PRIFYSGOL CYMRU ABERTAWE
UNIVERSITY OF WALES SWANSEA

UNIVERSITY OF WALES SWANSEA

REPORT SERIES

Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel

by

Oliver Kullmann, Inês Lynce and João Marques-Silva

Report # CSR 3-2006

Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel

Oliver Kullmann*
Computer Science Department
University of Wales Swansea
Swansea, SA2 8PP, UK
e-mail: O.Kullmann@Swansea.ac.uk
http://cs-svr1.swan.ac.uk/~csoliver

Inês Lynce
Departamento de Engenharia Informatica
Instituto Superior Tecnico
Universidade Tecnica de Lisboa
e-mail: ines@sat.inesc-id.pt
http://sat.inesc-id.pt/~ines

João Marques-Silva
School of Electronics and Computer Science
University of Southampton
Highfield, Southampton SO17 1BJ, UK
e-mail: jpms@soton.ac.uk
http://www.ecs.soton.ac.uk/~jpms

March 18, 2006

Abstract

Finding out that a SAT problem instance F is unsatisfiable is not enough for applications, where *good reasons* are needed for explaining the inconsistency (so that for example the inconsistency may be repaired). Previous attempts of finding such good reasons focused on finding some minimally unsatisfiable sub-clause-set F' of F, which in general suffers

*Supported by EPSRC Grant GR/S58393/01

from the non-uniqueness of F' (and thus it will only find *some reason*, albeit there might be others).

In our work, we develop a fuller approach, enabling a more fine-grained analysis of necessity and redundancy of clauses, supported by meaningful semantical and proof-theoretical characterisations. We combine known techniques for searching and enumerating minimally unsatisfiable sub-clause-sets with (full) autarky search. To illustrate our techniques, we give a detailed analysis of well-known industrial problem instances.

1 Introduction

Explaining the causes of unsatisfiability of Boolean formulas is a key requirement in a number of practical applications. A paradigmatic example is SAT-based model checking, where analysis of unsatisfiability is an essential step ([7, 22]) for ensuring completeness of bounded model checking ([3]). Additional examples include fixing wire routing in FPGAs ([24]), and repairing inconsistent knowledge from a knowledge base ([21]).

Existing work on finding the causes of unsatisfiability can be broadly organised into two main categories. The first category includes work on obtaining a *reasonable* unsatisfiable sub-formula, with no guarantees with respect to the size of the sub-formula ([5, 11, 28, 4]). The second category includes work that provides some *guarantees* on the computed sub-formulas ([10, 20, 23]). Most existing work has focused on computing one minimally unsatisfiable sub-formula or all minimally unsatisfiable sub-formulas. Thus also relevant here is the literature on minimally unsatisfiable clause-sets, for example the characterisation of minimally unsatisfiable clause-sets of small deficiency ([1, 9, 6, 12]), where [12] might be of special interest here since it provides an algorithm (based on matroids) searching for "simple" minimally unsatisfiable sub-clause-sets.

In this paper now we seek to obtain a more differentiated picture of the (potentially many and complicated) causes of unsatisfiability by a characterisation of (single) clauses based on their contribution to the causes of unsatisfiability. The following subsection gives on overview on the this categorisation of clauses.

1.1 From necessary to unusable clauses

The problem is to find some "core" in an unsatisfiable clause-set F: Previous attempts were (typically) looking for some minimally unsatisfiable sub-clause-set $F' \subseteq F$, that is, selecting some element $F' \in \mathrm{MU}(F)$ from the set of all minimally unsatisfiable sub-clause-sets of F. The problem here is that $\mathrm{MU}(F)$ in general has many elements, and thus it is hard to give meaning to this process. So let us examine the role the elements of F play for the unsatisfiability of F.

At the base level we have *necessary clauses*, which are clauses whose removal renders F satisfiable. These clauses can also be characterised by the condition that they must be used in every resolution refutation of F, and the set of all necessary clauses is $\bigcap \mathrm{MU}(F)$ (the intersection of all minimally unsatisfiable sub-clause-sets). Determining $\bigcap \mathrm{MU}(F)$ is not too expensive (assuming the SAT decision for F and sub-clause-sets is relatively easy), and every "core analysis" of F should determine these clauses as the core parts of F. It is $\bigcap \mathrm{MU}(F)$ itself unsatisfiable if and only if F has exactly one minimally unsatisfiable sub-clause-set (that is, $|\mathrm{MU}(F)| = 1$ holds), and in this case our job is finished. However, in many situations we do not have a unique minimally unsatisfiable core, but $\bigcap \mathrm{MU}(F)$ has to be "completed" in some sense to achieve unsatisfiability.

At the next level we consider *potentially necessary clauses*, which are clauses which can become necessary clauses when removing some other (appropriately chosen) clauses. The set of all potentially necessary clauses is $\bigcup \mathrm{MU}(F)$ (the union of all minimally unsatisfiable sub-clause-sets); $\bigcup \mathrm{MU}(F)$ is unsatisfiable and seems to be the best choice for a canonical unsatisfiable core of F. However, it is harder to compute than $\bigcap \mathrm{MU}(F)$, and the best method in general seems to consist in enumerating in some way all elements of $\mathrm{MU}(F)$. Clauses which are potentially necessary but which are not necessary are called *only potentially necessary*; these are clauses which make an essential contribution to the unsatisfiability of F, however not in a unique sense (other clauses may play this role as well).

$\bigcup \mathrm{MU}(F)$ is the set of all clauses in F which can be forced to be used in every resolution refutation by removing some other clauses. Now at the third and weakest level of our categorisation of "core clauses" we consider all *usable clauses*, that is, all clauses which can be used in *some* resolution refutation (without dead ends); the set of all usable clauses of F is $\mathrm{N_a}(F)$ (see below for an explanation for this notation). Clauses which are usable but not potentially necessary are called *only usable*; these clauses are superfluous from the semantical point of view (if C is only usable in F, and $F' \subseteq F$ is unsatisfiable, then also $F' \setminus \{C\}$ is unsatisfiable), however their use may considerably shorten resolution refutations of F, as can be seen by choosing F as a pigeonhole formula extended by appropriate clauses introduced by Extended Resolution: Those new clauses are only usable, but without them pigeonhole formulas require exponential resolution refutations, while with them resolution refutations become polynomial.

Dual to these three categories of "necessity" we have the corresponding degrees of "redundancy", where a SAT solver might aim at removing redundant clauses to make its life easier; however this also can backfire (by making the problem harder for the solver and even for non-deterministic proof procedures). The weakest notion is given by *unnecessary clauses*; the set of all unnecessary clauses is $F \setminus \bigcap \mathrm{MU}(F)$. Removing one such a clause still leaves the clause-set unsatisfiable, but in general we cannot remove two unnecessary clauses simultaneously (after removal of some clauses other clauses might become necessary).

At the next (stronger) level we have *never necessary clauses*, that is, clauses which are not potentially necessary; the set of all never necessary clauses is $F \setminus \bigcup \mathrm{MU}(F)$. Here now we can remove several never necessary clauses at the same time, and still we are guaranteed to maintain unsatisfiability; however it might be that after removal of never necessary clauses the resolution complexity is (much) higher than before.

For necessary clauses we have a "proof-theoretical" characterisation, namely that they must be used in any resolution refutation, and an equivalent "semantical" characterisation, namely that removal of them renders the clause-set satisfiable. Now for unnecessary clauses we also have a semantical criterion, namely a clause is never necessary iff it is contained in every maximal satisfiable sub-clause-set.

Finally the strongest notion of redundancy is given by *unusable clauses*; the set of unusable clauses is $F \setminus \mathrm{N_a}(F)$. These clauses can always be removed without any harm (that is, at least for a non-deterministic resolution-based SAT algorithm). As shown in [15], a clause $C \in F$ is unusable if and only if there exists an *autarky* for F satisfying C. This enables a non-trivial computation of $\mathrm{N_a}(F)$ (as discussed in Section 4), which is among the categorisation algorithms considered here the least expensive one, and thus can be used for example as a preprocessing step.

1.2 Organisation of the paper

The paper is organised as follows. The next section introduces the notation used throughout the paper. Section 3 develops the proposed clause categorisation for unsatisfiable clause sets. A discussion on the computation of the lean kernel is included in Section 4. Section 5 presents results for the well-known Daimler-Chrysler's [27] problem instances. Finally, Section 6 concludes the paper and outlines future research work.

2 Preliminaries

2.1 Clause-sets and autarkies

We are using a standard environment for (boolean) clause-sets, partial assignments and autarkies; see [15, 17] for further details and background. Clauses are complement-free (i.e., non-tautological) sets of literals, clause-sets are sets of clauses. The application of a partial assignment φ to a clause-set F is denoted by $\varphi * F$. An autarky for a clause-set F is a partial assignment φ such that every clause $C \in F$ touched by φ (i.e., $\mathrm{var}(\varphi) \cap \mathrm{var}(C) \neq \emptyset$) is satisfied by

φ.[1] Applying autarkies is a satisfiability-equivalent reduction, and repeating the process until no further autarkies are found yields the (uniquely determined) *lean kernel* $\mathrm{N_a}(F) \subseteq F$.

2.2 Hypergraphs

A hypergraph here is a pair $G = (V, E)$, where V is a (finite) set of vertices and $E \subseteq \mathbb{P}(V)$ is a set of subsets.

Let $\mathsf{C}(G) := (V(G), \{V(G) \setminus E : E \in E(G)\})$ be the *complement hypergraph* of G. Obviously we have $\mathsf{C}(\mathsf{C}(G)) = G$.

A *transversal* of G is a subset $T \subseteq V(G)$ such that for all $E \in E(G)$ we have $T \cap E \neq \emptyset$; the hypergraph with vertex set V and hyperedge set the set of all minimal transversals of G is denoted by $\mathbf{Tr}(G)$; we have the well-known fundamental fact (see for example [2])

$$\mathrm{Tr}(\mathrm{Tr}(G)) = \min(G), \tag{1}$$

where $\min(G)$ is the hypergraph with vertex set $V(G)$ and hyperedges all inclusion minimal elements of G (the dual operator is $\max(G)$).

An *independent set* of G is a subset $I \subseteq V(G)$ such that $V(G) \setminus I$ is a transversal of G; in other words, the independent sets of G are the subsets $I \subseteq V(G)$ such that no hyperedge $E \in E(G)$ with $E \subseteq I$ exists. Let $\mathbf{Ind}(G)$ denote the hypergraph with vertex set G and as hyperedges all maximal independent sets of G. By definition we have

$$\mathrm{Ind}(G) = \mathsf{C}(\mathrm{Tr}(G)). \tag{2}$$

2.3 Sub-clause-sets

For a clause-set F let $\boldsymbol{\mathcal{USAT}}(F)$ be the hypergraph with vertex set F and hyperedges the set of all unsatisfiable sub-clause-sets of F, and let $\mathbf{MU}(F) := \min(\mathcal{USAT}(F))$. Thus $\mathrm{MU}(F)$ has as hyperedges all minimally unsatisfiable sub-clause-sets of F, and $\mathrm{MU}(F) = \emptyset \Leftrightarrow F \in \mathcal{SAT}$.

And let $\boldsymbol{\mathcal{SAT}}(F)$ be the hypergraph with vertex set F and hyperedges the set of all satisfiable sub-clause-sets of F, and $\mathbf{MS}(F) := \max(\mathcal{SAT}(F))$. Thus $\mathrm{MS}(F)$ has as hyperedges all maximal satisfiable sub-clause-sets of F, and $F \in \mathrm{MS}(F) \Leftrightarrow F \in \mathcal{SAT}$; we always have $\mathrm{MS}(F) \neq \emptyset$.

Finally let $\mathbf{CMU}(F) := \mathsf{C}(\mathrm{MU}(F))$ and $\mathbf{CMS}(F) := \mathsf{C}(\mathrm{MS}(F))$.

In [20] the observation of Bailey and Stuckey has been used that for every clause-set F we have

$$\mathrm{MU}(F) = \mathrm{Tr}(\mathrm{CMS}(F)). \tag{3}$$

[1] Equivalently, φ is an autarky for F iff for all $F' \subseteq F$ we have $\varphi * F' \subseteq F'$.

This can be shown as follows: By definition we have $\mathrm{MS}(F) = \mathrm{Ind}(\mathrm{MU}(F))$, whence $\mathrm{MS}(F) = \complement(\mathrm{Tr}(\mathrm{MU}(F)))$ by (2), and thus $\complement(\mathrm{MS}(F)) = \mathrm{Tr}(\mathrm{MU}(F))$; applying Tr to both sides we get $\mathrm{Tr}(\complement(\mathrm{MS}(F))) = \mathrm{Tr}(\mathrm{CMS}(F)) = \mathrm{MU}(F)$ by (1).

3 Classification

Let $F \in \mathcal{USAT}$ be an unsatisfiable clause-set for this section. When we speak of a resolution refutation "using" a clause C then we mean the refutation uses C as an axiom (and we consider here only resolution refutations without "dead ends"; since we are not interested in resolution *complexity* here this can be accomplished most easily by only considering *tree* resolution refutations).

3.1 Necessary clauses

The highest degree of necessity is given by "necessary clauses", where a clause $C \in F$ is called **necessary** if every resolution refutation of F must use C. By completeness of resolution, a clause C is necessary iff there exists a partial assignment φ satisfying $F \setminus \{C\}$. So we can compute all necessary clauses by running through all clauses and checking whether removal renders the clause-set satisfiable. The set of all necessary clauses of F is $\bigcap \mathrm{MU}(F)$.

Clause-sets with $F = \bigcap \mathrm{MU}(F)$, that is, clause-sets where every clause is necessary, are exactly the minimally unsatisfiable clause-sets. So the complexity of computing $\bigcap \mathrm{MU}(F)$ is closely related to deciding whether a clause-set F is minimally unsatisfiable, which is a D^P-complete decision problem (see [25]).

The corresponding (weakest) notion of redundancy is that of clauses which are **unnecessary**, which are clauses $C \in F$ such that $F \setminus \{C\}$ still is unsatisfiable, or, equivalently, clauses for which resolution refutations of F exist not using this clause.

3.2 Potentially necessary clauses

$C \in F$ is called **potentially necessary** if there exists an unsatisfiable $F' \subseteq F$ with $C \in F'$ such that C is necessary for F'. In other words, potentially necessary clauses become necessary (can be forced into every resolution refutation) by removing some other clauses. Obviously the set of potentially necessary clauses is $\bigcup \mathrm{MU}(F)$ (and every necessary clause is also potentially necessary).

The class of (unsatisfiable) clause-sets F with $F = \bigcup \mathrm{MU}(F)$ (unsatisfiable clause-sets, where every clause is potentially necessary) has been considered in [15], and it is mentioned that these clause-sets are exactly those clause-sets

obtained from minimally unsatisfiable clause-sets by the operation of crossing out variables: The operation of crossing out a set of variables V in F is denoted by $V * F$. That if F is minimally unsatisfiable, then $V * F$ is the union of minimally unsatisfiable clause-sets, has been shown in [26]. For the converse direction consider the characteristic case of two minimally unsatisfiable clause-sets F_1, F_2. Choose a new variable v and let $F := \{C \cup \{v\} : C \in F_1\} \cup \{C \cup \{\overline{v}\} : C \in F_2\}$; obviously F is minimally unsatisfiable and $\{v\} * F = F_1 \cup F_2$.

So given (unsatisfiable) F with $F = \bigcup \mathrm{MU}(F)$, we have a (characteristic) representation $F = V * F_0$ for some minimally unsatisfiable F_0; it is conceivable but not known to the authors whether such a representation might be useful (considering "good" F_0). The complexity of deciding whether for a clause-set F we have $F = \bigcup \mathrm{MU}(F)$ is not known to the authors; by definition the problem is in PSPACE, and it seems to be a very hard problem. See below for the computation of $\bigcup \mathrm{MU}(F)$.

Clauses which are potentially necessary, but which are not necessary (i.e., the clauses in $\bigcup \mathrm{MU}(F) \setminus \bigcap \mathrm{MU}(F)$), are called **only potentially necessary**.

By Lemma 4.3 in [12] we have $\bigcup \mathrm{MU}(F) = F \setminus \bigcap \mathrm{MS}(F)$, i.e., a clause is potentially necessary iff there exists a maximally satisfiable sub-clause-set not containing this clause, or, in other words, a clause is not potentially necessary iff the clause is in every maximally satisfiable sub-clause-set.

Thus for computing $\bigcup \mathrm{MU}(F)$ we see two possibilities:

1. Enumerating $\mathrm{MU}(F)$ and computing $\bigcup \mathrm{MU}(F)$.
2. Enumerating $\mathrm{MS}(F)$ and computing $\bigcup \mathrm{MU}(F) = F \setminus \bigcap \mathrm{MS}(F)$ (this is more efficient than using (3), since for applying (3) we must store all elements of $\mathrm{MS}(F)$, and furthermore it is quite possible that while $\mathrm{MS}(F)$ is a small set, $\mathrm{MU}(F)$ is a big set).

The corresponding (medium) degree of redundancy is given by clauses which are **never necessary** (not potentially necessary), that is, clauses which can not be forced into resolution refutations by removing some other clauses, or equivalently, clauses which are contained in every maximally satisfiable sub-clause-set. A clause which is never necessary is also unnecessary.

Blocked clauses (see [13]), and, more generally, clauses eliminated by repeated elimination of blocked clauses, are never necessary; an interesting examples for such clauses are clauses introduced by extended resolution (see [14]).

3.3 Usable clauses

The weakest degree of necessity if given by "usable clauses", where $C \in F$ is called **usable** if there exists some tree resolution refutation of F using C. Obviously every potentially necessary clause is a usable clause.

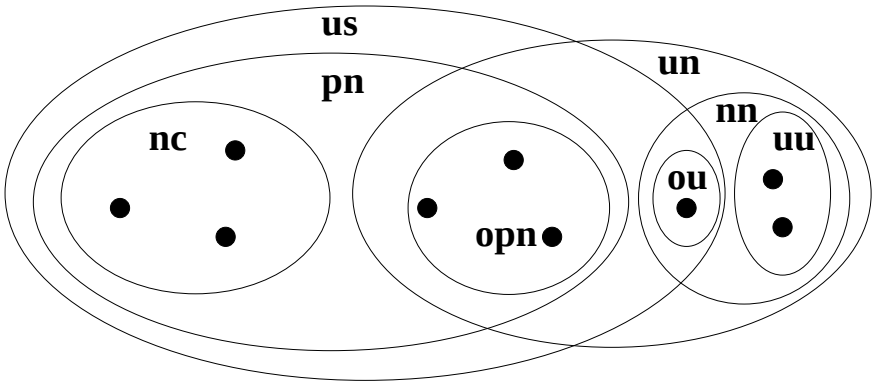

Figure 1: Clause classification: an example.

By Theorem 3.16 in [15] the set of usable clauses is exactly the lean kernel $\mathrm{N_a}(F)$. The set of F with $\mathrm{N_a}(F) = F$, which are called *lean clause-sets* (every clause is usable) has been studied in [17], and the decision problem whether a clause-set is lean has been shown to be co-NP complete. In Section 4 we discuss the computation of the lean kernel in greater detail.

The corresponding strongest degree of redundancy is given by **unusable clauses**, clauses $C \in F$ which are not used in any resolution refutation, which are exactly the clauses for which an autarky φ for F exists satisfying C. An unusable clause is never necessary.

Clauses which are never necessary but are which are usable are called **only usable**, and are given for example by clauses (successfully) introduced by Extended Resolution: They are never necessary as discussed before, but they are usable (since we assumed the introduction to be "successful"), and actually these clauses can exponentially speed up the resolution refutation as shown in [8].

3.4 Discussion

Figure 1 relates the concepts introduced above. Consider a formula with 9 clauses (represented with bullets). These clauses can be partitioned into necessary clauses (nc) and unnecessary clauses (un). The unnecessary clauses can be partitioned into only potentially necessary clauses (opn) and never necessary clauses (nn). The (disjoint) union of the only potentially necessary clauses with the necessary clauses gives the potentially necessary clauses (pn). In addition, the never necessary clauses can be partitioned into only usable clauses (ou) and unusable clauses (uu). The (disjoint) union of the potentially necessary clauses with the only usable clauses gives the usable clauses (us).

3.5 Finding the cause

Given is an unsatisfiable clause-set F, which is partitioned into $F = F_\mathrm{s} \uplus F_\mathrm{u}$, where F_s come from "system axioms", while F_u comes from a specific "user requirements". The unsatisfiability of F means that the user requirements together with the system axioms are inconsistent, and the task now is to find "the cause" of this problem.

First if F_u is already unsatisfiable, then the user made a "silly mistake", while if F_s already is unsatisfiable, then the whole system is corrupted. So we assume that F_u as well as F_s is satisfiable.

The natural first step now is to consider $\bigcap \mathrm{MU}(F)$. The best case is that $\bigcap \mathrm{MU}(F)$ is already unsatisfiable (i.e., F has a unique minimally unsatisfiable sub-clause-set). Now $F_\mathrm{u} \cap \bigcap \mathrm{MU}(F)$ are the critical user requirements which together with the system properties $F_\mathrm{s} \cap \bigcap \mathrm{MU}(F)$ yield the (unique) contradiction. So assume that $\bigcap \mathrm{MU}(F)$ is satisfiable in the sequel.

That $F_\mathrm{s} \cap \bigcap \mathrm{MU}(F) \neq \emptyset$ is the case typically does not reveal much; it can be a very basic requirement which when dropped (or when "some piece is broken out of it") renders the whole system meaningless (if for example numbers would be used, then we could have "if addition wouldn't be addition, then there would be no problem"). However if $F_\mathrm{u} \cap \bigcap \mathrm{MU}(F) \neq \emptyset$ holds, then this could contain valuable information: These clauses could also code some very basic part of the user requirement, where without these requirements the whole user requirement breaks down, and then (again) we do not know much more than before; if however at least some clauses code some very specific requirement, then perhaps with their identification already the whole problem might have been solved.

In general the consideration of $\bigcap \mathrm{MU}(F)$ is not enough to find "the cause" of the unsatisfiability of F. Finding some $F' \in \mathrm{MU}(F)$ definitely yields some information: F' will contain some system clauses and some user clauses which together are inconsistent, however this inconsistency might not be the only inconsistency. Also if $F \setminus F'$ is satisfiable (which is guaranteed if $\bigcap \mathrm{MU}(F) \neq \emptyset$) we do not gain much, again because some very fundamental pieces might now be missing.

So what really is of central importance here is $\bigcup \mathrm{MU}(F)$. The clauses $F_\mathrm{u} \cap \bigcup \mathrm{MU}(F)$ are exactly all (pieces) of user requirements which can cause trouble, while the clauses $F_\mathrm{s} \cap \bigcup \mathrm{MU}(F)$ are exactly all pieces of basic requirements needed (under certain circumstances) to complete the contraction. The clauses in $F \setminus \bigcup \mathrm{MU}(F)$, the unnecessary clauses, might be helpful to see some contradiction with less effort, but they are never really needed.

So what now is the role of $\mathrm{N_a}(F)$ (the lean kernel, or, in other words, the set of usable clauses) here?! To identify the causes of inconsistency the clauses in $\mathrm{N_a}(F) \setminus \bigcup \mathrm{MU}(F)$ (the only usable clauses) are not needed. One role of $\mathrm{N_a}(F)$

is as a stepping stone for the computation of $\bigcup \mathrm{MU}(F)$, since the computation of $\mathrm{N_a}(F)$ is easier than the computation of $\bigcup \mathrm{MU}(F)$, and removing the "fat" helps to get easier to the potentially necessary clauses. Another, quite different role now is, that the set $F \setminus \mathrm{N_a}(F)$ of unusable clauses are the clauses satisfied by a maximal autarky φ; and this φ can be considered as the largest "conservative model", which doesn't remove any possibilities to satisfy further clauses.

Satisfying any clause from $\mathrm{N_a}(F)$ necessarily implies that some other clause is touched but not satisfied. Trying to satisfy these touched clauses will lead to an element $F' \in \mathrm{MS}(F)$, which are characterised by the condition that every satisfying assignment φ for F' must falsify all clauses in $F \setminus F'$ (whence these satisfying assignment are normally not useful here). In a certain sense a maximal autarky φ for F is the largest generally meaningful model for some part. Finding such a model yields a fulfilment of the "really harmless" user requirements. So with the set of potentially necessary clauses we covered all causes of the unsatisfiability, while with the set of unusable clauses we covered everything what can be "truly satisfied" (without remorse).

4 Computing the lean kernel

In Section 6 of [16] the following procedure for the computation of a "maximal autarky" φ for F (that is, an autarky φ for F with $\varphi * F = \mathrm{N_a}(F)$) has been described, using a SAT solver $\mathcal{A}$ which for a satisfying input F returns a satisfying assignment φ with $\mathrm{var}(\varphi) \subseteq \mathrm{var}(F)$, while for an unsatisfiable input F a set $V \subseteq \mathrm{var}(F)$ of variables is returned which is the set of variables used in some (tree) resolution refutation of F:

1. Apply $\mathcal{A}(F)$; if F is satisfiable then return φ.
2. Otherwise let $F := F[V]$, and go to Step 1.

Here $F[V]$ is defined as $(V * F) \setminus \{\bot\}$, where $V * F$ denotes the operation of removing all literals x from F with $\mathrm{var}(x) \in V$, while $\bot$ is the empty clause. So the above procedure can be outlined as follows: Apply the given SAT solver $\mathcal{A}$ to F. If we obtain a satisfying assignment, then φ is a maximal autarky for the original input (and applying it we obtain the lean kernel). Otherwise we obtain a set V of variable used in a resolution refutation of F; cross out all these variables from F, remove the (necessarily obtained) empty clause, and repeat the process.

Correctness follows immediately with Theorem 3.16 in [15] together with Lemma 3.5 in [15]. More specifically, Lemma 3.5 in [15] guarantees that if by iterated reduction $F \to F[V]$ for arbitrary sets V of variables at the end we obtain some satisfiable F^* then any satisfying assignment φ for F^* with

$\mathrm{var}(\varphi) \subseteq \mathrm{var}(F^*)$ is an autarky for F (thus the above process returns only autarkies). In other direction (which is the non-trivial part) Theorem 3.16 guarantees that by using such V coming from resolution refutations we don't loose any autarky.

The computation of V by a SAT solver can be done following directly the correspondence between tree resolution refutations and semantic trees (for a detailed treatment see [18]). Since the set of used variables needs to be maintained only on the active path, the space required by this algorithm is (only) quadratic in the input size; the only implementation of this algorithm we are aware of is in `OKsolver` (as participated in the SAT 2002 competition), providing an implementation of "intelligent backtracking" without learning; see [19] for a detailed investigation.

By heuristical reasoning, a procedure computing some unsatisfiable $F' \subseteq \mathrm{N_a}(F)$ for unsatisfiable F has been given in [28], also based on computing resolution refutations. Compared to the autarky approach, F' is some set of usable clauses, while $\mathrm{N_a}(F)$ is the set of *all* usable clauses. Furthermore $\mathrm{N_a}(F)$ comes with an autarky (a satisfying assignment for all the other clauses, not touching $\mathrm{N_a}(F)$), and the computation of $\mathrm{N_a}(F)$ can be done quite space-efficient (as outlined above), while [28] compute the whole resolution tree, and thus the space requirements can be exponential in the input size.

5 Experimental results

The main goal of this section is to analyse a set of problem instances with respect to the concepts described above. To achieve this goal, we have selected 38 problem instances from the DC family ([27])[2)]. These instances are obtained from the validation and verification of automotive product configuration data and encode different consistency properties of the configuration data base which is used to configure Daimler Chrysler's Mercedes car lines. For example, some instances refer to the stability of the order completion process (SZ), while others refer to the order independence of the completion process (RZ) or to superfluous parts (UT). We have chosen these instances because they are well known for having small minimal unsatisfiable cores and usually more than one minimal unsatisfiable core [20]. Hence, they provide an interesting testbed for the new concepts introduced in the paper.

The size of DC problem instances analysed in this paper ranges from 1659 to 1909 variables and 4496 to 8686 clauses. However, and as mentioned in [20], these formulas have a few repeated clauses and also repeated literals in clauses. Also, there are some variable codes that are not used. Consequently, we have performed a preprocessing step to eliminate the repeated clauses and literals,

[2)] Available from http://www-sr.informatik.uni-tuebingen.de/~sinz/DC/.

as well as non-used variables. In the resulting formulas the number of variables ranges from 1513 to 1805 and the number of clauses ranges from 4013 to 7562.

Table 1 gives the number of variables, the number of clauses and the average clause size for each of the 38 problem instances from the DC family. Table 1 also gives the number of minimal unsatisfiable sub-clause-sets (#MU) contained in each formula, the number of maximal satisfiable sub-clause-sets (#MS) (recall (3)), the percentage of necessary clauses (nc) and the percentage of the number of clauses in the smallest (min) and largest (max) minimal unsatisfiable sub-clause-set. Furthermore Table 1 shows the percentages of only potentially necessary clauses (opn) and the percentage of only potentially necessary clauses (pn), as well as the percentage of only usable clauses (ou) and the percentage of usable clauses (us). Then redundant clauses are considered: the percentage of unusable clauses (un), the percentage of never necessary clauses (nn) and the percentage of unnecessary clauses (un). Recall that uu stands for the clauses which can be satisfied by some autarky; in the final column we give the percentage of the uu-clauses which can be covered by (iterated) elimination of pure literals alone.[3)]

These results have been obtained using a tool provided by the authors of [20], and also a Perl script for computing the lean kernel that iteratively invokes a SAT solver ([28]) which identifies variables used in a resolution refutation. From this table some conclusions can be drawn. As one would expect in general to be the case, as the number of mus's increases, the relative number of necessary clauses decreases. Regarding the number of mus's we see of lot of variation: Although half of the problem instances have only a few mus's, there are also many problems with many mus's. In addition, there seems to be no relation between the number of mus's and the number of mss's.

Looking at the levels of necessity, we may observe the following. For all instances the percentage of clauses in the smallest mus is quite small (in most cases less than 1%) and the largest mus is usually not much larger than the smallest one. The number of potentially necessary clauses is is typically somewhat bigger than the size of the largest mus, but for all instances the set of potentially necessary clauses is still fairly small. The percentage of usable clauses is typically substantially larger, but only for the `UT`-family more than half of all causes are usable.

Looking at the levels of redundancy, we see that in many cases autarky reduction to a large part boils down to elimination of pure literals. In most cases most never necessary clauses are already unusable, with the notable exceptions of the `UT`- and (to a somewhat lesser degree) the `SZ`-family, while almost all unnecessary clauses are already never necessary.

[3)] Since all instances contain necessary clauses, the maximum (size) maximal satisfiable sub-clause-sets are always as large as possible (only one clause missing); the minimum (size) maximal satisfiable sub-clause-sets here are never much smaller, so we considered these number negligible.

6 Conclusions

This paper proposes a categorisation of clauses in unsatisfiable instances of SAT, with the objective of developing new insights into the structure of unsatisfiable formulas. The paper also addresses which sets of clauses are relevant when dealing with unsatisfiable instances of SAT. Finally, the paper evaluates the proposed categorisation of clauses in well-known unsatisfiable problem instances, obtained from industrial test cases [27].

We see the following main directions for future research:

- Regarding the industrial test cases considered, we were mainly interested in them as proof of concept, and likely there are many more interesting relations hidden in the data (especially when combining them with special insights into the structure of these formulas).
- In Subsection 3.5 we outlined a general approach for finding causes of unsatisfiability in a scenario motivated by the [27]; it would now be interesting to see how helpful these considerations are in practice.
- Obviously there are many non-trivial problems regarding the complexity of the algorithms involved. A main problem here, which according to our knowledge has not been tackled until now, is the complexity regarding the computation of $\bigcup \mathrm{MU}(F)$.
- For the computation of the lean kernel in this paper we considered an algorithm exploiting the "duality" between resolution proofs and autarkies. It would be interesting to compare this approach with a direct approach (directly searching for autarkies).
- Finally, it would be interesting to perform an analysis as in Table 1 on many other classes of SAT problems and to see how useful these statistics are for the categorisation of classes of problem instances.

References

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

[2] Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. North Holland, Amsterdam, 1989. ISBN 0 444 87489 5; QA166.23.B4813 1989.

Table 1: The structure of MU(F) for formulas F from the DC family, the number of variables, the number of clauses, the average clause size, the number of minimally unsatisfiable and maximal satisfiable sub-clause-sets, and as percentages the number of necessary clauses, the minimal and maximal sizes of the minimally unsatisfiable sub-clause-sets, the only potentially necessary clauses, the potentially necessary clauses, the only usable clauses, the usable clauses, the unusable clauses, the never necessary clauses and the unnecessary clauses; finally the percentage of uu (autarky reduction) achievable by pure literal elimination (iterated) alone.

Bench	#vars	#cls	μ w	#MU	#MS	nc	min	max	opn	pn	ou	us	uu	nn	un	pl
C208_FC_RZ_70	1513	4468	2.14	1	212	4.74	4.74	4.74	0	4.74	4.39	9.13	90.87	95.26	95.26	75.91
C208_FC_SZ_127	1513	4469	2.14	1	34	0.76	0.76	0.76	0	0.76	2.48	3.24	96.76	99.24	99.24	70.54
C208_FC_SZ_128	1513	4469	2.14	1	32	0.72	0.72	0.72	0	0.72	2.17	2.89	97.11	99.28	99.28	70.28
C208_FA_RZ_64	1516	4246	2.04	1	212	4.99	4.99	4.99	0	4.99	4.62	9.61	90.39	95.01	95.01	76.65
C208_FA_SZ_120	1516	4247	2.04	2	34	0.78	0.8	0.8	0.05	0.82	2.52	3.34	96.66	99.18	99.22	70.89
C208_FA_SZ_121	1516	4247	2.04	2	32	0.73	0.75	0.75	0.05	0.78	2.19	2.97	97.03	99.22	99.27	70.61
C208_FA_SZ_87	1516	4255	2.04	12884	139	0.31	0.42	0.63	0.8	1.1	1.06	2.16	97.84	98.9	99.69	69.59
C170_FR_RZ_32	1528	4067	2.28	32768	242	5.21	5.58	5.61	0.76	5.97	23.29	29.26	70.74	94.03	94.79	100
C170_FR_SZ_95	1528	4068	2.28	6863389	175	0.71	0.93	1.62	1.5	2.21	2.8	5.01	94.99	97.79	99.29	75.57
C170_FR_SZ_58	1528	4083	2.31	218692	177	0.91	1.13	1.54	1.44	2.35	30.69	33.04	66.96	97.65	99.09	100
C170_FR_SZ_92	1528	4195	2.28	1	131	3.12	3.12	3.12	0	3.12	31.97	35.09	64.91	96.88	96.88	100
C220_FV_RZ_14	1530	4013	2.13	80	20	0.12	0.27	0.45	0.5	0.62	16.97	17.59	82.14	99.11	99.88	95.19
C220_FV_RZ_13	1530	4014	2.13	6772	76	0.07	0.25	0.67	1.07	1.15	17.61	18.76	81.24	98.85	99.93	95.89
C220_FV_SZ_65	1530	4014	2.15	103442	198	0.45	0.57	1	2.12	2.57	15.94	18.51	81.49	97.43	99.55	95.93
C220_FV_RZ_12	1530	4017	2.13	80272	150	0.07	0.27	0.87	1.32	1.39	18.28	19.67	80.33	98.61	99.93	96.22
C220_FV_SZ_121	1530	4035	2.18	9	102	1.34	1.44	1.61	0.45	1.78	3.15	4.93	95.07	98.22	98.66	80.92
C202_FS_SZ_122	1556	5385	2.79	1	33	0.61	0.61	0.61	0	0.61	7.97	8.58	91.42	99.39	99.39	78.65
C202_FS_SZ_121	1556	5387	2.79	4	24	0.37	0.41	0.45	0.11	0.48	6.11	6.59	93.41	99.52	99.63	79.09
C202_FS_SZ_95	1556	5388	2.78	3415443	59307	0.45	0.65	0.91	1.57	2.02	1.69	3.71	96.29	97.98	99.55	80.53
C202_FS_RZ_44	1556	5399	2.79	4589596	2658	0.22	0.33	0.98	2.13	2.35	0.74	3.09	96.91	97.65	99.78	80.28
C202_FS_SZ_104	1556	5405	2.81	1626843	3593	0.48	0.48	1.72	2.09	2.57	18.1	20.67	79.33	97.43	99.52	97.07
C202_FW_RZ_57	1561	7434	3.32	1	213	2.87	2.87	2.87	0	2.87	2.63	5.5	94.5	97.13	97.13	58.33
C202_FW_SZ_124	1561	7435	3.32	1	33	0.44	0.44	0.44	0	0.44	30.41	30.85	69.15	99.56	99.56	86.5
C202_FW_SZ_123	1561	7437	3.32	4	38	0.46	0.48	0.51	0.08	0.54	30.56	31.1	68.9	99.46	99.54	86.46
C202_FW_SZ_118	1561	7562	3.3	4194235	257	0.53	0.54	1.75	1.52	2.05	27.89	29.94	70.06	97.95	99.47	98.74
C210_FS_RZ_40	1607	4891	2.87	15	212	2.74	2.86	3.54	1.53	4.27	3.89	8.16	91.84	95.73	97.26	67.76
C210_FS_SZ_129	1607	4894	2.87	1	33	0.67	0.67	0.67	0	0.67	26.87	27.54	72.46	99.33	99.33	97.67
C210_FS_SZ_130	1607	4894	2.87	1	31	0.63	0.63	0.63	0	0.63	26.91	27.54	72.46	99.37	99.37	87.96
C210_FW_RZ_59	1628	6381	3.75	15	212	2.1	2.19	2.71	1.18	3.28	2.97	6.25	93.75	96.72	97.9	61.43
C210_FW_SZ_135	1628	6384	3.75	1	33	0.52	0.52	0.52	0	0.52	34.47	34.99	65.01	99.48	99.48	90.36
C210_FW_SZ_136	1628	6384	3.75	1	31	0.49	0.49	0.49	0	0.49	34.5	34.99	65.01	99.51	99.51	90.36
C210_FW_SZ_129	1628	6595	3.93	3731050	584	1.88	1.9	4.26	2.5	4.38	35.13	39.51	60.49	95.62	98.12	87.96
C168_FW_UT_855	1804	6752	4.09	102	30	0.09	0.12	0.24	0.36	0.44	54.31	54.75	45.25	99.56	99.91	95.06
C168_FW_UT_854	1804	6753	4.09	102	30	0.09	0.12	0.24	0.36	0.44	54.32	54.76	45.24	99.56	99.91	95.06
C168_FW_UT_852	1804	6756	4.09	102	30	0.09	0.12	0.24	0.36	0.44	54.34	54.78	45.22	99.56	99.91	95.06
C168_FW_UT_851	1804	6758	4.08	102	30	0.09	0.12	0.24	0.36	0.44	54.35	54.79	45.21	99.56	99.91	95.06
C208_FA_UT_3254	1805	6153	4.38	17408	155	0.47	0.65	1.2	1.12	1.59	54.45	56.04	43.96	98.41	99.53	99.26
C208_FA_UT_3255	1805	6156	4.38	52736	155	0.47	0.65	1.2	1.19	1.66	54.38	56.04	43.96	98.34	99.53	99.26

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Y. Zhu. *Highly Dependable Software*, volume 58 of *Advances in Computers*, chapter Bounded model checking. Elsevier, 2003. ISBN 0-12-012158-1.

[4] Renato Bruni. On exact selection of minimally unsatisfiable subformulae. *Annals for Mathematics and Artificial Intelligence*, 43:35–50, 2005.

[5] Renato Bruni and Antonio Sassano. Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. In Henry Kautz and Bart Selman, editors, *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, volume 9 of *Electronic Notes in Discrete Mathematics (ENDM)*. Elsevier Science, June 2001.

[6] Hans Kleine Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107:83–98, 2000.

[7] P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *International Conference on Formal Methods in Computer-Aided Design*, 2002.

[8] Stephen A. Cook. A short proof of the pigeonhole principle using extended resolution. *SIGACT News*, pages 28–32, October-December 1976.

[9] Gennady Davydov, Inna Davydova, and Hans Kleine Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23:229–245, 1998.

[10] M. G. de la Banda, P. J. Stuckey, and J. Wazny. Finding all minimal unsatisfiable sub-sets. In *International Conference on Principles and Practice of Declarative Programming*, 2003.

[11] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Design, Automation and Test in Europe Conference*, pages 10886–10891, March 2003.

[12] Oliver Kullmann. An application of matroid theory to the SAT problem. In *Fifteenth Annual IEEE Conference on Computational Complexity (2003)*, pages 116–124.

[13] Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, July 1999.

[14] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97(1-3):149–176, 1999.

[15] Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107:99–137, 2000.

[16] Oliver Kullmann. On the use of autarkies for satisfiability decision. In Henry Kautz and Bart Selman, editors, *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, volume 9 of *Electronic Notes in Discrete Mathematics (ENDM)*. Elsevier Science, June 2001.

[17] Oliver Kullmann. Lean clause-sets: Generalizations of minimally unsatisfiable clause-sets. *Discrete Applied Mathematics*, 130:209–249, 2003.

[18] 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.

[19] Oliver Kullmann. Modelling the behaviour of a DLL SAT solver on random formulas. In preparation, 2006.

[20] Mark H. Liffiton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 173–186, Berlin, 2005. Springer. ISBN 3-540-26276-8.

[21] B. Mazure, L. Sais, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22(3-4):319–331, 1998.

[22] K. L. McMillan. Interpolation and SAT-based model checking. In *International Conference on Computer-Aided Verification*, 2003.

[23] M. N. Mneimneh, I. Lynce, Z. S. Andraus, K. A. Sakallah, and J. P. Marques-Silva. A branch and bound algorithm for extracting smallest minimal unsatisfiable formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, June 2005.

[24] G.-J. Nam, K. A. Sakallah, and R. A. Rutenbar. Satisfiability-based layout revisited: Detailed routing of complex FPGAs via search-based boolean SAT. In *International Symposium on Field-Programmable Gate Arrays*, February 1999.

[25] Christos H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal on Computer and System Sciences*, 28:244–259, 1984.

[26] Ma Shaohan and Liang Dongmin. A polynomial-time algorithm for reducing the number of variables in MAX SAT problem. *Science in China (Series E)*, 40(3):301–311, June 1997.

[27] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, January 2003.

[28] Lintao Zhang and Sharad Malik. Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In John Franco, Enrico Giunchiglia, Henry Kautz, Hans Kleine Büning, Hans van Maaren, Bart Selman, and Ewald Speckenmeyer, editors, *Sixth International Conference on Theory and Applications of Satisfiability Testing*, pages 239–249, 2003. Santa Margherita Ligure – Portofino (Italy), May 5, 2003 to May 8, 2003.