Existential Quantification as Incremental SAT

Jörg Brauer[1], Andy King[2,3] and Jael Kriener[3]

[1] Embedded Software Laboratory, RWTH Aachen University, Germany
[2] Portcullis Computer Security, Pinner, UK
[3] School of Computing, University of Kent, UK

Abstract. This paper presents an elegant algorithm for existential quantifier elimination using incremental SAT solving. This approach contrasts with existing techniques in that it is based solely on manipulating the SAT instance rather than requiring any reengineering of the SAT solver or needing an auxiliary data-structure such as a BDD. The algorithm combines model enumeration with the generation of shortest prime implicants so as to converge onto a quantifier-free formula presented in CNF. We apply the technique to a number of hardware circuits and transfer functions to demonstrate the effectiveness of the method.

1 Introduction

Elegant ideas and careful engineering have advanced DPLL-based SAT solvers to the point they can rapidly decide the satisfiability of structured problems that involve thousands of variables. SAT has thus been applied to the problem of existential quantifier elimination, yet existing algorithms are considered "inelegant" [9, slide 24]. This paper is concerned with computing a quantifier-free formula $\exists X : \varphi$ in CNF where X is a set of propositional variables and φ is itself presented in CNF. The quadratic nature of resolution renders it impractical when X is large compared to $\mathit{vars}(\varphi)$ and SAT-based techniques are favoured when the set of variables $Y = (\mathit{vars}(\varphi) \setminus X)$ is small compared to $\mathit{vars}(\varphi)$. These techniques apply a SAT solver to find a cube $c_1 = (\bigwedge Y_1) \wedge \neg(\bigvee Y_2)$ for which $\varphi \wedge c_1$ is satisfiable and Y_1 and Y_2 partition Y. The clause $\neg c_1$ is then added to φ and the process is repeated to enumerate all such cubes $C = \{c_1, \ldots, c_\ell\}$. During enumeration, these cubes are typically stored in a BDD which converges onto $\bigvee C$. A CNF representation of $\bigvee C$ can then be extracted from the BDD, for example, by following all paths (cubes) c which lead to 0 and then negating to obtain a clause $\neg c$. McMillan [29] critiqued this approach pointing out that:

> "CNF and SAT-based quantifier elimination can be exponentially more efficient than [..] BDDs in cases where the resulting fixed points have compact representations in CNF, but not as BDDs."

BDDs have been used to store the cubes as it is believed that they offer a space-efficient way of storing the image (quantifier-free formula). However, this does not preclude computing CNF directly, especially if the size of the CNF is smaller than that extracted from the BDD. This paper can be considered a response to the agenda set by McMillan and presents an efficient, and we believe elegant, method for computing the image as a compact CNF formula that does not require modification to a solver.

The quest for elegance is more than an exercise in aesthetics since existential quantifier elimination finds application in: unbounded model checking [29], dependency analysis [2], information flow analysis [20], transfer function synthesis [5] and the synthesis of ranking functions [14]. It also occurs in predicate abstraction [21] from which we take an example [9] that we develop in what follows. In predicate abstraction, a finite set of predicates is used to express properties of and relationships between program variables at different points in the program. State can then be described by a cube over the predicate symbols, and a set of states as a Boolean function. Quantifier elimination arises when computing successor states. Adapting an example from [9], suppose the predicates $X = \{x_1, \ldots, x_6\}$ and $Y = \{y_1, \ldots, y_6\}$ express state at two consecutive program points, and the transition relation between these states is expressed as a Boolean function:

$$\begin{aligned}\mu = \neg(x_2 \wedge y_2) \wedge \neg(y_2 \wedge y_1) \wedge ((x_4 \wedge x_6) \Rightarrow y_1) \wedge \\ (x_3 \Leftrightarrow y_4) \wedge (x_4 \Leftrightarrow y_3) \wedge \quad (x_5 \Leftrightarrow y_6) \wedge (x_6 \Leftrightarrow y_5)\end{aligned}$$

If $\xi = (x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4 \wedge x_5 \wedge \neg x_6) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5 \wedge x_6)$ describes the state at one point, then the state at the next is given by $\exists X : (\xi \wedge \mu)$.

To summarise our work, the paper contributes an algorithm for existential quantifier elimination based solely on SAT solving. The elimination problem is reduced to that of discovering a cube of size k which entails the formula; a problem that can be completely encoded as a SAT instance. This formulation finesses the need for a complicated DPLL-like algorithm based on internal implication graphs and the application of heuristics [29, Sect. 2]. Furthermore, with a BDD-based approach, the size of resulting CNF formula is very sensitive to the variable ordering (even when dynamic reordering is applied), whereas the algorithm proposed herein actually produces a compact CNF representation, challenging the belief that BDDs are necessary for elimination.

2 Worked Example

Let φ denote a quantifier-free propositional formula and X denote a set of propositional variables. The key idea behind our approach is to converge onto the set of solutions of the formula $\exists X : \varphi$ from above by adding clauses formed from a sub-class of prime implicants of $\neg\varphi$; namely those prime implicants that contain no positive or negative occurrence of any variable of X. This approach contrasts with existing techniques in that it is based solely on manipulating the formula φ, rather than requiring any reengineering of the solver itself [29] or needing an auxiliary data-structure such as a BDD [22]. Furthermore the technique possesses the "everyone a winner" [33] enumeration property, which means that, rather than enumerating and filtering potential clauses of $\exists X : \varphi$, a new clause of $\exists X : \varphi$ is found on (virtually) each application of a SAT solver. This property is highly desirable, because it couples the computational effort required to compute the quantifier-free version of φ with its size. We build towards this technique using $\varphi = (\xi \wedge \mu)$ from the introduction and demonstrate how to eliminate quantifiers from $\exists X : \varphi$. We henceforth refer to this problem as that of projecting φ onto Y, where $Y = (\mathit{vars}(\varphi) \setminus X) = \{y_1, \ldots, y_k\}$. Intuitively this problem is that of removing all information in φ pertaining to variables other than Y.

2.1 Enumerating Implicants

The first step of our method is to enumerate the implicants of φ in the projection space. To do so, we first convert φ into CNF, for which we introduce a set of Tseitin variables T [31] which are existentially quantified. The resulting formula in CNF, which is equisatisfiable to φ, is denoted by $\exists T : \psi$. We then derive a so-called dual-rail encoding [8] by applying a transformation inspired by [28]. This amounts to introducing two disjoint sets of fresh variables $Y^+ = \{y_1^+, \ldots, y_k^+\}$ and $Y^- = \{y_1^-, \ldots, y_k^-\}$ and replacing each occurrence of the literal y_i in φ with y_i^+, and likewise each occurrence of the literal $\neg y_i$ with y_i^-. To ensure that y_i^+ and y_i^- cannot hold simultaneously, the transformed formula is augmented with the clauses $\bigwedge_{i=1}^{k}(\neg y_i^+ \vee \neg y_i^-)$. Let $\tau(\psi)$ denote this syntactic transformation, which yields a formula in CNF, defined over the variables $V = X \cup Y^+ \cup Y^- \cup T$. Passing $\tau(\psi)$ to a SAT solver yields a model $\mathbf{m}_1 : V \to \mathbb{B}$, such as for example:

$$\mathbf{m}_1 = \left\{ \begin{array}{l} x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_3 \mapsto 1,\ x_4 \mapsto 0,\ x_5 \mapsto 1,\ x_6 \mapsto 0 \\ y_1^+ \mapsto 0,\ y_2^+ \mapsto 0,\ y_3^+ \mapsto 0,\ y_4^+ \mapsto 1,\ y_5^+ \mapsto 0,\ y_6^+ \mapsto 1 \\ y_1^- \mapsto 1,\ y_2^- \mapsto 1,\ y_3^- \mapsto 1,\ y_4^- \mapsto 0,\ y_5^- \mapsto 1,\ y_6^- \mapsto 0 \end{array} \right\}$$

(Note that the Tseitin variables T have been omitted for the purpose of presentation.) The same model $\mathbf{m}_1$ can be represented as a subset of V, namely, $\{v \in V \mid \mathbf{m}_1(v) = 1\}$ and, henceforth, we shall use this representation interchangeably with $\mathbf{m}_1$. The variables in $\mathbf{m}_1 \cap (Y^+ \cup Y^-)$ then define a conjunction of literals, a cube, $\xi(\mathbf{m}_1)$ over the variables in Y, which is given as:

$$\xi(\mathbf{m}_1) = \left(\bigwedge\{y_i \mid y_i^+ \in (\mathbf{m}_1 \cap Y^+)\}\right) \wedge \left(\bigwedge\{\neg y_i \mid y_i^- \in (\mathbf{m}_1 \cap Y^-)\}\right)$$

Therefore, we have $\xi(\mathbf{m}_1) = (\neg y_1 \wedge \neg y_2 \wedge \neg y_3 \wedge y_4 \wedge \neg y_5 \wedge y_6)$. Furthermore, the cube $\xi(\mathbf{m}_1)$ is a so-called implicant of $\exists X : \varphi$ since $\xi(\mathbf{m}_1) \models \exists X : \varphi$. It constitutes an under-approximation of $\exists X : \varphi$ since the set of all models of $\xi(\mathbf{m}_1)$ is a subset of the set of all models of $\exists X : \varphi$. To find another under-approximation, and specifically one that is not itself entailed by $\xi(\mathbf{m}_1)$, we augment $\tau(\psi)$ with the blocking clause

$$\beta(\mathbf{m}_1) = \left(\bigvee\{y_i^- \mid y_i^+ \in (\mathbf{m}_1 \cap Y^+)\}\right) \vee \left(\bigvee\{y_i^+ \mid y_i^- \in (\mathbf{m}_1 \cap Y^-)\}\right)$$

which gives $\beta(\mathbf{m}_1) = (y_1^+ \vee y_2^+ \vee y_3^+ \vee y_4^- \vee y_5^+ \vee y_6^-)$. Of course enumerating implicants in this way dovetails with the advances in incremental SAT.

Applying a solver to the augmented formula $\tau(\psi)' = \tau(\psi) \wedge \beta(\mathbf{m}_1)$ gives another model $\mathbf{m}_2$ as follows:

$$\mathbf{m}_2 = \left\{ \begin{array}{l} x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_3 \mapsto 1,\ x_4 \mapsto 0,\ x_5 \mapsto 1,\ x_6 \mapsto 0 \\ y_1^+ \mapsto 0, y_2^+ \mapsto 1, y_3^+ \mapsto 0, y_4^+ \mapsto 1, y_5^+ \mapsto 0, y_6^+ \mapsto 1 \\ y_1^- \mapsto 1, y_2^- \mapsto 0, y_3^- \mapsto 1, y_4^- \mapsto 0, y_5^- \mapsto 1, y_6^- \mapsto 0 \end{array} \right\}$$

The model $\mathbf{m}_2$ defines another implicant $\xi(\mathbf{m}_2) = (\neg y_1 \wedge y_2 \wedge \neg y_3 \wedge y_4 \wedge \neg y_5 \wedge y_6)$ of $\exists X : \varphi$, hence $\xi(\mathbf{m}_1) \vee \xi(\mathbf{m}_2) \models \exists X : \varphi$. Repeating this strategy to derive implicants yields an unsatisfiable formula after the fourth step, and thus

$$\bigvee_{i=1}^{4} \xi(\mathbf{m}_i) = \left\{ \begin{array}{l} (\neg y_1 \wedge \neg y_2 \wedge \neg y_3 \wedge \ \ y_4 \wedge \neg y_5 \wedge \ \ y_6) \vee \\ (\neg y_1 \wedge \ \ y_2 \wedge \neg y_3 \wedge \ \ y_4 \wedge \neg y_5 \wedge \ \ y_6) \vee \\ \ \ (y_1 \wedge \neg y_2 \wedge \neg y_3 \wedge \ \ y_4 \wedge \neg y_5 \wedge \ \ y_6) \vee \\ \ \ (y_1 \wedge \neg y_2 \wedge \ \ y_3 \wedge \neg y_4 \wedge \ \ y_5 \wedge \neg y_6) \end{array} \right.$$

satisfies $\bigvee_{i=1}^{4} \xi(\mathbf{m}_i) = \exists X : \varphi$. However, observe that $\bigvee_{i=1}^{4} \xi(\mathbf{m}_i)$ is in DNF and also contains redundancies, e.g. $\xi(\mathbf{m}_1) \vee \xi(\mathbf{m}_2) = (\neg y_1 \wedge \neg y_3 \wedge y_4 \wedge \neg y_5 \wedge y_6)$. In the following section, we will demonstrate the use of cardinality constraints based on sorting networks to avoid such redundancies during model enumeration.

2.2 Enumerating Shortest Implicants

Observe that the aforementioned redundancy would not have occured, had the SAT solver first found the following model:

$$\mathbf{m}_1' = \left\{ \begin{array}{l} x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_3 \mapsto 1,\ x_4 \mapsto 0,\ x_5 \mapsto 1,\ x_6 \mapsto 0 \\ y_1^+ \mapsto 0, y_2^+ \mapsto 0, y_3^+ \mapsto 0, y_4^+ \mapsto 1, y_5^+ \mapsto 0, y_6^+ \mapsto 1 \\ y_1^- \mapsto 1, y_2^- \mapsto 0, y_3^- \mapsto 1, y_4^- \mapsto 0, y_5^- \mapsto 1, y_6^- \mapsto 0 \end{array} \right\}$$

The model $\mathbf{m}_1'$ defines an implicant $\xi(\mathbf{m}_1') = (\neg y_1 \wedge \neg y_3 \wedge y_4 \wedge \neg y_5 \wedge y_6)$, which is entailed by both $\xi(\mathbf{m}_1)$ and $\xi(\mathbf{m}_2)$ (since, given two models $\mathbf{m}_1$ and $\mathbf{m}_2$, $\xi(\mathbf{m}_2) \models \xi(\mathbf{m}_1)$ if $\mathbf{m}_1 \cap (Y^+ \cup Y^-) \subseteq \mathbf{m}_2 \cap (Y^+ \cup Y^-)$). This suggests the possibility of searching for shortest implicants. To derive shortest implicants $\xi(\mathbf{m})$ of $\exists X : \varphi$, we turn to sorting networks [25], which are used to force the number of literals in $\xi(\mathbf{m})$ (i.e. its length) to be ℓ for increasing $\ell \in \{1, \ldots, k\}$. The value of a sorting network is that it can be applied to express the sum of k propositional variables [18] in no more than $12k(\lceil \log_2(k) \rceil + 1)$ ternary clauses where the sum is represented in unary fashion. By instantiating the outputs of a sorting network, a cardinality constraint can be obtained. For example, constraining the outputs of a 6-bit sorter to 110000 ensures that exactly two input bits are set. Such cardinality constraints can be imposed in conjunction with the formula $\tau(\psi)$ in order to force the discovery of the shortest (i.e. strongest) implicants first and thereby prevent the discovery of redundant implicants. The construction proceeds by introducing a set $Y^{\pm} = \{y_1^{\pm}, \ldots, y_k^{\pm}\}$ of fresh variables, which serve as inputs to a sorting network. Each variable $y_i^{\pm}$ indicates whether y_i^+ or y_i^- appears in the implicant, and we thus we constrain $\bigwedge_{i=1}^{k} (y_i^{\pm} \Leftrightarrow (y_i^+ \vee y_i^-))$ by introducing k clauses of the form $c_i = (\neg y_i^{\pm} \vee y_i^{+} \vee y_i^{-}) \wedge (y_i^{\pm} \vee \neg y_i^{+}) \wedge (y_i^{\pm} \vee \neg y_i^{-})$. Given a k-bit sorter σ with output variables $\{o_1, \ldots, o_k\}$, a formula whose models describe implicants of $\exists X : \varphi$ of length ℓ is obtained by augmenting $\tau(\psi)$ as follows:

$$\tau_\ell(\psi) = \tau(\psi) \wedge \sigma \wedge \left(\bigwedge_{i=1}^{k} c_i\right) \wedge \left(\bigwedge_{i=1}^{\ell} o_i\right) \wedge \left(\bigwedge_{i=\ell+1}^{k} \neg o_i\right)$$

Since $\tau_\ell(\psi)$ is unsatisfiable for $l \in \{1, \ldots, 4\}$, $\exists X : \varphi$ does not possess implicants shorter than 5. Testing $\tau_5(\psi)$ for satisfiability yields the model $\mathbf{m}'_1$ as above. Then, adding $\beta(\mathbf{m}'_1)$ to $\tau_5(\psi)$ to derive other implicants of length 5 yields an unsatisfiable formula. We thus proceed with $\tau_6(\psi) \wedge \beta(\mathbf{m}'_1)$ to give two implicants:

$$\begin{aligned} \xi(\mathbf{m}'_2) &= (y_1 \wedge \neg y_2 \wedge \neg y_3 \wedge y_4 \wedge \neg y_5 \wedge y_6) \\ \xi(\mathbf{m}'_3) &= (y_1 \wedge \neg y_2 \wedge y_3 \wedge \neg y_4 \wedge y_5 \wedge \neg y_6) \end{aligned}$$

Since $\tau_6(\psi) \wedge \bigwedge_{i=1}^{3}(\beta(\mathbf{m}'_i))$ is unsatisfiable, we have $\bigvee_{i=1}^{3} \xi(\mathbf{m}'_i) = \exists X : \varphi$, and we have represented the projection in just 3 cubes. Note that although the clauses $(\bigwedge_{i=1}^{\ell} o_i) \wedge (\bigwedge_{i=\ell+1}^{k} \neg o_i)$ must be rescinded once all the implicants of length ℓ have been found, this sub-formula is itself a cube. The force of this is that SAT solvers support assumptions which are cubes. The assumption is added to the instance, thereby binding some variables, but these bindings are discarded once a model is found, in readiness for the next call to the solver. Conveniently, this lightweight version of incremental SAT is sufficient to support the above algorithm.

2.3 Over-Approximation by Dualisation

Recall that we are interested in obtaining CNF, whereas the construction we have presented so far yields formulae in DNF. Direct conversion of a formula in DNF to an equivalent one in CNF may increase the size of the formula exponentially. However, observe that since $\exists X : \varphi = \bigvee_{i=1}^{3} \xi(\mathbf{m}'_i)$, $\neg\exists X : \varphi = \neg \bigvee_{i=1}^{3} \xi(\mathbf{m}'_i) = \bigwedge_{i=1}^{3} \neg\xi(\mathbf{m}'_i)$; latter formula can be converted into CNF straightforwardly by pushing negations inward. We can thus reapply the above construction to infer implicants of $\neg\exists X : \varphi$. Given a cube ν such that $\nu \models \neg\exists X : \varphi$, the contrapositive holds, giving $\exists X : \varphi \models \neg\nu$. Therefore $\neg\nu$ over-approximates $\exists X : \varphi$, i.e. each model of $\exists X : \varphi$ is also a model of $\neg\nu$. In order to apply the above method on the dual of $\bigvee_{i=1}^{3} \xi(\mathbf{m}'_i)$, we start by negating the formula to give:

$$\neg\exists X : \varphi = \begin{cases} (y_1 \vee y_3 \vee \neg y_4 \vee y_5 \vee \neg y_6) \wedge (\neg y_1 \vee y_2 \vee y_3 \vee \neg y_4 \vee y_5 \vee \neg y_6) \wedge \\ (\neg y_1 \vee y_2 \vee \neg y_3 \vee y_4 \vee \neg y_5 \vee y_6) \end{cases}$$

Denote this formula by ω and apply τ to ω to give:

$$\tau(\omega) = \begin{cases} (y_1^{-} \vee y_3^{-} \vee y_4^{+} \vee y_5^{+} \vee y_6^{+}) \qquad \wedge (y_1^{+} \vee y_2^{-} \vee y_3^{-} \vee y_4^{+} \vee y_5^{-} \vee y_6^{+}) \wedge \\ (y_1^{+} \vee y_2^{-} \vee y_3^{+} \vee y_4^{-} \vee y_5^{+} \vee y_6^{-}) \wedge (\bigwedge_{i=1}^{6} \neg(y_i^{+} \wedge y_i^{-})) \end{cases}$$

We then solve $\tau_1(\omega)$, which is unsatisfiable: $\neg \bigvee_{i=1}^{3} \xi(\mathbf{m}'_i)$ does not posses implicants of length 1. Passing $\tau_2(\omega)$ to a SAT solver yields a model $\mathbf{m}''_1$ as follows:

$$\mathbf{m}''_1 = \left\{ \begin{array}{l} y_1^{+} \mapsto 0, y_2^{+} \mapsto 1, y_3^{+} \mapsto 0, y_4^{+} \mapsto 0, y_5^{+} \mapsto 0, y_6^{+} \mapsto 0 \\ y_1^{-} \mapsto 0, y_2^{-} \mapsto 0, y_3^{-} \mapsto 0, y_4^{-} \mapsto 0, y_5^{-} \mapsto 0, y_6^{-} \mapsto 1 \end{array} \right\}$$

We then extract a cube $\xi(\mathbf{m}_1'') = (y_2 \wedge \neg y_6)$ from $\mathbf{m}_1''$. From $\xi(\mathbf{m}_1'') \models \neg\exists X : \varphi$, we deduce $\exists X : \varphi \models \neg\xi(\mathbf{m}_1'')$ and $\neg\xi(\mathbf{m}_1'')$ is a clause. We add a blocking clause to suppress the cube as before and retrieve a model $\mathbf{m}_2''$ for $\tau_2(\omega) \wedge \beta(\mathbf{m}_1'')$, which induces a cube $\xi(\mathbf{m}_2'') = (y_3 \wedge y_6)$. Then $\exists X : \varphi \models \neg\xi(\mathbf{m}_1'') \wedge \neg\xi(\mathbf{m}_2'')$ and $\tau_2(\omega) \wedge \beta(\mathbf{m}_1'') \wedge \beta(\mathbf{m}_2'')$ is unsatisfiable. We proceed with cubes of length 3 and solve $\tau_3(\omega) \wedge \beta(\mathbf{m}_1'') \wedge \beta(\mathbf{m}_2'')$, which gives rise to a cube $\xi(\mathbf{m}_3'') = (\neg y_2 \wedge \neg y_5 \wedge \neg y_6)$. By adding blocking clauses and enumerating all cubes $\xi(\mathbf{m}_i'')$ for $i \in \{1, \ldots, m\}$, we could derive a CNF formula $\bigwedge_{i=1}^{m} \neg\xi(\mathbf{m}_i'')$ equivalent to $\exists X : \varphi$.

However, we can improve on this and produce a denser CNF representation by searching for a sub-cube of $\xi(\mathbf{m}_3'')$ which is itself an implicant of ω. To do this, let $N = (Y^+ \cup Y^-) \setminus \mathbf{m}_3'' = \{y_1^+, y_1^-, y_2^+, y_3^+, y_3^-, y_4^-, y_5^+, y_6^+\}$. We then solve $\tau_2(\omega)$ in conjunction with the cube $\bigwedge\{\neg y_i^+ \mid y_i^+ \in N \cap Y^+\} \wedge \bigwedge\{\neg y_i^- \mid y_i^- \in N \cap Y^-\}$ which we pass the solver as an assumption. The solver produces a model $\mathbf{m}_4''$ which defines $\xi(\mathbf{m}_4'') = (\neg y_5 \wedge \neg y_6)$; thus $\neg\xi(\mathbf{m}_4'') = (y_5 \vee y_6) \models \exists X : \varphi$. Since $\mathbf{m}_4'' \cap (Y^+ \cup Y^-) \subset \mathbf{m}_3'' \cap (Y^+ \cup Y^-)$, we have $\mathbf{m}_3'' \models \mathbf{m}_4''$ and $\xi(\mathbf{m}_3'') \models \xi(\mathbf{m}_4'')$. We thus discard $\xi(\mathbf{m}_3'')$ and proceed with $\tau_4(\omega) \wedge \beta(\mathbf{m}_1'') \wedge \beta(\mathbf{m}_2'') \wedge \beta(\mathbf{m}_4'')$. Whenever a fresh cube is discovered, we apply the same strategy to weaken it to the most general one that still entails ω. It is interesting to note that an implicant of length ℓ can be generalised using at most $\lceil \log_2(\ell) \rceil$ calls a solver by applying dichotomic search (though we do not apply this technique because ℓ is typically small).

Repeatedly applying this generalise scheme we derive the following minimal (though not unique) CNF representation of $\exists X : \varphi$ in five more iterations:

$$\exists X : \varphi = \begin{cases} (\neg y_2 \vee y_6) \wedge (\neg y_3 \vee \neg y_6) \wedge (y_5 \vee y_6) & \wedge\ (y_3 \vee \neg y_5) \wedge \\ (y_4 \vee \neg y_6) \wedge (y_1 \vee y_6) \quad \wedge (\neg y_1 \vee \neg y_2) & \wedge\ (\neg y_4 \vee y_6) \end{cases}$$

Since the search is exhaustive, this is no longer an over-approximation of the projection, but equivalent to it. Our implementation of this algorithm using MiniSat takes 0.0012s and 0.0009s for the first and second stages of the algorithm (corresponding to Sections 2.2 and 2.3 respectively) thus taking 0.0021s overall.

2.4 Reprise and Reflection

One may wonder why not to enumerate prime implicants of $\neg\varphi$ directly as previously proposed [6]. To find an over-approximation $\neg\nu$ of $\exists X : \varphi$, put $\neg\varphi$ into CNF using a formula κ such that $\exists T : \kappa \equiv \neg\varphi$. Further, observe that $\nu \models \forall X : \exists T : \kappa$ iff $\neg\forall X : \exists T : \kappa \models \neg\nu$ iff $\exists X : \neg\exists T : \kappa \models \neg\nu$ iff $\exists X : \varphi \models \neg\nu$. Hence, to find an over-approximation of $\exists X : \varphi$, it suffices to find an implicant of $\forall X : \exists T : \kappa$. Since $\forall X : \exists T : \kappa \models \exists X : \exists T : \kappa$, each implicant of $\forall X : \exists T : \kappa$ is also an implicant of $\exists X : \exists T : \kappa$. This suggests enumerating each implicant ν of $\exists X : \exists T : \kappa$ and discarding those ν which are not implicants of $\forall X : \exists T : \kappa$, that is, those that fail the entailment check $\varphi \models \neg\nu$. However, this is hardly an "everyone a winner" strategy as this method produces very large numbers of spurious implicants that fail the entailment check. By combining model enumeration with the generation of prime implicants on the dual formula, our new method does not generate any spurious candidates, which explains performance improvements of several orders of magnitude (see Sect. 4).

3 Formal Correctness

Let Bool_V denote the class of propositional Boolean formulae over the set of variables V, which is partitioned into two disjoint subsets X and Y, i.e. $V = X \cup Y$ and $X \cap Y = \emptyset$. We shall consider the problem of computing an implicant of $\exists X : \varphi$, where the formula $\varphi \in \mathsf{Bool}_V$ is in CNF. The transformation is formalised as a map τ on the set of literals $\mathsf{Lit}_V = \{v, \neg v \mid v \in V\}$ over V. This map is, in turn, defined in terms of propositional variables $Y^+ = \{y^+ \mid y \in Y\}$ and $Y^- = \{y^- \mid y \in Y\}$ where $Y^+ \cap Y^- = \emptyset$ and $(Y^+ \cup Y^-) \cap V = \emptyset$.

Definition 1. The literal transformation map $\tau : \mathsf{Lit}_V \to \mathsf{Lit}_{X \cup Y^+ \cup Y^-}$ and its inverse $\tau^{-1} : \mathsf{Lit}_{X \cup Y^+ \cup Y^-} \to \mathsf{Lit}_V$ are defined as follows:

$$\tau(l) = \begin{cases} y^+ & \text{if } l = y \text{ and } y \in Y \\ y^- & \text{if } l = \neg y \text{ and } y \in Y \\ l & \text{otherwise} \end{cases} \qquad \tau^{-1}(l) = \begin{cases} y & \text{if } l = y^+ \\ \neg y & \text{if } l = y^- \\ l & \text{otherwise} \end{cases}$$

To lift τ to clauses, a clause is considered to be merely a set of literals. Then $\tau(C) = \{\tau(l) \mid l \in C\}$ for a clause $C \subseteq \mathsf{Lit}_V$. The literal transformation map τ is lifted to cubes and implicants (which is a particular type of cube) by likewise considering these to be sets of conjoined literals. The transformation relates cubes with literals drawn from Lit_V to cubes with literals drawn from $\mathsf{Lit}_{X \cup Y^+ \cup Y^-}$. We then define non-trivial cubes (which do not contain opposing literals) as below:

Definition 2

$$\mathsf{Cube}_V = \left\{ C \subseteq \mathsf{Lit}_V \,\middle|\, \forall v \in V : \{v, \neg v\} \not\subseteq C \right\}$$
$$\mathsf{Cube}_{X,Y} = \left\{ C \cup C' \,\middle|\, C \in \mathsf{Cube}_X \wedge C' \subseteq Y^+ \cup Y^- \wedge \forall y \in Y : \{y^+, y^-\} \not\subseteq C' \right\}$$

Note that a formula φ represented in CNF can be considered to be a set of implicitly conjoined clauses F. This is used to state the following equivalence result which asserts that implicants are preserved by the transformation τ:

Proposition 1 (Equivalence). Let $\varphi = \bigwedge\{\bigvee C \mid C \in F\}$ where $F \subseteq 2^{\mathsf{Lit}_V}$ and put $\varphi' = \bigwedge\{\bigvee \tau(C) \mid C \in F\}$. Then

- If $D \in \mathsf{Cube}_V$ and $(\bigwedge D) \models \varphi$ then $(\bigwedge \tau(D)) \models \varphi'$.
- If $D' \in \mathsf{Cube}_{X,Y}$ and $(\bigwedge D') \models \varphi'$ then $(\bigwedge \tau^{-1}(D')) \models \varphi$.

The following corollary of the above relates implicants with literals drawn from Lit_Y to the satisfiability of the transformed clause set:

Corollary 1. Suppose φ and φ' are defined as above. Then

– If $D \in \mathsf{Cube}_Y$ and $\bigwedge D \models \exists X : \varphi$ then $(\bigwedge \tau(D)) \wedge \varphi'$ is satisfiable.
– If $D' \in \mathsf{Cube}_{Y,\emptyset}$ and $(\bigwedge D') \wedge \varphi'$ is satisfiable then $(\bigwedge \tau^{-1}(D')) \models \exists X : \varphi$.

To state how to compute an image by enumerating implicants, the unusual notion of a blocking clause introduced in Sect. 2 is now formalised:

Definition 3. The mapping $\beta : \mathsf{Cube}_{Y,\emptyset} \to \mathsf{Cube}_{Y,\emptyset}$ is defined:

$$\beta(D') = \{y_i^- \mid y_i^+ \in D'\} \cup \{y_i^+ \mid y_i^- \in D'\}$$

Theorem 1 (correctness). Suppose φ and φ' are defined as above.
Let $D'_1, \ldots, D'_\ell \in \mathsf{Cube}_{Y,\emptyset}$ be a sequence such that:

– $(\bigwedge_{l \in D'_k} l) \wedge \varphi' \wedge (\bigwedge_{i=1}^{k-1} (\bigvee_{l \in \beta(D'_i)} l))$ is satisfiable for all $k \in \{1, \ldots, \ell\}$ and
– $\varphi' \wedge (\bigwedge_{i=1}^{\ell} (\bigvee_{l \in \beta(D'_i)} l))$ is unsatisfiable.

Then $\bigvee_{i=1}^{\ell} \wedge \tau^{-1}(D'_i) = \exists X : \varphi$.

The following proposition dovetails with the theorem to show how a CNF representation of the projection can be derived in a two phase process. The corollary that follows is immediate and states that the computation of implicants, in the second phase at least, can be aborted prematurely without sacrificing correctness.

Proposition 2 (dualisation). Let $\psi = \bigvee_{i=1}^{\ell} (\bigwedge_{d \in D_i} d)$ where $D_1, \ldots, D_\ell \in \mathsf{Cube}_Y$. Further, let $\exists X : \varphi = \bigvee_{i=1}^{m} (\bigwedge_{e \in E_i} e)$ where $E_1, \ldots, E_m \in \mathsf{Cube}_Y$ and $\varphi = \bigwedge_{i=1}^{\ell} (\bigvee_{l \in D_i} \neg l)$. Then $\psi = \bigwedge_{i=1}^{m} (\bigvee_{l \in E_i} \neg l)$.

Corollary 2 (anytime). Let $\psi = \bigvee_{i=1}^{\ell} (\bigwedge_{d \in D_i} d)$ where $D_1, \ldots, D_\ell \in \mathsf{Cube}_Y$. Let $\bigwedge_{i=1}^{m} E_i \models \exists X : \varphi$ where $E_1, \ldots, E_m \in \mathsf{Cube}_Y$ and $\varphi = \bigwedge_{i=1}^{\ell} (\bigvee_{l \in D_i} \neg l)$. Then $\psi \models \bigwedge_{i=1}^{m} (\bigvee_{l \in E_i} \neg l)$.

The above results are presented in terms of any implicants, rather than prime implicants only. This is because, while the latter govern the rate of convergence, they do not affect correctness. Nevertheless, a prime implicant of an existentially quantified formula can be formulated as two satisfability conditions. To state the corollary, let $[\![\varphi]\!] \subseteq 2^V$ denote the set of models of the Boolean function φ. For example, if $V = \{x, y\}$ then $[\![x \vee y]\!] = \{\{x\}, \{y\}, \{x, y\}\}$.

Corollary 3. Suppose φ, φ' and $F \subseteq 2^{\mathsf{Lit}_V}$ are defined as above and put $\psi = \varphi' \wedge (\bigwedge_{y \in Y} (\neg y^+ \vee \neg y^-))$. Then $D \in \mathsf{Cube}_Y$ is a prime implicant of $\exists X : \varphi$ iff $D = \tau^{-1}(M^\star \cap (Y^+ \cup Y^-))$ where $M^\star \in [\![\psi]\!]$ and $|M^\star \cap (Y^+ \cup Y^-)| \leq |M \cap (Y^+ \cup Y^-)|$ for all $M \in [\![\psi]\!]$.

Note that ψ does not include any cardinality constraint on the set $M^\star \cap (Y^+ \cup Y^-)$, hence the need to define a prime implicant in terms of an implicant no longer than any other. The above result can straightforwardly be adapted to specify how an implicant of a given size can be defined as a SAT instance.

To conclude the elaborations on correctness, we observe that the greedy generation of prime implicants does not necessarily yield a minimal CNF formula. To see this, suppose $\varphi = (\neg w \wedge x \wedge y) \vee (\neg x \wedge \neg y \wedge \neg z)$ and consider $\exists X : \varphi$ where $X = \{x\}$. Clearly $\exists X : \varphi = D_1 \vee D_2$ where $D_1 = (\neg w \wedge y)$ and $D_2 = (\neg y \wedge \neg z)$. But also $\exists X : \varphi = E_1 \vee E_2 \vee E_3$ where $E_1 = (\neg w \wedge \neg z)$, $E_2 = (\neg w \wedge y \wedge z)$ and $E_3 = (w \wedge \neg y \wedge \neg z)$. Observe $|D_1| \leq |D_2|$ and likewise $|E_1| \leq |E_2| \leq |E_3|$, and indeed either CNF formulae can be generated, though the latter is sub-optimal.

4 Experiments

We have implemented the techniques described in this paper in C++ using MiniSat with the express aim of answering the following questions:

- What is the overhead of using primes compared to standard enumeration?
- How are the primes distributed in terms of size within the two phases of the algorithm, i.e. for DNF generation and CNF conversion?
- How does the method compare against BDD-based projection scheme, both in terms of the size of CNF formulae and the time required to produce them?

To answer these questions, we compared our technique against a hybrid SAT/BDD approach. We implemented our method on top of MiniSat v2.2. Cudd v2.4.2 was used for the BDD package since it offers direct support for enumerating the prime implicants of a BDD. We chose bitonic sorting for the sorting network, though smaller (albeit less regular) networks exist [25]. All experiments were performed on a 2.6 GHz MacBook Pro equipped with 4 GB of RAM.

4.1 Benchmarks

As benchmarks, we selected several circuits from the 74X and ISCAS-89 benchmark series as well as projection problems arising from range analysis of microcontroller code [3,5]. The 74X circuits include an ALU (74181), a carry-look-ahead generator (74182), an adder (74283) and a magnitude comparator (74L85). The ALU is the hardest to analyse since it implements 16 different functions, depending on 4 control bits. The ISCAS-89 benchmarks consist of a traffic light controller (s298), two implementations of a 4×4 add-shift multiplier (s344 and s349), and a combinatorial circuit with randomly inserted flip-flops (s1196). All circuits were projected onto their input and output variables so as to express their semantics without reference to any intermediate variables.

The microcontroller code was exported from [mc]square [36] for the purpose of synthesising transfer functions [5] for propagating ranges across blocks of

Table 1. Information regarding the benchmark set; column φ contains the name of the formula as referred to later on, followed by information about the origin of the respective formula and its size; the benchmarks at the bottom are generated from blocks of ATmega16 binary code; for these benchmarks, column *info* contains the number of instructions and whether they were generated for set abstraction (*set*) or transfer function synthesis (*tf*).

φ	info	$\|V\|$	$\|\varphi\|$
74181	74x series	1001	2368
74182	74x series	227	526
74283	74x series	267	646
74L85	74x series	413	1084
add	3 (set)	74	119
increment	3 (set)	66	119
parity_mit	15 (set)	2066	6725
parity_swap	21 (set)	275	745
randerson	13 (set)	18658	61696
triple_swap	9 (set)	89	192
s298	ISCAS-89	1327	3164
s344	ISCAS-89	1665	3880
s349	ISCAS-89	1678	3914
s1196	ISCAS-89	5422	12870
adc	4 (tf)	19	290
admdswpcmp	11 (tf)	66	154
adsb2shad	8 (tf)	114	322
ilsh	5 (tf)	66	170
irsh	5 (tf)	66	170
iswp	8 (tf)	130	386

Atmel ATmega16 code. Transfer function synthesis is essentially an existential quantifier problem We also considered projection problems that arise when over-approximating the set of values that a register can take in a block (when a block is considered in isolation to those blocks that flow into it [3]). Table 1 presents the key statistics for each of these projection problems.

4.2 Projecting Using Prime Implicants

Table 2 presents the results for DNF generation (resp. CNF conversion) using prime implicants, giving the number of implicants (resp. clauses) in the resulting formulae and the time required to compute them. Analogous figures are given for the hybrid approach. It is interesting to see that for the circuits s344 and s349, only 512 implicants in DNF are generated, but exhaustive model enumeration yields 65792 disjuncts. This is because 256 out of 512 implicants are of length 12, and thus already cover a large number of models in the projection space. This suggests that our method can make model enumeration tractable where the classical approach fails. For other cases, as exemplified by the 74181 and s1196 circuits, our approach offers no clear advantage. However, it is important to see that transformation never seriously degrades performance; this is noteworthy because one cannot know the distribution of the primes up front.

The percental distribution of the lengths of clauses that arise in CNF conversion are depicted in Fig. 1. For reasons of space, graphs are given only for the 74X series (though these distributions are typical). For DNF generation the distributions are less interesting for these benchmarks, often consisting of a single spike, but sometimes consist of two spikes, as for s344 and s349 at lengths 12 and 20. It is in these latter cases that primes improve over classic model enumeration.

Table 2. Experimental results for projection using prime implicant enumeration and comparison to BDD-based method; the best results are emphasized

φ	$\lvert Y\rvert$	Primes					model		Hybrid		
		DNF		CNF		total	enum		BDD		total
		size	time	size	time	time	size	time	size	time	time
74181	22	16384	1.477	686	7.096	8.574	16384	1.421	**476**	**1.320**	2.798
74182	13	320	0.025	26	**0.009**	0.035	320	0.009	**23**	0.014	0.039
74283	14	512	0.022	**98**	0.147	0.169	512	0.023	270	**0.077**	0.099
74L85	14	2048	0.108	**144**	0.107	0.215	2048	0.092	145	**0.053**	0.162
s298	9	4	0.001	7	**0.003**	0.004	4	0.004	7	0.006	0.007
s344	20	512	0.068	16	**0.018**	0.087	65792	12.811	16	0.030	0.098
s349	20	512	0.070	16	**0.017**	0.088	65792	12.001	16	0.029	0.099
s1196	28	16384	11.182	**570**	**5.465**	16.653	16384	11.374	822	5.810	16.993
adder	16	256	0.007	16	**0.012**	0.020	1024	0.025	16	0.024	0.031
	24	1024	0.030	31	**0.054**	0.086	4096	0.117	**29**	0.090	0.120
increment	8	4	0.001	10	**0.001**	0.003	4	0.001	10	0.006	0.007
	16	256	0.004	14	**0.007**	0.012	256	0.003	14	0.010	0.014
	24	256	0.008	**32**	**0.024**	0.033	256	0.004	34	0.027	0.035
parity_mit	8	100	0.033	4	**0.001**	0.036	200	0.005	4	0.006	0.039
	16	12800	2.363	16	1.361	3.727	25600	5.227	**10**	**0.284**	2.647
	24	40960	8.543	**40**	6.316	14.875	51200	11.348	41	**1.155**	9.698
parity_swap	8	16	0.002	4	**0.001**	0.004	16	0.001	4	0.007	0.009
	16	256	0.008	12	**0.008**	0.017	256	0.010	12	0.011	0.019
	24	256	0.013	**37**	**0.038**	0.051	256	0.011	40	0.101	0.114
randerson	8	64	0.102	2	**0.001**	0.104	64	0.051	2	0.006	0.108
	16	256	0.136	14	**0.010**	0.147	256	0.089	14	0.013	0.149
	24	256	0.140	**27**	**0.023**	0.164	256	0.092	30	0.058	0.198
triple_swap	8	16	0.002	12	**0.001**	0.004	64	0.005	12	0.007	0.009
	16	512	0.013	**20**	0.029	0.042	512	0.009	22	**0.015**	0.028
adc	8	128	0.004	7	**0.002**	0.006	512	0.010	7	0.006	0.010
	16	128	0.005	**47**	**0.018**	0.023	512	0.012	52	0.036	0.041
	24	128	0.006	**80**	**0.047**	0.054	512	0.012	92	0.116	0.122
admdswpcmp	8	191	0.003	7	**0.003**	0.006	191	0.002	7	0.008	0.011
	16	191	0.007	**54**	**0.021**	0.029	191	0.005	60	0.049	0.057
	24	191	0.009	**56**	**0.025**	0.035	191	0.006	66	0.071	0.080
adsb2shad	16	154	0.008	**67**	**0.026**	0.034	512	0.012	71	0.089	0.097
	24	310	0.013	**124**	**0.045**	0.058	1024	0.021	129	0.095	0.108
ilsh	8	32	0.002	3	**0.001**	0.003	32	0.002	3	0.008	0.010
	16	256	0.008	13	**0.009**	0.017	256	0.005	13	0.011	0.019
	24	256	0.009	**44**	**0.023**	0.032	256	0.006	46	0.035	0.046
irsh	8	16	0.001	4	**0.001**	0.001	64	0.003	4	0.007	0.008
	16	16	0.002	22	**0.004**	0.006	64	0.003	**21**	0.009	0.011
	24	16	0.003	45	**0.012**	0.016	64	0.003	45	0.017	0.020
iswp	16	4096	0.103	16	0.235	0.339	4096	0.061	16	**0.075**	0.179
	24	4096	0.126	27	0.251	0.379	4096	0.073	27	**0.140**	0.266

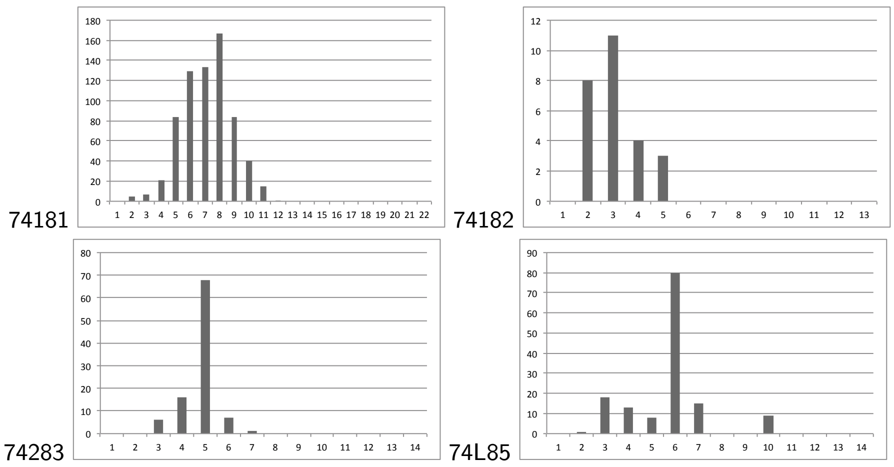

Fig. 1. Distribution of implicants by length for the 74X benchmarks

4.3 Projecting Using BDDs

In the hybrid SAT/BDD based approach, DNF to CNF conversion is realised with a BDD. To support this, CUDD provides a dedicated operation that computes prime implicants of a given BDD by finding a shortest path from the root to 1 leaves (though the lengths of the implicants and their number depend on the variable ordering). In terms of size of the resulting CNF formula, it is interesting to see that BDDs do not necessarily give the smallest representation; far from it.

In terms of running times, there is no clear winner: for the largest problem, s1196, the SAT-based approach is faster for CNF conversion whereas the BDD-based method is superior for 74181. However, we suspect that the balance may well shift towards SAT if solvers continue to advance in performance. Furthermore, the implementation of the SAT-based scheme required less than 100 lines of code which itself makes it attractive.

4.4 Generalising Implicants

During the development of the method described in this paper, we found that generalisation (as described in Sect. 2.3) had to be applied in tandem with search for a shortest implicant for the formula that is accompanied with the blocking clauses (as described in Sect. 2.2). Enumerating implicants without generalisation yields a much larger number of clauses; for the 74283 benchmark, 932 instead of 98. Strangely the runtimes are almost equal, which is the typical pattern. We conclude that generalisation is advisable since, though it does not improve the runtime, it does improve the density of the resulting CNF formula.

5 Related Work

The complexity of the shortest implicant problem for DNF formulae has been studied by Umans [38] who showed that it is $GC(\log_2(n), coNP)$-complete. Even though this result is not directly transferable to CNF, it suggests the parallel problem in CNF may be similarly difficult and thereby supports the application of SAT solvers to the derivation of shortest implicants.

5.1 Consensus Method and Resolution

The consensus method has been proposed [4,32,35] as a way of enumerating all the prime implicants of a propositional function in DNF. If f is in CNF, then it is straightforward to derive a DNF representation of $\neg f$, to which the consensus procedure can be applied to find its prime implicants. One might think that this provides a way to compute projection, but the key step of the consensus method combines two elementary conjunctions of $\neg f$, say, $x \wedge C$ and $(\neg x) \wedge D$, to form $C \wedge D$, which is isomorphic to resolution. Hence the consensus method shares the inefficiency problems associated with applying resolution to a formula in CNF.

5.2 Hybrid Methods and McMillan's Method

SAT has been used before to compute projections [26,29] as have BDDs [10,26,39]. Hybrid approaches that combine SAT solving and BDDs typically represent state sets as BDDs and express the transition relation in CNF [22,37], though some approaches combine BDDs and SAT solving in different ways. For example, Damiano and Kukula [16] substitute clauses with BDDs in a DPLL solver, Jin and Somenzi [23] combine BDDs and SAT solving using CNF to avoid explosion in the sizes of the resulting BDDs, whereas Aloul et al. [1] study the connection between CNF formulae and BDDs for good variable orderings. The approach of Cavada et al. [11] recursively computes quantifications for subtrees, which are then combined; SMT solving ensures consistency of the transformations.

McMillan [29] has shown how to perform universal projection for CTL modalities such as $\mathsf{AX}\varphi$ using DPLL-like enumeration and also explained how to represent an arbitrary Boolean encoding of φ in CNF without existential quantification. The key idea of his `toCNF(`φ`)` procedure [29, Sect. 3] is to deduce a clause from a satisfying assignment of φ whose complement rules out some cases that violate φ. His approach requires a modified DPLL-engine and resolution coupled with several heuristics — which literals to analyse, which variables to resolve on and suchlike — which strongly affect the performance of the approach [29, Sect. 2]. Our approach, in comparison, builds on top of an existing SAT library and is therefore both straightforward to implement and will immediately benefit from any improvement to the library itself. Nevertheless, we consider the SAT-based algorithm of McMillan to be an important work that has indeed found application in the predicate abstraction of hardware circuits [13] and post-image computation [12]. A variation on the McMillan algorithm is given by Sheng and Hsiao [37] who apply a success-driven rather than a conflict-driven search for models (recall that DPLL-style algorithms use a conflict-driven search). However, Sheng and Hsiao store their results in a BDD rather than generating a CNF formula.

5.3 Methods Based on Integer Linear Programming

Integer linear programming has been used to find shortest implications, as have SAT engines which have been modified to support inequalities [28]. In this work a transformation is described which is similar to τ. However, the work is not concerned with quantifier elimination, hence 0-1 variables are introduced for each variable in the formula rather than merely those in the projection space.

5.4 Methods Based on Primes and Cubes

Prime implicants have been directly applied to widening Boolean functions represented as ROBDDs [24]. By applying a recursive meta-product construction [15], collections of short primes can be used to derive an ROBDD that is an over-approximation of the input. Our work on applying SAT to projection was motivated by the empirical finding that collections of short primes often yield good approximations of Boolean formulae [24, Sect. 5.1].

Lahiri et al. [26] have described how to enumerate cubes in the projection space using SAT so as to perform image computation for predicate abstraction. Blocking clauses are chosen heuristically, though details of the heuristics are not given, and the approach does not guarantee to infer cubes of minimal size. This work was further developed by Lahiri et al. [27] who used DPLL(T)-based SMT solving to enumerate models. Each model is then stored in a BDD from which the results are extracted as disjunctions of prime implicants. They search for cubes $c_{1,k}, \ldots, c_{n,k}$ of increasing length k such that $\varphi \models \bigvee_{i=1}^{n} c_{i,k}$, which chimes with our approach. In contrast, however, we apply prime implicants in two different ways, that is, for enumerating cubes as well as clauses, so that our final quantifier-free formula is presented in CNF. To illustrate the conceptual difference between the methods, consider the benchmarks `s344` and `s349` from Sect. 4, for which DNF enumeration yields 256 cubes of lengths 12 and 22. The method of Lahiri et al. enumerates all intermediate cubes of lengths $13, \ldots, 21$ to converge onto $\exists X : \varphi$, whereas our approach leapfrogs these intermediate cubes by specifying the requirement of a cube of size k within the SAT instance itself. The MathSat SMT solver [7] uses an algorithm that also relies on a formula transformation similar to τ. However, rather than adding cardinality constraints to the SAT instance, they modified the solver so that it takes 0 decisions during SAT solving. Earlier approaches [17,19,34] to predicate abstraction invoke a solver for each cube c to discover if $\varphi \wedge c$ is satisfiable. To reduce the number of calls to the decision procedure, they start with small cubes, and only if $\varphi \wedge c$ is satisfiable, they proceed with cubes of the form $\varphi \wedge c \wedge d$ and $\varphi \wedge c \wedge \neg d$. This approach is based on a large number of SAT/SMT calls, typically requires many unsatisfiability proofs (which are often more difficult for SAT solvers to provide than find a model), and does not fit as well with incremental SAT [27, Sect. 3.2]. More recently, Monniaux [30] described a method for quantifier elimination called lazy model enumeration. The key idea of his algorithm is to derive a cube that implies a given formula, which is then generalised towards a weaker implicant. By way of comparison, our algorithm starts with implicants as short as possible; weakening is required by the encoding. Although similar in spirit, his algorithm proceeds diametrically opposed to ours, and moreover generates DNF.

6 Concluding Discussion

This paper advocates using SAT to eliminate existential quantifiers from formulae presented in CNF. The method is based on a two-phase approach: first, SAT solving is applied to enumerate the prime implicants of a quantified formula; second, cubes are translated into clauses to derive a CNF representation of the projection. The second phase is anytime in that it can be stopped early without compromising soundness. This can be considered a pragmatic response to the complexity of DNF to CNF conversion. As well as exploiting advances in incremental SAT and finessing the need to modify a solver, it provides an efficient way of storing projections without BDDs, whilst avoiding the blow-up in the number of intermediate clauses that comes with applying resolution.

Acknowledgements. The first author was supported, in part, by the DFG research training group 1298 *Algorithmic Synthesis of Reactive and Discrete-Continuous Systems* and by the DFG Cluster of Excellence on Ultra-high Speed Information and Communication, German Research Foundation grant DFG EXC 89. The second author was funded, in part, by a Royal Society travel grant, reference TG092357, and a Royal Society Industrial Fellowship, reference IF081178. We thank Edd Barrett, Armin Biere, Stefan Kowalewski, Axel Simon and the anonymous referees for helpful comments.

References

1. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Faster SAT and Smaller BDDs via Common Function Structure. In: ICCAD, pp. 443–448 (2001)
2. Armstrong, T., Marriott, K., Schachte, P., Søndergaard, H.: Two Classes of Boolean Functions for Dependency Analysis. Sci. Comp. Program. 31(1), 3–45 (1998)
3. Barrett, E., King, A.: Range and Set Abstraction Using SAT. Electronic Notes in Theoretical Computer Science 267(1), 17–27 (2010)
4. Blake, A.: Canonical expressions in Boolean algebra. University of Chicago, Chicago (1938)
5. Brauer, J., King, A.: Automatic abstraction for intervals using boolean formulae. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 167–183. Springer, Heidelberg (2010)
6. Brauer, J., King, A.: Approximate quantifier elimination for propositional boolean formulae. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 73–88. Springer, Heidelberg (2011)

7. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MATHSAT 4 SMT solver. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 299–303. Springer, Heidelberg (2008)
8. Bryant, R.: Boolean analysis of MOS circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 6(4), 634–649 (1987)
9. Bryant, R.E.: A view from the engine room: Computational support for symbolic model checking. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 145–149. Springer, Heidelberg (2008), `http://www.slidefinder.net/m/mc25/7464626`
10. Burch, J.R., Clarke, E.M., McMillan, K.L.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98, 142–170 (1992)
11. Cavada, R., Cimatti, A., Franzén, A., Kalyanasundaram, K., Roveri, M., Shyamasundar, R.K.: Computing predicate abstractions by integrating BDDs and SMT solvers. In: FMCAD, pp. 69–76. IEEE Computer Society Press, Los Alamitos (2007)
12. Chauhan, P., Clarke, E.M., Kroening, D.: A SAT-based algorithm for reparameterization in symbolic simulation. In: DAC, pp. 524–529. ACM Press, New York (2004)
13. Clarke, E., Talupur, M., Veith, H., Wang, D.: SAT based predicate abstraction for hardware verification. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 78–92. Springer, Heidelberg (2004)
14. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 236–250. Springer, Heidelberg (2010)
15. Coudert, O., Madre, J.C.: Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In: DAC, pp. 36–39. IEEE Computer Society Press, Los Alamitos (1992)
16. Damiano, R.F., Kukula, J.H.: Checking satisfiability of a conjunction of BDDs. In: DAC, pp. 818–823. ACM Press, New York (2003)
17. Das, S., Dill, D.L., Park, S.: Experience with predicate abstraction. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 160–171. Springer, Heidelberg (1999)
18. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. JSAT 2(1-4), 1–26 (2006)
19. Flanagan, C., Qadeer, S.: Predicate Abstraction for Software Verification. In: POPL, pp. 191–202. ACM Press, New York (2002)
20. Genaim, S., Giacobazzi, R., Mastroeni, I.: Modeling Secure Information Flow with Boolean Functions. In: IFIP WG 1.7, ACM Workshop on Issues in the Theory of Security, Barcelona, Spain, pp. 55–66 (2004)
21. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
22. Gupta, A., Yang, Z.-J., Ashar, P., Gupta, A.: SAT-based image computation with application in reachability analysis. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 354–371. Springer, Heidelberg (2000)
23. Jin, H., Somenzi, F.: CirCUs: A hybrid satisfiability solver. In: Holger Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 211–223. Springer, Heidelberg (2005)
24. Kettle, N., King, A., Strzemecki, T.: Widening rOBDDs with prime implicants. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 105–119. Springer, Heidelberg (2006)

25. Knuth, D.E.: The Art of Computer Programming, vol. 3. Addison-Wesley, London (1997)
26. Lahiri, S.K., Bryant, R.E., Cook, B.: A symbolic approach to predicate abstraction. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 141–153. Springer, Heidelberg (2003)
27. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT techniques for fast predicate abstraction. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 424–437. Springer, Heidelberg (2006)
28. Manquinho, V.M., Flores, P.F., Silva, J.P.M., Oliveira, A.L.: Prime Implicant Computation Using Satisfiability Algorithms. In: International Conference on Tools with Artificial Intelligence, pp. 232–239. IEEE Computer Society Press, Los Alamitos (1997)
29. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 250–264. Springer, Heidelberg (2002)
30. Monniaux, D.: Quantifier elimination by lazy model enumeration. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 585–599. Springer, Heidelberg (2010)
31. Plaisted, D., Greenbaum, S.: A Structure-Preserving Clause Form Translation. Journal of Symbolic Computation 2(3), 293–304 (1986)
32. Quine, W.V.: A way to simplify truth functions. American Mathematical Monthly 62(9), 627–631 (1955)
33. Read, R.C.: Everyone a Winner. Annals of Discrete Mathematics 2, 107–120 (1978)
34. Saïdi, H., Shankar, N.: Abstract and model check while you prove. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 443–454. Springer, Heidelberg (1999)
35. Samson, E.W., Mills, B.E.: Circuit minimization: Algebra and algorithms for new Boolean canonical expressions. Technical Report TR, pp. 21–54. United States Air Force, Cambridge Research Lab (1954)
36. Schlich, B.: Model checking of software for microcontrollers. ACM Trans. Embedded Comput. Syst 9(4), article Number 36 (2010)
37. Sheng, S., Hsiao, M.S.: Efficient Preimage Computation Using A Novel Success-Driven ATPG. In: DATE, pp. 10822–10827. IEEE Computer Society Press, Los Alamitos (2003)
38. Umans, C.: The Minimum Equivalent DNF Problem and Shortest Implicants. In: FOCS, pp. 556–563. IEEE Computer Society Press, Los Alamitos (1998)
39. Weaver, S., Franco, J.V., Schlipf, J.S.: Extending Existential Quantification in Conjunctions of BDDs. JSAT 1(2), 89–110 (2006)