Prime Compilation of Non-Clausal Formulae

A. Previti
UCD CASL
Dublin, Ireland
alessandro.previti@ucdconnect.ie

A. Ignatiev
INESC-ID, IST
Lisbon, Portugal
aign@sat.inesc-id.pt

A. Morgado
INESC-ID, IST
Lisbon, Portugal
ajrmorgado@gmail.com

J. Marques-Silva
INESC-ID, IST, Lisbon, Portugal
UCD CASL Dublin, Ireland
jpms@tecnico.ulisboa.pt

Abstract

Formula compilation by generation of prime implicates or implicants finds a wide range of applications in AI. Recent work on formula compilation by prime implicate/implicant generation often assumes a Conjunctive/Disjunctive Normal Form (CNF/DNF) representation. However, in many settings propositional formulae are naturally expressed in non-clausal form. Despite a large body of work on compilation of non-clausal formulae, in practice existing approaches can only be applied to fairly small formulae, containing at most a few hundred variables. This paper describes two novel approaches for the compilation of non-clausal formulae either with prime implicants or implicates, that is based on propositional Satisfiability (SAT) solving. These novel algorithms also find application when computing all prime implicates of a CNF formula. The proposed approach is shown to allow the compilation of non-clausal formulae of size significantly larger than existing approaches.

1 Introduction

The compilation of the prime implicants and implicates of propositional formulae can be traced to the work of Blake [Blake, 1937], having been the subject of extensive research over the years (e.g. [Marquis, 2000] for a comprehensive survey on this topic).

Prime implicants and implicates are fundamental in the minimization of propositional formulae [Quine, 1952; 1959; McCluskey, 1956], and find a number of relevant applications and extensions. These include, among others, knowledge compilation [Darwiche and Marquis, 2002; Cadoli and Donini, 1997], model-based diagnosis [de Kleer, 1992], debugging of incoherent terminologies [Schlobach *et al.*, 2007], contingent planning [To *et al.*, 2011], existential quantification [Brauer *et al.*, 2011], extraction of feature models from propositional formulae [Czarnecki and Wasowski, 2007], inductive generalization in model checking [Bradley and Manna, 2007], and applications in modal logic [Bienvenu, 2009]. Depending on the application, the goal is either prime compilation, finding a prime cover of a formula, or extracting some prime implicants/implicates.

Most work on finding all prime implicants (or implicates) of a propositional formula assumes the formula to be represented in Conjunctive (resp. Disjunctive) Normal Form (CNF, resp. DNF). However, in practice most problem representations are not in normal form [Stuckey, 2013]. A simple way to handle non-clausal[1] formulae is to exploit Shannon's expansion [Shannon, 1949], but this approach is worst-case exponential even *before* prime generation. As a result, a number of alternative approaches have been proposed, which work with non-CNF formulae [Coudert and Madre, 1992; Ramesh *et al.*, 1997; Matusiewicz *et al.*, 2009; Simon and del Val, 2001]. However, reported results indicate that these approaches are unable to scale for formulae with large numbers of variables, being limited to formulae with at most a few hundred variables.

This paper develops two novel approaches for prime generation of non-clausal formulae, that build on work for prime implicant generation of CNF formulae using SAT [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014]. Whereas prime implicant generation for CNF formulae starts from a CNF representation that is given, and so the focus is solely on the computation of the prime implicants, our approach proposes to *construct* the CNF representation of a formula (i.e. a (prime) implicate cover, represented with a set of prime implicates, from which a CNF formula F' equivalent to the non-clausal formula F can be obtained), while simultaneously finding and blocking all the prime implicants. Observe that earlier works based on SAT that start from CNF formulae cannot directly compute all prime implicates, whereas the novel approaches proposed in this paper can. Moreover, although similar in the overall organization, the two novel approaches differ in key aspects that tend to favor the performance of the second.

Preliminary experimental results show that the proposed approach scales significantly better than earlier work [Simon and del Val, 2001; Ramesh *et al.*, 1997; Matusiewicz *et al.*, 2009] for different classes of formulae, namely classification theorems for quasigroups [Meier and Sorge, 2005], cryptographic benchmark formulae [Geffe, 1973; Otpuschennikov *et al.*, 2014] and specific crafted formulae [Bordeaux and Marques-Silva, 2012; Beame *et al.*, 2004] with fairly large numbers of variables, but also without large numbers of primes. (In contrast, (Z)BDD-based approaches [Simon and del Val, 2001; Coudert and Madre, 1992] can compile formulae having at most a couple of hundred variables but that

[1]Throughout the paper, the term *non-clausal* is used to denote propositional formulae not necessarily represented as sets of sets of literals, i.e. either CNF or DNF.

can have a very large number of primes.) The two novel approaches proposed in this paper can compile example formulae with thousands of variables, and with hundreds of thousands of primes, which is well beyond the capability of alternative approaches.

2 Related Work

This section briefly surveys related work on finding prime implicant or implicate compilations (e.g. [Marquis, 2000] represents a comprehensive survey, covering work on prime compilations until 2000).

A number of methods for computing all prime implicants (or implicates) are based on iterated consensus (or resolution) operations, with a number of improvements introduced over the years [Quine, 1952; 1959; Tison, 1967; Kean and Tsiknis, 1990]. All these approaches start from a DNF (resp. CNF) formula representation of a formula F and generate all prime implicants (resp. implicates) of F.

Besides iterated consensus (or resolution), a number of alternative approaches have been proposed over the years, also for formulae represented in either DNF or CNF. These include, among others, the use of semantic resolution [Slagle *et al.*, 1970], the use of minimal hitting sets with SE-trees [Rymon, 1994], the matrix method and extensions [Jackson, 1992], the unionist product [Castell, 1996], and the use of problem reformulation [Jabbour *et al.*, 2014], which is motivated by a formulation based on ILP [Palopoli *et al.*, 1999]. Some of this recent work starts from a clausal representation of a formula F and generates the prime implicants of F [Castell, 1996; Palopoli *et al.*, 1999; Jabbour *et al.*, 2014]. Additional work also includes the use of improved data structures [de Kleer, 1992] and extensions for knowledge compilation [Marquis, 1995]. Moreover, the minimal hitting set duality between prime implicants and implicates has been exploited in the use of SE-trees [Rymon, 1994] and can be traced to the work of Reiter [Reiter, 1987].

Most past work on prime compilations require either DNF or CNF representations of Boolean functions (and the above references are evidence of this observation). Shannon's expansion [Shannon, 1949] can be used for converting non-clausal formulae to DNF (or CNF), but this in general does not scale for large formulae. In our case, we are still able to use the more succinct Tseitin encoding and make use of modern CNF based SAT solvers [Eén and Sörensson, 2003]. Further details are provided in Section 4.

Nevertheless, a number of approaches have been considered for non-clausal formulae. One example is an algorithm dedicated to formulae represented as the conjunction of DNF formulae [Ngair, 1993]. Additional examples include the use of BDDs [Coudert and Madre, 1992; Simon and del Val, 2001], the use of NNF with path dissolution [Ramesh *et al.*, 1997] and, more recently, the application of different variants of tries [Matusiewicz *et al.*, 2009]. ZRes [Simon and del Val, 2001] is a tool for the enumeration of prime implicates. Although originally proposed in the context of clausal theories, the tool is able to eliminate Tseitin variables (if known), thus providing a way to return primes for the original non-clausal formula. The tool builds on top of Tison's work and uses a ZBDD in order to compactly encode large clause sets. Finally a new multiresolution rule is presented that is able to deal with this compact representation.

3 Preliminaries

Standard definitions on knowledge compilation and consequence finding are assumed [Cadoli and Donini, 1997; Marquis, 2000; Darwiche and Marquis, 2002]. Moreover, definitions common in propositional satisfiability (SAT) solving are also assumed [Biere *et al.*, 2009]. In what follows, F denotes a propositional formula, that can be in non-clausal form. $\mathsf{var}(F)$ denotes the set of variables of F. A model is an assignment satisfying the formula. A model is said to be *maximal* when it contains a maximal number of variables assigned to *true*. A term t is a conjunction of literals and a clause c is a disjunction of literals.

Definition 1 *A term I_n is called an implicant of F if $I_n \vDash F$.*

Definition 2 *A clause I_e is called an implicate of F if $F \vDash I_e$.*

Definition 3 *An implicant I_n of F is called prime if any subset $I'_n \subsetneq I_n$ is not an implicant of F.*

Definition 4 *An implicate I_e of F is called prime if any subset $I'_e \subsetneq I_e$ is not an implicate of F.*

Given F, $PI_n(F)$ and $PI_e(F)$ denote, respectively, the sets of all prime implicants and prime implicates of F.

Approaches based on problem reformulation can be traced to original work on using ILP for computing prime implicants of CNF formulae [Palopoli *et al.*, 1999]. This paper follows the basic reformulation [Jabbour *et al.*, 2014] from a CNF formula F to a CNF formula H, defined on a different set of variables. This approach is summarized next.

Let F denote a CNF formula. Create a formula $H = L \cup C \cup B$ as follows. For each $v \in \mathsf{var}(F)$, the pair of variables $\{x_v, x_{\neg v}\} \in \mathsf{var}(H)$ encodes the non-occurence, the negative or the positive occurrence of v in a possible prime implicant of F, respectively $x_v = x_{\neg v} = 0$, $x_v = 0 \wedge x_{\neg v} = 1$, and $x_v = 1 \wedge x_{\neg v} = 0$. $L = \{(\neg x_v \vee \neg x_{\neg v}) \,|\, v \in \mathsf{var}(F)\}$. C is created from the clauses of F such that each clause $c_i = (l_1 \vee \ldots, l_k) \in F$ is transformed into a clause $c_i^H = (l_1^H \vee \ldots, l_k^H) \in C$, where each literal $l_j^H = x_v$ if $l_j = v$, and $l_j^H = x_{\neg v}$ if $l_j = \neg v$. The cost function of the ILP model is given by the sum of the variables of H, $\mathsf{var}(H) = \{x_v, x_{\neg v} \,|\, v \in \mathsf{var}(F)\}$, but it is not used in our approach. Finally, initially $B = \emptyset$, and is used to block any computed prime implicants.

Observe that L disallows the assignment $x_v = x_{\neg v} = 1$ for each pair of variables $\{x_v, x_{\neg v}\}$. Moreover, C encodes an *implicate cover* of F, using the new set of variables. Translating the clauses in C to the variables of F we get a formula that is equivalent to F. (In this case, the resulting formula is F itself.)

For non-clausal formulae, a number of algorithms have been proposed for computing a prime implicate given an implicate of a propositional formula [Bradley and Manna, 2007]. Among these, the one requiring the asymptotically fewest calls to a SAT solver corresponds to the QuickXplain algorithm [Junker, 2004; Bradley and Manna, 2007].

4 Non-Clausal Prime Compilation

The approach proposed in this paper is motivated by the reformulation approach for computing the sets of prime implicants of CNF formulae [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014], and outlined in the previous section. Recall that, starting from F, the reformulation approach creates a CNF formula H containing three components, $H = L_- \cup B_n \cup C_e$, where L_-

limits the number of assignments to 3^n (with $n = \mathsf{var}(F)$), C_e encodes F, and so can be viewed as encoding an implicate cover of F, and finally, B_n is built as the compilation progresses and blocks already computed prime implicants.

For the case of non-clausal formulae, we propose to start with C_e as the empty set, and iteratively refine both B_n and C_e. The approach used in this paper works on two different formulae, being similar to recent work on exploiting dualization for solving Max-SAT, model-based diagnosis and finding MUSes [Bailey and Stuckey, 2005; Davies and Bacchus, 2011; Stern *et al.*, 2012; Previti and Marques-Silva, 2013; Liffiton *et al.*, 2015]. At each iteration, a maximal model is extracted from a working formula H, which includes B_n, C_e as well as L_-. This model is used to extract either a prime implicant or a prime implicate of F, which is then either added to B_n, if a prime implicant was computed, or to C_e, if instead a prime implicate was computed. An essential aspect for the correctness of the approach is that, while clauses for blocking prime implicants use negated literals (similar to earlier work [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014]), clauses for blocking prime implicates use positive literals (similar to the encoding of the original CNF formula in earlier work [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014]). The algorithm iterates until it computes all prime implicants and a (prime) implicate cover of F. Observe that this prime implicate cover is essentially what C already represents when F is represented in CNF [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014].

It should be pointed out that prime implicants and implicates must be extracted from F, which is a non-clausal formula. Whereas extracting a prime implicant from a satisfying assigment is a polynomial time procedure for CNF formulae [Schrag, 1996] (and similarly for extracting implicates for DNF formulae), this is not the case for non-clausal formulae. A non-clausal formula F can be converted into a CNF F_{CNF} by means of a Tseitin encoding and tested for satisfiability. Unfortunately, a prime implicant (or implicate) for the encoded F_{CNF} is not a prime implicant (resp. implicate) for F. In order to extract a prime implicant (or implicate) for the original formula, we have to perform a sequence of queries on F_{CNF}. Each query checks for the necessity of a literal in the computed model. For example, suppose that m is a model for F_{CNF}. So we have that $m \vDash F_{CNF}$ and that $m \wedge \neg F_{CNF}$ is unsatisfiable. A prime implicant can be extracted from m by testing one literal at a time. At each step i, a new assignment m' is created by removing a literal $l \in m_i$ (with $m_0 = m$). If $m' \wedge \neg F_{CNF}$ is still unsatisfiable then l is not part of the prime implicant under construction and $m_{i+1} = m'$. Otherwise, it means that l is necessary and $m_{i+1} = m_i$. In this case, l is marked in order to avoid to test it again. However, the extraction of a prime implicant from a satisfying assignment can be achieved more efficiently using the QuickXplain algorithm [Junker, 2004; Bradley and Manna, 2007]. The two functions `ReduceImplicant` and `ReduceImplicate`, used in our algorithms, hide the details of the procedure just explained. As such, even if not mentioned explicitly, all the SAT calls are performed on CNF encoded formulae. The same applies to the call to the SAT oracle that is shown in the two algorithms.

The next section formalizes the approach outlined above and proves it correct for prime implicant (or implicate) compilation.

```
input  : Formula $F$
output: $PI_n(F)$ and prime implicate cover of $F$
1  $H \leftarrow \{(\neg x_v \vee \neg x_{\neg v}) \mid v \in \mathsf{var}(F)\}$
2  while true do
3      $(\mathsf{st}, A^H) \leftarrow \mathsf{MaxModel}(H)$
4      if not st then return
5      $A^F \leftarrow \mathsf{Map}(A^H)$
6      $\mathsf{st} \leftarrow \mathsf{SAT}(A^F \cup \neg F)$
7      if not st then            # $A^F \vDash F$; i.e. $A^F$ is an implicant
8          $I_n \leftarrow$ ReduceImplicant$(A^F, F)$
9          PrintImplicant$(I_n)$
10         $b \leftarrow \{\neg x_l \mid l \in I_n\}$
11     else                      # $F \vDash \neg A^F$; i.e. $\neg A^F$ is an implicate
12         $I_e \leftarrow$ ReduceImplicate$(A^F, F)$
13         PrintImplicate$(I_e)$
14         $b \leftarrow \{x_l \mid l \in I_e\}$
15     $H \leftarrow H \cup \{b\}$
```

Algorithm 1: Basic prime compilation

4.1 Basic Algorithm

This section describes the first approach for non-clausal prime compilation. As outlined in the previous section, during the algorithm's execution, H is organized in three components:

$$H = L_- \cup B_n \cup C_e \tag{1}$$

Observe that B_n denotes the set of clauses blocking prime implicants (and the literals are pure and negative), and C_e denotes the set of clauses blocking prime implicates (and the literals are pure and positive). Moreover, B_n^A is the set of blocking clauses for *all* prime implicants of F, C_e^T is the set of blocking clauses for a cover of F by prime implicates. Similar definitions apply for all prime implicates and a cover by prime implicants, respectively B_e^A and C_n^T. In addition, A^F represents an assignment on F and A^H the corresponding assignment on H.

Algorithm 1 summarizes the main steps of the proposed approach. At each step, a maximal model A^H of H is computed. The assignment A^F of F associated with A^H either satisfies or falsifies F (see Lemma 2). If A^F satisfies F, then a prime implicant is extracted and blocked by adding a new clause to B_n (see Theorem 1). Otherwise, if A^F falsifies F, then a prime implicate is extracted and blocked by adding a new clause to C_e (also see Theorem 1). Algorithm 1 terminates when all prime implicants of F have been computed and a prime implicate cover of F has also been computed (see Theorem 2).

The startup phase of Algorithm 1 consists of initializing H with $L_- = \{(\neg x_v \vee \neg x_{\neg v}) \mid v \in \mathsf{var}(F)\}$. For each $v \in \mathsf{var}(F)$, there exist two corresponding variables in H, x_v represents the literal v whereas $x_{\neg v}$ represents the literal $\neg v$. The reason for this translation is that now for each original variable v, there are four possible assignments:

1. $(x_v = 1$ and $x_{\neg v} = 0) \Rightarrow v = 1$
2. $(x_v = 0$ and $x_{\neg v} = 1) \Rightarrow v = 0$
3. $(x_v = 0$ and $x_{\neg v} = 0) \Rightarrow v$ is a don't care
4. $(x_v = 1$ and $x_{\neg v} = 1) \Rightarrow$ forbidden by L_-

This encoding is known as *dual rail encoding* [Bryant *et al.*, 1987; Roorda and Claessen, 2005] and is necessary for the

Table 1: Example run of Algorithm 1

A^H $x_a x_{\neg a} x_b x_{\neg b} x_c x_{\neg c}$	A^F	Entail.	B_n/C_e
100101	$A_1^F = a, \neg b, \neg c$	$F \models \neg A_1^F$	x_c
100110	$A_2^F = a, \neg b, c$	$A_2^F \models F$	$(\neg x_a \vee \neg x_c)$
010110	$A_3^F = \neg a, \neg b, c$	$F \models \neg A_3^F$	$(x_a \vee x_b)$
011010	$A_4^F = \neg a, b, c$	$A_4^F \models F$	$(\neg x_b \vee \neg x_c)$

computation of the complete set of prime implicants. Without this encoding, Algorithm 1 could return just a prime implicant cover.

Example 1 *Table 1 summarizes the execution of Algorithm 1 on $F = (((a \wedge b) \vee (a \wedge \neg b)) \wedge c) \vee (b \wedge c)$. In the first iteration, a maximal model of H is for example $A_1^H = \{x_a = 1, x_{\neg a} = 0, x_b = 0, x_{\neg b} = 1, x_c = 0, x_{\neg c} = 1\}$, which corresponds to the assignment in F, $A_1^F = \{a = 1, b = 0, c = 0\}$. By inspection, A_1^F falsifies F, and so $F \models \neg A_1^F$. As a result, the prime implicate (c) is extracted, and so the clause (x_c) is added to C_e. The second maximal model of H is $A_2^H = \{x_a = 1, x_{\neg a} = 0, x_b = 0, x_{\neg b} = 1, x_c = 1, x_{\neg c} = 0\}$, and so the corresponding $A_2^F = \{a = 1, b = 0, c = 1\}$. It is immediate that $A_2^F \models F$. As a result, the prime implicant $a \wedge c$ is extracted, and so the clause $(\neg x_a \vee \neg x_c)$ is added to B_n. In total, four maximal models are computed, resulting in two prime implicants and two prime implicates.*

Lemma 1 *If $\neg A^F$ is an implicate on F, then A^H satisfies all the clauses in B_n^A.*

Proof. Suppose that $P^F = (p_1 \vee \ldots \vee p_m)$ is a prime implicate on F and A^F the corresponding falsifying assignment $(\neg p_1 \wedge \ldots \wedge \neg p_m)$. Let $A^H = (x_{\neg p_1} \wedge \ldots \wedge x_{\neg p_m})$ be the corresponding assignment on H. Note that in H the prime implicate P^F is blocked with the clause $P^H = (x_{p_1} \vee \ldots \vee x_{p_m})$. Due to the duality between the set of prime implicants and prime implicates [Rymon, 1994][2], at least one literal in the set $\{\neg x_{p_1}, \ldots, \neg x_{p_m}\}$ is contained in every clause in B_n^A. Since we have that $x_{\neg p_1} = 1, \ldots, x_{\neg p_m} = 1$ (A^H) and we also have the clauses in L_-, this implies that $x_{p_1} = 0, \ldots, x_{p_m} = 0$. Thus all the clauses in B_n^A are satisfied. To prove that the statement is valid also when we extend P^F to an implicate, notice that the assignment for $x_{p_1}, \ldots, x_{p_m}$ remains unchanged. □

Lemma 2 *Every assignment A^F induced by a maximal model computed on H, either satisfies or falsifies F.*

Proof. It is easy to see that if an assignment is complete, it either satisfies or falsifies F. Suppose now that A^F is a partial assignment induced by a maximal model A^H computed on the formula H and suppose that: (i) A^F does not satisfy F; and (ii) A^F does not falsify F. We know that due to the way we block the prime implicates (i.e. by using clauses with positive literals), every induced assignment has to contain at least one literal for each prime implicate already computed. Suppose now to extend A^F in such a way as to falsify F (it is always possible, unless A^F is already an implicant). Let us call this assignment A_e^F. Note that $\neg A_e^F$ is an implicate that does not contain any previously computed prime implicates. We now want to show that the corresponding assignment A_e^H is a model for H. In particular we want to prove that A_e^H is a model for L_-, B_n and C_e, and so it is a model for H.

1. A_e^H is a model for L_- because A_e^F does not contain complementary literals.
2. A_e^H is a model for C_e because A^H already satisfies all the clauses in C_e and we have not flipped any of its variables assigned value 1 when we extended it to A_e^H.
3. Since A_e^F is an implicate, by Lemma 1, all the clauses in B_n are satisfied.

So we proved that A_e^H is a model for H. But since the set of '1' contained in A_e^H is a superset of those contained in A^H, this contradicts our hypothesis that A^H is a maximal model. □

A consequence of Lemma 2 is that if A^F satisfies F, then Algorithm 1 extracts a prime implicant. Otherwise, the algorithm extracts a prime implicate.

Theorem 1 *At each step of Algorithm 1, either a prime implicant $I_n \in PI_n(F)$ is computed, or a prime implicate $I_e \in PI_e(F)$ is computed.*

Proof. Lemma 2 already proved that every induced assignment A^F either satisfies or falsifies F. Note that A^F contains at least one literal from every prime implicate already computed and differs in at least one literal from every prime implicant already computed. This means that either a new prime implicate or a new prime implicant is extracted by Algorithm 1. □

Theorem 2 *Algorithm 1 computes $PI_n(F)$ and a prime implicate cover of F.*

Proof. By Theorem 1, we know that at each iteration, Algorithm 1 computes a new $I_n \in PI_n(F)$ or a new $I_e \in PI_e(F)$. We now want to prove that:

1. Algorithm 1 does not terminate before all the prime implicants of F have been computed;
2. Algorithm 1 does not terminate before a prime implicate cover of F has been computed;
3. Algorithm 1 terminates when all the prime implicants of F and a prime implicate cover of F are computed.

Claims 1 and 2 state that when a prime implicant or a prime implicate from a cover of F is missing, then H is satisfiable. Claim 3 guarantees that when we have computed all the $I_n \in PI_n(F)$ and a prime implicate cover of F, then H is unsatisfiable. We start by proving Claim 1. Suppose now, without loss of generality, that Algorithm 1 has already computed a prime implicate cover C of F but is missing a single prime implicant $P_1 = p_1, \ldots, p_m$. We want to show that H must be satisfiable, by showing the existence of a model $m = p_1, \ldots, p_m, \neg p_{m+1} \ldots, \neg p_n$ that satisfies the three parts of H (L_-, B_n, C_e). Due to the duality between $PI_n(F)$ and $PI_e(F)$, the set of literals of P_1 is guaranteed to hit all the prime implicates in the cover C (again, this follows from [Rymon, 1994]). So all the clauses in C_e are satisfied. Moreover, since for each literal in the original formula, we

[2]The minimal hitting set duality between prime implicants and implicates is a simple consequence of [Rymon, 1994, Theorem 2.3], but also from [Reiter, 1987].

```
input : Formula $F$
output: $PI_n(F)$ and prime implicate cover of $F$
1  $H \leftarrow \{(\neg x_v \vee \neg x_{\neg v}) \mid v \in \mathsf{var}(F)\}$
2  while true do
3      $(\mathsf{st}, A^H) \leftarrow \mathsf{MinModel}(H)$
4      if not st then return
5      $A^F \leftarrow \mathsf{Map}(A^H)$
6      $(\mathsf{st}, M^{\neg F}) \leftarrow \mathsf{SAT}(A^F \cup \neg F)$
7      if st then        # $F \vDash \neg M^{\neg F}$; i.e. $\neg M^{\neg F}$ is an implicate
8          $I_e \leftarrow$ ReduceImplicate($M^{\neg F}, F$)
9          PrintImplicate($I_e$)
10         $b \leftarrow \{x_l \mid l \in I_e\}$
11     else                    # $A^F \vDash F$; i.e. $A^F$ is an implicant
12         $I_n \leftarrow A^F$
13         PrintImplicant($I_n$)
14         $b \leftarrow \{\neg x_l \mid l \in I_n\}$
15     $H \leftarrow H \cup \{b\}$
```

Algorithm 2: Alternative prime compilation

have a corresponding variable on the reformulated formula, this means that on H, every pair of prime implicants (P_i, P_j) differs in at least one variable. This means that the set of unassigned variables $U_V = \mathsf{var}(H) \setminus \{p_1, \ldots, p_m\}$ (all literals are positive) contains at least one variable for each previously computed prime implicant. Therefore, we can set these variables to 0, thus satisfying all the clauses in B_n. We now want to prove that the set of clauses L_- is satisfied. This means that for each variable v, $(x_v = 1) \wedge (x_{\neg v} = 1)$ does not hold. This is easy to see, because if $x_v \in \{p_1, \ldots, p_m\}$ then $x_{\neg v} \in U_V$ and we set all the variables in U_V to 0.

We are now going to prove that Claim 3 also holds. This means that when $B_n = B_n^A$ and C_e represents a prime implicate cover of F, then H is unsatisfiable. In order to prove this, we want to show that every model satisfying $L_- \cup C_e$ falsifies at least one clause in B_n. Note that since set C_e encodes a prime implicate cover of F, it corresponds to F', equivalent to F. So, the reformulation encoding for F' (without blocked implicants) is the formula $L_- \cup C_e$. (Observe that this corresponds to the standard reformulation encoding [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014] for prime implicant compilation.) In fact every minimal model on this formula corresponds to a prime implicant. It is easy to see that every model corresponds to an implicant. But being B_n the set of blocking clauses for the set of all prime implicants of F, this means that every model of $L_- \cup C_e$, falsifies at least one clause on B_n.

Finally, the proof of Claim 2 is similar to the one of Claim 1 and is omitted due to lack of space. □

4.2 Alternative Algorithm

This section describes an alternative approach whose main characteristic is to avoid the reduction of computed implicants. Algorithm 2 summarizes this approach. (Observe that this algorithm can be related with the original Dualize and Advance algorithm [Bailey and Stuckey, 2005].) At each step a partial assignment A^F is returned. If A^F satisfies the formula, it is guaranteed to be a prime implicant. Otherwise, A^F is extended to a complete model $M^{\neg F}$ on $\neg F$ and then reduced to a prime implicate. The key idea is to replace the computation of a maximal model with the one of a *minimal* model. In this way, A^F is guaranteed to hit the set of prime implicates computed so far using a minimal number of literals. When we have a prime implicate cover, this obviously corresponds to a prime implicant [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014]. Otherwise it represents a subset of a prime implicant, and as a consequence it does not necessarily imply F. This means that the call $\mathsf{SAT}(A^F \cup \neg F)$ could return a model $M^{\neg F}$ on $\neg F$. As a result, $\neg M^{\neg F}$ is an implicate of F, that we can reduce to a prime implicate. If instead $(A^F \cup \neg F)$ is unsatisfiable, this means that $A^F \vDash F$ and, since A^F hits the set of prime implicates using a minimal number of literals, A^F has to be a prime implicant.

Table 2: Example run of Algorithm 2

A^H $x_a x_{\neg a} x_b x_{\neg b} x_c x_{\neg c}$	A^F	$\neg M^{\neg F}/\neg\mathsf{st}$	B_n/C_e
000000	$A_1^F = \emptyset$	$\neg a, \neg b, \neg c$	$(x_a \vee x_b)$
001000	$A_2^F = b$	$\neg a, b, \neg c$	x_c
001010	$A_3^F = b, c$	$\neg\mathsf{st}$	$(\neg x_b \vee \neg x_c)$
100010	$A_4^F = a, c$	$\neg\mathsf{st}$	$(\neg x_a \vee \neg x_c)$

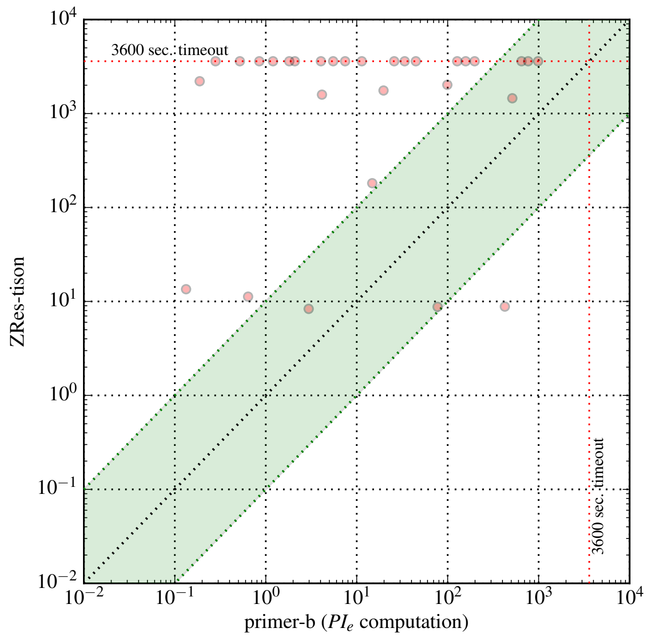
Example 2 *Table 2 summarizes the execution of Algorithm 2 on $F = (((a \wedge b) \vee (a \wedge \neg b)) \wedge c) \vee (b \wedge c)$. The first minimal model leaves A^F fully unspecified. The SAT solver picks an assignment, that falsifies F, from which a prime implicate is extracted. The second minimal model operates similarly, and another prime implicate is extracted. The third minimal model cannot falsify F, and so represents a prime implicant $b \wedge c$. The fourth minimal model also yields a prime implicant. Afterwards, the formula becomes unsatisfiable.*

A final observation is that the two algorithms proposed in this paper can be adapted for computing $PI_e(F)$ and a prime implicant cover of F, by exploiting duality.

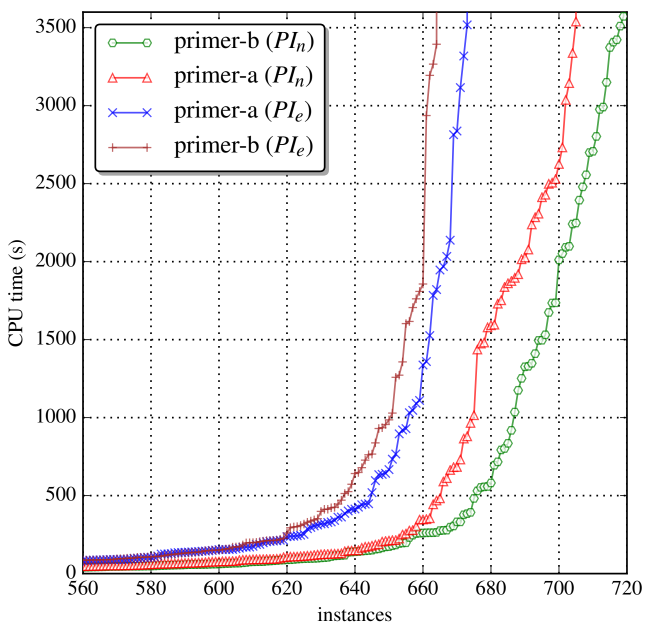
5 Preliminary Results

This section evaluates the two algorithms proposed in this paper. The experiments were performed on an Intel Xeon E5-2630 2.60GHz, with 64GByte of memory, and running Ubuntu Linux. The time limit was set to 3600s and the memory limit to 10GByte. Algorithms 1 and 2 were implemented on top of MiniSAT[3] [Eén and Sörensson, 2003] in a prototype called *primer* (*PRIMe compilER*). The versions of primer corresponding to Algorithm 1 and Algorithm 2 are referred to as *primer-a* and *primer-b*, respectively. Their performance was compared to the state-of-the-art approach to formula compilation, which is based on the well-known Tison method for prime implicate enumeration but using ZBDDs [Simon and del Val, 2001]. In the performed experimental evaluation the corresponding software tool is referred to as ZRes-tison. Other alternatives [Ramesh *et al.*, 1997; Matusiewicz *et al.*, 2009] are known not to scale for formula sizes considered in this section. Note that although ZRes-tison requires input formulae in CNF, it is able to get rid of all auxiliary variables (introduced by translation to CNF) once they are specified explicitly. In the performed evaluation, ZRes-tison was provided with CNF formulae containing additional information about all such variables. In order to

[3]Available from https://github.com/niklasso/minisat.



(a) primer-b (PI_e) vs. ZRes-tison



(b) PI_n and PI_e computation with primer

Figure 1: Performance comparison

assess the efficiency of the new algorithms, the following sets of non-clausal instances were considered.

Quasigroup classification problems. This set of benchmarks called QG6 was proposed in [Meier and Sorge, 2005] when encoding classification theorems for quasigroups. Out of 256 non-clausal formulae we chose 83 that are satisfiable.

Cryptanalysis of the Geffe stream generator. Originally proposed in [Geffe, 1973], the Geffe stream generator is a combination of 3 LFSRs (linear feedback shift register). One of them controls which of the other 2 LFSRs is connected to the output at each moment of time. For our experimental evaluation we constructed 3 Geffe generators of the total input size (i.e. sum of the length of all LFSRs) 32, 64, and 96 bits, respectively. The corresponding Boolean formulae were constructed with the use of the Transalg system[4], which is able to translate algorithms into Boolean formulae [Otpuschennikov *et al.*, 2014]. Given the constructed formulae and some fixed output sequence of the generator, the problem is to find an input sequence that produces the output. Note that depending on the output sequence's length, there can be many input sequences producing the same output. For each input size considered in the evaluation, 200 different output sequences were generated (600 instances in total).

Crafted formulae. This set comprises *crafted* non-clausal formulae with a manageable number of prime implicants and implicates, which is a generalization of the construction proposed in [Bordeaux and Marques-Silva, 2012]. Consider two kinds of formula parameterized by m and n: $F_m \vee PHP_n$ and $F_m \vee GT_n$, where PHP_n are well-known *pigeon-hole principle* formulae, and GT_n are based on the ordering principle that any partial order on a finite set must have a maximal element [Beame *et al.*, 2004]. Formulae F_m are of the form $(x_1 \vee y_1) \wedge \cdots \wedge (x_m \vee y_m)$ and have 2^m prime implicants and m prime implicates. The experiments consider m ranging from 10 to 20 while n ranges from 6 to 10 for PHP_n, and from 12 to 20 for GT_n. In total, 30 instances were constructed with the PHP_n subformulae, and 30 instances with GT_n.

Table 3: Number of solved instances

	QG6	Geffe gen.	F+PHP	F+GT	Total
ZRes-tison	0	0	11	0	11
primer-a (PI_n)	53	**596**	**30**	26	705
primer-a (PI_e)	28	588	**30**	27	673
primer-b (PI_n)	**64**	595	**30**	**30**	**719**
primer-b (PI_e)	30	577	**30**	27	664

Approaches similar to [Coudert and Madre, 1992] were not tried, since these require building a BDD for a propositional formula and, with the exception of $F_m \vee PHP_n$ (which are easy for CUDD[5]), CUDD is unable to construct a BDD for any of the remaining formulae within 1 hour. Table 3 shows the number of solved instances for each of the considered approaches. Note that given a 1 hour timeout, ZRes-tison cannot compile any formulae from almost all the considered benchmark sets with an exception being crafted $F_m \vee PHP_n$ where it can solve 11 formulae. Thus, a more detailed comparison of ZRes-tison and primer-b (both set to compute all the prime implicates PI_e) is shown in Figure 1a. As can be observed, Table 3 indicates that both primer-a and primer-b can successfully deal with the considered benchmarks. Observe that formulae with hundreds of thousands of prime implicants/implicates are not an obstacle for the new algorithms (e.g. see the results for $F_m \vee PHP_n$ and $F_m \vee GT_n$ formulae, where the number of prime implicants varies from 2^{10} to 2^{20}).

As for the comparison between primer-a and primer-b and

[4]The source code of the Transalg system is available at https://www.gitorious.org/transalg.

[5]CUDD BDD-package can be downloaded from http://vlsi.colorado.edu/ fabio/CUDD/.

according to Table 3, one can readily conclude that computing all prime implicants PI_n is generally more efficient for the Geffe generator and QG6 benchmarks than computing PI_e. This can be explained by $|PI_n|$ being significantly smaller than $|PI_e|$ for these benchmarks. However, this is not the case for $F_m \vee PHP_n$ and $F_m \vee GT_n$ instances. Finally and as shown in Figure 1b, primer-b set to compute PI_n consistently outperforms primer-a, thus being in general expected to find more prime implicants/implicates within a given time limit.

6 Conclusions

This paper proposes two novel approaches for prime compilation of non-clausal formulae based on iterative SAT solving. Both approaches exploit the reformulation approach for compiling the prime implicants of CNF formulae [Palopoli *et al.*, 1999; Jabbour *et al.*, 2014], but integrate reformulation with recent algorithms that exploit dualization [Bailey and Stuckey, 2005; Davies and Bacchus, 2011; Stern *et al.*, 2012; Previti and Marques-Silva, 2013; Liffiton *et al.*, 2015], in our concrete case between prime implicants and prime implicates [Rymon, 1994]. The experimental results indicate that the new prime compilation approaches are more efficient and scale better than alternative approaches for different classes of formulae studied in the paper, namely classification theorems for quasigroups, cryptographic benchmark formulae and specific crafted formulae providing an efficient alternative for formulae with a large number of variables, but without large numbers of prime implicants and implicates. Moreover, the wide range of applications of prime implicants and implicates motivates the practical deployment of this work, allowing also a more comprehensive evaluation of the proposed algorithms.

7 Acknowledgements

This work is partially supported by SFI PI grant BEACON (09/IN.1/ I2618), FCT grant POLARIS (PTDC/EIA-CCO/123051/2010) and national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[Bailey and Stuckey, 2005] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *PADL*, pages 174–186, 2005.

[Beame *et al.*, 2004] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *JAIR*, 22:319–351, 2004.

[Bienvenu, 2009] Meghyn Bienvenu. Prime implicates and prime implicants: From propositional to modal logic. *JAIR*, 36:71–128, 2009.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, 2009.

[Blake, 1937] Archie Blake. *Canonical expressions in Boolean algebra*. PhD thesis, Univ. Chicago, 1937.

[Bordeaux and Marques-Silva, 2012] Lucas Bordeaux and João Marques-Silva. Knowledge compilation with empowerment. In *SOFSEM*, pages 612–624, 2012.

[Bradley and Manna, 2007] Aaron R. Bradley and Zohar Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, pages 173–180, 2007.

[Brauer *et al.*, 2011] Jörg Brauer, Andy King, and Jael Kriener. Existential quantification as incremental SAT. In *CAV*, pages 191–207, 2011.

[Bryant *et al.*, 1987] Randal E. Bryant, Derek L. Beatty, Karl S. Brace, K. Cho, and Thomas J. Sheffler. COSMOS: A compiled simulator for MOS circuits. In *DAC*, pages 9–16, 1987.

[Cadoli and Donini, 1997] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Commun.*, 10(3-4):137–150, 1997.

[Castell, 1996] Thierry Castell. Computation of prime implicates and prime implicants by a variant of the Davis and Putnam procedure. In *ICTAI*, pages 428–429, 1996.

[Coudert and Madre, 1992] Olivier Coudert and Jean Christophe Madre. Implicit and incremental computation of primes and essential primes of boolean functions. In *DAC*, pages 36–39, 1992.

[Czarnecki and Wasowski, 2007] Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *SPLC*, pages 23–34, 2007.

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *CP*, pages 225–239, 2011.

[de Kleer, 1992] Johan de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI*, pages 780–785, 1992.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[Geffe, 1973] P. Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, pages 99–101, Jan. 4 1973.

[Jabbour *et al.*, 2014] S. Jabbour, J. Marques Silva, L. Sais, and Y. Salhi. Enumerating prime implicants of propositional formulae in conjunctive normal form. In *JELIA*, 2014.

[Jackson, 1992] Peter Jackson. Computing prime implicates incrementally. In *CADE*, pages 253–267, 1992.

[Junker, 2004] Ulrich Junker. QuickXplain: Preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172, 2004.

[Kean and Tsiknis, 1990] Alex Kean and George K. Tsiknis. An incremental method for generating prime implicants/impicates. *J. Symb. Comput.*, 9(2):185–206, 1990.

[Liffiton *et al.*, 2015] Mark H Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible mus enumeration. *Constraints*, pages 1–28, 2015.

[Marquis, 1995] Pierre Marquis. Knowledge compilation using theory prime implicates. In *IJCAI*, pages 837–845, 1995.

[Marquis, 2000] Pierre Marquis. Consequence finding algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 41–145. 2000.

[Matusiewicz *et al.*, 2009] Andrew Matusiewicz, Neil V. Murray, and Erik Rosenthal. Prime implicate tries. In *Tableaux*, pages 250–264, 2009.

[McCluskey, 1956] Edward J McCluskey. Minimization of boolean functions. *Bell Sys. Tech. J.*, 35(6):1417–1444, 1956.

[Meier and Sorge, 2005] Andreas Meier and Volker Sorge. A new set of algebraic benchmark problems for SAT solvers. In *SAT*, pages 459–466, 2005.

[Ngair, 1993] Teow-Hin Ngair. A new algorithm for incremental prime implicate generation. In *IJCAI*, pages 46–51, 1993.

[Otpuschennikov *et al.*, 2014] Ilya Otpuschennikov, Alexander Semenov, and Stepan Kochemazov. Transalg: a tool for translating procedural descriptions of discrete functions to SAT (tool paper). *CoRR*, abs/1405.1544, 2014.

[Palopoli *et al.*, 1999] Luigi Palopoli, Fiora Pirri, and Clara Pizzuti. Algorithms for selective enumeration of prime implicants. *Artif. Intell.*, 111(1-2):41–72, 1999.

[Previti and Marques-Silva, 2013] Alessandro Previti and Joao Marques-Silva. Partial MUS enumeration. In *AAAI*, 2013.

[Quine, 1952] Willard Quine. The problem of simplifying truth functions. *Amer. Math. Month.*, 59(8):521–531, 1952.

[Quine, 1959] Willard Quine. On cores and prime implicants of truth functions. *Amer. Math. Month.*, 66(9):755–760, 1959.

[Ramesh *et al.*, 1997] Anavai Ramesh, George Becker, and Neil V. Murray. CNF and DNF considered harmful for computing prime implicants/implicates. *J. Autom. Reasoning*, 18(3):337–356, 1997.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[Roorda and Claessen, 2005] Jan-Willem Roorda and Koen Claessen. A new sat-based algorithm for symbolic trajectory evaluation. In *CHARME'05*, pages 238–253. Springer, 2005.

[Rymon, 1994] Ron Rymon. An SE-tree-based prime implicant generation algorithm. *Ann. Math. Artif. Intell.*, 11(1-4):351–366, 1994.

[Schlobach *et al.*, 2007] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *J. Autom. Reason.*, 39(3):317–349, 2007.

[Schrag, 1996] Robert Schrag. Compilation for critically constrained knowledge bases. In *AAAI*, pages 510–515, 1996.

[Shannon, 1949] Claude Shannon. The synthesis of two-terminal switching circuits. *Bell Sys. Tech. J.*, 28(1):59–98, 1949.

[Simon and del Val, 2001] Laurent Simon and Alvaro del Val. Efficient consequence finding. In *IJCAI*, pages 359–370, 2001.

[Slagle *et al.*, 1970] James R Slagle, Chin-Liang Chang, and Richard CT Lee. A new algorithm for generating prime implicants. *IEEE Trans. Computers*, 100(4):304–310, 1970.

[Stern *et al.*, 2012] Roni Tzvi Stern, Meir Kalech, Alexander Feldman, and Gregory M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI*, 2012.

[Stuckey, 2013] Peter J. Stuckey. There are no CNF problems. In *SAT*, pages 19–21, 2013.

[Tison, 1967] Pierre Tison. Generalization of consensus theory and application to the minimization of boolean functions. *IEEE Trans. Elect. Computers*, (4):446–456, 1967.

[To *et al.*, 2011] Son Thanh To, Tran Cao Son, and Enrico Pontelli. Conjunctive representations in contingent planning: Prime implicates versus minimal CNF formula. In *AAAI*, 2011.