Widening ROBDDs with Prime Implicants

Neil Kettle[1], Andy King[1], and Tadeusz Strzemecki[2]

[1] University of Kent, Canterbury, CT2 7NF, UK
[2] Fordham University, New York, NY 10023, USA

Abstract. Despite the ubiquity of ROBDDs in program analysis, and extensive literature on ROBDD minimisation, there is a dearth of work on approximating ROBDDs. The need for approximation arises because many ROBDD operations result in an ROBDD whose size is quadratic in the size of the inputs. Furthermore, if ROBDDs are used in abstract interpretation, the running time of the analysis is related not only to the complexity of the individual ROBDD operations but also the number of operations applied. The number of operations is, in turn, constrained by the number of times a Boolean function can be weakened before stability is achieved. This paper proposes a widening that can be used to both constrain the size of an ROBDD and also ensure that the number of times that it is weakened is bounded by some given constant. The widening can be used to either systematically approximate from above (i.e. derive a weaker function) or below (i.e. infer a stronger function).
Keywords: ROBDD, widening, approximation, abstract interpretation.

1 Introduction

Reduced-Ordered Binary Decision Diagrams (ROBDDs) have numerous applications in model checking [4], program analysis [25] and abstract interpretation [1]. The popularity of ROBDDs stems from their memory-efficient encoding of Boolean functions and a canonical representation that supports the memoisation of ROBDD operations. The worst-case complexity of many ROBDD operations is quadratic in the size of the inputs [2], but the inherent intractability of Boolean function manipulation inevitably manifests itself; even though ROBDDs are constructed so as to factor out all replicated sub-ROBDDs, Boolean functions exist whose size is exponential in the number of variables no matter what variable ordering is employed [3]. Intractably large ROBDDs can [6] and do [11] arise in program analysis. In particular, when an analysis associates each program variable with n attributes and m program variables appear in scope, then an ROBDD over $m\lceil \lg(n) \rceil$ propositional variables are required to encode the dependencies between the attributes of the program variables. Even with the use of sophisticated tree-automata techniques to improve the encoding [12], problematically large ROBDDs still arise even when $m \approx 100$ [11].

ROBDDs are not only problematic in terms of space but also in terms of time. This is not only due to the complexity of individual ROBDD operations, but because the number of ROBDD operations is itself potentially exponential.

In the context of abstract interpretation, this has particular relevance as analysis is typically formulated as a fixpoint. Suppose, for example, that the result of an analysis is conceived as the least fixpoint of a series of equations:

$$\begin{aligned} f_1 &= F_1(f_1, \ldots, f_n) \\ \vdots \; & \;\vdots \qquad \vdots \\ f_n &= F_n(f_1, \ldots, f_n) \end{aligned}$$

where each f_i is a propositional function over m variables $x_1, \ldots, x_m$ and each F_i is an operation on $f_1, \ldots, f_n$ obtained by, say, composing monotonic operations such as disjunction $f_i \vee f_j$, conjunction $f_i \wedge f_j$ and existential quantification $\exists_{x_i}(f_j)$. The least fixpoint can be computed by setting $f_i = false$ and then reapplying the n equations until stability is achieved. In the worst-case, each application of n equations might weaken exactly one f_i by adding a single model. Since each f_i can possess 2^m models, a chain of $n2^m$ iterates are required in the worst-case which violates the general requirement for a polynomial analysis. (The reader is referred to [6] for examples that manifest this behaviour).

In program analysis, it is generally better to return an approximate answer in an acceptable time than an exact answer in an exorbitant time. To this end, widening operators have been proposed [9] that accelerate convergence on computational domains that possess either infinite or very long chains. This use of widening trades precision for time. However, widening can also be used to trade precision for space, for example, replace one ROBDD with another that has more models yet has a more compact representation [11, 15, 18, 21]. Despite extensive literature on reducing the size of an ROBDD by selecting a propitious variable ordering (the reader is referred to citations of the classic paper [19] on variable reordering and minimisation), the problem of widening ROBDDs has received relatively scant attention. This paper plugs this gap by proposing a new widening for ROBDDs based upon the enumeration of prime implicants [8] that has a number of attractive properties:

- The widening can ensure that each f_i is not weakened more than a prescribed number of times. Previous attempts at bounding the iterations have confined the analysis to a fixed sub-domain of Boolean formulae [13]. The widening can support richer classes of dependencies without sacrificing scalability.
- The widening can compute dense approximations of an ROBDD. Moreover, by constructing the new ROBDD in terms of the progressively longer implicants, the widening can be tuned to achieve the desired degree of precision.
- The widening is not dependent on the variable ordering. State-of-the-art in ROBDD approximation is represented by heuristic algorithms [18, 21] that prune branches from an ROBDD by checking whether each branch is subsumed by its sibling. These algorithms are syntactic in that they are informed only by the structure of the ROBDD. In this paper, widening is formulated in terms of the prime implicants of the underlying Boolean function. The advantage of this semantic approach is that the widening is not sensitive to the variable ordering, hence improving the predictability of the analysis.

– The widening can be realised in a surprisingly straightforward manner by introducing a cardinality constraint into the algorithm of Coudert and Madre [8] that removes all prime implicants of excessive length. Experimental work suggests that although this widening produces accurate approximations, the running time of our implementation is not significantly worse than state-of-the-art methods [18, 21].

The paper is structured as follows: Section 2 presents the necessary preliminaries. Section 3 specifies a widening for ROBDDs and Sect. 4 details algorithms for realising it. Section 5 presents the experimental results. Finally, Sect. 6 surveys the related work and Sect. 7 concludes.

2 Preliminaries

2.1 Boolean Functions

A Boolean function is a mapping $f : Bool^n \to Bool$ where $Bool = \{0, 1\}$ that is conventionally written as a propositional formula defined over a totally ordered set of propositional variables $X = \{x_1, \ldots, x_n\}$. For instance, $x_1 \vee x_2$ represents the dyadic function $\{\langle 0, 0\rangle \mapsto 0, \langle 0, 1\rangle \mapsto 1, \langle 1, 0\rangle \mapsto 1, \langle 1, 1\rangle \mapsto 1\}$. The set of propositional formulae over X is denoted $Bool_X$ and henceforth functions and formulae will be used interchangeably. We define the set of models of a Boolean function f as the mapping $model_X(f) : Bool_X \to \wp(Bool^n)$ such that $model_X(f) = \{\langle b_1, \ldots, b_n\rangle \mid f(b_1, \ldots, b_n) = 1\}$ where $\wp$ denotes the power-set operator. For example, if $X = \{x_1, x_2, x_3\}$ then $model_X(x_1 \wedge (x_2 \to x_3)) = \{\langle 1, 0, 0\rangle, \langle 1, 0, 1\rangle, \langle 1, 1, 1\rangle\}$. One Boolean function f_1 entails another f_2, denoted $f_1 \models f_2$ iff $model_X(f_1) \subseteq model_X(f_2)$. The structure $\langle Bool_X, \models, \vee, \wedge, 0, 1\rangle$ is a finite lattice where 0 and 1 abbreviate the Boolean functions $\lambda\boldsymbol{b}.0$ and $\lambda\boldsymbol{b}.1$ respectively and $\boldsymbol{b} \in Bool^n$. A chain of Boolean functions C is a set $C \subseteq Bool_X$ such that either $f \models f'$ or $f' \models f$ for all $f, f' \in C$. An anti-chain of Boolean functions A is a set $A \subseteq Bool_X$ such that $f \not\models f$ or $f = f'$ for all $f, f' \in A$. The Shannon co-factor of a Boolean function f w.r.t. a variable x_i and a Boolean constant b is defined by $f|_{x_i \leftarrow b} = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$. Finally, we denote existential quantification w.r.t. a variable x_i by $\exists_{x_i}(f)$ which can be computed using Schröder elimination, that is, by $\exists_{x_i}(f) = f|_{x_i \leftarrow 0} \vee f|_{x_i \leftarrow 1}$.

A cube p is a Boolean function of the form $(\wedge_{y \in Y} y) \wedge (\wedge_{z \in Z} \neg z)$ such that $Y \cup Z \subseteq X$ and $Y \cap Z = \emptyset$ where Y, Z are sets of variables; moreover, the length of p is denoted $\|p\|$ and defined by $\|p\| = \|Y\| + \|Z\|$. An implicant p of a Boolean function f is a cube p such that $p \models f$. The Boolean function 1 is the cube obtained by putting $Y = Z = \emptyset$. A prime implicant p of a Boolean function f is an implicant p of f such that there exists no other implicant p' of f where $p \models p'$ and $p' \neq p$. Let $primes(f)$ denote the set of prime implicants of the Boolean function f. To illustrate, consider $f = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_3 \vee x_4)$ and $p = (\neg x_1 \wedge \neg x_3)$. Observe that $p \models f$ and therefore p is a implicant of f. Further, suppose $p \models p'$ and $p \neq p'$. Then $p' = \neg x_1$ or $p' = \neg x_3$ and, in either case, $p' \not\models f$. Hence p is a prime implicant of f. In fact, $primes(f) = \{\neg x_1 \wedge \neg x_3, \neg x_2 \wedge \neg x_3, \neg x_1 \wedge x_4\}$. Finally, observe $primes(1) = \{1\}$ and $primes(0) = \emptyset$.

2.2 Binary Decision Diagrams

A Binary Decision Diagram (BDD) [2] is a rooted directed acyclic graph where each internal node is labelled with a variable x_i. Each internal node has one successor node connected via an edge labelled 0, and another successor connected via an edge labelled 1. An external (leaf) node is represented by one of two nodes labelled with the Boolean constants 0 or 1. The Boolean function represented by a BDD can be evaluated for a given variable assignment by traversing the graph from the root, taking the 1 edge at a node when the variable is assigned to 1 and the 0 edge when the variable is assigned to 0. The external node reached in this traversal indicates the value of the Boolean function for the assignment. Observe that each sub-BDD of a BDD also itself represents a Boolean function.

An ROBDD is a BDD that obeys the following restrictions to obtain a canonical representation and thereby permit constant-time equivalence checks. Firstly, the label of a node x_i is always less than the label x_j of any internal node immediately reachable via its successors, that is, $i < j$. Secondly, there can exist no sub-ROBDD that is rooted at a node labelled with x_i that represents the function f such that $f|_{x_i \leftarrow 0} = f|_{x_i \leftarrow 1}$. Thirdly, there are no two nodes labelled with the same variable that have identical successor nodes.

3 Specification of the widening

To decouple the widening from implementation concerns, we first specify how to widen Boolean functions for both space and time using prime implicants.

3.1 Widening for space

The ROBDD approximation algorithms of Shiple [21] and Ravi *et al* [18] seek to improve the density of an ROBDD which is defined as the ratio of minterms in the represented function to the number of nodes in the representing ROBDD. Both algorithms identify the non-dense sub-ROBDDs within a ROBDD and substitute them with other sub-ROBDDs which are denser and yet possess more models. Ultimately this culminates in a dense upper-approximation. Although this approach is well-intended, density comparisons and ROBDD restructuring is limited to those sub-ROBDDs that actually arise in the ROBDD whose presence, in turn, depends on the variable ordering. Our thesis is that prime implicants are natural variable order-independent candidates for reasoning about density. To illustrate this, consider the set of implicants $S = \{p \mid p \models f\}$ of a function f. Any $S' \subseteq S$ is a sound under-approximation of f in the sense that $\vee S' \models f$ yet different S', even of the same size, can yield better approximations. For instance, consider an implicant $p \in S$ and a prime implicant p' strictly contained within it, that is, $p \models p'$ and $p \neq p'$. Then $\|p'\| < \|p\|$. Hence p' contributes $2^{n-\|p'\|}$ minterms to f whereas p contributes only $2^{n-\|p\|}$. Thus p' is a better candidate for inclusion in S' than p. Moreover, since p' is shorter than p, it is likely to contribute a shorter path in an ROBDD that represents $\vee S'$. The following family of widening operators draw together these ideas to compute a sound over-approximation by combining negation with systematic under-approximation.

Definition 1. The family of operators $\nabla_k : Bool_X \to Bool_X$ where $k \in \mathbb{N} \cup \{0\}$ are defined by $\nabla_k(f) = \bigwedge\{\neg p \mid p \in primes(\neg f) \wedge \|p\| \leq k\}$

The proposition asserts that ∇_k is anti-monotonic in its parameter k and hence ∇_k is uniformly more precise than ∇_{k-1}. Furthermore, in the limit, $\nabla_k(f)$ converges onto f from above. The widening is also monotonic in its argument f.

Proposition 1. Suppose $\|X\| = n$. Then

- If $f \in Bool_X$ then $f = \nabla_n(f) \models \nabla_{n-1}(f) \models \ldots \models \nabla_0(f) = 1$
- If $f, f' \in Bool_X$, $f \models f'$ and $0 \leq k \leq n$ then $\nabla_k(f) \models \nabla_k(f')$.

Proof.

- Since $f = \bigvee primes(f)$ (Blake canonical form [10]), $f = \nabla_n(f)$. Let $0 \leq k < n$. Note $\{\neg p \mid p \in primes(\neg f) \wedge \|p\| \leq k\} \subseteq \{\neg p \mid p \in primes(\neg f) \wedge \|p\| \leq k+1\}$, hence $\nabla_{k+1}(f) \models \nabla_k(f)$. Finally observe $\nabla_0(f) = \wedge\emptyset = 1$ as required.
- Let $0 \leq k \leq n$, $p' \in primes(\neg f')$ and $\|p'\| \leq k$. Then $p' \models \neg f' \models \neg f$. There exists $p \in primes(\neg f)$ such that $p' \models p$. Thus $\|p\| \leq \|p'\| \leq k$. Since $p' \models p$, $\neg p \models \neg p'$, hence $\nabla_k(f) = \bigwedge\{\neg p \mid p \in primes(\neg f) \wedge \|p\| \leq k\} \models \neg p'$. Therefore $\nabla_k(f) \models \nabla_k(f')$ as required. $\square$

3.2 Widening for time

To explain the role of implications in widening for time, consider a chain of functions $\{f_1, f_2, \ldots\} \subseteq Bool_X$ where $f_{i+1} = F(f_i)$ and $F : Bool_X \to Bool_X$ is a monotonic operator. The problem is to extract an invariant from that chain, that is, find a function g such that $f_i \models g$ for all f_i. Such an invariant can be found, whilst applying F only a bounded number of times, by constructing a set of m Boolean functions $S_1 = \{g_1, \ldots, g_m\}$ such that $f_1 \models g_i$ for all g_i. This set is then iteratively pruned until stability is reached. This is realised by constructing $S_{i+1} = \{g \in S_i \mid F(\wedge S_i) \models g\}$. By construction $f_1 \models \wedge S_1$ and, because F is monotonic, it follows by induction that $f_i \models \wedge S_i$. If S_l denotes the limit, that is $S_l = S_{l+1}$, then $f_i \models \wedge S_l$ for all f_i, hence $\wedge S_l$ is an invariant. The key point about this construction is that F is applied at most m iterations rather than possibly $2^{\|X\|}$ times. This gives a performance guarantee and a parameter m that can be increased (if necessary) to improve precision. This merely leaves the problem of constructing S_1.

An uninformed approach to computing S_1 is to extract m arbitrary implicants of $\neg f_1$, that is, $p \models \neg f_1$. Then each $\neg p$ is a clause of f_1. However, consider a prime implicant p' of $\neg f_1$ such that $p \models p'$. Then $\neg p' \models \neg p$, therefore substituting a prime implicant p' for p we obtain a more accurate initial $\wedge S_1$, without increasing its size. This motivates constructing S_1 from prime implicants. Furthermore, consider two prime implicants p and p' such that $\|p'\| < \|p\|$. Then p' is a more propitious candidate for inclusion in S_1 since the clause $\neg p'$ possesses fewer minterms than $\neg p$ which motivates a greedy approach to constructing S_1 in terms of prime implicants of minimal length.

One may wonder whether a bound k on the length of prime implicants, induces a bound on the number m of primes, and hence a bound on the number of iterates. A straightforward relationship between k and m follows from the observation that there are ${}^nC_1 2^1, {}^nC_2 2^2, \ldots, {}^nC_k 2^k$ different cubes of length $1, 2, \ldots, k$ respectively where ${}^iC_j = \frac{i!}{(i-j)!j!}$. Hence a bound on m is $\min(\{2^n, \sum_{i=1}^{k} {}^nC_k 2^k\})$ where $n = \|X\|$. However, by adapting an argument relating to anti-chains of implicants [5, proof of Theorem 2.2], the following tighter bound can be obtained:

Proposition 2. $\|\{p \in \mathit{primes}(f) \mid \|p\| \leq k\}\| \leq \max(\{{}^nC_1 2^1, \ldots, {}^nC_k 2^k\})$

Proof. Let $f \in \mathit{Bool}_X$, $X = \{x_1, \ldots, x_n\}$, C denote the set of cubes over X and $P = \{p \in \mathit{primes}(f) \mid \|p\| \leq k\}$. P is anti-chain of Bool_X and also C. It has been shown [14] that in a poset such as $\langle C, \models\rangle$, there exists a maximal anti-chain which is invariant under any isomorphism of C. Let A be such an anti-chain. Now let $c, c' \in A$ such that $\|c\| = \|c'\|$ and consider a mapping $F : Y \to Y$ where $Y = \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ such that $F(x_i) = \neg F(\neg x_i)$. Suppose that $F(c) = c'$ where F is extended from Y to C in the natural way. Since F is an automorphism, it follows that $c' \in A$, hence $\{c'' \in C \mid |c| = |c''|\} \subseteq A$. Since A is an anti-chain, then if $p \in C$ and $\|p\| < \|c\|$ then there exists $c'' \in A$ such that $p \models c''$ and $p \neq c''$. Hence $p \notin A$. Similarly, if $\|p\| > \|c\|$ then $p \notin A$. Therefore $\|A\| = {}^nC_{\|c\|} 2^{\|c\|}$ and, since $\|P\| \leq \|A\|$, the result follows. $\square$

Whenever $3k \leq 2(n+1)$ the above bound on m collapses to ${}^nC_k 2^k$. This follows since ${}^nC_1 2^1 \leq \ldots \leq {}^nC_{k-1} 2^{k-1} \leq {}^nC_k 2^k$ iff $\frac{1}{(n-k+1)} \leq \frac{2}{k}$ iff $3k \leq 2(n+1)$. Because this bound is so conservative, a more pragmatic tactic is needed for generating the shortest m prime implicants. One such tactic is to compute all prime implicants of length 1 for f_1, then all primes whose length does not exceed 2, then all primes whose length does not exceed 3 *etc*, until m prime implicants are discovered. The following section presents new algorithm that is designed for solving this specific form of the prime implicant enumeration problem.

4 Implementation of the widening

The complexity of finding the shortest prime implicant given a DNF formula over n variables is in $GC(\log^2 n, \mathit{coNP})$-complete [24], hence at least as hard as coNP, and therefore one would expect the widening to be completely intractable. However, Coudert and Madre [8] give an elegant algorithm for computing all the prime implicants of a Boolean function presented as an ROBDD. The primes are, in turn, represented in an ROBDD and hence the complexity of prime enumeration is not necessarily reliant on the number of implicants but the size of the ROBDD. Alas, a detailed analysis of the complexity of this algorithm has not been forthcoming and it is unknown whether the algorithm is polynomial in the size of the input ROBDD [7]. Furthermore, it remains unclear how the results of Umans [24] relate to the complexity of this algorithm.

This section proposes a refinement to the algorithm of Coudert and Madre [8] that enumerates all primes implicants whose length does not exceed k. This refined algorithm can be applied iteratively to find a shortest prime implicant and thus is unlikely to be polynomial. The essence of the Coudert and Madre [8] scheme is a transformation that maps an ROBDD representing f over the variables X to another representing a function f' over the variables $o_1, s_1, \ldots, o_n, s_n$ where $n = \|X\|$. The idea is that any implicant p' of f' can be reinterpreted as a prime implicant of f in the sense that p is a prime implicant of f whenever:

$$p = (\wedge\{x_i \mid p' \models o_i \wedge p' \models s_i\}) \wedge (\wedge\{\neg x_i \mid p' \models o_i \wedge p' \models \neg s_i\})$$

The intuition is that o_i indicates whether the variable x_i occurs within a prime and s_i encodes the polarity of that occurrence. Coudert and Madre [8] present an ROBDD transformation that recursively builds f' from f. Our new insight is that it is possible to build f' from f whilst enforcing the cardinality constraint $\sum_{i=1}^{n} o_i \leq k$. The following algorithm builds toward the refined algorithm by generating an ROBDD which expresses the cardinality constraint. The constraint is realised as a cascade of n full-adders that together output the sum that is expressed in $\lceil \lg(n) \rceil$ bits. These bits are then constrained so as to not exceed k.

Algorithm 1 `CONSTRAIN`(k)

for $i \leftarrow 1$ **to** $\lceil \lg n \rceil$ **do**
  $sum[i] \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  $c \leftarrow o_i$
  for $j \leftarrow 1$ **to** $\lceil \lg n \rceil$ **do**
    $c' \leftarrow c \wedge sum[j]$
    $sum[j] \leftarrow sum[j] \oplus c$
    $c \leftarrow c'$
$f \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  $f \leftarrow (\neg sum[i] \wedge k[i]) \vee ((sum[i] \leftrightarrow k[i]) \wedge f)$
return f

In the above algorithm, the bound k is represented as an array of $\lceil \lg n \rceil$ bits $k[i]$ such that $k = k[1] + 2k[2] + \ldots + 2^{\lceil \lg n \rceil} k[\lceil \lg n \rceil]$. The first loop initialises the elements of the temporary array $sum[i]$ to $false$. The second loop iteratively calculates $o_1 + \ldots + o_n$ and stores the result in the temporary array sum. The i^{th} iteration of the loop initialises the carry c to be o_i and then proceeds to add the carry into the sum that has accumulated thus far. The formula $sum[j] \oplus c$ merely denotes the exclusive-or of the j^{th} bit of sum with the carry c. The third loop constrains the array sum to not exceed the k vector. Algorithm 2 details how this constraint can be integrated in the algorithm of Coudert and Madre [8]. Because of reasons of space, those readers who wish to follow the structure of the algorithm and the underlying meta-product construct are referred to [8].

Algorithm 2 PRIMESLEQ(f, k)

$x_i \leftarrow$ `var`(f)
$g \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 0} \wedge f|_{x_i \leftarrow 1}, k)$
$g' \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 1}, k) \wedge \neg g$
$g'' \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 0}, k) \wedge \neg g$
return $((\neg o_i \wedge g) \vee (o_i \wedge s_i \wedge g') \vee (o_i \wedge \neg s_i \wedge g'')) \wedge$ CONSTRAIN(k)

Algorithm 2 repeatedly imposes the cardinality constraint which trims the size of all intermediate ROBDDs. The astute reader will notice that each call to `PRIMESLEQ` operates on a sub-ROBDD that is only defined over $\{x_j, \ldots, x_n\}$. However, `CONSTRAIN`(k) imposes a constraint over $\{x_1, \ldots, x_n\}$. This is no error since $\sum_{i=1}^{n} o_i \leq k$ entails $\sum_{i=j}^{n} o_i \leq k$ and therefore it is not necessary to manufacture a different cardinality constraint for each level in the ROBDD.

When widening for time, it is necessary to extract m primes from the transformed ROBDD. This can be accomplished by a partial, depth-first traversal that sweeps the ROBDD until m primes have been retrieved. When widening for space, an ROBDD over-approximation is required. The following algorithm details how this can be constructed by applying existential quantification:

Algorithm 3 PRIMES2BDD(f)

for $i \leftarrow 0$ **to** n **do**
  $f' \leftarrow \exists_{s_i}(\exists_{o_i}(f \wedge (o_i \rightarrow (x_i \leftrightarrow s_i))))$
  $f \leftarrow f'$
return f

5 Experimental Results

To assess the precision and tractability of the widening, it was implemented within the CUDD [22] Decision Diagram package. This package supports the algorithms of Shiple [21] and Ravi *et al.* [18] which, following the CUDD naming scheme, will henceforth be referred to as `bddOverApprox` and `remapOverApprox` respectively. Table 1 presents details of the Boolean functions, drawn from the MCNC and ISCAS benchmark circuits, used to assess the widening. For ease of reference, all Boolean functions are labelled with a numeric identifier. The second and third columns give the circuit name and specific output number taken from the circuits; outputs were selected so as to evaluate the widening on ROBDDs with varying size. The fourth, fifth, sixth and seventh columns respectively give the number of variables, number of ROBDD nodes, the number of minterms of the Boolean function represented by the ROBDD and the density of the ROBDD. All experiments were performed on an UltraSPARC IIIi 900MHz based system, equipped with 16GB RAM, running the Solaris 9 Operating System, and using `getrusage` to calibrate CPU usage in seconds.

Table 1. Benchmark formulae

ID	Circuit	#	$\|\mathbf{X}\|$	size	minterms	density
1.	pair	177	51	26253	1.86×10^{14}	7.08×10^{9}
2.		182	53	33190	8.12×10^{14}	2.45×10^{10}
3.	mm9b	420	31	94328	1.61×10^{9}	1.71×10^{4}
4.		421	31	96875	1.62×10^{9}	1.67×10^{4}
5.	s9234	288	76	655192	3.59×10^{22}	5.48×10^{16}
6.		488	75	1304371	1.95×10^{22}	1.49×10^{16}
7.	rot	149	53	1315	5.18×10^{15}	3.94×10^{12}
8.		172	55	1700	1.08×10^{16}	6.35×10^{12}

5.1 Our Method

The topmost graph of Fig. 1 presents the time required to apply Algorithm 2 and then Algorithm 3 to the benchmarks for various k. (Note that this time is dominated by the cost of applying Algorithm 2 and therefore the times reported in the table closely tally with the times required to apply Algorithm 2 and then walk the ROBDD to extract a bounded number of primes). Interestingly, Coudert and Madre [8] suggest that "[their] procedures have costs that are independent of the sizes of [the prime] sets", "since there is no relation between the size of a set and the size of the [ROBDD] that denotes it". However, this does not square with our results which suggest that the size of the ROBDDs depends, at least to some extent, on the number of primes that it represents. This is witnessed by the sharp increase in runtime that occurs for some circuits as k increases. However, the crucial point is not that the runtime spikes, but the degree of precision achieved before the escalation in complexity. To this end, the middle graph plots the ratio of minterms of the original Boolean function against that of the approximation for increasing values of k. Observe that the quality of the approximation rapidly converges onto 1 as k increases. This suggests the tactic of incrementally increasing k until either the precision is acceptable or a timeout is reached. Applying this tactic achieves precision rates of $70, 80$, and 90% yielding runtimes of less than $5, 20$ and 60 seconds respectively. On the other hand, repeatedly incrementing k until the accumulated runtime exceeds 30 seconds, achieves minterm precision rates for benchmarks 1–8 of $99, 99, 99, 99, 99, 92, 96, 95\%$ respectively. This realises an anytime approach to prime generation and ROBDD approximation in which the quality of the result is limited only be the quantity of resource available. Incrementing k until at least 1024 prime implicants are found (which if anything is rather high for the purposes of analysis), requires the following values of k: 5, 5, 7, 7, 5, 6, 7, 7.

It should be noted that these figures are, if anything, rather pessimistic for many types of program analysis. For example, in the context of groundness analysis that is widely used in logic programming, it has been observed that the vast majority of clauses that arise during analysis are very small in length [13].

This implies that widening with small k is unlikely to have any discernable impact on the overall precision.

The value of an approximation algorithm has traditionally been reported in terms of density [18, 21] which gives an indication as to the compactness of the approximating ROBDD. The lower graph thus reports how the density varies with k. By comparing the densities reported in Table 1 against those presented in the graph, it can be seen that the widening can significantly improve on the density of the original ROBDD.

5.2 Comparison Against Existing Methods

Table 2 summaries the results obtained by exercising the `bddOverApprox` and `remapOverApprox` algorithms on the circuits in our benchmark suit. The table is partitioned horizontally, into three groups of rows according to whether the `bddOverApprox` algorithm, `remapOverApprox` algorithm, or the widening algorithm proposed in this paper was applied. The second and third columns give the size of the approximating ROBDD and the number of minterms in its underlying Boolean function. The fourth and fifth columns detail the ratio of these values with respect to the size and number of minterms in the original ROBDD (as given in Table 1). The `bddOverApprox` and `remapOverApprox` algorithms are parameterised by a quality parameter $q \in [0, 1]$, that specifies the minimal acceptable density improvement. That is, these algorithms ensure that the new density d' satisfies $q \geq d/d'$ where d is the density of the original ROBDD. As Shiple himself says [21], "The `bddUnderApprox` method is highly sensitive to the [quality] parameter". Added to this, there is no clear way to choose q so as to obtain a desired reduction in ROBDD size.

For purposes of comparison, we chose to reduce the size of an ROBDD by at least 50%, but ideally not significantly more than 50% (it was the desire to solve this particular analysis problem that motivated this study). Both `bddOverApprox` and `remapOverApprox` were called repeatedly under the bisection algorithm to search for a quality value that yielded an acceptable reduction in size. The algorithm terminated when the difference between the high and lower quality bounds was less than 0.01. The notes column gives the particular quality values that achieved the best ROBDD approximation and the time column presents the total time required to call bisection which, of course, was dominated by the time to approximate the ROBDDs. Despite the systematic use of bisection, the reduction in ROBDD size was often significantly more than 50%. This was due to the ROBDD collapsing at certain quality thresholds.

The lower rows of the Table 2 summarise the results of incrementing k until a space reduction of at least 50% was obtained. The notes column gives the required values of k and the cumulative execution time. Observe that the minterm ratios thus obtained compare favourably with those derived using `bddOverApprox` and `remapOverApprox` whilst the overall execution time is also reduced. Note that other variable orderings may give different results for the `bddOverApprox` and `remapOverApprox`. (As a sanity check, the widening was tested to verify that it delivered the same approximations under different variable orderings.)

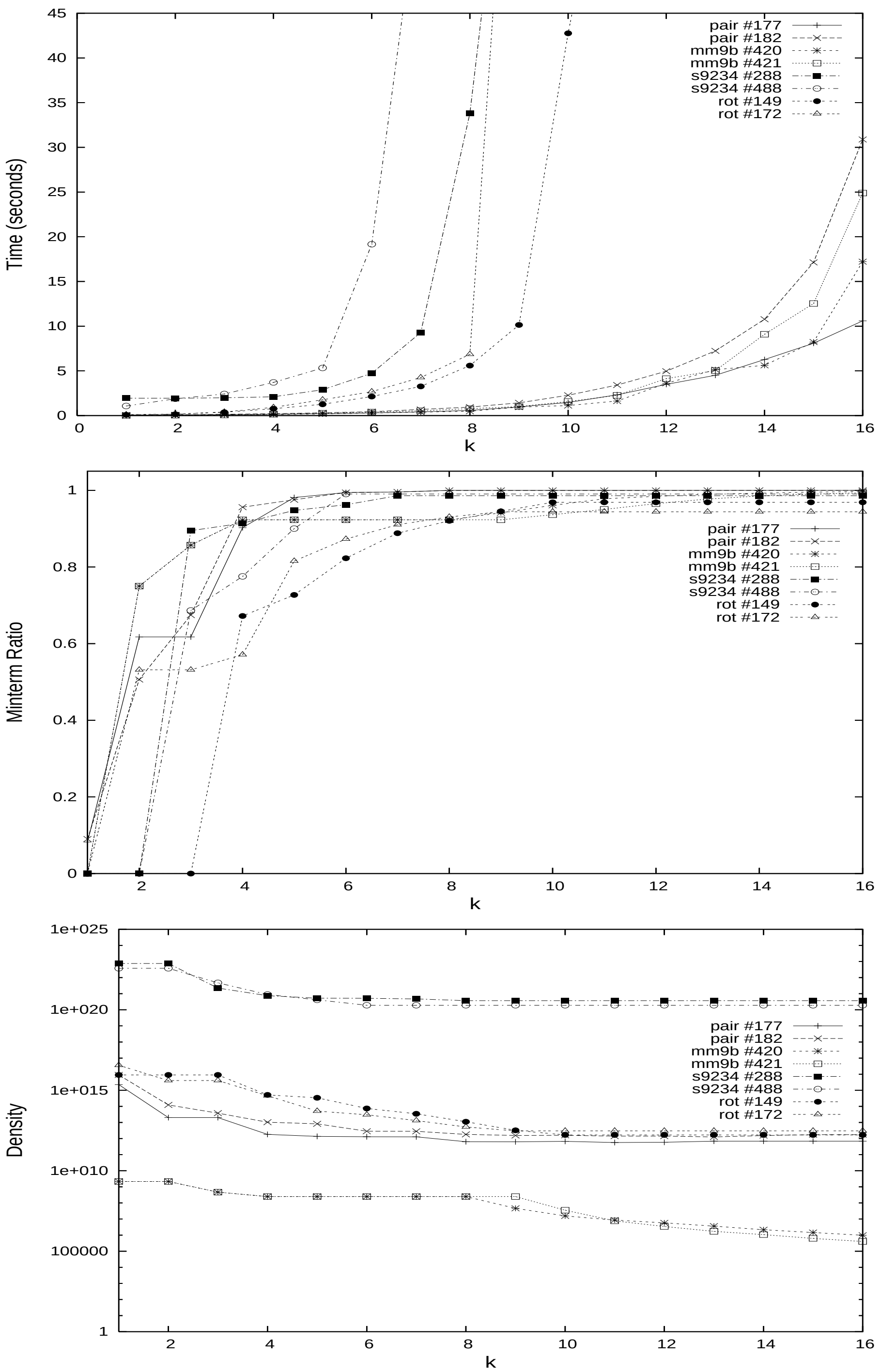

Fig. 1. Time, Minterm ratio and Density against k

Table 2. Comparison of approximation

ID		Approximation size	Approximation minterms	Ratios size	Ratios minterms	Time	Notes
[18]	1.	8382	3.40×10^{14}	0.32	1.83	4.61	**q:** 0.94
	2.	9711	1.47×10^{15}	0.29	1.81	6.32	**q:** 0.84
	3.	933	1.88×10^{9}	0.01	1.16	10.85	**q:** 0.75
	4.	722	1.88×10^{9}	0.01	1.16	11.96	**q:** 0.84
	5.	15	5.68×10^{22}	0.01	1.58	1086.12	**q:** 0.88
	6.	11	2.89×10^{22}	0.01	1.49	2321.68	**q:** 0.92
	7.	91	7.30×10^{15}	0.07	1.41	1.13	**q:** 0.96
	8.	838	2.96×10^{16}	0.49	2.75	1.50	**q:** 0.98
[21]	1.	8385	1.72×10^{15}	0.32	10.85	4.86	**q:** 0.92
	2.	9714	8.06×10^{15}	0.29	9.93	6.35	**q:** 0.81
	3.	933	1.88×10^{9}	0.01	1.16	12.39	**q:** 0.75
	4.	722	1.88×10^{9}	0.01	1.16	13.10	**q:** 0.84
	5.	15	5.68×10^{22}	0.01	1.58	1057.62	**q:** 0.87
	6.	11	2.89×10^{22}	0.01	1.49	2562.30	**q:** 0.92
	7.	168	8.10×10^{15}	0.13	1.56	1.25	**q:** 0.92
	8.	837	1.73×10^{16}	0.49	1.60	1.67	**q:** 0.94
§ 3	1.	11027	2.06×10^{14}	0.42	1.11	0.58	**k:** 5
	2.	7301	8.32×10^{14}	0.22	1.03	0.85	**k:** 6
	3.	44334	1.68×10^{9}	0.47	1.02	6.38	**k:** 12
	4.	39718	1.69×10^{9}	0.41	1.05	8.19	**k:** 11
	5.	75	3.64×10^{22}	0.01	1.01	20.36	**k:** 7
	6.	103	1.96×10^{22}	0.01	1.01	47.53	**k:** 6
	7.	289	6.29×10^{15}	0.22	1.21	0.88	**k:** 7
	8.	527	1.09×10^{16}	0.31	1.01	1.66	**k:** 7

6 Related Work

Quite apart from the heuristic algorithms of Shiple [21] and Ravi *et al.* [18] that both reside in $O(\|G\|^2)$ where $\|G\|$ is the number of nodes in the ROBDD, other less well-known widenings have been proposed in the literature. Mauborgne [15] shows how to perform strictness analysis with an ROBDD variant referred to as a typed decision graph (TDG). Mauborgne advocates widening TDGs for space, using an operator $\nabla(l, f)$ that takes, as input, a TDG that encodes a Boolean function f and returns, as output, a TDG g with at most l nodes such that $f \models g$. The first widening he proposes is in $O(\|G\|^4)$ where $\|G\|$ is the number of nodes in the TDG. To improve efficiency, Mauborgne suggests a second widening that resembles those of Shiple and Ravi *et al.* This algorithm computes the TDGs $f_1, \ldots, f_n$ obtained by replacing each node i with 1. The f_i are filtered to remove those TDGs whose size exceed $\|G\|/2$. Of the remaining f_i, an f_{max} is selected which "gives best results" and the widening is reapplied to f_{max} if its TDG contains more than l nodes. More recently, Schachte and Søndergaard [20] have presented elegant ROBDD algorithms for approximating functions to various sub-domains of Boolean formulae. Although complexity theoretic issues still remain, these algorithms are potentially useful as widenings.

Algorithm 4 EquivVars(f)

```
x_i ← var(f)
if f|_{x_i←0} = true then
   return ⟨i, 0, 1⟩ :: ε
if f|_{x_i←1} = true then
   return ⟨i, 1, 0⟩ :: ε
if f|_{x_i←0} = false then
   return ⟨i, 1, 0⟩ ::EquivVars(f|_{x_i←1})
if f|_{x_i←1} = false then
   return ⟨i, 0, 1⟩ ::EquivVars(f|_{x_i←0})
v_1 ← ⟨i, 0, 1⟩ ::EquivVars(f|_{x_i←0})
v_2 ← ⟨i, 1, 0⟩ ::EquivVars(f|_{x_i←1})
return anti_unify(v_1, v_2)
```

The widening presented in this paper relies upon the generation of prime implicants. This problem was first addressed by Quine [17] and, since then, there has been much interest in developing efficient prime implicant enumeration algorithms (interested readers should consult [23] for a detailed history of the problem and known algorithms). Interestingly, the ROBDD literature already suggests an approach to widening ROBDDs that is based on prime implicants (albeit of a restricted form). Bagnara and Schachte [1] propose an $O(n^2\|G\|)$ ROBDD algorithm for finding all pairs $x, y \in X$ such that $\neg(x \leftrightarrow y) \models f$ where $n = \|X\|$. The formula $\neg(x \leftrightarrow y) = (x \wedge \neg y) \vee (\neg x \wedge y)$ is actually a quadratic prime implicant and this hints at an ROBDD widening. The algorithm sketched in Algorithm 4 applies Plotkin's anti-unification algorithm [16] to detect all quadratic prime implicants of the form $\neg(x \leftrightarrow y)$ and $(x \leftrightarrow y)$ whilst reducing the complexity from $O(n^2\|G\|)$ [1] to $O(n \lg n\|G\|)$ where n is the number of variables in the ROBDD. A run of the algorithm is illustrated for an ROBDD representing $f = (x_1 \wedge (x_2 \vee x_3)) \wedge (x_2 \leftrightarrow x_4) \wedge (x_2 \leftrightarrow \neg x_5)$. The intuition behind the algorithm is that lists such as $\langle 2, 1, 0\rangle : \langle 4, 1, 0\rangle : \langle 5, 0, 1\rangle$ and $\langle 2, 0, 1\rangle : \langle 3, 1, 0\rangle : \langle 4, 0, 1\rangle : \langle 5, 1, 0\rangle$ represent $x_2 \wedge x_4 \wedge \neg x_5$ and $\neg x_2 \wedge x_3 \wedge \neg x_4 \wedge x_5$. Anti-unification can then be applied to these lists to obtain $\langle 2, A, B\rangle : \langle 4, A, B\rangle : \langle 5, B, A\rangle$ which encodes $(x_2 \leftrightarrow x_4) \wedge (x_4 \leftrightarrow \neg x_5)$ where A and B are special symbols that represent simple dependencies between variables. The algorithm finally returns a list that represents $x_1 \wedge (x_2 \leftrightarrow x_4) \wedge (x_4 \leftrightarrow \neg x_5)$ which, indeed, is a safe upper approximation of f. By adapting an argument given in [13], it can be shown that this algorithm returns an upper-approximation in a sub-class of Boolean formulae that admits chains of maximal length $2n$. Although not as general as the approach proposed in this paper, this algorithm offers a compromise between efficiency and generality that might suit some analyses [13].

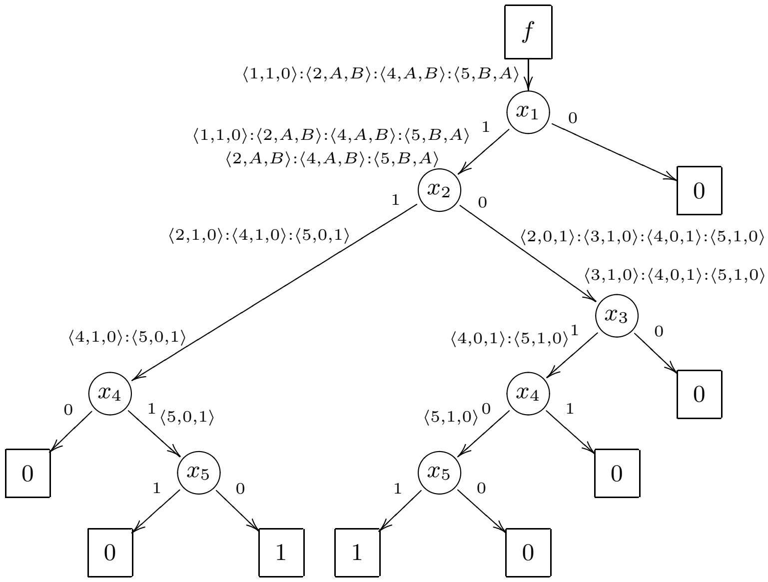

Fig. 2. Algorithm 4 when applied to $f = (x_1 \wedge (x_2 \vee x_3)) \wedge (x_2 \leftrightarrow x_4) \wedge (x_2 \leftrightarrow \neg x_5)$

7 Conclusions

The paper has proposed a new widening for ROBDDs and an algorithm for realising it. The widening can be used to either bound the number of times that an ROBDD is updated in an iterative analysis or approximate an ROBDD with another that has a more space-efficient representation. Empirical evidence suggests that the widening is potentially useful and surprisingly tractable.

Acknowledgements We thank Jacob Howe, Laurent Mauborgne, Axel Simon, Peter Schachte and Harald Søndergaard for useful discussions. This work was funded by EPSRC Grant EP/C015517 and the British Council Grant PN 05.021.

References

[1] R. Bagnara and P. Schachte. Factorizing Equivalent Variable Pairs in ROBDD-Based Implementations of *Pos*. In *Algebraic Methodology and Software Technology*, volume 1548 of *LNCS*, pages 471–485. Springer, 1999.

[2] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[3] R. E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Inform Comput*, 98(2):142–170, 1992.

[5] A. K. Chandra and G. Markowsky. On The Number of Prime Implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

[6] M. Codish. Worst-Case Groundness Analysis using Positive Boolean Functions. *J Logic Program*, 41(1):125–128, 1999.

[7] O. Coudert. Two Open Questions On ROBDDs and Prime Implicants. http://www.informatik.uni-trier.de/Design_and_Test/abstract30.html.

[8] O. Coudert and J. C. Madre. Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In *Proceedings of the Design Automation Conference*, pages 36–39. IEEE, 1992.

[9] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[10] Y. Crama and P. L. Hammer. *Boolean Functions*. To appear.

[11] C. Fecht. *Abstrakte Interpretation logischer Programme: Theorie, Implementierung, Generierung*. PhD thesis, Universität des Saarlandes, 1997.

[12] J. P. Gallagher, K. S. Henriksen, and G. Banda. Techniques for Scaling Up Analyses Based on Pre-interpretations. In *International Conference on Logic Programming*, volume 3668 of *LNCS*, pages 280–296. Springer, 2005.

[13] A. Heaton, M. Abo-Zaed, M. Codish, and A. King. A Simple Polynomial Groundness Analysis for Logic Programs. *J Logic Program*, 45:143–156, 2000.

[14] D. J. Kleitman, M. Edelberg, and D. Lubell. Maximal Sized Antichains in Partial Orders. *Discrete Mathematics*, 1(1):47–53, 1971.

[15] L. Mauborgne. Abstract Interpretation Using Typed Decision Graphs. *Science of Computer Programming*, 31(1):91–112, 1998.

[16] G. Plotkin. A Note on Inductive Generalisation. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[17] W. V. Quine. The Problem of Simplifying Truth Functions. *American Mathematical Monthly*, (52):521–531, 1952.

[18] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and Decomposition of Binary Decision Diagrams. In *Proceedings of the Design Automation Conference*, pages 445–450. IEEE, 1998.

[19] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *International Conference on Computer-Aided Design*, pages 42–47. IEEE, 1993.

[20] P. Schachte and H. Søndergaard. Closure Operators for ROBDDs. In *Proceedings of the Seventh International Conference on Verification, Model Checking and Abstract Interpretation*, volume 3855 of *LNCS*, pages 1–16. Springer, 2006.

[21] T. R. Shiple. *Formal Analysis of Synchronous Circuits*. PhD thesis, University of California at Berkeley, Electronics Research Laboratory, 1996.

[22] F. Somenzi. CUDD Package, Release 2.4.1. http://vlsi.colorado.edu/∼fabio/.

[23] T. Strzemecki. Polynomial-time Algorithms for Generation of Prime Implicants. *ACM Journal of Complexity*, 8(1):37–63, 1992.

[24] C. Umans. On the Complexity and Inapproximability of Shortest Implicant Problems. In *International Colloqium on Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 687–696. Springer, 1999.

[25] J. Whaley and M. S. Lam. Cloning-Based Context-Sensitive Pointer Alias Analysis Using Binary Decision Diagrams. In *Programming Language Design and Implementation*, pages 131–144. ACM Press, 2004.