CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2010, Article 11, pages 1–26
http://cjtcs.cs.uchicago.edu/

SPECIAL ISSUE FOR CATS 2009

An Algorithm for Affine Approximation of Binary Decision Diagrams

Kevin Henshall Peter Schachte Harald Søndergaard
Leigh Whiting

Received: January 14, 2009; revised: July 16, 2009; published: June 22, 2010.

Abstract:

This paper is concerned with the problem of Boolean approximation in the following sense: given a Boolean function class and an arbitrary Boolean function, what is the function's best proxy in the class? Specifically, what is its strongest logical consequence (or *envelope*) in the class of *affine* Boolean functions. We prove various properties of affine Boolean functions and their representation as ROBDDs. Using these properties, we develop an ROBDD algorithm to find the affine envelope of a Boolean function.

1 Introduction

Various classes of Boolean functions play important roles in computer science. The classes of interest include all of the co-clones [54] (such as the Horn and Krom classes) and others. In this paper we focus on the affine class.

For any particular way of representing Boolean functions, and any particular class, a number of algorithmic problems suggest themselves. One is *identification*: How does one decide whether a given function belongs to the class? Another problem is *Boolean approximation*: Given a Boolean function, how does one find its best proxy from the given class; for example, how to find its strongest Horn consequence?

Boolean approximation finds use in diverse areas such as abstract interpretation, circuit verification, and machine learning. The bulk of research in Boolean approximation has arguably sprung from

DOI: 10.4086/cjtcs.2010.011

areas of artificial intelligence. One recurrent quest has been the efficient inference from possibly large propositional formulas, or knowledge-bases. An important approach, which was initially proposed by Selman and Kautz [63], is to query (and perform deductions from) upper and lower approximations of the given formula. By choosing approximations that allow more efficient inference, it is often possible to quickly determine that some logical consequence of the knowledge-base entails the query, and therefore so does the original knowledge-base, avoiding the costly inference from the original. When this fails, it may be possible to quickly show that the query is not entailed by some implicant, and therefore not entailed by the full knowledge-base. Only when both of these fail must the full knowledge-base be used for inference. This approach to deduction is particularly attractive if the knowledge-base is relatively stable (that is, many queries are handled between changes to the knowledge-base), because in that case, the amortised cost of calculating the approximations is small.

In the field of artificial intelligence it is usually assumed that Boolean functions are represented in clausal form, and that approximations are Horn [63, 22]. In this setting, inference from Horn formulas may be exponentially more efficient than from unrestricted formulas. However, it has been noted that there are many other well-understood classes that have computational properties that include some of the attractive properties of the Horn class.

Zanuttini [66, 67] discusses the use of other classes of Boolean functions for approximation and points out that *affine* approximations have certain advantages over Horn approximations, most notably the fact that they do not blow out in size. This is certainly the case when affine functions are represented in the form of modulo-2 congruence equations. The more general sets-of-models representation is also considered by Zanuttini. In this paper, we consider a third, general, representation, namely reduced ordered binary decision diagrams (ROBDDs). We prove some important properties of affine functions and their ROBDD representation. Utilising these properties we design a new ROBDD algorithm for deriving strongest affine consequences (also known as affine envelopes). Schachte and Søndergaard [58, 59] have previously given ROBDD algorithms for finding monotone, Krom, and Horn envelopes, but also noticed that while those algorithms could be expressed as instances of a common scheme, the same scheme did not apply to affine functions. A different, less compositional, approach is needed in this case.

This paper is an extended version of [35] and it proceeds as follows. In Section 2 we recapitulate the definition of the Boolean affine class, and we establish some of its important properties. We also briefly introduce ROBDDs, but mainly to fix our notation, as we assume that the reader is familiar with Boolean functions and their representation as decision diagrams. Section 3 recalls the model-based affine envelope algorithm, and develops an ROBDD-based algorithm, whose correctness rests on results established in Section 2.2. Section 4 describes our testing methodology, including our algorithm for generating random ROBDDs, and presents our results. Section 5 discusses related work and applications, and concludes.

2 Boolean Approximation and ROBDDs

We use ROBDDs [8, 10] to represent Boolean functions. Our choice of ROBDDs as a data structure is due to the fact that it offers a canonical representation for any Boolean function—a representation that is highly suitable for inductive reasoning.

Zanuttini [66] suggests using modulo 2 congruence equations to represent affine Boolean functions, and proves a polynomial complexity bound for computing affine envelopes in this representation. However, using a specialised representation has a cost in implementation complexity where affine and non-affine Boolean functions must be used together. Certainly the algorithm for evaluating whether one ROBDD entails another is straightforward. Similarly, systems which repeatedly construct an affine approximation, manipulate it as a general Boolean function, and then approximate the result again, have much simpler implementations with a single universal representation than with the combination of a specialised affine representation and a universal one. For our purposes, computing envelopes as ROBDDs permits us to use the same representation for approximation to many different Boolean classes. Additionally, ROBDD-based inference is fast, and in particular, checking whether a valuation is a model, or finding a model, of an n-place function given by an ROBDD requires a path traversal of length no more than n.

2.1 Boolean functions

Let $\mathcal{B} = \{\mathit{0}, \mathit{1}\}$ and let $\mathcal{V}$ be a denumerable set of variables. A *valuation* $\mu : \mathcal{V} \to \mathcal{B}$ is a (total) assignment of truth values to the variables in $\mathcal{V}$. Let $\mathcal{I} = \mathcal{V} \to \mathcal{B}$ denote the set of $\mathcal{V}$-valuations. A *partial valuation* $\mu : \mathcal{V} \to \mathcal{B} \cup \{*\}$ assigns truth values to some variables in $\mathcal{V}$, and $*$ to others. Let $\mathcal{I}_{\mathsf{p}} = \mathcal{V} \to \mathcal{B} \cup \{*\}$. We use the notation $\mu[x \mapsto i]$, where $x \in \mathcal{V}$ and $i \in \mathcal{B}$, to denote the valuation μ updated to map x to i, that is,

$$\mu[x \mapsto i](v) = \begin{cases} i & \text{if } v = x \\ \mu(v) & \text{otherwise.} \end{cases}$$

A Boolean function over $\mathcal{V}$ is a function $\varphi : \mathcal{I} \to \mathcal{B}$. We let $\mathbf{B}$ denote the set of all Boolean functions over $\mathcal{V}$. The ordering on $\mathcal{B}$ is the usual: $x \leq y$ iff $x = \mathit{0} \vee y = \mathit{1}$. $\mathbf{B}$ is ordered pointwise, so that the ordering relation corresponds exactly to classical entailment, $\models$. It is convenient to overload the symbols for truth and falsehood. Thus we let $\mathit{1}$ denote the largest element of $\mathbf{B}$ (that is, $\lambda\mu.\mathit{1}$) as well as of $\mathcal{B}$. Similarly $\mathit{0}$ denotes the smallest element of $\mathbf{B}$ (that is, $\lambda\mu.\mathit{0}$) as well as of $\mathcal{B}$. A valuation μ is a *model* for φ, denoted $\mu \models \varphi$, if $\varphi(\mu) = \mathit{1}$. We let $\mathit{models}(\varphi)$ denote the set of models of φ. Conversely, the unique Boolean function that has exactly the set M as models is denoted $\mathit{fn}(M)$. A Boolean function φ is said to be *independent of* a variable x when for all valuations μ, $\mu[x \mapsto 0] \models \varphi$ iff $\mu[x \mapsto 1] \models \varphi$; otherwise it is said to be *dependent* on x.

Existential quantification is defined as follows. Let φ be a Boolean function and $M = \mathit{models}(\varphi)$, then

$$\exists v(\varphi) = \mathit{fn}(\{\mu[v \mapsto \mathit{0}] \mid \mu \in M\} \cup \{\mu[v \mapsto \mathit{1}] \mid \mu \in M\}).$$

Clearly $\exists v(\varphi)$ is independent of v.

In the context of an ordered set of n variables of interest, $x_1, \ldots, x_n$, we may identify with μ the binary sequence $\mathsf{bits}(\mu)$ of length n:

$$\mu(x_1), \ldots, \mu(x_n)$$

which we will write simply as a bit-string of length n. Similarly we may think of, and write, the set of valuations M as a set of bit-strings:

$$\mathsf{bits}(M) = \{\mathsf{bits}(\mu) \mid \mu \in M\}.$$

As it could hardly create confusion, we shall present valuations variously as functions or bitstrings. We denote the *zero valuation*, which maps x_i to $\mathit{0}$ for all $1 \leq i \leq n$, by $\vec{0}$.

We use the Boolean connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $+$ (exclusive or, or "xor"). These connectives operate on Boolean functions, that is, on elements of **B**. Traditionally they are overloaded to also operate on truth values, that is, elements of $\mathcal{B}$. However, we deviate at this point, as the distinction between xor and its "bit-wise" analogue will be critical in what follows. Hence we denote the $\mathcal{B}$ (bit) version by $\oplus$. We extend this to valuations and bit-strings in the natural way:

$$(\mu_1 \oplus \mu_2)(x) = \mu_1(x) \oplus \mu_2(x)$$

and we let $\oplus_3$ denote the "xor of three" operation $\lambda \mu_1 \mu_2 \mu_3 . \mu_1 \oplus \mu_2 \oplus \mu_3$. We follow Zanuttini [66] in further overloading '$\oplus$', using the notation

$$M_\mu = \mu \oplus M = \{\mu \oplus \mu' \mid \mu' \in M\}.$$

We read M_μ as "M translated by μ". The function $t_\mu : \mathbf{B} \to \mathbf{B}$ similarly performs translation of a Boolean function:

$$t_\mu(\varphi) = \mathit{fn}(M_\mu)$$

where $M = \mathit{models}(\varphi)$. Note that for any set M, the function $\lambda \mu . M_\mu$ is an involution: $(M_\mu)_\mu = M$, and hence t_μ is an involution too.

A final overloading results in the following definition. For $\varphi \in \mathbf{B}$, and $\mu \in \mathcal{I}$, let $\varphi \oplus \mu = \mathit{fn}(M_\mu)$ where $M = \mathit{models}(\varphi)$. We also use a distributed version of $\oplus$: $\bigoplus\{\varphi_1, \ldots, \varphi_n\} = \varphi_1 \oplus \cdots \oplus \varphi_n$.

Since we shall make frequent use of existential quantification, it is worth noting that $\exists v$ does not distribute over +, as for example,

$$\exists x(\neg x + (x \wedge \neg y)) = \mathit{1} \quad \neq \quad y = \exists x(\neg x) + \exists x(x \wedge \neg y).$$

However, it is easy to verify that $\exists v(\varphi) + \exists v(\psi) \models \exists v(\varphi + \psi)$ for all functions φ and ψ.

2.2 The affine class

An *affine* function is one whose set of models is closed under pointwise application of $\oplus_3$ [61]. Affine functions have a number of attractive properties, as we shall see. Syntactically, a Boolean function is affine iff it can be written as a conjunction of affine equations

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n = c_0$$

where $c_i \in \{0, 1\}$ for all $i \in \{0, .., n\}$.[1] This is well known, but for completeness we prove it below, as Proposition 2.4.

The affine class contains $\mathit{1}$ and is closed under conjunction. Hence the concept of a unique best affine upper-approximation is well defined, and the function that takes a Boolean function and returns its best affine upper-approximation is an upper closure operator, that is, it is monotone, increasing, and idempotent [51, 65]. For convenience, let us introduce a name for this operator:

[1] In some circles, such as the cryptography/coding community, the term "affine" is used only for a function that can be written $c_1 x_1 + c_2 x_2 + \ldots + c_n x_n + c_0$, with $n \geq 0$ (the latter is what Post [55] called an "alternating" function). The resulting set of "affine" functions is not closed under conjunction. Our more common use agrees with the use in linear algebra, where an affine space is a vector space translated by some vector.

Definition 2.1. Let φ be a Boolean function. The *affine envelope*, *aff*(φ), of φ is defined:

$$\mathit{aff}(\varphi) = \bigwedge\{\psi \mid \varphi \models \psi \text{ and } \psi \text{ is affine}\}.$$

There are numerous other classes of interest, including isotone, antitone, Krom, Horn, contra-dual Horn and all other co-clones [54], k-Horn [21], and k-quasi-Horn functions. For all of these, the concept of an envelope is well-defined, as each class contains *1* and is closed under conjunction.[2]

Zanuttini [66] exploits the close connection between vector spaces and the sets of models of affine functions. A set $S \subseteq \mathcal{B}^k$ of bitstrings is a *vector space* iff $\vec{0} \in S$ and S is closed under $\oplus$. The set of vector spaces also contains *1* and is closed under conjunction (intersection), so the concept of a tightest enclosing vector space, given a set of valuations (or bit vectors), is well defined.

Definition 2.2. Let φ be a Boolean function. The *linear envelope*, *lin*(φ), of φ is defined:

$$\mathit{lin}(\varphi) = \bigwedge\{\psi \mid \varphi \models \psi \text{ and } \mathit{models}(\psi) \text{ is closed under } \oplus\}.$$

Note that by this, $\mathit{aff}(0) = \mathit{lin}(0) = 0$. Also note that for a satisfiable φ, the models of $\mathit{lin}(\varphi)$ form a vector space.

The next proposition suggests how one can simplify the task of doing model-closure under $\oplus_3$.

Proposition 2.3. [66] Given a non-empty set of models M and a valuation $\mu \in M$, M is closed under $\oplus_3$ iff M_μ is a vector space.

Proof: Let μ be an arbitrary element of M. Clearly M_μ contains $\vec{0}$, so the right-hand side of the claim amounts to M_μ being closed under $\oplus$.

For the ‘if’ direction, assume M_μ is closed under $\oplus$ and consider $\mu_1, \mu_2, \mu_3 \in M$. Since $\mu \oplus \mu_2$ and $\mu \oplus \mu_3$ are in M_μ, so is $\mu_2 \oplus \mu_3$. And since furthermore $\mu \oplus \mu_1$ is in M_μ, so is $\mu \oplus \mu_1 \oplus \mu_2 \oplus \mu_3$. Hence $\mu_1 \oplus \mu_2 \oplus \mu_3$ is in M.

For the ‘only if’ direction, assume M is closed under $\oplus_3$, and consider $\mu_1, \mu_2 \in M_\mu$. All of $\mu, \mu \oplus \mu_1$ and $\mu \oplus \mu_2$ are in M, and so $\mu \oplus (\mu \oplus \mu_1) \oplus (\mu \oplus \mu_2) = \mu \oplus \mu_1 \oplus \mu_2 \in M$. Hence $\mu_1 \oplus \mu_2 \in M_\mu$. ■

Proposition 2.4. A Boolean function is affine iff it can be written as a conjunction of equations

$$c_1x_1 + c_2x_2 + \ldots + c_nx_n = c_0$$

where $c_i \in \mathcal{B}$ for all $i \in \{0,..,n\}$.

Proof: Assume the Boolean function φ is given as a conjunction of equations of the indicated form and let μ_1, μ_2 and μ_3 be models. That is, for each equation we have

$$\begin{aligned}
c_1\mu_1(x_1) + c_2\mu_1(x_2) + \ldots + c_n\mu_1(x_n) &= c_0\\
c_1\mu_2(x_1) + c_2\mu_2(x_2) + \ldots + c_n\mu_2(x_n) &= c_0\\
c_1\mu_3(x_1) + c_2\mu_3(x_2) + \ldots + c_n\mu_3(x_n) &= c_0
\end{aligned}$$

[2] Other classes that are commonly considered in AI are not closed under conjunction and therefore do not have well-defined concepts of (unique) envelopes. Examples are the *unate* functions (a unate function is one that can be turned into an isotone function by systematic negation of zero or more variables) and the *renamable Horn* functions (a renamable Horn function is similarly one that can be turned into a Horn function by systematic negation of zero or more variables). For example, $x \rightarrow y$ and $x \leftarrow y$ both are unate, while $x \leftrightarrow y$ is not, so the “unate envelope” of the latter is not well-defined.

Adding left-hand sides and adding right-hand sides, making use of the fact that '$\cdot$' distributes over '$+$', we get

$$c_1\mu(x_1)+c_2\mu(x_2)+\ldots+c_n\mu(x_n)=c_0+c_0+c_0=c_0$$

where $\mu=\mu_1\oplus\mu_2\oplus\mu_3$. As μ thus satisfies each equation, μ is a model of φ. This establishes the 'if' direction.

For the 'only if' part, note that by Proposition 2.3, we obtain a vector space M_μ from any non-empty set M closed under $\oplus_3$ by translating each element of M by $\mu\in M$. Now form a basis B for M_μ by taking one non-$\vec{0}$ vector at a time from M_μ and adding it to the set of basis vectors collected so far iff it is linearly independent of that set. Let $j=n-|B|$ (note that $0\le j\le n$). B can be extended to a basis for $\mathcal{B}^n$ by bringing B (read as a $|B|\times n$ matrix) into echelon form and adding j vectors $V=\{\vec{v}_1,\ldots,\vec{v}_j\}$ (these can be chosen from the natural basis for $\mathcal{B}^n$). From B and V we can compute a set of j linear equations

$$\begin{array}{ccccccc} a_{11}x_1 & \oplus & \cdots & \oplus & a_{1n}x_n & = & 0 \\ a_{21}x_1 & \oplus & \cdots & \oplus & a_{2n}x_n & = & 0 \\ & & \vdots & & & \vdots & \\ a_{j1}x_1 & \oplus & \cdots & \oplus & a_{jn}x_n & = & 0 \end{array} \tag{2.1}$$

that have exactly M_μ as their set of models. For each $i\in\{1,\ldots,j\}$, the coefficients $\vec{a}_i=(a_{i1},\ldots,a_{in})$ are uniquely determined by the set of n equations

$$\begin{array}{rcll} \vec{a}_i\cdot\vec{v}_i & = & 1 & \\ \vec{a}_i\cdot\vec{v}_k & = & 0 & 1\le k\le j, k\neq i \\ \vec{a}_i\cdot\vec{b} & = & 0 & \vec{b}\in B. \end{array}$$

This construction guarantees that $\vec{x}=(x_1,\ldots,x_n)$ satisfies the conjunction of equations (2.1) iff $\vec{x}$ is in the span of B (that is, in M_μ). Each function

$$f_i=\lambda\vec{x}.\vec{a}_i\cdot\vec{x}$$

is linear, so for $\nu\in M_\mu$, $f_i(\nu\oplus\mu)=f_i(\nu)+f_i(\mu)=f_i(\mu)$. Hence M can be described by the set of j affine equations

$$\begin{array}{ccccccc} a_{11}x_1 & \oplus & \cdots & \oplus & a_{1n}x_n & = & f_1(\mu) \\ a_{21}x_1 & \oplus & \cdots & \oplus & a_{2n}x_n & = & f_2(\mu) \\ \vdots & & & & & \vdots & \\ a_{j1}x_1 & \oplus & \cdots & \oplus & a_{jn}x_n & = & f_j(\mu) \end{array}$$

as desired. ■

Example 2.5. In $\mathcal{B}^4$, the set of models $M=\{0100,0111,1001,1010\}$ is closed under $\oplus_3$ and so determines an affine function. Choosing $\mu=0100$ as translation, we have $M_\mu=\{0000,0011,1101,1110\}$. One basis for M_μ is $\{0011,1101\}$, which can be extended to a basis for $\mathcal{B}^4$ by adding $V=\{0100,0001\}$. Hence M_μ can be described by the conjunction

$$\begin{array}{ccccccc} a_{11}x_1 & \oplus & \cdots & \oplus & a_{14}x_4 & = & 0 \\ a_{21}x_1 & \oplus & \cdots & \oplus & a_{24}x_4 & = & 0 \end{array}$$

where the coefficients are determined by solving

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

In other words, M_μ is described by

$$\begin{aligned} x_1 \oplus x_2 &= 0 \\ x_1 \oplus x_3 \oplus x_4 &= 0 \end{aligned}$$

In the case of $(x_1, x_2, x_3, x_4) = \mu = (0, 1, 0, 0)$, the left-hand sides evaluate to 1 and 0, respectively. Hence M is described by

$$\begin{aligned} x_1 \oplus x_2 &= 1 \\ x_1 \oplus x_3 \oplus x_4 &= 0 \end{aligned}$$

Zanuttini [66] shows that the complexity of generating the equational form from an affine function's set of models is $\mathcal{O}(n^4)$. Also note that it follows from the syntactic characterisation that the number of models possessed by an affine function is either 0 or a power of 2.

Lemma 2.6. Let φ be a satisfiable Boolean function with set M of models, and let $\mu \models \mathit{aff}(\varphi)$. Then there is an odd positive integer k and a subset M' of M, such that $|M'| = k$ and $\mu = \bigoplus M'$.

Proof: Define

$$\begin{aligned} M_0 &= M \\ M_i &= M_{i-1} \cup \{\mu_1 \oplus \mu_2 \oplus \mu_3 \mid \mu_1, \mu_2, \mu_3 \in M_{i-1}\} \end{aligned}$$

for $i > 0$. Then $\{M_i\}_{i \geq 0}$ is an increasing sequence of sets of models, stabilising in a finite number of steps, that is, for some non-negative j,

$$M_i = M_j = \mathit{models}(\mathit{aff}(\varphi))$$

for all $i \geq j$.

An induction on i now shows that for all i and all $\mu \in M_i$, μ can be written as a sum $\bigoplus M'$ of an odd number of models of $M_0 = M$ ("odd plus odd plus odd equals odd"). In particular this holds for μ in M_j, that is, for each model of $\mathit{aff}(\varphi)$. ■

Proposition 2.7. Let φ be a satisfiable Boolean function and let μ be a model. Then $t_\mu(\mathit{aff}(\varphi)) = \mathit{lin}(t_\mu(\varphi))$.

Proof: As φ is satisfiable, so is $\mathit{lin}(t_\mu(\varphi))$, so let ν be a model of $\mathit{lin}(t_\mu(\varphi))$. Then $\nu = \nu_1 \oplus \cdots \oplus \nu_m$, with each $\nu_1, \ldots, \nu_m$ satisfying $t_\mu(\varphi)$. So each of $\nu_1 \oplus \mu, \ldots, \nu_m \oplus \mu$ satisfies φ. Since $\vec{0} \models t_\mu(\varphi)$, we can assume that m is odd: if m is even, $\vec{0}$ can be added to $\{\nu_1, \ldots, \nu_m\}$ (or removed from the set, as appropriate) without changing the sum. And for odd m, clearly $\nu_1 \oplus \cdots \oplus \nu_m \oplus \mu = \nu_1 \oplus \mu \oplus \cdots \oplus \nu_m \oplus \mu$ is a model of $\mathit{aff}(\varphi)$. Hence ν satisfies $t_\mu(\mathit{aff}(\varphi))$.

Conversely, if ν satisfies $t_\mu(\mathit{aff}(\varphi))$ then $\nu \oplus \mu$ satisfies $\mathit{aff}(\varphi)$, and hence, by Lemma 2.6 $\nu \oplus \mu$ can be written as $\nu_1 \oplus \cdots \oplus \nu_m$, for some odd m, with each of $\nu_1, \ldots, \nu_m$ satisfying φ. That is, $\nu = \nu_1 \oplus \mu \oplus \cdots \nu_m \oplus \mu$, with each of $\nu_1 \oplus \mu, \ldots, \nu_m \oplus \mu$ satisfying $t_\mu(\varphi)$. It follows that ν satisfies $\mathit{lin}(t_\mu(\varphi))$. ■

To express a number of interesting properties of affine Boolean functions, it is convenient to introduce a concept of a "characteristic" valuation for a variable.

Definition 2.8. In the context of a set of variables V, let $v \in V$. The *characteristic valuation* for v, χ_v, is defined by

$$\chi_v(x) = \begin{cases} \mathit{1} & \text{if } x = v \\ \mathit{0} & \text{otherwise.} \end{cases} \quad ■$$

Note that $\mu \oplus \chi_v$ is the valuation which agrees with μ for all variables except v. Moreover, if $\mu \models \varphi$, then both of μ and $\mu \oplus \chi_v$ are models of $\exists v(\varphi)$.

Existential quantification is also an upper closure operator, that is, $\exists v$ is monotone, increasing, and idempotent. Moreover, existential quantification commutes with translation:

Proposition 2.9. Let φ be a Boolean formula, μ a valuation, and v a variable. Then $t_\mu(\exists v(\varphi)) = \exists v(t_\mu(\varphi))$.

Proof: If φ is unsatisfiable, the statement clearly holds, so assume that φ, and hence $t_\mu(\exists v(\varphi))$ is satisfiable. Let $\nu \models t_\mu(\exists v(\varphi))$. Then $\mu \oplus \nu \models \exists v(\varphi)$, and so $\mu \oplus \nu$ satisfies φ, or $\mu \oplus \nu \oplus \chi_v$ does (or both do). For reasons of symmetry we can assume that $\mu \oplus \nu \models \varphi$. Hence $\nu \models t_\mu(\varphi)$ and, since $\exists v$ is increasing, $\nu \models \exists v(t_\mu(\varphi))$.

Conversely, if $\nu \models \exists v(t_\mu(\varphi))$ then $\mu \oplus \nu$ or $\mu \oplus \nu \oplus \chi_v$ satisfies φ (or both do). It follows that $\nu \models t_\mu(\exists v(\varphi))$. ■

Existential quantification also commutes with *lin* and with *aff*:

Proposition 2.10. Let φ be a Boolean function and v a variable. Then

(a) $\mathit{lin}(\exists v(\varphi)) = \exists v(\mathit{lin}(\varphi))$

(b) $\mathit{aff}(\exists v(\varphi)) = \exists v(\mathit{aff}(\varphi))$

Proof: Clearly $lin(\exists v(\varphi)) = 0$ iff $\varphi = 0$ iff $\exists v(lin(\varphi)) = 0$. So assume $lin(\exists v(\varphi))$ is satisfiable and let $\mu \models lin(\exists v(\varphi))$. Then $\mu = \mu_1 \oplus \cdots \oplus \mu_k$ for some non-empty subset $\{\mu_1, \ldots, \mu_k\}$ of $models(\exists v(\varphi))$, and this set in turn is a subset of $models(\exists v(lin(\varphi)))$, as $\exists v$ is monotone and *lin* is increasing. Hence $\mu \models \exists v(lin(\varphi))$.

Conversely, let $\mu \models \exists v(lin(\varphi))$. Then either μ or $\mu \oplus \chi_v$ is a model of $lin(\varphi)$ (or both are). Hence μ (or $\mu \oplus \chi_v$ as the case may be) can be written as a sum $\mu_1 \oplus \cdots \oplus \mu_k$ of k models of φ. It follows that both $\mu_1 \oplus \cdots \oplus \mu_k$ and $\mu_1 \oplus \cdots \oplus \mu_k \oplus \chi_v$ are models of $\exists v(\varphi)$. Hence $\mu \models lin(\exists v(\varphi))$. This establishes item (a).

For item (b), note that $aff(\exists v(\varphi)) = 0$ iff $\varphi = 0$ iff $\exists v(aff(\varphi)) = 0$. So assume that φ is satisfiable and let $\mu \models \varphi$. From item (a) we have

$$lin(\exists v(t_\mu(\varphi))) = \exists v(lin(t_\mu(\varphi)))$$

so that by Proposition 2.9,

$$lin(t_\mu(\exists v(\varphi))) = \exists v(lin(t_\mu(\varphi))).$$

Hence

$$t_\mu(lin(t_\mu(\exists v(\varphi)))) = t_\mu(\exists v(lin(t_\mu(\varphi))))$$

so that by Proposition 2.9,

$$t_\mu(lin(t_\mu(\exists v(\varphi)))) = \exists v(t_\mu(lin(t_\mu(\varphi)))).$$

That is, by Proposition 2.7, $aff(\exists v(\varphi)) = \exists v(aff(\varphi))$. ■

Proposition 2.10 shows that neither linear nor affine approximation introduce variables.

Corollary 2.11. If the Boolean function φ is independent of variable v, so are $lin(\varphi)$ and $aff(\varphi)$.

As mentioned, both *aff* and $\exists v$ are upper closure operators, but there was no *a priori* reason to assume that they commute [51]. Indeed, there are natural classes of Boolean functions for which envelopes are well-defined, but where approximation into the class does not commute with existential quantification. As an example take the class of 1-valid functions [61]. A function is *1-valid* iff it evaluates to *1* when all variables are *1*. This class contains *1* and is closed under conjunction, so we can define $\eta(\varphi)$ to be the 1-valid envelope of φ. The reader can now verify that in $\mathcal{B}^2$, for example,

$$\eta(\exists x(\neg x \wedge \neg y)) = \eta(\neg y) = x \vee \neg y \quad \neq \quad 1 = \exists x(x \leftrightarrow y) = \exists x(\eta(\neg x \wedge \neg y)).$$

Hence *1*-valid approximation and variable elimination do not commute.

While Proposition 2.10 is interesting, the justification of Section 3's affine envelope algorithm requires some stronger results, which we now establish. In particular, independence follows from a weaker property which we call *somewhere-redundancy*.

Definition 2.12. Let φ be a Boolean function, v be a Boolean variable, and μ be a model of φ. We say v is *redundant* for φ and μ iff $\mu \oplus \chi_v \models \varphi$. We say v is *somewhere-redundant* for φ iff there is some model ν of φ such that v is redundant for φ and ν.

We now show that if the Boolean function φ has two models that differ for exactly one variable v, then both its linear and affine envelopes are independent of v.

Proposition 2.13. Let φ be a Boolean function whose set of models M forms a vector space, and assume that for some valuation μ and some variable v, μ and $\mu \oplus \chi_v$ both satisfy φ. Then φ is independent of v.

Proof: The set M of models contains at least two elements, and since it is closed under $\oplus$, χ_v is a model. Hence for *every* model ν of φ, $\nu \oplus \chi_v$ is another model. It follows that φ is independent of v. ■

Proposition 2.14. Let φ be a Boolean function. If v is somewhere-redundant for φ then $lin(\varphi) = \exists v(lin(\varphi)) = lin(\exists v(\varphi))$.

Proof: Note that φ is satisfiable, by assumption. Let μ be a model of φ, with $\mu \oplus \chi_v$ also a model. For *every* model ν of φ, we have that $\nu \oplus \mu \oplus (\mu \oplus \chi_v)$ satisfies $lin(\varphi)$, that is, $\nu \oplus \chi_v \models lin(\varphi)$. Now since both ν and $\nu \oplus \chi_v$ satisfy $lin(\varphi)$, it follows that $\exists v(lin(\varphi))$ cannot have a model that is not already a model of $lin(\varphi)$ (and the converse holds trivially). Hence $lin(\varphi) = \exists v(lin(\varphi))$. The second equation follows immediately from Proposition 2.10(a). ■

Corollary 2.15. Let φ be a Boolean function. If v is somewhere-redundant for φ then $aff(\varphi) = \exists v(aff(\varphi)) = aff(\exists v(\varphi))$.

Proof: Note that φ is satisfiable, by assumption. Let $\mu \models \varphi$. Note that since v is somewhere-redundant for φ, v is somewhere-redundant for $t_\mu(\varphi)$ as well. So by Proposition 2.14,

$$lin(t_\mu(\varphi)) = \exists v(lin(t_\mu(\varphi))).$$

But then, by Proposition 2.10(a),

$$t_\mu(lin(t_\mu(\varphi))) = \exists v(t_\mu(lin(t_\mu(\varphi))))$$

and so, by Proposition 2.7, $aff(\varphi) = \exists v(aff(\varphi))$. The second equation follows immediately from Proposition 2.10(b). ■

These results justify an aggressive approach to the elimination of variables in an affine envelope algorithm. We shall utilise this in the next section.

2.3 ROBDDs

We briefly recall the essentials of ROBDDs [11]. Let the set $\mathcal{V}$ of propositional variables be equipped with a total ordering $\prec$. *Binary decision diagrams* (*BDDs*) are defined inductively as follows:

- 0 is a BDD.
- 1 is a BDD.
- If $x \in \mathcal{V}$ and R_1 and R_2 are BDDs then $\mathsf{ite}(x, R_1, R_2)$ is a BDD.

Algorithm 1 The "or" operator for ROBDDs

```
$\mathsf{or}(1, \_) = 1$
$\mathsf{or}(0, R) = \mathrm{R}$
$\mathsf{or}(\_, 1) = 1$
$\mathsf{or}(R, 0) = \mathrm{R}$
$\mathsf{or}(\mathsf{ite}(x, T, E), \mathsf{ite}(x', T', E'))$
    $\mid x \prec x' = \mathsf{mknd}(x, \mathsf{or}(T, \mathsf{ite}(x', T', E')), \mathsf{or}(E, \mathsf{ite}(x', T', E')))$
    $\mid x' \prec x = \mathsf{mknd}(x', \mathsf{or}(\mathsf{ite}(x, T, E), T'), \mathsf{or}(\mathsf{ite}(x, T, E), E'))$
    $\mid$ **otherwise** $= \mathsf{mknd}(x, \mathsf{or}(T, T'), \mathsf{or}(E, E'))$
```

Let $R = \mathsf{ite}(x, R_1, R_2)$. We say a BDD R' *appears in* R iff $R' = R$ or R' appears in R_1 or R_2. We define $\mathsf{vars}(R) = \{v \mid \mathsf{ite}(v, \_, \_)$ appears in $R\}$.

The meaning of a BDD is given as follows.

$$\begin{aligned} [\![0]\!] &= \mathit{0} \\ [\![1]\!] &= \mathit{1} \\ [\![\mathsf{ite}(x, R_1, R_2)]\!] &= (x \wedge [\![R_1]\!]) \vee (\neg x \wedge [\![R_2]\!]). \end{aligned}$$

A BDD is an *Ordered binary decision diagram* (*OBDD*) iff it is 0 or 1 or if it is $\mathsf{ite}(x, R_1, R_2)$, R_1 and R_2 are OBDDs, and $\forall x' \in \mathsf{vars}(R_1) \cup \mathsf{vars}(R_2) : x \prec x'$.

An OBDD R is a *Reduced Ordered Binary Decision Diagram* (*ROBDD* [10, 11]) iff for all BDDs R_1 and R_2 appearing in R, $R_1 = R_2$ when $[\![R_1]\!] = [\![R_2]\!]$. Practical implementations [8] use a function $\mathsf{mknd}(x, R_1, R_2)$ to create all ROBDD nodes as follows:

1. If $R_1 = R_2$, return R_1 instead of a new node, as $[\![\mathsf{ite}(x, R_1, R_2)]\!] = [\![R_1]\!]$.

2. If an identical ROBDD was previously built, return that one instead of a new one; this is accomplished by keeping a hash table, called the *unique table*, of all previously created nodes.

3. Otherwise, return $\mathsf{ite}(x, R_1, R_2)$.

This ensures that ROBDDs are strongly canonical: a shallow equality test is sufficient to determine whether two ROBDDs represent the same Boolean function.

Figure 1 shows an example of an ROBDD. In general we depict the ROBDD $\mathsf{ite}(x, R_1, R_2)$ as a directed acyclic graph rooted in x, with a solid arc from x to the dag for R_1 and a dashed line from x to the dag for R_2. However, to avoid unnecessary clutter, we omit the 0 node (sink) and all arcs leading to that sink. The ROBDD in Figure 1 denotes the function which has five models: $\{00011, 00110, 01001, 01101, 10101\}$.

As a typical example of an ROBDD algorithm, Algorithm 1 generates the disjunction of two given ROBDDs. This operation will be used by the affine approximation algorithm presented in Section 3. (Most of our algorithms are presented in a functional programming style, using Haskell-style pattern matching and guarded equations.)

Algorithm 2 is used to extract a model from an ROBDD. For an unsatisfiable ROBDD (that is, 0) we return $\perp$. Although presented here in recursive fashion, it is better implemented in an iterative manner

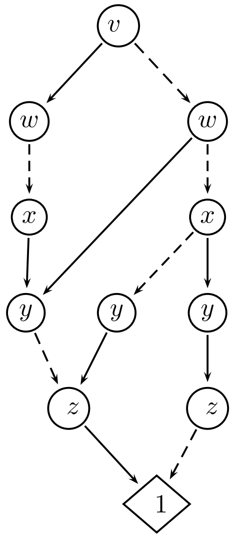


Figure 1: An example of our diagrammatic representation of an ROBDD. Our diagrams leave out the 0 sink and all arcs to it.

Algorithm 2 get_model algorithm for ROBDDs

```
$\mathsf{get\_model}(0) = \bot$
$\mathsf{get\_model}(1) = \lambda v.*$
$\mathsf{get\_model}(\mathsf{ite}(x,T,E)) =$
    $\mathbf{let}\ \mu = \mathsf{get\_model}(T)\ \mathbf{in}$
        $\mathbf{if}\ \mu = \bot\ \mathbf{then}\ \mathsf{get\_model}(E)[x \mapsto 0]\ \mathbf{else}\ \mu[x \mapsto 1]$
```

whereby we traverse through the ROBDD, one pointer moving down the “else” branch at each node, a second pointer trailing immediately behind. If a 1 sink is found, we return the path traversed thus far and note that any further variables which we are yet to encounter may be assigned any value. If a 0 sink is found, we use the trailing pointer to step up a level, follow the “then” branch for one step and continue searching for a model by following “else” branches. This method relies on the fact that ROBDDs are “reduced”, so that if no 1 sink can be reached from a node, then the node itself is the 0 sink.

We shall use the following obvious corollary of Proposition 2.13:

Corollary 2.16. Let ROBDD R represent a function whose set of models forms a vector space. Then every path from R's root node to the 1 sink contains the same sequence of variables, namely $\mathsf{vars}(R)$ listed in variable order.

It is important to take advantage of *fan-in* to create efficient ROBDD algorithms. Often some ROBDD nodes will appear multiple times in a given ROBDD, and algorithms that traverse that ROBDD will meet

Algorithm 3 The sets-of-models based affine envelope algorithm

```
Input: The set $M$ of models for function $\varphi$.
Output: $\mathit{aff}(M)$ — the set of models of $\varphi$'s affine envelope.

if $M = \varnothing$ then
    return $M$
end if
$N \leftarrow \varnothing$
choose $\mu \in M$
$New \leftarrow M_\mu$
repeat
    $N \leftarrow N \cup New$
    $New \leftarrow \{\mu_1 \oplus \mu_2 \mid \mu_1, \mu_2 \in N\} \setminus N$
until $New = \varnothing$
return $N_\mu$
```

these nodes multiple times. Many algorithms can avoid repeated work by keeping a cache of previously seen inputs and their corresponding outputs, called a *computed table* [8]. We silently use computed tables for the recursive ROBDD algorithms presented here.

3 Finding Affine Envelopes for ROBDDs

Zanuttini [66] gives an algorithm, here presented as Algorithm 3, for finding the affine envelope, assuming a Boolean function φ is represented as a set of models. This algorithm is justified by Proposition 2.3.

Example 3.1. To see Algorithm 3 in action, refer to Figure 2. Assume that φ has four models, $M = \{01011, 01100, 10111, 11001\}$. We randomly pick $\mu = 01100$ and obtain M_μ as shown. The first round of completion under '$\oplus$' adds three bit-strings: $\{11100, 10010, 01110\}$, and another round adds 01001 to produce N. Finally, "adding back" $\mu = 01100$ yields the affine envelope $N_\mu = \mathit{aff}(M)$.

We are interested in developing an algorithm for ROBDDs. We can improve on Algorithm 3 and at the same time make it more suitable for ROBDD manipulation. The idea is to build the result N step by step, by picking the models ν of M_μ one at a time and computing $N := N \cup N_\nu$ at each step. We can start from $N = \{\vec{0}\}$, as $\vec{0}$ has to be in M_μ. This leads to Algorithm 4.

This formulation is well suited to ROBDDs, as the operation N_ν, that is, taking the xor of a model ν with each model of the *ROBDD N* can be implemented by traversing N and, for each v-node with $\nu(v) = \mathit{1}$, swapping that node's children. And we can do better, utilising two observations.

First, during its construction, there is no need to traverse the ROBDD N for each individual model ν. A full traversal of N will find all its models systematically, eliminating a need to remove them one by one.

Second, the ROBDD being constructed can be simplified aggressively during its construction, by utilising Propositions 2.14 and 2.10(a). Namely, as we traverse ROBDD R systematically, paths from the root to the 1 sink may be found that do not contain every variable in $\mathsf{vars}(R)$. Each such path corresponds

$$M = \left\{ \begin{array}{c} 01011 \\ 01100 \\ 10111 \\ 11001 \end{array} \right\} \qquad \mu = 01100 \qquad M_\mu = \left\{ \begin{array}{c} 00111 \\ 00000 \\ 11011 \\ 10101 \end{array} \right\}$$

$$N = \left\{ \begin{array}{c} 00111 \\ 00000 \\ 11011 \\ 10101 \\ 11100 \\ 10010 \\ 01110 \\ 01001 \end{array} \right\} \qquad N_\mu = \mathit{aff}(M) = \left\{ \begin{array}{c} 01011 \\ 01100 \\ 10111 \\ 11001 \\ 10000 \\ 11110 \\ 00010 \\ 00101 \end{array} \right\}$$

Figure 2: Steps in Algorithm 3

Algorithm 4 A variant of Algorithm 3

```
Input: The set $M$ of models for function $\varphi$.
Output: $\mathit{aff}(M)$ — the set of models of $\varphi$'s affine envelope.

if $M = \varnothing$ then
    return $M$
end if
$N \leftarrow \{\vec{0}\}$
choose $\mu \in M$
$M' \leftarrow M_\mu \setminus \{\vec{0}\}$
for all $\nu \in M'$ do
    $N \leftarrow N \cup N_\nu$
end for
return $N_\mu$
```

to a model *set* of cardinality 2^k, k being the number of "skipped" variables, and each skipped variable is what was termed "somewhere-redundant" in Section 2.2. Proposition 2.14 tells us that, eventually, the linear (and hence also the affine) envelope will be independent of all such "skipped" variables, and Proposition 2.10 guarantees that variable elimination can be interspersed arbitrarily with the process of "xor-ing" models, that is, we can eliminate variables aggressively.

This leads to Algorithm 5. The algorithm combines several operations in an effort to amortise their cost. In what follows we step through the details of the algorithm.

The to_aff function finds an initial model μ of R, before translating R by calling translate. This initial call has the effect of "xor-ing" μ with all of the models of R. Once translated, the xor closure is taken, before translating again using the initial model μ to obtain the affine closure.

Algorithm 5 Affine envelopes for ROBDDs

Input: An ROBDD R.

Output: The affine envelope of R.

```
to_aff(0) = 0
to_aff($R$) = let $\mu$ = get_model($R$) in translate(xor_close(translate($R, \mu$)), $\mu$)

translate(0, _) = 0
translate(1, _) = 1
translate(ite($x, T, E$), $\mu$)
        | ($\mu(x) = 0$) = cons($x$, translate($T, \mu$), translate($E, \mu$), $\mu$)
        | ($\mu(x) = 1$) = cons($x$, translate($E, \mu$), translate($T, \mu$), $\mu$)

xor_close($R$) = trav($R, \lambda v.*, \bigwedge\{\bar{v} \mid v \in$ vars($R$)$\}$)

trav(0, _, $S$) = $S$
trav(1, $\mu, S$)
        | ($\mu \models S$) = $S$
        | otherwise = extend($S, S, \mu$)
trav(ite($x, T, E$), $\mu, S$) = trav($T, \mu[x \mapsto 1]$, trav($E, \mu[x \mapsto 0], S$))

cons($x, T, E, \mu$)
        | ($\mu(x) = *$) = or($T, E$)
        | otherwise = mknd($x, T, E$)

extend(1, _, _) = 1
extend(_, 1, _) = 1
extend(0, $S, \mu$) = translate($S, \mu$)
extend(ite($x, T, E$), 0, $\mu$) = cons($x$, extend($T, 0, \mu$), extend($E, 0, \mu$), $\mu$)
extend(ite($x, T, E$), ite($x, T', E'$), $\mu$)
        | ($\mu(x) = 1$) = mknd($x$, extend($T, E', \mu$), extend($E, T', \mu$))
        | otherwise = cons($x$, extend($T, T', \mu$), extend($E, E', \mu$), $\mu$)
```

translate is the function that is responsible for computing the xor of a model with an ROBDD. As mentioned above, its operation relies on the observation that for a given node v in the ROBDD, if $\mu(v) = 1$, then the operation is equivalent to exchanging the "then" and "else" branches of v.

xor_close is used to compute the xor-closure of an ROBDD R. The third argument passed to trav is an accumulator in which the result is constructed. As in Algorithm 4, we know that $\vec{0}$ will be a model of the result, so we initialise the accumulator as (the ROBDD for) $\bigwedge\{\bar{v} \mid v \in \mathsf{vars}(R)\}$.

trav implements a recursive traversal of the ROBDD, and when a model is found in μ, we "extend" the affine envelope to include the newly found model. Namely, extend(R, S, μ) produces (the ROBDD

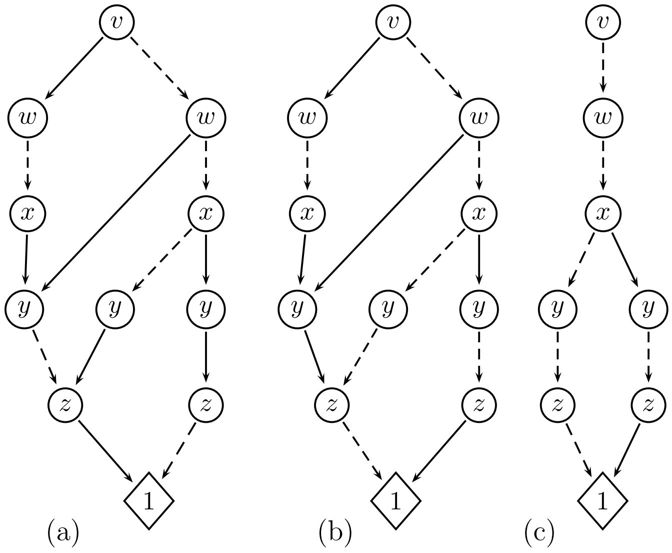


Figure 3: (a): The ROBDD R from Figure 1. (b): The translated version R_μ. (c): The vector space S that has been extended to cover 00101.

for) $R \vee S_\mu$. Note that once a model is found during the traversal, trav checks if μ is already present within the xor-closure, and if it is not, invokes extend accordingly. This simple check avoids making unnecessary calls to extend.

The cons function represents a special case of mknd. It takes an additional argument in μ and uses it to determine whether to restrict away the corresponding node being constructed. It is the correctness of this function that rests on Propositions 2.14 and 2.10, as discussed (showing that affine approximation can be interspersed with variable elimination).

Finally, once a model is found during a traversal, extend is used to build up the affine closure of the ROBDD. The last equation requires some explanation. In the context of the initial call extend(S,S,μ), Corollary 2.16 ensures that the pattern of the last equation for extend is sufficient: If neither argument is a sink, the two will have the same root variable. If $\mu(x) = \mathit{0}$, we simply build the x-node and recurse. If $\mu(x) = \mathit{1}$, we build the x-node but swap the branches of the second ROBDD before we recurse (recall that we are building an ROBDD for $R \vee S_\mu$, and S is the second argument to extend). Finally, if $\mu(x) = *$, the x node should not be created, as we wish to take the existential quantification over x; the call to cons will achieve this.

Example 3.2. Consider the ROBDD R shown in Figure 3(a). The corresponding set of models is $\{00011, 00110, 01001, 01101, 10101\}$. Picking $\mu = 00011$ and translating gives R_μ, shown in Figure 3(b). This ROBDD represents a set of vectors $\{00000, 00101, 01010, 01110, 10110\}$ which is to be extended to a vector space.

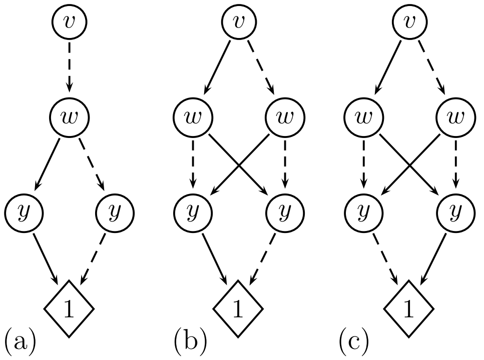


Figure 4: (a): The vector space S after being extended to cover $01{*}10$. (b): S after extending to cover 10110. (c): S translated to give the affine closure of R.

The algorithm now builds up S, the xor-closure of R_μ, by taking one vector v at a time from R_μ and extending S to a vector space that includes v. S begins as the zero vector.

The first step of the algorithm just adds 00101 to the existing zero vector (Figure 3(c)). The next step comes across the vector $01{*}10$ (which actually represents two valuations) and existentially quantifies away the variable x (Figure 4(a)). Note that the variable z also disappears: this is due to the extension required to include $01{*}10$ that adds enough valuations such that z is "covered" by the vector space.

Extending to cover 10110 simply requires every model to be copied, with v mapped to *1* (Figure 4(b)). Finally, translating back by μ produces A, the affine closure of R, shown in Figure 4(c).

Proposition 2.14 justified the elimination of what may be called "skipped" variables in the input ROBDD: variables that were missing on *some* path from the root to the *1*-sink. The reader may wonder whether the calculation of the affine envelope could be reduced to just a sequence of existential quantifications (in which case our algorithms would be unnecessarily complex). In other words, under the assumptions made in Corollary 2.15, does $\mathit{aff}(\varphi) \models \exists v(\varphi)$ hold as well? Figure 5 gives an example to show that the answer is no. For the function $(x \wedge y \wedge z) \vee (\neg x \wedge (y+z))$ there are no skipped variables, but the function is not affine, as it has three models. The ROBDD for the affine envelope, $x+y+z$, is shown on the right, and is dependent on all three variables.

4 Experimental Evaluation

To evaluate Algorithms 3 and 5 we have run both algorithms on two suites of Boolean functions. It should be stressed that Algorithm 3 was not intended as a practical proposal, but introduced for didactic purposes—we use it here simply to have some baseline for comparison. The algorithms have been run on both randomly generated functions and structured functions, sourced from SAT-based approaches to combinatorial problem solving.

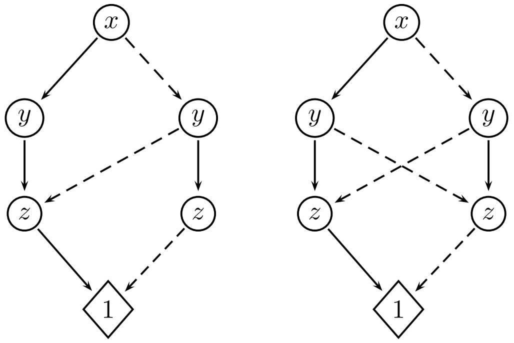


Figure 5: An ROBDD (on the left) without skipped variables on any 1-path, and its affine envelope (on the right).

Algorithm 6 Generation of random Boolean functions as ROBDDs

Input: The number n of variables in the random function,
pr a calibrator set so that the probability
of a valuation being a model is 2^{-pr}.

Output: A random Boolean function represented as an ROBDD.

```
gen_rand_bdd($n, pr$) = rand_bdd($0, n-1, pr$)

rand_bdd($m, n, pr$)
      | $(m = n) =$ mknd($m$, rand_sink(), rand_sink())
      | **otherwise** $=$ mknd($m, T, E$)
      **where**
            $T$ = **if** $(m > n - pr) \wedge$ cointoss() **then** rand_bdd($m+1, n, pr$) **else** $0$
            $E$ = **if** $(m > n - pr) \wedge$ cointoss() **then** rand_bdd($m+1, n, pr$) **else** $0$

rand_sink()= **if** cointoss() **then** $1$ **else** $0$

cointoss() returns *1* or *0* with equal probability.
```

The structured functions have been translated from DIMACS CNF syntax. They are: `ais6`, an all-interval-series instance from SATLIB, `queens`N, solving the N-queens problem for $N = 4,5,6$, and `sudoku`N, solving a 4×4 sudoku instance with N squares already filled, for $1 \leq N \leq 5$.

We generated random Boolean functions of varying arity, with an additional parameter to control the density of the generated function, that is, to set the likelihood of each valuation being a model.

The random Boolean functions have been generated using Algorithm 6. The function call gen_rand_bdd(n, pr) builds, in the form of an ROBDD R, a random Boolean function with the prop-

Function	Variables	Algorithm 3	Algorithm 5
random	12	0.02	0.02
random	15	5.99	0.27
random	18	—	0.41
random	21	—	1.71
random	24	—	14.97
queens4	17	0.35	0.03
queens5	26	6826.20	2.48
queens6	37	31.32	0.11
ais6	61	$> 3.6 \cdot 10^6$	42702.00
sudoku1	64	$> 3.6 \cdot 10^6$	12319.20
sudoku2	64	154633.33	53.60
sudoku3	64	6291.00	6.90
sudoku4	64	106.20	0.79
sudoku5	64	4.09	0.11

Table 1: Average time in milliseconds to compute an affine envelope

erty that the likelihood of an arbitrary valuation satisfying R is 2^{-pr}. This is done by invoking $\mathsf{rand\_bdd}(0, n-1, pr)$. This recursive algorithm builds a ROBDD of $(n - pr)$ variables and, at depth $(n - pr)$, a random choice is made as to whether to continue generating the random function or to simply join the branch with a 0 sink. If the choice is to continue, then the algorithm recursively applies $\mathsf{rand\_bdd}(m+1, n, pr)$ to the branch. Note that cointoss is non-deterministic, so the T and E used in the algorithm are not in general equal.

To time the generation of envelopes for random functions, we generated 10,000 12-place random Boolean functions, in each case with the probability of 1/1024 for a valuation to be a model. We did the same for 15-, 18-, 21-, and 24-place random Boolean functions. To time the generation of the affine envelope of each of the 10 structured Boolean functions, we repeated the generation n times and took the average time. The parameter n was chosen between 10 and 100,000, so as to ensure that the n repetitions took at least 3 seconds.

Table 1 shows the average times (in milliseconds) taken by each of the algorithms. Timing data were collected on a machine running Solaris 9, with two Intel Xeon CPUs running at 2.8GHz and 4GB of memory. Only one CPU was used and tests were run under minimal load on the system. Our implementation of Algorithm 3 uses sorted arrays of bitstrings (so that search for models is logarithmic). As the number of models grows exponentially with the number of variables, it is not surprising that memory consumption for some tests exceeded available space. As mentioned, the comparison is not that interesting anyway.

Given a function φ, the number of nodes in $\mathit{aff}(\varphi)$'s ROBDD may be smaller or larger than that of φ's ROBDD. (In our experiments, we have observed that, on average, the envelope is smaller than the original function.) Note that the ROBDD for $\mathit{aff}(\varphi)$ has a depth which is no larger that of φ's ROBDD. This is because an envelope cannot introduce variables, and will often remove some.

5 Conclusion

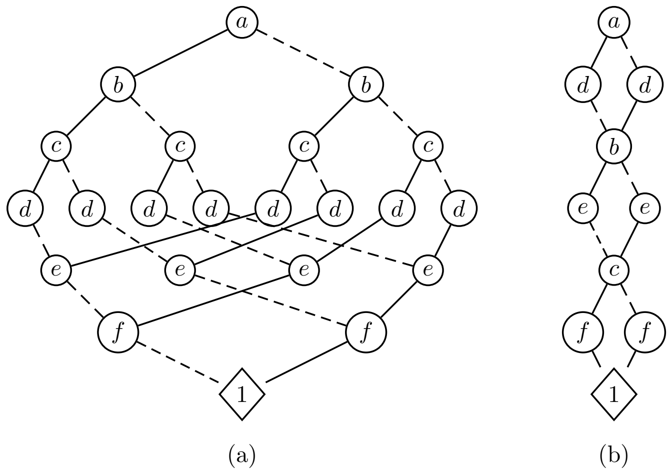
Boolean approximation poses interesting algorithmic challenges. Envelopes for Boolean formulas have a number of different applications and for example find use in speeding up the querying of knowledge-bases. Previous research has focused on the use of Horn approximations represented in conjunctive normal form (CNF). In this paper, following a suggestion by Zanuttini [66], we instead focused on the class of affine functions. Zanuttini exemplifies the utility of this and points out that using the affine envelope instead of the original function leads to no loss of precision at all when the logical consequences tested are affine (as would be the case when one tests parity properties of a circuit, say).

As could be expected, our initial (baseline) implementation using a naive sets-of-models (as arrays of bitstrings) representation was of limited value, because, even for functions with very few models, the affine envelope often has very many models (the affine envelope of a majority of Boolean functions is *1*). So storing sets of models as an array becomes prohibitive even for functions over rather few variables.

ROBDDs have proved to be an appropriate representation for many applications of Boolean functions. Functions with very many models, as well as very few, have compact ROBDD representations. Thus we have developed a new affine envelope algorithm using ROBDDs. Our approach is based on the same principle as Zanuttini's, but takes advantage of some useful characteristics of ROBDDs, together with certain properties of affine Boolean functions. Propositions 2.14 and 2.10 establish the most important of these properties, including the fact that affine approximation commutes with existential quantification. This is what allows an algorithm for the generation of affine envelopes to eliminate variables aggressively, often significantly reducing the sizes of the representations being manipulated earlier than would happen otherwise.

Table 1 suggests that this "aggressive" approach pays off uniformly, with the benefit generally increasing as functions grow in arity. The benefit also appears to be present across the complete lattice of Boolean functions. The `sudoku` series of functions were designed to investigate that point. As the parameter N in `sudokuN` grows, the functions, which all have the same number of variables, grow stronger (as more numbers are placed on the initially empty sudoku board, more constraints are added, and the number of models decreases). The aggressive approach has an advantage across that sequence of functions.

We have not been able to obtain a precise complexity analysis of the algorithm, and we leave this as an open problem. We note, however, that in the worst case, an ROBDD for an n-place affine function can reach the maximal size for an n-place ROBDD, namely $3 \cdot 2^{n/2} - 1$ nodes. For example, $(x_1 + y_1)(x_2 + y_2) \cdots (x_{\frac{n}{2}} + y_{\frac{n}{2}})$ (n even) gives rise to an ROBDD with $3 \cdot 2^{n/2} - 1$ nodes, assuming the variable ordering is $x_1 \prec x_2 \prec \cdots \prec x_{\frac{n}{2}} \prec y_1 \prec y_2 \prec \cdots y_{\frac{n}{2}}$. (On the other hand, with variable ordering $x_1 \prec y_1 \prec x_2 \prec y_2 \prec \cdots \prec x_{\frac{n}{2}} \prec y_{\frac{n}{2}}$, the ROBDD is linear in n, as the ROBDD has $\frac{3n}{2} + 2$ nodes. Figure 6(a) exemplifies this for $n = 6$. On the left is the ROBDD that uses the first ordering. With 23 nodes (as usual, we omit the 0-sink), it clearly has the greatest number of nodes that any ROBDD for a 6-place function can have. On the right is the ROBDD, with 11 nodes, that uses the second ordering.



(a)

(b)

Figure 6: (a) shows a worst-case affine ROBDD, representing $(a+d)(b+e)(c+f)$ under the variable ordering $a \prec b \prec c \prec d \prec e \prec f$. (b) shows the same affine function using the variable ordering $a \prec d \prec b \prec e \prec c \prec f$.

Acknowledgements

We wish to thank the reviewers for their helpful suggestions which led to many improvements to this paper.

References

[1] M. ANTHONY AND N. BIGGS: *Computational Learning Theory*. Volume 30 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

[2] T. ARMSTRONG, K. MARRIOTT, P. SCHACHTE, AND H. SØNDERGAARD: Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.

[3] R. BAGNARA, P. HILL, AND E. ZAFFANELLA: Set-sharing is redundant for pair-sharing. In P. VAN HENTENRYCK, editor, *Static Analysis: Proc. Fourth Int. Symp.*, volume 1302 of *LNCS*, pp. 53–67. Springer, 1997.

[4] R. BAGNARA AND P. SCHACHTE: Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. HAEBERER, editor, *Proc. Seventh Int. Conf. Algebraic Methodology and Software Technology (AMAST'98)*, volume 1548 of *LNCS*, pp. 471–485. Springer, 1998.

[5] G. BIRKHOFF: *Lattice Theory*. American Mathematical Society, third edition, 1973.

[6] E. BÖHLER, N. CREIGNOU, S. REITH, AND H. VOLLMER: Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM SIGACT News*, 35(1):22–35, 2004.

[7] Y. BOUFKHAD: Algorithms for propositional KB approximation. In *Proc. Fifteenth Nat. Conf. Artificial Intelligence*, pp. 280–285. AAAI Press / MIT Press, 1998.

[8] K. BRACE, R. RUDELL, AND R. BRYANT: Efficient implementation of a BDD package. In *Proceedings of the Twenty-seventh ACM/IEEE Design Automation Conference*, pp. 40–45, 1990. 2, 11, 13

[9] F. M. BROWN: *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publ., 1990.

[10] R. BRYANT: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C–35(8):677–691, 1986. 2, 11

[11] R. BRYANT: Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. 10, 11

[12] M. CADOLI AND F. SCARCELLO: Semantical and computational aspects of Horn approximations. *Artificial Intelligence*, 119:1–17, 2000.

[13] V. CHANDRU AND J. N. HOOKER: Extended Horn sets in propositional logic. *Journal of the ACM*, 38(1):205–221, 1991.

[14] C. CHANG AND H. KEISLER: *Model Theory*. Volume 73 of *Studies in Logic and Foundations of Mathematics*. North-Holland, 1973.

[15] M. CODISH AND H. SØNDERGAARD: The Boolean logic of set sharing analysis. In C. PALAMIDESSI, H. GLASER, AND K. MEINKE, editors, *Principles of Declarative Programming*, volume 1490 of *LNCS*, pp. 89–101. Springer, 1998.

[16] M. CODISH, H. SØNDERGAARD, AND P. J. STUCKEY: Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 21(5):948–976, 1999.

[17] P. COUSOT AND R. COUSOT: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, pp. 238–252. ACM Press, 1977.

[18] P. COUSOT AND R. COUSOT: Static determination of dynamic properties of recursive procedures. In E. J. NEUHOLD, editor, *Formal Description of Programming Concepts*, pp. 237–277. North-Holland, 1978.

[19] P. COUSOT AND R. COUSOT: Systematic design of program analysis frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pp. 269–282. ACM Press, 1979.

[20] P. DART: On derived dependencies and connected databases. *Journal of Logic Programming*, 11(2):163–188, 1991.

[21] R. DECHTER AND J. PEARL: Structure identification in relational data. *Artificial Intelligence*, 58:237–270, 1992. 5

[22] A. DEL VAL: First order LUB approximations: Characterization and algorithms. *Artificial Intelligence*, 162:7–48, 2005. 2

[23] T. EITER, G. GOTTLOB, AND K. MAKINO: New results on monotone dualization and generating hypergraph transversals. In *Proc. Thirty-fourth Ann. ACM Symp. Theory of Computing*, pp. 14–22. ACM Press, 2002.

[24] O. EKIN, S. FOLDES, P. L. HAMMER, AND L. HELLERSTEIN: Equational characterizations of Boolean function classes. *Discrete Mathematics*, 211:27–51, 2000.

[25] S. GENAIM AND A. KING: Goal-independent suspension analysis for logic programs with dynamic scheduling. In P. DEGANO, editor, *Proc. European Symp. Programming 2006*, volume 2618 of *LNCS*, pp. 84–98. Springer, 2003.

[26] S. GENAIM AND A. KING: Inferring non-suspension conditions for logic programs with dynamic scheduling. Technical Report 20–04, University of Kent Computing Laboratory, 2004.

[27] R. GIACOBAZZI: *Semantic Aspects of Logic Program Analysis*. PhD thesis, University of Pisa, Italy, 1993.

[28] R. GIACOBAZZI, F. RANZATO, AND F. SCOZZARI: Making abstract domains condensing. *ACM Trans. Computational Logic*, 6(1):33–60, 2005.

[29] R. GIACOBAZZI AND F. SCOZZARI: A logical model for relational abstract domains. *ACM Trans. Programming Languages and Systems*, 20(5):1067–1109, 1998.

[30] K. GLYNN, P. J. STUCKEY, M. SULZMANN, AND H. SØNDERGAARD: Exception analysis for non-strict languages. In *Proc. 2002 ACM SIGPLAN Int. Conf. Functional Programming*, pp. 98–109. ACM Press, 2002.

[31] O. GOLDREICH, S. GOLDWASSER, AND D. RON: Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[32] P. R. HALMOS: Algebraic logic i: Monadic Boolean algebras. *Compositio Mathematica*, 12:217–249, 1954–1956.

[33] P. R. HALMOS: *Lectures on Boolean Algebras*. Springer-Verlag, 1963.

[34] A. HEATON, M. ABO-ZAED, M. CODISH, AND A. KING: A simple polynomial groundness analysis for logic programs. *J. Logic Programming*, 45(1-3):143–156, 2000.

[35] K. HENSHALL, P. SCHACHTE, H. SØNDERGAARD, AND L. WHITING: Boolean affine approximation with binary decision diagrams. In R. DOWNEY AND P. MANYEM, editors, *Theory of Computing 2009*, volume 94 of *Conferences in Research and Practice in Information Technology*, pp. 121–129, 2009. 2

[36] T. HORIYAMA AND T. IBARAKI: Ordered binary decision diagrams as knowledge-bases. *Artificial Intelligence*, 136:189–213, 2002.

[37] T. HORIYAMA AND T. IBARAKI: Translation among CNFs, characteristic models and ordered binary decision diagrams. *Inf. Processing Letters*, 85:191–198, 2003.

[38] J. HOWE AND A. KING: Implementing groundness analysis with definite Boolean functions. In G. SMOLKA, editor, *Programming Languages and Systems*, volume 1782 of *LNCS*, pp. 200–214. Springer, 2000.

[39] J. M. HOWE AND A. KING: Positive Boolean functions as multiheaded clauses. In P. CODOGNET, editor, *Int. Conf. Logic Programming*, volume 2237 of *LNCS*, pp. 120–134. Springer, 2001.

[40] J. M. HOWE AND A. KING: Efficient groundness analysis in Prolog. *Theory and Practice of Logic Programming*, 3(1):95–124, 2003.

[41] J. M. HOWE, A. KING, AND L. LU: Analysing logic programs by reasoning backwards. In M. BRUYNOOGHE AND K.-K. LAU, editors, *Program Development in Computational Logic*, volume 3049 of *LNCS*, pp. 152–188. Springer, 2004.

[42] T. IBARAKI, A. KOGAN, AND K. MAKINO: Functional dependencies in Horn theories. *Artificial Intelligence*, 108:1–30, 1999.

[43] H. KAUTZ, M. KEARNS, AND B. SELMAN: Horn approximations of empirical data. *Artificial Intelligence*, 74:129–145, 1995.

[44] D. KAVVADIAS, C. PAPADIMITRIOU, AND M. SIDERI: On Horn envelopes and hypergraph transversals. In K. NG, P. RAGHAVAN, N. BALASUBRAMANIAN, AND F. CHIN, editors, *Proc. Fourth Int. Symp. Algorithms and Computation*, volume 762 of *LNCS*, pp. 399–405. Springer, 1993.

[45] D. KAVVADIAS AND M. SIDERI: The inverse satisfiability problem. *SIAM Journal of Computing*, 28:152–163, 1998.

[46] R. KHARDON: Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.

[47] R. KHARDON AND D. ROTH: Reasoning with models. *Artificial Intelligence*, 87:187–213, 1996.

[48] M. KURIHARA AND H. KONDO: Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In R. ORCHARD, C. YANG, AND M. ALI, editors, *Innovations in Applied Artificial Intelligence: Proc. 17th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'04)*, volume 3029 of *LNAI*, pp. 827–837. Springer, 2004.

[49] K. MARRIOTT AND H. SØNDERGAARD: Precise and efficient groundness analysis for logic programs. *ACM Lett. Programming Languages and Systems*, 2(1–4):181–196, 1993.

[50] A. MYCROFT: *Abstract Interpretation and Optimising Transformations for Applicative Programs*. PhD thesis, University of Edinburgh, Scotland, 1981.

[51] O. ORE: Combinations of closure relations. *Annals of Mathematics*, 44(3):514–533, 1943. 4, 9

[52] M. PARNAS, D. RON, AND A. SAMORODNITSKY: Testing basic Boolean formulae. *SIAM J. Discrete Mathematics*, 16(1):20–46, 2002.

[53] F. J. PELLETIER AND N. M. MARTIN: Post's functional completeness theorem. *Notre Dame Journal of Formal Logic*, 31(2), 1990.

[54] N. PIPPENGER: *Theories of Computability*. Cambridge University Press, 1941. 1, 5

[55] E. POST: *The Two-Valued Iterative Systems of Mathematical Logic*. Princeton University Press, 1941. Reprinted in M. Davis, Solvability, Provability, Definability: The Collected Works of Emil L. Post, pages 249–374, Birkhaüser, 1994. 4

[56] S. RUDEANU: *Boolean Functions and Equations*. North-Holland, 1974.

[57] P. SCHACHTE: Efficient ROBDD operations for program analysis. In K. RAMAMOHANARAO, editor, *ACSC'96: Proc. Nineteenth Australasian Computer Science Conference*, pp. 347–356. Australian Computer Science Communications, 1996.

[58] P. SCHACHTE AND H. SØNDERGAARD: Closure operators for ROBDDs. In E. A. EMERSON AND K. NAMJOSHI, editors, *Proceedings of the Seventh International Conference on Verification, Model Checking and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pp. 1–16. Springer, 2006. 2

[59] P. SCHACHTE AND H. SØNDERGAARD: Boolean approximation revisited. In I. MIGUEL AND W. RUML, editors, *Abstraction, Reformulation and Approximation: Proceedings of SARA 2007*, volume 4612 of *Lecture Notes in Artificial Intelligence*, pp. 329–343. Springer, 2007. 2

[60] P. SCHACHTE, H. SØNDERGAARD, L. WHITING, AND K. HENSHALL: Information loss in knowledge compilation. 2009. Submitted for publication.

[61] T. J. SCHAEFER: The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pp. 216–226. ACM Press, 1978. 4, 9

[62] E. SCHRÖDER: *Der Operationskreis des Logikkalkuls*. B. G. Teubner, Leibzig, Germany, 1877.

[63] B. SELMAN AND H. KAUTZ: Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996. 2

[64] A. SIMON, A. KING, AND J. M. HOWE: Two variables per linear inequality as an abstract domain. In M. LEUSCHEL, editor, *Logic Based Program Development and Transformation*, volume 2664 of *LNCS*, pp. 71–89. Springer, 2002.

[65] M. WARD: The closure operators of a lattice. *Annals of Mathematics*, 43(2):191–196, 1942. 4

[66] B. ZANUTTINI: Approximating propositional knowledge with affine formulas. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI'02)*, pp. 287–291. IOS Press, 2002. 2, 4, 5, 7, 13, 20

[67] B. ZANUTTINI: Approximation of relations by propositional formulas: Complexity and semantics. In S. KOENIG AND R. HOLTE, editors, *Abstraction, Reformulation and Approximation: Proceedings of SARA 2002*, volume 2371 of *Lecture Notes in Artificial Intelligence*, pp. 242–255. Springer, 2002. 2

AUTHORS

Kevin Henshall
Department of Computer Science and Software Engineering
The University of Melbourne
Vic. 3010, Australia

Peter Schachte
Department of Computer Science and Software Engineering
The University of Melbourne
Vic. 3010, Australia

Harald Søndergaard
Department of Computer Science and Software Engineering
The University of Melbourne
Vic. 3010, Australia

Leigh Whiting
Department of Computer Science and Software Engineering
The University of Melbourne
Vic. 3010, Australia