Prime Implicant Enumeration via QBF Solvers

Noel Arteche and Montserrat Hermo

University of the Basque Country, Faculty of Computer Science
Manuel Lardizabal 1, 20018 Donostia / San Sebastián, Spain
{noel.arteche, montserrat.hermo}@ehu.eus

Abstract. We present a simple QBF encoding that can be used together with incremental calls to a QBF solver to enumerate all the prime implicants of a propositional formula. This is work-in-progress on the use of quantified Boolean formula solvers to compute prime implicants and Boolean formula minimization, problems where traditionally only SAT-based algorithms have been used.

Keywords: Prime implicant · Formula minimization · QBF

1 Introduction

Boolean formula minimization is a well-known problem of big practical applicability, well-researched since the 1950s. Complexity-wise, today we know that the decisional version of DNF-minimization is $\mathbf{\Sigma}_2^p$-complete [6], while the related problem of deciding whether a certain partial assignment is a prime implicant was shown to be **DP**-complete [3].

The existing algorithmic solutions, at least to the extent of our knowledge, tend to rely on SAT solvers. Today, overcoming the limitations of the classical Quine-McCluskey algorithm [5], recent approaches like those of Ignatiev et al. [4] show that it is quite efficient in practice to solve formula simplification and minimization using SAT solvers for prime implicants/implicate enumeration.

Despite the practical efficiency of these methods, given that the problems are conjectured to be strictly beyond **NP**, we believe that using QBF solvers seems a natural, elegant and straightforward solution to find prime implicants and minimize formulas.

In this paper, we present incipient work-in-progress directed at an attempt to compute prime implicants using QBF solvers. Section 2 presents a simple QBF encoding that captures the notion of prime implicant for a given propositional formula. In particular, we first present an encoding close to the **DP**-completeness of the problem, while Appendix A discusses a formula that is linearly smaller, but using a harder quantifier prefix. Section 3 discusses how to use this encoding together with incremental calls to a QBF solver to enumerate all prime implicants of a given formula. Besides, we discuss a possible variation of this encoding, as well as its limitations. We conclude with a brief outline of the work that is yet to be conducted.

2 Encoding Prime Implicants into QBF

In what follows, given a Boolean formula φ over variables $x_1, \ldots, x_n$, we think of a (partial) assignment α directly in the binary representation we later use. That is, $\alpha = (\overline{u}, \overline{a}) \in \{0,1\}^n \times \{0,1\}^n$, such that bit $u_i = 1$ if and only if variable x_i is left unassigned; else, we read in a_i the value assigned to variable x_i.

Definition 1 (X-restrictions and extensions). *Let φ be a Boolean formula over variables $x_1, \ldots, x_n$, let $\alpha = (\overline{u}, \overline{a})$ be a partial assignment to φ, and let $X \subseteq \{x_i \mid u_i = 0\}$. We say an assignment $\alpha' = (\overline{u}', \overline{a})$ is an X-*restriction *of α whenever $u'_i = 1$ if $x_i \in X$ and $u'_i = u_i$ otherwise. If $X = \{x_i\}$, we simply say that α' is an x_i-*restriction. *An* extension *of α is an assignment $\alpha' = (0^n, \overline{a}')$ such that $a'_i = a_i$ whenever $u_i = 0$.*

Definition 2 (Implicants and prime implicants). *We say that α is an* implicant *of φ if every extension of α makes φ true. An implicant α is* prime *if for every X-restriction α' of α, α' is not an implicant.*

For convenience, we often express a partial assignment $\alpha = (\overline{u}, \overline{a})$ as a term $\alpha_t = \bigwedge_{\substack{u_i=0 \\ a_i=1}} x_i \wedge \bigwedge_{\substack{u_i=0 \\ a_i=0}} \neg x_i$.

Example 1. Consider the formula $\varphi(x, y, z) = (\neg x \vee y) \wedge (y \vee z)$. In this case, $\neg x \wedge y$ is an implicant, for every extension of this partial assignment makes the formula true. However, it is not prime, for the x-restriction y is an implicant. The prime implicants of φ are in this case y and $\neg x \wedge z$.

Crucially, to identify prime implicants it suffices to restrict attention to x-restrictions.

Proposition 1. *An implicant α is prime if and only if no x-restriction of α is an implicant.*

Proof. The forward direction is immediate. For the converse, if α is not prime, there is an X-restriction α' that is an implicant. Fix an $x \in X$. The x-restriction of α is an implicant, for otherwise α' would not be an implicant in the first place. $\square$

2.1 Encoding Implicants

Deciding whether a given partial assignment $\alpha = (\overline{u}, \overline{a})$ is an implicant of φ is **coNP**-complete (see Theorem 3.7 in [2]). Accordingly, we can encode this property into a universal statement.

As discussed, u_i is intended to encode whether variable x_i is being left unassigned. If $u_i = 0$, then x_i takes the value stored in a_i; otherwise, variable x_i should take any value, and so we set it to a new variable $\star_i$, universally quantified. For every $i \in [n]$, we define a new variable v_i as $v_i := (u_i \rightarrow \star_i) \wedge (\neg u_i \rightarrow a_i)$ encoding precisely whether we read the value in a_i or not. Then, the QBF

$$\mathrm{Imp}(\overline{u}, \overline{a}) = \forall \star_1 \ldots \forall \star_n : \varphi(v_1, \ldots, v_n)$$

is true if and only if $(\overline{u}, \overline{a})$ is an implicant of φ.

2.2 Forcing Implicants to Be Prime

We can now encode what it means for an implicant to be prime. This is, as noted in Proposition 1, that no x-restriction is an implicant.

Checking this property is in **NP**. The certificate is a sequence of n variable assignments, one for every variable, such that if x_i was assigned a value according to α, then the assignment $\overline{b}_i = b_{i,1}, \dots, b_{i,n}$ is an extension of α that makes φ false. Intuitively, what we encode is that for every variable x_i, dropping x_i from α is not possible, because we can find an assignment that falsifies the formula.

Therefore, given a formula φ and an implicant $\alpha = (\overline{u}, \overline{a})$, we claim that α is prime if and only if the following formula is true:

$$\mathrm{Prime}(\overline{u}, \overline{a}) = \exists b_{1,1} \dots b_{1,n} \dots \exists b_{n,1} \dots b_{n,n} : \bigwedge_{i=1}^{n} \neg u_i \to \neg\varphi(w_{i,1}, \dots, w_{i,n})$$

where $w_{i,j} := (u_j \to b_{i,j}) \wedge (\neg u_j \to a_j)$ if $i \neq j$ and $w_{i,i} := b_{i,i}$.

Combining this with the formula form the previous section we get that

$$\varPi(\overline{u}, \overline{a}) := \mathrm{Imp}(\overline{u}, \overline{a}) \wedge \mathrm{Prime}(\overline{u}, \overline{a})$$

encodes the property of being a prime implicant.

Remark 1. Note that $\varPi$ is the conjunction of a **coNP** predicate (Imp) and an **NP** predicate (Prime). This corresponds tightly to the prime implicant recognition problem being **DP**-complete. However, if we allow ourselves more expressive power in the quantifier prefix, we can rewrite $\varPi$ as a $\mathbf{\Pi}_2^p$ predicate that is linearly smaller in n. This alternative encoding is discussed in Appendix A.

3 Prime Implicant Enumeration

Based on the encoding presented above, we can obtain all the prime implicants of a formula by existentially quantifying $\varPi$ and incrementally calling a QBF solver that can extract a certificate for the existential variables at the front. This is summarised in Algorithm 1.

Algorithm 1 Enumeration of the prime implicants of φ

```
$\varPi \leftarrow \exists\overline{u}\exists\overline{a} : \mathrm{Imp}(\overline{u},\overline{a}) \wedge \mathrm{Prime}(\overline{u},\overline{a})$
primes $\leftarrow \emptyset$
while QBF-SOLVER($\varPi$) = "SAT" do
    $\alpha \leftarrow$ EXTRACT-CERT($\varPi$)
    primes $\leftarrow$ primes $\cup \{\alpha\}$
    $\varPi \leftarrow \exists\overline{u}\exists\overline{a} : \bigwedge_{\alpha\in\mathrm{primes}} \neg\alpha_t \wedge \mathrm{Imp}(\overline{u},\overline{a}) \wedge \mathrm{Prime}(\overline{u},\overline{a})$
end while
return primes
```

Note, however, that in most applications we are not interested in computing all prime implicants, but rather a set of non-redundant ones. Consider the following example.

Example 2. The formula $\psi(x, y, z) = (x \wedge \neg y) \vee (y \wedge z)$ has three prime implicants: $x \wedge \neg y$, $y \wedge z$ and $x \wedge z$. Note, however, that the third one is redundant, for the formula was already expressed using only the first two.

The QBF Π can be slightly modified, such that the enumeration algorithm stops as soon as the current set of prime implicants covers the entire formula. That is, given a partial set $\{\alpha_1, \ldots, \alpha_p\}$ of prime implicants computed so far, in every iteration we can update the body of Π with the additional constraint $\neg(\varphi(\star_1, \ldots, \star_n) \leftrightarrow \bigvee_{i=1}^{p} \alpha_i(\star_1, \ldots, \star_n))$, since the $\star$ variables are universally quantified. Nevertheless, this solution is not fully satisfactory, for its efficiency depends on the seemingly arbitrary order in which the QBF solver finds the prime implicants.

4 Conclusions and Future Work

The present work was developed in a context in which quickly computing a set of prime implicants was needed for a practical implementation. Unfortunately, the empirical results so far are discouraging and showcase the disappointing situation that QBF solvers are still not ready to compete against SAT-based solutions —and, for that matter, not even against less optimized procedures.

At the time of writing, we have conducted small scale tests of the encodings presented here using both the circuit-based solver QuAbS[1] as well as the PCNF solver DepQBF[2], after normalization to this format. In both cases, the QBF procedure took considerably longer than the SAT-based analogues. In particular, we compared the performance of our algorithm to Bica[3], the implementation of the SAT-based formula minimization algorithm described in [4], and against a more naif procedure using the Z3 SMT solver[4], where we first compute all models of a formula and then iterate over it to keep the prime implicants only.

Though the results obtained so far point at the SAT-based alternatives as the clear winners, we intend to keep investigating this approach further. In particular, we are interested in how performance compares between the encoding from Section 2 and the one in Appendix A. We expect to be able to show some meaningful empirical results in the near future.

All in all, even if the current state of QBF solving tools does not allow for practical use of this approach, we believe the encoding presented here is a paradigmatic example of how QBF solvers could be useful. A fast solving tool would be able to replace the multiple SAT solver calls needed by modern formula simplification algorithms in a way that is conceptually simpler and more intuitive.

[1] `https://github.com/ltentrup/quabs`
[2] `https://lonsing.github.io/depqbf/`
[3] `https://alexeyignatiev.github.io/software/bica/`
[4] `https://github.com/Z3Prover/z3`

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
2. Biere, A., Heule, M., van Maaren, H.: Handbook of Satisfiability, vol. 185. IOS press (2009)
3. Goldsmith, J., Hagen, M., Mundhenk, M.: Complexity of DNF minimization and isomorphism testing for monotone formulas. Information and Computation **206**(6), 760–775 (2008). https://doi.org/10.1016/j.ic.2008.03.002
4. Ignatiev, A., Previti, A., Marques-Silva, J.: SAT-based formula simplification. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 287–298. Springer (2015)
5. Quine, W.V.: The problem of simplifying truth functions. The American Mathematical Monthly **59**(8), 521–531 (1952). https://doi.org/10.1080/00029890.1952.11988183
6. Umans, C.: The minimum equivalent DNF problem and shortest implicants. Journal of Computer and System Sciences **63**(4), 597–611 (2001). https://doi.org/https://doi.org/10.1006/jcss.2001.1775

A Alternative Encoding

The formula Prime$(\overline{u},\overline{a})$ presented in Section 2.2 is quantified with the prefix

$$\exists b_{1,1}\dots b_{1,n}\dots\exists b_{n,1}\dots b_{n,n}$$

of size $\Theta(n^2)$. Furthermore, the body of the formula needs to copy a modified version of φ a total of n times.

Though this encoding corresponds tightly to the **DP**-completeness of the problem, we can obtain an encoding that is linearly smaller if we allow the use of universal quantifiers in the prefix. The idea is to recover the original definition of prime implicant in terms of X-restrictions. That is, we will encode that an implicant α is prime is *for every* X-restriction α', α' is not an implicant, because it can be extended into an assignment $\overline{b}$ that makes φ false.

Given a formula φ and an implicant $\alpha=(\overline{u},\overline{a})$, we claim that α is prime if and only if the following formula is true:

$$\mathrm{AltPrime}(\overline{u},\overline{a}) = \forall u'_1\dots\forall u'_n\exists b_1\dots\exists b_n : \left(\underbrace{\bigwedge_{i=1}^{n}(u_i\to u'_i)}_{(i)}\wedge\underbrace{\bigvee_{i=1}^{n}(\neg u_i\wedge u'_i)}_{(ii)}\right)\to\neg\varphi(w_1,\dots,w_n)$$

where $w_i := (u'_i\to b_i)\wedge(\neg u'_i\to a_i)$ for every $i\in[n]$.

Note that the conjunction (i) imposes that $\overline{u}'$ *agrees* with $\overline{u}$ on the variables that are left unassigned (i.e., if x_i was unassigned in α, it must remain unassigned in α'). On the other hand, (ii) ensures that α' *disagrees* with α in at least one point (i.e., α' is a restriction of α, in that there must be at least one variable x_i that α assigned but α' did not). Finally, the consequent makes sure that the restriction $\overline{u}'$ can be extended into an assignment that falsifies φ.

Indeed, AltPrime($\overline{u}, \overline{a}$) is smaller, both in the number of variables and in the size of the matrix. At the cost of introducing universal quantifiers, the prefix has now size $\Theta(n)$ and the matrix only contains φ once instead of n times. In total, AltPrime is linearly smaller than Prime.