Boolean Approximation Revisited

Peter Schachte* and Harald Søndergaard

*NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Vic. 3010, Australia

{schachte,harald}@csse.unimelb.edu.au

Abstract. Most work to date on Boolean approximation assumes that Boolean functions are represented by formulas in conjunctive normal form. That assumption is appropriate for the classical applications of Boolean approximation but potentially limits wider use. We revisit, in a lattice-theoretic setting, so-called envelopes and cores in propositional logic, identifying them with upper and lower closure operators, respectively. This leads to recursive representation-independent characterisations of Boolean approximation for a large class of classes. We show that Boolean development can be applied in a representation-independent setting to develop approximation algorithms for a broad range of Boolean classes, including Horn and Krom functions.

1 Introduction

Since the seminal work by Selman and Kautz [22] there has been considerable interest in Horn approximations of propositional formulas. The original motivation for Horn approximation was the fact that it could allow faster query answering with propositional knowledge bases. But concepts of Boolean function "approximation" and "abstraction" are found in other fields of computer science, including computational learning, symbolic problem-solving, property testing and program analysis. A typical task in any of these may be to find the best, say, monomial, monotone, or Horn theory supporting a given set of data, or "covering" a given theory.

Selman and Kautz's idea of querying (and performing deductions from) upper and lower Horn approximations of a knowledge-base has subsequently been adapted and extended in various directions, and additional uses of Horn approximation have been suggested. Most notable is the recent contribution by del Val [8]. Del Val shows that Kautz and Selman's Horn envelope algorithm carries over to all Boolean function classes closed under subsumption and he proposes an improved algorithm that is applicable if, additionally, the complement of a class is closed under resolution. Moreover, del Val discusses the case of first-order predicate logic, showing how the original concepts can be extended in this direction too. Note that the concepts that are central to del Val [8], such as closure under subsumption and resolution, reflect the underlying assumption of clausal-form representation.

Zanuttini [24, 25] discusses the use of other classes of Boolean functions for approximation, in particular affine functions, and also the use of approximations in the setting of abduction. It is argued that affine approximations have certain advantages over Horn approximations, most notably the fact that they do not blow out in size. (Note, however, that the affine functions are very unevenly distributed across the lattice of Boolean functions, having only sets of models whose cardinality is 2^n for some n. Hence with affine approximation, roughly speaking the "weakest half" of the Boolean functions are *all* approximated to the vacuous function "true".) Zanuttini [24] proves that the affine envelope of a relation $R \in \{0,1\}^n$ can be computed in time $O(|R|n^3 + n^4)$. He recalls a result from Dechter and Pearl [7] that the Krom envelope can be computed in time $O(|R|n^2)$.

There have been proposals for representations other than clausal form, such as characteristic models [13]. Horiyama and Ibaraki [11] suggest the use of ROBDDs for knowledge bases and give algorithms to recognise unate and Horn functions represented as ROBDDs. Schachte and Søndergaard [20] give algorithms for the approximation of Boolean functions represented as ROBDDs, covering several classes, including monotone and Horn functions. Khardon [15] and Horiyama and Ibaraki [12] are concerned with the translation between different representations and establish many interesting results.

A large body of work (see for example Cadoli and Scarcello [3]) is primarily interested in the problem of finding maximal lower Horn approximations. While the results in this paper also apply to lower approximations, we are only interested in the case where approximations are unique, and so we make no contribution to the particular discussion about lower Horn bounds.

A large variety of special classes of Boolean functions, including Horn, are used in program analysis, to automatically reason about runtime properties of programs. In all kinds of static program analysis, approximation plays a pivotal role. The runtime properties of interest are almost always undecidable, so reasoning is necessarily approximate, and abstraction is therefore integral to the definition of a program analysis. Boolean approximation, in the sense of calculating the strongest logical consequence, in some class, of a given Boolean function, is used in at least two different ways. One is to accelerate convergence of the analysis via so-called widening [6]. The other is where approximation finds a role in basic operations on "runtime state descriptions", as it happens in set-sharing analysis for logic programs. In one view [4] this analysis uses positive Boolean functions (those that evaluate to true when all arguments are true) to express how the instantiation of one variable may affect other variables. For example, the formula $x \leftrightarrow y$ would express the constraint that any goal that further instantiated program variable `x` would necessarily further instantiate `y`.

The ubiquity of applications of propositional logic, together with the fact that concepts of Boolean function approximation are found in many different fields of computer science, suggests that it may be fruitful to revisit the approximation problem outside the context of clausal-form representations. In this paper we consider representation-independent aspects of approximation, as well

as algorithms for Boolean approximation that can use a variety of data structures to represent Boolean functions. We view the approximation problem under a lattice-theoretic lens and suggest a general approach to finding approximation algorithms for an important class of classes.

We assume the reader is familiar with propositional logic and elementary lattice and order theory. Section 2 gives relevant definitions and introduces some Boolean function classes of interest. In Section 3 we discuss the view of Boolean classes as closure operators more formally. Section 4 introduces a class of classes, for which a general approach to finding approximations is possible, and we explain the approach. In Section 5 we instantiate the general characterisation to different classes, including Horn, Krom, monotone and antitone functions. Section 6 concludes.

2 Preliminaries: Boolean functions

Let $\mathcal{B} = \{0, 1\}$ and let $\mathcal{V}$ be a countably enumerable set of variables. A *valuation* $\mu : \mathcal{V} \to \mathcal{B}$ is an 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 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 $x = 0 \vee y = 1$. $\mathbf{B}$ 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 1 denote the largest element of $\mathbf{B}$ (that is, $\lambda\mu.1$) as well as of $\mathcal{B}$. Similarly 0 also denotes the smallest element of $\mathbf{B}$ (that is, $\lambda\mu.0$) as well as of $\mathcal{B}$.

A valuation μ is a *model* for φ, denoted $\mu \models \varphi$, if $\varphi(\mu) = 1$. 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,

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

Also, to facilitate a definition (in Section 4) of "unbiased" Boolean function classes, that is, classes defined without reference to any *specific* variables, we define the concept of "swapping" variables in a valuation:

$$\mu_{[x \leftrightarrows y]}(v) = \begin{cases} \mu(y) & \text{if } v = x \\ \mu(x) & \text{if } v = y \\ \mu(v) & \text{otherwise} \end{cases}$$

We lift this to apply to Boolean functions by defining $\varphi_{[x \leftrightarrows y]}(\mu) = \varphi(\mu_{[x \leftrightarrows y]})$. That is, $\varphi_{[x \leftrightarrows y]}$ simultaneously replaces all occurrences of x in φ with y and occurrences of y with x.

For $\varphi \in \mathbf{B}$ we use $\overline{\varphi}$ to denote φ's negation. Let $\overline{\mathcal{V}} = \{\overline{x} \mid x \in \mathcal{V}\}$ be the set of negated variables. A *literal* is a member of the set $\mathcal{V} \cup \overline{\mathcal{V}}$, that is, a variable or a negated variable. We use φ^i_x to stand for φ with x instantiated to i, that

is, $\varphi^i_x(\mu) = \varphi(\mu[x \mapsto i])$. We say that φ is *independent* of literal x (and also of literal $\overline{x}$) when $\varphi^0_x = \varphi^1_x = \varphi$, and we write this $\varphi \nsim x$. We say that φ *depends* on x iff φ is not independent of x.

The *dual* of a Boolean function φ is the function that is obtained by interchanging the roles of the truth values *0* and *1*. A simple way of turning a formula for φ into a formula for φ's dual is to change the sign of every literal in φ and negate the whole resulting formula. For example, the dual of $x \wedge (\overline{y} \vee z)$ is $x \vee (\overline{y} \wedge z)$ — De Morgan's laws can be regarded as duality laws.

Define $\widetilde{\varphi}$ as the dual of $\overline{\varphi}$. Following Halmos [10], we call $\widetilde{\varphi}$ the *contra-dual* of φ. Clearly, given a formula for φ, a formula for $\widetilde{\varphi}$ is obtained by changing the sign of each literal in φ. As an example, if $\varphi = (x \leftrightarrow y) \rightarrow z$ then $\widetilde{\varphi} = (x \leftrightarrow y) \rightarrow \overline{z}$. Given a truth table for a Boolean function, the truth table for its contra-dual is obtained by turning the result column upside down. The mapping $\varphi \mapsto \widetilde{\varphi}$ is an involution, and monotone: $\psi \models \varphi$ iff $\widetilde{\psi} \models \widetilde{\varphi}$. For any class $\Delta \subseteq \mathbf{B}$, we let $\widetilde{\Delta}$ denote the class $\{\widetilde{\varphi} \mid \varphi \in \Delta\}$.

Function classes Δ central to this paper include:

H: A *Horn* function is one whose set of models is closed under pointwise conjunction. That is, **H** (and only **H**) functions φ satisfy the requirement that for all valuations μ and μ', if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \wedge \mu' \models \varphi$. **H** is the set of functions that can be written in conjunctive normal form $\bigwedge(\ell_1 \vee \cdots \vee \ell_n)$, $n \geq 0$, with at most one positive literal ℓ per clause.

$\widetilde{\mathbf{H}}$: A *contra-dual Horn* function φ satisfies the requirement that for all valuations μ and μ', if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \vee \mu' \models \varphi$. A $\widetilde{\mathbf{H}}$ function can be written in CNF with each clause containing at most one negative literal.

M: A *monotone* function φ satisfies the requirement that for all valuations μ and μ', $\mu \vee \mu' \models \varphi$ when $\mu \models \varphi$. Here $\vee$ denotes pointwise disjunction. Monotone functions are sometimes referred to as *isotone*. Syntactically the class is most conveniently described as the functions generated by $\{\wedge, \vee, 0, 1\}$, see for example Rudeanu's [19] Theorem 11.3.

$\widetilde{\mathbf{M}}$: An *antitone* function φ has the property that, for all valuations μ and μ', if $\mu \models \varphi$ then $\mu \wedge \mu' \models \varphi$.

K: A *Krom* function is one whose set of models is closed under pointwise application of the majority-of-3 function $\lambda x, y, z.(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$. It is generated by formulas in CNF with at most two literals per clause, and its members are also referred to as 2-CNF or *bijunctive*.

L: This is the class 1-CNF consisting of functions that can be written as conjunctions of single-literal (or empty) clauses.

S: This is the intersection of **H** and $\widetilde{\mathbf{H}}$, that is, **L** extended with simple dependencies of the form $x \rightarrow y$.

V: This is **L** restricted to positive literals.

$\widetilde{\mathbf{V}}$: This is **L** restricted to negative literals.

P: A *positive* function is one that is satisfied by the unit valuation $\lambda v.1$.

D: A *definite* function is one that is both positive and Horn.

C: This class consists of the constant functions *0* and *1*.

1: This is the class consisting of the constant function *1* only.

All of these classes, apart from **P**, **D**, and **1**, contain *0*. It is immediate that $\mathbf{M} \subseteq \widetilde{\mathbf{H}}$ and $\widetilde{\mathbf{M}} \subseteq \mathbf{H}$. Figure 1 shows the classes as a Hasse diagram, ordered by the subset ordering.

These classes find widespread use in computer science and several play central roles in the theory of propositional expressiveness, in computational complexity theory, or both. **M** and **P** are the classes "A:a" and "β" of Post's functional completeness result [18] (made more accessible by Pelletier and Martin [17]), Post's remaining three classes being the dual of **P**, the alternating functions, and the self-dual functions. Schaefer's celebrated SAT dichotomy result [21] makes use of six classes, five of which are: **H** and $\widetilde{\mathbf{H}}$ (called "weakly negative" and "weakly positive" respectively), **P** and its contra-dual ("1-valid" and "0-valid"), and **K** (or "bijunctive"). The sixth is the class of affine functions.

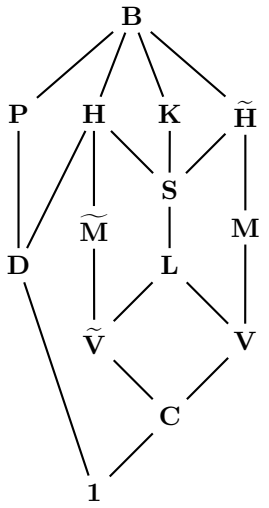

Fig. 1. Boolean function classes

3 Approximation as closure operators

We are interested in the problem of approximating a Boolean function φ, in the sense of finding, when it exists, the strongest function ψ from a given class Δ, entailed by φ. This approximation is sometimes referred to as the "Δ *envelope*" of φ [14]. We denote this $\Delta_{\uparrow}(\varphi)$. Also of interest, for some classes Δ, is the weakest function $\psi \in \Delta$ which entails φ. Such a ψ is sometimes referred to as the "Δ *core*" of φ, denoted $\Delta_{\downarrow}(\varphi)$.

What are the essential properties of an approximation operator ρ, whether it produces envelopes or cores? A first natural requirement is that it is *idempotent*, that is, $\rho(\varphi) = \rho(\rho(\varphi))$ for all $\varphi \in \mathbf{B}$. In other words, ρ acts instantaneously and is the identity function on the set of approximations $\rho(\mathbf{B})$. A second natural requirement is that it is *monotone*, that is, $\varphi \models \varphi'$ implies $\rho(\varphi) \models \rho(\varphi')$ for all Boolean functions φ and φ'. In other words, ρ preserves entailment and thus does not squander information. Together the two requirements say that ρ is a *retraction*.

The only requirement remaining is the one that provides a direction for the approximation. An *upper* approximation operator (yielding "envelopes") is *extensive*, that is, $\varphi \models \rho(\varphi)$ for all φ. A *lower* approximation operator (yielding

"cores") is *reductive*, that is, $\rho(\varphi) \models \varphi$ for all φ. An extensive retraction is known as an *upper closure operator*, or uco, and a reductive retraction is a *lower closure operator*, or lco. Retractions exists that are neither upper nor lower approximation operators, but they are not of interest here.

All these concepts are well known in lattice and order theory [1]. The definition of approximation operators make sense for operators defined on lattices more generally—$\mathbf{B}$ is just a special case.

3.1 Upper closure operators

General properties of closure operators [5, 23] hold for $\mathbf{B}$. If $\rho : \mathbf{B} \to \mathbf{B}$ is a uco then $\rho(\mathbf{B})$ is a (complete) lattice with least element $\rho(0)$, greatest element 1, greatest lower bound operation $\bigwedge$, and least upper bound operation $\lambda S.\rho(\bigvee S)$. It is a sublattice of L if and only if ρ is additive, that is, $\rho(\bigvee S) = \bigvee \rho(S)$ for all $S \subseteq \mathbf{B}$. In any case,

$$\rho(\bigwedge S) \quad \models \quad \bigwedge \rho(S) \quad = \quad \rho(\bigwedge \rho(S)) \tag{1}$$

$$\bigvee \rho(S) \quad \models \quad \rho(\bigvee S) \quad = \quad \rho(\bigvee \rho(S)) \tag{2}$$

It follows that $\rho(\mathbf{B})$ always contains 1 and is closed under conjunction.[1] Conversely, any $\Delta \subseteq \mathbf{B}$ containing 1 and closed under conjunction uniquely determines a uco, defined by

$$\Delta_\uparrow(\varphi) = \bigwedge\{\psi \mid \psi \in \Delta \text{ and } \varphi \models \psi\}$$

A family of ucos $\{\exists_v\}_{v \in Var}$ is given by existential quantification on $\mathbf{B}$: $\exists_x = \lambda\varphi \,.\, \exists x.\varphi$ is a uco, as is easily verified.

Given two upper closure operators ρ and ρ' on $\mathbf{B}$, $\rho \circ \rho'$ need not be an upper closure operator. For example, with the uco ρ defined by

$$\rho(\varphi) = \begin{cases} x & \text{if } \varphi \models x \\ 1 & \text{otherwise} \end{cases}$$

$\rho \circ \exists_x$ is not idempotent, as $\rho(\exists_x(0)) = x \neq 1 = \exists_x(\rho(x))$. However, if $\rho \circ \rho' = \rho' \circ \rho$ then the composition is also an upper closure operator, and $\rho(\rho'(\mathbf{B})) = \rho'(\rho(\mathbf{B})) = \rho(\mathbf{B}) \cap \rho'(\mathbf{B})$ [9, 16].

Proposition 1. Let $\Delta_\uparrow$ be a uco on $\mathbf{B}$. If $\exists_x \circ \Delta_\uparrow = \Delta_\uparrow \circ \exists_x$ then Δ is closed under $\exists_x$.

Proof: Assume that $\exists_x \circ \Delta_\uparrow = \Delta_\uparrow \circ \exists_x$. For $\varphi \in \Delta$ we have $\exists_x(\varphi) = \exists_x(\Delta_\uparrow(\varphi)) = \Delta_\uparrow(\exists_x(\varphi))$. Hence $\exists_x(\varphi)$ is in Δ. ∎

[1] These consequences are also easy to show directly: Since $1 \models \rho(1)$, $\rho(1) = 1$. Moreover, by monotonicity, $\rho(\varphi \wedge \varphi')$ entails both $\rho(\varphi)$ and $\rho(\varphi')$, and so $\rho(\varphi \wedge \varphi') \models \rho(\varphi) \wedge \rho(\varphi')$. If φ and φ' are fixed points for ρ then the last statement reduces to $\rho(\varphi \wedge \varphi') \models \varphi \wedge \varphi'$. Since $\varphi \wedge \varphi' \models \rho(\varphi \wedge \varphi')$, $\varphi \wedge \varphi'$ is also a fixed point. In other words, if φ and φ' are in $\rho(\mathbf{B})$, so is $\varphi \wedge \varphi'$.

The converse does not hold. Take $\mathbf{P}$, that class of Boolean functions satisfied by the unit valuation $\lambda v.\ \mathit{1}$. As $\exists_x(\varphi) = \varphi_x^{\mathit{0}} \vee \varphi_x^{\mathit{1}}$, and φ_x^{1} is in $\mathbf{P}$ whenever φ is, $\mathbf{P}$ is closed under $\exists_x$. However,

$$\exists_x(\mathbf{P}_\uparrow(\mathit{0})) = \exists_x(\bigwedge_{v\in\mathcal{V}} v) = \bigwedge_{v\in\mathcal{V}\setminus\{x\}} v \neq \bigwedge_{v\in\mathcal{V}} v = \mathbf{P}_\uparrow(\mathit{0}) = \mathbf{P}_\uparrow(\exists_x(\mathit{0}))$$

Also note that $\exists_x \circ \Delta_\uparrow = \Delta_\uparrow \circ \exists_x$ does not imply closure under instantiation. The uco induced by $\mathbf{P} \cup \{\mathit{0}\}$ has the former property, but is not closed under instantiation—for example, $(y \to x)_x^{\mathit{0}} = \overline{y}$.

3.2 Lower closure operators

We can develop analogous results for lower closure operators. We shall not do that in detail, but note that a class of Boolean functions induced by an lco contains $\mathit{0}$ and is closed under disjunction. Conversely, any $\Delta \subseteq \mathbf{B}$ containing $\mathit{0}$ and closed under disjunction uniquely determines an lco

$$\Delta_\downarrow(\varphi) = \bigvee\{\psi \mid \psi \in \Delta \text{ and } \psi \models \varphi\}$$

A family of lcos is given by universal quantification, namely $\lambda\varphi\ .\ \forall x.\varphi$ is an lco.

3.3 Boolean development

The characterisations of envelopes that we develop in the next section have come about by considering how closure operators can be applied to Boolean functions expressed through Boolean development, that is, the principle[2] that

$$\varphi = (\overline{x} \wedge \varphi_x^{\mathit{0}}) \vee (x \wedge \varphi_x^{\mathit{1}}) \tag{3}$$

or, by duality,

$$\varphi = (\overline{x} \vee \varphi_x^{\mathit{1}}) \wedge (x \vee \varphi_x^{\mathit{0}}) \tag{4}$$

The latter form proves more useful in the context of upper closure operators.

4 Computing approximations

The approximation techniques presented in this paper apply to a broad range of Boolean classes. However, some restrictions must be imposed to permit the approach to work.

[2] The principle, also known as Shannon expansion, goes back to Boole, albeit in the equivalent form $\varphi = (\overline{x} \wedge \varphi_x^{\mathit{0}}) + (x \wedge \varphi_x^{\mathit{1}})$ where '+' denotes exclusive or.

4.1 Decomposable and unbiased classes

Definition 1. We say a set $\Delta \subseteq \mathbf{B}$ is *unbiased* iff for all $\psi \in \Delta$ and variables $x, y \in \mathcal{V}$, $\psi_{[x \leftrightarrows y]} \in \Delta$. ∎

Thus an unbiased class does not treat any variable differently than any other. An example of a class that is not unbiased is the class of functions that entail y. However, all the usual Boolean classes are unbiased, and restricting our attention to unbiased classes is not a significant limitation on the applicability of our approach. "Unbiased" and "closed under existential quantification" are independent concepts: neither entails the other or its negation.

Definition 2. We say a class $\Delta \subseteq \mathbf{B}$ is *decomposable* iff for any $\varphi, \psi \in \mathbf{B}$ and variable $x \in \mathcal{V}$, if $(x \vee \varphi) \wedge (\overline{x} \vee \psi) \in \Delta$ then $x \vee \varphi \in \Delta$ and $\overline{x} \vee \psi \in \Delta$. ∎

Most well-known classes of Boolean functions are decomposable. For example, **P**, **H**, **M**, and **K** are decomposable. Some classes, however, are not decomposable. Section 2 mentioned the alternating and affine classes, and these fall outside the scope of our method. To see that the alternating, and hence affine, classes are not decomposable, note that $(x \vee y) \wedge (\overline{x} \vee \overline{y})$ is alternating, but neither conjunct is.

4.2 Quotient classes

In the following definitions, we shall make use of certain classes, which we call quotient classes, related to the class to which we want to approximate. We shall see that, if we can approximate to a class's quotient classes, we can approximate to the class. Happily, a class's quotient classes are generally easier to approximate to than the class itself, as will be seen in Section 5.

Definition 3. For each class $\Delta \subseteq \mathbf{B}$ we define the following quotient classes:

$$\Delta^{\vee} = \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \nsim x, (x \vee \psi) \in \Delta\}$$
$$\Delta^{\bar{\vee}} = \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \nsim x, (\overline{x} \vee \psi) \in \Delta\}$$
$$\Delta^{\wedge} = \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \nsim x, (x \wedge \psi) \in \Delta\}$$
$$\Delta^{\bar{\wedge}} = \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \nsim x, (\overline{x} \wedge \psi) \in \Delta\}$$
$$\Delta^{\mathbf{C}} = \Delta \cap \mathbf{C} \quad ∎$$

Note that $\Delta^{\mathbf{C}}$ is $\mathbf{0}$, $\mathbf{1}$, or $\mathbf{C}$, according as *0*, *1*, or both are in $\mathbf{C}$. The next results shows that a quotient class is a closure operator when the original class is.

Proposition 2. For any $\Delta \subseteq \mathbf{B}$, if Δ is closed under conjunction and includes *1*, then the same is true of $\Delta^{\vee}$, $\Delta^{\bar{\vee}}$, and $\Delta^{\mathbf{C}}$. Similarly, if $0 \in \Delta$ and Δ is closed under disjunction, then the same is true of $\Delta^{\wedge}$, $\Delta^{\bar{\wedge}}$, and $\Delta^{\mathbf{C}}$.

Proof: Both claims trivially hold for $\Delta^{\mathbf{C}}$, and $1 \in \Delta^{\vee}$, $1 \in \Delta^{\bar{\vee}}$, $0 \in \Delta^{\wedge}$, and $0 \in \Delta^{\bar{\wedge}}$ by construction.

We prove $\Delta^{\vee}$ is closed under conjunction when Δ is; the proof for the other classes is similar. Let $\Delta \subseteq \mathbf{B}$ be any class closed under conjunction and ψ, ψ' be any members of $\Delta^{\vee}$, and $x \in \mathcal{V}$ be any variable independent of ψ and ψ'. Then $x \vee \psi$ and $x \vee \psi'$ are in Δ, and so $(x \vee \psi) \wedge (x \vee \psi') = x \vee (\psi \wedge \psi')$ is in Δ. It follows that $\psi \wedge \psi' \in \Delta^{\vee}$. ∎

4.3 The approximation scheme

Recall that for any $\Delta \subseteq \mathbf{B}$ such that Δ is closed under conjunction, we can define

$$\Delta_{\uparrow}(\varphi) = \bigwedge \{\psi \mid \psi \in \Delta \text{ and } \varphi \models \psi\}$$

and for any $\Delta \subseteq \mathbf{B}$ closed under disjunction, we can define

$$\Delta_{\downarrow}(\varphi) = \bigvee \{\psi \mid \psi \in \Delta \text{ and } \psi \models \varphi\}$$

Unfortunately, these definitions do not readily lend themselves to practical implementation. However, if we restrict our attention to unbiased decomposable classes, the following equivalent definitions, which are readily implemented, can be used.

Definition 4. For $\Delta \subseteq \mathbf{B}$ and $\varphi \in \mathbf{B}$, we define:

$$\mathcal{U}(\varphi) = \bigwedge_{x \in \mathcal{V}} \left((\Delta^{\vee}_{\uparrow}(\varphi^0_x) \vee x) \wedge (\Delta^{\bar{\vee}}_{\uparrow}(\varphi^1_x) \vee \overline{x}) \right) \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$

$$\mathcal{L}(\varphi) = \bigvee_{x \in \mathcal{V}} \left((\Delta^{\wedge}_{\downarrow}(\varphi^0_x) \wedge x) \vee (\Delta^{\bar{\wedge}}_{\downarrow}(\varphi^1_x) \wedge \overline{x}) \right) \vee \Delta^{\mathbf{C}}_{\downarrow}(\varphi)$$

Now we show that, for decomposable closure operators, these definitions indeed specify the Δ envelope and core, respectively.

Theorem 1. For any unbiased decomposable class $\Delta \subseteq \mathbf{B}$ such that $1 \in \Delta$ and Δ is closed under conjunction, $\Delta_{\uparrow}(\varphi) = \mathcal{U}(\varphi)$, and for any unbiased decomposable class $\Delta \subseteq \mathbf{B}$ closed under disjunction and including 0, $\Delta_{\downarrow}(\varphi) = \mathcal{L}(\varphi)$.

Proof: We prove only the first part; the second part is its dual. Assume unbiased decomposable class $\Delta \subseteq \mathbf{B}$ is closed under conjunction.

$$\Delta_{\uparrow}(\varphi) = \bigwedge \{\zeta \mid \zeta \in \Delta \text{ and } \varphi \models \zeta\}$$

For every ζ except 0, we can develop any variable. We handle 0 separately.

$$= \bigwedge_{v \in \mathcal{V}} \bigwedge \left\{ \begin{array}{c} (v \vee \psi) \\ \wedge\ (\overline{v} \vee \psi') \end{array} \;\middle|\; \begin{array}{c} (v \vee \psi) \wedge (\overline{v} \vee \psi') \in \Delta, \psi, \psi' \nsim v \\ \text{and } \varphi \models (v \vee \psi) \wedge (\overline{v} \vee \psi') \end{array} \right\} \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$

Because Δ is decomposable, we can divide the class membership condition. We can also divide the entailment condition, so we can divide the entire set comprehension into positive and negative halves.

$$= \bigwedge_{v \in \mathcal{V}} \left(\begin{array}{c} \bigwedge\{v \vee \psi \mid v \vee \psi \in \Delta, \psi \nsim v \text{ and } \varphi \models v \vee \psi\} \\ \wedge \bigwedge\{\overline{v} \vee \psi \mid \overline{v} \vee \psi \in \Delta, \psi \nsim v \text{ and } \varphi \models \overline{v} \vee \psi\} \end{array} \right) \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$

Δ is unbiased and $\psi \nsim v$, so $v \vee \psi \in \Delta$ iff $\forall u.u \vee \psi \in \Delta$, and similarly for $\overline{v}$. Also, $\varphi \models v \vee \psi$ exactly when $\overline{v} \wedge \varphi \models \psi$.

$$= \bigwedge_{v \in \mathcal{V}} \left(\begin{array}{c} \bigwedge\{v \vee \psi \mid \forall u.u \vee \psi \in \Delta, \psi \nsim v \text{ and } \overline{v} \wedge \varphi \models \psi\} \\ \wedge \bigwedge\{\overline{v} \vee \psi \mid \forall u.\overline{u} \vee \psi \in \Delta, \psi \nsim v \text{ and } v \wedge \varphi \models \psi\} \end{array} \right) \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$

Since the first set collects $v \vee \psi$, cases of ψ making v false do not matter to the result, and conversely for the second set. For these cases, we need consider only consequences of φ^0_v (φ^1_v in the second set). We also observe that the class membership constraint in each set exactly specifies a quotient class. Finally, we factor out the common $v\vee$ or $\overline{v}\vee$ from each set.

$$= \bigwedge_{v \in \mathcal{V}} \left(\begin{array}{c} \left(v \vee \bigwedge\{\psi \mid \psi \in \Delta^{\vee} \text{ and } \psi \nsim v \text{ and } \varphi^0_v \models \psi\}\right) \\ \wedge \left(\overline{v} \vee \bigwedge\{\psi \mid \psi \in \Delta^{\bar{\vee}} \text{ and } \psi \nsim v \text{ and } \varphi^1_v \models \psi\}\right) \end{array} \right) \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$

Each set exactly specifies a quotient upper closure operator.

$$= \bigwedge_{v \in \mathcal{V}} \left((v \vee \Delta^{\vee}_{\uparrow}(\varphi^0_v)) \wedge (\overline{v} \vee \Delta^{\bar{\vee}}_{\uparrow}(\varphi^1_v)) \right) \wedge \Delta^{\mathbf{C}}_{\uparrow}(\varphi)$$
$$= \mathcal{U}(\varphi)$$

4.4 Closure under instantiation

The algorithms of del Val [8] apply only to classes closed under subsumption. This concept presupposes a clausal representation; from a representation-independent perspective, the equivalent concept is closure under instantiation.

Definition 5. We say a class $\Delta \subseteq \mathbf{B}$ is *closed under instantiation* when for every $\psi \in \Delta$ and $v \in \mathcal{V}$, $\psi^0_v \in \Delta$ and $\psi^1_v \in \Delta$. ∎

While many important classes, such as **H**, **M**, **K**, and the affine functions are closed under instantiation, some well-known and important classes are not. For example, we can see that both **P** and **D** are not closed under instantiation by observing that $x \to y$ is both positive and definite, but instantiating y to *0* leaves $\overline{x}$, which is neither positive nor definite.

The characterisations in Section 4.3 do not require closure under instantiation. However, we note an interesting characteristic of classes that do happen to be closed under instantiation.

Proposition 3. For any class $\Delta \subseteq \mathbf{B}$ closed under instantiation, all the quotient classes are subsets of Δ.

Proof: Firstly, $\Delta^{\mathbf{C}} \subseteq \Delta$ by construction. To see that $\Delta^{\vee} \subseteq \Delta$, consider some Δ closed under instantiation, $x \in \mathcal{V}$, and $\psi \in \mathbf{B}$ such that $x \vee \psi \in \Delta$ and $\psi \nsim x$; we must show that $\psi \in \Delta$. By closure under instantiation, $(x \vee \psi)^0_x \in \Delta$. But $(x \vee \psi)^0_x = \psi^0_x$, and because $\psi \nsim x$, $\psi^0_x = \psi$. Thus $\psi \in \Delta$. The argument for the other quotient classes is similar. ∎

5 Instantiating the scheme

We can now use the representation-independent proposition from Section 4 to express envelopes for a number of interesting classes. As is easily verified, $\mathbf{H}$, $\widetilde{\mathbf{H}}$, $\mathbf{M}$, $\widetilde{\mathbf{M}}$, $\mathbf{K}$, $\mathbf{L}$, $\mathbf{S}$, $\mathbf{P}$, $\mathbf{D}$, $\mathbf{C}$, and $\mathbf{1}$ are all decomposable and unbiased, and all contain *1* and are closed under conjunction.

5.1 Expressing the envelopes

First we shall present the quotients of these classes, and then the characterisations of approximation that arise.

Proposition 4.

$\mathbf{H}^{\vee} = \widetilde{\mathbf{M}}$	$\mathbf{H}^{\bar{\vee}} = \mathbf{H}$	$\mathbf{H}^{\mathbf{C}} = \mathbf{C}$
$\widetilde{\mathbf{H}}^{\vee} = \widetilde{\mathbf{H}}$	$\widetilde{\mathbf{H}}^{\bar{\vee}} = \mathbf{M}$	$\widetilde{\mathbf{H}}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{M}^{\vee} = \mathbf{M}$	$\mathbf{M}^{\bar{\vee}} = \mathbf{1}$	$\mathbf{M}^{\mathbf{C}} = \mathbf{C}$
$\widetilde{\mathbf{M}}^{\vee} = \mathbf{1}$	$\widetilde{\mathbf{M}}^{\bar{\vee}} = \widetilde{\mathbf{M}}$	$\widetilde{\mathbf{M}}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{K}^{\vee} = \mathbf{L}$	$\mathbf{K}^{\bar{\vee}} = \mathbf{L}$	$\mathbf{K}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{L}^{\vee} = \mathbf{C}$	$\mathbf{L}^{\bar{\vee}} = \mathbf{C}$	$\mathbf{L}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{S}^{\vee} = \widetilde{\mathbf{V}}$	$\mathbf{S}^{\bar{\vee}} = \mathbf{V}$	$\mathbf{S}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{V}^{\vee} = \mathbf{C}$	$\mathbf{V}^{\bar{\vee}} = \mathbf{1}$	$\mathbf{V}^{\mathbf{C}} = \mathbf{C}$
$\widetilde{\mathbf{V}}^{\vee} = \mathbf{1}$	$\widetilde{\mathbf{V}}^{\bar{\vee}} = \mathbf{C}$	$\widetilde{\mathbf{V}}^{\mathbf{C}} = \mathbf{C}$
$\mathbf{P}^{\vee} = \mathbf{B}$	$\mathbf{P}^{\bar{\vee}} = \mathbf{P}$	$\mathbf{P}^{\mathbf{C}} = \mathbf{1}$
$\mathbf{D}^{\vee} = \widetilde{\mathbf{M}}$	$\mathbf{D}^{\bar{\vee}} = \mathbf{D}$	$\mathbf{D}^{\mathbf{C}} = \mathbf{1}$

Proof: All cases follow easily from the well-known syntactic characterisations of the classes and the definitions of the quotient classes. ∎

Now we are ready to show the instantiations of the general characterisation to the individual classes. Note that where a quotient class is $\mathbf{1}$, it can be trivially omitted, as it always returns *1*. Similarly, where a quotient class is $\mathbf{B}$, it need not be applied to its argument as it is the identity operator. Also note that the conjunct $\mathbf{C}_\uparrow(\varphi)$ has no effect if φ is satisfiable, and for unsatisfiable φ, it becomes *0*.

Corollary 1. Let φ be a Boolean function. Then

$$
\begin{array}{rcl}
\mathbf{H}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\widetilde{\mathbf{M}}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{H}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{H}}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\widetilde{\mathbf{H}}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{M}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{M}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\mathbf{M}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{1}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{M}}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\mathbf{1}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\widetilde{\mathbf{M}}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{K}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\mathbf{L}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{L}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{L}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\mathbf{C}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{C}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{S}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\widetilde{\mathbf{V}}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{V}_\uparrow(\varphi^1_v) \vee \overline{v})) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{V}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} (\mathbf{C}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{V}}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} (\mathbf{C}_\uparrow(\varphi^1_v) \vee \overline{v}) \ \wedge\ \mathbf{C}_\uparrow(\varphi) \\
\mathbf{P}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\varphi^0_v \vee v) \ \wedge\ (\mathbf{P}_\uparrow(\varphi^1_v) \vee \overline{v})) \\
\mathbf{D}_\uparrow(\varphi) & = & \bigwedge_{v\in\mathcal{V}} ((\widetilde{\mathbf{M}}_\uparrow(\varphi^0_v) \vee v) \ \wedge\ (\mathbf{D}_\uparrow(\varphi^1_v) \vee \overline{v})) \qquad \blacksquare
\end{array}
$$

Other instances exist, most notably we can generalise $\mathbf{L}$ (1-CNF) and $\mathbf{K}$ (2-CNF) to k-CNF: $(k+1)\text{-CNF}^\vee = (k+1)\text{-CNF}^{\bar{\vee}} = k\text{-CNF}$, and $(k+1)\text{-CNF}^{\mathbf{C}} = \mathbf{C}$. We can similarly extend these results to k-quasi-Horn.

5.2 Algorithmic aspects

Corollary 1 characterises the envelopes for a number of interesting function classes in a representation-independent manner. They do not always suggest the most efficient way of calculating envelopes, which in general depends on how Boolean functions are represented. We also note that there are cases where

an envelope cannot be provided in the absence of information about the "variables of interest". For example, we cannot say what the $\mathbf{D}$ envelope of $\overline{x}$ is, unless we know the set of variables of which $\overline{x}$ is supposed to be a function. Using Church's lambda notation helps; using it we can state for example that $\mathbf{D}_{\uparrow}(\lambda x, y.\overline{x}) = x \to y$ whereas $\mathbf{D}_{\uparrow}(\lambda x, y, z.\overline{x}) = x \to (y \wedge z)$.

It is interesting to compare our characterisations, which were derived using Boolean development, with recursive definitions used for ROBDDs (also resting on Boolean development). We present below the algorithms for $\mathbf{H}$ and $\widetilde{\mathbf{M}}$—algorithms for the other classes can be derived [20].

Recall that binary decision diagrams are defined inductively:

- 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.

and the meaning of a BDD is given as follows.

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

ROBDDs are then BDDs with a fixed variable order, satisfying the constraints that in any BDD $\mathsf{ite}(x, R_1, R_2)$, $R_1 \neq R_2$, and that for any distinct BDDs R_1 and R_2 appearing in R, $[\![R_1]\!] \neq [\![R_2]\!]$. As is common, we use a function $\mathsf{mknd}(x, R_1, R_2)$ to create all ROBDD nodes according to these rules:

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 [2].
3. Otherwise, return $\mathsf{ite}(x, R_1, R_2)$.

Algorithm 1 To find the Horn envelope of an ROBDD:

$$\begin{aligned}
&\mathbf{H}_{\uparrow}(0) = 0 \\
&\mathbf{H}_{\uparrow}(1) = 1 \\
&\mathbf{H}_{\uparrow}(\mathsf{ite}(x, R_1, R_2)) \\
&\qquad = \mathsf{mknd}(x, R^t, R^f) \\
&\qquad\quad \textbf{where}\ \ R' = \mathbf{H}_{\uparrow}(\mathsf{or}(R_1, R_2)) \\
&\qquad\qquad \textbf{and}\ \ R^t = \mathbf{H}_{\uparrow}(R_1) \\
&\qquad\qquad \textbf{and}\ \ R^f = \mathsf{and}(\widetilde{\mathbf{M}}_{\uparrow}(R_2), R')
\end{aligned}$$

$$\begin{aligned}
&\widetilde{\mathbf{M}}_{\uparrow}(0) = 0 \\
&\widetilde{\mathbf{M}}_{\uparrow}(1) = 1 \\
&\widetilde{\mathbf{M}}_{\uparrow}(\mathsf{ite}(x, R_1, R_2)) \\
&\qquad = \mathsf{mknd}(x, R'_1, \mathsf{or}(R'_1, R'_2)) \\
&\qquad\quad \textbf{where}\ \ R'_1 = \widetilde{\mathbf{M}}_{\uparrow}(R_1) \\
&\qquad\qquad \textbf{and}\ \ R'_2 = \widetilde{\mathbf{M}}_{\uparrow}(R_2)
\end{aligned}$$

■

Note that $\widetilde{\mathbf{M}}_{\uparrow}(\varphi \vee \psi) = \widetilde{\mathbf{M}}_{\uparrow}(\varphi) \vee \widetilde{\mathbf{M}}_{\uparrow}(\psi)$.

6 Discussion

Several contributors to the field of approximate knowledge compilation have suggested departures from the classical setting, regarding both the classes of Boolean functions used, and the data structures used to represent these functions. The lattice-theoretic concepts of upper and lower closure operators provide an abstract and useful lens for the study of envelopes and cores in propositional logic, independent of representation. In the first half of this paper we have put forward this view in greater detail. The framework is general. While we focus on lattices of Boolean functions, note that no assumptions were made about the properties of the lattices. In particular they need not be Boolean lattices, that is, they are neither restricted to be complemented nor distributive. Indeed, the majority of the function classes considered do not form complemented lattices, and many are not distributive. For example, to see that **H** is not distributive, note that x, y, and $x \leftrightarrow y$ are all Horn, but

$$(x \sqcup y) \sqcap (x \leftrightarrow y) \;=\; 1 \wedge (x \leftrightarrow y) \;\neq\; x \wedge y \;=\; (x \sqcap (x \leftrightarrow y)) \sqcup (y \sqcap (x \leftrightarrow y))$$

where $\sqcap$ is the meet operation on **H** (that is, conjunction), and $\sqcup$ is the join (which is not disjunction).

Our main contribution, expressed as Theorem 1, is a generic characterisation of envelopes and cores in a large variety of Boolean function classes. Many instantiations of the theorem, including versions for Horn and Krom functions, are provided in Section 5.

It remains to be seen to what extent the algorithms we have derived can be made efficient for various representations. So far we are in the process of implementing a range of the algorithms for ROBDDs, together with algorithms for finding least upper bounds and greatest lower bounds for sets of functions in various classes. (The use of the term "LUB" in much of the literature on Horn approximation is somewhat incongruous with standard usage, and "GLB" even more so.) Another challenge is to develop an algorithm to produce affine envelopes of ROBDDs.

We would also like to better understand the relations between the framework offered by del Val [8] and the one proposed here. For example, at least on the surface it would seem that closure under subsumption corresponds exactly to closure under instantiation (by the latter we mean $\varphi_x^0, \varphi_x^1 \in \Delta$ whenever $\varphi \in \Delta$, for all $x \in \mathcal{V}$). However, we note that our development of the generic algorithm did not require an assumption about closure under instantiation.

References

1. G. Birkhoff. *Lattice Theory.* American Mathematical Society, third edition, 1973.
2. K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Proc. Twenty-seventh ACM/IEEE Design Automation Conf.*, pages 40–45, 1990.
3. M. Cadoli and F. Scarcello. Semantical and computational aspects of Horn approximations. *Artificial Intelligence*, 119:1–17, 2000.

4. 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.
5. P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E. J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 237–277. North-Holland, 1978.
6. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282. ACM Press, 1979.
7. R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 58:237–270, 1992.
8. A. del Val. First order LUB approximations: Characterization and algorithms. *Artificial Intelligence*, 162:7–48, 2005.
9. R. Giacobazzi. *Semantic Aspects of Logic Program Analysis*. PhD thesis, University of Pisa, Italy, 1993.
10. P. R. Halmos. *Lectures on Boolean Algebras*. Springer-Verlag, 1963.
11. T. Horiyama and T. Ibaraki. Ordered binary decision diagrams as knowledge-bases. *Artificial Intelligence*, 136:189–213, 2002.
12. T. Horiyama and T. Ibaraki. Translation among CNFs, characteristic models and ordered binary decision diagrams. *Inf. Processing Letters*, 85:191–198, 2003.
13. H. Kautz, M. Kearns, and B. Selman. Horn approximations of empirical data. *Artificial Intelligence*, 74:129–145, 1995.
14. D. Kavvadias, C. Papadimitriou, and M. Sideri. On Horn envelopes and hypergraph transversals. In K. Ng *et al.*, editor, *Proc. Fourth Int. Symp. Algorithms and Computation*, volume 762 of *LNCS*, pages 399–405. Springer, 1993.
15. R. Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.
16. O. Ore. Combinations of closure relations. *Ann. Math.*, 44(3):514–533, 1943.
17. F. J. Pelletier and N. M. Martin. Post's functional completeness theorem. *Notre Dame Journal of Formal Logic*, 31(2), 1990.
18. E. L. 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.
19. S. Rudeanu. *Boolean Functions and Equations*. North-Holland, 1974.
20. 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*, pages 1–16. Springer, 2006.
21. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. Tenth Ann. ACM Symp. Theory of Computing*, pages 216–226, 1978.
22. B. Selman and H. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.
23. M. Ward. The closure operators of a lattice. *Ann. Math.*, 43(2):191–196, 1942.
24. B. Zanuttini. Approximating propositional knowledge with affine formulas. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI'02)*, pages 287–291. IOS Press, 2002.
25. B. Zanuttini. Approximation of relations by propositional formulas: Complexity and semantics. In S. Koenig and R. Holte, editors, *Proceedings of SARA 2002*, volume 2371 of *Lecture Notes in Artificial Intelligence*, pages 242–255. Springer, 2002.