in favour of full-resolution, despite the stricter structure allowing for easier upper and lower bounds proofs. However, any tree-resolution complexity upper-bounds on families of CNF clause-sets are also full-resolution upper-bounds, making tree-resolution a good candidate for offering stronger guarantees about proof length and structure. Also, the separation between tree-resolution and full resolution is for specific families of CNF *clause-sets*; once one allows the introduction of new variables as in the Extended Resolution proof system, introduced by Tseitin in [156], we have that extended tree-resolution can polynomially simulate extended full resolution. In fact, despite the existence of families of CNF clause-sets with exponential lower bounds for full-resolution (see [82]), there are no such lower bounds for extended resolution, and in fact extended resolution can polynomially simulate powerful proof systems such as extended Frege, as pointed out in Theorem 8.1 of [157].

2.6 Prime implicates and implicants

A boolean function can have many CNF clause-set representations, and so when translating a boolean function to a clause-set representation the task becomes one of optimisation. How to find short representations of the boolean function? A key concept in understanding short representations of boolean functions f is that of its prime implicates and implicants – the clauses (resp. partial assignments, see Definition 2.2.4) of minimal size which follow from f under the CNF resp. DNF interpretations (of the clause).

Definition 2.6.1 *Consider a boolean function f.*

- *Let* $\mathbf{prc_0(f)}$ *be the set of all $C \in \mathcal{CL}$ which as CNF are* ***prime implicates*** *of f, that is, $f \models \mathrm{CNF}(\{C\})$ while $f \not\models \mathrm{CNF}(\{C'\})$ for every $C' \subset C$.*
- *Let* $\mathbf{prc_1(f)}$ *be the set of all $C \in \mathcal{CL}$ which as DNF are* ***prime implicants*** *of f, that is, $\mathrm{DNF}(\{C\}) \models f$ while $\mathrm{DNF}(\{C'\}) \not\models f$ for every $C' \subset C$.*

Remarks:

1. Regarding the extreme cases we have for all finite $V \subset \mathcal{VA}$:
   (a) $\mathrm{DNF}(0) = \mathrm{prc}_1(0) = \mathrm{CNF}(1) = \mathrm{prc}_0(1) = \top$
   (b) $\mathrm{DNF}(1) = \mathrm{prc}_1(1) = \mathrm{CNF}(0) = \mathrm{prc}_0(0) = \{\bot\}$.
2. Consider literals $x_0, x_1, \ldots, x_n$, $n \in \mathbb{N}_0$, with $\mathrm{var}(x_i) \neq \mathrm{var}(x_j)$ for $i \neq j$:
   (a) $\mathrm{prc}_0(\bigvee_{i=1}^n x_i) = \{\{x_1, \ldots, x_n\}\}$

(b) $\mathrm{prc}_0(\bigwedge_{i=1}^n x_i) = \{\{x_i\}\}_{i\in\{1,\ldots,n\}}$
(c) $\mathrm{prc}_0(x_0 \to \bigvee_{i=1}^n x_i) = \{\{\overline{x_0}, x_1, \ldots, x_n\}\}$
(d) $\mathrm{prc}_0(x_0 \to \bigwedge_{i=1}^n x_i) = \{\{\overline{x_0}, x_i\}\}_{i\in\{1,\ldots,n\}}$
(e) $\mathrm{prc}_0(x_0 \leftarrow \bigvee_{i=1}^n x_i) = \{\{x_0, \overline{x_i}\}\}_{i\in\{1,\ldots,n\}}$
(f) $\mathrm{prc}_0(x_0 \leftarrow \bigwedge_{i=1}^n x_i) = \{\{x_0, \overline{x_1}, \ldots, \overline{x_n}\}\}$
(g) $\mathrm{prc}_0(x_0 \leftrightarrow \bigvee_{i=1}^n x_i) = \mathrm{prc}_0(x_0 \to \bigvee_{i=1}^n x_i) \uplus \mathrm{prc}_0(x_0 \leftarrow \bigvee_{i=1}^n x_i)$
(h) $\mathrm{prc}_0(x_0 \leftrightarrow \bigwedge_{i=1}^n x_i) = \mathrm{prc}_0(x_0 \to \bigwedge_{i=1}^n x_i) \uplus \mathrm{prc}_0(x_0 \leftarrow \bigwedge_{i=1}^n x_i)$

where $\uplus$ is disjoint-union (i.e., $A \uplus B = A \cup B$ but we know that $A \cap B = \emptyset$).

3. For variable-disjoint boolean functions f, g we have:
   (a) If f, g are not constant-0, then $\mathrm{prc}_0(f \wedge g) = \mathrm{prc}_0(f) \uplus \mathrm{prc}_0(g)$.
   (b) If f, g are not constant-1, then $\mathrm{prc}_1(f \vee g) = \mathrm{prc}_1(f) \uplus \mathrm{prc}_1(g)$.
4. A key property of $\mathrm{prc}_0(f)$ is that for all $\varphi \in \mathcal{PASS}$ such that $\varphi * f = 0$ we have $\bot \in \varphi * \mathrm{prc}_0(f)$. This is because by definition there is some $C \in \mathrm{prc}_0(f)$ with $C \subseteq C_\varphi$ ($\mathrm{prc}_0(f)$ is a clausal representation of all minimal falsifying assignments for f).
5. Recalling Definition 2.3.1, we can read off the forced literals from the prime implicates, namely x is forced for f iff $\mathrm{prc}_0(f) \cap \{\bot, \{x\}\} \neq \emptyset$.

As usual, the default-interpretation for a clause-set is the CNF-interpretation, however, since here there might arise some confusion, the definitions are given explicitly:

Definition 2.6.2 *For $F \in \mathcal{CLS}$ we define:*

$$\begin{aligned} \mathbf{prc_0(F)} &:= \mathrm{prc}_0(\mathrm{CNF}(F)) \\ \mathbf{prc_1(F)} &:= \mathrm{prc}_1(\mathrm{CNF}(F)). \end{aligned}$$

Remarks:

1. Important here is that prc_1 computes the prime implicants (which is a DNF clause-set) but for an *input* clause-set we intentionally always take the CNF interpretation. We are simply interested in the prime implicates resp. implicants of the input boolean function, which by default is always given as a CNF.

Well-known is (since [24]) the determination of prime-implicates of clause-sets by resolution:

Lemma 2.6.3 *For $F \in \mathcal{CLS}$ we have*

$$\mathrm{prc}_0(F) = \{C \in \mathcal{CLS} \mid F \vdash C \wedge \forall C' \subset C : F \nvdash C'\}$$

(recall Definition 2.5.2).

Remarks:

1. It is important to note here that resolution as a syntactic operation acts on clause-sets, irrespective of whether we take a CNF or DNF interpretation. This means that also the set $\mathrm{prc}_0(F)$ is defined for a clause-set irrespective of whether we take the CNF or DNF interpretation. In this sense, we are justified in talking of the **prime clauses** of F, i.e., those elements of $\mathrm{prc}_0(F)$.

Prime-implicants of clause-sets are minimal satisfying assignments, and can be described as follows:

Lemma 2.6.4 *For $F \in \mathcal{CLS}$ we have*

$$\mathrm{prc}_1(F) = \{C \in \mathcal{CLS} \mid \forall C' \in F : C \cap C' \neq \emptyset \wedge \quad \forall C' \subset C \, \exists C'' \in F : C' \cap C'' = \emptyset\}.$$

Prime-clauses represent the CNF- resp. DNF-interpretations in the following sense:

Lemma 2.6.5 *For a clause-set F we have*

$$\begin{array}{rcl} \mathrm{CNF}(\mathrm{prc}_0(F)) & \cong & \mathrm{CNF}(F) \\ \mathrm{DNF}(\mathrm{prc}_1(F)) & \cong & \mathrm{DNF}(F) \end{array}$$

Corollary 2.6.6 *For clause-sets $F, G \in \mathcal{CLS}$ we have:*

$$\begin{array}{rcl} \mathrm{CNF}(F) \cong \mathrm{CNF}(G) & \Leftrightarrow & \mathrm{prc}_0(F) = \mathrm{prc}_0(G) \\ \mathrm{DNF}(F) \cong \mathrm{DNF}(G) & \Leftrightarrow & \mathrm{prc}_1(F) = \mathrm{prc}_1(G). \end{array}$$

If a clause-set F now contains only prime clauses (i.e., only those in $\mathrm{prc}_0(F)$) then we called it **primal**:

Definition 2.6.7 *A clause-set $F \in \mathcal{CLS}$ is called **CNF-primal** resp. **DNF-primal** if $F \subseteq \mathrm{prc}_0(\mathrm{CNF}(F))$ resp. $F \subseteq \mathrm{prc}_1(\mathrm{DNF}(F))$.*

Remarks:

1. F is CNF-primal iff there are no clauses $C \subset D$ with $D \in F$ and $\mathrm{CNF}(F) \models \mathrm{CNF}(\{C\})$. And F is DNF-primal iff there are no clauses $C \subset D$ with $D \in F$ and $\mathrm{DNF}(\{C\}) \models \mathrm{DNF}(F)$.
2. $\top$ and $\{\bot\}$ are CNF- and DNF-primal.
3. As discussed in the remarks of 2.6.3, we have the notion of a prime clause, irrespective of the CNF resp. DNF interpretation, and hence we may also speak here of **primal** clause-sets, i.e., those that contain only prime clauses.

For a boolean function f, the smallest clause-set F with $\mathrm{CNF}(F) = f$ will necessarily be primal and **irredundant**.

Definition 2.6.8 *A clause-set $F \in \mathcal{CLS}$ is called **irredundant** if for all clauses $C \in F$ it holds that $\mathrm{CNF}(F \setminus \{C\}) \not\models \mathrm{CNF}(\{C\})$ (or, equivalently, $\mathrm{CNF}(F) \not\cong \mathrm{CNF}(F \setminus \{C\})$).*

That the minimality of F (w.r.t the number of literals) requires irredundancy follows by definition. That primality is necessary follows by the observation that non-primal clauses in F can be replaced with any subsuming primal clauses without breaking equivalence.

Lemma 2.6.9 *Consider a clause-set F. For all $C \in F \setminus \mathrm{prc}_0(F)$ and all $C' \in \mathrm{prc}_0(F)$ such that $C' \subseteq C$ we have that $\mathrm{CNF}(F) \cong \mathrm{CNF}((F \setminus \{C\}) \cup \{C'\})$.*

However, for any given boolean function f there are potentially many irredundant primal clause-sets equivalent to f (see Example 4.5.3 for an illustrative case). To achieve a basic size-lower-bound for *any* such clause-set F with $\mathrm{CNF}(F) = f$ one can consider **essential prime implicates**.

Definition 2.6.10 *Consider a boolean function $f \in \mathcal{BF}$. A prime implicate $C \in \mathrm{prc}_0(f)$ is* ***essential*** *if $\mathrm{prc}_0(F) \setminus \{C\}$ is not equivalent to F.*

Lemma 2.6.11 *Consider $F \in \mathcal{CLS}$, and let $P \subseteq \mathrm{prc}_0(F)$ be the set of essential prime implicates of F. Now for every $F' \in \mathcal{CLS}$ equivalent to F there exists an injection $i : P \to F'$ such that for all $C \in P$ it holds that $C \subseteq i(C)$. Thus $c(F') \geq c(P)$.*

Proof: For every $C' \in F'$ there exists a $C \in \mathrm{prc}_0(F)$ ($= \mathrm{prc}_0(F')$ by Corollary 2.6.6) such that $C \subseteq C'$; replacing every $C' \in F$ by such a chosen C we obtain $F'' \subseteq \mathrm{prc}_0(F)$ with $P \subseteq F''$. This is possible because C follows from F', and so $F' \cup \{C\}$ is equivalent to F', but then C' follows from C (hence from $F \cup \{C\}$) and so $(F' \cup \{C\}) \setminus \{C'\}$ is equivalent to F'. □

2.7 Special classes of clause and clause-sets

The most fundamental of classes of clause-set for polynomial time satisfiability are now introduced, which it will be shown (for our purposes) are subsumed at different levels of the $\mathcal{UC}_k$ hierarchy (see Lemma 4.3.2). The use of these classes and their generalisations, both as theoretical frameworks (showing tractability, fixed-parameter tractability etc) as well as in practical settings (for example the use of Horn clauses in logic programming, as discussed in [101]) also helps illustrate the utility of the hierarchies defined in this thesis.

2.7.1 2–$\mathcal{CLS}$

An elementary hierarchy of clause-set classes is that of the classes of clause-set with clauses of size at most k:

Definition 2.7.1 *The class of clause-sets with clauses of size at most k is denoted by:*

$$k\text{–}\mathcal{CLS} := \{F \in \mathcal{CLS} \mid \forall C \in F : |C| \leq k\}$$

In general, it is (now) well-known that deciding satisfiability for k–$\mathcal{CLS}$ for $k \geq 3$ is NP-complete (proven in [38], Cook's original paper on the NP-completeness of SAT). In 1967 in [103], Krom showed that for 2-CNFs (where all disjunctions are of size ≤ 2) computing the resolution closure and checking for the empty-clause (empty-disjunction) yields a quadratic satisfiability algorithm. Later in [3] a linear time algorithm was introduced based on detecting strongly connected components of the graph defined by the 2–$\mathcal{CLS}$ clause-set.

Lemma 2.7.2 *For all $F \in 2\text{–}\mathcal{CLS}$ the question "$F \in \mathcal{SAT}$?" is decidable in time $O(n)$.*

This is important as a tractability result for satisfiability and many problems can be reduced to 2–$\mathcal{CLS}$ for certain fixed parameters. For example in minimizing channel density in VLSI design in [29], polytime sports scheduling ("home-away assignment") in [124], inferring haplotypes for sets of individuals in bioformatics in [57], as well as showing the tractability of restricted constraint languages in [128]; for an overview see Chapter 5 in [42]. Furthermore, unlike 3–$\mathcal{CLS}$ and above, one can find the smallest equivalent 2–$\mathcal{CLS}$ formula in polynomial time (see Theorem 4.5.4 in Chapter 4).

2.7.2 Horn clause-sets and generalisations

The concept of a Horn clause, and Horn formulas, were introduced in [91]. Horn clauses intuitively represent implications of the form $v_1 \wedge \ldots \wedge v_k \to v$ (for variables $v_1, \ldots, v_k, v \in \mathcal{VA}$) making Horn clause-sets prime candidates for modelling functional dependencies in database theory and logic programming (for example the use of Horn-logic as the underlying predicate language in [101];see below for further details).

Definition 2.7.3 *A clause $C \in \mathcal{CL}$ is a **Horn clause** if $|C \cap \mathcal{VA}| \leq 1$ and is a **pure Horn clause** if $|C \cap \mathcal{VA}| = 1$. A **Horn clause-set** resp. **pure Horn clause-set** is a clause-set with only Horn resp. pure Horn clauses. Let $\mathcal{HO} \subset \mathcal{CLS}$ be the set of all **Horn clause-sets**, that is, $\mathcal{HO} := \{F \in \mathcal{CLS} \mid \forall C \in F : |C \cap \mathcal{VA}| \leq 1\}$, while $\mathcal{HO}^+ := \{F \in \mathcal{CLS} \mid \forall C \in F : |C \cap \mathcal{VA}| = 1\}$ is the set of all **pure Horn clause-sets**.*

Some simple properties of Horn clause-sets are:

1. $\mathcal{HO}$ and $\mathcal{HO}^+$ are stable under union (being a Horn clause or pure Horn clause is a local syntactic property).
2. Every $F \in \mathcal{HO}^+$ is satisfiable via $\langle v \to 1 : v \in \mathrm{var}(F)\rangle * F = \top$.
3. Every $F \in \mathcal{HO}$ not containing the empty clause or positive unit-clauses is satisfiable via $\langle v \to 0 : v \in \mathrm{var}(F)\rangle * F = \top$.
4. $\mathcal{HO}$ is stable under elimination of literal occurrences and elimination of clauses (and thus also under application of partial assignments).
5. $\mathcal{HO}^+$ is stable under elimination of clauses, but not under elimination of literal occurrences and not under application of partial assignments. $\mathcal{HO}^+$ is stable under elimination of negative literal occurrences and under application of partial assignments which set no variable to 0.
6. $\mathcal{HO}$ as well as $\mathcal{HO}^+$ is stable under addition of resolvents, and thus we have:
   (a) For $F \in \mathcal{HO}$ holds $\mathrm{prc}_0(F) \in \mathcal{HO}$.
   (b) For $F \in \mathcal{HO}^+$ holds $\mathrm{prc}_0(F) \in \mathcal{HO}^+$.
7. Checking whether a clause-set F is a Horn clause-set (or pure Horn) is possible in linear time ($O(l(F))$ - simply check each clause is Horn resp. pure Horn).

As first shown in [53], there is a linear time algorithm for the determining satisfiability of the special case of Horn formulas. In fact, not only is satisfiability of Horn clause-sets solvable in linear time, but it is actually solvable using unit-clause propagation (this was originally shown in [86] but is repeated here for pedagogical reasons) in linear time (due to [53] as mentioned in Section 2.4).

Lemma 2.7.4 *Consider a Horn clause-set $F \in \mathcal{HO}$. We have that*

$$\mathrm{r}_1(F) = \{\bot\} \iff F \in \mathcal{USAT}.$$

Proof: That $\mathrm{r}_1(F) = \{\bot\} \implies F \in \mathcal{USAT}$ follows from the fact that r_1 only sets forced assignments. To show that $F \in \mathcal{USAT} \implies \mathrm{r}_1(F) = \{\bot\}$ we consider the contraposition $\mathrm{r}_1(F) \neq \{\bot\} \implies F \in \mathcal{SAT}$. Consider an F such that $F' := \mathrm{r}_1(F) \neq \{\bot\}$. If $F' = \top$ then

obviously $F \in \mathcal{SAT}$. Otherwise F' is a Horn clause-set (recall Horn clause-sets are closed under application of partial assignments) and by the definition of r_1 we know that all clauses in F' are of size ≥ 2. Therefore every clause in F contains at least one negated variable and so the assignment $\langle v \to 0 : v \in \mathrm{var}(F') \rangle$ satisfies F'. □

In general, not every boolean function has an equivalent representation in $\mathcal{HO}$ (for example, those given by monotone CNFs – CNFs for which every clause contains only positive literals), however despite this the class is important as both the foundation of logic programming (as discussed in [101] [6]), as approximation classes for knowledge representation (see for example using Horn upper and lower bounds as a method for producing approximate knowledge bases in [142]), for modelling of functional dependencies in database systems in [58], and more (a good overview of uses of Horn clause-sets and Horn functions can be found in Chapter 6 of [42]).

2.7.2.1 Generalisations of Horn clause-sets

Due to the fact that $\mathcal{HO}$ is insufficient to model all boolean functions and numerous simple classes of clause-set are not included, an obvious question becomes how to generalise Horn clause-sets to larger classes which allow more to be represented while maintaining important properties. A first attempt at generalising Horn clause-sets comes by observing that the "essential" syntactical structure is preserved by renamings.

Definition 2.7.5 *By $\mathcal{RHO} \subset \mathcal{CLS}$ we denote the class of* ***renamable Horn clause-sets****, that is, the class of clause-sets which are isomorphic to some Horn clause-sets.*

Remarks:

1. Checking whether a clause-set F is in $\mathcal{RHO}$ can be done in linear time as shown in [56].
2. $\mathcal{RHO}$ is now closed only under *disjoint* union, not union of arbitrary members (observe that for any clause C the clause-set $\{C\}$ is renamable Horn but there are clause-sets which are not in $\mathcal{RHO}$).

While $\mathcal{RHO}$ clearly includes more clause-sets than $\mathcal{HO}$ (for example, at least monotone CNFs are now included), it is still not complete with respect to representing boolean functions (e.g., parity functions, which have unique minimal CNF representations via their sets of prime implicates, have no equivalent clause-set representation in $\mathcal{RHO}$ – despite satisfiability for these representations being possible via just checking for $\bot$). To address the inability of $\mathcal{HO}$ and $\mathcal{RHO}$ to represent certain families of boolean functions, there have been various generalisations of $\mathcal{HO}$ and $\mathcal{RHO}$ to hierarchies of poly-time SAT classes. One of the first such classes which maintains many of the properties of the Horn class is that of generalised Horn clause-sets introduced in [162, 66] and later recharacterised in [99].

[6] Horn logic with respect to logic programming is actually concerned with Horn clauses in predicate logic, however the underlying structure is still the same and the fundamentals properties that ensure the poly-time solvability of propositional Horn clause-sets is the same as that that underlies the structure of predicate Horn formulae.

Definition 2.7.6 *Let $\mathcal{HO}_1 := \mathcal{HO}$, while for $k > 1$ we define $\mathcal{HO}_k \subset \mathcal{CLS}$ as the set of all $F \in \mathcal{CLS}$ such that there exists an ordering $F = \{C_1, \ldots, C_m\}$, $m := c(F)$, of clauses, and there exist sets of positive literals $V_i \subset \mathcal{VA}$ for $i \in \{1, \ldots, m\}$ such that we have*

$$V_1 \subseteq \ldots \subseteq V_m$$
$$V_1 \subseteq C_1, \ldots, V_m \subseteq C_m$$
$$\{C_i \setminus V_i : i \in \{1, \ldots, m\}\} \in \mathcal{HO}_{k-1}.$$

Remarks:

1. By choosing all $V_i = \emptyset$ we see that we have $\mathcal{HO}_k \subseteq \mathcal{HO}_{k'}$ for $1 \leq k \leq k'$.
2. For $F \in \mathcal{HO}_k$, $k \in \mathbb{N}$, $k \geq 2$, and every finite $V \subseteq \mathcal{VA}$ we have $\{C \cup V : C \in F\} \in \mathcal{HO}_k$ as well.

We collect some tools for recognition of generalised Horn clause-sets in the following lemma (all follow by definition or via a simple inductive proof):

Lemma 2.7.7 *For $k \in \mathbb{N}$ the classes $\mathcal{HO}_k$ are stable under the following operations:*

1. *removal of clauses;*
2. *crossing out of variables (the operation $V * F$ for variable-sets V);*
3. *application of partial assignments (note that this can be achieved by crossing out of variables and removal of clauses);*
4. *addition and removal of negative literal occurrences;*
5. *addition of a clause C for which there is $D \in F$ with $D \subseteq C$ such that $C \setminus D$ contains only negative literals.*

Note that these operations can lead to contraction of clauses. The class $\mathcal{HO}_1$ is also stable under removal of arbitrary literal occurrences and under DP-reduction.

Although not immediately obvious, we see that in the limit the $\mathcal{HO}_k$ hierarchy is capable of representing all boolean functions:

Lemma 2.7.8 *For a full unsatisfiable clause-set F with $n := n(F) > 0$ we have:*

1. *$F \in \mathcal{HO}_n$.*
2. *If $n(F) > 1$, then $F \notin \mathcal{HO}_{n-1}$.*

Proof: To see that $F \in \mathcal{HO}_n$, we proceed by induction on n. For $n = 0$ and $n = 1$ that $F \in \mathcal{HO}_1 = \mathcal{HO}$ follows from the definition. For $n > 1$ choose a variable $v \in \mathrm{var}(F)$ and order the clauses $C_1, \ldots, C_{2^n}$ of F such that those clauses containing $\overline{v}$ come first, followed by those with v (note this covers all clauses). Now set $V_1 = \cdots = V_{2^{n-1}} = \emptyset$ and $V_{2^{n-1}+1} = \cdots = V_{2^n} = \{v\}$. We have that $F'' := \{C_i \setminus V_i : i \in \{1, \ldots, m\}\} = F' \cup \{C \cup \{\overline{v}\} : C \in F'\}$ where F' is the full unsatisfiable clause-set on $\mathrm{var}(F) \setminus \{v\}$. By induction $F' \in \mathcal{HO}_{n-1}$ and so by part 5 of Lemma 2.7.7 we have that $F'' \in \mathcal{HO}_{n-1}$, hence $F \in \mathcal{HO}_n$.

To see that $F \notin \mathcal{HO}_{n-1}$, we again proceed by induction on n. For $n = 2$ observe by definition that $F \notin \mathcal{HO}$ due to the binary clause with all positive literals. For $n > 2$ assume for the sake of contradiction that $F \in \mathcal{HO}_{n-1}$ and hence that there exist $V_1, \ldots, V_{2^n}$ and an

ordering $C_1, \ldots, C_{2^n}$ of the clause of F such that $V_1 \subseteq \ldots \subseteq V_{2^n}$, $V_1 \subseteq C_1, \ldots, V_{2^n} \subseteq C_{2^n}$ and $F' := \{C_i \setminus V_i : i \in \{1, \ldots, m\}\} \in \mathcal{HO}_{n-2}$. If $V_1 = \cdots = V_{2^n} = \emptyset$ then $F' = F$ and for any variable $v \in \mathrm{var}(F)$ we have by induction that $\langle v \to 0 \rangle * F' \notin \mathcal{HO}_{n-2}$ and so by part 3 of Lemma 2.7.7 $F \notin \mathcal{HO}_{n-1}$. Otherwise, consider the smallest $i \in \{1, \ldots, 2^n\}$ such that $V_i \neq \emptyset$ and some witness variable $v \in V_i$. For all $j > i$ we have that $V_i \subseteq C_j$ and so all clauses containing $\overline{v}$ occur in $C_1, \ldots C_{i-1}$, i.e., $\{C \cup \{\overline{v}\} : C \in F''\} \subseteq \{C_1, \ldots, C_{i-1}\} \subseteq F'$, where F'' is the full clause-set on $\mathrm{var}(F) \setminus \{v\}$. Therefore $\langle v \to 0 \rangle * F' = F''$ which by induction is not in $\mathcal{HO}_{n-2}$ and so by part 3 of Lemma 2.7.7 neither is F', a contradiction. □

By Part 1 of Lemma 2.7.8 and Part 1 of Lemma 2.7.7:

Corollary 2.7.9 *For all boolean functions f we have that* $\mathrm{CNF}(f) \in \mathcal{HO}_{n(f)}$.

There have been numerous other examples of classes and hierarchies generalising Horn clause-sets, including $\mathcal{SLUR}$ (introduced in [141] to capture classes of clause-set solvable via UCP (r_1); discussed in Section 4.1 of Chapter 4), $\mathcal{SLUR}(k)$ and $\mathcal{SLUR}*(k)$ (based on $\mathcal{SLUR}$ and introduced in [36, 10]), $(\Pi_k)_{k \in \mathbb{N}_0}$ and $(\Upsilon_k)_{k \in \mathbb{N}_0}$ (based on nested structures of renamable Horn clause-sets and introduced in [35], CANON(k) (based on bounded depth resolution, and introduced in [36, 10]), as well as the $G_k(\mathcal{U}, \mathcal{S})$ hierarchies from [104, 110] (on which for *unsatisfiable* clause-sets the hierarchy $\mathcal{UC}_k$ in this thesis is based). These hierarchies are discussed in more detail in Chapter 4, where they are compared to the hierarchies introduced in this thesis.

2.7.3 Hitting clause-sets

Another simple class of clause-sets with poly-time satisfiability testing is that of hitting clause-sets, sometimes called "orthogonal" or "disjoint" when referring to the DNF interpretation.

Definition 2.7.10 *A clause-set F is **hitting** if every two different clauses $C, C' \in F$ clash in at least one literal, i.e., $C \cap \overline{C'} \neq \emptyset$. The set of hitting clause-sets is denoted by $\mathcal{HIT} \subset \mathcal{CLS}$.*

Remarks:

1. In [42] we see the use of the term orthogonal rather than hitting. That is, an orthogonal DNF is a hitting clause-set under the DNF representation. While for example in [154] we see the term disjoint used, referring to the fact that every clause in a hitting clause-set represents a disjoint sets of assignments (i.e., $\mathrm{CNF}(\{C\})^{-1}(0) \cap \mathrm{CNF}(\{C'\})^{-1}(0) = \emptyset$ for all different $C, C' \in F \in \mathcal{HIT}$).

Unlike for 2-$\mathcal{CLS}$ and Horn clause-sets the poly-time satisfiability of hitting clause-sets is not related to the resolution calculus but purely to a counting property given by the structure of the clause-set. If we take a CNF interpretation of a hitting clause-set, then each clause C represents a *disjoint* set of total falsifying assignments, and so we can determine unsatisfiability by counting up all falsifying total assignments covered by each clause to see if all total assignments are covered.

Lemma 2.7.11 *For all clause-sets $F \in \mathcal{HIT}$ we have that*

$$|\mathrm{CNF}(F)^{-1}(0)| = |\mathrm{DNF}(F)^{-1}(1)| = \sum_{C \in F} 2^{n(F)-|C|}$$

and that $F \in \mathcal{USAT} \iff |\mathrm{CNF}(F)^{-1}(0)| = 2^{n(F)}$. Hence the satisfiability of $F \in \mathcal{HIT}$ (as a CNF clause-set) can be determined in time $O(l(F))$.

An overview of hitting clause-sets (and orthogonal DNFs) is given in Chapter 7 of [42] and the combinatorics of clashing in clause-sets is given in [109]. While the focus of this thesis is on hierarchies based on complexity measures for resolution (and hence the classes considered are orthogonal to hitting clause-sets – all full clause-sets are hitting while the class of full-clause-sets doesn't fit into any of the classes introduced here), we will see in Chapter 6 that the properties of hitting clause-sets can be useful for translation into the classes introduced in this thesis, and so the definition and intuition are useful.

2.8 Constraint Programming and satisfiability

The primary focus of this thesis is on the power (and limits) of clausal representations of boolean functions and the combinatorial properties of these representations. However, in the outlook it is envisaged that the classes defined will be conceptually useful in the understanding of translations of constraints to SAT. That is, modelling some decision problem as a set of high level constraints and then encoding and translating these constraints to CNF. In this way, then solving with a SAT solver (as done for example in the CSP solver Sugar in [155]), rather than solving the constraints directly using specialised constraint solvers.

When translating a network of constraints to satisfiability, it is important to understand the basic notions from Constraint Programming. An overview of constraint programming can be found in [139]; the notions directly relevant to this thesis are now summarised, formulated with notations closely matching those for clauses and clause-sets.

Definition 2.8.1 *Consider a fixed universe $\mathcal{VA}$ of variables with $\mathbb{N} \subseteq \mathcal{VA}$, together with a fixed set (the "universal domain") $\mathcal{DOM}$ of "values", where we assume $0, 1 \in \mathcal{DOM}$ (and thus $|\mathcal{DOM}| \geq 2$). A **constraint satisfaction problem (CSP) P** is a triple $P := (V, D, F)$ such that*

1. *$V \subseteq \mathcal{VA}$ is the variable set;*
2. *$D : V \to \mathbb{P}(\mathcal{DOM})$ specifies the domain of each variable;*
3. *$F \subseteq \{ (V', R) : V' \subseteq V, R \subseteq \prod_{v \in V'} D(v) \}$ is the set of **constraints** (i.e., a subset of the variables and a relation over their domains). For a constraint C we denote the variable set by V_C and the relation by R_C.*

*A **partial assignment** $\varphi : V' \to \mathcal{DOM}$ to a CSP assigns values to variables; φ is **total** for a CSP if $V' = V$. A partial assignment φ is consistent with a constraint C if there is a partial assignment $\varphi' : V_C \to \mathcal{DOM}$ extending φ (i.e., $\varphi' \supseteq \varphi$) such that $\varphi' \in R_C$ holds. A partial assignment φ satisfies a constraint C if $\varphi_{|V_C}$ (i.e., φ restricted to V_C) is consistent with C. A partial assignment φ satisfies resp. "is consistent" with a CSP if φ satisfies resp. "is consistent" with every constraint in P. A CSP is then **satisfiable** if there is some partial assignment which satisfies it. The application of a partial assignment to a constraint and/or CSP then follows by removing variables and restricting constraint relations in the natural way.*

To solve an instance of a CSP then means to find a satisfying assignment or show none exists. Methods for solving constraint problems are similar and very much related to SAT (e.g., various forms of intelligent backtracking are also used; see Chapter 4 in [139]), and similar techniques are often used (e.g. "nogood" learning, related to clause-learning, see Section 4.5 in Chapter 4 of [139]; and watched literals in [69]) in modern CSP solvers such as [68]. The advantage of CSP solvers (in general) is that they benefit from higher level specialised representations

of constraints, which can be conceptually easier for the user and can allow the solver to use specialised propagation algorithms for each constraint (unlike SAT where typically the CNF clause-set is considered as a "whole"). SAT, on the other hand, benefits from the simple and yet rich combinatorial structure of the CNF representation, allowing for work to be devoted to very efficient algorithms and data-structures for the general problem.

To represent a CSP as a clause-set, (essentially) the task is then to find a clause-set representing each individual constraint and take the union to form a representation of the whole CSP. In general, representing a constraint as a clause-set has two conceptual stages, which we call here **encoding** and **translation**. The **encoding** step takes the constraint C and *encodes* the non-boolean variables and their domains into boolean variables, resulting in a boolean function f_C. The **translation** stage then translates this boolean function f_C to a CNF clause-set F_C. This is not just a conceptual separation, but a real one; the boolean function f_C exists for any CNF clause-set F representing some constraint C by just taking CNF(F) and this boolean function and its relation to the CNF representation F have their own properties and can be studied in their own right.

Example 2.8.2 *Consider the constraint* $C := (\{v_1, v_2\}, R)$ *with underlying domains* $D(v_1) = \{1,2,3\}$ *and* $D(v_2) = \{4,5\}$, *and* $R := \{(1,4),(2,5),(3,4),(1,5)\}$. *The* ***direct encoding****, discussed in [160], of* C *into SAT encodes each possible assignment* $v = d$ *(for variable* v *and domain value* d*) as a single variable* $v_{v=d}$ *which is true if this assignment is made, and false otherwise. For* C *the direct encoding yields the boolean function* $f_C : \{0,1\}^V \to \{0,1\}$ *with* $V = \{v_{v_1=1}, v_{v_1=2}, v_{v_1=3}, v_{v_2=4}, v_{v_2=5}\}$ *with*

$$f_C(\varphi) = 1 \iff \forall i \in D(v_1), j \in D(v_2) : (\varphi(v_{v_1=i}) = 1 \wedge \varphi(v_{v_2=j}) = 1) \leftrightarrow (i,j) \in R.$$

f_C *then has the following natural clause-set representation* F_C*:*

$$F_C := \underbrace{\{\overline{v_{v_1=1}}, \overline{v_{v_1=2}}\}, \{\overline{v_{v_1=1}}, \overline{v_{v_1=3}}\}, \{\overline{v_{v_1=2}}, \overline{v_{v_1=3}}\}}_{v_1 \text{ is assigned at most one value}}, \underbrace{\{\overline{v_{v_2=4}}, \overline{v_{v_2=5}}\}}_{v_2 \text{ is assigned at most one value}},$$
$$\underbrace{\{v_{v_1=1}, v_{v_1=2}, v_{v_1=3}\}}_{v_1 \text{ assigned at least one value}}, \underbrace{\{v_{v_2=4}, v_{v_2=5}\}}_{v_2 \text{ assigned at least one value}} \underbrace{\{\overline{v_{v_1=2}}, \overline{v_{v_2=4}}\}, \{\overline{v_{v_1=3}}, \overline{v_{v_2=5}}\}}_{(2,4) \notin R \text{ and } (3,5) \notin R}$$

The ***support encoding****, introduced in [67], of* C *uses the same underlying encoding* f_C *of* C*, i.e., each variable of* f_C *represents that* $v = d$ *for some variable* v *and value* $d \in D(v)$. *However, to improve propagation, it introduces additional clauses. That is, it uses a different* translation *of* f_C *(see Example 2.8.5).*

One of the central notions in CP is that of maintaining consistency of constraints and/or the network. The aim of "maintaining (some form of) consistency"is to ensure that under some (restricted type of) partial assignment if certain assignments $v = d$ are inconsistent with the CSP then this is directly forced in the network, e.g., by removing d from the domain of v. There are many forms of consistency considered, including arc-consistency, path consistency, as well as generalisations such as k-consistency and (i,j)-consistency (for a full overview see Chapter 3 in [139]). Path, k-, and (i,j)- consistency notions focus on consistency across multiple constraints, while this thesis focuses on the translations of individual boolean functions, ignoring the possibility of decomposition. Therefore, most relevant to this thesis is (relational) **arc-consistency** as defined in [50].

Definition 2.8.3 *A constraint $C := (V', R)$, in some constraint satisfaction problem $P := (V, D, F)$, is (generalised) arc-consistent if for every variable $v \in \mathrm{var}(C)$ and every value $d \in D(v)$ the partial assignment $\langle v \to d \rangle$ is consistent with C. A constraint problem P is arc-consistent if every constraint is arc-consistent.*

Path consistency, k-consistency and (i, j)-consistency generalise arc-consistency in various ways by considering assignments of size greater than 2 across multiple constraints. When solving a CSP problem by translating to SAT (i.e., to a CNF clause-set), the typical aim in the literature has been to try to ensure these notions of consistency are "maintained" by the SAT solver. There has been considerable work on maintaining arc-consistency in SAT via unit-clause-propagation (see Definition 2.8.4), for example translating cardinality constraints while maintaining arc-consistency in [135, 7, 147]; for restricted classes of pseudo-boolean constraints in [8]; for arbitrary crisp constraints (i.e., finite constraints where the relation is given explicitly) in [67, 6]; for smooth DNNFs (restricted forms of boolean formulae with a "decomposability property" allowing efficient queries in knowledge compilation; see [46]) in [98].

Definition 2.8.4 *Unit-clause propagation (r_1) maintains arc-consistency for a constraint C on a clause-set representation F if F represents C (i.e., satisfying assignments for C correspond to satisfying assignments for F and vice versa) and for all partial assignments ψ* to the variables *of C then if for some variable v and value $d \in D(v)$ we have that $v \neq d$ is forced by C under ψ then r_1 forces the corresponding variables in F.*

What "the corresponding variables in F" are will be highly dependent on the encoding of the non-boolean constraint variables into the boolean variables of the SAT instance. A concrete example is given in Example 2.8.5.

Example 2.8.5 *Consider the constraint C from Example 2.8.2. The support encoding, introduced in [67], encodes C in the same way as the direct encoding (i.e., the encoded boolean function is f_C from Example 2.8.2) but now translates f_C to a clause-set by encoding the support for each $v = d$ assignment, rather than encoding the "nogoods" (i.e., $(2, 4) \notin R$). The* ***support*** *of a value assignment $v = d$ is the set of possible total assignments to the other variables in the constraint which are consistent with the assignment $v = d$. The final clause-set translation of C using the support encoding is given by encoding for each assignment $v = d$ that at least one of its supports must also be assigned to satisfy the constraint. The support encoding F'_C for C is given by the following clauses where $AMO(v_i)$ and $ALO(v_i)$ indicate that the corresponding clauses encode that v_i takes at-most resp. at-least one value in its domain:*

$$
\begin{aligned}
F'_C := \{ & \underbrace{\{\overline{v_{v_1=1}}, \overline{v_{v_1=2}}\}, \{\overline{v_{v_1=1}}, \overline{v_{v_1=3}}\}, \{\overline{v_{v_1=2}}, \overline{v_{v_1=3}}\}}_{AMO(v_1)}, \underbrace{\{\overline{v_{v_2=4}}, \overline{v_{v_2=5}}\}}_{AMO(v_2)}, \underbrace{\{v_{v_1=1}, v_{v_1=2}, v_{v_1=3}\}}_{ALO(v_1)}, \underbrace{\{v_{v_2=4}, v_{v_2=5}\}}_{ALO(v_2)} \\
& \underbrace{\{\overline{v_{v_1=1}}, v_{v_2=4}, v_{v_2=5}\}}_{(v_1=1) \to ((v_2=4) \vee (v_2=5))}, \underbrace{\{\overline{v_{v_1=2}}, v_{v_2=5}\}}_{(v_1=2) \to (v_2=5)}, \underbrace{\{\overline{v_{v_1=3}}, v_{v_2=4}\}}_{(v_1=3) \to (v_2=4)}, \\
& \underbrace{\{\overline{v_{v_2=4}}, v_{v_1=1}, v_{v_1=3}\}}_{(v_2=4) \to ((v_1=1) \vee (v_1=3))}, \underbrace{\{\overline{v_{v_2=5}}, v_{v_1=1}, v_{v_1=2}\}}_{(v_2=5) \to ((v_1=1) \vee (v_1=2))} \}
\end{aligned}
$$

In [67], it is shown that the support encoding "maintains arc-consistency via unit-clause propagation". In this particular example, this means that

1. *for all partial assignments $\psi : V \to \{1, 2, 3, 4, 5\}$ (for $V \subseteq \{v_1, v_2\}$) to C and*
2. *for all $i \in \{1, 2\}$ and $d \in D(v_i)$*

if $v_{v_i=d} \in \mathrm{var}(\mathrm{r}_1(\langle v_{v_i=\psi(v)} : v \in V\rangle * F'_C))$ *then* $v_i = d$ *is consistent with* C *under partial assignment* ψ. *In other words, any assignment* $v = d$ *which is inconsistent at the constraint level results in the forced literal* $\overline{v_{v=d}}$ *in* f_C *which is then found and propagated by* r_1. *That unit-clause propagation propagates* $\overline{v_{v_i=d}}$ *follows from the fact that if* $v_i = d$ *is not consistent with* C *under* ψ *then it has no support and so the support clause for* $v_i = d$ *yields a unit-clause. So for example, if we consider the partial assignment* $\psi : \{v_2\} \to \{1,2,3,4,5\}$ *with* $\psi(v_2) = 4$ *then neither* $v_2 = 5$ *or* $v_1 = 2$ *are consistent with* $\psi * C$ *and hence unit-clause propagation forces* $v_{v_2=5}$ *and* $v_{v_1=2}$ *to false:*

1. $\mathrm{r}_1(\langle v_{v_2=4} \to 1\rangle * F'_C) = \mathrm{r}_1(\langle v_{v_2=4} \to 1, v_{v_2=5} \to 0\rangle * F'_C)$ *(due to* $\{\overline{v_{v_2=4}}, \overline{v_{v_2=5}}\} \mapsto \{\overline{v_{v_2=5}}\}$*);*
2. $\mathrm{r}_1(\langle v_{v_2=4} \to 1, v_{v_2=5} \to 0\rangle * F'_C) = \mathrm{r}_1(\langle v_{v_2=4} \to 1, v_{v_2=5} \to 0, v_{v_1=2} \to 0\rangle * F'_C)$ *(due to* $\{\overline{v_{v_1=2}}, v_{v_2=5}\} \mapsto \{\overline{v_{v_1=2}}\}$*);*
3. *Finally* $\mathrm{r}_1(\langle v_{v_2=4} \to 1, v_{v_2=5} \to 0, v_{v_1=2} \to 0\rangle * F'_C) = \{\{\overline{v_{v_1=1}}, \overline{v_{v_1=3}}\}, \{v_{v_1=1}, v_{v_1=3}\}\}$.

In the case of both the direct and support encodings, the translations of the encoded boolean functions do not use introduce new variables. However, other constraint translations, for example translations of cardinality constraints in [7] do introduce new variables as a means of producing small translations.

In this thesis, the focus is purely on representations of boolean functions, and so the results apply purely to the *translation* aspect of constraint translation. That is, on the translation of a boolean function to a CNF. The important fact here is that maintaining arc-consistency via UCP on a clause-set F_C representing a constraint C requires for all partial assignments to the constraint variables, translated to partial assignments on the variables the encoded boolean function f_C, that the relevant propagations are made on *these* variables. However, if F_C uses auxiliary variables to represent f_C, there is no requirement that arbitrary partial assignments setting auxiliary variables must have efficient propagation properties. This is the heart of the difference between *relative* and *absolute* notions of hardness as discussed in Section 6.1 of Chapter 6. Restricting attention to translations of boolean functions (i.e., assuming the encoding step has already been done), UCP maintaining arc-consistency w.r.t a boolean function f on a clause-set F is equivalent to F having p-hardness 1 relative to f as discussed in Section 6.1 of Chapter 6.

Chapter 3

The hardness of clause-sets

This chapter is devoted to the discussion of $\mathrm{hd} : \mathcal{CLS} \to \mathbb{N}_0$. It is the central concept of the thesis, from which the hierarchy $\mathcal{UC}_k$ is derived (Definition 4.2.1). The basic idea is to start with some measurement $h : \mathcal{USAT} \to \mathbb{N}_0$ of "the complexity" of unsatisfiable clause-sets F. This measure is extended to arbitrary $F \in \mathcal{CLS}$ by maximising over all "sub-instances" of F, that is, over all unsatisfiable $\varphi * F$ for (arbitrary) partial assignments φ. A first guess for $h : \mathcal{USAT} \to \mathbb{N}_0$ is to take something like the logarithm of the tree-resolution complexity of F. However this measure is too fine-grained (how do we separate levels of the hierarchy? 0.01? 0.1? 1?) and doesn't yield a hierarchy like $\mathcal{UC}_k$ (introduced in Chapter 4), where each level brings a qualitative enhancement. Another approach is algorithmical, measuring how far F is from being refutable by unit-clause propagation. As shown in [104, 110], actually these two lines of thought can be brought together by the hardness measure $\mathrm{hd} : \mathcal{USAT} \to \mathbb{N}_0$. Why only tree-resolution, and not dag-resolution (i.e., full resolution)? The tree-resolution approach is the natural starting point, and what is easy for tree-resolution is also easy for dag-resolution. The basic approach here towards the more complicated handling of dag-resolution is given by whd (from Section 3.6.2), made into the $\mathcal{WC}_k$ hierarchy in Section 4.6.

The outline of this section is as follows. $\mathrm{hd}(F)$ is defined and discussed for unsatisfiable F in Subsection 3.3. The general case (arbitrary F) is handled in Subsection 3.4 by reduction to the unsatisfiable cases within F (as produced by applying partial assignments). The central result of this section can be seen in Theorem 4.2.2, which shows that a clause-set F having $\mathrm{hd}(F) \leq k$ is equivalent to the condition that all prime implicates of F can be derived by some resolution tree with a Horton-Strahler number at most k. In this way some form of geometric intuition is gained, and a machinery becomes available. The first applications are given by the various lemmas in Section 4.3 for determining hardness under various circumstances.

When considering only unsatisfiable clause-sets F, in [104, 110] actually a general concept of "hardness" was introduced, parameterised by an oracle $\mathcal{U} \subseteq \mathcal{USAT}$ for ("easy") detection of special cases of unsatisfiability. In this thesis only $\mathcal{U} = \{F \in \mathcal{CLS} : \bot \in F\}$ is used, but a general theory using oracles is expected to become important in the future (see discussion in the outlook in Section 8.2 of Chapter 8).

Most lemmas in Sections 3.1 to 3.3.1 recap results from [104, 110], excluding Lemma 3.3.5 and Corollary 3.3.6 which introduce new conceptual insight. From Section 3.4 onwards all results are new to this thesis (although published in [76, 78]) and constitute original research contributions.

3.1 Generalised unit-clause-propagation

In this section we review the approximations of forced assignments as computed by the hierarchy of reductions $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ from [104, 110] for $k \in \mathbb{N}_0$. First we introduce the semantical notion of forced literals/assignments in Subsection 2.3 together with the limit-reduction $\mathrm{r}_\infty : \mathcal{CLS} \to \mathcal{CLS}$, which eliminates *all* forced assignments. In Subsection 3.1.1 then the r_k-reductions themselves (eliminating some forced assignments) are defined and basic properties discussed. In Subsection 3.2 finally we introduce generalised (nested) input resolution and its main parameter, the "Horton-Strahler number" of the corresponding resolution tree, generalising the well-known refutational equivalence between unit resolution and input resolution, and providing the proof-theoretic background.

For further discussions of these reductions, in the context of SAT decision and in their relations to various consistency and width-related notions, see [104, 110] and Section 3 in [112]. It seems here that the r_k-reductions establish the SAT-counterpart to consistency-notions from the constraint literature (see [16] for an overview). We have the following basic distinction between SAT and CSP: SAT has the extremely "thin" clauses, enabling the global point of view ("no (or flat) hierarchies"), while CSP has "fat" constraints, the "lumping together" of clauses. In the SAT world, the r_k-reductions approximate global consistency via approaching all assignments of r_∞, while in the CSP world, consistency means making the constraints stronger and stronger (lumping more and more clauses together), until only one constraint is left. Thus the (stronger) consistency-notions of CSP are more related to width-restricted resolution, while, as shown in [104, 110], the r_k-reductions are much weaker (each only using linear space). Making a clause-set F "consistent" in the SAT world thus means here to find a "representation" F' of F (see Section 6.1 in Chapter 5 for some discussion on "representations"), where via r_k for some $k \in \mathbb{N}_0$ we can derive "everything", which is embodied in its most elementary form in the $\mathcal{UC}_k$-hierarchy, that is, via the condition $F' \in \mathcal{UC}_k$ (Definition 4.2.1).

3.1.1 A hierarchy of reductions

We now review the hierarchy $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$, $k \in \mathbb{N}_0$, of reductions ([104]), which achieves approximating r_∞ by poly-time computable functions. The basic idea is that unit-clause propagation in a sense computes the most direct forced assignments (at "level $k = 1$"), and generalisations like failed-literal elimination (level $k = 2$) find more forced assignments.

Definition 3.1.1 ([104]) *The maps $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ for $k \in \mathbb{N}_0$ are defined as follows (for $F \in \mathcal{CLS}$):*

$$\mathrm{r}_0(F) := \begin{cases} \{\bot\} & \text{if } \bot \in F \\ F & \text{otherwise} \end{cases}$$

$$\mathrm{r}_{k+1}(F) := \begin{cases} \mathrm{r}_{k+1}(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \mathrm{lit}(F) : \mathrm{r}_k(\langle x \to 0\rangle * F) = \{\bot\} \\ F & \text{otherwise} \end{cases}.$$

r_1 is unit-clause propagation, r_2 is (full) failed literal elimination. We call r_k **generalised unit-clause-propagation of level** k. In [104] one finds the basic observations in Lemma 3.1.2 proven.

Lemma 3.1.2 *For all $k \in \mathbb{N}_0$, clause-sets $F \in \mathcal{CLS}$ and partial assignments $\varphi \in \mathcal{PASS}$ we have that:*

- *The map $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ is well-defined (does not depend on the choices)*[1)].
- *The map r_k is a reduction in the sense of Definition 2.4.1.*
- *r_k applies only forced assignments (and so $\mathrm{r}_k(F)$ is satisfiability-equivalent to F).*
- *$\mathrm{r}_k(F)$ is computable in time $O(\ell(F) \cdot n(F)^{2(\max(k,1)-1)})$ and linear space.*
- *$\mathrm{r}_k(F) = \{\bot\}$ implies $\mathrm{r}_k(\varphi * F) = \{\bot\}$.*
- *$\mathrm{r}_k(\varphi * \mathrm{r}_k(F)) = \mathrm{r}_k(\varphi * F)$.*

Quasi-automatisation of tree-resolution is achieved for inputs $F \in \mathcal{USAT}$ by applying $\mathrm{r}_0(F), \mathrm{r}_1(F), \ldots$ until unsatisfiability has been achieved ([104]). Actually, a more general form was introduced in [104], namely $\mathrm{r}_k^{\mathcal{U}}$ for some oracle $\mathcal{U}$ deciding unsatisfiability at level 0. It is likely that this generalisation is important for future progress (see Section 6.1), however in the majority of this thesis only the trivial oracle $\mathcal{U} = \{F \in \mathcal{CLS} : \bot \in F\}$ is considered, which recognises unsatisfiability at level 0 iff the empty clause occurs (see Subsection 4.3.2 for some discussion of this choice). A further generalisation to constraint-like systems (via an abstract, axiomatic approach) was achieved in [110], however in this initial study attention is restricted to boolean values and CNF-representations.

Example 3.1.3 *Computing some $\mathrm{r}_k(F)$ (using literals $x_1, \ldots, x_n, x, y$ with pairwise different underlying variables):*

1. $\mathrm{r}_k(\{\bot\}) = \{\bot\}$ *for* $k \geq 0$.
2. $\mathrm{r}_k(\top) = \top$ *for* $k \geq 0$.
3. *For* $F := \{\{x_1\}, \ldots, \{x_n\}\}$: $\mathrm{r}_0(F) = F$, $\mathrm{r}_k(F) = \top$ *for* $k \geq 1$.
4. *For* $F' := F \cup \{\{x, y\}\}$: $\mathrm{r}_0(F') = F'$, $\mathrm{r}_k(F') = \{\{x, y\}\}$ *for* $k \geq 1$ *(note that* $\{\{x, y\}\}$ *has no forced assignments).*
5. *For* $F := \{\{x, y\}, \{x, \overline{y}\}\}$: $\mathrm{r}_k(F) = F$ *for* $k \leq 1$, $\mathrm{r}_k(F) = \top$ *for* $k \geq 2$.
6. *For* $F := \{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\}$: $\mathrm{r}_k(F) = F$ *for* $k \leq 1$, $\mathrm{r}_k(F) = \{\bot\}$ *for* $k \geq 2$.

Via the reductions r_k we can approximate the implication relation $F \models C$ as follows:

Definition 3.1.4 ([104, 110]) *For $k \in \mathbb{N}_0$, clause-sets F and clauses C the relation $\boldsymbol{F \models_k C}$ holds if $\mathrm{r}_k(\varphi_C * F) = \{\bot\}$.*

As it is well-known, $F \models_1 C$ iff some subclause of C follows from F via input resolution.

Example 3.1.5 *Consider $k \in \mathbb{N}_0$ and literals x, y, w:*

1. *For all $k \geq 0$ and all clauses C we have:*

[1)] This follows essentially via the diamond lemma ([126, 92]) and the fact that any rewrite system which sets only forced literals is locally confluent (setting one forced literal to true can't remove another forced literal) and terminating - see e.g., Lemma 3.13 in [104].

(a) $F \models_k C$ *if there is* $D \in F$ *with* $D \subseteq C$ *(note* $\bot \in \varphi_C * F$*).*

(b) $\{\bot\} \models_k C$ *and* $\top \not\models_k C$.

2. $\{\{x, y\}, \{x, \overline{y}\}\} \models_k \{x\}$ *iff* $k \geq 1$.

3. *For* $F := \{\{\overline{x}, y\}, \{\overline{y}, z\}\}$ *we have* $F \models_k \{\overline{x}, z\}$ *iff* $k \geq 1$.

4. *For* $F := \{\{\overline{x}, y, w\}, \{\overline{y}, z, w\}, \{\overline{x}, y, \overline{w}\}, \{\overline{y}, z, \overline{w}\}\}$ *we have* $F \models_k \{\overline{x}, z\}$ *iff* $k \geq 2$ *(note that* $\langle x \to 1, z \to 0\rangle * F \in 2\text{-}\mathcal{CLS}$*).*

In order for r_k to "trigger", it must find a clause of length at most k (somewhat strengthening Lemma 3.18, Part 2(b) in [104]; the proof follows by definition, using an easy induction):

Lemma 3.1.6 *Consider* $F \in \mathcal{CLS}$ *and* $k \in \mathbb{N}_0$. *If for all clauses* $C \in F$ *we have* $|C| > k$, *then* $\mathrm{r}_k(F) = F$.

Furthermore, for any given clause-set F, $\mathrm{r}_{n(F)}$ will find all forced assignments:

Lemma 3.1.7 *For* $F \in \mathcal{CLS}$ *holds* $\mathrm{r}_\infty(F) = \mathrm{r}_{n(F)}(F)$.

Proof: If $\mathrm{var}(F) = \emptyset$ then trivially by definition we have $\mathrm{r}_\infty(\{\bot\}) = \mathrm{r}_0(\{\bot\}) = \{\bot\}$ and $\mathrm{r}_\infty(\top) = \mathrm{r}_{n(F)}(\top) = \top$. For $n > 0$ we proceed by induction on $|\mathrm{var}(F)|$. Consider a forced literal x of F. We have $\langle x \to 1\rangle * F \in \mathcal{USAT}$ and so by induction $\mathrm{r}_{n(F)-1}(\langle x \to 1\rangle * F) = \{\bot\}$ and hence by definition $\mathrm{r}_{n(F)}(F) = \mathrm{r}_{n(F)}(\langle x \to 1\rangle * F)$. □

3.2 Generalised input resolution

In [104], Chapter 4, the *leveled height* "$h(T)$" of branching trees T has been introduced, which was further generalised in [110], Chapter 3 (to a general form of constraint satisfaction problems). It handles satisfiable as well as unsatisfiable clause-sets. In this thesis we will mostly use the unsatisfiable case. In this case the measure reduces to a well-known measure which only considers the structure of the tree. As discussed in Subsections 4.2, 4.3 of [104], the leveled height of splitting trees for unsatisfiable clause-sets appeared at many places in the literature. [2] used the term "Horton-Strahler number" (sometimes also "Strahler number"): it seems the oldest source (from 1945), however disconnected from its various (re-)inventions in computer science. We use this notation here, since we consider the leveled height mostly for the unsatisfiable case; as in [2], the Horton-Strahler number of the trivial tree is 0.

Definition 3.2.1 *Consider a resolution tree* T *(recall Definition 2.5.2). The* ***Horton-Strahler number*** $\mathbf{hs(T)} \in \mathbb{N}_0$ *is defined as* $\mathrm{hs}(T) := 0$, *if* T *is trivial (consists only of one node), while otherwise we have two subtrees* T_1, T_2, *and we set* $\mathrm{hs}(T) := \max(\mathrm{hs}(T_1), \mathrm{hs}(T_2))$ *if* $\mathrm{hs}(T_1) \neq \mathrm{hs}(T_2)$, *while in case of* $\mathrm{hs}(T_1) = \mathrm{hs}(T_2)$ *we set* $\mathrm{hs}(T) := \mathrm{hs}(T_1) + 1$ *(*$= \mathrm{hs}(T_2) + 1$*).*

Remarks:

1. See Sections 4.2, 4.3 in [104] for various characterisations of $\mathrm{hs}(T)$.

Example 3.2.2 *Examples of trees with their Horton-Strahler numbers. We denote by* T_1 *and* T_2 *in each example the left and right sub-trees of the root.*

Tree T	hs(T)	*Explanation*
	0	*trivial tree*
	1	$\mathrm{hs}(T_1) = 0$, $\mathrm{hs}(T_2) = 0$.
	1	$\mathrm{hs}(T_1) = 0$, $\mathrm{hs}(T_2) = 1$.
	1	$\mathrm{hs}(T_1) = 0$, $\mathrm{hs}(T_2) = 1$.
	2	$\mathrm{hs}(T_1) = 1$, $\mathrm{hs}(T_2) = 1$.
	2	$\mathrm{hs}(T_1) = 1$, $\mathrm{hs}(T_2) = 2$.

In [104], Chapter 7 (generalised in [110], Chapter 5), *generalised input resolution* was introduced. We use the notation "$\vdash_k$" for it:

Definition 3.2.3 *For a clause-set F and a clause C the relation $F \vdash_k C$ (C can be derived from F by **k-times nested input resolution**) if there exists a resolution tree T and $C' \subseteq C$ with $T : F \vdash C'$ and $\mathrm{hs}(T) \leq k$.*

By Theorem 7.5 in [104], generalised in Corollary 5.12 in [110][2)].

Lemma 3.2.4 *For clause-sets F, clauses C and $k \in \mathbb{N}_0$ we have $F \models_k C$ if and only if $F \vdash_k C$.*

3.3 Hardness of unsatisfiable clause-sets

In [104] the following hardness parameter was introduced and investigated (further generalised in [110]). The core idea in [104] was to introduce mechanisms and classes for polynomial time satisfiability with poly-time recognition and strong connections to resolution complexity. In Section 3.4, a new hardness measure will be introduced which will form the basis of the "target-classes" introduced in this thesis, where poly-time satisfiability and the strong connections to resolution are maintained but, as we will see in Theorem 4.4.5 in Chapter 4, poly-time recognition is lost.

Definition 3.3.1 *The **hardness** $\mathrm{hd}(F)$ of an unsatisfiable $F \in \mathcal{CLS}$ is the minimal $k \in \mathbb{N}_0$ such that $\mathrm{r}_k(F) = \{\bot\}$.*

Remarks:

1. In [104, 110] the notation "$\mathrm{h}(F)$" was used (resp., more generally, "$\mathrm{h}_{\mathcal{U},\mathcal{S}}(F)$", using oracles for unsatisfiability and satisfiability detection), which seems now to us too unspecific.

[2)] Lemma 3.2.4 essentially follows from the general correspondence between regular tree-resolution proofs and backtracking trees (which r_k form), as well as the fact that minimum-size tree-resolution proofs are regular.

2. By Lemma 3.7 of [104], we have that for unsatisfiable F and fixed k we can check "Is $\mathrm{hd}(F) \leq k$?" in running time $O(\ell(F) \cdot n(F)^{2k})$.

By Theorem 7.8 (and Corollary 7.9) in [104] (or, more generally, Theorem 5.14 in [110]) we have for $F \in \mathcal{USAT}$:

$$2^{\mathrm{hd}(F)} \leq \frac{1}{2}(\mathrm{Comp}^*_{\mathrm{R}}(F) + 1) \leq (n(F) + 1)^{\mathrm{hd}(F)}$$

(recall Definition 2.5.2; recall that a binary tree with $k \in \mathbb{N}$ leaves has exactly $2k - 1$ nodes, and note that in [104, 110] the numbers of leaves are counted, while according to Definition 2.5.2 here we count nodes). This shows a direct connection between hd and tree-resolution complexity and in Section 7.2.1 of [104] we see for unsatisfiable F that $\mathrm{hd}(F) + 1$ is precisely the *tree resolution space complexity.*

Lemma 3.3.2 *Consider an unsatisfiable clause-set $F \in \mathcal{USAT}$ and $k \in \mathbb{N}_0$. The following are equivalent:*

1. *F has hardness k, i.e., k is the smallest value s.t. $F \models_k \bot$ and equivalently $F \vdash_k \bot$.*
2. *F has tree resolution space complexity $k + 1$.*
3. *k is the optimal value of the Pudlák-Impagliazzo-game on F (in [133], Pudlák-Impagliazzo showed that the optimal value of their game theoretic complexity measurement was the tree resolution space-complexity plus one).*

Example 3.3.3 *Here are some basic determinations of* $\mathrm{hd}(F)$ *for unsatisfiable clause-sets F, using literals x, y, z with distinct underlying variables:*

1. $\mathrm{hd}(F) = 0$ *iff* $\bot \in F$.
2. $\mathrm{hd}(\{\{x\}, \{\overline{x}\}\}) = 1$.
3. $\mathrm{hd}(\{\{x\}, \{\overline{x}, y\}, \{\overline{y}, z\}, \{\overline{z}\}\}) = 1$.
4. $\mathrm{hd}(\{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\}) = 2$.
5. $\mathrm{hd}(\{\{x, \overline{y}\}, \{\overline{x}, y\}, \{y, \overline{z}\}, \{\overline{y}, z\}, \{x, y, z\}, \{\overline{x}, \overline{y}, \overline{z}\}\}) = 2$.

By Lemma 3.2.4 we get:

Lemma 3.3.4 *For an unsatisfiable clause-set F and $k \in \mathbb{N}_0$ the following conditions are equivalent:*

1. $\mathrm{hd}(F) \leq k$
2. $F \models_k \bot$
3. $F \vdash_k \bot$.

By applying partial assignments we can reach all hardness-levels in a clause-set:

Lemma 3.3.5 *For an unsatisfiable clause-set F and every $0 \leq k \leq \mathrm{hd}(F)$ there exists a partial assignment φ with $n(\varphi) = k$ and $\mathrm{hd}(\varphi * F) = \mathrm{hd}(F) - k$.*

Proof: We proceed by induction on $n(F)$. As $k \leq \mathrm{hd}(F) \leq n(F)$, for the base case we consider $n(F) = k$. If $n(F) = k$ then all φ with $n(\varphi) = k$ have $\mathrm{hd}(\varphi * F) = \mathrm{hd}(\{\bot\}) = 0 = \mathrm{hd}(F) - k$. For $n(F) > k$, we make a case distinction on the value of k. If $k = 0$ then choose $\varphi = \langle\rangle$. If $k = 1$ then:

1. Assume for the sake of contradiction that there is no $x \in \mathrm{lit}(F)$ such that $\mathrm{hd}(\langle x \to 1\rangle * F) = \mathrm{hd}(F) - 1$; otherwise we are done.

2. If for all $x \in \mathrm{lit}(F)$ we had $\mathrm{hd}(\langle x \to 1\rangle * F) \leq \mathrm{hd}(F) - 2$ then by Definition 3.3.1 we would have $\mathrm{hd}(F) \leq k - 1$, a contradiction.

3. Therefore there must exist an $x \in \mathrm{lit}(F)$ such that

$$\mathrm{hd}(F) = \mathrm{hd}(\langle x \to 1\rangle * F) > \mathrm{hd}(\langle x \to 0\rangle * F) + 1.$$

4. By induction hypothesis we have a partial assignment φ with $n(\varphi) = 1$ such that $\mathrm{hd}(\varphi * (\langle x \to 1\rangle * F)) = \mathrm{hd}(F) - 1$.

5. Application of partial assignments doesn't increase hardness (Lemma 3.11 of [104]) and so we have

$$\mathrm{hd}(\varphi * F) \geq \mathrm{hd}(\langle x \to 1\rangle * (\varphi * F)) = \mathrm{hd}(F) - 1.$$

6. By our choice of x we have

$$\begin{aligned} \mathrm{hd}(\langle x \to 1\rangle * (\varphi * F)) &= \mathrm{hd}(F) - 1 \\ \mathrm{hd}(\langle x \to 0\rangle * (\varphi * F)) &\leq \mathrm{hd}(F) - 2, \end{aligned}$$

   therefore by Definition 3.3.1 we have $\mathrm{hd}(\varphi * F) \leq \mathrm{hd}(F) - 1$.

7. Thus we have that $\mathrm{hd}(\varphi * F) = \mathrm{hd}(F) - 1$.

Finally, for $k > 1$, we apply induction using the $k = 1$ case; once we can reduce by 1 we can reduce by k. □

As a corollary of Lemma 3.3.5, we see that to prove $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_k(\varphi * F)$ it suffices to find some $\varphi' \supseteq \varphi$ with $\mathrm{r}_{k+1}(\varphi' * F) \neq \mathrm{r}_k(\varphi' * F)$, allowing a poly-size witness φ' that $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_k(\varphi * F)$ with a poly-time check for fixed k (i.e., running r_k and r_{k+1}).

Corollary 3.3.6 *For all clause-sets F, partial assignments φ, and $k \in \mathbb{N}_0$ if $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_k(\varphi * F)$ then there exists a partial assignment $\varphi' \supseteq \varphi$ such that $\mathrm{r}_{k+1}(\varphi' * F) \neq \mathrm{r}_k(\varphi' * F)$.*

Proof: If $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_k(\varphi * F)$ then by definition there exists some forced literal $x \in \mathrm{lit}(\mathrm{r}_k(\varphi * F))$ for which $\mathrm{r}_{k-1}(\langle x \to 0\rangle * \mathrm{r}_k(\varphi * F)) \neq \{\bot\}$. Denote by $\varphi' \supseteq \varphi$ the partial assignment extending φ such that $\varphi' * F = \mathrm{r}_k(\varphi * F)$ (recall r_k just assigns literals). If $\mathrm{hd}(\langle x \to 0\rangle * (\varphi' * F)) = k$ then by definition $\mathrm{r}_{k+1}(\varphi' * F) \neq \mathrm{r}_k(\varphi' * F) = \mathrm{r}_k(\varphi * F)$ because r_{k+1} sets x while r_k does not. Otherwise $\mathrm{hd}(\langle x \to 0\rangle * (\varphi' * F)) \geq k + 1$ and $\langle x \to 0\rangle * (\varphi' * F)$ is unsatisfiable (recall x was forced) and so by Lemma 3.3.5 there is some $\varphi'' \supseteq \langle x \to 0\rangle \circ \varphi'$ such that $\mathrm{hd}(\varphi'' * F) = k + 1$, so $rk_{k+1}(\varphi'' * F) \neq \mathrm{r}_k(\varphi'' * F)$. □

3.3.1 Lower bounding hardness

For showing lower bounds on the hardness for unsatisfiable clause-sets, we can use the methodology developed in Subsection 3.4.2 of [104]. A simplified version of Lemma 3.17 from [104], sufficient for our purposes, is as follows (we give a proof for the sake of completeness):

Lemma 3.3.7 *Consider $\mathcal{C} \subseteq \mathcal{USAT}$ and a function $h : \mathcal{C} \to \mathbb{N}_0$. For $k \in \mathbb{N}_0$ let $\mathcal{C}_k := \{F \in \mathcal{C} : h(F) \geq k\}$. Then $\forall F \in \mathcal{C} : \mathrm{hd}(F) \geq h(F)$ holds if and only if $\mathcal{UC}_0 \cap \mathcal{C}_1 = \emptyset$ (where $\mathcal{UC}_0 = \{F \in \mathcal{CLS} : \mathrm{hd}(F) = 0\}$), and for all $k \in \mathbb{N}$, $F \in \mathcal{C}_k$ and $x \in \mathrm{lit}(F)$ there exist clause-sets $F_0, F_1 \in \mathcal{CLS}$ fulfilling the following three conditions:*

1. *$n(F_\varepsilon) \leq n(\langle x \to \varepsilon\rangle * F)$ for both $\varepsilon \in \{0,1\}$;*
2. *$\mathrm{hd}(F_\varepsilon) \leq \mathrm{hd}(\langle x \to \varepsilon\rangle * F)$ for both $\varepsilon \in \{0,1\}$;*
3. *$F_0 \in \mathcal{C}_k$ or $F_1 \in \mathcal{C}_{k-1}$.*

Proof: The given condition is necessary, since we can choose $F_\varepsilon := \langle v \to \varepsilon\rangle * F$ for $\varepsilon \in \{0,1\}$. To see sufficiency, assume for the sake of contradiction that there is $F \in \mathcal{C}$ with $\mathrm{hd}(F) < h(F)$, and consider such an F with minimal $n(F)$. If $\mathrm{hd}(F) = 0$, so $h(F) = 0$ (otherwise $F \in \mathcal{UC}_0 \cap \mathcal{C}_1$), and thus $\mathrm{hd}(F) \geq 1$ holds. It follows that there is $x \in \mathrm{lit}(F)$ with $\mathrm{hd}(\langle x \to 1\rangle * F) < \mathrm{hd}(F)$. Let $k := h(F)$; so $F \in \mathcal{C}_k$. By assumption there are $F_0, F_1 \in \mathcal{CLS}$ with $\mathrm{hd}(F_\varepsilon) \leq \mathrm{hd}(\langle x \to \varepsilon\rangle * F)$ for both $\varepsilon \in \{0,1\}$, and $F_0 \in \mathcal{C}_k$ or $F_1 \in \mathcal{C}_{k-1}$. If $F_0 \in \mathcal{C}_k$, then $\mathrm{hd}(F_0) \leq \mathrm{hd}(F) < k \leq h(F_0)$, while $n(F_0) < n(F)$, contradicting minimality of F. And if $F_1 \in \mathcal{C}_{k-1}$, then $\mathrm{hd}(F_1) \leq \mathrm{hd}(F) - 1 < k - 1 \leq h(F_1)$, while $n(F_1) < n(F)$, contradicting again minimality of F. □

3.4 Hardness of (arbitrary) clause-sets

The hardness $\mathrm{hd}(F)$ of arbitrary clause-sets can now be defined as the maximum hardness over all unsatisfiable instances obtained by partial assignments, in this way quantifying the hardness of clausal entailment for any clause.

Definition 3.4.1 *The **hardness** $\mathbf{hd(F)} \in \mathbb{N}_0$ for $F \in \mathcal{CLS}$ is the minimal $k \in \mathbb{N}_0$ such that for all clauses C with $F \models C$ we have $F \models_k C$ (recall Definition 3.1.4; by Lemma 3.2.4 this is equivalent to $F \vdash_k C$, i.e., $\mathrm{r}_k(\varphi * F) = \{\bot\}$).*

Remarks:

1. This differs from the hardness notion for *satisfiable* clause-sets from [104], which instead measures the maximum number of decisions (i.e., assignments which aren't determined to be forced by r_k) needed to deduce the empty clause-set.

In other words, if $F \neq \top$ then $\mathrm{hd}(F)$ is the maximum of $\mathrm{hd}(\varphi * F)$ for partial assignments φ such that $\varphi * F \in \mathcal{USAT}$. To our knowledge, the measure $\mathrm{hd}(F)$ for satisfiable F was mentioned the first time in the literature in [2], Definition 8 (the only result there concerning this measure is Lemma 9, relating it to another hardness-alternative for satisfiable F). Note that one can restrict attention in Definition 3.4.1 to $C \in \mathrm{prc}_0(F)$. Hardness 0 means that all prime clauses are there, i.e., $\mathrm{hd}(F) = 0$ iff $\mathrm{prc}_0(F) \subseteq F$. Especially $\mathrm{hd}(\top) = 0$.

Lemma 3.3.5, stating that $\mathrm{hd}(\varphi * F)$ takes exactly the values from 0 to $\mathrm{hd}(F)$, extends by definition to satisfiable $F \in \mathcal{CLS}$, when adding to the size of the partial assignment φ the minimum size of a partial assignment ψ with $\psi * F \in \mathcal{USAT}$ and $\mathrm{hd}(\psi * F) = \mathrm{hd}(F)$. Furthermore,

a trivial but fundamental lemma following from Lemma 3.11 of [104] and by the definition of $\mathrm{hd}(F)$:

Lemma 3.4.2 *For $F \in \mathcal{CLS}$ and $\varphi \in \mathcal{PASS}$ we have $\mathrm{hd}(\varphi * F) \leq \mathrm{hd}(F)$.*

Important here is that due to the closure of hd under partial assignment, when considering partial assignments that lead to unsatisfiability, it suffices now to consider minimal satisfying assignments, i.e., prime implicates.

Lemma 3.4.3 *For all clause-sets F we have that $\mathrm{hd}(F)$ is the minimal $k \in \mathbb{N}_0$ such that for all prime clauses $C \in \mathrm{prc}_0(F)$ we have $F \models_k C$ (recall Definition 3.1.4; by Lemma 3.2.4 this is equivalent to $F \vdash_k C$).*

A first result using this new notion is that, based on the results from [104, 110], a powerful proof-theoretic characterisation of clause-sets F with $\mathrm{hd}(F) \leq k$ can be given:

Theorem 3.4.4 *For $k \in \mathbb{N}_0$ and $F \in \mathcal{CLS}$ we have*

$$\mathrm{hd}(F) \leq k \iff \forall C \in \mathrm{prc}_0(F) : F \vdash_k C.$$

Thus if every $C \in \mathrm{prc}_0(F)$ has a tree-resolution refutation using at most $2^{k+1} - 1$ leaves (i.e., $\mathrm{Comp}^*_{\mathrm{R}}(\varphi_C * F) < 2^{k+1}$), then $\mathrm{hd}(F) \leq k$.

Proof: The equivalence $\mathrm{hd}(F) \leq k \Leftrightarrow \forall C \in \mathrm{prc}_0(F) : F \vdash_k C$ follows from Lemma 3.2.4. And if $\mathrm{hd}(F) > k$, then there is $C \in \mathrm{prc}_0(F)$ with $F \nvdash_k C$, and then every tree-resolution derivation of C from F needs at least 2^{k+1} leaves due to $2^{\mathrm{hd}(\varphi_C * F)} \leq \mathrm{Comp}^*_{\mathrm{R}}(\varphi_C * F)$ (as stated before). □

Example 3.4.5 *Here are some basic determinations of hardness for satisfiable clause-sets (for unsatisfiable F see Example 3.3.3), using Theorem 3.4.4 (w, x, y, z are literals with distinct underlying variables):*

1. $\mathrm{hd}(\top) = 0$.
2. $\mathrm{hd}(\{\{x\}\}) = 0$.
3. *For $F := \{\{x, y\}, \{x, \overline{y}\}\}$ we have $\mathrm{hd}(F) = 1$:*
   (a) $\mathrm{prc}_0(F) = \{\{x\}\}$.
   (b) $\mathrm{hd}(\langle x \to 0\rangle * F) = \mathrm{hd}(\{\{y\}, \{\overline{y}\}\}) = 1$.
4. *For $F := \{\{\overline{x}, y\}, \{\overline{y}, z\}\}$ we have $\mathrm{hd}(F) = 1$:*
   (a) $\mathrm{prc}_0(F) = \{\{\{\overline{x}, y\}, \{\overline{y}, z\}, \{\overline{x}, z\}\}\}$.
   (b) $\mathrm{hd}(\langle x \to 1, y \to 0\rangle * F) = \mathrm{hd}(\{\bot\}) = 0$.
   (c) $\mathrm{hd}(\langle y \to 1, z \to 0\rangle * F) = \mathrm{hd}(\{\bot\}) = 0$.
   (d) $\mathrm{hd}(\langle x \to 1, z \to 0\rangle * F) = \mathrm{hd}(\{\{y\}, \{\overline{y}\}\}) = 1$.
5. *For $F := \{\{z, x, y\}, \{z, x, \overline{y}\}, \{z, \overline{x}, y\}, \{z, \overline{x}, \overline{y}\}\}$ we have $\mathrm{hd}(F) = 2$:*
   (a) $\mathrm{prc}_0(F) = \{\{z\}\}$.
   (b) $\mathrm{hd}(\langle z \to 0\rangle * F) = \mathrm{hd}(\{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\} = 2$.

3.4.1 Some basic hardness determinations

The following basic lemma follows directly by definition and Lemma 3.4.2:

Lemma 3.4.6 *If two clause-sets F and F' are variable-disjoint, then we have:*

1. *If $F, F' \in \mathcal{SAT}$, then* $\mathrm{hd}(F \cup F') = \max(\mathrm{hd}(F), \mathrm{hd}(F'))$.
2. *If $F \in \mathcal{SAT}$ and $F' \in \mathcal{USAT}$, then* $\mathrm{hd}(F \cup F') = \mathrm{hd}(F')$.
3. *If $F, F' \in \mathcal{USAT}$, then* $\mathrm{hd}(F \cup F') = \min(\mathrm{hd}(F), \mathrm{hd}(F'))$.

Proof: First consider case 1. To see that $\mathrm{hd}(F \cup F') \geq \max(\mathrm{hd}(F), \mathrm{hd}(F'))$, assume w.l.o.g that $\mathrm{hd}(F') \geq \mathrm{hd}(F)$ and consider a satisfying assignment φ for F'. We then have that $\mathrm{hd}(F \cup F') \geq \mathrm{hd}(\varphi * (F \cup F')) = \mathrm{hd}(F')$ (φ satisfies all the clauses in F but doesn't touch F') by Lemma 3.4.2. To see that $\mathrm{hd}(F \cup F') \leq \max(\mathrm{hd}(F), \mathrm{hd}(F'))$, consider for the sake of contradiction that there is a partial assignment φ observing that $\mathrm{hd}(F \cup F') > \max(\mathrm{hd}(F), \mathrm{hd}(F'))$ (i.e., $F \cup F' \not\vdash_{\max(\mathrm{hd}(F),\mathrm{hd}(F'))} C_\varphi$). φ must falsify F or F' individually (otherwise just extend both to a total satisfying assignment; possible due to disjointness). However, this means that $F \vdash_{\mathrm{hd}(F)} C_\varphi$ or $F' \vdash_{\mathrm{hd}(F')} C_\varphi$ hence $F \vdash_{\min(\mathrm{hd}(F),\mathrm{hd}(F'))} C_\varphi$, hence $F \vdash_{\min(\mathrm{hd}(F),\mathrm{hd}(F'))} C_\varphi$ (recall that a resolution refutation for a subset of a clause-set refutes the superset), a contradiction and so $\mathrm{hd}(F \cup F') \leq \max(\mathrm{hd}(F), \mathrm{hd}(F'))$. Case 2 follows in essentially the same way.

Finally consider case 3. That $\mathrm{hd}(F \cup F') \leq \min(\mathrm{hd}(F), \mathrm{hd}(F'))$ follows from the same reasoning as for case 1. To see that $\mathrm{hd}(F \cup F') \geq \min(\mathrm{hd}(F), \mathrm{hd}(F'))$, assume for the sake of contradiction that there is some resolution proof $T : F \cup F' \vdash_k \bot$ for $k < \min(\mathrm{hd}(F), \mathrm{hd}(F'))$. F and F' are variable-disjoint and so T must be a resolution proof from either F or F' and so $\min(\mathrm{hd}(F), \mathrm{hd}(F')) \leq k$, a contradiction. □

Via full clause-sets A_n with n variables and 2^n clauses we obtain (unsatisfiable, simplest) examples with $\mathrm{hd}(A_n) = n$, and when removing one clause for $n \geq 1$, then we obtain satisfiable examples A'_n with $\mathrm{hd}(A'_n) = n - 1$, maximising hardness:

Lemma 3.4.7 *Consider a full clause-set $F \in \mathcal{CLS}$ (i.e., each clause contains all variables).*

1. $\mathrm{hd}(\top) = 0$.
2. *If F is unsatisfiable then* $\mathrm{hd}(F) = n(F)$.
3. *If $F \neq \top$, then* $\mathrm{hd}(F) = n(F) - \min_{C \in \mathrm{prc}_0(F)} |C|$.
4. *If for F no two clauses are resolvable, then* $\mathrm{hd}(F) = 0$.

Proof: Part 1 follows by Definition, Part 2 is Lemma 3.18 in [104], while Part 4 follows from Part 3. It remains to show Part 3. If F is unsatisfiable, then we get Part 2. For satisfiable F and a partial assignment φ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$ it is $\varphi * F$ a full clause-set with $n(\varphi * F) = n(F) - n(\varphi)$, and so the assertion follows by reduction to the unsatisfiable case. □

The following lemma yields a way of pumping up hardness:

Lemma 3.4.8 *Consider $F \in \mathcal{CLS}$ and $v \in \mathcal{VA} \setminus \mathrm{var}(F)$. Let $F' := \{C \cup \{v\} : C \in F\} \cup \{C \cup \{\overline{v}\} : C \in F\}$. Then we have* $\mathrm{hd}(F') = \mathrm{hd}(F) + 1$.

Proof: We have $\mathrm{hd}(F') \leq \mathrm{hd}(F) + 1$ by definition (if v is not set by the test-assignment, then it can be set to an arbitrary value, yielding a forced assignment at level $\mathrm{hd}(F)$). Now consider a partial assignment φ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$, $\varphi * F \in \mathcal{USAT}$ and $\mathrm{hd}(\varphi * F) = \mathrm{hd}(F)$. Now also $\varphi * F' \in \mathcal{USAT}$ holds, where $\varphi * F' = \{C \cup \{v\} : C \in \varphi * F\} \cup \{C \cup \{\overline{v}\} : C \in \varphi * F\}$. Thus we have reduced the assertion of the lemma to the special case where $F \in \mathcal{USAT}$, and where $\mathrm{hd}(F') \geq \mathrm{hd}(F) + 1$ is left to be shown. This now follows easily by induction on the number of variables. □

3.5 Propagation-hardness

While hardness measures the level of r_k necessary to decide consistency (i.e., checking $\varphi * F \in \mathcal{USAT}$ by $\mathrm{r}_k(\varphi * F) = \{\bot\}$), another possibility is to consider the level of r_k necessary to find all forced assignments.

Definition 3.5.1 *For $F \in \mathcal{CLS}$ we define the* ***propagation-hardness*** *("p-hardness")* $\mathbf{phd}(F) \in \mathbb{N}_0$ *as the minimal $k \in \mathbb{N}_0$ such that for all partial assignments $\varphi \in \mathcal{PASS}$ we have*

$$\mathrm{r}_k(\varphi * F) = \mathrm{r}_\infty(\varphi * F).$$

Remarks:

1. A clause-set $F \in \mathcal{CLS}$ having p-hardness $\leq k$ means that for every instantiation (application of partial assignments) the r_k-reduction catches all forced assignments.
2. Regarding zero-measures we have:
   (a) For a clause-set $F \in \mathcal{CLS}$ we have $\mathrm{phd}(F) = 0$ iff either $F = \top$ or $\bot \in F$ (since if there is $C \in F$ of minimal length with $|C| > 0$, then for $x \in C$ we have for $\varphi := \varphi_{C \setminus \{X\}}$ that $x \in \mathrm{fl}(\varphi * F)$ and $\mathrm{r}_0(\varphi * F) = \varphi * F \neq \mathrm{r}_\infty(\varphi * F)$).
   (b) If $\mathrm{hd}(F) = 0$, then $\mathrm{phd}(F) \leq 1$.
   (c) For unsatisfiable F we have $\mathrm{phd}(F) = \mathrm{hd}(F)$ in case of $\mathrm{hd}(F) \geq 1$.

As for hardness, a fundamental property of this hardness measure deriving from properties of the r_k reduction is its closure under application of partial assignment.

Lemma 3.5.2 *For $F \in \mathcal{CLS}$ and $\mathcal{PASS}$ we have* $\mathrm{phd}(\varphi * F) \leq \mathrm{phd}(F)$.

Proof: This follows from the fact that any partial assignment φ' witnessing that $\mathrm{phd}(\varphi * F) > k$ (i.e., $\mathrm{r}_\infty(\varphi' * (\varphi * F)) \neq \mathrm{r}_k(\varphi' * (\varphi * F))$) for some k yields the witness $\varphi' \circ \varphi$ for $\mathrm{phd}(F) \geq k$ (i.e., $\mathrm{r}_\infty((\varphi' \circ \varphi) * F) \neq \mathrm{r}_k((\varphi' \circ \varphi) * F)$). □

So propagation-hardness ensures that forced literals are found as well as inconsistency and so by the definition of forced literals it follows that $\mathrm{phd}(F) \leq \mathrm{hd}(F) + 1$:

Lemma 3.5.3 *For $F \in \mathcal{CLS}$ we have* $\mathrm{hd}(F) \leq \mathrm{phd}(F) \leq \mathrm{hd}(F) + 1$.

Proof: That $\mathrm{hd}(F) \leq \mathrm{phd}(F)$ follows by definition (any witnessing partial assignment for $\mathrm{hd}(F) \geq k$ also witnesses $\mathrm{phd}(F) \geq k$). To see that $\mathrm{phd}(F) \leq \mathrm{hd}(F) + 1$, assume for the sake of contradiction that $\mathrm{phd}(F) > \mathrm{hd}(F) + 1$, i.e., that there is some partial assignment φ such that $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_{\mathrm{hd}(F)+1}(\varphi * F)$. By definition of r_∞ there is some forced literal x, i.e.,

$\langle x \to 0\rangle * (\varphi * F) \in \mathcal{USAT}$, for which $\mathrm{r}_{\mathrm{hd}(F)}(\langle x \to 0\rangle * (\varphi * F)) \neq \{\bot\}$. This contradicts the definition of $\mathrm{hd}(F)$. □

Remarks:

1. A simple example where the upper bound of Lemma 3.5.3 is attained is the clause-set $F := \{\{a, b, c\}, \{\overline{a}, b, c\}, \{a, \overline{b}, c\}\}$: By Lemma 3.4.7 we have $\mathrm{hd}(F) = 1$, while $\mathrm{phd}(F) = 2$, since for $\langle c \to 0\rangle * F$ the two assignments $a, b \to 1$ are forced.

However, despite the ability for p-hardness to be higher than hardness, the number of variables is still a natural limit, as evidenced by the fact that $\mathrm{r}_{n(F)}$ will find all forced assignments of F (see Lemma 3.1.7).

Lemma 3.5.4 *For $F \in \mathcal{CLS}$ we have* $\mathrm{phd}(F) \leq n(F)$.

Lemma 3.5.2 along with Lemma 3.1.6 leads to simple yet not always immediately obvious applications, such as the separation of hardness and p-hardness for full-clause-sets.

Lemma 3.5.5 *For all full clause-sets $F \in \mathcal{CLS}$, $n := n(F)$, we have*

1. *If $0 < c(F) < 2^n$ then* $\mathrm{phd}(F) = \mathrm{hd}(F) + 1$.
2. *If $c(F) = 0$ or $c(F) = 2^n$ then* $\mathrm{phd}(F) = \mathrm{hd}(F)$.

(Recall Lemma 3.4.7 for $\mathrm{hd}(F)$*.)*

Proof: The assertions follow by Lemma 3.1.6, using the stability of the class of full clause-sets under partial assignments, while upper bounds follow via Lemma 3.5.4 and Lemma 3.5.3.

More precisely, to see that $\mathrm{phd}(F) \geq \mathrm{hd}(F) + 1$ for *all* full clause-sets with $0 < c(F) < 2^n$ (i.e., not just those F directly containing a forced literal) observe that by Theorem 3.4.4 there is a prime implicate C of F with $\varphi_C * F \in \mathcal{USAT}$ and $\mathrm{hd}(\varphi_C * F) = \mathrm{hd}(F)$ $(= n(\varphi_C * F))$. As C is a prime implicate, for all $x \in C$ we have that $\varphi_{C\setminus\{x\}} * F$ is satisfiable with forced literal x, and as F is a full-clause-set, so is $\varphi_{C\setminus\{x\}} * F$, hence $n(\varphi_{C\setminus\{x\}} * F) = n(\varphi_C * F) + 1 = \mathrm{hd}(F) + 1$. So by Lemma 3.1.6 and Lemma 3.5.2 we have $\mathrm{phd}(F) \geq \mathrm{phd}(\varphi_{C\setminus\{x\}} * F) \geq n(\varphi_{C\setminus\{x\}} * F) = \mathrm{hd}(F) + 1$. □

One might think for a clause-set F with a forced literal x that if $\mathrm{hd}(\langle x \to 0\rangle * F) = k$ then naturally $\mathrm{phd}(F) = k + 1$, i.e., r_{k+1} is needed to find x because r_k is needed to determine unsatisfiability of $\langle x \to 0\rangle * F$. However, this is not *always* the case, as illustrated in Lemma 3.5.6.

Lemma 3.5.6 *Consider $n \in \mathbb{N}$ and a full unsatisfiable clause-set F with $n(F) = n$. Consider a literal x with* $\mathrm{var}(x) \notin \mathrm{var}(F)$*, and $C \in F$, and let $F' := (F \setminus \{C\}) \cup \{C \cup \{x\}\}$. Then* $\mathrm{hd}(F') = \mathrm{phd}(F') = n$.

Proof: It suffices to show $\mathrm{hd}(F') \geq n$ and $\mathrm{phd}(F') \leq n$:

1. $\mathrm{hd}(F') \geq n$ follows from $\mathrm{hd}(F') \geq \mathrm{hd}(\langle x \to 0\rangle * F') = \mathrm{hd}(F) = n$.
2. To show that $\mathrm{phd}(F') \leq n$, we must show that for all partial assignments φ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F')$ holds $\mathrm{r}_n(\varphi * F') = \mathrm{r}_\infty(\varphi * F')$:
   (a) If $n(\varphi) \geq 1$ then $n(\varphi * F') \leq n$ and so $\mathrm{r}_n(\varphi * F') = \mathrm{r}_\infty(\varphi * F')$.

(b) Otherwise $\varphi * F' = F'$. Consider any literal $y \in C \setminus \{x\}$ (using $n \geq 1$ here). Observe that $\langle y \to 1\rangle * F'$ is full and unsatisfiable with $n(\langle y \to 1\rangle * F') \leq n-1$, and thus $\mathrm{hd}(\langle y \to 1\rangle * F') \leq n-1$. Therefore, by definition of r_n, we have $\mathrm{r}_n(F') = \mathrm{r}_n(\langle y \to 0\rangle * F')$. Now $n(\langle y \to 0\rangle * F') \leq n$, and therefore $\mathrm{r}_n(\langle y \to 0\rangle * F') = \mathrm{r}_\infty(\langle y \to 0\rangle * F') = \mathrm{r}_\infty(F')$ (since $\overline{y}$ is forced for F'). □

On the other hand, we can always "raise" the p-hardness of a clause-set up above the hardness via a simple construction:

Lemma 3.5.7 *Consider $F \in \mathcal{CLS}$ and a literal x with $\mathrm{var}(x) \notin \mathrm{var}(F)$. Let $F' := \{C \cup \{x\} : C \in F\}$. Then $\mathrm{phd}(F') = \mathrm{hd}(F) + 1$, while $\mathrm{hd}(F') = \mathrm{hd}(F)$.*

Proof: That $\mathrm{hd}(F') \geq \mathrm{hd}(F)$ follows by Lemma 3.4.2 and the fact that $F = \langle x \to 0\rangle * F'$. That $\mathrm{hd}(F') \leq \mathrm{hd}(F)$ follows from the fact that any partial assignment φ witnessing $\mathrm{hd}(F') > k$ (i.e., $\varphi * F \in \mathcal{USAT}$ and $\mathrm{r}_k(\varphi * F) \neq \{\bot\}$) for some k must set x to 0 (as $\langle x \to 1\rangle * F' = \top$), hence $\varphi * F' = \varphi * F$ and so φ also witnesses $\mathrm{hd}(F) > k$.

That $\mathrm{phd}(F') \leq \mathrm{hd}(F) + 1$ follows by Lemma 3.5.3. Finally, to see that $\mathrm{phd}(F') > \mathrm{hd}(F)$ assume for the sake of contradiction that $\mathrm{phd}(F') \leq \mathrm{hd}(F)$ and consider a partial assignment φ witnessing the hardness of F, i.e., such that $\varphi * F \in \mathcal{USAT}$ and $\mathrm{hd}(\varphi * F) = \mathrm{hd}(F)$. We have that $\varphi * F' = \{C \cup \{x\} : C \in \varphi * F\}$, so x is a forced literal for $\varphi * F'$ and so we have that $\mathrm{r}_\infty(\varphi * F') = \mathrm{r}_\infty(\langle x \to 1\rangle * F') = \top \neq F'$. Therefore by the definition of r_k there exists some literal x' such that $\mathrm{r}_{\mathrm{hd}(F)-1}(\langle x' \to 0\rangle * (\varphi * F)) = \{\bot\}$ and $\mathrm{r}_{\mathrm{hd}(F)}(\langle x \to 1\rangle * (\varphi * F')) = \top$. However, if $x' \neq x$ then setting x to 1 satisfies $\langle x' \to 0\rangle * (\varphi * F')$ and so $\mathrm{r}_{\mathrm{hd}(F)-1}(\langle x' \to 0\rangle * (\varphi * F')) \neq \{\bot\}$. Therefore we must have $x' = x$, but then $\mathrm{r}_{\mathrm{hd}(F)-1}(\langle x \to 0\rangle * (\varphi * F')) = \mathrm{r}_{\mathrm{hd}(F)-1}(\varphi * F) \neq \{\bot\}$ due to the hardness of F, a contradiction. □

Due to the close connection between r_k and k times nested input resolution from Lemma 3.2.4, we have that a clause-set F has a propagation-hardness $\leq k$ if under any partial assignment φ all forced literals follow via k times nested input resolution:

Lemma 3.5.8 *Consider a clause-set F and $k \in \mathbb{N}_0$. We have that $\mathrm{phd}(F) \leq k$ iff for all partial assignments $\varphi \in \mathcal{PASS}$ and literals $x \in \mathcal{LIT}$ that $\varphi * F \models x \implies \varphi * F \vdash_k x$.*

Remarks:

1. Recall that if $\varphi * F \in \mathcal{USAT}$ then $\varphi * F \models x$ for all literals x and so this is a strengthening of Definition 3.4.1 but less elegant due to the need to refer directly to the partial assignment φ rather than considering the complexity of deriving C_φ as in Definition 3.4.1.

We will see in Chapter 4.6 that this alternative characterisation of p-hardness links it directly to the class $\mathcal{PC}$ considered in [26] when $k = 1$ and allows a generalised hierarchy $\mathcal{PC}_k$ to be defined for higher k.

3.6 Width-based hardness

In Section 2.5, we saw that full-resolution is strictly more powerful than tree-resolution. Therefore, while a clause-set having low hardness offers a stronger guarantee (i.e., there are proofs in the weaker tree-resolution proof system, which are then also full-resolution proofs) it makes sense to contrast, compare and develop in tandem measures based on full-resolution. One of the core measures for full-resolution complexity is width.

A basic weakness of the standard notion of width-restricted resolution, which demands that *both* parent clauses must have length at most k for some fixed $k \in \mathbb{N}_0$ (the "width"; see [12]), is that even Horn clause-sets require unbounded width in this sense. The correct solution, as investigated and discussed in [104, 110], is to use the notion of "k-resolution" as introduced in [99], where only *one* parent clause needs to have length at most k (thus properly generalising unit-resolution). Nested input-resolution ([104, 110]) is the proof-theoretic basis of hardness, and approximates tree-resolution. In the same vein, k-resolution is the proof-theoretic basis of "w-hardness", and approximates dag-resolution (see Theorem 6.12 in [110]).

3.6.1 k-resolution

In [99] k-resolution was introduced, and further investigated in [104, 110]:

Definition 3.6.1 *Consider $k \in \mathbb{N}_0$.*

- *Two clauses C, D are **k-resolvable** if they are resolvable (recall Definition 2.5.1) and $|C| \le k \vee |D| \le k$, and the resolvent $C \diamond D$ is then called a **k-resolvent**.*
- *We use $F \vdash^k C$ if there is a resolution proof R with $R : F \vdash C'$ for some $C' \subseteq C$ (recall Definition 2.5.2) such that every resolvent in R is a k-resolvent.*

Remarks:

1. In Definition 3.1.2 we defined $F \models_k C$, in Definition 3.2.3 $F \vdash_k C$, and in Lemma 3.2.4 we showed that $F \models_k C \Leftrightarrow F \vdash_k C$.
2. We have $F \vdash_k C \Leftrightarrow F \vdash^k C$ for $k \le 1$, while for general $k \in \mathbb{N}_0$ holds $F \vdash_k C \Rightarrow F \vdash^k C$ (see [104, 110]).

3.6.2 Width-based hardness

Using the notion of k-resolution rather than k-times nested input resolution, we get a notion of hardness related to the (asymmetric) width complexity of full resolution, rather than tree resolution as with hd:

Definition 3.6.2 *For $F \in \mathcal{USAT}$ let $\mathbf{whd}(F) \in \mathbb{N}_0$ ("width hardness" or "w-hardness") be the minimal $k \in \mathbb{N}_0$ such that $F \vdash^k \bot$ holds. And for $F \in \mathcal{CLS}$ let $\mathrm{whd}(F) \in \mathbb{N}_0$ be the minimal $k \in \mathbb{N}_0$ such that for all partial assignments φ holds $\varphi * F \in \mathcal{USAT} \Rightarrow \varphi * F \vdash^k \bot$.*

Remarks:

1. We have $\mathrm{whd}(F) = k \Leftrightarrow \mathrm{hd}(F) = k$ for $k \in \{0, 1\}$, while in general $\mathrm{whd}(F) \le \mathrm{hd}(F)$ holds.
2. In general, unlike for hd, we do not have that $\mathrm{whd}(F) \le k$ iff $\forall C \in \mathcal{CL} : F \models C \iff F \vdash^k C$. This is because while hd measures a global structural property of tree-resolution proofs (i.e., the Horton-Strahler number), whd measures the local notion of width.

We have $\mathrm{whd}(F) \leq 1 \iff \mathcal{UC}(F) \leq 1$ and for all $k \in \mathbb{N}_0$ holds $\mathrm{whd}(F) \leq \mathrm{hd}(F)$ (this follows by Lemma 6.8 in [110] for unsatisfiable clause-sets, which extends to satisfiable clause-sets by definition). For unsatisfiable F, whether $\mathrm{whd}(F) = k$ holds for $k \in \{0,1,2\}$ can be decided in polynomial time; this is non-trivial for $k = 2$ ([33]) and unknown for $k > 2$.

If a clause-set $F \in \mathcal{USAT}$ has a k-resolution refutation R then via induction on R for all partial assignments φ there is a k-resolution proof of $\varphi * F$. Therefore, again as for hardness, we have the fundamental property w-hardness is closed under application of partial assignment.

Lemma 3.6.3 *For $F \in \mathcal{CLS}$ and $\mathcal{PASS}$ we have* $\mathrm{whd}(\varphi * F) \leq \mathrm{whd}(F)$.

As a special case of Theorem 6.12 in [110] we obtain for $F \in \mathcal{USAT}$, $n(F) \neq 0$, the following general lower bound on resolution complexity:

$$\mathrm{Comp_R}(F) > b^{\frac{\mathrm{whd}(F)^2}{n(F)}},$$

where $b := e^{\frac{1}{8}} = 1.1331484\ldots$, while $\mathrm{Comp_R}(F) \in \mathbb{N}$ is the minimal number of different clauses in a (tree-)resolution refutation of F. Similar to Theorem 14 in [76] resp. Theorem 5.7 in [77, 78] we thus obtain:

Lemma 3.6.4 *For $F \in \mathcal{CLS}$ and $k \in \mathbb{N}_0$, such that for every $C \in \mathrm{prc}_0(F)$ with $|C| < n(F)$ there exists a resolution proof of C from F using at most $b^{\frac{(k+1)^2}{n(F)-|C|}}$ different clauses, we have* $\mathrm{whd}(F) \leq k$.

Illustrating the "weaker" nature of whd, in Example 3.6.5 we see a simple case where whd is lower than hd.

Example 3.6.5 *Consider the following clause-set for literals $x_1, x_2, x'_1, x'_2, y_1$ with different underlying variables:*

$$F := \{ \underbrace{\{\overline{x_1}, x_2\}, \{x_1, \overline{x_2}\}}_{x_1 = x_2}, \underbrace{\{\overline{x_1}, \overline{x_2}, \overline{y_1}\}, \{x_1, x_2, \overline{y_1}\}}_{y_1 \to (x_1 \neq x_2)}, \underbrace{\{\overline{x'_1}, x'_2\}, \{x'_1, \overline{x'_2}\}}_{x'_1 = x'_2}, \underbrace{\{\overline{x'_1}, \overline{x'_2}, y_1\}, \{x'_1, x'_2, y_1\}}_{\overline{y_1} \to (x'_1 \neq x'_2)} \}$$

We have that $F \in \mathcal{USAT}$ and $\mathrm{whd}(F) = 2$ *but* $\mathrm{hd}(F) = 3$. *That* $\mathrm{whd}(F) \geq 2$ *comes from the fact that there are no unit-clauses. That* $\mathrm{whd}(F) \leq 2$ *is evidenced by the following 2-resolution proof:*

$$\dfrac{\dfrac{\dfrac{\{x_1, \overline{x_2}\} \quad \{\overline{x_1}, \overline{x_2}, \overline{y_1}\}}{\{\overline{x_2}, \overline{y_1}\}} \quad \dfrac{\{\overline{x_1}, x_2\} \quad \{x_1, x_2, \overline{y_1}\}}{\{x_2, \overline{y_1}\}}}{\{\overline{y_1}\}} \quad \dfrac{\dfrac{\{x'_1, \overline{x'_2}\} \quad \{\overline{x'_1}, \overline{x'_2}, y_1\}}{\{\overline{x'_2}, y_1\}} \quad \dfrac{\{\overline{x'_1}, x'_2\} \quad \{x'_1, x'_2, y_1\}}{\{x'_2, y_1\}}}{\{y_1\}}}{\bot}$$

That $\mathrm{hd}(F) \leq 3$ *is evidenced by the fact that the above resolution proof has Horton-Strahler number* 3. *To see that* $\mathrm{hd}(F) \geq 3$ *observe that for all literals $x \in \mathrm{lit}(F)$ we have* $\mathrm{r_1}(\langle x \to 0 \rangle * F) \neq \{\bot\}$ *($\mathrm{r_2}(F) \neq \{\bot\}$ hence $\mathrm{hd}(F) > 2$) and hence* $\mathrm{r_3}(F) = F \neq \{\bot\}$.

In general, from [127] it is known that there are families of unsatisfiable clause-sets with constant symmetric-width (hence also asymmetric) but linear full-resolution space-complexity. In Theorem 6.4.7 we shall see another example of such a linear separation between width (i.e., whd) and *tree*-resolution space complexity (i.e., hd).

3.7 Remarks on the term "hardness"

In general, if one speaks of the "hardness measure" $\mathrm{hd} : \mathcal{CLS} \to \mathbb{N}_0$ (Definition 3.4.1) in context with other measures, then one should call it more specifically *tree-hardness* ("t-hardness"), denoted by $\mathrm{thd}(F)$, due to its close relation to tree-resolution (and its space complexity). So we have three basic types of hardness-measures, namely t-hardness $\mathrm{thd}(F)$, p-hardness $\mathrm{phd}(F)$, and w-hardness $\mathrm{whd}(F)$. In this thesis, since $\mathrm{thd}(F)$ is still most important here, we denote it by $\mathrm{hd}(F) = \mathrm{thd}(F)$.

In what respect is the terminology "hardness" appropriate? The hardness measure $\mathrm{hd}(F)$ has been introduced in [104, 110], based on quasi-automatisation of tree-resolution, that is, on a specific algorithmic approach (close to Stålmarcks approach).[3] In [2] $\mathrm{hd}(F)$ for unsatisfiable F was proposed as measure of SAT-solver-hardness in general. This was criticised in [95] by the argument, that conflict-driven SAT solvers would be closer to dag-resolution (full resolution) than tree-resolution. Due to their heuristical nature, it seems to us that there is no robust measure of SAT-solver-hardness. Instead, our three basic measures, which are robust and mathematically meaningful, measure how good a clause-set F is in representing an underlying boolean function in the following sense:

- Regarding instantiation we take a worst-case approach, that is, we consider all partial assignments φ and their applications $\varphi * F$ (insofar they create unsatisfiability or forced literals).

  A SAT-solver only uses *certain* partial assignments, and thus this worst-case approach is overkill. However when using F in *any* context, then it makes sense to consider all partial assignments.

- Regarding algorithms, we take a breadth-first approach, that is, the smallest k such that r_k or k-resolution succeeds. For $k > 1$ a SAT-solver might not find these inferences. However those algorithms need polynomial time, and thus are implementable, and furthermore the maximisation over all partial assignments needs to be complemented with a minimisation in order to yield something interesting.

[3] Using the simplest oracle, on unsatisfiable instances the measure from [104, 110] yields $\mathrm{hd}(F)$. But on satisfiable instances the approach of [104, 110] is very different, namely an algorithmic polynomial-time approach is taken, extending the breadth-first search for tree-resolution refutations in a natural way.

Chapter 4

Classes and hierarchies for SAT knowledge compilation

The $\mathcal{SLUR}$ and $\mathcal{UC}$ classes are now described, and the hierarchies $\mathcal{SLUR}_k$ and $\mathcal{UC}_k$ derived. The fundamental result is then presented that $\mathcal{SLUR}_k = \mathcal{UC}_k$ is proven in Theorem 4.4.4 for two previously (thought to be) distinct classes, and bringing together the two conceptual streams of intuition from polynomial time SAT solving ($\mathcal{SLUR}$) and knowledge compilation ($\mathcal{UC}$). The basic properties of the hierarchies are proven, as well as the relations of $\mathcal{UC}_k$ to existing hierarchies, and the related hierarchies $\mathcal{PC}_k$ and $\mathcal{WC}_k$ defined and connected to $\mathcal{UC}_k$.

4.1 The SLUR class and extensions

The SLUR-algorithm and the class $\mathcal{SLUR} \subset \mathcal{CLS}$ have been introduced in [141]. The SLUR-algorithm for input $F \in \mathcal{CLS}$ is an incomplete polynomial-time SAT algorithm, which either returns "SAT", "UNSAT" (in both cases correctly) or gives up. This algorithm is non-deterministic, and $\mathcal{SLUR}$ is the class of clause-sets where it never gives up (and thus SAT-decision for $F \in \mathcal{SLUR}$ can be done in polynomial time). Due to an observation attributed to Truemper in [61], the SLUR-algorithm can be implemented such that it runs in linear time. Decision of membership, that is whether $F \in \mathcal{SLUR}$ holds, by definition is in coNP, but only in [36] it was finally shown that this decision problem is coNP-complete.

The original motivation was that $\mathcal{SLUR}$ contains several other classes, including renamable Horn, extended Horn, hidden extended Horn, simple extended Horn and CC-balanced clause-sets, where for each class it was known that the SAT problem is solvable in polynomial time, but with in some cases rather complicated proofs, while it is trivial to see that the SLUR-algorithm runs in polynomial time. In [61, 62] probabilistic properties of $\mathcal{SLUR}$ have been investigated.[1)]

In this section we first give a semantic definition of $\mathcal{SLUR}$ in Subsection 4.1.1. In a nutshell, $\mathcal{SLUR}$ is the class of clause-sets where either UCP (unit-clause propagation aka r_1) creates the empty clause, or where otherwise iteratively making assignments followed by UCP will always yield a satisfying assignment, given that these transitions do not obviously create unsatisfiable results, i.e., do not create the empty clause. In order to understand this definition (and its various

[1)] At this point a popular misunderstanding should be avoided: The well-known dichotomy result of Schaefer (see Subsection 8.6) states that under certain conditions there are precisely six classes of problem instances with polytime SAT solving (unless P=NP). However this has no bearing on the classes considered here, since they do not fall within the restricted framework of Schaefer's theorem.

extensions) clearly, we present a precise mathematical (non-algorithmic) definition, based on the transition relation $F \xrightarrow{\text{SLUR}} F'$ (Definition 4.1.3), which represents one non-deterministic step of the SLUR algorithm: If r_1 on input $F \in \mathcal{CLS}$ does not determine unsatisfiability (in which case we have $F \in \mathcal{SLUR}$), then $F \in \mathcal{SLUR}$ iff $\top$ can be reached by this transition relation, while everything else reachable from F is not an end-point of this transition relation.

In [36, 10] recently three approaches towards generalising $\mathcal{SLUR}$ have been considered, and we discuss them in Subsection 4.1.2. Our generalisation, called $\mathcal{SLUR}_k$, which we see as the natural completion of these approaches, will be presented in Section 4.4.

4.1.1 SLUR

The SLUR-algorithm ("Single Lookahead Unit Resolution") from [141] is described for input $F \in \mathcal{CLS}$ as follows (fully formalised in Definition 4.1.1 and Definition 4.1.3):

1. First run UCP, that is, reduce $F \rightsquigarrow F'$ where $F' = \mathrm{r}_1(F)$.
2. If now $\bot \in F'$ then we determined F unsatisfiable.
3. If not, then the algorithm guesses a satisfying assignment for F', by repeated transitions $F' \xrightarrow{\text{SLUR}} F''$, where F'' is obtained by assigning one variable and then performing UCP, i.e., $F'' := \mathrm{r}_1(\langle x \to 1\rangle * F')$ for some literal x.
4. The "lookahead" means that transitions with $F'' = \{\bot\}$ are not allowed.
5. The algorithm might find a satisfying assignment in this way, or it gets stuck, that is, for the chosen literal both assignments $x \to 1$ and $\overline{x} \to 1$ yield $\{\bot\}$, in which case it "gives up".
6. It is important to note - there is *no backtracking*, the algorithm is linear time and may return "satisfiable" or "unsatisfiable" (correctly), or instead it might "give up".

The SLUR class is defined as the class of clause-sets where this algorithm never gives up. The precise details are as follows. First we define the underlying transition relation (one non-failing transition from F to F'):

Definition 4.1.1 *For clause-sets $F, F' \in \mathcal{CLS}$ the relation $\boldsymbol{F \xrightarrow{SLUR} F'}$ holds if there is $x \in \mathrm{lit}(F)$ such that $F' = \mathrm{r}_1(\langle x \to 1\rangle * F)$ and $F' \neq \{\bot\}$. The transitive-reflexive closure is denoted by $\boldsymbol{F \xrightarrow{SLUR}_* F'}$.*

Example 4.1.2 *Considering when we have $F \xrightarrow{SLUR}_* F'$ and when not:*

1. *$F \xrightarrow{SLUR}_* \top$ iff $F \in \mathcal{SAT}$.*
2. *$\{C\} \xrightarrow{SLUR} \top$ precisely for all clauses $C \neq \bot$.*
3. *$\{\{x, y\}, \{x, \overline{y}\}\} \xrightarrow{SLUR} \top$.*
4. *$\{\{\overline{x}, y\}, \{\overline{y}, z\}\} \xrightarrow{SLUR} \top$ (due to e.g. $\mathrm{r}_1(\langle x \to 1\rangle * \{\{\overline{x}, y\}, \{\overline{y}, z\}\}) = \top$).*
5. *$F \xrightarrow{SLUR} F'$ does not hold if there is no literal to set, or if r_1 detects unsatisfiability of F'. That is, there are **no** clause-sets F, F' such that any of the following hold:*

(a) $\top \xrightarrow{SLUR} F$.

(b) $\{\bot\} \xrightarrow{SLUR} F$.

(c) $F \xrightarrow{SLUR} F$.

(d) $F \xrightarrow{SLUR} F'$ *where* $\mathrm{r}_1(F') = \{\bot\}$.

Via the transition-relation $F \xrightarrow{\mathrm{SLUR}} F'$ we can now easily define the class $\mathcal{SLUR}$, which will find a natural generalisation in Definition 4.4.1 to $\mathcal{SLUR}_k$ for $k \in \mathbb{N}_0$ (where $\mathcal{SLUR} = \mathcal{SLUR}_1$):

Definition 4.1.3 *The set of all fully reduced clause-sets reachable from $F \in \mathcal{CLS}$ is denoted by*

$$\mathbf{slur}(F) := \{F' \in \mathcal{CLS} \mid F \xrightarrow{SLUR}_* F' \wedge \neg\exists\, F'' \in \mathcal{CLS} : F' \xrightarrow{SLUR} F''\}.$$

Finally the class of all clause-sets which are either identified by UCP to be unsatisfiable, or where by SLUR-reduction always a satisfying assignment is found, is denoted by $\boldsymbol{\mathcal{SLUR}} := \{F \in \mathcal{CLS} : \mathrm{r}_1(F) \neq \{\bot\} \Rightarrow \mathrm{slur}(F) = \{\top\}\}$.

We could define $\xrightarrow{\mathrm{SLUR}}$ as $F \xrightarrow{\mathrm{SLUR}} \langle x \to 1\rangle * F$ iff $\mathrm{r}_1(\langle x \to 1\rangle * F) \neq \bot$, and this would yield the same class $\mathcal{SLUR}$ but a different transition relation (one would not be forced to immediately make forced assignments).

Example 4.1.4 *Computing* $\mathrm{slur}(F)$ *for clause-sets F:*

1. $\mathrm{slur}(F) \neq \emptyset$ *(in the "worst" case we have* $F \in \mathrm{slur}(F)$*).*
2. $\mathrm{slur}(\{\bot\}) = \{\{\bot\}\}$.
3. $\mathrm{slur}(\top) = \{\top\}$.
4. $\mathrm{slur}(\{C\}) = \{\top\}$ *iff* $C \neq \bot$.
5. *If* $\mathrm{r}_1(F) = \top$ *then* $\mathrm{slur}(F) = \{\top\}$.
6. $\mathrm{slur}(\{\{x, y\}, \{x, \overline{y}\}\}) = \{\top\}$.
7. $\mathrm{slur}(\{\{\overline{x}, y\}, \{\overline{y}, z\}\}) = \{\top\}$.
8. *For* $F := \{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\}$ *we have* $\mathrm{slur}(F) = \{F\}$.
9. *For* $F' := \{\{z, x, y\}, \{z, x, \overline{y}\}, \{z, \overline{x}, y\}, \{z, \overline{x}, \overline{y}\}\}$ *we have* $\top, F \in \mathrm{slur}(F')$.

4.1.2 Previous approaches for SLUR hierarchies

In [36, 10] three hierarchies $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ $(k \in \mathbb{N})$ and $\mathrm{CANON}(k)$ $(k \in \mathbb{N}_0)$ have been introduced. In Section 4 of [10] it is shown that $\mathcal{SLUR}(k) \subset \mathcal{SLUR}*(k)$ for all $k \in \mathbb{N}$ and so we restrict our attention to $\mathcal{SLUR}*(k)$ and $\mathrm{CANON}(k)$.

$\mathrm{CANON}(k)$ is defined to be the set of clause-sets F such that every $C \in \mathrm{prc}_0(F)$ can be derived from F by a resolution tree of height at most k. Note that basically by definition (using stability of resolution proofs under application of partial assignments) we get that each $\mathrm{CANON}(k)$ is stable under application of partial assignments and under variable-disjoint union.

The $\mathcal{SLUR}*(k)$ hierarchy is derived in [10] from the $\mathcal{SLUR}$ class by extending the reduction r_1. We provide an alternative formalisation here, in the same manner as in Section 4.1.1. The

main question is the transition relation $F \rightsquigarrow F'$. The $\mathcal{SLUR}*(k)$-hierarchy provides stronger and stronger witnesses that F' might be satisfiable, by longer and longer assignments (making "k decisions") not yielding the empty clause:

Definition 4.1.5 *That partial assignment $\varphi \in \mathcal{PASS}$ **makes k decisions** for some $k \in \mathbb{N}_0$ w.r.t. $F \in \mathcal{CLS}$ is defined recursively as follows: For $k = 0$ this relation holds if $\varphi * F = \mathrm{r}_1(F)$, while for $k > 0$ this relation holds if either there is $k' < k$ such that φ makes k' decision w.r.t. F and $\varphi * F = \top$, or there exists $x \in \mathrm{lit}(F)$ and a partial assignment φ' making $k-1$ decision for $\mathrm{r}_1(\langle x \to 1\rangle * F)$, and where $\varphi * F = \varphi' * \mathrm{r}_1(\langle x \to 1\rangle * F)$.*

Now $\boldsymbol{F \xrightarrow{SLUR*k} F'}$ for $k \geq 1$ by definition holds if there is a partial assignment φ making k decision w.r.t. F with $F' = \varphi * F$, where $F' \neq \{\bot\}$. The reflexive-transitive closure is $\xrightarrow{SLUR*k}_*$.

Finally we can define the hierarchy:

$$\begin{aligned} \mathbf{slur}*(\boldsymbol{k})(\boldsymbol{F}) &:= \{F' \in \mathcal{CLS} \mid F \xrightarrow{\mathrm{SLUR}*k}_* F' \wedge \neg\exists\, F'' : F' \xrightarrow{\mathrm{SLUR}*k} F''\} \\ \boldsymbol{\mathcal{SLUR}}*(\boldsymbol{k}) &:= \{F \in \mathcal{CLS} : \mathrm{slur}*(k)(F) \neq \{F\} \Rightarrow \mathrm{slur}*(k)(F) = \{\top\}\}. \end{aligned}$$

The unsatisfiable elements of $\mathcal{SLUR}*(k)$ are those $F \neq \top$ with $\mathrm{slur}*(k)(F) = \{F\}$. By definition each $\mathcal{SLUR}*(k)$ is stable under application of partial assignments, but not stable under variable-disjoint union, since the number of decision variables is bounded by k (in Lemma 4.3.4 we will see that our hierarchy is stable under variable-disjoint union, which is natural since it strengthens the CANON(k)-hierarchy).

Example 4.1.6 *Some examples for* CANON(k) *and $\mathcal{SLUR}*(k)$ ($k \in \mathbb{N}$):*

1. *Consider the unsatisfiable clause-set $F := \{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\}$.*
   (a) *$F \notin \mathcal{SLUR}$ because F is unsatisfiable but $\mathrm{r}_1(F) \neq \{\bot\}$.*
   (b) *$F \in \mathcal{SLUR}*(1)$ because $\mathrm{r}_1(\langle x' \to 1\rangle * F) = \{\bot\}$ for all $x' \in \mathrm{lit}(F)$ and so $\mathrm{slur}*(1)(F) = \{F\}$.*
   (c) *This establishes $\mathcal{SLUR} \subset \mathcal{SLUR}*(1)$.*
   (d) *$F \in \mathrm{CANON}(2) \setminus \mathrm{CANON}(1)$ because actually all tree-resolution refutations of F are full binary trees of height 2.*
2. *Consider the satisfiable clause-set $F' := \{\{x_1, \ldots, x_k\} \cup C \mid C \in F\}$.*
   (a) *$F' \notin \mathcal{SLUR}*(k)$ because $F' \xrightarrow{SLUR*k}_* F$, where F is unsatisfiable and thus $\neg(F \xrightarrow{SLUR*k}_* \top)$, whence $\mathrm{slur}*(k)(F') \neq \{\top\}$.*
   (b) *$F' \in \mathcal{SLUR}*(k+1)$ because we have $r_1(\varphi * F') \in \{\top, \{\bot\}\}$ for all partial assignments φ of length $k+1$ on variables of F' hence $\mathrm{slur}*(k)(F_1) = \{\top\}$.*
   (c) *$F' \in \mathrm{CANON}(2)$ because the only prime implicate is $\{x_1, \ldots, x_k\}$ and actually all its tree-resolution proofs are full binary trees of height 2.*

4.2 $\mathcal{UC}_k$: unit-refutation complete clause-sets at level-k

The second source for our investigations is the class $\mathcal{UC}$ of *unit-refutation complete clause-sets*, introduced in [51] as a target class for knowledge compilation. Via the theory of "hardness" of clause-sets as developed in [104, 110, 2] we obtain a natural generalisation $\mathcal{UC}_k$, containing those clause-sets which are "unit-refutation complete of level k", which is the same as having hardness at most k.

Definition 4.2.1 *For $k \in \mathbb{N}_0$ let $\mathcal{UC}_k := \{F \in \mathcal{CLS} : \mathrm{hd}(F) \le k\}$ (the class of **unit-refutation complete clause-sets of level** k).* [2)]

The class $\mathcal{UC}_1$ has been introduced in [51] for knowledge compilation. Various (resolution-based) algorithms computing for clause-sets F some equivalent set $F' \in \mathcal{UC}_1$ of prime implicates are discussed there. Based on the results from [104, 110] restated in Theorem 4.2.2, a powerful proof-theoretic characterisation for all classes $\mathcal{UC}_k$ can be given:

Theorem 4.2.2 *For $k \in \mathbb{N}_0$ and $F \in \mathcal{CLS}$ we have*

$$F \in \mathcal{UC}_k \iff \forall C \in \mathrm{prc}_0(F) : F \vdash_k C.$$

Thus if every $C \in \mathrm{prc}_0(F)$ has a tree-resolution refutation using at most $2^{k+1} - 1$ leaves (i.e., $\mathrm{Comp}^*_{\mathrm{R}}(\varphi_C * F) < 2^{k+1}$), then $\mathrm{hd}(F) \le k$.

A precursor A generalisation of $\mathcal{UC}$ was already discussed in [52]. Assuming a polytime SAT-decision algorithm $P : C \to \{0,1\}$ for some $C \subseteq \mathcal{CLS}$, the class $\mathcal{PC} \subseteq \mathcal{CLS}$ of "P-complete" clause-sets is defined as the set of $F \in \mathcal{CLS}$ such that for all implicates C holds $P(\varphi_C * F) = 0$.[3)] This is an obvious generalisation of $\mathcal{UC}_k$, when using $\mathcal{C}_k := \{F \in \mathcal{CLS} : \mathrm{r}_k(F) \in \{\top, \{\bot\}\}\}$ and $P_k : \mathcal{C}_k \to \{0,1\}$ with $P_k(F) = 1 \Leftrightarrow \mathrm{r}_k(F) = \top$. But it does not cover the hierarchies $\mathcal{PC}_k$ or $\mathcal{WC}_k$, which are based on different principles.[4)] We note the conceptual weakness of demanding a SAT-decision algorithm P, where actually only a means for detecting *unsatisfiability* is needed.

[52] continues by considering the (generic) hierarchy $(\Pi_k)_{k \in \mathbb{N}_0}$ from [132], a precursor of [104]. $\Pi_0 \subseteq \mathcal{CLS}$ in principal is arbitrary, but is assumed to be polytime decidable and SAT-decidable. Then Π_k for $k > 0$ is the set of $F \in \mathcal{CLS}$ such that $F \in \Pi_{k-1}$ or there is a literal $x \in \mathrm{lit}(F)$ with $\langle x \to 1\rangle * F \in \Pi_{k-1}$ and $\langle x \to 1\rangle * F \in \Pi_k$. We note that if we choose $\Pi_0 = \{F \in \mathcal{CLS} : \bot \in F\}$, then $\Pi_k = \mathcal{UC}_k \cap \mathcal{USAT}$ for all $k \ge 0$. However this choice for Π_0 was never considered for that hierarchy from [132], which might have two reasons: Implicit preference is given to classes Π_0 closed under sub-clause-set formation (see Section 6.3 in [78] for more discussions on this issue). And furthermore SAT and UNSAT is not distinguished in [132] and in subsequent work directly relying on it; see Subsection 1.2 in [104] for a discussion of this. So the four choices for Π_0 considered in [52] are $\mathcal{HO}$, 2-$\mathcal{CLS}$, $\mathcal{RHO}$ and $\mathcal{QHO}$. Accordingly $\mathcal{UC}_0 \cap \mathcal{USAT}$ is not contained at any Π_k, and so not even r_1 on unsatisfiable clause-set is covered by the considered hierarchies.

Due to these weaknesses, [52] does not consider a hierarchy generalising $\mathcal{UC}$. From our point of view one could say, that Π_k is only considered as a resource for polytime recognition of certain instances for $\mathcal{UC}_k$ resp. $\mathcal{UC}_{k+1}$; compare Subsections 6.2, 6.3 in [78] for results in this direction.

4.3 Fundamental properties of $\mathcal{UC}_k$

In Subsection 3.4.1 we determined hardness for various constructions. In Subsection 4.3.1 we consider various classes contained in some $\mathcal{UC}_k$ together with stability properties of $\mathcal{UC}_k$. Relations to alternative hierarchies from the literature are discussed in Subsection 4.3.2. We conclude

[2)] To elaborate, $\mathcal{UC}_k$ is the class of clause-sets F which have that for all partial assignments $\varphi \in \mathcal{PASS}$ if $\varphi * F \in \mathcal{USAT}$ then $\mathrm{r}_k(F) = \{\bot\}$. That is, under any falsifying assignment, r_k is sufficient to determine unsatisfiability.

[3)] [52] actually favours adding unit-clauses to F, but we consider applying partial assignments as more fundamental.

[4)] This is obvious for $k \ge 1$ and $\mathcal{PC}_k$, since $\mathcal{PC}_k \cap \mathcal{USAT} = \mathcal{UC}_k \cap \mathcal{USAT}$, while $\mathcal{PC}_k \cap \mathcal{SAT} \subset \mathcal{UC}_k \cap \mathcal{SAT}$. We conjecture that for $k \ge 3$ there is no (polytime) P with $\mathcal{PC} = \mathcal{WC}_k$ (as remarked in Section 3.6.2, for $k \in \{0,1,2\}$ there exists such a P).

our discussion of basic properties of hardness in Subsection 4.3.3, considering the most basic cases of precise hardness-computations. We stress that (algorithmic) computation of hardness for arbitrary instances is less important here[5], since we aim more at constructing "soft" (low hardness) representations than measuring hardness of given instances. What is needed is a theory to identify general constructions.

4.3.1 Containment and stability properties

The following fundamental lemma is obvious from the definition:

Lemma 4.3.1 *Consider $\mathcal{C} \subseteq \mathcal{CLS}$ stable under application of partial assignments and $k \in \mathbb{N}_0$. If $\mathcal{C} \cap \mathcal{USAT} \subseteq \mathcal{UC}_k$ then $\mathcal{C} \subseteq \mathcal{UC}_k$.*

We apply Lemma 4.3.1 to various well-known classes $\mathcal{C}$ (stating in brackets the source for the bound on the unsatisfiable cases).

Lemma 4.3.2 *Consider $F \in \mathcal{CLS}$.*

1. *For $\varphi \in \mathcal{PASS}$ and all $k \in \mathbb{N}_0$ we have that $F \in \mathcal{UC}_k$ implies $\varphi * F \in \mathcal{UC}_k$ (by Lemma 3.11 in [104]).*
2. *$F \in \mathcal{UC}_{n(F)}$ (by Lemma 3.18 in [104]).*
3. *$2\text{-}\mathcal{CLS} \subseteq \mathcal{UC}_2$ (by Lemma 5.6 in [104]).*[6]
4. *$\mathcal{HO} \subseteq \mathcal{UC}_1$ (by Lemma 5.8 in [104]).*
5. *$\mathcal{QHO} \subseteq \mathcal{UC}_2$ (by Lemma 5.12 in [104]; see Section 6.10.2 in [42], and [158]).*
6. *For all $k \in \mathbb{N}$ we have $\mathcal{HO}_k \subseteq \mathcal{UC}_k$ (by Lemma 5.10 in [104]; see [99]).*

Obviously Part 4 of Lemma 4.3.2 can be generalised to $F \in \mathcal{RHO}$ (see Lemma 4.3.4, Part 3). And considering Part 3, by a standard autarky-argument for $2\text{-}\mathcal{CLS}$ (see [100]) we can sharpen the hardness-upper-bound 2 for *satisfiable* clause-sets:

Lemma 4.3.3 *We have that $2\text{-}\mathcal{CLS} \cap \mathcal{SAT} \subseteq \mathcal{UC}_1$.*

Proof: Consider a partial assignment φ with unsatisfiable $\varphi * F$. Now we have $\mathrm{r}_1(\varphi * F) = \{\bot\}$, since otherwise $\mathrm{r}_1(\varphi * F) \subseteq F$ (all $C \in F$ must have $|C| = 2$ and r_1 doesn't introduce new 2-clauses), and thus $\mathrm{r}_1(\varphi * F)$ would be satisfiable. □

[5] Decision of membership in $\mathcal{UC}_k$ for $k \geq 1$ is coNP-complete, as shown in Theorem 4.4.5, which seems natural for classes with strong expressive power (for example, we see less expressive classes such as $\mathcal{HO}, \mathcal{HO}_k$ with polytime decidability are able to capture fewer boolean functions (at fixed levels), or simply collapsing as in Lemma 8.1.4). A core implication of this is that hd will sometimes be impractical as a *measure*, which is precisely why one instead considers $\mathcal{UC}_k$ as target classes *by construction*.

[6] To see that $\mathrm{hd}(F) \leq 2$ for $F \in 2\text{-}\mathcal{CLS} \cap \mathcal{USAT}$, observe that if $\mathrm{r}_2(F) = F' \neq \{\bot\}$ then $F' \in 2\text{-}\mathcal{CLS} \cap \mathcal{USAT}$ with all $C \in F$ having $|C| = 2$ (otherwise $\mathrm{r}_2(F) = \mathrm{r}_2(F') \neq F'$). Consider a minimally unsatisfiable subset $F'' \subseteq F'$ and a variable $x \in \mathrm{occlit}(F'')$. If $\mathrm{r}_1(\langle x \to 0\rangle * F'') = \{\bot\}$ then $\mathrm{r}_2(F) = \mathrm{r}_2(F') \neq F'$ (a contradiction), otherwise $\mathrm{r}_1(\langle x \to 0\rangle * F'') \subseteq F''$ is still unsatisfiable, contradicting the minimal unsatisfiability of F''.

We have the following stability properties:

Lemma 4.3.4 *Consider $k \in \mathbb{N}_0$.*

1. *$\mathcal{UC}_k$ is stable under application of partial assignments (with Lemma 4.3.2, Part 1; this might reduce hardness).*
2. *$\mathcal{UC}_k$ is stable under variable-disjoint union (with Lemma 3.4.6).*
3. *$\mathcal{UC}_k$ is stable under renaming variables and switching polarities (by definition).*
4. *$\mathcal{UC}_k$ is stable under subsumption-elimination (by basic properties of resolution).*
5. *$\mathcal{UC}_k$ is stable under addition of inferred clauses (by definition; this might reduce hardness).*

Example 4.3.5 *Examples for non-stability:*

1. *$\mathcal{UC}_0$ is obviously not stable under removal of clauses.*
2. *$\mathcal{UC}_0$ is not stable under removal of literal occurrences, for example $\{\{x,y\},\{\overline{x},\overline{y}\}\} \in \mathcal{UC}_0$, but $\{\{x\},\{\overline{x},\overline{y}\}\} \notin \mathcal{UC}_0$.*
3. *$\mathcal{UC}_0$ is not stable under crossing out of variables, e.g. $\{\{x,y\},\{\overline{x},\overline{y}\}\} \in \mathcal{UC}_0$, but when crossing out variable x we obtain $\{\{y\},\{\overline{y}\}\} \notin \mathcal{UC}_0$.*
4. *$\mathcal{UC}_0$ is not stable under addition of clauses, for example $\{\{x\}\} \in \mathcal{UC}_0$, but $\{\{x\},\{\overline{x}\}\} \notin \mathcal{UC}_0$.*
5. *$\mathcal{UC}_0$ is not stable under addition of literal occurrences, e.g. $\{\{x\},\{y\}\} \in \mathcal{UC}_0$, but $\{\{x,\overline{y}\},\{y\}\} \notin \mathcal{UC}_0$.*

4.3.2 Alternative hierarchies

No class $\mathcal{UC}_k$ is stable under removal of clauses. We will see in this subsection that this boils down to the class $\mathcal{U}_0$ of clause-sets containing the empty clauses not being stable under removal of clauses. Some classes contained in $\mathcal{UC}_1$ however are stable under removal of clauses, for examples renamable Horn clause-sets ($\mathcal{RHO}$), and in [35] hierarchies based on this more restricted class have been considered. To understand the connection to our approach, some comments on the use of "oracles" in this setting are needed (see Chapter 8 for future developments).

In [104, 110] the hierarchy $G_k(\mathcal{U},\mathcal{S}) \subseteq \mathcal{CLS}$ ($k \in \mathbb{N}_0$) has been introduced, using oracles $\mathcal{U} \subseteq \mathcal{USAT}$ for unsatisfiability detection and $\mathcal{S} \subseteq \mathcal{SAT}$ for satisfiability detection:

Definition 4.3.6 *For all $k \in \mathbb{N}_0$ and classes $\mathcal{U},\mathcal{S} \subseteq \mathcal{CLS}$, the class $G_k(\mathcal{U},\mathcal{S})$ is defined as follows. Let*

$$G_0^0(\mathcal{U},\mathcal{S}) = \mathcal{U} \text{ and } G_0^1(\mathcal{U},\mathcal{S}) = \mathcal{S}.$$

While

- *$F \in G_{k+1}^0(\mathcal{U},\mathcal{S})$ if either $F = \{\bot\}$ or there is some $(v,\varepsilon) \in \mathrm{var}(F) \times \{0,1\}$ such that*

$$\langle v \to e\rangle * F \in G_k^0(\mathcal{U},\mathcal{S}) \qquad \text{and} \qquad \langle v \to \overline{e}\rangle * F \in G_{k+1}^0(\mathcal{U},\mathcal{S}).$$

- $F \in \boldsymbol{G^1_{k+1}(\mathcal{U},\mathcal{S})}$ *if either* $F = \top$ *or there is some* $(v,\varepsilon) \in \mathrm{var}(F) \times \{0,1\}$ *such that*

$$\langle v \to e\rangle * F \in G^1_k(\mathcal{U},\mathcal{S}) \quad \text{or}$$
$$\langle v \to e\rangle * F \in G^0_k(\mathcal{U},\mathcal{S}) \quad \text{and} \quad \langle v \to \overline{e}\rangle * F \in G^1_{k+1}(\mathcal{U},\mathcal{S}).$$

Finally, $\boldsymbol{G_k(\mathcal{U},\mathcal{S})} := G^0_k(\mathcal{U},\mathcal{S}) \cup G^1_k(\mathcal{U},\mathcal{S})$.

1. The minimal oracles considered there are $\mathcal{U}_0 := \{F \in \mathcal{CLS} : \bot \in F\}$ and $\mathcal{S}_0 := \{\top\}$.
2. One uses $G^0_k(\mathcal{U},\mathcal{S}) := G_k(\mathcal{U},\mathcal{S}) \cap \mathcal{USAT}$ and $G^1_k(\mathcal{U},\mathcal{S}) := G_k(\mathcal{U},\mathcal{S}) \cap \mathcal{SAT}$. Since $G^0_k(\mathcal{U},\mathcal{S})$ does not depend on $\mathcal{S}$, one writes $G^0_k(\mathcal{U}) := G^0_k(\mathcal{U},\mathcal{S})$.
3. For all $k \in \mathbb{N}_0$ holds $G^0_k(\mathcal{U}_0) = \mathcal{UC}_k \cap \mathcal{USAT}$. On satisfiable instances in general the hierarchies are incomparable.
4. If $\mathcal{C} \subseteq \mathcal{CLS}$ is stable under application of partial assignments, then each class $G_k(\mathcal{C}) := G_k(\mathcal{C} \cap \mathcal{USAT}, \mathcal{C} \cap \mathcal{SAT})$ (for $k \in \mathbb{N}_0$) is also stable under partial assignments (Lemma 4.2 in [110]). So if $\mathcal{C} \cap \mathcal{USAT} \subseteq \mathcal{UC}_{k'}$ for some $k' \in \mathbb{N}_0$, then we have $G_k(\mathcal{C}) \subseteq \mathcal{UC}_{k+k'}$ (using Lemma 4.3.1). This is the basis of all inclusion-relations of Section 4.3.
5. In [104, 110] it is assumed that $\mathcal{U}_0 \subseteq \mathcal{U}$ holds. This ensures that $\mathcal{UC}_k \cap \mathcal{USAT} \subseteq G^0_k(\mathcal{C})$ always holds, but in most cases makes classes $G_k(\mathcal{U},\mathcal{S})$ unstable under elimination of clauses.

In [35] two hierarchies $(\Pi_k)_{k\in\mathbb{N}_0}$, $(\Upsilon_k)_{k\in\mathbb{N}_0}$ have been introduced; the basic motivations and the relations to our hierarchies are as follows:

Definition 4.3.7 *For all* $\mathcal{C} \subseteq \mathcal{CLS}$ *and all* $k \in \mathbb{N}_0$ *consider the class* $G'_k(\mathcal{C})$ *defined as follows*

- $G'_0(F) := \mathcal{C}$.
- *for all clause-sets* $F \in \mathcal{CLS}$ *we have* $F \in G'_{k+1}(\mathcal{C})$ *iff* $F \in \{\top, \{\bot\}\}$ *or there exists* $(v,\varepsilon) \in \mathrm{var}(F) \times \{0,1\}$ *such that* $\langle v \to \varepsilon\rangle * F \in G'_k(\mathcal{C})$ *and* $\langle v \to \overline{\varepsilon}\rangle * F \in G'_{k+1}(\mathcal{C})$.

The hierarchies $\boldsymbol{(\Pi_k)_{k\in\mathbb{N}_0}}$ *and* $\boldsymbol{(\Upsilon_k)_{k\in\mathbb{N}_0}}$ *are defined as* $(G'_k(\mathcal{RHO}))_{k\in\mathbb{N}_0}$, $(G'_k(\mathcal{QHO}))_{k\in\mathbb{N}_0}$ *respectively.*

1. We have $\Pi_k \cap \mathcal{USAT} = G^0_k(\mathcal{RHO})$ and $\Pi_k \cap \mathcal{SAT} \subseteq G^1_k(\mathcal{RHO})$ (with $\Pi_0 = \mathcal{RHO}$). Note that we do not have $\mathcal{U}_0 \subseteq \mathcal{RHO}$ here.
2. It is $\mathcal{RHO} \cap \mathcal{USAT} \subset G^0_1(\mathcal{U}_0)$ (Lemma 4.3.2, Part 4), while $\mathcal{RHO} \cap \mathcal{SAT}$ is not included in any $G^1_k(\mathcal{U},\mathcal{S}_0)$. More generally we have $\Pi_k \cap \mathcal{USAT} \subset G^0_{k+1}(\mathcal{U}_0)$ for all $k \geq 0$.
3. So the choice of the oracle $\mathcal{RHO}$ is less powerful on unsatisfiable instances than the choice of $\mathcal{U}_0$ (when going up one level in the hierarchy), while the special recognition of satisfiability for $\mathcal{RHO}$ is (naturally) not captured by any level of the G_k-hierarchy, when using only the trivial satisfiability-oracle $\mathcal{S}_0$ (even using $\mathcal{U} = \mathcal{USAT}$ does not change this, since this only yields full handling of all *forced* assignments, while a satisfiable instance in $\mathcal{RHO}$ might not have any forced assignment).
4. For $k \geq 1$ we have $\Pi_k \cap \mathcal{SAT} \subset G^1_k(\mathcal{RHO})$, where an example for $F \in G^1_k(\mathcal{RHO}) \setminus \Pi_k$ is given by $F := \{\{v\} \cup C : C \in F'\}$ for some $F' \in \mathcal{CLS} \setminus \Pi_k$ and $v \in \mathcal{VA} \setminus \mathrm{var}(F')$. The point is that recognition for the $G_k(\mathcal{U},\mathcal{S})$-hierarchy includes satisfiability-decision at lower levels, and if one branch, here $\langle v \to 1\rangle$, yields a satisfiable instance, then the other branch ($\langle v \to 0\rangle$) is not inspected — which however is the case for Π_k.

5. $\mathcal{RHO}$ is stable under application of partial assignments, and, that is its main feature, stable under removal of clauses. This yields that all Π_k are stable under removal of clauses, which is the main motivation for this choice of the base oracle.

6. $\mathcal{U}_0$ is not contained in any Π_k, and thus there are unsatisfiable clause-sets of hardness 0 not contained in any given Π_k.

7. [35] considered also (shortly) the hierarchy $\Upsilon_k \subset \mathcal{CLS}$ ($k \in \mathbb{N}_0$), with $\Upsilon_k \cap \mathcal{USAT} = G_k^0(\mathcal{QHO})$ and $\Upsilon_k \cap \mathcal{SAT} \subseteq G_k^1(\mathcal{QHO})$, based on the stronger oracle $\mathcal{QHO} \supset \mathcal{RHO}$ of q-Horn clause-sets (again stable under application of partial assignments and removal of clauses). We have $\Upsilon_k \cap \mathcal{USAT} \subset G_{k+2}^0(\mathcal{U}_0)$ for all $k \geq 0$ (Lemma 4.3.2, Part 5).

By Lemma 4.3.1 we get:

Lemma 4.3.8 *For all $k \in \mathbb{N}_0$ we have $\Pi_k \subset \mathcal{UC}_{k+1}$ and $\Upsilon_k \subset \mathcal{UC}_{k+2}$ for the hierarchies Π_k, Υ_k introduced in [35].*

4.3.3 Determining hardness computationally

By the well-known computation of $\mathrm{prc}_0(F)$ via resolution-closure we obtain:

Lemma 4.3.9 *Whether for $F \in \mathcal{CLS}$ we have $\mathrm{hd}(F) = 0$ or not can be decided in polynomial time, namely $\mathrm{hd}(F) = 0$ holds if and only if F is stable under resolution modulo subsumption (which means that for all resolvable $C, D \in F$ with resolvent R there exists $E \in F$ with $E \subseteq R$).*

Thus if the hardness is known to be at most 1, we can compute it efficiently:

Corollary 4.3.10 *Consider a class $\mathcal{C} \subseteq \mathcal{CLS}$ of clause-sets where $\mathcal{C} \subseteq \mathcal{UC}_1$ is known. Then for $F \in \mathcal{C}$ one can compute $\mathrm{hd}(F) \in \{0,1\}$ in polynomial time.*

Examples for $\mathcal{C}$ are given by $\mathcal{HO} \subset \mathcal{UC}_1$ (Lemma 4.3.2) and in Subsection 4.1.1. Another example class with known hardness is given by 2-$\mathcal{CLS} \subset \mathcal{UC}_2$ (Lemma 4.3.2), and also here we can compute the hardness efficiently:

Lemma 4.3.11 *For $F \in$ 2-$\mathcal{CLS}$ one can compute $\mathrm{hd}(F) \in \{0,1,2\}$ in polynomial time.*

Proof: One method is to observe that for elements of 2-$\mathcal{CLS}$ the set of prime-implicates can be determined in polynomial time, while SAT-decision can be done in linear time. More efficient is the following:

1. Determine first whether F is satisfiable or not.

2. If F is satisfiable, then $\mathrm{hd}(F) \in \{0,1\}$ by Lemma 4.3.3, and whether $\mathrm{hd}(F) = 0$ or not can be determined by Lemma 4.3.9.

3. If F is unsatisfiable, then it suffices to compute $\mathrm{r}_0(F)$ and $\mathrm{r}_1(F)$. □

See Theorem 4.4.5 for coNP-completeness of determining an upper bound on hardness.

4.4 The SLUR hierarchy

We now define the $\mathcal{SLUR}_k$ hierarchy, generalising $\mathcal{SLUR}$ (recall Subsection 4.1.1) in a natural way, by replacing r_1 with r_k. In Subsection 4.4.1 we show $\mathcal{SLUR}_k = \mathcal{UC}_k$, and as application obtain coNP-completeness of membership decision for $\mathcal{UC}_k$ for $k \geq 1$. In Section 4.4.2 we determine the relations to the previous hierarchies $\mathcal{SLUR}*(k)$ and CANON(k) as discussed in Subsection 4.1.2.

Definition 4.4.1 *Consider $k \in \mathbb{N}_0$. For clause-sets $F, F' \in \mathcal{CLS}$ the relation $\boldsymbol{F \xrightarrow{SLUR_k} F'}$ holds if there is $x \in \mathrm{lit}(F)$ such that $F' = \mathrm{r}_k(\langle x \to 1\rangle * F)$ and $F' \neq \{\bot\}$. The transitive-reflexive closure is denoted by $\boldsymbol{F \xrightarrow{SLUR_k}_* F'}$. The set of all fully reduced clause-sets reachable from F is denoted by*

$$\mathbf{slur_k(F)} := \{F' \in \mathcal{CLS} \mid F \xrightarrow{SLUR_k}_* F' \wedge \neg\exists\, F'' \in \mathcal{CLS} : F' \xrightarrow{SLUR_k} F''\}.$$

Finally the class of all clause-sets which are either identified by r_k to be unsatisfiable, or where by k-SLUR-reduction always a satisfying assignment is found, is denoted by $\boldsymbol{\mathcal{SLUR}_k} := \{F \in \mathcal{CLS} : \mathrm{r}_k(F) \neq \{\bot\} \Rightarrow \mathrm{slur}_k(F) = \{\top\}\}$.

We have $\mathcal{SLUR}_1 = \mathcal{SLUR}$ (recall Definition 4.1.3). Note also the following simple properties for $F \in \mathcal{CLS}$:

1. $\top \in \mathrm{slur}_k(F) \Leftrightarrow F \in \mathcal{SAT}$.
2. For $F' \in \mathrm{slur}_k(F) \setminus \{\top\}$ we have $F' \in \mathcal{USAT}$, and if $F \in \mathcal{SAT}$, then $\mathrm{r}_k(F') \neq \{\bot\}$.
3. If $F \in \mathcal{SLUR}_k$, then $F \in \mathcal{SAT}$ and $F \xrightarrow{\mathrm{SLUR}_k}_* F'$ implies $F' \in \mathcal{SAT}$. Note in this case that $F' \in \mathrm{slur}_k(F)$ implies $F' = \top$ and $F \notin \mathrm{slur}_k(F)$ implies $F' \xrightarrow{\mathrm{SLUR}_k}_* \top$.

Again we could define the transition relation in a less restricted way, as $F \xrightarrow{\mathrm{SLUR}_k} \langle x \to 1\rangle * F$ iff $\mathrm{r}_k(\langle x \to 1\rangle * F) \neq \bot$, and this would yield the same class $\mathcal{SLUR}_k$.

Example 4.4.2 *Some examples for $\mathcal{SLUR}_2 \setminus \mathcal{SLUR}_1$:*

1. *Consider the unsatisfiable clause-set $F := \{\{x, y\}, \{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{x}, \overline{y}\}\}$.*
   (a) *$F \notin \mathcal{SLUR}_1$ because F is unsatisfiable but $\mathrm{r}_1(F) \neq \{\bot\}$.*
   (b) *$F \in \mathcal{SLUR}_2$ because $\mathrm{r}_2(F) = \{\bot\}$.*
2. *Consider the satisfiable clause-set $F' := \{\{x_1, x_2\} \cup C \mid C \in F\}$.*
   (a) *$F' \notin \mathcal{SLUR}_1 = \mathcal{SLUR}$ because $F' \xrightarrow{SLUR}_* F = \langle x_1, x_2 \to 0\rangle * F'$, where $\mathrm{slur}(F) = \{F\}$ and so $F \in \mathrm{slur}(F')$.*
   (b) *$F' \in \mathcal{SLUR}_2$ because for any φ such that $F' \xrightarrow{SLUR_2}_* \varphi * F'$ and $F' \neq \top$ we have one of the following two cases:*
      i. *$\varphi * F'$ is satisfiable, and so $\varphi * F' \notin \mathrm{slur}_2(F)$.*
      ii. *$\varphi * F'$ is unsatisfiable and so $\langle x_1 \to 0, x_2 \to 0\rangle \subseteq \varphi$, but this contradicts the fact that $F' \xrightarrow{SLUR_2}_* \varphi * F'$. That is, after setting either x_1 or x_2 to 0, lookahead with r_2 detects unsatisfiability of $\varphi * F'$ and so one can never transition to $\varphi * F'$ from F'.*

Therefore $\mathrm{slur}_2(F') = \{\top\}$.

More generally we have $\{\{x_1, \ldots, x_k\} \cup C \mid C \in F\} \in \mathcal{SLUR}_2 \setminus \mathcal{SLUR}*(k)$ *(recall Example 4.1.6).*

Lemma 4.4.3 *We have for* $F \in \mathcal{CLS}$, $k \in \mathbb{N}_0$ *and a partial assignment* φ *with* $\mathrm{r}_k(\varphi * F) \neq \{\bot\}$ *that* $F \xrightarrow{SLUR_k}_* \mathrm{r}_k(\varphi * F)$ *holds.*

Proof: The assignments of φ can be performed via SLUR-k-transitions. □

4.4.1 SLUR = UC

For $F \in \mathcal{UC}_k$ there is the following polynomial-time SAT decision: F is unsatisfiable iff $\mathrm{r}_k(F) = \{\bot\}$. And a satisfying assignment can be found for satisfiable F via self-reduction, that is, probing variables, where unsatisfiability again is checked for by means of r_k. For $k = 1$ this means exactly that the nondeterministic "SLUR"-algorithm will not fail. And that implies that $F \in \mathcal{SLUR}$ holds, where $\mathcal{SLUR}$ is the class of clause-sets where that algorithm never fails. So $\mathcal{UC}_1 \subseteq \mathcal{SLUR}$. Now it turns out, that actually this property characterises $\mathcal{UC}_1$, that is, $\mathcal{UC}_1 = \mathcal{SLUR}$ holds, which makes available the results on $\mathcal{SLUR}$.

We now show that this equality between $\mathcal{UC}$ and $\mathcal{SLUR}$ holds in full generality for the $\mathcal{UC}_k$ and $\mathcal{SLUR}_k$ hierarchies.

Theorem 4.4.4 *For all* $k \in \mathbb{N}_0$ *holds* $\mathcal{SLUR}_k = \mathcal{UC}_k$.

Proof: Consider $F \in \mathcal{CLS}$. We have to show $F \in \mathcal{SLUR}_k \Leftrightarrow \mathrm{hd}(F) \leq k$. For $F \in \mathcal{USAT}$ this follows from the definitions, and thus we assume $F \in \mathcal{SAT}$.

First consider $F \in \mathcal{SLUR}_k$. Consider a partial assignment φ such that $\varphi * F \in \mathcal{USAT}$. We have to show $\mathrm{r}_k(\varphi * F) = \{\bot\}$, and so assume $\mathrm{r}_k(\varphi * F) \neq \{\bot\}$. It follows $F \xrightarrow{\mathrm{SLUR}_k}_* \mathrm{r}_k(\varphi * F)$ by Lemma 4.4.3, whence $\mathrm{r}_k(\varphi * F) \in \mathcal{SAT}$ contradicting $\varphi * F \in \mathcal{USAT}$.

Now assume $\mathrm{hd}(F) \leq k$, and we show $F \in \mathcal{SLUR}_k$, i.e., $\mathrm{slur}_k(F) = \top$. Assume there is $F' \in \mathrm{slur}_k(F) \setminus \{\top\}$. By Property 2 for Definition 4.4.1 we get $F' \in \mathcal{USAT}$ and $\mathrm{r}_k(F') \neq \{\bot\}$. However by Lemma 4.3.2, Part 1 we get $\mathrm{hd}(F') \leq k$, and thus $\mathrm{r}_k(F') = \{\bot\}$. □

It seemed an essential feature of the class $\mathcal{SLUR}$, that its most natural definition is by the SLUR-algorithm; for example in [64] we find the quote "I find it interesting that the algorithm seems simpler than the conditions under which it is a decision procedure." By Theorem 4.4.4 now we have a simple characterisation of these conditions, namely that unsatisfiability after instantiation is always detected by unit-clause propagation. Using the characterisation $\mathcal{SLUR} = \mathcal{UC}$, we can show coNP-completeness of hardness-determination:

Theorem 4.4.5 *For fixed* $k \in \mathbb{N}$ *the decision whether* $\mathrm{hd}(F) \leq k$ *(i.e., whether* $F \in \mathcal{UC}_k$*, or, by Theorem 4.4.4, whether* $F \in \mathcal{SLUR}_k$*) is coNP-complete.*

Proof: The decision whether $F \notin \mathcal{SLUR}_k$ is in NP by definition of $\mathcal{SLUR}_k$ (or use Lemma 3.3.5). By Theorem 3 in [36] we have that $\mathcal{SLUR}$ is coNP-complete, which by Lemma 3.4.8 can be lifted to higher k. □

4.4.2 Comparison to the previous hierarchies

The alternative hierarchies $\mathcal{SLUR}*(k)$ and CANON(k) (recall Subsection 4.1.2) do not generalise r_1 by r_k, but extend r_1 in various ways (maintaining linear-time computation for the (non-deterministic) transitions). In this way in [36, 10] rather complicated argumentations arise, in contrast to our elegant characterisation of the classes $\mathcal{UC}_k$ in Theorem 4.2.2. As a consequence, we can give short proofs that the alternative hierarchies are subsumed by our hierarchy, while already the second level of our hierarchy is (naturally) not contained in any levels of these two hierarchies (naturally, since the time-exponent for deciding whether a (non-deterministic) transition can be done w.r.t. hierarchy $\mathcal{SLUR}_k$ depends on k).

First we simplify and generalise the main result of [10], that CANON(1) $\subseteq \mathcal{SLUR}$. By definition we have CANON(0) $= \mathcal{UC}_0$.

Theorem 4.4.6 *For all $k \in \mathbb{N}_0$ we have:*

1. CANON(k) $\subseteq \mathcal{UC}_k$.
2. $\mathcal{UC}_1 \not\subseteq$ CANON(k) *(and thus* CANON(k) $\subset \mathcal{UC}_k$ *for $k \geq 1$).*

Proof: By Theorem 4.2.2 and the fact, that the Horton-Strahler number of a tree is at most the height, we see that CANON(k) $\subseteq \mathcal{UC}_k$. That $\mathcal{UC}_1 \not\subseteq$ CANON(k) can be seen by observing that there are formulas in $\mathcal{HO} \cap \mathcal{USAT}$ with arbitrary resolution-height complexity and so $\mathcal{HO} \not\subseteq$ CANON(k). By $\mathcal{HO} \subset \mathcal{UC}_1$ we get $\mathcal{UC}_1 \not\subseteq$ CANON(k). □

Also the other hierarchy $\mathcal{SLUR}*(k)$ is strictly contained in our hierarchy:

Theorem 4.4.7 *For all $k \in \mathbb{N}_0$ we have:*

1. $\mathcal{SLUR}*(k) \subset \mathcal{SLUR}_{k+1}$.
2. $\mathcal{SLUR}_2 \not\subseteq \mathcal{SLUR}*(k)$.

Proof: Part 1 follows most easily by using Lemma 4.3.1 together with the simple fact that $\mathrm{slur}*(k)(F) = \{F\}$ for $F \neq \top$ implies $\mathrm{r}_{k+1}(F) = \{\bot\}$; for the strictness of the inclusion use Part 2. Part 2 follows from CANON(2) $\not\subseteq \mathcal{SLUR}*(k)$ (Lemma 13 in [10]), while by Theorem 4.4.6 we have CANON(2) $\subseteq \mathcal{SLUR}_2$. □

Part 1 of Theorem 4.4.7 can not be improved, since $\mathcal{SLUR}*(k)$ and $\mathcal{SLUR}_k$ are incomparable:

Lemma 4.4.8 *For $k \geq 2$ holds $\mathcal{SLUR}*(k) \not\subseteq \mathcal{SLUR}_k$ and $\mathcal{SLUR}_k \not\subseteq \mathcal{SLUR}*(k)$.*

Proof: That $\mathcal{SLUR}_k \not\subseteq \mathcal{SLUR}*(k)$ follows by Part 2 of Theorem 4.4.7. That $\mathcal{SLUR}*(k) \not\subseteq \mathcal{SLUR}_k$ follows from the fact that for the full unsatisfiable clause-set F_k on k variables (i.e., containing all 2^k clauses of length k) we have $F_{k+1} \in \mathcal{SLUR}*(k)$ by Lemma 10 in [10] but $F_{k+1} \notin \mathcal{SLUR}_k$ by Part 2 of Lemma 3.4.7. □

4.5 Optimisation

We conclude by considering the question of finding, for an input-clause-set F, short equivalent clause-sets $F' \in \mathcal{UC}_k$ for fixed k. Definition 4.5.1 provides the appropriate notion of "irredundancy" via the notion of a "k-base", where irredundancy refers to both removal of literal occurrences and removal of clauses. In Theorem 4.5.4 we show that the problem is solvable in polynomial time for inputs $F \in 2\text{-}\mathcal{CLS}$, while in Theorem 4.5.5 we show that the problem is NP-complete even when restricting the input to Horn clause-sets with very few prime implicates.

Definition 4.5.1 *A clause-set F is a k-**base** for some $k \in \mathbb{N}_0 \cup \{+\infty\}$ if $\mathrm{hd}(F) \leq k$, and after removing any literal occurrence or any clause from F, the result F' is either not equivalent to F or has $\mathrm{hd}(F') > k$.*

Remarks:

1. Every k-base F is primal, that is, $F \subseteq \mathrm{prc}_0(F)$ (see Lemma 4.5.2).
2. A clause-set F is a 0-base iff $F = \mathrm{prc}_0(F)$, while F is an ∞-base iff F is primal and irredundant (removal of any clause yields a clause-set not equivalent to F).
3. For a given clause-set F, we consider the problem of computing a shortest (w.r.t. the number of clauses or the number of literal occurrences) equivalent k-base F', which we call a **k-base for F**:
   (a) By [140] for $k = \infty$ this problem is Σ_2-complete.
   (b) A special case of interest here is when $F = \mathrm{prc}_0(F)$, in which case $F' \subseteq F$ must hold. Since all prime implicates are given as input, for $k < \infty$ the decision problem whether F has a k-base of size at most k (k is part of the input) is now in NP. In Theorem 4.5.5 we will see that this decision problem is actually NP-complete, even under rather restricted circumstances.

Lemma 4.5.2 *Consider a clause-set F. If F is a k-base then F is primal.*

Proof: Assume for the sake of contradiction that this is not the case, that there is a clause-set F which is a k-base but is not primal. In this case there is a (non prime) clause $D \in F$ with a prime clause $C \subset D$ of F. If $F' := (F \setminus \{D\}) \cup \{C\} \in \mathcal{UC}_k$ then F is clearly not a k-base as F' is smaller and still represents the same clause-set. Therefore, there must exist a clause C' following from F such that $R : F \vdash_k C'$ but $F' \nvdash_k C'$. However, we can get a (possibly smaller) k-times nested input resolution proof of C' from F' by simply replacing D with C in R. Therefore $F' \vdash_k C'$, a contradiction. □

Example 4.5.3 *Consider the clause-set*

$$F := \{ \underbrace{\{v_1, \overline{v_3}, \overline{v_4}\}}_{C_1}, \underbrace{\{v_2, v_3, \overline{v_4}\}}_{C_2}, \underbrace{\{v_2, \overline{v_3}, v_4\}}_{C_3}, \underbrace{\{\overline{v_2}, v_3, v_4\}}_{C_4}, \underbrace{\{v_1, v_3, v_4\}}_{C_5}, \underbrace{\{v_1, v_2\}}_{C_6} \}.$$

and clause-sets $F_1 := F \setminus \{C_5\}$ and $F_2 := F \setminus \{C_6\}$. We have that:

1. *F is a 0-base, that is, $\mathrm{prc}_0(F) = F$.*

   We have to show that F is closed under resolution modulo subsumption. We have the following possible resolutions in F with the associated subsuming clauses: $C_1 \diamond C_2 \supset C_6$, $C_1 \diamond C_3 \supset C_6$, $C_2 \diamond C_5 \supset C_6$, $C_3 \diamond C_5 \supset C_6$, $C_4 \diamond C_6 = C_5$.

2. *F, F_1 and F_2 are the only k-bases ($k \in \mathbb{N}_0$) that are equivalent to F.*

   To show that there are no other k-bases equivalent to F we must show that all other subsets of F are not equivalent to F. It suffices to show that the clauses C_1, C_2, C_3, C_4 are irredundant (i.e., occur in all primal clause-sets equivalent to F) and the clause-set $F_3 := F \setminus \{C_5, C_6\}$ is not equivalent to F. The irredundancy of C_1, C_2, C_3, C_4 is seen by the fact that they are not obtained as resolvents. That F_3 is not equivalent to F follows from the fact that F_3 does not contain positive clauses while F does.

3. *F_1 is a 1-base (and 2-base) and is equivalent to F but is not a 0-base.*

   We have $C_4 \diamond C_6 = C_5$ and thus $F_1 \models C_5$. To see $\mathrm{hd}(F_1) = 1$, observe $\mathrm{hd}(\varphi_{C_5} * F_1) = \mathrm{hd}(\{\{\overline{v_2}\}, \{v_2\}\}) = 1$.

4. *F_2 is a 2-base and is equivalent to F but is not a 1-base.*

   We have $(C_1 \diamond C_3) \diamond (C_2 \diamond C_5) = C_6$ and thus $F_2 \models C_6$. Furthermore, we have that $\mathrm{hd}(\varphi_{C_6} * F_2) = \mathrm{hd}(\{\{\overline{v_3}, \overline{v_4}\}, \{v_3, \overline{v_4}\}, \{\overline{v_3}, v_4\}, \{v_3, v_4\}\}) = 2$.

5. *Thus F is neither a 1-base nor a 2-base.*

Theorem 4.5.4 *For clause-sets $F \in 2\text{-}\mathcal{CLS}$ we can compute shortest-size (minimum number of clauses or minimum number of literal occurrences) equivalent k-bases F' for all $k \in \mathbb{N}_0 \cup \{+\infty\}$ in polynomial time as follows:*

1. *If F is unsatisfiable, then the best possibility is $F' := \{\bot\}$. So assume in the sequel that F is satisfiable.*

2. *If $F = \top$, then $F' := \top$. So assume in the sequel that $F \neq \top$.*

3. *If F has a forced literal x, then any k-base for F contains $\{x\}$, and we can split off x by considering an optimal k-base for $\langle x \to 1 \rangle * F$. So we can assume w.l.o.g. in the sequel that F has no forced literals. (Thus F as well as $\mathrm{prc}_0(F)$ contains only clauses of length equal 2.)*

4. *Since all k-bases of F without new variables are subsets of $\mathrm{prc}_0(F)$, when considering "shortest k-bases" now there is no differences between the measures c (number of clauses) and ℓ (number of literal occurrences), and we can just speak of "shortest k-bases".*

5. *The (unique) 0-base of F, the set $\mathrm{prc}_0(F) \in 2\text{-}\mathcal{CLS}$ of all prime-implicates, can be computed in polynomial time by the methods discussed in Section 5.8 in [42].*

6. *Every ∞-base of F without new variables is a 1-base (Lemma 4.3.3), and thus w.r.t. k-bases for $k \in \mathbb{N}_0 \cup \{+\infty\}$ only the determination of shortest 1-bases is left, where the shortest 1-bases are precisely the smallest subsets of $\mathrm{prc}_0(F)$ equivalent to F.*

7. *Finally in Chapter 9 of [37] (affirmed in [85]) it is shown how to compute shortest equivalent sets of prime-implicates, and thus shortest 1-bases can be computed in polynomial time.*

Theorem 4.5.5 *Consider $k \in \mathbb{N}_0 \cup \{+\infty\}$.*

1. *Assume $k \geq 1$. The decision problem "For inputs $F \in \mathcal{HO}^+ \cap 3\text{-}\mathcal{CLS}$ with $\mathrm{prc}_0(F) = F$ and $m \in \mathbb{N}_0$, decide whether there is a k-base F' of F with $c(F') \leq m$." (note that here $F' \subseteq F$ must hold) is NP-complete.*

2. *For $k = 0$ the decision problem "For input $F \in \mathcal{HO}$ and $m \in \mathbb{N}_0$, decide whether there is a k-base F' of F with $c(F) \leq m$." is in P.*

Proof: For Part 2 one enumerates with polynomial delay the prime implicates of F (see Section 6.5 in [42] for efficient methods): if this process stops with at most m prime implicates found, then the answer is "yes", otherwise the answer is "no".

For Part 1 we first note that the problem is in NP, since all prime clauses are given, and $\mathrm{hd}(F) \leq 1$. The heart of the completeness is Theorem 6.18 in [42], which states that "Horn minimisation w.r.t. the number of clauses remains NP-complete even if the input is restricted to cubic pure Horn expressions.", plus the fact from the underlying report [27], that for the considered $G \in \mathcal{HO}^+ \cap 3\text{-}\mathcal{CLS}$ all prime implicates are also of length at most 3, and thus we can take as input $F := \mathrm{prc}_0(G) \in \mathcal{HO}^+ \cap 3\text{-}\mathcal{CLS}$ (which can be computed in polynomial time). □

4.6 Interleaved hierarchies: $\mathcal{UC}_k$, $\mathcal{PC}_k$, and $\mathcal{WC}_k$.

Having introduced $\mathcal{UC}_k$, which is based on the hardness measure hd from Chapter 3, attention is now turned to related hierarchies based on p-hardness and width-based hardness measures introduced in Section 3.5 and Section 3.6 of Chapter 3. These hierarchies are not the central topic of study for this thesis but are directly related to $\mathcal{UC}_k$ and so it is both necessary to define them and relate them to $\mathcal{UC}_k$.

4.6.1 $\mathcal{PC}_k$: propagation completeness at level k

Complementary to "unit-refutation completeness" there is the notion of "propagation completeness", as investigated in [130, 26]. This is captured and generalised by a corresponding measure $\mathrm{phd} : \mathcal{CLS} \to \mathbb{N}_0$ of "propagation-hardness" introduced in Section 3.5, and the class $\mathcal{PC}$ of "propagation-complete clause-sets" can be properly generalised as follows:

Definition 4.6.1 *For $k \in \mathbb{N}_0$ let $\mathcal{PC}_k := \{F \in \mathcal{CLS} : \mathrm{phd}(F) \leq k\}$ (the class of **propagation-complete clause-sets of level k**).*

Remarks:

1. By definition of r_k we see that $\mathcal{PC}_k \subseteq \mathcal{PC}_{k+1}$ for all $k \in \mathbb{N}_0$ (i.e., we get a hierarchy). In Lemma 4.6.2 we'll see that this hierarchy is strict.
2. By Lemma 3.5.4 we get $\bigcup_{k \in \mathbb{N}_0} \mathcal{PC}_k = \mathcal{CLS}$.
3. By Lemma 3.5.8 we have $\mathcal{PC} = \mathcal{PC}_1$ from [26].

These classes lie (strictly) between the $\mathcal{UC}_k$-classes (stronger than Lemma 3.5.3):

Lemma 4.6.2 *For $k \in \mathbb{N}_0$ we have $\mathcal{PC}_k \subset \mathcal{UC}_k \subset \mathcal{PC}_{k+1}$.*

Proof: By Lemma 3.5.6 we get $\mathcal{PC}_{k+1} \setminus \mathcal{UC}_k \neq \emptyset$ (Lemma 3.5.6 provides $F \in (\mathcal{UC}_{k+1} \cap \mathcal{PC}_{k+1}) \setminus \mathcal{UC}_k$) , while by Lemma 3.5.7 we get $\mathcal{UC}_k \setminus \mathcal{PC}_k \neq \emptyset$. □
Remarks:

1. The initial five (strict) inclusions $\mathcal{PC}_0 \subset \mathcal{UC}_0 \subset \mathcal{PC}_1 \subset \mathcal{UC}_1 \subset \mathcal{PC}_2 \subset \mathcal{UC}_2$ are certified by the clause-sets

(a) $\{\{v_1\}\} \in \mathcal{UC}_0 \setminus \mathcal{PC}_0$

(b) $\{\{v_1, v_2\}, \{\overline{v_1}\}\} \in \mathcal{PC}_1 \setminus \mathcal{UC}_0$

(c) $\{\{v_1, v_2\}, \{\overline{v_1}, v_2\}\} \in \mathcal{UC}_1 \setminus \mathcal{PC}_1$

(d) $\{\{v_1, v_2, v_3\}, \{\overline{v_1}, v_2\}, \{v_1, \overline{v_2}\}, \{\overline{v_1}, \overline{v_2}\}\} \in \mathcal{PC}_2 \setminus \mathcal{UC}_1$

(e) $\{\{v_1, v_2, v_3\}, \{\overline{v_1}, v_2, v_3\}, \{v_1, \overline{v_2}, v_3\}, \{\overline{v_1}, \overline{v_2}, v_3\}\} \in \mathcal{UC}_2 \setminus \mathcal{PC}_2$.

From Lemma 4.3.2 and the fact that $\mathcal{UC}_k \subseteq \mathcal{PC}_{k+1}$ (or by definition) we get:

Lemma 4.6.3 *Consider $F \in \mathcal{CLS}$.*

1. *If $F \in 2\text{-}\mathcal{CLS} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C| \leq 2\}$, then $F \in \mathcal{PC}_3$ (by definition).*
2. *If $F \in \mathcal{HO} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C \cap \mathcal{VA}| \leq 1\}$ (Horn clause-sets), then $F \in \mathcal{PC}_2$.*
3. *More generally, if $F \in \mathcal{QHO}$, the set of q-Horn clause-sets (see Section 6.10.2 in [42], and [158]), then $F \in \mathcal{PC}_3$.*
4. *Generalising Horn clause-sets to the hierarchy $\mathcal{HO}_k$ from [99] (with $\mathcal{HO}_1 = \mathcal{HO}$): if $F \in \mathcal{HO}_k$ for $k \in \mathbb{N}$, then $F \in \mathcal{PC}_{k+1}$ (that $\mathcal{HO}_k \not\subseteq \mathcal{PC}_k$ follows by Lemma 3.5.5).*

Strengthening Lemma 4.3.2, Part 3 and Lemma 4.6.3, Part 1:

Lemma 4.6.4 *Consider $F \in 2\text{-}\mathcal{CLS}$.*

1. *If F is satisfiable, then* $\mathrm{hd}(F) \leq 1$.
2. $\mathrm{phd}(F) \leq 2$ *(and thus $2\text{-}\mathcal{CLS} \subseteq \mathcal{PC}_2$).*
3. *If F has no forced literals and $F \neq \top$, then* $\mathrm{phd}(F) = 1$.

Proof: Part 2 follows from Part 1 and Lemma 4.3.2, Part 3. For Part 1 consider a partial assignment φ with unsatisfiable $\varphi * F$. Now we have $\mathrm{r}_1(\varphi * F) = \{\bot\}$, since otherwise $\mathrm{r}_1(\varphi * F) \subseteq F$, and thus $\mathrm{r}_1(\varphi * F)$ would be satisfiable. Finally for Part 3 consider a partial assignment φ. If $\mathrm{r}_1(\varphi * F) \neq \{\bot\}$, then $\mathrm{r}_1(\varphi * F) \subseteq F$, and since F has no forced literals, also $\mathrm{fl}(\mathrm{r}_1(\varphi * F)) = \emptyset$ holds, whence $\mathrm{phd}(F) \leq 1$. □

Remarks:

1. Examples from $2\text{-}\mathcal{CLS}$ for different values of hd and phd:
   (a) $F_0 \in \{\top, \{\{\bot\}\}\}$: $\mathrm{hd}(F_0) = \mathrm{phd}(F_0) = 0$.
   (b) $F_1 := \{\{a, b\}\} \in 2\text{-}\mathcal{CLS}$: $\mathrm{hd}(F_1) = 0$ and $\mathrm{phd}(F_1) = 1$.
   (c) $F_2 := \{\{a, b\}, \{\overline{b}, c\}\} \in 2\text{-}\mathcal{CLS}$: $\mathrm{hd}(F_2) = \mathrm{phd}(F_2) = 1$.
   (d) $F_3 := \{\{a, b\}, \{a, \overline{b}\}\} \in 2\text{-}\mathcal{CLS}$: $\mathrm{hd}(F_3) = 1$ and $\mathrm{phd}(F_3) = 2$.
   (e) $F_4 := \{\{a, b\}, \{a, \overline{b}\}, \{\overline{a}, b\}, \{\overline{a}, \overline{b}\}\} \in 2\text{-}\mathcal{CLS}$: $\mathrm{hd}(F_4) = \mathrm{phd}(F_4) = 2$.
   (f) $F_5 := \{\{a, b, v\}, \{a, \overline{b}\}, \{\overline{a}, b\}, \{\overline{a}, \overline{b}\}\} \notin 2\text{-}\mathcal{CLS}$ but $\mathrm{hd}(F_5) = \mathrm{phd}(F_5) = 2$ (hence $2\text{-}\mathcal{CLS} \subset \mathcal{PC}_2$).
   (g) The statements for F_1, F_3, F_4 are covered by Lemmas 3.4.7, 3.5.5, and F_5 is covered by Lemma 3.5.6.
   (h) Note that F_1, F_2, F_3 and F_5 are satisfiable, whereas F_4 is not. By Part 1 of Lemma 4.6.4 there are no satisfiable clause-sets $F \in 2\text{-}\mathcal{CLS}$ with $\mathrm{hd}(F) = 2$.

Corollary 4.6.5 *Consider a clause-set $F \in 2\text{-}\mathcal{CLS}$. If F is primal then $F \in \mathcal{PC}_1$.*

Proof: If F is primal then either $F = \{\bot\}$ hence $F \in \mathcal{UC}_0 \subseteq \mathcal{PC}_1$ or *all* forced assignments occur as unit-clauses in F and hence $\mathrm{r}_1(F)$ yields a clause-set without forced assignments, which by Lemma 4.6.4 is in $\mathcal{PC}_1$ and so by Lemma 3.1.2 and the definition of $\mathcal{PC}_k$ we have $F \in \mathcal{PC}_1$. □

Due to the interleaved nature of $\mathcal{PC}_k$ with $\mathcal{UC}_k$ we get also coNP-completeness of the membership problem for $k > 2$:

Theorem 4.6.6 *For fixed $k \in \mathbb{N}$ with $k \geq 2$ the decision whether $\mathrm{phd}(F) \leq k$ (i.e., whether $F \in \mathcal{PC}_k$) is coNP-complete.*

Proof: That the decision whether $F \notin \mathcal{PC}_k$ is in NP follows from the fact that if $F \notin \mathcal{PC}_k$ then there is a partial assignment φ witnessing this such that $\mathrm{r}_\infty(\varphi * F) \neq \mathrm{r}_k(\varphi * F)$. By Lemma 3.3.6 φ can be extended to a (still poly-size) witness φ' for which $\mathrm{r}_{k+1}(\varphi' * F) \neq \mathrm{r}_k(\varphi' * F)$ yielding a poly-size witness that $F \notin \mathcal{PC}_k$ which is checkable in poly-time via running r_{k+1} and r_k. Therefore $F \in \mathcal{PC}_k$ is in coNP. By Theorem 4.4.5 we have that checking whether $F \in \mathcal{UC}_k$ is coNP-complete; deciding $F \in \mathcal{UC}_k$ can be translated to deciding $F' \in \mathcal{PC}_{k+1}$ by Lemma 3.5.7 and so by reduction, checking $F \in \mathcal{PC}_k$ for any fixed $k \geq 2$ is coNP-complete. □

Note that Theorem 4.6.6 does not prove that membership for $\mathcal{PC}$ is coNP-complete. Membership for $\mathcal{UC}_0$ is poly-time and so the simple proof technique using Lemma 3.5.7 does not imply that membership for $\mathcal{PC}$ is coNP-complete. This is left as a conjecture:

Conjecture 4.6.7 *The decision whether $\mathrm{phd}(F) \leq 1$ (i.e., whether $F \in \mathcal{PC}_1$) is coNP-complete.*[7)]

4.6.2 $\mathcal{WC}_k$: width-k refutation completeness

Complementary to "unit-refutation completeness" but in a *different* direction is the notion of "width-refutation completeness". This is captured and generalised by the corresponding measure $\mathrm{whd} : \mathcal{CLS} \to \mathbb{N}_0$ of "width-based hardness" introduced in Section 3.6.2 (generalising the asymmetric notion of width from [104, 110]), and the class $\mathcal{WC} = \mathcal{UC}$ can be properly generalised as follows:

Definition 4.6.8 *For $k \in \mathbb{N}_0$ let $\boldsymbol{\mathcal{WC}_k} := \{F \in \mathcal{CLS} : \mathrm{whd}(F) \leq k\}$ (the class of **refutation complete clauses-sets of width** k).*

Remarks:

1. By definition of r_k we see that $\mathcal{WC}_k \subseteq \mathcal{WC}_{k+1}$ for all $k \in \mathbb{N}_0$ (i.e., we get a hierarchy). In Lemma 4.6.2 we'll see that this hierarchy is strict.
2. By Lemma 3.6.3 we have that $\mathcal{WC}_k$ is closed under application of partial assignments.

We have $\mathcal{WC}_0 = \mathcal{UC}_0$, $\mathcal{WC}_1 = \mathcal{UC}_1$, and for all $k \in \mathbb{N}_0$ holds $\mathcal{UC}_k \subseteq \mathcal{WC}_k$ (this follows by Lemma 6.8 in [110] for unsatisfiable clause-sets, which extends to satisfiable clause-sets by definition). For unsatisfiable F, whether $\mathrm{whd}(F) = k$ holds for $k \in \{0, 1, 2\}$ can be decided in polynomial time; this is non-trivial for $k = 2$ ([33]) and unknown for $k > 2$. Nevertheless, the clausal entailment problem $F \models C$ for $F \in \mathcal{WC}_k$ and fixed $k \in \mathbb{N}_0$ is decidable in polynomial time,

[7)] Petr Kucera and Ondrej Cepek (the authors of [36], which shows that $\mathcal{SLUR}_k$ (= $\mathcal{UC}_k$) membership is coNP-complete) have stated that they are in the process of publishing a paper showing that $\mathcal{PC}_1$ membership is coNP-complete.

as shown in Subsection 6.5 of [110], by actually using a slight strengthening of k-resolution, which combines width-bounded resolution and input resolution. While space-complexity of the decision $F \models C$ for $F \in \mathcal{UC}_k$ is linear (for fixed k), now for $\mathcal{WC}_k$ space-complexity is $O(\ell(F) \cdot n(F)^{O(k)})$.

Lemma 4.6.9 *For all $k \in \mathbb{N}_0$ we have that $\mathcal{WC}_k \subset \mathcal{WC}_{k+1}$.*

Proof: That $\mathcal{WC}_k \subseteq \mathcal{WC}_{k+1}$ follows by definition. To see that $\mathcal{WC}_{k+1} \setminus \mathcal{WC}_k \neq \emptyset$, consider a full clause-set F with $k+1$ variables and observe that $F \in \mathcal{WC}_{k+1} \setminus \mathcal{WC}_k \neq \emptyset$. □

From Lemma 4.3.2 and the fact that $\mathcal{UC}_k \subseteq \mathcal{WC}_k$ (or by definition) we get:

Lemma 4.6.10 *Consider $F \in \mathcal{CLS}$.*

1. *If $F \in 2\text{-}\mathcal{CLS} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C| \leq 2\}$, then $\mathrm{whd}(F) \leq 2$ (by definition).*
2. *If $F \in \mathcal{HO} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C \cap \mathcal{VA}| \leq 1\}$ (Horn clause-sets), then $F \in \mathcal{WC}_1$.*
3. *More generally, if $F \in \mathcal{QHO}$, the set of q-Horn clause-sets (see Section 6.10.2 in [42], and [158]), then $F \in \mathcal{WC}_2$.*
4. *Generalising Horn clause-sets to the hierarchy $\mathcal{HO}_k$ from [99] (with $\mathcal{HO}_1 = \mathcal{HO}$): if $F \in \mathcal{HO}_k$ for $k \in \mathbb{N}$, then $F \in \mathcal{WC}_k$ (that $\mathcal{HO}_k \not\subseteq \mathcal{WC}_{k-1}$ follows by considering full unsatisfiable clause-sets on k variables).*

In Theorem 6.4.7 we shall see an example of a family of clause-sets $(F_n)_{n \in \mathbb{N}_0}$ which all fit in $\mathcal{WC}_3$ but for which $F_{n+1} \in \mathcal{UC}_{n+1} \setminus \mathcal{UC}_n$, demonstrating that simplicity in terms of clause-set membership that the $\mathcal{WC}_k$ hierarchy is a much more powerful hierarchy than $\mathcal{UC}_k$. In the outlook it is hypothesised in Conjecture 8.1.1 that for representing boolean functions (even using new variables) that $\mathcal{WC}_2$ can represent boolean functions (with short representations) for which there are no poly-size representations in any fixed level of the $\mathcal{UC}_k$ hierarchy.

4.7 Knowledge compilation properties

In [46] we saw that the motivation for defining $\mathcal{UC}$ was to introduce a class of clause-sets for knowledge compilation where certain types of query have the same query complexity as the PI class (see Figure 1.4 in Section 1.3.2 of Chapter 1). Theorem 4.7.1 now shows that $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ all fulfill the same criteria, and in Chapter 5 we will see that each level of these hierarchies offers potentially exponentially more succinctness. These two results mean that $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ offer intermediate target classes for knowledge compilation inbetween the CNF and PI classes where the parameter k allows query time to be traded for size.

Theorem 4.7.1 *For all fixed $k \in \mathbb{N}_0$ and all $F, F' \in \mathcal{WC}_k \supseteq \mathcal{UC}_k \supseteq \mathcal{PC}_k$ we have that the following queries are decidable in polynomial time (in $c(F)$):*

- *Consistency checking (CO - i.e., whether $F \models \bot$).*
- *Clausal entailment checking (CE - i.e., whether $F \models C$ for arbitrary $C \in \mathcal{CL}$).*
- *Validity checking (VA - i.e., whether $\top \models F$).*
- *Implicant checking (IM - i.e., whether $\top \models \varphi_C^1 * F$ for an arbitrary clause $C \in \mathcal{CL}$; whether C as a DNF clause is an implicant of F).*

- *Equivalence checking (EQ - i.e., whether* $\text{CNF}(F) \cong \text{CNF}(F')$*).*
- *Semantic Entailment (SE - i.e., whether* $F \models F'$*).*

Furthermore, Model Enumeration (ME - i.e., enumerating all satisfying assignments) is possible in time $p(c(F), m)$ *for polynomial* p *where* m *is the number of satisfying total assignments for* F*.*

Proof: That clausal entailment is decidable in poly-time for $\mathcal{WC}_k$ is shown in Subsection 6.5 of [104] (see 4.6.2 of this thesis for more discussion). That validity and implicant checking are poly-time follows from the fact that $\mathcal{WC}_k$ are CNF clause-sets and so implicant checking is simply checking whether $\varphi^1_C * F = \top$. That semantic entailment is poly-time decidable follows from the fact that $F \models F'$ can be checked by checking if $F \models C$ for all $C \in F'$ and so semantic entailment can be checked by $c(F')$ (poly-time) clausal entailment checks. That equivalence checking then follows from the ability to checking semantic entailment (i.e., just check $F \models F'$ and $F' \models F$). Finally, that all models can be enumerated in poly-time in $c(F)$ and m follows from the fact that CE and VA are possible in polynomial time (see Lemma A.3 in [46]). In particular, since CE and VA are possible in polynomial time then this allows derivation of a minimal decision tree with at most m $\top$ leaves and at most $n(F) \cdot m$ $\bot$ leaves. □

Chapter 5

Strict hierarchies for equivalent representations

Having now introduced the hierarchies $\mathcal{UC}_k$, $\mathcal{PC}_k$, and $\mathcal{WC}_k$, this chapter now deals with the strictness of these hierarchies for representation of boolean functions. More precisely, the main result is that each level $\mathcal{UC}_{k+1} \subseteq \mathcal{WC}_{k+1}$ contains families of clause-sets which are poly-size but for which there is no equivalent poly-size family of clause-sets in $\mathcal{UC}_k \subseteq \mathcal{WC}_k$ (see Theorem 5.3.13).

In Section 5.1 minimal premise sets are introduced as a method for characterising and understanding the structure of unsatisfiable sub-clause-sets of a clause-set F. Doping is then introduced, providing a way to map unsatisfiable sub-clause-sets to prime implicates (and back), providing clause-sets with well characterised prime implicates. Finally, in Section 5.2 these notions are applied to our source of hard examples (saturated minimally unsatisfiable clause-sets of deficiency 1), leading to the main separation result in Section 5.3.

5.1 Minimal premise sets and doped clause-sets

In this section we study "minimal premise sets", "mps's" for short, introduced in [116], together with the properties of "doped" clause-sets, generalising a construction used in [148]. Mps's are generalisations of minimally unsatisfiable clause-sets stronger than irredundant clause-sets, while doping relates prime implicates and sub-mps's (i.e., minimal premise sets as subsets of a clause-set).

Recall that a clause-set F is minimally unsatisfiable if $F \in \mathcal{USAT}$, while for all $C \in F$ holds $F \setminus \{C\} \in \mathcal{SAT}$. The set of all minimally unsatisfiable clause-sets is $\mathcal{MU} \subset \mathcal{CLS}$; see [100] for more information. In other words, for $F \in \mathcal{CLS}$ we have $F \in \mathcal{MU}$ if and only if $F \models \bot$ and F is minimal regarding this entailment relation. Now an mps is a clause-set F which minimally implies some clause C, i.e., $F \models C$, while $F' \not\models C$ for all $F' \subset F$. In Subsection 5.1.1 we study the basic properties of mps's F, and determine the unique minimal clause implied by F as $\mathrm{puc}(F)$, the set of pure literals of F.

For a clause-set F its doped version $\mathrm{D}(F) \in \mathcal{CLS}$ receives an additional new ("doping") variable for each clause. The basic properties are studied in Subsection 5.1.2, and in Theorem 5.1.18 we show that the prime implicates of $\mathrm{D}(F)$ correspond 1-1 to the mps's contained in F. In Subsection 5.1.3 we determine the hardness of doped clause-sets.

5.1.1 Minimal premise sets

In Section 4.1 in [116] basic properties of *minimal premise sets* are considered:

Definition 5.1.1 *A clause-set $F \in \mathcal{CLS}$ is a* ***minimal premise set*** *("mps")* ***for a clause*** *$C \in \mathcal{CL}$ if $F \models C$ and $\forall F' \subset F : F' \not\models C$, while F is a* ***minimal premise set*** *if there exists a clause C such that F is a minimal premise set for C. The set of all minimal premise (clause-)sets is denoted by $\mathcal{MPS}$.*

Remarks:

1. $\top$ is not an mps (since no clause follows from $\top$).
2. An unsatisfiable clause-set is an mps iff it is minimally unsatisfiable, i.e., $\mathcal{MPS} \cap \mathcal{USAT} = \mathcal{MU}$. In Corollary 5.1.8 we will see that the minimally unsatisfiable clause-sets are precisely the mps's without pure literals.
3. Every minimal premise clause-set is irredundant (no clause follows from the other clauses).
4. For a clause-set F and any implicate $F \models C$ there exists a minimal premise sub-clause-set $F' \subseteq F$ for C.
5. A single clause C yields an mps $\{C\}$.
6. Two clauses $C \neq D$ yield an mps $\{C, D\}$ iff C, D are resolvable.
7. If $F_1, F_2 \in \mathcal{MPS}$ with $\mathrm{var}(F_1) \cap \mathrm{var}(F_2) = \emptyset$, then $F_1 \cup F_2 \notin \mathcal{MPS}$ except in case of $F_1 = F_2 = \{\bot\}$.

Example 5.1.2 *$\{\{a\}, \{b\}\}$ for variables $a \neq b$ is irredundant but not an mps.*

With Corollary 4.5 in [116] we see that every clause-set minimally entails at most one clause:

Lemma 5.1.3 *For $F \in \mathcal{MPS}$ there exists exactly one $C \in \mathrm{prc}_0(F)$ such that F is a minimal premise set for C, and C is the smallest element of the set of clauses for which F is a minimal premise set.*

We remark that Lemma 5.1.3 does not mean that $|\mathrm{prc}_0(F)| = 1$ for $F \in \mathcal{MPS}$; indeed, F can have many $F' \subset F$ with $F' \in \mathcal{MPS}$, and each such F' might contribute a prime implicate, as we will see later. We wish now to determine that unique prime implicate C which follows minimally from an mps F. It is clear that C must contain all pure literals from F, since all clauses of F must be used, and we can not get rid of pure literals.

Definition 5.1.4 *For $F \in \mathcal{CLS}$ the* ***pure clause of*** *F, denoted by* $\mathbf{puc(F)} \in \mathcal{CL}$*, is the set of pure literals of F, that is, $\mathrm{puc}(F) := L \setminus (L \cap \overline{L})$, where $L := \bigcup F$ is the set of literals occurring in F.*

Example 5.1.5 *For $F = \{\{a, b\}, \{\overline{a}, c\}\}$ we have $\mathrm{puc}(F) = \{b, c\}$.*

The main observation for determining C is that the conclusion of a regular resolution proof consists precisely of the pure literals of the axioms (this follows by definition):

Lemma 5.1.6 *For a regular resolution proof $T : F \vdash C$, where every clause of F is used in T, we have $C = \mathrm{puc}(F)$.*

Due to the completeness of regular resolution we thus see, that $\mathrm{puc}(F)$ is the desired unique prime implicate:

Lemma 5.1.7 *For $F \in \mathcal{MPS}$ the unique prime implicate C, for which F is a minimal premise set (see Lemma 5.1.3), is $C = \mathrm{puc}(F)$.*

Proof: Consider a regular resolution proof $T : F \vdash C$ (recall that regular resolution is complete); due to $F \in \mathcal{MPS}$ every clause of F must be used in T, and thus the assertion follows by Lemma 5.1.6. □

Corollary 5.1.8 *If we have $F \in \mathcal{MPS}$ with $\mathrm{puc}(F) = \bot$, then $F \in \mathcal{MU}$.*

By Lemma 4.4 in [116] we get the main characterisation of mps's, namely that after elimination of pure literals they must be minimally unsatisfiable:

Lemma 5.1.9 *Consider a clause-set $F \in \mathcal{CLS}$. Then $F \in \mathcal{MPS}$ if and only if the following two conditions hold for $\varphi := \varphi_{\mathrm{puc}(F)}$ (setting precisely the pure literals of F to false):*

1. *$\varphi * F \in \mathcal{MU}$ (after removing the pure literals we obtain a minimal unsatisfiable clause-sets).*
2. *φ is contraction-free for F, that is, for clauses $C, D \in F$ with $C \neq D$ we have $\varphi * \{C\} \neq \varphi * \{D\}$.*

These two conditions are equivalent to stating that $\varphi * F$ as a multi-clause-set (not contracting equal clauses) is minimally unsatisfiable.

Remarks:

1. Note that if we didn't have condition 2 then we could have e.g., $F := \{\{x_1, y_1\}, \{x_1, y_2\}, \{\overline{x_1}\}\}$ where $\varphi_{\mathrm{puc}(F)} = \langle y_1 \to 0, y_2 \to 0\rangle$ and $\varphi * F = \{\{x_1\}, \{\overline{x_1}\}\} \in \mathcal{MU}$ but every clause following from F follows from a strict subset (note that if $\{y_1\}$ follows then also $\{y_1, y_2\}$ does).

Thus we obtain all mps's by considering some minimally unsatisfiable clause-sets and adding new variables in the form of pure literals:

Corollary 5.1.10 *The following process generates precisely the $F' \in \mathcal{MPS}$:*

1. *Choose $F \in \mathcal{MU}$.*
2. *Choose a clause P with $\mathrm{var}(P) \cap \mathrm{var}(F) = \emptyset$ ("P" like "pure").*
3. *Choose a map $e : F \to \mathbb{P}(P)$ ("e" like "extension").*
4. *Let $F' := \{C \cup e(C) : C \in F\}$.*

For unsatisfiable clause-sets the set of minimally unsatisfiable sub-clause-sets has been studied extensively in the literature; see [120] for a recent overview. The set of subsets which are mps's strengthen this notion (now for all clause-sets):

Definition 5.1.11 *For a clause-set $F \in \mathcal{CLS}$ by* $\mathbf{mps(F)} \subset \mathcal{CLS}$ *the set of all minimal premise sub-clause-sets is denoted:* $\mathrm{mps}(F) := \mathbb{P}(F) \cap \mathcal{MPS}$.

We have $|\mathrm{mps}(F)| \leq 2^{c(F)} - 1$ (there is a typo in Corollary 4.6 of [116], misplacing the "-1" into the exponent). The minimal elements of $\mathrm{mps}(F)$ are $\{C\} \in \mathrm{mps}(F)$ for $C \in F$. Since every prime implicate of a clause-set has some minimal premise sub-clause-set, we get that running through all sub-mps's in a clause-set F and extracting the clauses with the pure literals we obtain at least all prime implicates:

Lemma 5.1.12 *For $F \in \mathcal{CLS}$ the map $F' \in \mathrm{mps}(F) \mapsto \mathrm{puc}(F') \subseteq \{C \in \mathcal{CL} : F \models C\}$ covers $\mathrm{prc}_0(F)$ (i.e., its range contains the prime implicates of F).*

Example 5.1.13 *Examples where we have more minimal premise sub-clause-sets than prime implicates are given by $F \in \mathcal{MU}$, where $\mathrm{prc}_0(F) = \{\bot\}$, while in the most extreme case every non-empty subset of F can be a minimal premise sub-clause-set (see Theorem 5.2.11).*

5.1.2 Doping clause-sets

"Doping" is the process of adding a unique new variable to every clause of a clause-set. It enables us to follow the usage of this clause in resolution derivations:

Definition 5.1.14 *For every clause-set $F \in \mathcal{CLS}$ we assume an injection $u^F : F \to \mathcal{VA} \setminus \mathrm{var}(F)$ in the following, assigning to every clause C a different variable u_C^F. For a clause $C \in \mathcal{CL}$ and a clause-set $F \in \mathcal{CLS}$ we then define the* ***doping*** $\mathbf{D_F(C)} := C \cup \{u_C^F\} \in \mathcal{CL}$, *while* $\mathbf{D(F)} := \{\mathrm{D}_F(C) : C \in F\} \in \mathcal{CLS}$.

Remarks:

1. "Doping" has various meanings, where here we mean the meaning as used when "doping" semiconductors to modulate their conductivity properties (see [161]).
2. In the following we drop the upper index in "u_C^F", i.e., we just use "u_C".
3. We have $\mathrm{D} : \mathcal{CLS} \to \mathcal{SAT}$.
4. For $F \in \mathcal{CLS}$ we have $n(\mathrm{D}(F)) = n(F) + c(F)$ and $c(\mathrm{D}(F)) = c(F)$.
5. For $F \in \mathcal{CLS}$ we have $\mathrm{puc}(\mathrm{D}(F)) = \mathrm{puc}(F) \cup \{u_C : C \in F\}$.

We are interested in the prime implicates of doped clause-sets. It is easy to see that all doped clauses are themselves essential prime implicates:

Lemma 5.1.15 *For $F \in \mathcal{CLS}$ we have $\mathrm{D}(F) \subseteq \mathrm{prc}_0(\mathrm{D}(F))$, and furthermore all elements of $\mathrm{D}(F)$ are essential prime implicates.*

Proof: Every resolvent of clauses from $\mathrm{D}(F)$ contains at least two doping variables, and thus the clauses of $\mathrm{D}(F)$ themselves (which contain only one doping variable) are prime and necessary. □

Thus by Lemma 2.6.11 among all the clause-sets equivalent to $\mathrm{D}(F)$ this clause-set itself is the smallest. Directly by Lemma 5.1.9 we get that a clause-set is an mps iff its doped form is an mps:

Lemma 5.1.16 *For $F \in \mathcal{CLS}$ holds $F \in \mathcal{MPS} \Leftrightarrow \mathrm{D}(F) \in \mathcal{MPS}$. Thus the map $F' \in \mathrm{mps}(F) \mapsto \mathrm{D}(F')$ is a bijection from $\mathrm{mps}(F)$ to $\mathrm{mps}(\mathrm{D}(F))$.*

For doped clause-sets the surjection, from $\mathrm{mps}(F)$ to $\mathrm{prc}_0(F)$, of Lemma 5.1.12 is bijective:

Lemma 5.1.17 *Consider a clause-set $F \in \mathcal{CLS}$, and let $G := \mathrm{D}(F)$.*

1. *The map $F' \in \mathrm{mps}(G) \mapsto \mathrm{puc}(F') \in \mathcal{CL}$ is a bijection from $\mathrm{mps}(G)$ to $\mathrm{prc}_0(G)$.*
2. *The inverse map from $\mathrm{prc}_0(G)$ to $\mathrm{mps}(G)$ obtains from $C \in \mathrm{prc}_0(G)$ the clause-set $F' \in \mathrm{mps}(G)$ with $\mathrm{puc}(F') = C$ as $F' = \{\mathrm{D}(D) : D \in F \wedge u_D \in \mathrm{var}(C)\}$.*

Proof: By Lemma 5.1.12 it remains to show that the map of Part 1 is injective and does not have subsumptions in the image. Assume for the sake of contradiction there are $G', G'' \in \mathrm{mps}(G)$, $G' \neq G''$, with $\mathrm{puc}(G') \subseteq \mathrm{puc}(G'')$. Since every clause of F has a different doping-variable, $G' \subset G''$ must hold. Consider the $F', F'' \in \mathrm{mps}(F)$ with $\mathrm{D}(F') = G'$ and $\mathrm{D}(F'') = G''$. We have $F' \subset F''$, and thus $\mathrm{puc}(F') \not\subseteq \mathrm{puc}(F'')$, since for every $F \in \mathcal{MPS}$ the clause $\mathrm{puc}(F)$ is a prime implicate of F. It follows that $\mathrm{puc}(G') \not\subseteq \mathrm{puc}(G'')$, contradicting the assumption. □

By Lemma 5.1.16 and Lemma 5.1.17 we obtain:

Theorem 5.1.18 *Consider $F \in \mathcal{CLS}$. Then the map $F' \in \mathrm{mps}(F) \mapsto \mathrm{puc}(\mathrm{D}(F')) \in \mathcal{CL}$ is a bijection from $\mathrm{mps}(F)$ to $\mathrm{prc}_0(\mathrm{D}(F))$.*

Theorem 5.1.18 together with the description of the inversion map in Lemma 5.1.17 yields computation of the set $\mathrm{mps}(F)$ for $F \in \mathcal{CLS}$ via computation of $\mathrm{prc}_0(\mathrm{D}(F))$.

Corollary 5.1.19 *For $F \in \mathcal{CLS}$ we obtain a map from $\mathrm{prc}_0(\mathrm{D}(F))$ to the set of implicates of F covering $\mathrm{prc}_0(F)$ by the mapping $C \in \mathrm{prc}_0(\mathrm{D}(F)) \mapsto C \setminus V$ for $V := \{u_C : C \in F\}$.*

Proof: The given map can be obtained as a composition as follows: For $C \in \mathrm{prc}_0(\mathrm{D}(F))$ take (the unique) $F' \in \mathrm{mps}(F)$ with $\mathrm{puc}(\mathrm{D}(F')) = C$, and we have $C \setminus V = \mathrm{puc}(F')$. □

5.1.3 Hardness of doped clause-sets

The hardness of a doped clause-set is the maximal hardness of sub-clause-sets of the original clause-set:

Lemma 5.1.20 *For $F \in \mathcal{CLS}$ we have $\mathrm{hd}(\mathrm{D}(F)) = \max_{F' \subseteq F} \mathrm{hd}(F')$.*

Proof: We have $\mathrm{hd}(F') \leq \mathrm{hd}(\mathrm{D}(F))$ for all $F' \subseteq F$, since via applying a suitable partial assignment we obtain F' from F, setting the doping-variables in F' to false, and the rest to true. And if we consider an arbitrary partial assignment φ with $\varphi * \mathrm{D}(F) \in \mathcal{USAT}$, then w.l.o.g. all doping variables are set (we can set the doping-variables not used by φ to true, since these variables are all pure), and then we have a partial assignment making F' unsatisfiable for that $F' \in \mathcal{USAT}$ given by all the doping variables set by φ to false. □

Example 5.1.21 *For an example of a clause-set $F \in \mathcal{USAT}$ with $\mathrm{hd}(\mathrm{D}(F)) > \mathrm{hd}(F)$ consider any clause-set $F' \in \mathcal{CLS}$ with $\mathrm{hd}(F') > 0$, and then take $F := F' \cup \{\bot\}$ (note that $\bot \notin F'$). Thus $\mathrm{hd}(F) = 0$. And by Part 1 of Lemma 6.5 in [77, 78], all $\mathcal{UC}_k$ are closed under partial assignments, so for $\varphi := \langle u_\bot \to 1\rangle \cup \langle u_C \to 0 \mid C \in F'\rangle$ we have $\mathrm{hd}(\mathrm{D}(F)) \geq \mathrm{hd}(\varphi * \mathrm{D}(F)) = \mathrm{hd}(F') > \mathrm{hd}(F) = 0$.*

5.2 Doping tree clause-sets

As explained in Subsection 1.3, we want to construct boolean functions (given by clause-sets) with a large number of prime implicates, and where we have strong control over these prime implicates. For the purpose we dope "minimally unsatisfiable clause-sets of deficiency 1", that is the elements of $\mathcal{SMU}_{\delta=1}$. First we review in Subsection 5.2.1 the background (for more information see [100]). Then in Subsection 5.2.2 we consider doping of these special clause-sets. In Theorem 5.2.11 we show that $F \in \mathcal{SMU}_{\delta=1}$ are precisely the clause-sets such that every non-empty subset is an mps, and in Theorem 5.2.18 we determine basic properties of $\mathrm{D}(F)$.

5.2.1 Preliminaries on minimal unsatisfiability

A minimally unsatisfiable $F \in \mathcal{MU}$ is *saturated minimally unsatisfiable* iff for all clause $C \in F$ and for every literal x with $\mathrm{var}(x) \notin \mathrm{var}(C)$ the clause-set $(F \setminus C) \cup (C \cup \{x\})$ is satisfiable. The set of all saturated minimally unsatisfiable clause-sets is denoted by $\boldsymbol{\mathcal{SMU}} \subset \mathcal{MU}$. By $\boldsymbol{\mathcal{SMU}_{\delta=k}}$ we denote the set of $F \in \mathcal{SMU}$ with $\delta(F) = k$, where the *deficiency* of a clause-set F is given by $\delta(F) := c(F) - n(F)$. In [107] (generalised in [116]) it is shown that the elements of $\mathcal{SMU}_{\delta=1}$ are exactly the clause-sets introduced in [39]. The details are as follows. For rooted trees T we use $\mathbf{nds}(\boldsymbol{T})$ for the set of nodes and $\mathbf{lvs}(\boldsymbol{T}) \subseteq \mathrm{lvs}(T)$ for the set of leaves, and we set $\#\mathbf{nds}(\boldsymbol{T}) := |\mathrm{nds}(T)|$ and $\#\mathbf{lvs}(\boldsymbol{T}) := |\mathrm{lvs}(T)|$. In our context, the nodes of rooted trees are just determined by their positions, and do not have names themselves. Another useful notation for a tree T and a node w is $\boldsymbol{T_w}$, which is the sub-tree of T with root w; so $\mathrm{lvs}(T) = \{w \in \mathrm{nds}(T) : \#\mathrm{nds}(T_w) = 1\}$. Recall that for a full binary tree T (i.e., a tree where every non-leaf node has two children) we have $\#\mathrm{nds}(T) = 2\,\#\mathrm{lvs}(T) - 1$.

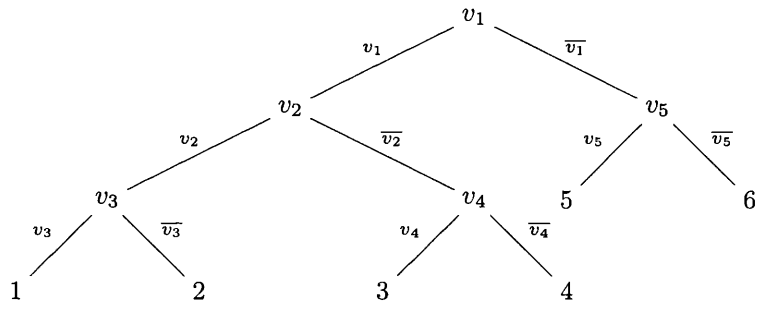
Definition 5.2.1 *Consider a full binary tree T and an injective vertex labelling $u : (\mathrm{nds}(T) \setminus \mathrm{lvs}(T)) \to \mathcal{VA}$ for the inner nodes; the set of all such pairs is denoted by $\boldsymbol{\mathcal{T}_1}$. The induced edge-labelling assigns to every edge from an inner node w to a child w' the literal $u(w)$ resp. $\overline{u(w)}$ for a left resp. right child. We define the* ***clause-set representation*** $\mathbf{F^1(T, u)}$ *(where "1" reminds of deficiency 1 here; see Lemma 5.2.2) to be $\mathbf{F^1(T, u)} := \{C_w : w \in \mathrm{lvs}(T)\}$, where clause $\boldsymbol{C_w}$ consists of all the literals (i.e., edge-labels) on the path from the root of T to w.*

By Lemma C.5 in [107]:

Lemma 5.2.2 $\mathrm{F}^1 : \mathcal{T}_1 \to \mathcal{SMU}_{\delta=1}$ *is a bijection.*

By $\mathbf{T^1} : \mathcal{SMU}_{\delta=1} \to \mathcal{T}_1$ we denote the inversion of F^1. Typically we identify $(T, u) \in \mathcal{T}_1$ with T, and let the context determine u. So $\mathrm{T}^1(F)$ is the full binary tree, where the variable v labelling the root (for $F \neq \{\bot\}$) is the unique variable occurring in every clause of F, and the clause-sets determining the left resp. right subtree are $\langle v \to 0\rangle * F$ resp. $\langle v \to 1\rangle * F$. By $\boldsymbol{w_C}$ for $C \in F$ we denote the leaf w of $\mathrm{T}^1(F)$ such that $C_w = C$. Furthermore we identify the literals of F with the edges of $\mathrm{T}^1(F)$. Note that $c(F) = \#\mathrm{lvs}(\mathrm{T}^1(F))$ and $n(F) = \#\mathrm{nds}(\mathrm{T}^1(F)) - \#\mathrm{lvs}(\mathrm{T}^1(F))$.

Example 5.2.3 *Consider the following labelled binary tree T:*



Then $\mathrm{F}^1(T) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, \overline{v_3}\}, \{v_1, \overline{v_2}, v_4\}, \{v_1, \overline{v_2}, \overline{v_4}\}, \{\overline{v_1}, v_5\}, \{v_1, \overline{v_5}\}\}$, *where for example* $C_3 = \{v_1, \overline{v_2}, v_4\}$ *and* $w_{\{v_1, \overline{v_5}\}} = 6$.

The effect of applying a partial assignment to some element of $\mathcal{SMU}_{\delta=1}$ is easily described as follows:

Lemma 5.2.4 *Consider $F \in \mathcal{SMU}_{\delta=1}$ and a literal $x \in \mathrm{lit}(F)$; let $\varphi := \langle x \to 1\rangle$ and $F' := \varphi * F$. We have:*

1. $F' \in \mathcal{SMU}_{\delta=1}$.
2. *Let $T := \mathrm{T}^1(F)$ and $T' := \mathrm{T}^1(F')$. The tree T' is obtained from T as follows:*
   (a) *Consider the node $w \in T$ labelled with $\mathrm{var}(x)$. Let $T_x, T_{\overline{x}}$ be the two subtrees hanging at w, following the edge labelled with x resp. $\overline{x}$.*
   (b) *Now T' is obtained from T' by removing subtree T_x, and attaching $T_{\overline{x}}$ directly at position w.*

Corollary 5.2.5 $\mathcal{SMU}_{\delta=1}$ *is stable under application of partial assignments, that is, for $F \in \mathcal{SMU}_{\delta=1}$ and $\varphi \in \mathcal{PASS}$ holds $\varphi * F \in \mathcal{SMU}_{\delta=1}$.*

From Lemma 5.2.2 follows $\mathcal{SMU}_{\delta=1} \subset \mathcal{UHIT}$, where $\mathcal{HIT} \subset \mathcal{CLS}$ is the set of *hitting clause-sets*, that is, those $F \in \mathcal{CLS}$ where every two clauses clash in at least one literal, i.e., for all $C, D \in F$, $C \neq D$, we have $|C \cap \overline{D}| \geq 1$, and $\boldsymbol{\mathcal{UHIT}} := \mathcal{HIT} \cap \mathcal{USAT}$. It is well-known that $\mathcal{UHIT} \subset \mathcal{SMU}$ holds (for a proof see Lemma 2 in [117]).

5.2.2 Doping $\mathcal{SMU}_{\delta=1}$

We are interested in clause-sets which have as many sub-mps's as possible:

Definition 5.2.6 *A clause-set $F \neq \top$ is a **total mps** if* $\mathrm{mps}(F) = \mathbb{P}(F) \setminus \{\top\}$.

Every total mps is an mps.

Example 5.2.7 $\{\{a,b\},\{\overline{a},b\},\{\overline{b}\}\}$ *is a total mps, while* $\{\{a,b\},\{\overline{a}\},\{\overline{b}\}\}$ *is an mps (since minimally unsatisfiable), but not a total mps.*

By Lemma 5.1.9 and Corollary 5.1.8 we get:

Lemma 5.2.8 *A clause-set F is a total mps if and only if $F' := \varphi_{\mathrm{puc}(F)} * F$ is total mps, and $\varphi_{\mathrm{puc}(F)}$ is contraction-free for F. If F is a total mps, then thus we have $F' \in \mathcal{MU}$.*

To determine all total mps's, it remains to determine the minimally unsatisfiable total mps's. Before we can prove that these are precisely the saturated minimally unsatisfiable clause-sets of deficiency 1, we need to state a basic property of these clause-sets, which follows by definition of $\mathrm{T}^1(F)$ for $F \in \mathcal{SMU}_{\delta=1}$ (recall Subsection 5.2.1):

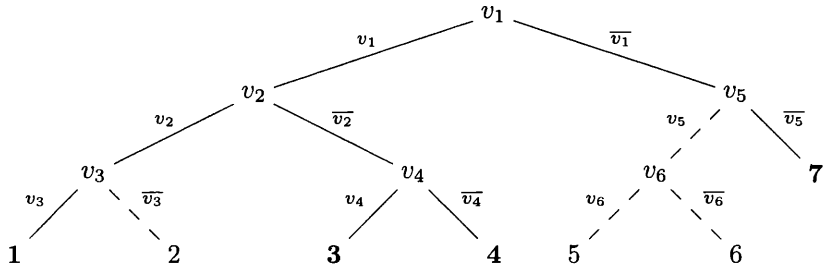
Lemma 5.2.9 *Consider $F \in \mathcal{SMU}_{\delta=1}$ and $F' \subseteq F$. Let $T := \mathrm{T}^1(F)$. The set* $\mathrm{puc}(F')$ *of pure literals of F' can be determined as follows:*

1. *Let $W_{F'} := \{w_C : C \in F'\} \subseteq \mathrm{lvs}(T)$ be the set of leaves corresponding to the clauses of F'.*
2. *For a literal $x \in \mathrm{lit}(F)$ let $w \in \mathrm{nds}(T)$ be the node labelled with $\mathrm{var}(x)$, and let T_x the subtree of w reached by x, and let $T_{\overline{x}}$ be the subtree of w reached by $\overline{x}$.*
3. *Now $x \in \mathrm{puc}(F')$ if and only if $W_{F'} \cap \mathrm{lvs}(T_x) \neq \emptyset$ and $W_{F'} \cap \mathrm{lvs}(T_{\overline{x}}) = \emptyset$.*

Example 5.2.10 *Consider the clause-set*

$$F := \{ \underbrace{\{v_1,v_2,v_3\}}_{\boldsymbol{C_1}}, \underbrace{\{v_1,v_2,\overline{v_3}\}}_{C_2}, \underbrace{\{v_1,\overline{v_2},v_4\}}_{\boldsymbol{C_3}}, \underbrace{\{v_1,\overline{v_2},\overline{v_4}\}}_{\boldsymbol{C_4}}, \underbrace{\{\overline{v_1},v_5,v_6\}}_{C_5}, \underbrace{\{\overline{v_1},v_5,\overline{v_6}\}}_{C_6}, \underbrace{\{\overline{v_1},\overline{v_5}\}}_{\boldsymbol{C_7}} \}$$

and the subset $F' := \{C_1, C_3, C_4, C_7\}$. The tree $\mathrm{T}^1(F)$ is as follows, with the dashed edges representing literals not in $\bigcup F' = \{v_1, v_2, v_3, v_4, \overline{v_1}, \overline{v_2}, \overline{v_4}, \overline{v_5}\}$:



We have $W_{F'} = \{1,3,4,7\}$ and

$$\mathrm{puc}(F') = \bigcup F' \setminus \{ \underbrace{v_2,\overline{v_2}}_{\boldsymbol{C_1},\boldsymbol{C_3}} , \underbrace{v_1,\overline{v_1}}_{\boldsymbol{C_1},\boldsymbol{C_7}} , \underbrace{v_4,\overline{v_4}}_{\boldsymbol{C_3},\boldsymbol{C_4}} \} = \{v_3, \overline{v_5}\}.$$

Now consider $x \in \mathrm{lit}(F)$:

1. *For* $x = v_3$ *holds* $\mathrm{lvs}(T_{v_3}) \cap W_{F'} = \{1\}$ *and* $T_{\overline{v_3}} \cap W_{F'} = \emptyset$, *thus* $v_3 \in \mathrm{puc}(F')$.
2. *For* $x = \overline{v_5}$ *holds* $\mathrm{lvs}(T_{\overline{v_5}}) \cap W_{F'} = \{7\}$ *and* $T_{v_5} \cap W_{F'} = \emptyset$, *thus* $\overline{v_5} \in \mathrm{puc}(F')$.
3. *Considering for example* $x = v_1$, *we have* $\mathrm{lvs}(T_{v_1}) \cap W_{F'} = \{1,3\}$ *and* $\mathrm{lvs}(T_{\overline{v_1}}) \cap W_{F'} = \{7\}$, *thus* $v_1 \notin \mathrm{puc}(F')$, *while for* $x = v_6$ *we have* $\mathrm{lvs}(T_{v_6}) \cap W_{F'} = \emptyset$ *and* $\mathrm{lvs}(T_{\overline{v_6}}) \cap W_{F'} = \emptyset$, *thus* $v_6 \notin \mathrm{puc}(F')$.

Theorem 5.2.11 *An unsatisfiable clause-set* $F \in \mathcal{USAT}$ *is a total mps if and only if* $F \in \mathcal{SMU}_{\delta=1}$.

Proof: First assume that F is a total mps. Then every two clauses $C, D \in F$, $C \neq D$, clash in exactly one literal (otherwise $\{C, D\} \notin \mathcal{MPS}$). In [109], Corollary 34, it was shown that an unsatisfiable clause-sets F has precisely one clash between any pair of different clause-sets iff $F \in \mathcal{SMU}_{\delta=1}$ holds (an alternative proof was found in [148]).[1)] Now assume $F \in \mathcal{SMU}_{\delta=1}$, and we have to show that F is a total mps. So consider $F' \in \mathbb{P}(F) \setminus \{\top\}$, and let $C := \mathrm{puc}(F)$, $\varphi := \varphi_C$. Since F' is a hitting clause-set, φ is contraction-free for F', and according to Lemma 5.1.9 it remains to show that $F'' := \varphi * F'$ is unsatisfiable (recall that hitting clause-sets are irredundant). Assume that F'' is satisfiable, and consider a partial assignment ψ with $\psi * F'' = \top$ and $\mathrm{var}(\psi) \cap \mathrm{var}(\varphi) = \emptyset$. We show that then $\varphi \cup \psi$ would be a satisfying assignment for F, contradicting the assumption. To this end it suffices to show that for all $D \in F \setminus F'$ holds $\overline{C} \cap D \neq \emptyset$. Consider $T := \mathrm{T}^1(F)$, and let $W_{F'}$ be defined as in Lemma 5.2.9. Starting from the leaf w_D, let w be the first node on the path to the root of T such that one of the two subtrees of w contains a leaf of $W_{F'}$. Let $\overline{x}$ be the literal at w on the path to w_D. So by Lemma 5.2.9 we have $x \in C$, while by definition $\overline{x} \in D$. □

The proof of Theorem 5.2.11 actually shows that for $F \in \mathcal{USAT}$ already from all 2-element subsets of F being mps's follows $F \in \mathcal{SMU}_{\delta=1}$. We are turning now our attention to a closer understanding of the prime implicates C of doped $F \in \mathcal{SMU}_{\delta=1}$. We start with their identification with non-empty sub-clause-sets F' of $\mathrm{D}(F)$:

Lemma 5.2.12 *Consider a clause-set* $F \in \mathcal{SMU}_{\delta=1}$. *By Theorem 5.2.11 each non-empty subset of* F *yields a minimal premise set. Thus by Theorem 5.1.18 we have:*

1. $\mathrm{prc}_0(\mathrm{D}(F)) = \{\mathrm{puc}(F') \mid \top \neq F' \subseteq \mathrm{D}(F)\}$.
2. $|\mathrm{prc}_0(\mathrm{D}(F))| = 2^{\mathrm{c}(F)} - 1$.

The main result of [148] is the stronger result that the clause-sets $F \in \mathcal{CLS}$ with $|\mathrm{prc}_0(F)| = 2^{\mathrm{c}(F)} - 1$ are precisely the clause-sets $\mathrm{D}(F)$ for $F \in \mathcal{SMU}_{\delta=1}$ when allowing to replace the single doping variable of a clause by any non-empty set of new (pure) literals. Back to the task at hand: Since the clauses of $\mathrm{D}(F)$ can be identified with leaves of the tree $\mathrm{T}^1(F)$, we obtain a bijection between non-empty sets V of leaves of the tree $\mathrm{T}^1(F)$ and prime implicates of $\mathrm{D}(F)$:

Definition 5.2.13 *For* $F \in \mathcal{SMU}_{\delta=1}$ *and* $\emptyset \neq V \subseteq \mathrm{lvs}(\mathrm{T}^1(F))$ *the clause* $\boldsymbol{C_V}$ *is the prime implicate* $\mathrm{puc}(\{C_w \in F \mid w \in V\})$ *of* $\mathrm{D}(F)$ *according to Lemma 5.2.12. For* $w \in \mathrm{lvs}(\mathrm{T}^1(F))$ *we furthermore set* $\boldsymbol{u_w} := u_{C_w}$.

By Lemma 5.2.12:

Lemma 5.2.14 *For* $F \in \mathcal{SMU}_{\delta=1}$ *holds* $\mathrm{prc}_0(\mathrm{D}(F)) = \{C_V \mid \emptyset \neq V \subseteq \mathrm{lvs}(\mathrm{T}^1(F))\}$.

[1)] In [109] the notation "$\mathcal{UHIT}$" was used to denote "uniform hitting clause-sets", which is now more appropriately called "(conflict-)regular hitting clause-sets", while "U" now stands for "unsatisfiable".

How precisely from $V \subseteq \mathrm{lvs}(\mathrm{T}^1(F))$ the prime implicate C_V is constructed shows the following lemma:

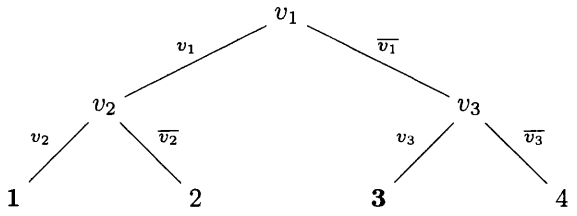
Lemma 5.2.15 *Consider $F \in \mathcal{SMU}_{\delta=1}$ and $\emptyset \neq V \subseteq \mathrm{lvs}(\mathrm{T}^1(F))$. We have $C_V = U_V \cup P_V$, $U_V \cap P_V = \emptyset$, where*

1. *$U_V := \{u_w \mid w \in V\}$, and*
2. *$P_V := \mathrm{puc}(F')$ for $F' := \{C_w : w \in V\}$ as given in Lemma 5.2.9, that is, P_V is the set of literals x such that $V \cap \mathrm{lvs}(T_x) \neq \emptyset$ and $V \cap \mathrm{lvs}(T_{\overline{x}}) = \emptyset$.*

Example 5.2.16 *Consider the clause-set*

$$F := \{\{v_1, v_2\}, \{v_1, \overline{v_2}\}, \{\overline{v_1}, v_3\}, \{\overline{v_1}, \overline{v_3}\}\} \in \mathcal{SMU}_{\delta=1}$$

corresponding to the tree



with the doped clause-set

$$\mathrm{D}(F) = \{\{v_1, v_2, u_1\}, \{v_1, \overline{v_2}, u_2\}, \{\overline{v_1}, v_3, u_3\}, \{\overline{v_1}, \overline{v_3}, u_4\}\}.$$

Now consider the set $V := \{1, 3\}$. According to Definition 5.2.13 we have that $C_V = \mathrm{puc}(\{\{v_1, v_2, u_1\}, \{\overline{v_1}, v_3, u_3\}\}$ $\{v_2, v_3, u_1, u_3\}$. By Lemma 5.2.15 we have that $C_V = U_V \cup P_V$, where $U_V = \{u_1, u_3\}$ and $P_V = \mathrm{puc}(\{\{v_1, v_2\}, \{\overline{v_1}, v_3\}\} = \{v_2, v_3\}$. Note that for both $x \in \{v_2, v_3\} = P_V$ we have that $\mathrm{lvs}(T_x) \cap V \neq \emptyset$ and $\mathrm{lvs}(T_{\overline{x}}) \cap V = \emptyset$, but we do not have this for $x \in \mathrm{lit}(F) \setminus \{v_2, v_3\}$.

The hardness of F as well as $\mathrm{D}(F)$ is the Horton-Strahler number of $\mathrm{T}^1(F)$:

Lemma 5.2.17 *Consider $F \in \mathcal{SMU}_{\delta=1}$, and let $k := \mathrm{hs}(\mathrm{T}^1(F))$. Then we have $\mathrm{hd}(F) = \mathrm{hd}(\mathrm{D}(F)) = k$.*

Proof: Let $T := \mathrm{T}^1(F)$. First we show $\mathrm{hd}(F) = k$. We have $\mathrm{hd}(F) \leq k$, since T is by definition of $F = \mathrm{F}^1(T)$ already a resolution tree (when extending the labelling of leaves to all nodes), deriving $\bot$ from F. To show $\mathrm{hd}(F) \geq k$, we use Lemma 3.3.7 with $\mathcal{C} := \mathcal{SMU}_{\delta=1}$ and $h(F) := \mathrm{hs}(\mathrm{T}^1(F))$. Based on Lemma 5.2.4, we consider the effect on the Horton-Strahler number of assigning a truth value to one variable $v \in \mathrm{var}(F)$. Let $w \in \mathrm{nds}(T)$ be the (inner) node labelled with v, and let T_0^w, T_1^w be the left resp. right subtree hanging at w. Now the effect of assigning $\varepsilon \in \{0, 1\}$ to v is to replace T_w with T_ε^w. Let T_ε be the (whole) tree obtained by assigning ε to v, that is, $T_\varepsilon := \mathrm{T}^1(\langle v \to \varepsilon \rangle * F)$. If $\mathrm{hs}(T_0^w) = \mathrm{hs}(T_1^w)$, then we have $\mathrm{hs}(T_\varepsilon) \geq k - 1$, since at most one increase of the Horton-Strahler number for subtrees is missed out now. Otherwise we have $\mathrm{hs}(T_0) = \mathrm{hs}(T)$ or $\mathrm{hs}(T_1) = \mathrm{hs}(T)$, since removal of the subtree with the smaller Horton-Strahler number has no influence on the Horton-Strahler number of the whole tree. So altogether Lemma 3.3.7 is applicable, which concludes the proof of $\mathrm{hd}(F) = k$.

For showing $\mathrm{hd}(\mathrm{D}(F)) = k$ we use Lemma 5.1.20: so consider $F' \subseteq F$ and $\varphi \in \mathcal{PASS}$ with $\varphi * F' \in \mathcal{USAT}$, let $F'' := \varphi * F'$, and we have to show $\mathrm{hd}(F'') \leq k$. W.l.o.g. $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F')$.

By Corollary 5.2.5 we have that $\varphi * F \in \mathcal{SMU}_{\delta=1}$, and thus $\varphi * F = F''$ must hold, and $\mathrm{hd}(F'') = \mathrm{hs}(\mathrm{T}^1(F''))$ (by the first part). By Lemma 5.2.4, $\mathrm{T}^1(F'')$ results from T by a sequence of removing subtrees, and it is easy to see, that thus $\mathrm{hs}(\mathrm{T}^1(F'')) \leq k$ holds. □

We summarise what we have learned about $\mathrm{D}(F)$ for $F \in \mathcal{SMU}_{\delta=1}$:

Theorem 5.2.18 *Consider $F \in \mathcal{SMU}_{\delta=1}$.*

1. *For each clause-set F' equivalent to $\mathrm{D}(F)$ there is an injection $i : \mathrm{D}(F) \to F'$ with $\forall C \in \mathrm{D}(F) : C \subseteq i(C)$ (by Lemma 5.1.15).*
2. *$\mathrm{D}(F)$ is a total mps (by Lemma 5.2.8 together with Theorem 5.2.11).*
3. *The prime implicates of $\mathrm{D}(F)$ are given by Lemmas 5.2.14, 5.2.15.*
4. *$\mathrm{hd}(\mathrm{D}(F)) = \mathrm{hs}(\mathrm{T}^1(F))$ (by Lemma 5.2.17).*

5.3 Lower bounds

This section proves the main result of this chapter, Theorem 5.3.13, which exhibits for every $k \geq 0$ sequences $(F_n^{k+1})_{n\in\mathbb{N}}$ of small clause-sets of hardness $k+1$, where every equivalent clause-set of hardness k (indeed of w-hardness k) is of exponential size. In this way we show that the $\mathcal{UC}_k$ hierarchy is useful, i.e., equivalent clause-sets with higher hardness can be substantially shorter. These F_n^{k+1} are doped versions of clause-sets from $\mathcal{SMU}_{\delta=1}$ (recall Theorem 5.2.18), which are "extremal", that is, their underlying trees $\mathrm{T}^1(F_n^{k+1})$ are for given Horton-Strahler number $k+1$ and height n as large as possible.

The organisation of this section is as follows: In Subsection 5.3.1 the main tool for showing size-lower-bounds for equivalent clause-sets of a given (w-)hardness is established in Theorem 5.3.4. Subsection 5.3.2 introduces the "extremal trees". Subsection 5.3.3 shows the main lower bound in Theorem 5.3.12, and applies it to show the separation Theorem 5.3.13.

5.3.1 Trigger hypergraphs

A hypergraph is a pair $G = (V, E)$, where V is a set (of "vertices") and $E \subseteq \mathbb{P}(V)$ (the set of hyperedges), where one uses $V(G) := V$ and $E(G) := E$. A *transversal* of a hypergraph G is a set $T \subseteq V(G)$ such that for all $E \in E(G)$ holds $T \cap E \neq \emptyset$. The minimum size of a transversal is denoted by $\boldsymbol{\tau(G)}$, the **transversal number**. And let $\boldsymbol{\nu(G)}$ be the **matching number** of G, the maximum number of pairwise disjoint hyperedges. Obviously we have $\tau(G) \geq \nu(G)$ for all hypergraphs G.

Definition 5.3.1 *Consider $k \in \mathbb{N}_0$ and $F \in \mathcal{CLS}$. The **trigger hypergraph** $T_k(F)$ is the hypergraph with the prime implicates of F as its vertices, and for every prime implicate C of F a hyperedge E_C^k. The hyperedge E_C^k contains all prime implicates $C' \in \mathrm{prc}_0(F)$ which are not satisfied by φ_C and yield a clause of size at most k under φ_C. That is,*

1. *$V(T_k(F)) := \mathrm{prc}_0(F)$, and*
2. *$E(T_k(F)) := \{E_C^k \mid C \in \mathrm{prc}_0(F)\}$,*

where $E_C^k := \{C' \in \mathrm{prc}_0(F) \mid C' \cap \overline{C} = \emptyset \wedge |C' \setminus C| \leq k\}$.

Note that the trigger hypergraph of $F \in \mathcal{CLS}$ depends only on the underlying boolean function of F, and thus for every equivalent F' we have $T_k(F') = T_k(F)$.

Example 5.3.2 *Consider the clause-set*

$$F := \{ \underbrace{\{v_1, \overline{v_3}, \overline{v_4}\}}_{C_1}, \underbrace{\{v_2, v_3, \overline{v_4}\}}_{C_2}, \underbrace{\{v_2, \overline{v_3}, v_4\}}_{C_3}, \underbrace{\{\overline{v_2}, v_3, v_4\}}_{C_4}, \underbrace{\{v_1, v_3, v_4\}}_{C_5}, \underbrace{\{v_1, v_2\}}_{C_6} \}.$$

As shown in Example 8.2 of [77, 78] we have $\mathrm{prc}_0(F) = F$. *The trigger hypergraph* $T_0(F)$ *is (as always) the hypergraph with all singleton sets, i.e.,* $E(T_0(F)) = \{ \{C_1\}, \ldots, \{C_6\} \}$. *The hypergraphs* $T_k(F)$ *for* $k \in \{1, 2\}$ *are represented by Figures 5.1, 5.2.*

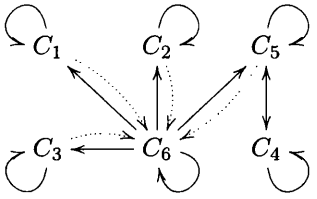


Figure 5.1: $T_1(F)$

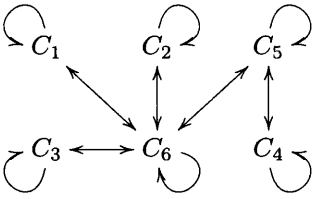


Figure 5.2: $T_2(F)$

To interpret the diagrams:

1. *An arrow from a clause C to a clause D represents that $C \in E^k_D$.*
2. *A dotted arrow from C to D represents that $|D \setminus C| > k$ (so $C \notin E^k_D$), but $C \cap \overline{D} = \emptyset$, and thus for some large enough $k' > k$ we will have $C \in E^{k'}_D$.*
3. *No arrow between C and D indicates that $C \cap \overline{D} \neq \emptyset$ (i.e., for all k' we have $C \notin E^k_D$ and $D \notin E^k_C$).*
4. *The size of a hyperedge E^k_D is the in-degree of the vertex D.*

Consider $E^1_{C_6} = \{C_6\}$ *and* $E^2_{C_6} = \{C_1, C_2, C_3, C_5, C_6\}$. *As we will see in Lemma 5.3.3, therefore every* $F' \subseteq F$ *equivalent to* F *such that* $F' \in \mathcal{UC}_1$ *must have* $C_6 \in F'$. *However,* $E^2_{C_6}$ *contains more clauses than* $E^1_{C_6}$, *and for example* $F \setminus \{C_6\} \in \mathcal{UC}_2 \setminus \mathcal{UC}_1$ *as shown in Example 4.5.3. Using the above diagrammatic notation, we can also see that for all* $k' \geq 2$ *we have* $T_{k'}(F) = T_2(F)$, *as there are no dotted lines for* $T_2(F)$ *(i.e., no clauses C and D such that* $|D \setminus C| > 2$ *but* $C \cap \overline{D} = \emptyset$*).*

Lemma 5.3.3 *Consider* $k \in \mathbb{N}_0$ *and* $F \in \mathcal{CLS}$ *with* $\mathrm{whd}(F) \leq k$. *Then there is a clause-set* F' *such that*

1. $F' \subseteq \mathrm{prc}_0(F)$ *and* F' *is equivalent to* F;
2. *there is an injection* $i : F' \to F$ *such that* $\forall C \in F' : C \subseteq i(C)$;
3. $\mathrm{whd}(F') \leq k$;
4. F' *is a transversal of* $T_k(F)$.

Proof: Obtain F' from F by choosing for every $C \in F$ some $C' \in \mathrm{prc}_0(F)$ with $C' \subseteq C$. Then the first two properties are obvious, while Property 3 follows from Part 1 of Lemma 6.1 in [110]. Assume that F' is not a transversal of $T_k(F)$, that is, there is $C \in \mathrm{prc}_0(F)$ with $F' \cap E^k_C = \emptyset$. Then $\varphi_C * F' \in \mathcal{USAT}$, but every clause has length strictly greater than k, and thus k-resolution does not derive $\bot$ from $\varphi_C * F'$, contradicting $\mathrm{whd}(F') \leq k$. □

Directly from Lemma 5.3.3 follows:

Theorem 5.3.4 *For $k \in \mathbb{N}_0$ and $F \in \mathcal{WC}_k$ we have $c(F) \geq \tau(T_k(F))$.*

5.3.2 Extremal trees

For a given hardness $k \geq 1$ we need to construct (full binary) trees which are as large as possible; this is achieved by specifying the height, and using trees which are "filled up" completely for the given parameter values:

Definition 5.3.5 *A pair $(k,h) \in \mathbb{N}_0^2$ with $h \geq k$ and $k = 0 \Rightarrow h = 0$ is called an **allowed parameter pair**. For an allowed parameter pair (k,h) a full binary tree T is called an **extremal tree of Horton-Strahler number k and height h** if*

1. $\mathrm{hs}(T) = k$, $\mathrm{ht}(T) = h$;
2. *for all T' with $\mathrm{hs}(T') \leq k$ and $\mathrm{ht}(T') \leq h$ we have $\mathrm{nds}(T') \leq \mathrm{nds}(T)$.*

We denote the set of all extremal trees with Horton-Strahler number k and height h by $\mathbf{HS}(k,h)$.

Note that for allowed parameter pairs (k,h) we have $k = 0 \Leftrightarrow h = 0$. Extremal trees are easily characterised and constructed as follows:

1. $\mathrm{HS}(0,0)$ contains only the trivial tree (with one node).
2. $\mathrm{HS}(1,h)$ for $h \in \mathbb{N}$ consists exactly of the full binary trees T with $\mathrm{hs}(T) = 1$ and $\mathrm{ht}(T) = h$, which can also be characterised as those full binary trees T with $\mathrm{ht}(T) = h$ such that every node has at least one child which is a leaf.
3. For $k \geq 2$ and $h \geq k$ we have $T \in \mathrm{HS}(k,h)$ iff T has the left subtree T_0 and the right subtree T_1, and there is $\varepsilon \in \{0,1\}$ with $T_\varepsilon \in \mathrm{HS}(k-1, h-1)$ and $T_{1-\varepsilon} \in \mathrm{HS}(\min(k, h-1), h-1)$.

Lemma 5.3.6 *For all allowed parameter pairs (k,h) we have $\mathrm{HS}(k,h) \neq \emptyset$.*

The unique elements of $\mathrm{HS}(k,k)$ for $k \in \mathbb{N}_0$ are the perfect binary trees of height k, which are the smallest binary trees of Horton-Strahler number k.

Lemma 5.3.7 *For an allowed parameter pair (k,h) and for $T \in \mathrm{HS}(k,h)$ we have $\#\mathrm{lvs}(T) = \boldsymbol{\alpha(k,h)} := \sum_{i=0}^{k} \binom{h}{i}$. We have $\alpha(k,h) = \Theta(h^k)$ for fixed k.*

Proof: For $k \leq 1$ we have $\alpha(0,0) = 1$ and $\alpha(1,h) = 1 + h$. which are obviously correct. Now consider $k \geq 2$. By induction hypothesis we get

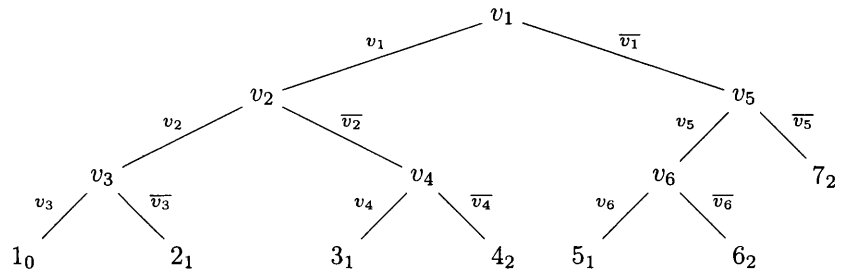
$$\#\mathrm{nds}(T) = \alpha(k-1, h-1) + \alpha(\min(k, h-1), h-1).$$

If $h = k$, then $\alpha(k,h) = 2^k$ (for all k), and we get $\#\mathrm{nds}(T) = \alpha(k-1,k-1) + \alpha(k-1,k-1) = 2 \cdot 2^{k-1} = 2^k = \alpha(k,k)$. Otherwise we have

$$\#\mathrm{nds}(T) = \alpha(k-1,h-1) + \alpha(k,h-1) =$$
$$\sum_{i=0}^{k-1} \binom{h-1}{i} + \sum_{i=0}^{k} \binom{h-1}{i} = \binom{h-1}{0} + \sum_{i=1}^{k} \binom{h-1}{i-1} + \binom{h-1}{i} =$$
$$\binom{h-1}{0} + \sum_{i=1}^{k} \binom{h}{i} = \sum_{i=0}^{k} \binom{h}{i} = \alpha(k,h).$$

□

Example 5.3.8 *Consider the following labelled binary tree T:*



Applying the recursive construction/characterisation we see $T \in \mathrm{HS}(2,3)$. By simple counting we see that T has 7 leaves, in agreement with Lemma 5.3.7, i.e., $\sum_{j=0}^{2} \binom{3}{j} = \binom{3}{0} + \binom{3}{1} + \binom{3}{2} = 1 + 3 + 3 = 7$. Assuming that of the two subtrees at an inner node, the left subtree has Horton-Strahler numbers at least as big as the right subtree, the idea is that the sum runs over the number *j* of right turns in a path from the root to the leaves. *In the above tree T, the number of right turns is indicated as an index to the leaf-name. If the Horton-Strahler number is k, with at most k right-turns we must be able to reach every leaf.*

We summarise the additional knowledge over Theorem 5.2.18 (using additionally that most leaves of $T \in \mathrm{HS}(k,h)$ have depth precisely h):

Lemma 5.3.9 *Consider an allowed parameter pair (k,h) and $T \in \mathrm{HS}(k,h)$, and let $F := \mathrm{F}^1(T)$.*

1. $n(\mathrm{D}(F)) = 2 \cdot \alpha(k,h) - 1$ *($= \Theta(h^k)$ for fixed k).*
2. $c(\mathrm{D}(F)) = \alpha(k,h)$ *($= \Theta(h^k)$ for fixed k).*
3. $\ell(\mathrm{D}(F)) \leq h \cdot \alpha(k,h)$ *($= \Theta(h^{k+1})$ for fixed k).*
4. $\mathrm{D}(F) \in \mathcal{UC}_k \setminus \mathcal{UC}_{k-1}$ *(for $k \geq 1$).*

In Theorem 5.3.13 we will see that these $\mathrm{D}(F)$ from Lemma 5.3.9 do not have short equivalent clause-sets of hardness $k-1$.

5.3.3 Exponential lower bounds on "better" equivalent clause-sets

The *depth* of a node w in a rooted tree T, denoted by $\mathbf{d_T(w)} \in \mathbb{N}_0$, is the length of the path from the root of T to w. Recall that two sets A, B are *incomparable* iff $A \not\subseteq B$ and $B \not\subseteq A$. Furthermore we call two sets A, B *incomparable on a set C* if the sets $A \cap C$ and $B \cap C$ are incomparable.

Definition 5.3.10 *Consider a full binary tree T, where every leaf has depth at least $k+1$. Consider furthermore $\emptyset \neq V, V' \subseteq \mathrm{lvs}(T)$. Then V and V' are* ***depth-k-incomparable for T*** *if V and V' are incomparable on $\mathrm{lvs}(T_w)$ for all $w \in \mathrm{nds}(T)$ with $\mathrm{d}_T(w) = k$.*

Note that for all allowed parameter pairs (k,h) and $T \in \mathrm{HS}(k,h)$ every leaf has depth at least k.

Lemma 5.3.11 *Consider $k \in \mathbb{N}_0$, $T \in \mathcal{T}_1$, and $\emptyset \neq V_0, V_1 \subseteq \mathrm{lvs}(T)$ which are depth-k-incomparable for T. Let $F := \mathrm{F}^1(T)$ and consider $T_k(F)$ (recall Definition 5.3.1). Then the hyperedges $E^k_{C_{V_0}}$, $E^k_{C_{V_1}}$ are disjoint (recall Definition 5.2.13).*

Proof: Assume that $E^k_{C_{V_0}}$, $E^k_{C_{V_1}}$ are not disjoint; thus there is $\emptyset \neq V \subseteq \mathrm{lvs}(T)$ with $C_V \in E^k_{C_{V_0}} \cap E^k_{C_{V_1}}$. We will show that there is $\varepsilon \in \{0,1\}$ with $|C_V \setminus C_{V_\varepsilon}| \geq k+1$, which contradicts the definition of $T_k(F)$.

Since $V \neq \emptyset$, there is $w \in V$. Consider the first $k+1$ nodes $w_1, \ldots, w_{k+1}$ on the path from the root to w. Let w'_i be the child of w_{i-1} different from w_i for $i \in \{2, \ldots, k+1\}$, and let $T_i := T_{w'_{i+1}}$ for $i \in \{1, \ldots, k\}$, while $T_{k+1} := T_{w_{k+1}}$; see Figure 5.3. We show that each of $T_1, \ldots, T_{k+1}$ contributes at least two unique literals to $|C_V \setminus C_{V_0}| + |C_V \setminus C_{V_1}|$, so that we get $|C_V \setminus C_{V_0}| + |C_V \setminus C_{V_1}| \geq (k+1) \cdot 2$, from which follows that there is $\varepsilon \in \{0,1\}$ with $|C_V \setminus C_{V_\varepsilon}| \geq k+1$ as claimed.

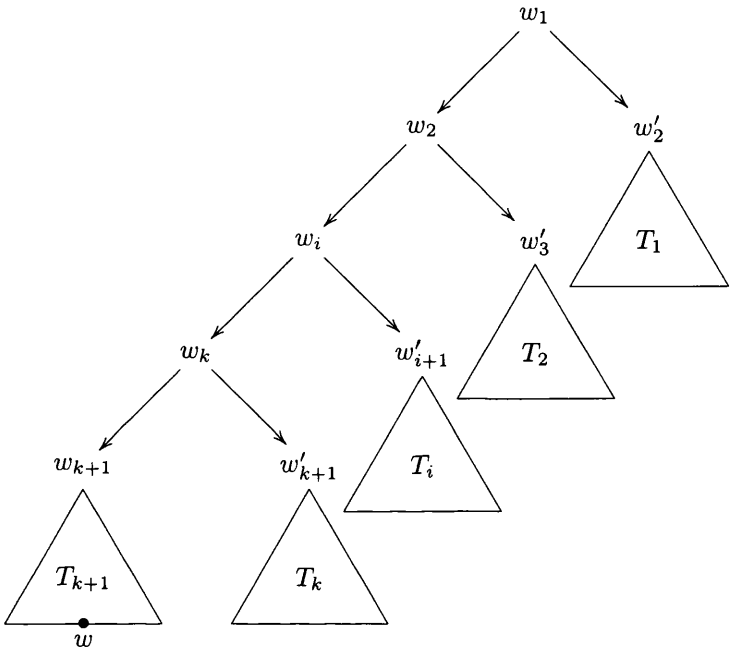


Figure 5.3: Illustration of sub-trees $T_1, \ldots, T_{k+1}$.

Due to the depth-k-incomparability of V, V', for each $i \in \{1, \ldots, k+1\}$ and each $\varepsilon \in \{0,1\}$ there are nodes v_i^ε with $v_i^\varepsilon \in (\mathrm{lvs}(T_i) \cap V_\varepsilon) \setminus V_{\overline{\varepsilon}}$. We have two cases now:

I If $v_i^\varepsilon \in V$, then $u_{v_i^\varepsilon} \in C_V \setminus C_{V_{\overline{\varepsilon}}}$.

II If $v_i^\varepsilon \notin V$, then consider the first node v on the path from v_i^ε to the root such that for the other child v' of v, not on that path to the root, holds $\mathrm{lvs}(T_{v'}) \cap V \neq \emptyset$: now for the literal x labelling the edge from v to v' we have $x \in C_V \setminus C_{V_\varepsilon}$. Note that v is below or equal to w_i (due to $w \in V$).

For each $\varepsilon \in \{0,1\}$, the literals collected in $C_V \setminus C_{V_\varepsilon}$ from these $k+1$ sources do not coincide, due to the pairwise node-disjointness of the trees $T_1, \ldots, T_{k+1}$. □

Theorem 5.3.12 *Consider $k \in \mathbb{N}_0$, $h \geq k+1$, and $T \in \mathrm{HS}(k+1,h)$; let $F := \mathrm{D}(\mathrm{F}^1(T))$ and $m := \alpha(1, h-k) = 1+h-k$. We have*

$$\nu(T_k(F)) \geq \binom{m}{\lfloor \frac{m}{2} \rfloor} > \frac{1}{\sqrt{2}} \frac{2^m}{\sqrt{m}} = \Theta(\frac{2^h}{\sqrt{h}}),$$

where the second inequality assumes $h \geq k+5$, while the Θ-estimation assumes fixed k.

Proof: For every $S \subseteq \mathbb{P}(\mathrm{lvs}(T))$ with $\emptyset \notin S$, such that every two different elements of S are depth-k-incomparable for T, we have $\nu(T_k(F)) \geq |S|$ by Lemma 5.3.11. We can actually determine the maximal size of such an S, which is $M := \binom{m}{m'}$, where $m' := \lfloor \frac{m}{2} \rfloor$, as follows. Let $\mathbb{T} := \{T_w : w \in \mathrm{nds}(T) \wedge \mathrm{d}_T(w) = k\}$; note that for $T', T'' \in \mathbb{T}$ with $T' \neq T''$ we have $\mathrm{lvs}(T') \cap \mathrm{lvs}(T'') = \emptyset$. Choose $T_0 \in \mathbb{T}$ with minimal $\#\mathrm{lvs}(T_0)$; by Lemma 5.3.7 we have $\#\mathrm{lvs}(T_0) = m$. Let $S_0 := \{V \cap \mathrm{lvs}(T_0) : V \in S\}$. Then S_0 is an antichain (i.e., the elements of S_0 are pairwise incomparable) and $|S_0| = |S|$. By Sperner's Theorem ([152]) holds $|S_0| \leq M$, and this upper bound M is realised, just observing the antichain-condition, by choosing for S_0 the set $\binom{\mathrm{lvs}(T_0)}{m'}$ of subsets of $\mathrm{lvs}(T_0)$ of size m'. This construction of S_0 can be extended to a construction of S (of the same size) by choosing for each $T' \in \mathbb{T}$ an injection $j_{T'} : S_0 \to \binom{\mathrm{lvs}(T')}{m'}$ and defining $S := \{\bigcup_{T' \in \mathbb{T}} j_{T'}(V)\}_{V \in S_0}$. The given estimation of M follows from Stirling's approximation. $\square$

We are now able to state the main result of this chapter, proving Conjecture 1.1 from [77, 78] that $\mathcal{UC}_k$, and indeed also $\mathcal{WC}_k$, is a proper hierarchy of boolean functions regarding polysize representations without new variables (see Subsection 6.1 for a discussion of "representations" in general):

Theorem 5.3.13 *Consider $k \in \mathbb{N}_0$. For $h \geq k+1$ choose one $T_h \in \mathrm{HS}(k+1,h)$ (note there is up to left-right swaps (of children in T_h) exactly one element in $\mathrm{HS}(k+1,h)$), and let $F_h := \mathrm{D}(\mathrm{F}^1(T_h))$. Consider the sequence $(F_h)_{h \geq k+1}$.*

1. *By Lemma 5.3.9 we have $n(F_h) = \Theta(h^{k+1})$ as well as $c(F_h) = \Theta(h^{k+1})$, and $F_h \in \mathcal{UC}_{k+1}$.*
2. *Consider a sequence $(F'_h)_{h \geq k+1}$ of clause-sets with F'_h equivalent to F_h, such that $F'_h \in \mathcal{WC}_k$. By Theorems 5.3.12, 5.3.4 we have $c(F'_h) = \Omega(\frac{2^h}{\sqrt{h}})$.*

The sequence $(F_h)_{h \geq k+1}$ from Theorem 5.3.13 then acts as a witness for the separation of the $\mathcal{UC}_k$ and $\mathcal{WC}_k$ hierarchies:

Theorem 5.3.14 *For all $k \in \mathbb{N}_0$ there are families of boolean functions with poly-size representations in $\mathcal{UC}_{k+1}$ but only exponential-size representations in $\mathcal{WC}_k$.*

We conjecture that Theorem 5.3.13 can be strengthened by including the PC-hierarchy in the following way[2)]:

Conjecture 5.3.15 *For every $k \in \mathbb{N}_0$ there exists a sequence $(F_n)_{n \in \mathbb{N}}$ of clause-sets in $\mathcal{PC}_{k+1}$, where for convenience we assume $n(F_n) = n$ for all n, such that $(\ell(F_n))_{n \in \mathbb{N}}$ is polynomially bounded, and such that for every sequence $(F'_n)_{n \in \mathbb{N}}$ in $\mathcal{WC}_k$, where for all $n \in \mathbb{N}$ holds that F'_n is equivalent to F_n, the sequence $(\ell(F'_n))_{n \in \mathbb{N}}$ is not polynomially bounded.*

[2)] Theorem 5.3.13 only separates $\mathcal{UC}_k$ from $\mathcal{UC}_{k+1}$ and so $\mathcal{PC}_k$ from $\mathcal{PC}_{k+2}$, while it is still possible that $\mathcal{PC}_k$ and $\mathcal{PC}_{k+1}$ can not be separated.

Chapter 6

Analysing the Tseitin translation

Chapters 4 and 5 defined the $\mathcal{UC}_k$, $\mathcal{PC}_k$ and $\mathcal{WC}_k$ hierarchies and answered the fundamental questions regarding their properties and relations to existing classes. Attention is now turned to questions regarding the use of these hierarchies as "target-classes" for good representations of boolean functions in satisfiability. In Chapters 4 and 5 the focus was on representations without new variables, allowing strong properties to be proven in this simpler, more restricted setting, where optimisation over all (equivalent) representations is possible. However, a large part of the power of CNF comes from the ability to use new variables to represent and replace repeated structures. One of the most well-known methods for representing boolean functions as a CNF using new variables is the Tseitin translation from [156]. The Tseitin translation converts arbitrary propositional formulae (or even circuits) to CNF by inductively replacing each sub-formulae with a new variable and introducing clauses defining the semantics of these new variables. This new-variable translation is investigated in this chapter for the restricted cases of DNF formulae (see the "canonical translation" in Section 6.2) and single XOR equations (see Section 6.4).

First, in Section 6.1, the notion of CNF representation is introduced (i.e., now allowing new variables) and a relaxed "relative" notion of hardness introduced, now requiring efficient clausal entailment only for implicates in the original variables of the boolean function. The concept of "maintaining arc-consistency via UCP" for a boolean function f then relates exactly to the translation having p-hardness 1 relative to f. In Sections 6.2 and 6.3, two variants of the Tseitin translation are introduced and applied to sub-classes of DNF clause-sets yielding CNF clause-sets with different hardness measures under the relative and "absolute" measures, and offering potential insight into differences in performance when using only one "polarity" of clauses in the Tseitin translation, i.e., with equivalence clauses (i.e., $v_F \leftrightarrow F$ for a sub-formula F) compared to using implication clauses (i.e., using $v_F \rightarrow F$ for a sub-formula F where possible), as discussed in [131, 93, 55]. Lastly, the Tseitin translation on XOR clauses is analysed and shown to fit into $\mathcal{UC}_1$ (i.e., it has absolute hardness 1), while for multiple XOR clauses we see that this simple translation can have unbounded hardness for both relative and absolute measures. In this way, examples are shown of the use of the theoretical framework for proving hardness bounds (i.e., membership in $\mathcal{UC}_k$ for some bounded k) for concrete representations and evidence is given that various existing translations actually satisfy the stronger absolute condition, rather than the typical weaker relative notions from the literature.

6.1 CNF representations

The most general notion useful in the SAT-context of a CNF-representation allowing new variables seems to allow existentially quantified new variables (see e.g., [31]), which yields the following basic definition:

Definition 6.1.1 *A **CNF-representation** of $F \in \mathcal{CLS}$ (as CNF) is a clause-set $F' \in \mathcal{CLS}$ with $\mathrm{var}(F) \subseteq \mathrm{var}(F')$ such that the satisfying assignments of F' (as CNF) projected to $\mathrm{var}(F)$ are precisely the satisfying assignments of F.*

Example 6.1.2 *Consider $F := \{\{a,b\}\}$. Then $F' := F \cup \{\{v,a\}\}$ is a CNF-representation of F, since the satisfying assignments of F can be extended to satisfying assignments of F' by assigning $v \to 1$, while no new satisfying assignments are present, since F' is a superset of F. Also $F \cup \{\{v\}\}$ is a CNF-representation of F, but $F \cup \{\{a\}\}$ is not, since the satisfying assignment $\langle a \to 0, b \to 1\rangle$ of F would be lost. Also $\{\{v,a,b\}\}$ is not a CNF-representation of F, since here now we would obtain a new satisfying assignment for F, namely $\langle a, b \to 0\rangle$.*

The CNF-representations F' of F *without new variables*, that is, with $\mathrm{var}(F') = \mathrm{var}(F)$, are precisely the clause-sets F' equivalent to F with $\mathrm{var}(F') = \mathrm{var}(F)$. We have conjectured in [77, 78] (Conjecture 9.4) that Theorem 5.3.13 (and Conjecture 5.3.15) also holds when allowing new variables, which in this context we can rephrase as follows, also extending the conjecture by including $\mathcal{WC}_k$ (see Conjecture 8.1.2 for a further strengthening):

Conjecture 6.1.3 *For every $k \in \mathbb{N}_0$ there exists a sequence $(F_n)_{n\in\mathbb{N}}$ of clause-sets, such that there is a sequence $(F'_n)_{n\in\mathbb{N}}$, where*

- *each F'_n is a CNF-representation of F_n,*
- *$\ell(F'_n)$ is polynomial in n,*
- *and we have $F'_n \in \mathcal{PC}_{k+1}$,*

but where there is no such sequence $(F''_n)_{n\in\mathbb{N}}$ with $F''_n \in \mathcal{WC}_k$.

Our basic condition for a "good" representation F' of $F \in \mathcal{CLS}$ is that $F' \in \mathcal{UC}_k$ holds for some "low" k (a constant if F depends on parameters). This is what we call the **absolute condition** — regarding the requirement of detecting unsatisfiability of $\varphi * F'$ for some partial assignment φ we do not distinguish between original variables (those in $\mathrm{var}(F)$) and new variables (those in $\mathrm{var}(F') \setminus \mathrm{var}(F)$), that is, $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F')$ is considered. If we consider only $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$, then we obtain the **relative condition**:

Definition 6.1.4 *For $F \in \mathcal{CLS}$ and $V \subseteq \mathcal{VA}$ we define:*

- *the **relative hardness** $\mathbf{hd}^V(F) \in \mathbb{N}_0$ is the minimum $k \in \mathbb{N}_0$ such that for all partial assignments $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) \subseteq V$ and $\varphi * F \in \mathcal{USAT}$ we have $\mathrm{r}_k(\varphi * F) = \{\bot\}$;*
- *the **relative p-hardness** $\mathbf{phd}^V(F) \in \mathbb{N}_0$ is the minimum $k \in \mathbb{N}_0$ such that for all $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) \subseteq V$ we have $\mathrm{r}_k(\varphi * F) = \mathrm{r}_\infty(\varphi * F)$;*
- *the **relative w-hardness** $\mathbf{whd}^V(F) \in \mathbb{N}_0$ is the minimum $k \in \mathbb{N}_0$ such that for all $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) \subseteq V$ and $\varphi * F \in \mathcal{USAT}$ we have that k-resolution derives $\bot$ from $\varphi * F$.*

We have the following obvious properties:

1. $V \subseteq V'$ implies $h^{V'}(F) \leq h^V(F)$ for $h \in \{\mathrm{hd}, \mathrm{phd}, \mathrm{whd}\}$.
2. $h^V(F) \leq h(F) = h^{\mathrm{var}(F)}(F)$ for $h \in \{\mathrm{hd}, \mathrm{phd}, \mathrm{whd}\}$.
3. For $F \in \mathcal{SAT}$ holds $\mathrm{hd}^{\emptyset}(F) = \mathrm{whd}^{\emptyset}(F) = 0$, while $\mathrm{phd}^{\emptyset}(F)$ is the smallest k such that $\mathrm{r}_k(F) = \mathrm{r}_\infty(F)$.
4. For $F \in \mathcal{USAT}$ holds $\mathrm{hd}^{\emptyset}(F) = \mathrm{phd}^{\emptyset}(F) = \mathrm{hd}(F) = \mathrm{phd}(F)$ and $\mathrm{whd}^{\emptyset}(F) = \mathrm{whd}(F)$.

Having a representation F' of F with $\mathrm{phd}^{\mathrm{var}(F)}(F') \leq 1$ is typically called *maintaining arc consistency* (that is, forced assignments, which via the encoding represent restrictions on (higher-level) constraint domains, are always propagated by r_1 - see Section 2.8). Having $\mathrm{hd}^{\mathrm{var}(F)}(F') = 0$ is equivalent to $\mathrm{prc}_0(F) \subseteq F'$, and thus for hardness 0 new variables are not helpful, neither for the relative nor the absolute condition.

In Theorem 1 in [17], it is (essentially) proven that for every boolean function f with a poly-size representation F of relative hardness ≤ 1 can be mapped to a poly-size monotone circuit encoding the consistency checking function f_c for f and back to a relative hardness ≤ 1 representation F_C of f_c via the Tseitin translation. From this, along with monotone circuit lower bound results (see Chapter 9 in [97] for an overview) it immediately follows that there are boolean functions, such as the `AllDifferent` constraint and satisfiable pigeon-hole formulas (PHP^m_m), which have poly-size CNF representations but for which all relative hardness ≤ 1 representations are of super-polynomial size.

Theorem 6.1.5 (Corollary 4 in [17]) *There are sequences of boolean functions with poly-size CNF representations for which all representations with relative hardness ≤ 1 are of super-polynomial size.*

The mapping from CNF representations F of relative hardness ≤ 1 to monotone circuits for the consistency checker works by encoding the process of unit-clause propagation directly as a monotone circuit yielding a representation of the consistency checker. In the same way, it should also be possible to construct a monotone circuit for the consistency checker of F by encoding r_k-propagation rather than r_1. In this case, any sequence of boolean functions with poly-size representation of relative hardness $\leq k$ would have poly-size monotone circuit representations of their consistency checkers which would then map back, via the Tseitin translation, to a relative hardness ≤ 1 representation F_C for f_c. By adding binary "transfer" clauses to map the meaning of the variables of F to variables of the consistency checker, it should be possible to convert F_C to a clause-set F' with $\mathrm{hd}^{\mathrm{var}(F)}(F') \leq 1$. Such a translation from relative hardness $\leq k$ representations to relative hardness ≤ 1 representations would result in the collapse of the $\mathcal{UC}_k$ hierarchy to the first level for relative hardness.

Conjecture 6.1.6 *For every $k \geq 1$ there is a polytime function $t(F, V)$, which takes a clause-set F and a finite set V of variables as arguments, such that in case of $\mathrm{phd}^V(F) \leq k$ the output $t(F, V)$ is a representation of F with $\mathrm{phd}^V(t(F)) \leq 1$.*

More involved would be the collapse of the $\mathcal{WC}_k$-hierarchy to the first level regarding relative hardness. The poly-time SAT-decision mechanism for $\mathcal{WC}_k$ is more complex, however, the lack of constraint placed on the new variables by the relative condition means considerable simulation should be possible.

Conjecture 6.1.7 *For every $k \geq 1$ there is a polytime function $t(F, V)$, which takes a clause-set F and a finite set V of variables as arguments, such that in case of* $\mathrm{whd}^V(F) \leq k$ *the output $t(F, V)$ is a representation of F with* $\mathrm{whd}^V(t(F)) \leq 1$.

Note that for all $F \in \mathcal{CLS}$ and $V \subseteq \mathcal{VA}$ holds $\mathrm{whd}^V(F) \leq 1 \Leftrightarrow \mathrm{hd}^V(F) \leq 1$. The collapse of all considered hierarchies to their first level, when considering the relative condition, is for us a major argument in favour of the absolute condition: Within the class of representations of relative hardness at most 1 (when using new variables) there is a lot of structure, and many representations fulfil absolute conditions; some basic examples follow in the remainder of this chapter.

The focus in the remainder of this chapter turns to upper bounds, investigating cases where the Tseitin translation yields representations in $\mathcal{UC} \supseteq \mathcal{PC}$ for the absolute condition. Existing research (e.g., "maintaining arc-consistency via UCP") deals with relative p-hardness ≤ 1 and now we demonstrate that in the case of the Tseitin translation on hitting DNFs and reduced version of arbitrary DNFs, having absolute hardness ≤ 1 is possible. In Subsection 6.1 the general notion of "CNF representation" is discussed. In Subsection 6.2 the translation from DNF into CNF is discussed, which is considered here as a map from $\mathcal{CLS}$ to $\mathcal{CLS}$, and which we call the "canonical translation". Lemma 6.2.6 shows that the hardness of canonical translation results can be arbitrarily high. On the other hand, Lemma 6.2.7 shows that for hitting DNF the canonical translation result is in $\mathcal{UC}$, and Theorem 6.2.9 applies this to our lower bound examples, in contrast to Theorem 5.3.13 (so we see that new variables help here). By using only the necessary direction of the equivalences in the Tseitin translation, in Lemma 6.3.3 we see that for this "reduced canonical translation" the result is always in $\mathcal{UC}$. Finally in Section 6.4 the Tseitin translation is applied to sets of XOR equations, providing an example where the hardness of the Tseitin translation on general circuits is unbounded.

6.2 The canonical translation

If for the $F \in \mathcal{CLS}$ to be represented we have an equivalent DNF $G \in \mathcal{CLS}$, then we can apply the Tseitin translation, using one new variable v to express one DNF-clause, i.e., using $\mathrm{prc}_0(v \leftrightarrow \bigwedge_{x \in C} x)$ for $C \in G$. The details are as follows.

It is assumed that an injection $\mathrm{vct} : \{(F, C) \mid F \in \mathcal{CLS} \wedge C \in F\} \to \mathcal{VA}$ is given, yielding the variables of the canonical translation, such that $\mathrm{var}(F) \cap \{\mathrm{vct}(F, C)\}_{C \in F} = \emptyset$ holds for all $F \in \mathcal{CLS}$ (that is, these variables are new for F). We write $\mathbf{vct}_F^C := \mathrm{vct}(F, C)$.

Definition 6.2.1 *The map* $\mathbf{ct} : \mathcal{CLS} \to \mathcal{CLS}$ *is defined for* $F \in \mathcal{CLS}$ *as*

$$\mathrm{ct}(F) := \left\{ \{\overline{\mathrm{vct}_F^C}, x\} : C \in F \wedge x \in C \right\} \cup \left\{ \{\mathrm{vct}_F^C\} \cup \overline{C} : C \in F \right\} \cup \left\{ \{\mathrm{vct}_F^C\}_{C \in F} \right\}.$$

The first two types of clauses are the prime implicates of the boolean functions $\mathrm{vct}_F^C \leftrightarrow \bigwedge_{x \in C} x$, while the last type (a long, single clause) says that one of the (DNF-)clauses from F must be true. To emphasise: the map ct is a map from clause-sets to clause-sets, where the (implicit) interpretation of the input and the output is different: the input $F \in \mathcal{CLS}$ is interpreted as DNF, while the output $\mathrm{ct}(F) \in \mathcal{CLS}$ is interpreted as CNF. Some basic properties of the canonical translation:

1. The basic measures of the canonical translation for $F \in \mathcal{CLS}$ are given by
   (a) $n(\mathrm{ct}(F)) = n(F) + c(F)$

(b) $c(\mathrm{ct}(F)) = 1 + c(F) + \ell(F)$ for $F \neq \{\bot\}$.

2. $\mathrm{ct}(\top) = \{\bot\}$ and $\mathrm{ct}(\{\bot\}) = \{\{\mathrm{vct}^{\bot}_{\{\bot\}}\}\}$.

3. Consider $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$, and treat F as a multi-clause-set, that is, if application of φ to different non-satisfied clauses from F makes these clauses equal, then no contractions are performed. Then the canonical translation behaves homomorphic regarding application of partial assignments in the sense that $\mathrm{ct}(\overline{\varphi} * F)$ (recall that we need to treat F here as a DNF) is isomorphic to $(\varphi \cup \psi) * \mathrm{ct}(F)$, where ψ sets those vct^C_F to 0 for which there is $x \in C$ with $\varphi(x) = 0$.

Example 6.2.2 *We give some simple examples for canonical translations.*

1. *For* $F := \{\underbrace{\{v_1\}}_{C_1}, \bot\}$ *we have*

$$\mathrm{ct}(F) = \{\underbrace{\{\overline{\mathrm{vct}^{C_1}_F}, v_1\}, \{\mathrm{vct}^{C_1}_F, \overline{v_1}\}}_{\boldsymbol{v_1 \leftrightarrow \mathrm{vct}^{C_1}_F}}, \underbrace{\{\mathrm{vct}^{\bot}_F\}}_{\boldsymbol{1 \leftrightarrow \mathrm{vct}^{\bot}_F}}, \underbrace{\{\mathrm{vct}^{C_1}_F, \mathrm{vct}^{\bot}_F\}}_{\boldsymbol{\mathrm{vct}^{C_1}_F \vee \mathrm{vct}^{\bot}_F}} \}.$$

2. *For* $F := \{\underbrace{\{v_1, v_2, v_3\}}_{C_1}, \underbrace{\{v_1, v_2, v_4\}}_{C_2}\}$ *we have*

$$\mathrm{ct}(F) = \{\underbrace{\{\overline{\mathrm{vct}^{C_1}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_1}_F}, v_2\}, \{\overline{\mathrm{vct}^{C_1}_F}, v_3\}, \{\mathrm{vct}^{C_1}_F, \overline{v_1}, \overline{v_2}, \overline{v_3}\}}_{\boldsymbol{(v_1 \wedge v_2 \wedge v_3) \leftrightarrow \mathrm{vct}^{C_1}_F}},$$
$$\underbrace{\{\overline{\mathrm{vct}^{C_2}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_2}_F}, v_2\}, \{\overline{\mathrm{vct}^{C_2}_F}, v_4\}, \{\mathrm{vct}^{C_2}_F, \overline{v_1}, \overline{v_2}, \overline{v_4}\}}_{\boldsymbol{(v_1 \wedge v_2 \wedge v_4) \leftrightarrow \mathrm{vct}^{C_2}_F}}, \underbrace{\{\mathrm{vct}^{C_1}_F, \mathrm{vct}^{C_2}_F\}}_{\boldsymbol{\mathrm{vct}^{C_1}_F \vee \mathrm{vct}^{C_2}_F}} \}.$$

3. *Applying* $\varphi := \langle v_3 \to 1, v_4 \to 1\rangle$ *to the last example (Case 2) yields*

$$\varphi * \mathrm{ct}(F) = \{\underbrace{\{\overline{\mathrm{vct}^{C_1}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_1}_F}, v_2\}, \{\mathrm{vct}^{C_1}_F, \overline{v_1}, \overline{v_2}\}}_{\boldsymbol{(v_1 \wedge v_2) \leftrightarrow \mathrm{vct}^{C_1}_F}},$$
$$\underbrace{\{\overline{\mathrm{vct}^{C_2}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_2}_F}, v_2\}, \{\mathrm{vct}^{C_2}_F, \overline{v_1}, \overline{v_2}\}}_{\boldsymbol{(v_1 \wedge v_2) \leftrightarrow \mathrm{vct}^{C_2}_F}}, \underbrace{\{\mathrm{vct}^{C_1}_F, \mathrm{vct}^{C_2}_F\}}_{\boldsymbol{\mathrm{vct}^{C_1}_F \vee \mathrm{vct}^{C_2}_F}} \}.$$

4. *Applying* $\varphi := \langle v_3 \to 0\rangle$ *to Case 2 yields*

$$\varphi * \mathrm{ct}(F) = \{\underbrace{\{\overline{\mathrm{vct}^{C_1}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_1}_F}, v_2\}, \{\overline{\mathrm{vct}^{C_1}_F}\}}_{\boldsymbol{\overline{\mathrm{vct}^{C_1}_F}}},$$
$$\underbrace{\{\overline{\mathrm{vct}^{C_2}_F}, v_1\}, \{\overline{\mathrm{vct}^{C_2}_F}, v_2\}, \{\overline{\mathrm{vct}^{C_2}_F}, v_4\}, \{\mathrm{vct}^{C_2}_F, \overline{v_1}, \overline{v_2}, \overline{v_4}\}}_{\boldsymbol{(v_1 \wedge v_2 \wedge v_4) \leftrightarrow \mathrm{vct}^{C_2}_F}}, \underbrace{\{\mathrm{vct}^{C_1}_F, \mathrm{vct}^{C_2}_F\}}_{\boldsymbol{\mathrm{vct}^{C_1}_F \vee \mathrm{vct}^{C_2}_F}} \}.$$

5. *While applying* $\varphi := \langle v_3 \to 0 \rangle$ *and* $\psi := \{\mathrm{vct}_F^{C_1} \to 0\}$ *to Case 2 yields*

$$(\varphi \cup \psi) * \mathrm{ct}(F) = \{\underbrace{\{\overline{\mathrm{vct}_F^{C_2}}, v_1\}, \{\overline{\mathrm{vct}_F^{C_2}}, v_2\}, \{\overline{\mathrm{vct}_F^{C_2}}, v_4\}, \{\mathrm{vct}_F^{C_2}, \overline{v_1}, \overline{v_2}, \overline{v_4}\}}_{(v_1 \wedge v_2 \wedge v_4) \leftrightarrow \mathbf{vct}_F^{C_2}}, \underbrace{\{\mathrm{vct}_F^{C_2}\}}_{\mathbf{vct}_F^{C_2}}\}.$$

In Case 3 we see an example of why for the canonical translation to have the homomorphism property we must consider F *as a multi-clause-set. That is,* $\overline{\varphi} * F = \{\{v_1, v_2\}\}$*, and so* $\varphi * \mathrm{ct}(F) \neq \mathrm{ct}(\overline{\varphi} * F)$*: the clause* $\{v_1, v_2\}$ *is represented by two separate new variables in* $\varphi * \mathrm{ct}(F)$ *compared to only one in* $\mathrm{ct}(\overline{\varphi} * F)$*.*

In Case 4 we see an example where for the homomorphism property of the canonical translation not just renaming, but also some unit-clause elimination is needed. These unit-clauses are added in Case 5, extending the assignment to falsify the new variable $\mathrm{vct}_F^{C_1}$ *corresponding to falsified DNF-clause* C_1*.*

Lemma 6.2.3 *Consider* $F \in \mathcal{CLS}$ *(as CNF) and an equivalent DNF-clause-set* $G \in \mathcal{CLS}$*. Then* $\mathrm{ct}(G)$ *is a CNF-representation of* F*.*

Proof: $\mathrm{ct}(F)$ is true iff at least one of its vct-variables is set to true, which is precisely the case iff at least one of DNF-clauses of G is satisfied, where the (DNF-)clauses of G cover precisely the satisfying assignments of F. □

Lemma 6.2.4 *For* $F \in \mathcal{CLS}$ *we have* $\mathrm{hd}^{\mathrm{var}(F)}(\mathrm{ct}(F)) \leq 1$ *(recall Definition 6.1.4).*

Proof: Consider $\varphi \in \mathcal{PASS}$ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$ and $\varphi * \mathrm{ct}(F) \in \mathcal{USAT}$. Then all DNF-clauses of F are falsified, which yields via UCP that all vct-variables are set to false, and thus $\mathrm{r}_1(\varphi * \mathrm{ct}(F)) = \{\bot\}$. □

In [98] a more general version of Lemma 6.2.4 is proven, showing that for all "smooth" DNNFs (Disjoint Negation Normal Form) the Tseitin translation yields a clause-set which maintains arc-consistency via UCP (a somewhat stronger property than relative hardness ≤ 1 as in Lemma 6.2.4).[1)] That Lemma 6.2.4 only establishes the relative condition, and not the absolute one, is due to the fact that setting vct-variables to 0 can pose arbitrarily hard conditions; a concrete example follows, while a more drastic general construction is given in Lemma 6.2.6. However the difficulties can be overcome, by just removing them: In Lemma 6.3.3 we will see that when dropping the part of the canonical translation which gives meaning to setting vct-variables to 0, that then we actually can establish the absolute condition.

Example 6.2.5 *Consider the following clause-set with variables* $x_1, \ldots, x_5$*:*

$$F := \{\underbrace{\{x_1, x_2, x_3\}}_{C_1}, \underbrace{\{x_1, x_2, x_4\}}_{C_2}, \underbrace{\{x_1, x_2, x_5\}}_{C_3}\}.$$

[1)] There is a mistake in [98] in that it claims that the Tseitin translation of *all* DNNFs maintain arc-consistency via UCP, however this is shown only for smooth DNNFs as confirmed by George Katirelos via e-mail in January 2012.

The canonical translation is

$$\begin{aligned}\mathrm{ct}(F) \;=\; & \underbrace{\{\{x_1,\overline{\mathrm{vct}_F^{C_1}}\},\{x_2,\overline{\mathrm{vct}_F^{C_1}}\},\{x_3,\overline{\mathrm{vct}_F^{C_1}}\}\},\{\overline{x_1},\overline{x_2},\overline{x_3},\mathrm{vct}_F^{C_1}\}}_{\mathbf{vct}_F^{C_1} \leftrightarrow (x_1 \wedge x_2 \wedge x_3)} \cup \\ & \underbrace{\{\{x_1,\overline{\mathrm{vct}_F^{C_2}}\},\{x_2,\overline{\mathrm{vct}_F^{C_2}}\},\{x_4,\overline{\mathrm{vct}_F^{C_2}}\}\},\{\overline{x_1},\overline{x_2},\overline{x_4},\mathrm{vct}_F^{C_2}\}}_{\mathbf{vct}_F^{C_2} \leftrightarrow (x_1 \wedge x_2 \wedge x_4)} \cup \\ & \underbrace{\{\{x_1,\overline{\mathrm{vct}_F^{C_3}}\},\{x_2,\overline{\mathrm{vct}_F^{C_3}}\},\{x_5,\overline{\mathrm{vct}_F^{C_3}}\}\},\{\overline{x_1},\overline{x_2},\overline{x_5},\mathrm{vct}_F^{C_3}\}}_{\mathbf{vct}_F^{C_3} \leftrightarrow (x_1 \wedge x_2 \wedge x_5)} \cup \\ & \underbrace{\{\{\mathrm{vct}_F^{C_1},\mathrm{vct}_F^{C_2},\mathrm{vct}_F^{C_3}\}\}}_{(\mathbf{vct}_F^{C_1} \vee \mathbf{vct}_F^{C_2} \vee \mathbf{vct}_F^{C_3})} \;.\end{aligned}$$

Applying the partial assignment $\varphi := \langle x_3 \to 1, x_4 \to 1, x_5 \to 1, \mathrm{vct}_F^{C_3} \to 0\rangle$ *yields*

$$\begin{aligned}F' := \varphi * \mathrm{ct}(F) \;=\; & \underbrace{\{\{x_1,\overline{\mathrm{vct}_F^{C_1}}\},\{x_2,\overline{\mathrm{vct}_F^{C_1}}\}\},\{\overline{x_1},\overline{x_2},\mathrm{vct}_F^{C_1}\}}_{\mathbf{vct}_F^{C_1} \leftrightarrow (x_1 \wedge x_2)} \cup \\ & \underbrace{\{\{x_1,\overline{\mathrm{vct}_F^{C_2}}\},\{x_2,\overline{\mathrm{vct}_F^{C_2}}\}\},\{\overline{x_1},\overline{x_2},\mathrm{vct}_F^{C_2}\}}_{\mathbf{vct}_F^{C_2} \leftrightarrow (x_1 \wedge x_2)} \cup \\ & \underbrace{\{\{\overline{x_1},\overline{x_2}\}\}}_{\neg(x_1 \wedge x_2)} \cup \underbrace{\{\{\mathrm{vct}_F^{C_1},\mathrm{vct}_F^{C_2}\}\}}_{(\mathbf{vct}_F^{C_1} \vee \mathbf{vct}_F^{C_2})} \;.\end{aligned}$$

We have $F' \in \mathcal{USAT}$*, where* F' *has no unit-clauses, whence* $\mathrm{hd}(F') \geq 2$*, and so* $\mathrm{ct}(F) \notin \mathcal{UC}_1$.

For general input-DNFs, the hardness of the canonical translation can be arbitrary high:

Lemma 6.2.6 *Consider* $F \in \mathcal{CLS}$*. Let* $v \in \mathcal{VA} \setminus \mathrm{var}(F)$ *and* $F' := F \cup \{\{v\}\}$*. Then* $\mathrm{hd}(\mathrm{ct}(F')) \geq \mathrm{hd}(F)$.

Proof: Let $\varphi := \langle \mathrm{vct}_{F'}^C \to 0 : C \in F\rangle \cup \langle v \to 1, \mathrm{vct}_{F'}^{\{v\}} \to 1\rangle$. Then $\varphi * \mathrm{ct}(F') = F'' := \{\overline{C} : C \in F\}$, where $\mathrm{hd}(\mathrm{ct}(F')) \geq \mathrm{hd}(F'') = \mathrm{hd}(F)$. □

If we do not have just a DNF, but a "disjoint" or "orthogonal" DNF (see Section 1.6 and Chapter 7 in [42]), which are as clause-sets precisely the hitting clause-sets, then we obtain absolute hardness 1:

Lemma 6.2.7 *For $F \in \mathcal{HIT}$ we have $\mathrm{ct}(F) \in \mathcal{UC}$, where $\mathrm{ct}(F)$ is a representation of the DNF-clause-set F.*

Proof: Consider a partial assignment φ such that $\varphi * \mathrm{ct}(F)$ is unsatisfiable. Since $\mathcal{HIT}$ is stable under application of partial assignments, and furthermore here no contractions take place, w.l.o.g. we can assume that $\mathrm{var}(\varphi) \cap \mathrm{var}(F) = \emptyset$. If φ sets two or more vct-variables to true, then UCP yields a contradiction, since any two clauses from F clash. If φ would set precisely one vct-variable to true, then we had $\varphi * \mathrm{ct}(F) = \top$. So assume that φ sets no vct-variable to true. Now φ must set all vct-variables to false, since, as already mentioned, just setting one vct-variable to true satisfies $\mathrm{ct}(F)$. And thus $\bot \in \mathrm{ct}(F)$. □

We now want to show that via the canonical translation we can obtain representations of $\mathrm{D}(F)$ for $F \in \mathcal{UHIT}$. For this we show first that all such $\mathrm{D}(F)$ have short hitting DNF clause-sets. For $F \in \mathcal{CLS}$ let $\#\mathbf{sat}(\boldsymbol{F}) \in \mathbb{N}_0$ denote the number of satisfying assignments for F, that is, $\#\mathrm{sat}(F) = |\mathrm{DNF}(F)|$.

Lemma 6.2.8 *Consider $F \in \mathcal{UHIT}$, and let $m := n(F) + c(F)$.*

1. $\#\mathrm{sat}(\mathrm{D}(F)) = 2^{m-1}$.
2. *Let $F' := \{ \overline{C} \cup \{u_C\} \mid C \in F \}$; by definition we have $F' \in \mathcal{HIT}$. Furthermore $\#\mathrm{sat}(F') = 2^{m-1}$.*
3. *F' as a DNF-clause-set is equivalent to the CNF-clause-set $\mathrm{D}(F)$.*

Proof: We have $\sum_{C \in F} 2^{-|C|} = 1$ (see [100]). Thus $\sum_{C \in \mathrm{D}(F)} 2^{-|C|} = \frac{1}{2}$, which proves Part 1 (note $m = n(\mathrm{D}(F))$ and $\mathrm{D}(F) \in \mathcal{HIT}$). Part 2 follows from Part 1, since F' results from $\mathrm{D}(F)$ by flipping literals. Finally we consider Part 3. All elements of F', as DNF-clauses (i.e., conjunctions of literals), represent satisfying assignments for $\mathrm{D}(F)$, that is, for all $C \in F'$ and $D \in \mathrm{D}(F)$ we have $C \cap D \neq \emptyset$. By Part 2, precisely half of the total assignments of DNF-clause-set F' are falsifying, and thus precisely half of the total assignments are satisfying: since this is the same number as the satisfying assignments of $\mathrm{D}(F)$, we obtain that the DNF-clause-set F' is equivalent to the CNF-clause-set F. □

An alternative line of argumentation is that for $F \in \mathcal{UHIT}$ the (logical) negation of $\mathrm{D}(F)$ (as a CNF) is given by $\mathrm{D}(F)'$, which is obtained from $\mathrm{D}(F)$ by flipping all doping literals, i.e., replacing clauses $C \cup \{u_C\}$ by $C \cup \{\overline{u_C}\}$. That this is indeed the negation, follows from the two facts, that $\mathrm{D}(F) \cup \mathrm{D}(F)' \in \mathcal{HIT}$ by definition, and that $\mathrm{D}(F) \cup \mathrm{D}(F)'$ results from F by replacing each clause C with the two clauses $C \cup \{u_C\}, C \cup \{\overline{u_C}\}$, which are together equivalent to C.

By Lemma 6.2.8 and Lemma 6.2.7 we obtain now that doped unsatisfiable hitting clause-sets have good representations via the canonical translation:

Theorem 6.2.9 *For $F \in \mathcal{UHIT}$ there is a short CNF-representation (using new variables) of* $\mathrm{D}(F)$ *in $\mathcal{UC}$, namely $F' := \mathrm{ct}(\{\overline{C} \cup \{u_C\} : C \in F\}) \in \mathcal{UC}$, where:*

1. $n(F') = n(F) + 2c(F)$.
2. $c(F') = 1 + 2c(F) + \ell(F)$.

This applies especially for $F \in \mathcal{SMU}_{\delta=1} \subset \mathcal{UHIT}$.

6.3 The reduced canonical translation

Finally we show that when relaxing the canonical translation, using only the necessary direction of the constitutive equivalences, then we actually obtain representations in $\mathcal{UC}$ for every DNF-clause-set:

Definition 6.3.1 *The map $\mathbf{ct}^- : \mathcal{CLS} \to \mathcal{CLS}$ ("reduced canonical translation") is defined for $F \in \mathcal{CLS}$ as* $\mathrm{ct}^-(F) := \{\{\overline{\mathrm{vct}_F^C}, x\} : C \in F \wedge x \in C\} \cup \{\{\mathrm{vct}_F^C\}_{C \in F}\}$.

Note that all clauses of $\mathrm{ct}^-(F)$ are binary except of the long clause stating that one of the vct-variables must become true. And also note that in case of $\bot \notin F$ the additional clauses of $\mathrm{ct}(F)$, that is, the $C \in \mathrm{ct}(F) \setminus \mathrm{ct}^-(F)$, are all blocked for $\mathrm{ct}(F)$ (see [105, 106]), since C can not be resolved on the vct-variable in it. We have $\mathrm{var}(\mathrm{ct}^-(F)) = \mathrm{var}(\mathrm{ct}(F))$, and thus $n(\mathrm{ct}^-(F)) = n(F) + c(F)$, while $c(\mathrm{ct}^-(F)) = 1 + \ell(F)$, for all $F \in \mathcal{CLS}$. With the same proof as Lemma 6.2.3 we get:

Lemma 6.3.2 *Consider $F \in \mathcal{CLS}$ (as CNF) and an equivalent DNF-clause-set $G \in \mathcal{CLS}$. Then* $\mathrm{ct}^-(G)$ *is a CNF-representation of F.*

We show now that dropping the additional (blocked) clauses, present in the full form $\mathrm{ct}(F)$, actually leads to the hardness dropping to 1 for arbitrary input-DNFs (recall Lemma 6.2.6), exploiting that now there are less possibilities for making $\mathrm{ct}^-(F)$ unsatisfiable by instantiation:

Lemma 6.3.3 *For $F \in \mathcal{CLS}$ we have* $\mathrm{ct}^-(F) \in \mathcal{UC}$ *(i.e., $\mathrm{ct}^- : \mathcal{CLS} \to \mathcal{UC}$).*

Proof: For the sake of contradiction consider a partial assignment φ such that $F' := \mathrm{r}_1(\varphi * \mathrm{ct}^-(F)) \in \mathcal{USAT}$ but $F' \neq \{\bot\}$. Note that F' contains neither $\bot$ nor a unit-clause, and thus F' is a subset of $\mathrm{ct}^-(F)$ except of the possibly shortened or satisfied long vct-clause. If F' contains no new variables, then thus $F' = \top$, a contradiction. So there exists some $C \in F$ such that vct_F^C occurs in F'. Consider the assignment φ', which sets vct_F^C and all $x \in C$ to true, while setting all other (remaining) new variables to false: φ' satisfies F', a contradiction. □

Example 6.3.4 *We conclude our basic considerations of "canonical translations" by discussing "unique extension properties". A representation F' of F has the* unique extension property *("uep") if for every total satisfying assignment of F there is exactly one extension to a satisfying assignment of F'. For every $F \in \mathcal{CLS}$ the representation* $\mathrm{ct}(F)$ *of F has the uep, since a variable* vct_F^C *must be set to* 1 *precisely for those $C \in F$ which are satisfied by φ in the DNF-sense (i.e., $\overline{\varphi} * \{C\} = \{\bot\}$). On the other hand, the representation* $\mathrm{ct}^-(F)$ *of F in general has not the uep:*

The total satisfying assignments for $\mathrm{ct}(F)$ *extending* φ *are exactly those which set* at least *one of the variables* vct_F^C *true for those* $C \in F$ *which are satisfied in the DNF-sense.*

A representation F' *of* F *has the* strong unique extension property *if for every partial assignment* φ *with* $\bot \in \overline{\varphi} * F$ *(i.e.,* φ *satisfied at least one of the DNF-clauses) there is exactly one extension on the new variables (alone) to a satisfying assignment of* F'*. For* $F \in \mathcal{HIT}$ *the representation* $\mathrm{ct}(F)$ *of* F *has the strong uep, since the satisfying assignments given by the clauses of* F *are inconsistent with each other.*

6.4 XOR-clause-sets

Systems of linear equations over the two-element field $\{0,1\}$ and their SAT representations are the subject of this section. The framework is discussed in Subsection 6.4.1, some general tools for computing hardness from components are presented in Subsection 6.4.2, and finally we consider more general systems of XORs in Subsection 6.4.3.

6.4.1 The framework

As usual, an **XOR-clause** (also known as "parity constraint") is a (boolean) constraint of the form $x_1 \oplus \cdots \oplus x_n = 0$ for literals $x_1, \ldots, x_n$, which we just represent by the clause $\{x_1, \ldots, x_n\} \in \mathcal{CL}$, where w.l.o.g. we can restrict our attention to (proper) clauses (as clash-free sets). Note that $x_1 \oplus \cdots \oplus x_n = y$ is equivalent to $x_1 \oplus \cdots \oplus x_n \oplus y = 0$, while $x \oplus x = 0$ and $x \oplus \overline{x} = 1$, and $0 \oplus x = x$ and $1 \oplus x = \overline{x}$. An **XOR-clause-set** F is a set of XOR-clauses, which is just represented by an ordinary clause-set $F \in \mathcal{CLS}$ (with a different interpretation). A partial assignment $\varphi \in \mathcal{PASS}$ satisfies an XOR-clause-set F iff $\mathrm{var}(\varphi) \supseteq \mathrm{var}(F)$ and for every $C \in F$ the number of $x \in C$ with $\mathrm{var}(x) = 1$ is even. A CNF-representation of an XOR-clause-set $F \in \mathcal{CLS}$ is a clause-set $F' \in \mathcal{CLS}$ with $\mathrm{var}(F) \subseteq \mathrm{var}(F')$ such that the projections of the satisfying total assignments for F' (as CNF-clause-set) to $\mathrm{var}(F)$ are precisely the satisfying (total) assignments for F (as XOR-clause-set).

In this initial study we concentrate on CNF-representations of XOR-clause-sets F with $c(F) \leq 2$. First we consider $c(F) = 1$, that is, a single XOR-clause C, to which we often refer as "$x_1 \oplus \cdots \oplus x_n = 0$" (with $n = |C|$; more precise would be $\oplus_{x \in C} x = 0$). There is precisely one equivalent clause-set, i.e., there is exactly one representation without new variables, namely $X_0(C) := \mathrm{prc}_0(x_1 \oplus \cdots \oplus x_n = 0)$, the set of prime implicates of the underlying boolean function, which is unique since the prime implicates are not resolvable. $X_0(C)$ has 2^{n-1} clauses for $n \geq 1$ (while for $n = 0$ we have $X_0(C) = \top$), namely precisely those full clauses over $\{\mathrm{var}(x_1), \ldots, \mathrm{var}(x_n)\}$ where the parity of the number of complementations is different from the parity of the number of complementations in C. So representations of hardness 0 are only feasible for small n. In the following lemma we analyse a typical CNF-representation of the XOR-clause C, which uses new variables y_i (for $i \in \{2, \ldots, n-1\}$) to compute the xor of the first i bits, i.e., $y_i = x_1 \oplus \cdots \oplus x_i$.

Lemma 6.4.1 *We define $X_1 : \mathcal{CL} \to \mathcal{UC}$, such that $X_1(C)$ is a CNF-representation of the XOR-clause-set $\{C\}$, as follows, using $C = \{x_1, \ldots, x_n\}$, $n = |C|$.*

- *For $n \leq 2$ let $X_1(C)$ be the CNF-representation without new variables.*
- *For $n \geq 3$ consider pairwise different (new) variables $y_2, \ldots, y_{n-1} \in \mathcal{VA} \setminus \mathrm{var}(C)$. Let $X_1(C) := P_2 \cup \left(\bigcup_{i=3}^{n-1} P_i\right) \cup P_n$, where*

$$\begin{array}{rcl} P_2 & := & \mathrm{prc}_0(x_1 \oplus x_2 = y_2) = \{\{\overline{x_1}, x_2, y_2\}, \{x_1, \overline{x_2}, y_2\}, \{x_1, x_2, \overline{y_2}\}, \{\overline{x_1}, \overline{x_2}, \overline{y_2}\}\} \\ P_i & := & \mathrm{prc}_0(y_{i-1} \oplus x_i = y_i) = \{\{\overline{y_{i-1}}, x_i, y_i\}, \{y_{i-1}, \overline{x_i}, y_i\}, \{y_{i-1}, x_i, \overline{y_i}\}, \{\overline{y_{i-1}}, \overline{x_i}, \overline{y_i}\}\} \\ P_n & := & \mathrm{prc}_0(y_{n-1} \oplus x_n = 0) = \{\{y_{n-1}, \overline{x_n}\}, \{\overline{y_{n-1}}, x_n\}\}. \end{array}$$

We have $X_1(C) \in \mathcal{UC}$, with $n(F) = 2n - 2$, $c(F) = 4n - 6$ and $\ell(F) = 12n - 20$.

Proof: Let $F := X_1(C)$. Assume for the sake of contradiction that $F \notin \mathcal{UC}$. Thus there exists a partial assignment φ such that for $F' := \mathrm{r}_1(\varphi * F)$ we have $F' \in \mathcal{USAT}$, but $F' \neq \{\bot\}$. By definition F' has no clauses of size ≤ 1 and is non-empty. Observe that setting any variable in P_i for $i \in \{2, \ldots, n-1\}$ yields a pair of binary clauses representing an equivalence or anti-equivalence between the two remaining variables. Also if $P_i \cap F' \neq \emptyset$ for some $i \in \{2, \ldots, n-1\}$, then we have $P_i \subseteq F'$, since all clauses of P_i contain all variables of P_i. Therefore we have $F' = E \cup \bigcup_{i \in I} P_i$ for some subset $I \subseteq \{2, \ldots, n-1\}$, where E is a set of clauses representing a chain of equalities and inequalities. Consider the assignment $\varphi' := \langle x_i \to 0 : i \in I \rangle$. We have $\varphi' * P_i = \varphi' * \mathrm{prc}_0(y_{i-1} + x_i = y_i) = \mathrm{prc}_0(y_{i-1} = y_i)$; note that x_i is *only* in P_i, and so $\langle x_i \to 1 \rangle$ only touches P_i. So $\varphi' * F'$ now contains only variable-disjoint chains of equivalences and anti-equivalences, each trivially satisfiable, yielding a contradiction. □

Example 6.4.2 *For $n = 3$ we get*

$$X_1(C) = \{ \underbrace{\{x_1, x_2, \overline{y_2}\}, \{x_1, \overline{x_2}, y_2\}, \{\overline{x_1}, x_2, y_2\}, \{\overline{x_1}, \overline{x_2}, \overline{y_2}\}}_{x_1 \oplus x_2 = y_2}, \underbrace{\{y_2, \overline{x_3}\}, \{\overline{y_2}, x_3\}}_{y_2 \oplus x_3 = 0} \}.$$

6.4.2 Hardness under union

When applied piecewise to a system of linear equations (with different auxiliary variables for each single equation), the translation X_1 from Lemma 6.4.1 does not yield a clause-set in $\mathcal{UC}$, as shown in Theorem 6.4.7. To facilitate the precise computation of the hardness of the union of two such XOR-clause-translations, two general tools for upper bounds on hardness and one for lower bounds are presented.

Lemma 6.4.3 *Consider $F \in \mathcal{CLS}$ and $V \subseteq \mathrm{var}(F)$. Let P be the set of partial assignments ψ with $\mathrm{var}(\psi) = V$. Then $\mathrm{hd}(F) \leq |V| + \max_{\psi \in P} \mathrm{hd}(\psi * F)$.*

Proof: Consider a partial assignment φ with $\varphi * F \in \mathcal{USAT}$; we have to show $\mathrm{hd}(\varphi * F) \leq |V| + \max_{\psi \in P} \mathrm{hd}(\psi * F)$. Build a resolution refutation of $\varphi * F$ by first creating a splitting tree (possibly degenerated) on the variables of V; this splitting tree (a perfect binary tree) has height $|V|$, and at each of its leaves we have a clause-set $\varphi * (\psi * F)$ for some appropriate $\psi \in P$. Thus at each leaf we can attach a splitting tree of Horton-Strahler number of hardness at most $\max_{\psi \in P} \mathrm{hd}(\psi * F)$, and from that (via the well-known correspondence of splitting trees

and resolution trees; see [104, 110] for details) we obtain a resolution tree fulfilling the desired hardness bound. □

We obtain an upper bound on the hardness of the union of two clause-sets:

Corollary 6.4.4 *For* $F_1, F_2 \in \mathcal{CLS}$ *holds* $\mathrm{hd}(F_1 \cup F_2) \le \max(\mathrm{hd}(F_1), \mathrm{hd}(F_2)) + |\mathrm{var}(F_1) \cap \mathrm{var}(F_2)|$.

Proof: Apply Lemma 6.4.3 with $F := F_1 \cup F_2$ and $V := \mathrm{var}(F_1) \cap \mathrm{var}(F_2)$, and apply the general upper bound $\mathrm{hd}(F_1 \cup F_2) \le \max(\mathrm{hd}(F_1), \mathrm{hd}(F_2))$ for variable-disjoint F_1, F_2 (Lemma 15 in [76]). □

Substitution of literals can not increase (w-)hardness:

Lemma 6.4.5 *Consider a clause-set* $F \in \mathcal{USAT}$ *and (arbitrary) literals* x, y. *Denote by* $F_{x \leftarrow y} \in \mathcal{USAT}$ *the result of replacing* x *by* y *and* $\overline{x}$ *by* $\overline{y}$ *in* F, *followed by removing clauses containing complementary literals. Then we have* $\mathrm{hd}(F_{x \leftarrow y}) \le \mathrm{hd}(F)$ *and* $\mathrm{whd}(F_{x \leftarrow y}) \le \mathrm{whd}(F)$.

Proof: Consider $T : F \vdash \bot$. It is a well-known fact (and a simple exercise), that the substitution of y into x can be performed in T, obtaining $T_{x \leftarrow y} : F_{x \leftarrow y} \vdash \bot$. This is easiest to see by performing first the substitution with T itself, obtaining a tree T' which as a binary tree is identical to T, using "pseudo-clauses" with (possibly) complementary literals; the resolution rule for sets C, D of literals with $x \in C$ and $\overline{x} \in D$ allows to derive the clause $(C \setminus \{x\}) \cup (D \setminus \{\overline{x}\})$, where the resolution-variables are taken over from T. Now "tautological" clauses (containing complementary literals) can be cut off from T': from the root (labelled with $\bot$) go to a first resolution step where the resolvent is non-tautological, while one of the parent clauses is tautological (note that not both parent clauses can be tautological) — the subtree with the tautological clause can now be cut off, obtaining a new pseudo-resolution tree where clauses only got (possibly) shorter (see Lemma 6.1, part 1, in [110]). Repeating this process we obtain $T_{x \leftarrow y}$ as required. Obviously $\mathrm{hs}(T_{x \leftarrow y}) \le \mathrm{hs}(T)$, and if in T for every resolution step at least one of the parent clauses has length at most k for some fixed (otherwise arbitrary) $k \in \mathbb{N}_0$, then this also holds for $T_{x \leftarrow y}$. □

Example 6.4.6 *The simplest example showing that for satisfiable clause-sets* F *(w-)hardness can be increased by substitution is given by* $F := \{\{x\}, \{\overline{y}\}\}$ *for* $\mathrm{var}(x) \ne \mathrm{var}(y)$. *Here* $\mathrm{hd}(F) = 0$, *while* $F_{x \leftarrow y} = \{\{y\}, \{\overline{y}\}\}$, *and thus* $\mathrm{hd}(F_{x \leftarrow y}) = 1$.

6.4.3 Translating two XOR-clauses

Now we are ready to determine the (high) hardness of the union of the (piecewise) translation of two XOR-clauses for a basic special case:

Theorem 6.4.7 *For* $n \in \mathbb{N}$ *and (different) variables* $v_1, \ldots, v_n$ *consider the system*

$$\begin{aligned} v_1 \oplus v_2 \oplus \cdots \oplus v_n &= 0 \\ v_1 \oplus v_2 \oplus \cdots \oplus \overline{v_n} &= 0, \end{aligned}$$

that is, consider the XOR-clauses $C_1 := \{v_1, \ldots, v_n\}$ *and* $C_2 := \{v_1, \ldots, v_{n-1}, \overline{v_n}\}$. *First we remark that* $X_0(\{C_1, C_2\})$ *is the clause-set with all* 2^n *full clauses of* $\{v_1, \ldots, v_n\}$, *and thus* $\mathrm{hd}(X_0(\{C_1, C_2\}) = \mathrm{whd}(X_0(\{C_1, C_2\}) = n$. *Now let* $T_n := X_1(\{C_1, C_2\})$ *(recall Lemma 6.4.1). We have:*

1. $n(T_n) = 2 \cdot (2n-2) - n = 3n - 4$ *for* $n \geq 2$.
2. $c(T_n) = 8n - 12$ *for* $n \geq 2$.
3. $\ell(T_n) = 24n - 40$ *for* $n \geq 2$.
4. $T_n \in \mathcal{USAT} \cap 3\text{-}\mathcal{CLS}$.
5. $\mathrm{hd}(F) = n$.
6. *For* $n \geq 3$ *holds* $\mathrm{whd}(T_n) = \mathrm{wid}(T_n) = 3$.

Proof: The (w-)hardness of $X_0(\{C_1, C_2\})$ follows by Lemma 3.18 in [104]. From Corollary 6.4.4 and Lemma 6.4.1 we obtain $\mathrm{hd}(T_n) \leq n+1$. Better is to apply Lemma 6.4.3 with $V := \{v_2, \ldots, v_{n-1}\}$. By definition we see that $\psi * T_n \in 2\text{-}\mathcal{CLS}$ (i.e., all clauses have length at most two) for ψ with $\mathrm{var}(\psi) = V$. By Lemma 19 in [76] we have $\mathrm{hd}(\psi * T_n) \leq 2$, and thus $\mathrm{hd}(T_n) \leq (n-2) + 2 = n$. The lower bound is obtained by an application of Lemma 3.3.7. Consider any literal $x \in \mathrm{lit}(T_n)$. Setting x to true or false results either in an equivalence or in an anti-equivalence. Propagating this (anti-)equivalence yields a clause-set T' isomorphic to T_{n-1}, where by Lemma 6.4.5 this propagation does not increase hardness, so we have $\mathrm{hd}(\langle x \to 1\rangle * T_n) \geq \mathrm{hd}(T') = \mathrm{hd}(T_{n-1})$. The argumentation can be trivially extended for $n \in \{0,1,2\}$, and so by Lemma 3.3.7 we get $\mathrm{hd}(T_n) \geq n$.

We now turn to the w-hardness. To show the upper bound, consider the closure $\widehat{T_n}$ of T_n under 2-resolution. The binary clauses in T_n are exactly $\mathrm{prc}_0(y_{n-1} = x_n \wedge \overline{y'_{n-1}} = x_n)$. The resolution of these binary clauses with ternary clauses in T_n allows the corresponding substitutions $(y_{n-1} = x_n = \overline{y'_{n-1}})$ to be made in (other) clauses containing those variables, but this does not introduce any further clauses of size ≤ 2. Therefore, $\widehat{T_n}$ contains only clauses of size ≥ 2, so $\mathrm{whd}(T_n) \geq 3$.

To show $\mathrm{wid}(T_n) \leq 3$, we construct a resolution refutation as follows:

1. From $\mathrm{prc}_0(y_{n-1} \oplus x_n = 0) \cup \mathrm{prc}_0(y'_{n-1} \oplus \overline{x_n} = 0)$ $(\subseteq X_1(C,D))$ we derive $\mathrm{prc}_0(y_{n-1} = \overline{y'_{n-1}})$ in Figure 6.1.
2. For all $i \in \mathbb{N}$ from
$$\mathrm{prc}_0(y_{i-1} \oplus x_i = y_i) \cup \mathrm{prc}_0(y'_{i-1} \oplus x_i = y'_i) \cup \mathrm{prc}_0(y_i = \overline{y'_i})$$
we derive $\mathrm{prc}_0(y_{i-1} = \overline{y'_{i-1}})$ in Figure 6.2. Hence by induction on n from
$$\underbrace{\left(\bigcup_{i \in \{2,\ldots,n-1\}} \mathrm{prc}_0(y_{i-1} \oplus x_i = y_i) \cup \mathrm{prc}_0(y'_{i-1} \oplus x_i = y'_i)\right)}_{\subseteq X_1(C,D)} \cup \underbrace{\mathrm{prc}_0(y_{n-1} = \overline{y'_{n-1}})}_{\text{From step 1}}$$
we derive $\mathrm{prc}_0(y_2 = \overline{y'_2})$.
3. Finally, from
$$\underbrace{\mathrm{prc}_0(x_1 \oplus x_2 = y_2) \cup \mathrm{prc}_0(x_1 \oplus x_2 = y'_2)}_{\subseteq X_1(C,D)} \cup \underbrace{\mathrm{prc}_0(y_2 = \overline{y'_2})}_{\text{From step 2}}$$
we derive $\bot$ (see Figure 6.3).

From

$$\begin{aligned}\mathrm{prc}_0(y_{n-1} \oplus x_n = 0) &= \{\{\overline{y_{n-1}}, x_n\}, \{y_{n-1}, \overline{x_n}\}\} \\ \mathrm{prc}_0(y'_{n-1} \oplus \overline{x_n} = 0) &= \{\{\overline{y'_{n-1}}, \overline{x_n}\}, \{y'_{n-1}, x_n\}\},\end{aligned}$$

via 2-resolution we derive $\mathrm{prc}_0(y_{n-1} = \overline{y'_{n-1}}) = \{\{\overline{y_{n-1}}, \overline{y'_{n-1}}\}, \{y_{n-1}, y'_{n-1}\}\}$:

$$\frac{\{\overline{y_{n-1}}, x_n\} \qquad \{\overline{y'_{n-1}}, \overline{x_n}\}}{\{\overline{y_{n-1}}, \overline{y'_{n-1}}\}} \qquad \frac{\{y_{n-1}, \overline{x_n}\} \qquad \{y'_{n-1}, x_n\}}{\{y_{n-1}, y'_{n-1}\}}$$

Figure 6.1: Deriving $\mathrm{prc}_0(y_{n-1} = \overline{y'_{n-1}})$ from $\mathrm{prc}_0(y_{n-1} = x_n) \cup \mathrm{prc}_0(y'_{n-1} = \overline{x_n})$ via 2-resolution

From

$$\begin{aligned}\mathrm{prc}_0(y_{i-1} \oplus x_i = y_i) &= \{\underbrace{\{\overline{y_{i-1}}, \overline{x_i}, \overline{y_i}\}}_{\boldsymbol{C_1}}, \underbrace{\{\overline{y_{i-1}}, x_i, y_i\}}_{\boldsymbol{C_2}}, \underbrace{\{y_{i-1}, \overline{x_i}, y_i\}}_{\boldsymbol{C_3}}, \underbrace{\{y_{i-1}, x_i, \overline{y_i}\}}_{\boldsymbol{C_4}}\} \\ \mathrm{prc}_0(y'_{i-1} \oplus x_i = y'_i) &= \{\underbrace{\{\overline{y'_{i-1}}, \overline{x_i}, \overline{y'_i}\}}_{\boldsymbol{D_1}}, \underbrace{\{\overline{y'_{i-1}}, x_i, y'_i\}}_{\boldsymbol{D_2}}, \underbrace{\{y'_{i-1}, \overline{x_i}, y'_i\}}_{\boldsymbol{D_3}}, \underbrace{\{y'_{i-1}, x_i, \overline{y'_i}\}}_{\boldsymbol{D_4}}\} \\ \mathrm{prc}_0(y_i = \overline{y'_i}) &= \{\underbrace{\{\overline{y_i}, \overline{y'_i}\}}_{\boldsymbol{E_1}}, \underbrace{\{y_i, y'_i\}}_{\boldsymbol{E_2}}\}\end{aligned}$$

we derive $\mathrm{prc}_0(y_{i-1} = \overline{y'_{i-1}}) = \{\{\overline{y'_{i-1}}, \overline{y_{i-1}}\}, \{y'_{i-1}, y_{i-1}\}\}$:

$$\dfrac{\dfrac{\dfrac{C_1 \quad E_2}{\{\overline{y_{i-1}}, \overline{x_i}, y'_i\}}\, D_2 \qquad \dfrac{C_2 \quad E_1}{\{\overline{y_{i-1}}, x_i, \overline{y'_i}\}}\, D_1}{\{\overline{y'_{i-1}}, \overline{y_{i-1}}, y'_i\}} \qquad \dfrac{\{\overline{y'_{i-1}}, \overline{y_{i-1}}, \overline{y'_i}\}}{\boldsymbol{\{\overline{y'_{i-1}}, \overline{y_{i-1}}\}}}}{\boldsymbol{\{\overline{y'_{i-1}}, \overline{y_{i-1}}\}}} \qquad \dfrac{\dfrac{\dfrac{C_3 \quad E_1}{\{y_{i-1}, \overline{x_i}, \overline{y'_i}\}}\, D_4 \qquad \dfrac{C_4 \quad E_2}{\{y_{i-1}, x_i, y'_i\}}\, D_3}{\{y'_{i-1}, y_{i-1}, \overline{y'_i}\}} \qquad \dfrac{\{y'_{i-1}, y_{i-1}, y'_i\}}{\boldsymbol{\{y'_{i-1}, y_{i-1}\}}}}{\boldsymbol{\{y'_{i-1}, y_{i-1}\}}}$$

Figure 6.2: Deriving $\mathrm{prc}_0(y_{i-1} = \overline{y'_{i-1}})$ from $\mathrm{prc}_0(y_{i-1} \oplus x_i = y_i) \cup \mathrm{prc}_0(y'_{i-1} \oplus x_i = y'_i) \cup \mathrm{prc}_0(y_i = \overline{y'_i})$ via 3-resolution.

From

$$\begin{aligned}
\mathrm{prc}_0(x_1 \oplus x_2 = y_2) &= \{\underbrace{\{\overline{x_1}, \overline{x_2}, \overline{y_2}\}}_{\boldsymbol{C_1}}, \underbrace{\{\overline{x_1}, x_2, y_2\}}_{\boldsymbol{C_2}}, \underbrace{\{x_1, \overline{x_2}, y_2\}}_{\boldsymbol{C_3}}, \underbrace{\{x_1, x_2, \overline{y_2}\}}_{\boldsymbol{C_4}}\} \\
\mathrm{prc}_0(x_1 \oplus x_2 = y_2') &= \{\underbrace{\{\overline{x_1}, \overline{x_2}, \overline{y_2'}\}}_{\boldsymbol{D_1}}, \underbrace{\{\overline{x_1}, x_2, y_2'\}}_{\boldsymbol{D_2}}, \underbrace{\{x_1, \overline{x_2}, y_2'\}}_{\boldsymbol{D_3}}, \underbrace{\{x_1, x_2, \overline{y_2'}\}}_{\boldsymbol{D_4}}\} \\
\mathrm{prc}_0(y_2 = \overline{y_2'}) &= \{\underbrace{\{\overline{y_2}, \overline{y_2'}\}}_{\boldsymbol{E_1}}, \underbrace{\{y_2, y_2'\}}_{\boldsymbol{E_2}}\}
\end{aligned}$$

we derive $\bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{C_1 \quad E_2}{\{\overline{x_1}, \overline{x_2}, y_2'\}}\, D_1}{\{\overline{x_1}, \overline{x_2}\}} \quad \dfrac{\dfrac{C_2 \quad E_1}{\{\overline{x_1}, x_2, \overline{y_2'}\}}\, D_2}{\{\overline{x_1}, x_2\}}}{\{\overline{x_1}\}} \quad \dfrac{\dfrac{\dfrac{C_3 \quad E_1}{\{x_1, \overline{x_2}, \overline{y_2'}\}}\, D_3}{\{x_1, \overline{x_2}\}} \quad \dfrac{\dfrac{C_4 \quad E_2}{\{x_1, x_2, y_2'\}}\, D_4}{\{x_1, x_2\}}}{\{x_1\}}}{\bot}$$

Figure 6.3: Deriving $\bot$ from $\mathrm{prc}_0(x_1 \oplus x_2 = y_2) \cup \mathrm{prc}_0(x_1 \oplus x_2 = y_2') \cup \mathrm{prc}_0(y_2 = \overline{y_2'})$ via 3-resolution.

The number of clauses in this refutation (which uses only clauses of length at most 3) altogether is

1. $8n - 12$ clauses from T_n.
2. 2 clauses from the derivation of $\mathrm{prc}_0(y_{n-1} = \overline{y'_{n-1}})$ (see Figure 6.1).
3. $(n-3) \cdot 10$ clauses from $(n-3)$ induction steps (see Figure 6.2).
4. 11 clauses in the final refutation in step 3 (see Figure 6.3).

So in total, the resolution proof is of size $18n - 29$. □

Corollary 6.4.8 *The Tseitin translation, applied to a boolean circuit, has unbounded hardness in general, for the full form as well as the reduced form, as can be seen by the circuit computing via binary XORs in two chains the two sums $v_1 \oplus \cdots \oplus v_n$ and $v_1 \oplus \cdots \oplus \overline{v_n}$, and where the final circuit, computing the (single) output, is the equivalence of these two sums: The full Tseitin translation has hardness n by Theorem 6.4.7, and thus also the reduced Tseitin translation, which yields an (unsatisfiable) sub-clause-set, has hardness at least n.*

Chapter 7

Experiments

In this chapter some experiments are performed on the use of the three different mechanisms for representing boolean functions f studied in this thesis:

1. clause-sets F equivalent to f with $F \in \mathcal{UC}_k$ where k is as low as feasible;
2. the canonical translation $\mathrm{ct}(G)$ for a DNF-clause-set G equivalent to f;
3. and the reduced canonical translation $\mathrm{ct}^-(G)$.

The instances are described in Subsection 7.1, while the experimental results are discussed in Subsection 7.2. The focus is on gaining a better understanding of the interaction between solver behaviour and problem representation, and so we consider various representative complete SAT solvers. The tools for generating, running and analysing these experiments are available in the `OKlibrary` (see [113]).

7.1 The instances

For our experiments we want to take a boolean function $f_{k,h}$ as a constraint in a bigger SAT problem $G_{k,h}$. The "optimal" equivalent representation $F_{k,h}$ of $f_{k,h}$ shall have hardness k, and $f_{k,h}$ should also have a small equivalent DNF, so that the canonical and reduced canonical translation are available.

For $f_{k,h}$ we take the boolean functions from Theorem 5.3.12, which have the short CNFs (without new variables) $F_{k,h} := \mathrm{D}(\mathrm{F}^1(T_{k,h}))$ for $k \geq 2$ and $h \geq k+1$, where $T_{k,h} \in \mathrm{HS}(k,h)$. So $F_{k,h}$ has hardness k, while every equivalent clause-set of hardness at most $k-1$ contains at least $b(m) := \binom{m}{\lfloor \frac{m}{2} \rfloor}$ many clauses for $m := h - k$.

For the "completion" to $G_{k,h}$ let $F'_{k,h}$ be the negation of $F_{k,h}$ according to the remarks to Lemma 6.2.8, that is, $F'_{k,h}$ is obtained from $F_{k,h}$ by complementing the doping literals in each clause. Let $\overline{F} := \{\overline{C} : C \in F\}$ for $F \in \mathcal{CLS}$. Note that $\overline{F'_{k,h}}$ is the DNF for $F_{k,h}$. We define $G^i_{k,h}$ for $i = 1,2,3$ as always including $F'_{k,h}$, and additionally

1. $G^1_{k,h}$ uses $F_{k,h}$, i.e., $G^1_{k,h} := F'_{k,h} \cup F_{k,h}$.
2. $G^2_{k,h}$ uses the canonical translation according to Theorem 6.2.9 (and Lemma 6.2.8), i.e., $G^2_{k,h} := F'_{k,h} \cup \mathrm{ct}(\overline{F'_{k,h}})$.

3. $G^3_{k,h}$ uses the reduced canonical translation according to Lemma 6.3.3 (and Lemma 6.2.8), i.e., $G^3_{k,h} := F'_{k,h} \cup \mathrm{ct}^-(\overline{F'_{k,h}})$,

The sizes of the $G^i_{k,h}$ are determined as follows:

- By Lemma 5.3.9 we have $c(F_{k,h}) = \alpha(k,h)$, while $n(F_{k,h}) = 2c(F_{k,h}) - 1 = 2\alpha(k,h) - 1$.
- The size of $F'_{k,h}$ is precisely the same.
- So $n(G^1_{k,h}) = 2\alpha(k,h) - 1$ and $c(G^1_{k,h}) = 2\alpha(k,h)$, while $\ell(G^1_{k,h}) = 2\ell(F_{k,h})$.
- $n(G^2_{k,h}) = n(G^3_{k,h}) = 3\alpha(k,h) - 1$.
- $c(G^3_{k,h}) = 1 + \alpha(k,h) + \ell(F_{k,h})$ and $\ell(G^3_{k,h}) = \alpha(k,h) + 3\ell(F_{k,h})$.
- $c(G^2_{k,h}) = 1 + 2\alpha(k,h) + \ell(F_{k,h})$ and $\ell(G^3_{k,h}) = 2\alpha(k,h) + 4\ell(F_{k,h})$.

See Figure 7.1 for the numerical data. The lower bounds $b(h-k)$ there for the number of clauses in any clause-set F equivalent to $F_{k,h}$ and with $F \in \mathcal{WC}_{k-1}$ show that these representations are infeasible here. As an amusing fact one can note here that the number of clauses in $F \in \mathcal{WC}_0$ would be (precisely) $2^c - 1$, which even for the smallest example considered is a rather astronomical number. We can determine the hardness of the $G^i_{k,h}$ precisely; first an auxiliary lemma:

Lemma 7.1.1 *For $F \in \mathcal{CLS}$ and $F' \in \{\mathrm{ct}(\overline{F}), \mathrm{ct}^-(\overline{F})\}$ holds $F \cup F' \in \mathcal{USAT}$ and $\mathrm{hd}(F \cup F') \leq 2$.*

Proof: Due to $\mathrm{ct}^-(\overline{F}) \subseteq \mathrm{ct}(\overline{F})$ w.l.o.g. $F' = \mathrm{ct}^-(\overline{F})$, since $\mathcal{UC}_2 \cap \mathcal{USAT}$ is closed under formation of super-clause-sets by Lemma 6.7 in [78]. For all $C \in F$ and $x \in \overline{C}$ the binary clause $\mathrm{vct}^{\overline{C}} \to x$ is in F'. Thus setting $\mathrm{vct}^{\overline{C}}$ to 1 in $F \cup F'$ results in setting x to 1 via r_1, which altogether falsifies $C \in F$. Thus r_2 applied to $F \cup F'$ sets all $\mathrm{vct}^{\overline{C}}$ to 0, which falsifies $\{\mathrm{vct}^{\overline{C}} : C \in F\} \in F'$. □

We note that in the clause-set $F \cup F'$ of Lemma 7.1.1, the additional clauses of $F' = \mathrm{ct}(\overline{F})$ over $F' = \mathrm{ct}^-(\overline{F})$ are subsumed by the clauses of F, and thus here the difference between these two translations is very small. We can now determine the (total) hardness of the unsatisfiable SAT problems $G_{k,h}$ as follows:

Lemma 7.1.2 *Consider $k, h \in \mathbb{N}$ with $k \geq 2$ and $h \geq k+1$. We have:*

1. $\mathrm{hd}(G^1_{k,h}) = k+1$.
2. $\mathrm{hd}(G^2_{k,h}) = \mathrm{hd}(G^3_{k,h}) = 2$.

Proof: $\mathrm{hd}(G^1_{k,h}) = k+1$ follows from the fact, that by definition $G^1_{k,h} \in \mathcal{SMU}_{\delta=1}$ holds, where the corresponding tree $T := \mathrm{T}^1(G^1_{k,j})$ has Horton-Strahler number $k+1$ (recall Lemma 5.2.17): T is obtained from the underlying $T_{k,h}$ by replacing each leaf with the full binary tree with three nodes. $G^2_{k,h}$ and $G^3_{k,h}$ have hardness at least 2 since they are unsatisfiable and contain no unit-clauses. The remaining assertions follow by Lemma 7.1.1. □

7.2 Environment and solvers

For the experiments a 64-bit workstation with 32 GB RAM and Intel i5-2320 CPUs (6144 KB cache) running with 3 GHz was, where only a single CPU was employed. To emphasise again, the aim of these experiments was to obtain a qualitative picture of the behaviour of a range of contemporary SAT solvers, and not to find out which solver is "fastest". For the experimentation the following solvers[1)] were used, which give a good coverage of state of the art SAT solving and of the winners of recent SAT competitions and SAT races[2)]:

- Lookahead solvers (see [88, 114] for the general concepts):
  - `OKsolver` ([108]), a look-ahead solver with well-defined behaviour, no ad-hoc heuristics, and which applies complete r_2 (at every node). This solver won gold at the SAT 2002 competition.
  - `satz` ([119]), a look-ahead solver which applies partial r_2 and r_3. In the `OKlibrary` version 215 is maintained, with improved/corrected in/output and coding standard.
  - `march_pl` ([87]), a look-ahead solver applying partial r_2, r_3, and resolution- and equivalence-preprocessing. `march_pl` contains the same underlying technology as its sibling solvers `march_{rw,hi,ks,dl,eq}`, which won gold, silver and bronze at the 2004 to 2011 SAT competitions and SAT races. The `pl` (partial lookahead) version is used here.
- Conflict-driven solvers (see [121] for an overview):
  - `MiniSat` family:
    * `MiniSat` ([54, 151]), version 2.2.0, the latest version of the well-established `MiniSat` solver, used as starting point for many new conflict-driven solvers. Previous versions won gold at the SAT Race 2006 and 2008, as well as numerous bronze and silver awards at the SAT competition 2007. This solver is used as a standard reference at recent SAT competitions and races, and is referenced extensively in the literature.
    * `CryptoMiniSat` ([150, 149]), a `MiniSat` derivative designed specifically to tackle hard cryptographic problems. This solver won gold at SAT Race 2010 and gold and silver at the SAT competition 2011. Version 2.9.6 is used here.
    * `Glucose` ([5]), a `MiniSat` derivative utilising a new clause scoring scheme and aggressive learnt-clause deletion. This solver won gold in both SAT 2011 competition and SAT Challenge 2012. Versions 2.0 and 2.2 are used here.
  - `Lingeling` family:
    * `PicoSAT` ([18, 20]), a conflict-driven solver using an aggressive restart strategy, compact data-structures, and offering proof-trace options to allow for unsatisfiability checking. This solver won gold and silver at the SAT competition 2007. The latest version 913 is used here.
    * `PrecoSAT` ([19]), integrates the `SATeLite` preprocessor into `PicoSAT`, applying various reductions including partial r_2 at certain nodes in the search tree. This

[1)] all of them are available, together with build tools and the generators, in the `OKlibrary` ([113], `http://www.ok-sat-library.org`)

[2)] The (parent) SAT competition homepage is at `http://www.satcompetition.org` with links to each individual competition. Analyses of the SAT 2002–2004 competitions are available in [145, 14, 15]; analyses of the performance of SAT solvers on the random resp. industrial benchmarks can be found in [111, 163]; and benchmark and solver descriptions for the SAT competition 2009 resp. SAT Challenge 2012 are available in [13, 9].

solver won gold and silver at the SAT 2009 competition. The latest version 570 is used here.

* `Lingeling` ([20, 21]), based on `PrecoSAT`, focuses further on integrating preprocessing and search, introducing new algorithms and data-structures to speed up these techniques and reduce memory footprint. As with `PrecoSAT`, this solver applies partial r_2 at specially chosen nodes in the search tree. This solver won bronze at the SAT 2011 competition and silver at the SAT Race 2010. The latest version `ala-b02aa1a-121013` is used here.

7.3 Look-ahead solvers

First we consider the `OKsolver` (see Figure 7.1 for the data table and Figure 7.6 for the graph), as a look-ahead solver (see [88, 114] for the general concepts), as well as a solver with a "clean" behaviour, due to the minimisation of the use of heuristical shortcuts. For example, the `OKsolver` seems to be the only SAT-solver computing r_2, while all other solvers (recall the discussion in Subsection 1.4) only test selected literals for failed literals. We see that the $G^1_{k,h}$ are far easier than the $G^{2,3}_{k,h}$, although they require branching. Indeed, the straightforward heuristics choosing a variable occurring most often will find a backtracking tree of optimal, i.e., linear size (note that all $F \in \mathcal{SMU}_{\delta=1}$ have exactly one variable occurring in every clause, and splitting on this variable creates two variable-disjoint instances). In conformance with this, linear regression shows with high correlation that the instances $G^1_{k,h}$ are solved by the `OKsolver` in linear time, i.e., $O(\ell)$. Considering now $G^{2,3}_{k,h}$, recall that by Lemma 7.1.2 all these instances have hardness 2, that is, they can be solved without branching. And in fact the number of r_2-reductions of the `OKsolver` for these instances is precisely $c(F_{k,h}) - 1$, in accordance with Lemma 7.1.1. The worst-case running time for r_2 is $O(n^2 \cdot \ell)$, but in this case going once through the list of all variables is sufficient to find the contradiction. Again in conformance with this, linear regression shows with high correlation that the instances $G^{2,3}_{k,h}$ are solved in time $O(n \cdot \ell)$. We note here that the `OKsolver` is actually the fastest solver on these instances, for all three types, though this is not the focus of these experiments.

Other look-ahead solvers are not efficient on these instances:

- `satz` (see Figure 7.2 for the data table and Figure 7.7 for the graph) shows that finding a short resolution refutation is not guaranteed, even on the easiest instances: on $G^1_{2,22}$ it needed 4.1 sec, and on $G^1_{2,32}$ already 6878 sec, while the number of nodes for $G^1_{2,h}$ in general is (precisely) $2^{h-1} - 1$, thus showing a stable exponential behaviour.

- `march_pl` performed somewhat better, but was also not able to complete even the easier instances $G^1_{k,h}$; furthermore it crashed on various instances, and was thus also not considered further.

7.4 Conflict-driven solvers

Now we turn to the conflict-driven solvers (see [121] for a general introduction), where we consider the `MiniSat`-family (see Figure 7.3 and Figure 7.4 for the data tables and Figures 7.8, 7.9, 7.10, 7.11, 7.12, 7.13 for graphs) and the `Lingeling`-family (see Figure 7.5 for the data table and Figures 7.14, 7.15, and 7.16 for graphs). Considering $G^1_{k,h}$, we note that `MiniSat` as well as `PrecoSAT` always solve these instances by preprocessing (i.e., no decision/branching, no

conflicts). And actually `MiniSat -no-pre` (without preprocessing) solves these instances faster (by branching) than with preprocessing, as do the **Glucose** solvers. While `PicoSAT`, which also does not use preprocessing, is not much slower than `PrecoSAT`. With the largest instance $G^1_{5,35}$, all solvers except of `PrecoSAT` have considerable difficulties, but all can handle it (only `PicoSAT` aborts, likely due to a bug). That `OKsolver` is much faster here we believe is due to the fact, that in general look-ahead solvers should be better than conflict-driven solvers on unsatisfiable instances, where the shortest refutations are (close to) tree-like (and in this case the tree-like refutation of $F \in \mathcal{SMU}_{\delta=1}$ given by the underlying tree $\mathrm{T}^1(F)$ is the shortest possible overall).

Turning to $G^{2,3}_{k,h}$, we see that `CryptoMiniSat` as well as `PicoSAT` solve the easier instances with failed-literal elimination (without branching). Most of the time these instances are harder than their $G^1_{k,h}$ counterparts, and for $k \in \{4,5\}$ much more so, and actually no solver here was able to handle $k = 5$ with $h = 35$. There seems to be no essential difference between $G^2_{k,h}$ and $G^3_{k,h}$ (different from the `OKsolver`, whose running time was proportionally larger for $G^2_{k,h}$, according to the bigger size).

7.5 Discussion

Unsatisfiable clause-sets $G^i_{k,h}$ have been presented, which have a fixed part $F'_{k,h}$ and a "constraint" $F_{k,h}$ in three different representations, namely $F_{k,h}$ itself for $G^1_{k,h}$, the canonical translation for $G^2_{k,h}$, and the reduced canonical translation for $G^3_{k,h}$. It is worth considering a fourth "hidden" representation, namely some clause-set $F''_{k,h}$ equivalent to $F'_{k,h}$ with $\mathrm{hd}(F''_{k,h}) \leq k-1$ and minimal $c(F''_{k,h})$, yielding $G^0_{k,h}$.

The first goal of these experiments was to show that $G^0_{k,h}$ is a very bad problem representation, and this should be evident by inspecting Figure 7.1, where the numerical values for the lower bound $c(F''_{k,h}) > b(h-k)$ are presented. On the other hand, $G^1_{k,h}$ is for all solvers except of `satz` a very easy instance. This shows that there are cases where to represent a boolean function f by an equivalent clause-set F of some hardness k, the value of k is decisive. Note that the criterion of "maintaining arc-consistency" here means to choose $k = 1$, and thus the sequence $G^0_{2,h}$ demonstrates the infeasibility of arc-consistency without new variables (and that using just failed-literal reduction can solve the problem).

If the boolean function f has a short circuit, then we can use the Tseitin translation, in the full or the reduced form, to obtain a short representation of f. In Theorem 6.2.9 and in Lemma 6.3.3 we have seen two situations, where starting from a DNF we obtain a representation in $\mathcal{UC}_1$. These two tools apply here, and we get $G^2_{k,h}$ and $G^3_{k,h}$ from them. The question arises which of the three representations $G^i_{k,h}$ for $i \in \{1,2,3\}$ is best?

The experiments clearly show that here $G^1_{k,h}$ performs best, even in this special situation, where $G^{2,3}_{k,h}$ have very special properties (namely they have hardness 2). It seems that there is a general pattern: If for a boolean function f we have an equivalent CNF F of bounded ("small") hardness, which is not much bigger than an equivalent DNF F', then F will perform better for SAT solving than the canonical translation (reduced or full) of F'.

			Instance statistics					OKsolver statistics		
k	h	i	n	c	ℓ	$\alpha(k,h)$	$b(h-k)$	t (sec)	nds	r_2
2	22	1	507	508	8604	254	$1.8 \cdot 10^6$	0.0	43	232
		2	761	4811	17716			0.0	1	253
		3	761	4557	13160			0.0	1	253
	32	1	1057	1058	24994	529	$1.6 \cdot 10^9$	0.0	63	497
		2	1586	13556	51046			0.0	1	528
		3	1586	13027	38020			0.0	1	528
	42	1	1807	1808	54784	904	$1.4 \cdot 10^{12}$	0.0	83	862
		2	2711	29201	111,376			0.1	1	903
		3	2711	28297	83080			0.1	1	903
	52	1	2757	2758	101,974	1379	$1.3 \cdot 10^{15}$	0.1	103	1327
		2	4136	53746	206,706			0.4	1	1378
		3	4136	52367	154,340			0.2	1	1378
	62	1	3907	3908	170,564	1954	$1.2 \cdot 10^{18}$	0.2	123	1892
		2	5861	89191	345,036			1.0	1	1953
		3	5861	87237	257,800			0.5	1	1953
	72	1	5257	5258	264,554	2629	$1.1 \cdot 10^{21}$	0.4	143	2557
		2	7886	137,536	534,366			4.0	1	2628
		3	7886	134,907	399,460			1.0	1	2628
3	23	1	4095	4096	80594	2048	$1.8 \cdot 10^6$	0.0	507	1794
		2	6143	44394	165,284			0.2	1	2047
		3	6143	42346	122,939			0.1	1	2047
	33	1	12035	12036	327,384	6018	$1.6 \cdot 10^9$	0.2	1057	5489
		2	18053	175,729	666,804			4.8	1	6017
		3	18053	169,711	497,094			1.6	1	6017
	43	1	26575	26576	922,524	13288	$1.4 \cdot 10^{12}$	1.0	1807	12384
		2	39863	487,839	1,871,624			82.6	1	13287
		3	39863	474,551	1,397,074			28.9	1	13287
4	24	1	25901	25902	562,542	12951	$1.8 \cdot 10^6$	0.4	4095	10903
		2	38852	307,174	1,150,986			15.4	1	12950
		3	38852	294,223	856,764			4.5	1	12950
	34	1	105,911	105,912	3,150,408	52,956	$1.6 \cdot 10^9$	3.3	12035	46938
		2	158,867	1,681,117	6,406,728			843.4	1	52955
		3	158,867	1,628,161	4,778,568			410.8	1	52955
	44	1	299,971	299,972	11,326,724	149,986	$1.4 \cdot 10^{12}$	16.5	26575	136,698
		2	449,957	5,963,335	22,953,420			10233	1	149,985
		3	449,957	5,813,349	17,140,072			5296	1	149,985
5	25	1	136,811	136,812	3,202,912	68406	$1.8 \cdot 10^6$	2.7	25901	55455
		2	205,217	1,738,269	6,542,636			664.6	1	68405
		3	205,217	1,669,863	4,872,774			348.7	1	68405
	35	1	768,335	768,336	24,413,776	384,168	$1.6 \cdot 10^9$	31.2	105,911	331,212
		2	1,152,503	12,975,225	49,595,888			36743	1	384,167
		3	1,152,503	12,591,057	37,004,832			20062	1	384,167

Figure 7.1: Instance statistics for $G^i_{k,h}$, and solver statistics for the `OKsolver` with option "no-tree-pruning", turning off the intelligent backtracking, which consumes too much memory for the larger instances. "nds" is the number of nodes of the backtracking tree, while "r_2" is the number of r_2-reductions $F \rightsquigarrow r_2(r_1(\langle x \to 1\rangle * F))$ in case of $r_1(\langle x \to 0\rangle * F) = \{\bot\}$.

			satz		
k	h	i	t (sec)	nds	r_2
2	22	2	0.1	37	246
		3	0.0		
	32	2	0.7	57	511
		3	0.3		
	42	2	3.9	77	876
		3	1.6		
	52	2	13.8	97	1341
		3	5.7		
	62	2	37.7	117	1906
		3	16.7		
	72	2	89.9	137	2571
		3	39.5		
3	23	2	7.8	381	2055
		3	2.8		
	33	2	161.8	871	5890
		3	66.9		
	43	2	1327	1561	12925
		3	521.6		
4	24	2	469.5	2701	13481
		3	205.4		
	34	2	12957	9051	52896
		3	5666		
	44	2	143,558	21401	147,436
		3	60673		
5	25	2	13420	15093	73399
		3	5518		
	35	2	609,056	72913	392,371
		3	250,076		

Figure 7.2: Solver times for $G^i_{k,h}$ for satz

			MiniSat			MiniSat -no-pre			CryptoMiniSat		
k	h	i	t (sec)	decisions	confl	t (sec)	decisions	confl	t (sec)	decisions	confl
2	22	1	0.0	0	0	0.0	10227	365	0.0	1832	20
		2	0.0	1134	136	0.0	6706	416	0.0	0	0
		3	0.0	1134	136	0.0	6706	416	0.0	0	0
	32	1	0.0	0	0	0.0	36646	795	0.0	2165	25
		2	0.0	12816	740	0.0	18301	905	0.0	0	0
		3	0.0	12816	740	0.0	18301	905	0.0	0	0
	42	1	0.1	0	0	0.0	133.105	1366	0.0	5798	53
		2	0.2	29334	1529	0.1	32008	1563	0.0	0	0
		3	0.1	29334	1529	0.1	32008	1563	0.0	0	0
	52	1	0.2	0	0	0.0	206,925	1962	0.1	12291	50
		2	0.6	65019	2778	0.2	79259	2496	0.0	0	0
		3	0.6	65019	2778	0.2	79259	2496	0.0	0	0
	62	1	0.5	0	0	0.1	482,733	2861	0.2	21874	55
		2	1.1	129,523	3887	0.4	158,975	3697	0.1	0	0
		3	1.0	129,523	3887	0.4	158,975	3697	0.1	0	0
	72	1	1.1	0	0	0.1	533,500	3963	0.5	24866	56
		2	2.7	165,596	5417	1.4	137,582	4981	0.1	0	0
		3	2.4	165,596	5417	1.3	137,582	4981	0.1	0	0
3	23	1	0.2	0	0	0.1	726,328	3344	0.1	66885	349
		2	0.1	34343	1276	0.5	87038	2719	0.1	0	0
		3	0.1	34343	1276	0.5	84922	2683	0.1	0	0
	33	1	2.1	0	0	0.6	6,024,786	10163	0.7	767,860	1426
		2	15.0	245,555	8333	10.4	293,488	8213	0.4	0	0
		3	14.8	244,410	8272	10.5	303,064	8217	0.4	0	0
	43	1	14.6	0	0	3.3	30,413,289	23355	4.8	11,673,409	12547
		2	132.5	886,834	20033	89.5	764,994	20101	1.0	0	0
		3	134.7	837,910	19939	91.0	808,130	19817	1.2	0	0
4	24	1	5.5	0	0	3.4	26,310,775	23307	2.5	10,823,044	14335
		2	10.1	351,753	10427	71.8	603,915	15761	1.1	0	0
		3	9.9	351,468	10330	62.2	603,915	15761	1.4	0	0
	34	1	149.5	0	0	62.2	510,575,547	88280	57.8	121,886,023	65608
		2	6381	3,851,979	72123	5376	4,762,174	69651	706.2	1,080,246	30501
		3	6894	4,265,009	70250	4749	4,762,174	69651	614.8	1,165,228	30500
	44	1	2117	0	0	475.7	4,225,934,440	232,867	538.6	1,756,703,536	332,497
		2	**A**17749	10,905,675	62092	**A**50777	16,691,952	192,830	34461	5,708,264	114,958
		3	**S**			**A**31985	14,856,654	155,899	34850	4,565,988	105,312
5	25	1	143.3	0	0	74.9	702,026,588	109,898	76.5	168,438,898	66235
		2	3282	3,391,255	67344	4044	5,413,350	82751	1323	1,336,804	30561
		3	3209	3,202,774	66739	4058	5,413,350	82751	1283	1,281,716	30500
	35	1	11636	0	0	2633	30,154,061,700	608,180	4440	16,767,014,292	942,020
		2	**A**90649	9,481,265	68589	**S**			**L**		
		3	**A**60657	8,729,650	52968	**A**36000			**L**		

Figure 7.3: Solver times for $G^i_{k,h}$ for default `MiniSat` and `MiniSat -no-pre` (turning off preprocessing), and `CryptoMiniSat`. "S" marks a segmentation fault of the solver, "L" marks that the solver failed due to "too long clauses", and "A" marks a user-abortion.

			Glucose-2.0			Glucose-2.2			Glucose-2.2 -no-pre		
k	h	i	t (sec)	decisions	confl	t (sec)	decisions	confl	t (sec)	decisions	confl
2	22	1	0.0	11763	350	0.0	0	0	0.0	11763	350
		2	0.0	6433	383	0.0	1338	117	0.0	4528	372
		3	0.0	6442	400	0.0	1338	117	0.0	4525	372
	32	1	0.0	27529	853	0.0	0	0	0.0	28632	828
		2	0.0	19317	934	0.0	11854	821	0.0	14790	944
		3	0.0	21288	929	0.0	11854	821	0.0	14790	944
	42	1	0.0	113,533	1371	0.1	0	0	0.0	142,244	1337
		2	0.1	32674	1594	0.3	45090	1815	0.1	27105	1581
		3	0.1	32472	1544	0.3	37014	1810	0.1	27378	1586
	52	1	0.0	204,314	2017	0.2	0	0	0.0	248,484	2098
		2	0.2	44778	2694	0.5	67688	2780	0.2	69119	3149
		3	0.2	44535	2588	0.4	70132	2855	0.2	69122	3153
	62	1	0.1	256,891	3003	0.5	0	0	0.1	642,210	2869
		2	0.5	79563	3786	1.0	113,817	4049	0.4	113,672	4127
		3	0.5	79604	3800	1.0	113,369	4034	0.4	114,158	4072
	72	1	0.2	1,200,842	3969	1.1	0	0	0.1	713,326	4134
		2	1.0	152,734	6127	1.3	117,395	5428	0.8	212,232	5707
		3	1.0	152,734	6127	1.3	117,395	5428	0.8	211,843	5751
3	23	1	0.1	293,599	3642	0.2	0	0	0.1	506,971	3175
		2	0.4	50429	2601	0.1	31991	1203	0.5	85427	2681
		3	0.4	50883	2626	0.1	32318	1203	0.5	73480	2711
	33	1	0.5	2,573,291	9976	2.2	0	0	0.5	3,840,023	9642
		2	19.0	310,569	8770	16.7	290,082	8357	7.6	280,477	8222
		3	13.0	264,088	8633	16.4	317,545	8725	11.4	274,384	8333
	43	1	2.5	12,387,073	21567	14.7	0	0	2.0	16,213,348	21837
		2	98.7	834,345	22453	169.9	933,775	21365	102.6	997,026	21333
		3	107.9	894,762	22331	180.3	824,482	20823	108.3	1,082,633	21424
4	24	1	3.0	13,739,340	23265	5.8	0	0	2.8	24,791,124	21260
		2	93.4	746,936	16170	11.8	413,088	10756	75.2	815,951	16919
		3	73.6	624,916	16463	12.5	433,902	10558	93.5	782,268	17800
	34	1	73.1	404,205,131	92344	159.1	0	0	43.9	421,363,723	85660
		2	5889	3,856,879	73007	9479	4,159,301	79382	6503	5,511,016	79964
		3	5557	3,857,144	75795	7804	4,121,835	73212	5428	5,344,150	73399
	44	1	539.0	3,658,524,320	287,335	1911	0	0	537.9	5,428,781,274	269,273
		2	**A**32100			**A**90780			**A**90960		
		3	**A**31080			**A**68400			**A**104,100		
5	25	1	102.2	731,691,363	129,523	152.1	0	0	115.8	899,226,706	118,088
		2	12922	4,993,251	83431	4903	4,080,112	74150	9027	8,034,854	83562
		3	11711	5,333,175	83959	3116	3,466,754	70752	9720	7,240,354	81337
	35	1	4250	30,080,297,160	816,139	9959	0	0	2729	30,938,594,432	700,202
		2	**A**40440			**A**374,400			**A**86460		
		3	**A**32280			**A**77880			**A**81300		

Figure 7.4: Solver times for $G^i_{k,h}$ for default `Glucose-2.0`, `Glucose-2.2`, and `Glucose-2.2 -no-pre` (turning off pre-processing). "A" marks a user-abortion.

			Lingeling			PrecoSAT			PicoSAT		
k	h	i	t (sec)	decisions	confl	t (sec)	decisions	confl	t (sec)	decisions	confl
2	22	1	0.0	31414	100	0.0	0	1	0.0	5832	254
		2	0.0	972	100	0.0	20	16	0.0	0	254
		3	0.0	972	100	0.0	20	16	0.0	0	254
	32	1	0.2	55014	100	0.0	0	1	0.0	21843	585
		2	0.0	1593	100	0.1	18	20	0.0	0	529
		3	0.0	1593	100	0.1	18	20	0.0	0	529
	42	1	0.3	106,962	100	0.0	0	1	0.0	39980	964
		2	0.0	2413	100	0.4	39798	560	0.0	0	904
		3	0.0	2413	100	0.4	39798	560	0.0	0	904
	52	1	0.5	195,342	100	0.1	0	1	0.0	87623	1411
		2	0.1	3432	100	0.9	135,771	1438	0.1	0	1379
		3	0.1	3432	100	1.0	135,771	1438	0.0	0	1379
	62	1	2.1	2,908,253	1528	0.1	0	1	0.1	195,811	2023
		2	1.4	9993	338	2.0	268,652	2398	0.1	0	1954
		3	1.2	8754	343	2.2	268,652	2398	0.1	0	1954
	72	1	3.8	5,780,521	2100	0.2	0	1	0.1	358,396	2689
		2	3.4	41069	835	4.1	452,024	3493	0.2	0	2629
		3	1.6	17563	745	4.4	452,024	3493	0.1	0	2629
3	23	1	0.9	772,664	655	0.0	0	1	0.1	373,029	2217
		2	0.3	4730	100	0.9	21	17	0.1	0	2048
		3	0.3	4730	100	0.9	21	17	0.1	0	2048
	33	1	7.0	11,494,104	4470	0.2	0	1	0.4	1,832,220	6261
		2	15.6	133,145	4822	8.8	9954	209	3.6	301,757	7808
		3	34.8	187,408	4941	8.7	9954	209	3.8	397,594	7774
	43	1	54.6	103,646,649	13585	1.1	0	1	1.5	6,710,296	13635
		2	834.3	1,058,591	28616	53.9	2,137,226	17295	125.1	2,259,244	20567
		3	683.6	920,917	29862	54.7	2,137,226	17295	125.2	2,378,109	20760
4	24	1	33.2	61,516,109	13324	0.5	0	1	1.3	7,197,271	13337
		2	201.0	420,199	19113	30.4	87937	857	44.5	730,302	16283
		3	411.0	813,880	20978	29.9	87937	857	53.5	899,721	16270
	34	1	389.4	736,985,317	54187	9.2	0	1	15.9	77,852,480	54002
		2	25004	4,431,011	103,069	751.9	37,282,690	64688	5110	4,952,348	73501
		3	18593	5,206,665	119,524	735.8	37,282,690	64688	5822	6,354,378	73540
	44	1	3139	5,980,879,353	152,934	94.2	0	1	135.7	524,180,945	152,931
		2	**A**94270	7,284,838	72027	35356	1,035,463,259	410,510	**M**		
		3	**A**60882	7,688,765	71408	44808	1,035,463,259	410,510	**M**		
5	25	1	479.6	903,741,154	70177	10.1	0	1	25.6	120,756,190	69336
		2	37201	5,026,759	124,208	3636	31,539,722	31092	5523	4,436,819	83821
		3	19148	3,958,185	117,605	3484	31,539,722	31092	6540	5,328,658	83829
	35	1	14845	28,147,090,014	392,047	389.0	0	1	**F**478.5		
		2	**A**687,866	11,495,987	39217	**F**49440			**M**		
		3	**A**94779	6,939,217	30146	**F**49527			**M**		

Figure 7.5: Solver times for $G^i_{k,h}$ for Lingeling, PrecoSAT, and PicoSAT. "M" marks a failure of the solver due to "out of memory", "F" marks a self-declared failure of the solver, and "A" marks a user-abortion.

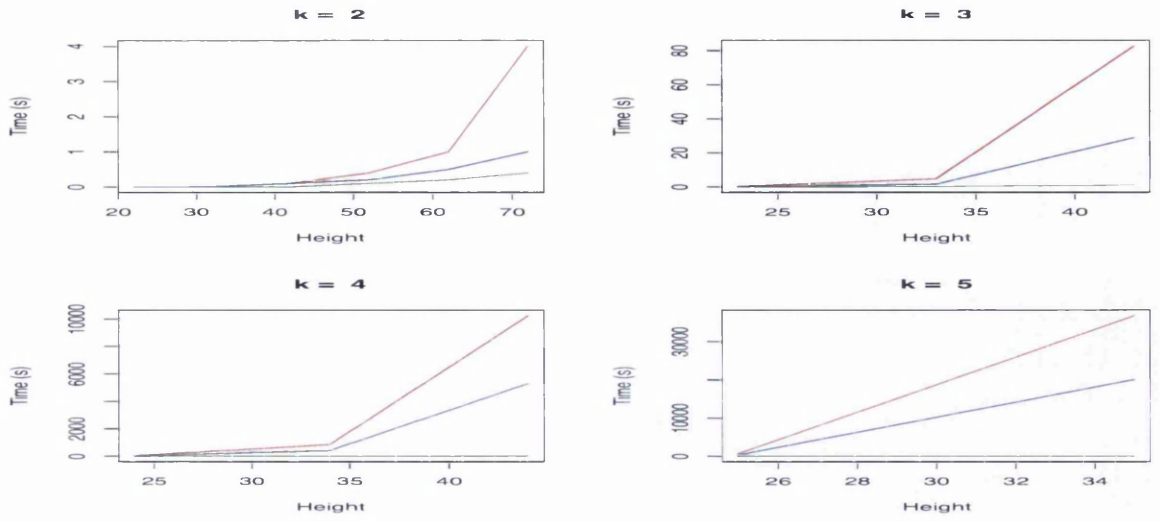


Figure 7.6: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `OKsolver` with option "no-tree-pruning". The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.1 for the full details.

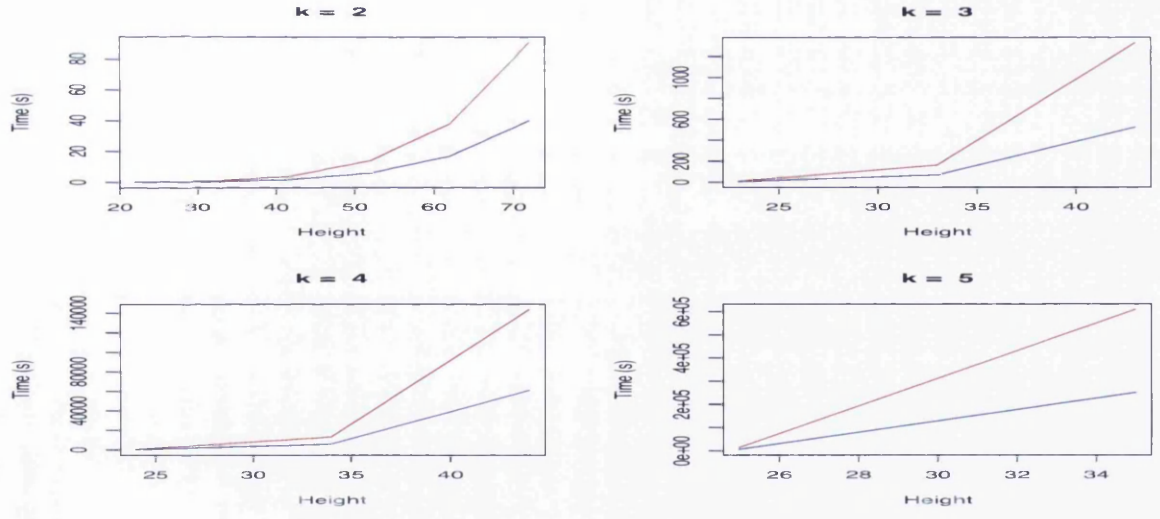


Figure 7.7: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `Satz`. The lines for $i = 2$ and 3 are coloured blue and red respectively; $i = 1$ is missing as explained in Section 7.3. See Figure 7.2 for the full details.

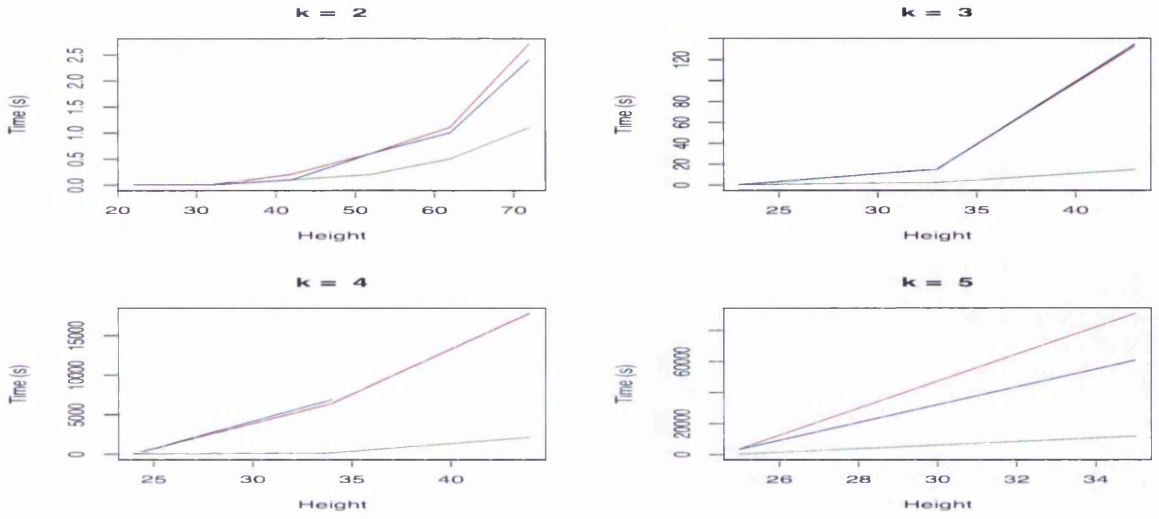


Figure 7.8: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `MiniSat`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.3 for the full details.

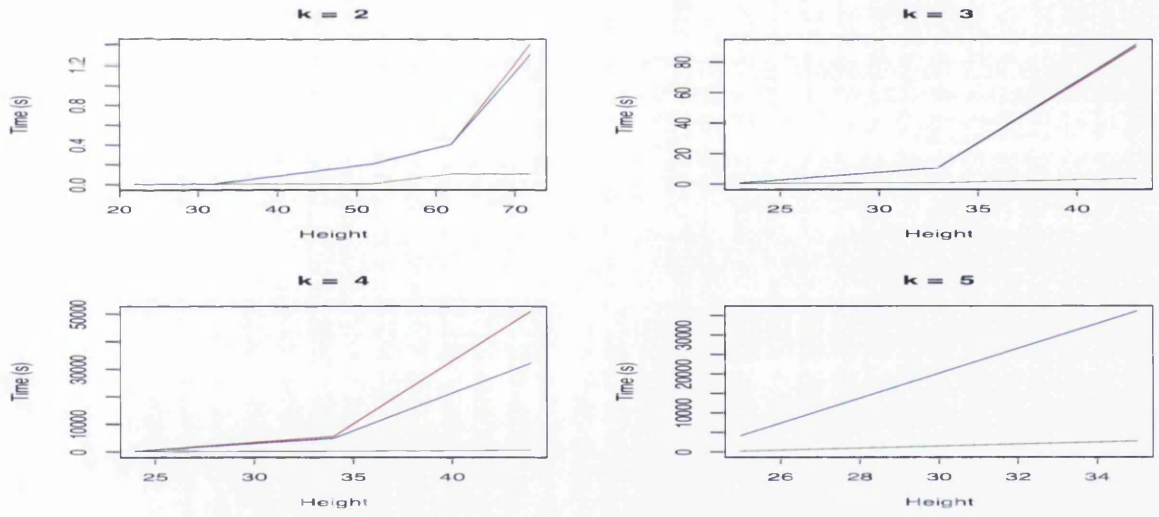


Figure 7.9: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `MiniSat -no-pre` (turning off pre-processing). The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.3 for the full details.

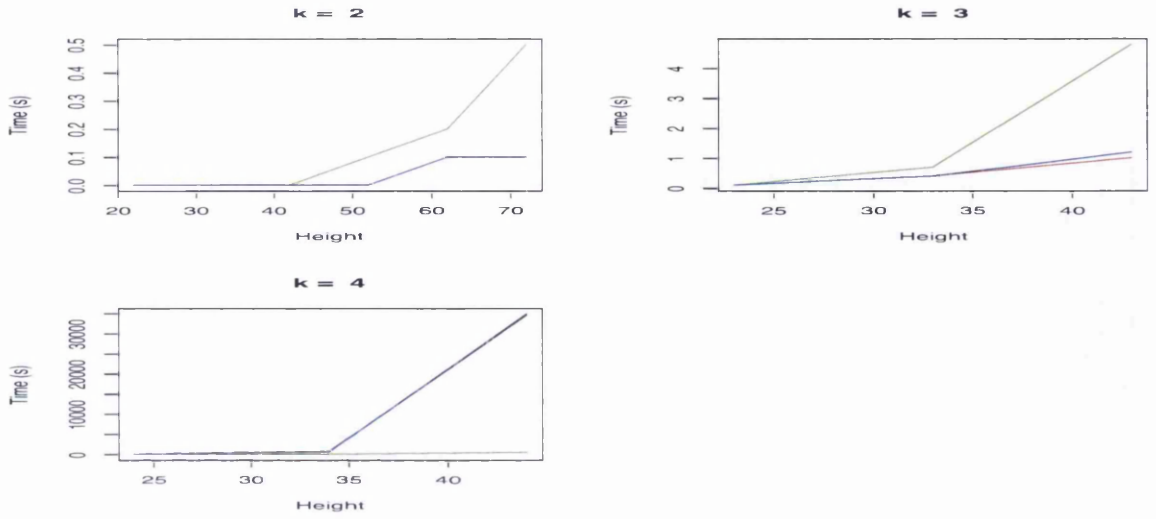


Figure 7.10: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `CryptoMiniSat`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively; note that data is missing for $k = 5$ as there is only a single value for $i = 2$ and $i = 3$. See Figure 7.3 for the full details.

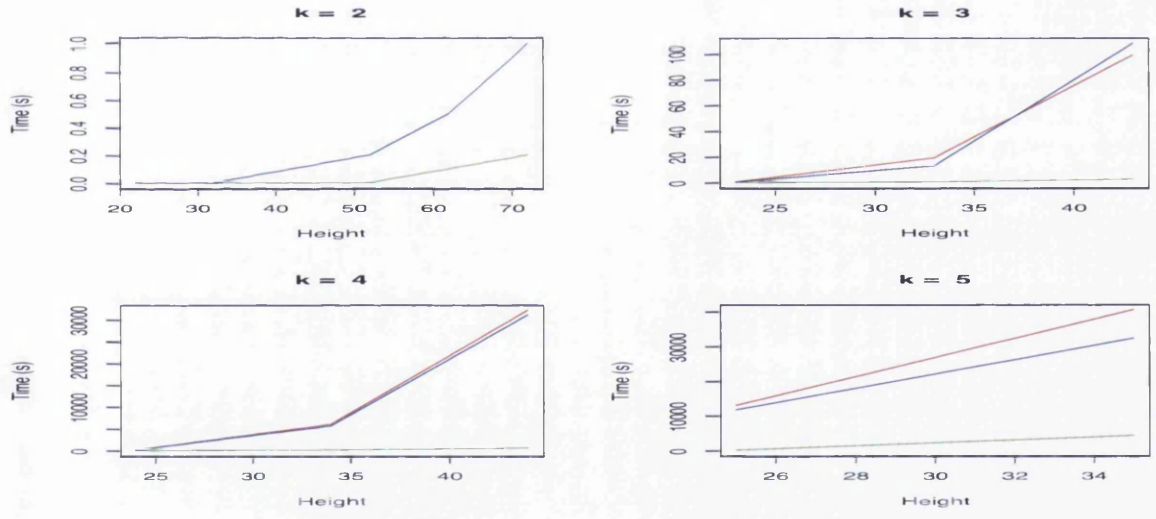


Figure 7.11: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `Glucose-2.0`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.4 for the full details.

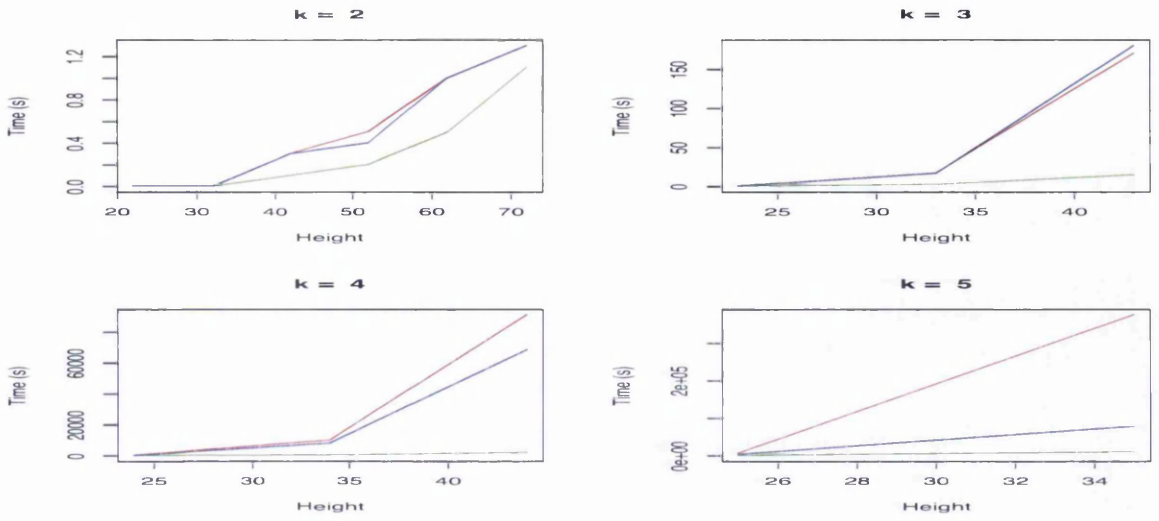


Figure 7.12: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `Glucose-2.2`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.4 for the full details.

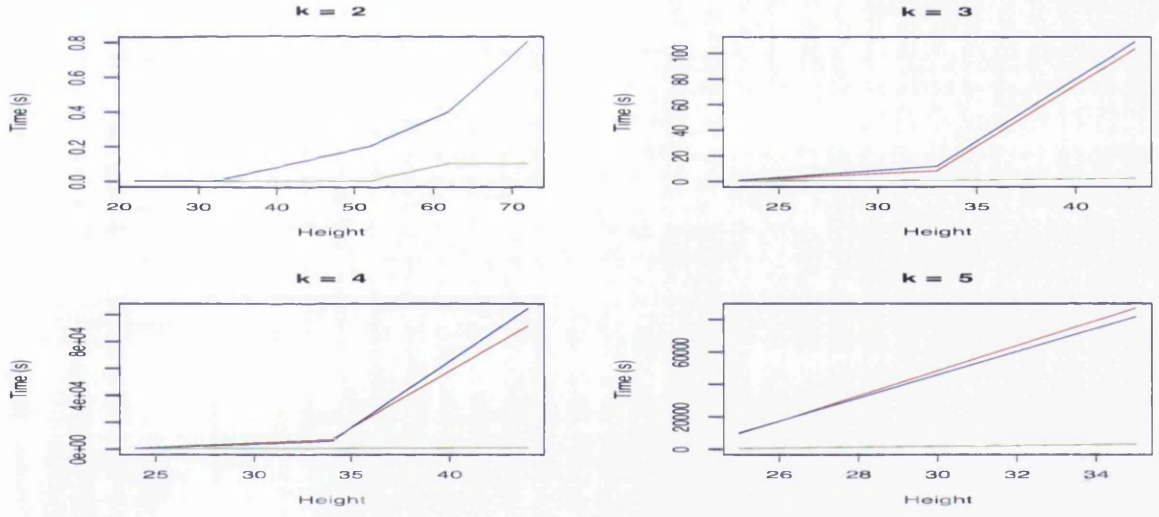


Figure 7.13: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `Glucose-2.2 -no-pre`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.4 for the full details.

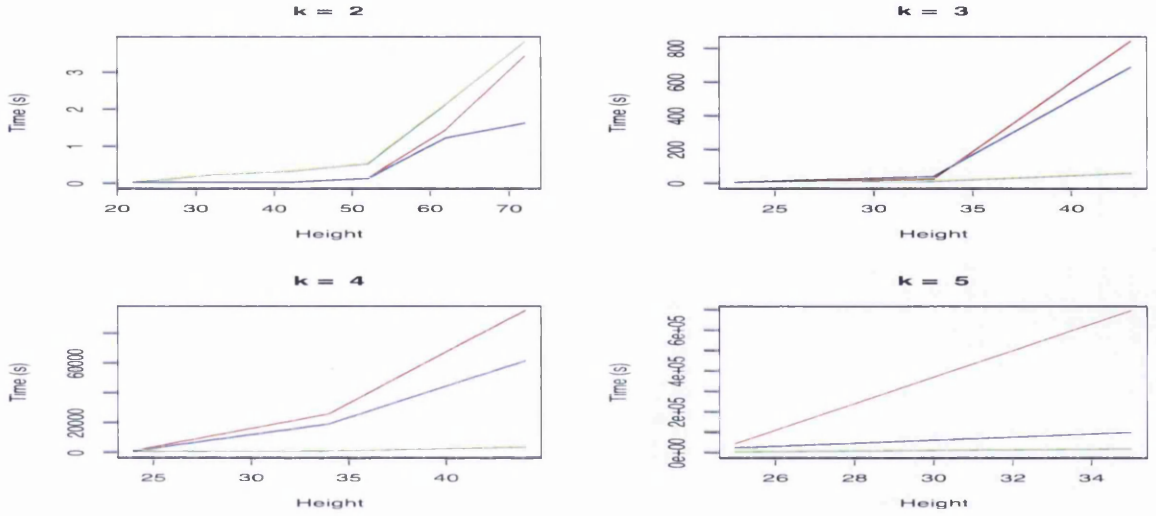


Figure 7.14: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `Lingeling`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.5 for the full details.

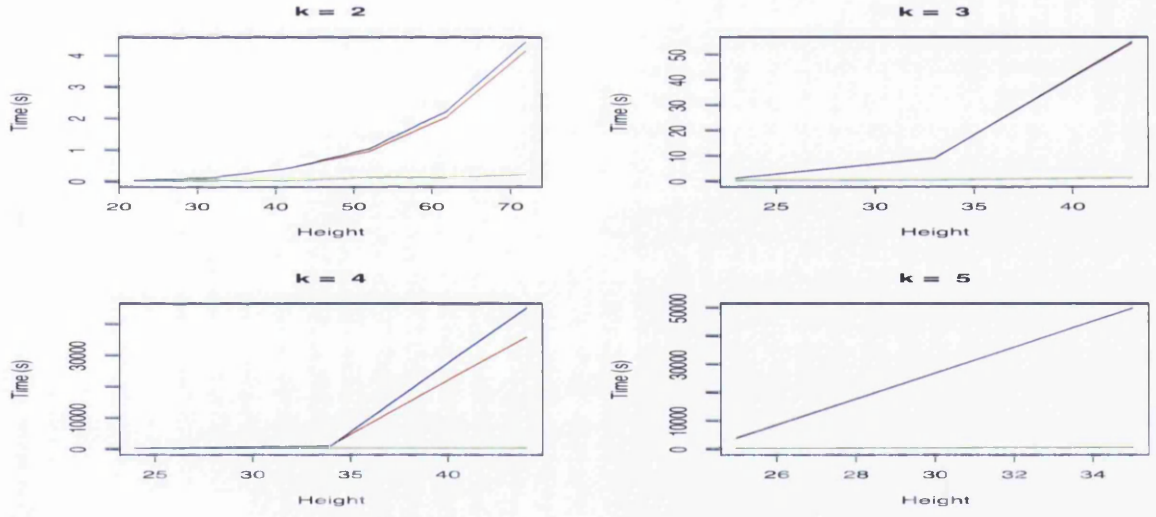


Figure 7.15: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `PrecoSAT`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively. See Figure 7.5 for the full details.

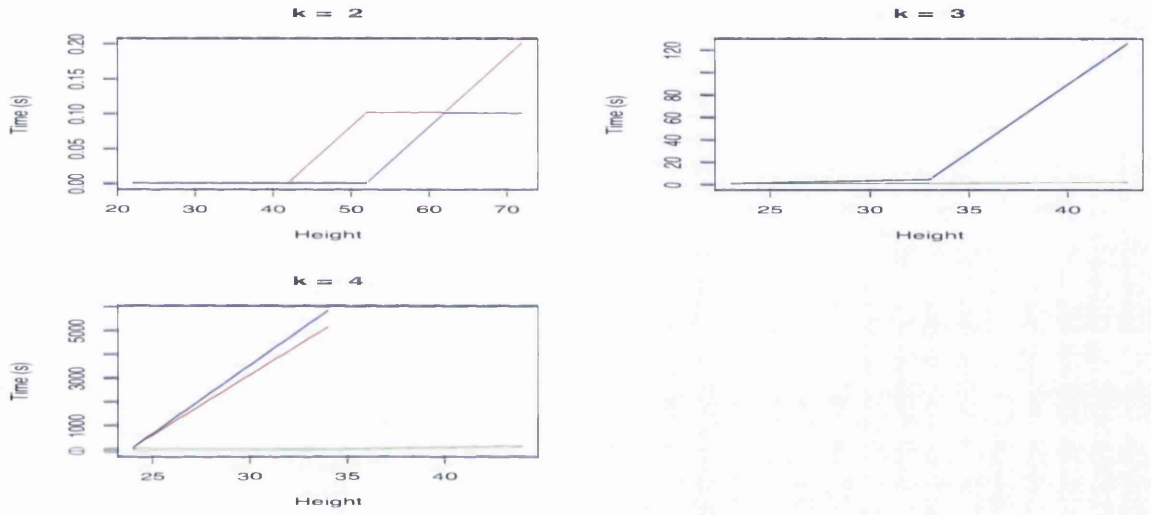


Figure 7.16: Solver times of Height (i.e., h) vs Time (i.e., t, in seconds) for `PicoSAT`. The lines for $i = 1$, 2 and 3 are coloured green, blue and red respectively; note that data is missing for $k = 5$ as there is only a single value for $i = 2$ and $i = 3$. See Figure 7.5 for the full details.

Chapter 8

Conclusion and outlook

This thesis has brought together two streams of research, one started by [51] in 1994, introducing $\mathcal{UC}$ for knowledge compilation, and one started by [141] in 1995, introducing $\mathcal{SLUR}$ for polytime SAT decision. Two natural generalisations, $\mathcal{UC}_k$ and $\mathcal{SLUR}_k$ have been provided, and the (actually surprising) identity $\mathcal{SLUR}_k = \mathcal{UC}_k$ provides both sides of the equation with additional tools. Various basic lemmas and theorems have been shown, providing a framework for elegant and powerful proofs. Regarding computational problems, the most basic questions have been answered.

Deriving from $\mathcal{UC}_k$, three hierarchies $\mathcal{PC}_k$, $\mathcal{UC}_k$ and $\mathcal{WC}_k$ of target classes for knowledge compilation and "good" SAT representations have been introduced and discussed. It has been shown that each level of $\mathcal{UC}_{k+1}$ contains clause-sets without equivalent short clause-sets in $\mathcal{WC}_k$. When using new variables, conditions were shown under which the Tseitin translation produces translations in $\mathcal{UC}$, while in general there are sequences of boolean circuits for which the Tseitin translation has unbounded hardness. Finally, in Chapter 7, experimental results were provided demonstrating both that modern SAT solvers can "handle" higher hardness (in certain cases) while polynomially larger representations in $\mathcal{UC}_1$ perform demonstratably worse. To conclude, future directions, conjectures and open questions are considered.

8.1 Strictness of hierarchies

A fundamental question is the strictness of the hierarchies $\mathcal{PC}_k$, $\mathcal{UC}_k$ and $\mathcal{WC}_k$ in each of the dimensions (with and without new variables; under the relative vs absolute condition). In Theorem 5.3.13 in Chapter 5 it was shown w.r.t. logical equivalence (i.e., without new variables) that the $\mathcal{UC}_k$ and $\mathcal{WC}_k$ hierarchies are strict (as representation hierarchies). It follows that for $\mathcal{PC}_k$ at least every second level yields an advance regarding logical equivalence (and polysize), i.e., that $\mathcal{PC}_{k+2}$ can be exponentially more succinct than $\mathcal{PC}_k$. These strictness results offer evidence that the $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ hierarchies are useful, for example using failed literal reduction can allow one to use exponentially smaller SAT or knowledge compilation translations. Open are the questions of strictness for the hierarchies allowing new variables. To summarise, the main conjectures are:

1. Conjecture 5.3.15 strengthens Theorem 5.3.13 by taking the PC-hierarchy into account (i.e., conjecturing that $\mathcal{PC}_{k+1}$ can be exponentially more succinct than $\mathcal{PC}_k$).
2. Conjecture 6.1.3 (roughly) says that all of $\mathcal{PC}_k, \mathcal{UC}_k$ and $\mathcal{WC}_k$ are strict (similar to Theorem 5.3.13), when allowing new variables under the *absolute* condition.

3. Conjecture 6.1.7 says that the $\mathcal{WC}_k$ hierarchy collapses to $\mathcal{WC}_1$ (and thus to $\mathcal{PC}_1$), when allowing new variables under the *relative* condition.

Considered together, under the *relative* condition only the levels $\mathcal{PC}_0 \subset \mathcal{UC}_0 \subset \mathcal{PC}_1$ are strict regarding polysize representations, where the two classes $\mathcal{PC}_0 \subset \mathcal{UC}_0$ do not gain anything from the new variables, while everything of $\mathcal{PC}_k, \mathcal{UC}_k$ and $\mathcal{WC}_k$ for $k \geq 1$ can be reduced (in polytime, with exponent depending on k) to $\mathcal{PC}_1 = \mathcal{PC}$. And $\mathcal{PC}$ under the relative condition is the same as the well-known condition of "arc consistency" for SAT translation. A major result of this thesis, that $\mathcal{PC}_k$, $\mathcal{UC}_k$ and $\mathcal{WC}_k$ for the absolute condition and without new variables do not collapse, shows that a rich structure was hidden under the carpet of the relative condition aka arc consistency. A basic difference between the relative and absolute condition is that under the relative condition the new variables can be used to perform certain "computations", since there are no conditions on the new variable other than not to distort the satisfying assignments, and it should be possible to use this to show the collapse to arc consistency, by encoding the stronger condition into the clause-sets in such a way that UCP can perform the "computations". The various strictness results and conjectures are summarised in Figure 8.1.

	Without new variables	**With new variables**	
		relative	**absolute**
$\mathcal{UC}_k$	Strict (Theorem 5.3.13)	Collapses? (Conjectures 6.1.6 and 6.1.7)	Strict? (Conjecture 6.1.3)
$\mathcal{PC}_k$	Strict? (Conjecture 5.3.15)		
$\mathcal{WC}_k$	Strict (Theorem 5.3.13)		

Figure 8.1: Summary of strictness results and conjectures for the $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ hierarchies. *Strict* here means that there exists a sequence of clause-sets $(F_n)_{n\in\mathbb{N}_0}$ all in $\mathcal{UC}_{k+1}$ resp. $\mathcal{PC}_{k+1}$ resp. $\mathcal{WC}_{k+1}$ such that all representations (with resp. without new variables) $(F'_n)_{n\in\mathbb{N}_0}$ of $(F_n)_{n\in\mathbb{N}_0}$ all in $\mathcal{UC}_k$ resp. $\mathcal{PC}_k$ resp. $\mathcal{WC}_k$ are of super-polynomial size, while otherwise the relevant hierarchy *collapses*.

8.1.1 Separating the different hierarchies

For stating the three main conjectures *relating* the three hierarchies, the following notions are used:

- A sequence $(F'_n)_{n\in\mathbb{N}}$ is called a CNF-representation of $(F_n)_{n\in\mathbb{N}}$ if for all $n \in \mathbb{N}$ the clause-set F'_n is a CNF-representation of F_n.
- A **polysize sequence in $\mathcal{C} \subseteq \mathcal{CLS}$** is a sequence $(F_n)_{n\in\mathbb{N}}$ with $F_n \in \mathcal{C}$ for all $n \in \mathbb{N}$, such that $(\ell(F_n))_{n\in\mathbb{N}}$ is polynomially bounded (i.e., there is a polynomial $p(x)$ with $\ell(F_n) \leq p(n)$ for all $n \in \mathbb{N}$).

It is conjectured that $\mathcal{WC}_2$, even without new variables, offers possibilities for good representations not offered by any $\mathcal{UC}_k$:

Conjecture 8.1.1 *There exists a polysize $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{WC}_2$, such that for no $k \in \mathbb{N}_0$ there exists a polysize CNF-representation $(F'_n)_{n\in\mathbb{N}}$ of $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{UC}_k$.*

A proof of Conjecture 8.1.1 needed, besides the new handling of the new variables, to develop lower-bounds methods specifically for hardness, since the method via trigger hypergraphs yields already lower bounds for w-hardness.

It is conjectured that new variables can not simulate higher hardness, strengthening Theorem 5.3.13, Conjecture 5.3.15 and Conjecture 6.1.3:

Conjecture 8.1.2 *For every $k \in \mathbb{N}_0$ there exists a polysize $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{PC}_{k+1}$, such that there is no polysize CNF-representation $(F'_n)_{n\in\mathbb{N}}$ of $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{WC}_k$.*

Finally it is conjectured that there is a sequence of boolean functions which has polysize arc-consistent representations, but no polysize representations of bounded hardness, even for the w-hardness:

Conjecture 8.1.3 *There exists a polysize $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{CLS}$, such that there is a polysize CNF-representation $(F'_n)_{n\in\mathbb{N}}$ of $(F_n)_{n\in\mathbb{N}}$ with $\mathrm{hd}^{\mathrm{var}(F_n)}(F'_n) \le 1$ for all $n \in \mathbb{N}$, while for no $k \in \mathbb{N}_0$ there is a polysize CNF-representation $(F''_n)_{n\in\mathbb{N}}$ of $(F_n)_{n\in\mathbb{N}}$ in $\mathcal{WC}_k$.*

Regarding the notion of a "polysize sequence" $(F_n)_{n\in\mathbb{N}}$, this is a very liberal notion, allowing to express arbitrary boolean functions, since the number of variables could be logarithmic in the index, and thus F_n could contain exponentially many clauses in the number of variables. The sequence of Theorem 5.3.13 also fulfils $n(F_n) = \Omega(n)$, and making this provision one could speak of "simple" boolean functions, however this would complicate the formulations of the conjectures, and so was avoided.

The considerations on hierarchies are concluded by considering the three hierarchies $\mathcal{SLUR}(k)$ introduced in [159], $\mathcal{SLUR}*(k)$ introduced in [36], and CANON(k) introduced in [10], which were compared to the UC-hierarchy in Chapter 4. From the point of view of polysize representations without new variables, the hierarchy CANON(k) collapses to CANON(0) $= \mathcal{UC}_0$:

Lemma 8.1.4 *For $F \in \mathcal{CLS}$ let $k(F)$ be the minimal $k \in \mathbb{N}_0$ such that $F \in$ CANON(k). Then the function $\mathrm{prc}_0 : \mathcal{CLS} \to$ CANON(0) $= \mathcal{UC}_0$ can be computed in time $O(c(F)^{3\cdot 2^k} \cdot \ell(F))$, when the input is F together with $k := k(F)$.*

Proof: Let $K := 2^k$. So for every $C \in \mathrm{prc}_0(F)$ there exists $F' \subseteq F$ with $F' \models C$ and $c(F') \le K$, since a resolution tree of height k has at most K leaves. Now we compute $\mathrm{prc}_0(F)$ as follows:

1. Set $P := \emptyset$.
2. Run through all $F' \subseteq F$ with $c(F') \le K$; their number is $O(c(F)^K)$.
3. For each F' determine whether $F' \models \mathrm{puc}(F')$ holds, in which case clause $\mathrm{puc}(F')$ is added to P; note that the test can be performed in time $O(2^K \cdot K)$.
4. The final P obtained has $O(c(F)^K)$ many elements. After performing subsumption elimination (in cubic time) we obtain $\mathrm{prc}_0(F)$ (by Lemma 5.1.7). □

It seems an interesting question whether the two other hierarchies $\mathcal{SLUR}(k)$, $\mathcal{SLUR}*(k)$ collapse or not, and whether they can be reduced to some $\mathcal{UC}_k$.

8.2 Hard boolean functions handled by oracles

As mentioned in Section 6.1, generalising [17], it should be possible to show that there are boolean functions such as the satisfiable pigeonhole formulas PHP_m^m which do not have polysize representations of bounded hardness even for the relative condition. One way to overcome this barrier is to generalise the theory started here via the use of oracles as in [104, 110] (recall Subsection 4.3.2), and then employing oracles which can handle pigeonhole formulas. The basic definitions are as follows.

Definition 8.2.1 *A **valid oracle** for generalised unit-clause propagation is some $\mathcal{U} \subseteq \mathcal{USAT}$ with $\{\bot\} \in \mathcal{U}$ which is stable under application of partial assignments. The oracle is **strong** if $\mathcal{U}_0 \subseteq \mathcal{U}$, where $\boldsymbol{\mathcal{U}_0} := \{F \in \mathcal{CLS} : \bot \in F\}$.*

Consider $k \in \mathbb{N}_0$. In [104] the reduction $\mathrm{r}_k^{\mathcal{U}} : \mathcal{CLS} \to \mathcal{CLS}$ has been defined. An equivalent definition (generalising Definition 3.1.1) is as follows for $F \in \mathcal{CLS}$:

$$\mathrm{r}_0^{\mathcal{U}}(F) := \begin{cases} \{\bot\} & \text{if } F \in \mathcal{U} \\ F & \text{otherwise} \end{cases}$$

$$\mathrm{r}_{k+1}^{\mathcal{U}}(F) := \begin{cases} \mathrm{r}_{k+1}^{\mathcal{U}}(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \mathrm{lit}(F) : \mathrm{r}_k^{\mathcal{U}}(\langle x \to 0\rangle * F) = \{\bot\} \\ F & \text{otherwise} \end{cases}.$$

Note $\mathrm{r}_k = \mathrm{r}_k^{\mathcal{U}_0}$. Generalising Definitions 3.3.1, 3.4.1:

Definition 8.2.2 *Consider a valid oracle $\mathcal{U}$. The **hardness** $\boldsymbol{\mathrm{hd}_{\mathcal{U}}(F)} \in \mathbb{N}_0$ ("hardness with oracle $\mathcal{U}$") of an unsatisfiable $F \in \mathcal{CLS}$ is the minimal $k \in \mathbb{N}_0$ such that $\mathrm{r}_k^{\mathcal{U}}(F) = \{\bot\}$. And for general $F \in \mathcal{CLS}$ we define $\mathrm{hd}_{\mathcal{U}}(\top) := 0$, while for $F \neq \top$ let*

$$\mathrm{hd}_{\mathcal{U}}(F) := \max\{\mathrm{hd}_{\mathcal{U}}(\varphi * F) : \varphi \in \mathcal{PASS} \wedge \varphi * F \in \mathcal{USAT}\} \in \mathbb{N}_0.$$

We have $\mathrm{hd} = \mathrm{hd}_{\mathcal{U}_0}$, and if $\mathcal{U}$ is strong then for all F holds $\mathrm{hd}_{\mathcal{U}}(F) \leq \mathrm{hd}(F)$. An interesting oracle $\mathcal{U}$ (with polytime membership decision) is given by the class of unsatisfiable clause-sets defined in [49] via semidefinite programming, for which we get $\mathrm{hd}_{\mathcal{U}}(\mathrm{PHP}_m^m) = 0$.

An important aspect of the theory to be developed must be the usefulness of the representation (with oracles) in context, that is, as a "constraint" in a bigger problem: a boolean function f represented by a clause-set F is typically contained in $F' \supset F$, where F' is the SAT problem to be solved (containing also other constraints). One approach is to require from the oracle also stability under addition of clauses, as we have it already for the resolution-based reductions like r_k, so that the (relativised) reductions $\mathrm{r}_k^{\mathcal{U}}$ can always run on the whole clause-set (an instantiation of F'). However for example for the semidefinite programming oracle mentioned above, this would be prohibitively expensive. And for some oracles, like detection of minimally unsatisfiable clause-sets of a given deficiency, the problems would turn from polytime to NP-hard in this way ([60, 32]). Furthermore, that we have some representation does not mean that in other parts of the problems also that oracle will be of help. So in many cases it is better to restrict the application of the oracle $\mathcal{U}$ to the subset $F \subset F'$, where to achieve the desired hardness the oracle is required.

8.3 Exploring (t-,w-)hardness

In [104] the notions of hardness and w-hardness for *unsatisfiable* clause-sets were extensively explored, along with their relations to tree- and full-resolution complexity. However, there remain

both open questions regarding the new generalisation to satisfiable clause-sets (hd from Definition 3.4.1) and also new questions raised for unsatisfiable clause-sets related to more recent results.

8.3.1 Exploring hardness

Regarding the underlying hardness notions discussed in Chapter 3, two directions of future research are further *charactisations* of hardness and the placement of the $\mathcal{UC}_k$ hierarchy in the landscape of all CNF clause-sets. In particular, the following are two possibilities for future research:

1. In Lemma 3.3.2 we saw that for unsatisfiable clause-sets $F \in \mathcal{CLS}$ that $\mathrm{hd}(F)$ is the optimal value for the Pudlák-Impagliazzo game from [133]. Can this characterisation be extended to give a game-theoretic characterisation of hardness (from Definition 3.4.1) for *all* clause-sets?

2. In [62] a probabilistic argument is used to estimate the proportion of k-CNF clause-sets that are in $\mathcal{SLUR}$ and other simple poly-time SAT classes. A next step in this direction would be to try to generalise these results to $\mathcal{UC}_k$ $(= \mathcal{SLUR}_k)$ to estimate how the proportion of $\mathcal{CLS}$ that $\mathcal{UC}_k$ makes up grows in k (tending towards 1 as k tends to ∞ - recall $\bigcup_{k\in\mathbb{N}_0} \mathcal{UC}_k = \mathcal{CLS}$).

8.3.2 Exploring w-hardness

It is to be expected that w-hardness can behave very differently from hardness. For example, as expressed by Conjecture 8.1.1, already its second level should contain short clause-sets not representable in any $\mathcal{UC}_k$. However yet we do not have tools at hand to handle w-hardness (we do not even have yet a conjectured example for such a separation). A first task is to investigate which of the results on hardness from this thesis and from [78] can be adapted to w-hardness.

Can the classes $\mathcal{WC}_k$ go beyond monotone circuits, which were shown in [17] to be strongly related to the expressive power of arc-consistent CNF representations (see Section 8.2 for some further remarks)? Conjecture 6.1.7 would show the contrary, namely that in the (unrestricted) presence of new variables also w-hardness boils down, modulo polytime computations, to $\mathcal{PC}_1$ (under the relative condition!). If this is true, then the believable greater power of $\mathcal{WC}_k$ over $\mathcal{UC}_k$ would all take place inside arc-consistency; and by Conjecture 8.1.3 it would take place strictly inside arc-consistency.

8.4 A theory of "good" SAT representations

The main future application, which brings the $\mathcal{UC}$-perspective and the $\mathcal{SLUR}$-perspective together, is in the area of "good SAT representations". This thesis considers the approach of representing a boolean function f via a clause-set $F \in \mathcal{UC}_k$ as the first beginning of what is envisaged as a theory of good SAT representations. The main open questions here and future directions for research are now enumerated.

1. **Full constraint translation:** Throughout this thesis the focus has been on representing boolean functions. However, in general, when translating to SAT one typically follows a more "full" process:

$$\textbf{Problem} \xrightarrow[\text{Modelling}]{} \textbf{Constraints} \xrightarrow[\text{Encoding}]{} \underbrace{\textbf{Boolean functions} \xrightarrow[\text{Translation}]{} \textbf{SAT}.}_{\textbf{focus of this thesis}}$$

As mentioned in Section 2.8 of Chapter 2, the current main methodology for deriving "good" SAT translations is to attack the problem from the *constraint* perspective - trying to derive SAT translations which maintain certain forms of consistency in the original constraint network via mechanisms such as unit-clause propagation in the SAT solver.

In light of the $\mathcal{UC}_k, \mathcal{PC}_k$ and $\mathcal{WC}_k$ hierarchies, another possibility is to consider the translation from the SAT perspective, relying on these "target-classes" for guarantees on solver performance, rather than on consistency notions at the higher level. Interesting future directions are:

(a) **Encoding vs translation:** The focus in this thesis on translation of boolean functions to CNF is both conceptually simpler than the constraint to CNF translation, useful on its own in a variety of cases, and useful as the *translation* component of the above SAT translation process. In future, it would be interesting to look both purely at the *encoding* part of SAT translation, i.e., assuming an ideal translation and asking when different encodings (i.e., direct, log, order etc) can maintain different types of constraint consistency, and also for fixed encodings, what are the best translations.

(b) **Unions:** Proven in [65], it is a well-known fact in the constraint community that maintaining arc-consistency on acyclic binary constraint networks enforces satisfiability (i.e., if the network is inconsistent the maintenance procedure will produce an empty domain for some constraint). From the SAT perspective, instead if one has clause-sets $F_0, \ldots, F_m \in \mathcal{PC}_k$ (for $k \in \mathbb{N}_0$) with an acyclic variable interaction hypergraph (nodes are variables, hyperedges are $\mathrm{var}(F_i)$ for all $i \in \{1, \ldots, m\}$) then in the same way one should have $\bigcup_{m' \in \{1,\ldots,m\}} F_m \in \mathcal{PC}_k$. This would then provide another method for upper-bounding the p-hardness of SAT instances and constructing (in a tree-like manner) p-*soft* representations. Of particular interest would be to consider additional constraints one could place on the variable-interaction hypergraph to allow only a constant or bounded increase in (p-)hardness, for example bounds on the tree-width and restrictions on the type of constraint.

2. **A database of constraint examples:** In Chapter 6 it is shown that a reduced version of Tseitin translation applied to DNF clause-sets yield CNF representations in $\mathcal{UC}$. The next step would be to reconsider all literature on "good" SAT representations (e.g., cardinality constraint translations from [147, 7], pseudo-boolean constraints in [55, 8], translations of arbitrary smooth DNNFs from [98] etc), particularly those representations which maintain arc-consistency via UCP and determine

   (a) whether the translations map into $\mathcal{UC}$, $\mathcal{PC}$ or $\mathcal{WC}$;

   (b) whether they map into these classes via the relative or absolute condition;

   (c) if not (for either condition), under what restrictions do they map to these classes;

   (d) does climbing the $\mathcal{UC}_k$, $\mathcal{PC}_k$ and the $\mathcal{WC}_k$ hierarchy allow generalisations of these constraints to be represented (particularly for example in the pseudo-boolean and DNNF cases)?

3. **Heuristical guidelines for "good" representations:** In Chapter 7 we saw evidence that for certain types of clause-set there is a size vs hardness trade-off for SAT solver performance. That is, it is not the case that translating a SAT problem to CNF by translating each constraint to $\mathcal{UC}_1$ will always be better than translating to $\mathcal{UC}_2$ or $\mathcal{UC}_3$ etc. Finding the "best SAT representation" is a very multidimensional process - one must (at least) optimise the hardness and the size, as well as higher level interactions between

constraints. A focus in future practical and empirical SAT research on applying a large variety of techniques (including hardness measures) and reporting on the successes, as well as the failures, would help to form a corpus of good heuristics for SAT translations.

8.5 Compilation procedures

For a given boolean function f and $k \in \mathbb{N}_0$, how do we find algorithmically a "small" equivalent $F \in \mathcal{UC}_k$? In Chapter 4 the notion of a "k-base for f" is introduced, which is an $F \in \mathcal{UC}_k$ equivalent to f, with $F \subseteq \mathrm{prc}_0(f)$ and where no clause can be removed without increasing the hardness or destroying equivalence. It is shown that if f is given as a 2-CNF, then a smallest k-base is computable in polynomial time, but even for f with given $\mathrm{prc}_0(f)$, where $\mathrm{prc}_0(f)$ is a Horn clause-set, deciding whether a k-base of a described size for a fixed $k \geq 1$ exists is NP-complete.

There are interesting applications where $\mathrm{prc}_0(f)$ is given (or can be computed), and where then some small equivalent $F \in \mathcal{UC}_k$ is sought. The most basic approach filters out unneeded prime implicates; see [75, 74] for some initial applications to cryptanalysis. A simple filtering heuristic, used in [75, 74], is to favour (keeping) short-clauses. In a first phase, starting with the necessary elements of $\mathrm{prc}_0(f)$, further elements are added (when needed) in ascending order of size for building up the initial $F \in \mathcal{UC}_k$ (which in general is not a base). In the second phase, clauses from F are removed in descending order of size when reducing to a k-base. The intuition behind this heuristic is that small clauses cover more total assignments (so fewer are needed), and they are also more likely to trigger r_k, making them more useful in producing small, powerful representations. Essentially the same heuristic is considered in [26] (called "length-increasing iterative empowerment") when generating representations in $\mathcal{PC}$.

For the case that f is given by a CNF F_0, in [51] one finds refinements of the resolution procedure applied to F_0, which would normally compute $\mathrm{prc}_0(f)$, i.e., the 0-base in $\mathcal{UC}_0$, and where by some form of "compression" now an equivalent $F \in \mathcal{UC}_1$ is computed. This approach needed to be generalised to arbitrary $\mathcal{UC}_k$. Another approach would also be to consider computations of prime implicate sets and k-bases implicitly, using implicit representations via BDD-like structures such as in [40].

8.6 Translating the Schaefer classes

To conclude some remarks are made on the four main classes from Schaefer's dichotomy result (see Section 12.2 in [45] for an introduction, and see [43] for an in-depth overview on recent developments). The point of view here is that we consider a boolean function f which is either Horn, dual Horn, bijunctive or affine, and we ask for a good representation $F \in \mathcal{CLS}$ of f:

- If f is Horn or dual Horn, then there is a (dual) Horn clause-set F equivalent to f, and by Part 4 of Lemma 4.3.2 we have $\mathrm{hd}(F) \leq 1$. So obtaining a representation $F \in \mathcal{UC}$ is trivial; however optimising the size of F is NP-complete (see Theorem 4.5.5).

- If f is bijunctive, then there is a 2-CNF F equivalent to f, and by Part 3 of Lemma 4.3.2 we have $\mathrm{hd}(F) \leq 2$. Moreover, by Theorem 4.5.4 we can reduce the hardness to 0 or 1 (as we wish) in polynomial time, and that by optimal (shortest) such F.

- If f is affine, that is, f is the conjunction of m linear equations $x_1 \oplus \cdots \oplus x_p = 0$ over $\{0,1\}$ viewed as a 2-element field, with addition $\oplus$ as exclusive-or, then the situation regarding the existence of a representation of bounded hardness is not fully understood yet:

1. If $m = 1$, then there is precisely one CNF-representation of f without new variables, containing 2^{p-1} clauses and being (trivially) of hardness 0. So without new variables we have a polysize representation of bounded hardness iff p is bounded.
2. While when allowing new variables, then for $m = 1$ there is a representation $F \in \mathcal{UC}$, as is shown in Lemma 6.4.1 of Chapter 6.
3. For arbitrary m there is definitely no small representation without new variables when the clause-length p is unbounded. When bounding p, or when allowing new variables, then the existence of a polysize $F \in \mathcal{UC}_k$ for some fixed k seems to be an interesting open problem; for some partial results see [118]. Perhaps no polysize representations $F \in \mathcal{UC}$ exist, even for the "relative condition", where propagation-conditions are posed only for the variables in the XOR-clauses; see again [17] for general tools for such lower bounds, and see Chapter 5 and Chapter 6 for more discussions.

Chapter 9

Bibliography

[1] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. In *STOC '02 Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 448–456. ACM, 2002.

[2] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. Measuring the hardness of SAT instances. In Dieter Fox and Carla Gomes, editors, *Proceedings of the 23th AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 222–228, 2008.

[3] 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, March 1979.

[4] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. In Kullmann [115], pages 114–127. ISBN 978-3-642-02776-5.

[5] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI'09 Proceedings of the 21st International Joint Conference on Artificial intelligence*, pages 399–404. AAAI, 2009.

[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, 2007.

[7] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming – CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122, 2003.

[8] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into CNF. In Kullmann [115], pages 181–194. ISBN 978-3-642-02776-5.

[9] Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, and Carsten Sinz. Proceedings of SAT challenge 2012; solver and benchmark descriptions. Technical Report B-2012-2, University of Helsinki, Department of Computer Science, 2012. Available at `https://helda.helsinki.fi/handle/10138/34218`. Competition website at `http://baldur.iti.kit.edu/SAT-Challenge-2012/`.

[10] Tomáš Balyo, Štefan Gurský, Petr Kučera, and Václav Vlček. On hierarchies over the SLUR class. In *Twelfth International Symposium on Artificial Intelligence and Mathematics (ISAIM 2012)*, January 2012. Available at `http://www.cs.uic.edu/bin/view/Isaim2012/AcceptedPapers`.

[11] Gregory Bard, Nicholas Courtois, and David Wagner. Algebraic and slide attacks on KeeLoq. In Kaisa Nyberg, editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2008.

[12] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow — resolution made simple. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 517–526, May 1999.

[13] Daniel Le Berre, Olivier Roussel, and Laurent Simon. SAT 2009 competitive events booklet: preliminary version. Booklet available at `http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf`, 2009. Competition website at `http://www.satcompetition.org/2009/`. Results available at `http://www.cril.univ-artois.fr/SAT09/`.

[14] Daniel Le Berre and Laurent Simon. The essentials of the SAT 2003 competition. In Giunchiglia and Tacchella [70], pages 452–467. Competition website at `http://www.satcompetition.org/2003/`.

[15] Daniel Le Berre and Laurent Simon. Fifty-five solvers in Vancouver: The SAT 2004 competition. In Hoos and Mitchell [90], pages 321–344. Competition website at `http://www.satcompetition.org/2004/`.

[16] Christian Bessiere. Constraint propagation. In Rossi et al. [139], chapter 3, pages 29–83. ISBN 0-444-52726-5.

[17] Christian Bessiere, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 412–418, 2009.

[18] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[19] Armin Biere. P{re,i}coSAT@SC'09. `http://fmv.jku.at/precosat/preicosat-sc09.pdf`, 2009.

[20] Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, August 2010. `http://fmv.jku.at/papers/Biere-FMV-TR-10-1.pdf`.

[21] Armin Biere. Lingeling and friends entering the SAT Challenge 2012. In Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, and Carsten Sinz, editors, *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of *Department of Computer Science Series of Publications B*, pages 33–34. University of Helsinki, 2012. `https://helda.helsinki.fi/bitstream/handle/10138/34218/sc2012_proceedings.pdf`.

[22] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*. Springer, 1999. `http://dx.doi.org/10.1007/3-540-49059-0_14`.

[23] Armin Biere, Marijn J.H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[24] Archie Blake. *Canonical expressions in Boolean algebra*. PhD thesis, Chicago, 1937. See [123].

[25] Lucas Bordeaux, Mikoláš Janota, Joao Marques-Silva, and Pierre Marquis. On unit-refutation complete formulae with existentially quantified variables. In *Knowledge Representation 2012 (KR 2012)*. Association for the Advancement of Artificial Intelligence (AAAI Press), June 2012.

[26] 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, 2012.

[27] Endre Boros and Ondřej Čepek. On the complexity of Horn minimization. Technical Report RRR 1-94, Rutcor Research Report, January 1994.

[28] Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Logical analysis of data: Classification with justification. Working paper, February 2009.

[29] Endre Boros, Peter L. Hammer, Michel Minoux, and David J. Rader Jr. Optimal cell flipping to minimize channel density in VLSI design and pseudo-boolean optimization. *Discrete Applied Mathematics*, 90:69–88, 1999.

[30] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[31] Uwe Bubeck and Hans Kleine Büning. The power of auxiliary variables for propositional and quantified boolean formulas. *Studies in Logic*, 3(3):1–23, 2010.

[32] Hans Kleine Büning and Xishun Zhao. The complexity of read-once resolution. *Annals of Mathematics and Artificial Intelligence*, 36(4):419–435, December 2002.

[33] Michael Buro and Hans Kleine Büning. On resolution with short clauses. *Annals of Mathematics and Artificial Intelligence*, 18(2-4):243–260, 1996.

[34] Marco Cadoli and Francesco M. Donini. A survey of knowledge compilation. *AI Communications*, 10(3,4):137–150, December 1997.

[35] Ondřej Čepek and Petr Kučera. Known and new classes of generalized Horn formulae with polynomial recognition and SAT testing. *Discrete Applied Mathematics*, 149:14–52, 2005.

[36] 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 *LNCS Lecture Notes in Computer Science*, pages 177–189. Springer, 2012.

[37] Tom Chang. Horn formula minimization. Master's thesis, Rochester Institute of Technology, May 2004.

[38] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[39] Stephen A. Cook. An exponential example for analytic tableaux. Manuscript (see [157], page 432), 1973.

[40] Olivier Coudert and Jean C.C. Madre. A new method to compute prime and essential prime implicants of boolean functions. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*. MIT Press, 1992.

[41] Nicolas T. Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the Hitag2 stream cipher. In Pierangela Samarati, Moti Yung, Faboi Martinelli, and Claudio A. Ardagna, editors, *Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 2009.

[42] Yves Crama and Peter L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2011. ISBN 978-0-521-84751-3.

[43] Nadia Creignou, Phokion Kolaitis, and Heribert Vollmer, editors. *Complexity of Constraints: An Overview of Current Research Themes*, volume 5250 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2008. ISBN-10 3-540-92799-9.

[44] Thomas W. Cusick and Pantelimon Stănică. *Cryptographic Boolean Functions and Applications*. Academic Press, 2009. ISBN 978-0-1237-4890-4.

[45] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In Biere et al. [23], chapter 12, pages 403–424.

[46] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[47] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communication of the ACM*, 5:394–397, 1962.

[48] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[49] Etienne de Klerk, Hans van Maaren, and Joost P. Warners. Relaxations of the satisfiability problem using semidefinite programming. *Journal of Automated Reasoning*, 24:37–65, 2000.

[50] Rina Dechter and Peter van Beek. Local and global relational consistency. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming - CP '95*, volume 976 of *Lecture Notes In Computer Science*, pages 240–257. Springer, 1995.

[51] Alvaro del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994.

[52] Alvaro del Val. On some tractable classes in deduction and abduction. *Artificial Intelligence*, 116(1-2):297–313, January 2000.

[53] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[54] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Giunchiglia and Tacchella [70], pages 502–518. ISBN 3-540-20851-8.

[55] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, March 2006.

[56] Thomas Eiter, Pekka Kilpeläinen, and Heikki Mannila. Recognizing renamable generalized propositional Horn formulas is NP-complete. *Discrete Applied Mathematics*, 59:23–31, 1995.

[57] Eleazar Eskin, Eran Halperin, and Richard M. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *Journal of Bioinformatics and Computational Biology*, 1(1):1–20, 2003.

[58] R. Fagin. Functional dependencies in a relational database and propositional logic. *IBM Journal of research and development*, 21(6):534–544, November 1977.

[59] Hélène Fargier and Piere Marquis. Extending the knowledge compilation map: Krom, Horn, Affine and beyond. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikos Avouris, editors, *ECAI 2008*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 50–54. IOS Press, 2008.

[60] Herbert Fleischner, Oliver Kullmann, and Stefan Szeider. Polynomial–time recognition of minimal unsatisfiable formulas with fixed clause–variable difference. *Theoretical Computer Science*, 289(1):503–516, November 2002.

[61] John Franco. Relative size of certain polynomial time solvable subclasses of satisfiability. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem: Theory and Applications (DIMACS Workshop March 11-13, 1996)*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 211–223. American Mathematical Society, 1997. ISBN 0-8218-0479-0.

[62] John Franco and Allen Van Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Applied Mathematics*, 125:177–214, 2003.

[63] John Franco and John Martin. A history of satisfiability. In Biere et al. [23], chapter 1, pages 3–74.

[64] John Franco and John Schlipf. 1997 final report: Describing new results under the research project entitled Complexity of algorithms for problems in propositional logic. covering the period january 1, 1994 - march 31, 1997. Technical report, University of Cincinnati and Office of Naval Research, April 1997. Available at `http://www.dtic.mil/docs/citations/ADA325949`.

[65] Eugene C. Freuder. A sufficient condition for backtrack-tree search. *Journal of the ACM*, 29(1):24–32, January 1982.

[66] Giorgio Gallo and Maria Grazia Scutellà. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–227, November 1988.

[67] Ian P. Gent. Arc consistency in SAT. In Frank van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 121–125. IOS Press, 2002.

[68] Ian P. Gent, Chris Jefferson, and Ian Miguel. MINION: A fast, scalable, constraint solver. In *ECAI 2006*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 98–102, Amsterdam, The Netherlands, 2006. IOS Press.

[69] Ian P. Gent, Chris Jefferson, and Ian Miguel. Watched literals for constraint propagation in Minion. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 182–197. Springer, 2006.

[70] Enrico Giunchiglia and Armando Tacchella, editors. *Theory and Applications of Satisfiability Testing 2003*, volume 2919 of *Lecture Notes in Computer Science*, Berlin, 2004. Springer. ISBN 3-540-20851-8.

[71] Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM Journal on Computing*, 22(4):661–683, August 1993.

[72] Ana Graça, Inês Lynce, João Marques-Silva, and Arlindo L. Oliveira. Haplotype inference by pure parsimony: A survey. *Journal of Computational Biology*, 17(8):969–992, 2010.

[73] Jan Friso Groote and Joost P. Warners. The propositional formula checker HeerHugo. *Journal of Automated Reasoning*, 24:101–125, 2000.

[74] Matthew Gwynne and Oliver Kullmann. Towards a better understanding of hardness. In *The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011): Doctoral Program Proceedings*, pages 37–42, September 2011. Proceedings available at `http://www.dmi.unipg.it/cp2011/downloads/dp2011/DP_at_CP2011.pdf`.

[75] Matthew Gwynne and Oliver Kullmann. Towards a better understanding of SAT translations. In Ulrich Berger and Denis Therien, editors, *Logic and Computational Complexity (LCC'11), as part of LICS 2011*, June 2011. 10 pages, available at `http://www.cs.swansea.ac.uk/lcc2011/`.

[76] Matthew Gwynne and Oliver Kullmann. Generalising and unifying SLUR and unit-refutation completeness. In Peter van Emde Boas, Frans C. A. Groen, Giuseppe F. Italiano, Jerzy Nawrocki, and Harald Sack, editors, *SOFSEM 2013: Theory and Practice of Computer Science*, volume 7741 of *Lecture Notes in Computer Science (LNCS)*, pages 220–232. Springer, 2013.

[77] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. Technical Report arXiv:1204.6529v5 [cs.LO], arXiv, January 2013.

[78] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. *Journal of Automated Reasoning*, 2013. To appear.

[79] Matthew Gwynne and Oliver Kullmann. Towards a theory of good SAT representations. Technical Report arXiv:1302.4421v4 [cs.AI], arXiv, May 2013.

[80] Matthew Gwynne and Oliver Kullmann. Trading inference effort versus size in CNF knowledge compilation. In preparation, June 2013.

[81] Matthew Gwynne and Oliver Kullmann. On sat representations of xor constraints. To appear in LATA 2014, 2014.

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

[83] Peter L. Hammer and Tibérius O. Bonates. Logical analysis of data – an overview: From combinatorial optimization to medical applications. *Annals of Operations Research*, 148(1):203–225, November 2006.

[84] John Harrison. Stålmarck's algorithm as a HOL derived rule. In *Theorem proving in higher order logics: 9th International Conference, TPHOLs'96*, Lecture Notes in Computer Science 1125, pages 221–234, 1996.

[85] Edith Hemaspaandra and Henning Schnoor. Minimization for generalized boolean formulas. In Toby Walsh, editor, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 1, pages 566–571. AAAI Press, 2011.

[86] Lawrence J. Henschen and Lawrence Wos. Unit refutations and Horn sets. *Journal of the Association for Computing Machinery*, 21(4):590–605, October 1974.

[87] Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In Hoos and Mitchell [90], pages 345–359. ISBN 3-540-27829-X.

[88] Marijn J. H. Heule and Hans van Maaren. Look-ahead based SAT solvers. In Biere et al. [23], chapter 5, pages 155–184.

[89] Marijn J.H. Heule. March: Towards a lookahead sat solver for general purposes. Master's thesis, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, March 2004.

[90] Holger H. Hoos and David G. Mitchell, editors. *Theory and Applications of Satisfiability Testing 2004*, volume 3542 of *Lecture Notes in Computer Science*, Berlin, 2005. Springer. ISBN 3-540-27829-X.

[91] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, March 1951.

[92] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.

[93] Paul Jackson and Daniel Sheridan. Clause form conversions for boolean circuits. In Hoos and Mitchell [90], pages 183–198. ISBN 3-540-27829-X.

[94] Matti Järvisalo and Tommi Junttila. Limitations of restricted branching in clause learning. *Constraints*, 14(3):325–356, 2009.

[95] Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In Michela Milano, editor, *Principles and Practice of Constraint Programming - CP 2012*, volume 7514 of *Lecture Notes In Computer Science*, pages 316–331. Springer, 2012.

[96] Matti Järvisalo and Ilkka Niemelä. The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study. *Journal of Algorithms*, 63(1):90–113, July 2008.

[97] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. ISBN 978-3-642-24507-7.

[98] Jean Christoph Jung, Pedro Barahoma, George Katsirelos, and Toby Walsh. Two encodings of DNNF theories, July 2008. Presented at ECAI'08 Workshop on Inference methods based on Graphical Structures of Knowledge. Proceedings at `http://www.irit.fr/LC/`.

[99] Hans Kleine Büning. On generalized Horn formulas and k-resolution. *Theoretical Computer Science*, 116:405–413, 1993.

[100] Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In Biere et al. [23], chapter 11, pages 339–401.

[101] Robert Kowalski. Predicate logic as programming language. In Jack L. Rosenfeld, editor, *Proceedings of IFIP Congress 74*, Information Processing, pages 569–574. North Holland Publishing, August 1974.

[102] Daniel Kroening. Software verification. In Biere et al. [23], chapter 16, pages 505–532.

[103] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

[104] Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity (ECCC), October 1999.

[105] Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, July 1999.

[106] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, October 1999.

[107] Oliver Kullmann. An application of matroid theory to the SAT problem. In *Fifteenth Annual IEEE Conference on Computational Complexity (2000)*, pages 116–124. IEEE Computer Society, July 2000.

[108] Oliver Kullmann. Investigating the behaviour of a SAT solver on random formulas. Technical Report CSR 23-2002, Swansea University, Computer Science Report Series (available from `http://www-compsci.swan.ac.uk/reports/2002.html`), October 2002. 119 pages.

[109] Oliver Kullmann. The combinatorics of conflicts between clauses. In Giunchiglia and Tacchella [70], pages 426–440. ISBN 3-540-20851-8.

[110] Oliver Kullmann. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 40(3-4):303–352, March 2004.

[111] Oliver Kullmann. The SAT 2005 solver competition on random instances. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:61–102, 2006.

[112] Oliver Kullmann. Present and future of practical SAT solving. In Creignou et al. [43], pages 283–319. ISBN-10 3-540-92799-9.

[113] Oliver Kullmann. The `OKlibrary`: Introducing a "holistic" research platform for (generalised) SAT solving. *Studies in Logic*, 2(1):20–53, 2009.

[114] Oliver Kullmann. Fundaments of branching heuristics. In Biere et al. [23], chapter 7, pages 205–244.

[115] Oliver Kullmann, editor. *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*. Springer, 2009. ISBN 978-3-642-02776-5.

[116] Oliver Kullmann. Constraint satisfaction problems in clausal form II: Minimal unsatisfiability and conflict structure. *Fundamenta Informaticae*, 109(1):83–119, 2011.

[117] Oliver Kullmann and Xishun Zhao. On Davis-Putnam reductions for minimally unsatisfiable clause-sets. Technical Report arXiv:1202.2600v5 [cs.DM], arXiv, December 2012.

[118] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Classifying and propagating parity constraints. In Michela Milano, editor, *Principles and Practice of Constraint Programming – CP 2012*, volume 7514 of *Lecture Notes in Computer Science (LNCS)*, pages 357–372. Springer, 2012.

[119] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371. Morgan Kaufmann Publishers, 1997.

[120] Joao Marques-Silva. Computing minimally unsatisfiable subformulas: State of the art and future directions. *Journal of Multiple-Valued Logic and Soft Computing*, 19(1-3):163–183, 2012.

[121] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [23], chapter 4, pages 131–153.

[122] Fabio Massacci and Laura Marraro. Logical cryptoanalysis as a SAT problem. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000 Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 343–375. IOS Press, Amsterdam, 2000. ISBN 1 58603 061 2.

[123] J. C. C. McKinsey. Archie Blake: Canonical expressions in Boolean algebra. Review, The Journal of Symbolic Logic (3), 1938.

[124] Ryuhei Miyashiro and Tomomi Matsui. A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters*, 33:235–241, 2005.

[125] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[126] Maxwell Herman Alexander Newman. On theories with a combinatorial definition of equivalence. *Annals of mathematics*, 43(2):223–243, 1942.

[127] Jakob Nordström. Narrow proofs may be spacious: Separating space and width in resolution. *SIAM Journal*, 39(1):59–121, 2009.

[128] Justyna Petke and Peter Jeavons. The order encoding: from tractable CSP to tractable SAT. Technical Report RR-11-04, University of Oxford, 2011.

[129] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 654–668. Springer, 2009.

[130] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

[131] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[132] Daniele Pretolani. Hierarchies of polynomially solvable satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 17(3-4):339–357, 1996.

[133] Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT (preliminary version). In *SODA*, pages 128–136. ACM/SIAM, 2000.

[134] Charles C. Ragin. *The Comparative Method: Moving Beyond Qualitative and Quantitative Strategies.* University of California Press, 1987. ISBN 978-0-520-06618-2.

[135] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eigth Innovative Applications of Artificial Intelligence Conference (AAAI / IAAI)*, volume 1, pages 209–215, 1996.

[136] Benoît Rihoux and Charles C. Ragin, editors. *Configurational Comparative Methods: Qualitative Comparative Analysis (QCA) and Related Techniques*, volume 51 of *Applied Social Research Methods Series.* SAGE Publications, 2009. ISBN 978-1-4129-4235-5.

[137] Jussi Rintanen. Planning and SAT. In Biere et al. [23], chapter 15, pages 483–504.

[138] J.A. Robinson. A machine-oriented logic based resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[139] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming.* Foundations of Artificial Intelligence. Elsevier, 2006. ISBN 0-444-52726-5.

[140] Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News*, 33(3):32–49, 2002.

[141] John S. Schlipf, Fred S. Annexstein, John V. Franco, and R.P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54:133–137, 1995.

[142] Bart Selman and Henry Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 904–909. MIT Press, July 1991.

[143] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In *FMCAD'98*, volume 1522 of *Lecture Notes in Computer Science*, pages 82–99, 1998.

[144] Joao P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[145] Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The SAT2002 competition. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):307–342, January 2005. Competition website at `http://www.satcompetition.org/2002/`.

[146] Carsten Sinz. Knowledge compilation for product conguration. In Frank van Harmelen, editor, *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 23–26, 2002. Available at `http://www.carstensinz.de/papers/ECAI-Config-WS-2002.pdf`.

[147] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming – CP 2005*, volume 3709 of *Lecture Notes in Computer Science (LNCS)*, pages 827–831. Springer, 2005.

[148] Robert H. Sloan, Balázs Sörényi, and György Turán. On k-term DNF with the largest number of prime implicants. *SIAM Journal on Discrete Mathematics*, 21(4):987–998, 2007.

[149] Mate Soos. Cryptominisat 2.5.0. `http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_13.pdf`, 2010.

[150] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Kullmann [115], pages 244–257. `http://planete.inrialpes.fr/~soos/publications/Extending_SAT_2009.pdf`.

[151] Niklas Sörensson. Minisat 2.2 and minisat++ 1.1. `http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_25+26.pdf`, 2010.

[152] Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

[153] Gunnar Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In B.K. Daniels, editor, *Safety of Computer Control Systems (SAFECOMP'90)*, pages 31–36, 1990.

[154] Balázs Szörényi. Disjoint DNF tautologies with conflict bound two. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:1–14, 2007.

[155] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[156] G.S. Tseitin. On the complexity of derivation in propositional calculus. In *Seminars in Mathematics*, volume 8. V.A. Steklov Mathematical Institute, Leningrad, 1968. English translation: Studies in mathematics and mathematical logic, Part II (A.O. Slisenko, editor), 1970, pages 115-125.

[157] Alasdair Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, 1995.

[158] Hans van Maaren. A short note on some tractable cases of the satisfiability problem. *Information and Computation*, 158(2):125–130, May 2000.

[159] V. Vlček. Classes of boolean formulae with effectively solvable SAT. In Jana Safrankova and Jiri Pavlu, editors, *Proceedings of the 19th Annual Conference of Doctoral Students - WDS 2010*, volume 1, pages 42–47. Matfyzpress, 2010.

[160] Toby Walsh. SAT v CSP. In Rina Dechter, editor, *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[161] John R Woodyard. Nonlinear circuit device utilizing germanium. Patent, 1950. US2530110 A.

[162] Susumu Yamasaki and Shuji Doshita. The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic. *Information and Control*, 59:1–12, 1983.

[163] Emmanuel Zarpas. Back to the SAT05 competition: an a posteriori analysis of solver performance on industrial benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:229–237, 2006.

[164] Hantao Zhang. Combinatorial designs by SAT solvers. In Biere et al. [23], chapter 17, pages 533–568.