Closure Operators for ROBDDs

Peter Schachte* and Harald Søndergaard

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

Abstract. Program analysis commonly makes use of Boolean functions to express information about run-time states. Many important classes of Boolean functions used this way, such as the monotone functions and the Boolean Horn functions, have simple semantic characterisations. They also have well-known syntactic characterisations in terms of Boolean formulae, say, in conjunctive normal form. Here we are concerned with characterisations using binary decision diagrams. Over the last decade, ROBDDs have become popular as representations of Boolean functions, mainly for their algorithmic properties. Assuming ROBDDs as representation, we address the following problems: Given a function ψ and a class of functions Δ, how to find the strongest $\varphi \in \Delta$ entailed by ψ (when such a φ is known to exist)? How to find the weakest $\varphi \in \Delta$ that entails ψ? How to determine that a function ψ belongs to a class Δ? Answers are important, not only for several program analyses, but for other areas of computer science, where Boolean approximation is used. We give, for many commonly used classes Δ of Boolean functions, algorithms to approximate functions represented as ROBDDs, in the sense described above. The algorithms implement upper closure operators, familiar from abstract interpretation. They immediately lead to algorithms for deciding class membership.

1 Introduction

Propositional logic is of fundamental importance to computer science. While its primary use has been within switching theory, there are many other uses, for example in verification, machine learning, cryptography and program analysis. In complexity theory, Boolean satisfiability has played a seminal role and provided deep and valuable results.

Our own interest in Boolean functions stems from work in program analysis. In this area, as in many other practical applications of propositional logic, we are not so much interested in solving Boolean equations, as in using Boolean functions to capture properties and relations of interest. In the process of analysing programs, we build and transform representations of Boolean functions, in order to provide detailed information about runtime states.

In this paper we consider various instances of the following problem. Given a Boolean function φ and a class of Boolean functions Δ, how can one decide

* Peter Schachte's work has been supported in part by NICTA Victoria Laboratories.

(efficiently) whether φ belongs to Δ? How does one find the strongest statement in Δ which is entailed by φ (assuming this is well defined)? Answers of course depend on how Boolean functions are represented.

ROBDDs [4] provide a graphical representation of Boolean functions, based on repeated Boolean development, that is, the principle that in any Boolean algebra, $\varphi = (\overline{x} \wedge \varphi_x^0) \vee (x \wedge \varphi_x^1)$ where φ_x^u denotes φ with x fixed to the truth value u. In this paper, ROBDDs are used to represent Boolean functions.

The classes of Boolean functions studied here are classes that have simple syntactic and semantic characterisations. They are are of importance in many areas of computer science, including program analysis. They include the Horn fragment, monotone and antitone Boolean functions, sub-classes of bijunctive functions, and others.

There are many examples of the use of approximation in program analysis. Trivial cases are where a program analysis tool uses intermediate results that are of a finer granularity that what gets reported to a user or used by an optimising compiler. Consider for example classical two-valued strictness analysis [18]. The strictness result for the functional program

```
g(x,y,z) = if even(x) then y/2 else 3*z + 1
```

is calculated to be $x \wedge (y \vee z)$. This contains a disjunctive component indicating that `g` needs at least one of its last two arguments, in addition to the first. This disjunctive information is not useful for a compiler seeking to replace call-by-name by call-by-value—instead of $x \wedge (y \vee z)$ a compiler needs the weaker statement x. (Once we have the definitions of Section 3 we can say that what is needed is the strongest $\mathbf{V}$ consequence of the calculated result.) That the more fine-grained $x \wedge (y \vee z)$ is useful as an intermediate result, however, becomes clear when we consider the function

```
f(u,v) = g(u,v,v)
```

whose strictness result is $u \wedge (v \vee v)$, that is, $u \wedge v$. Without the disjunctive component in `g`'s result, the result for `f` would be unnecessarily weak.

Less trivial cases are when approximation is needed in intermediate results, to guarantee correctness of the analysis. Genaim and King's suspension analysis for logic programs with dynamic scheduling [9] is an example. The analysis, essentially a greatest-fixed-point computation, produces for each predicate p a Boolean formula φ_i expressing groundness conditions under which the atomic formulae in the body of p may be scheduled so as to obtain suspension-free evaluation. In each iteration, the re-calculation of the formulae φ includes a crucial step where φ is replaced by its weakest monotone implicant. Similarly, set-sharing analysis as presented by Codish *et al.* [5] relies on an operation that replaces a positive formula by its strongest definite consequence.

The contribution of the present paper is two-fold. First, we view a range of important classes of Boolean functions from a new angle. Studying these classes of Boolean functions through the prism of Boolean development (provided by ROBDDs) yields deeper insight both into ROBDDs and the classes themselves.

The first three sections of this paper should therefore be of value to anybody with an interest in the theory of Boolean functions. Second, we give algorithms for ROBDD approximation. The algorithms are novel, pleasingly simple, and follow a common pattern. This more practical contribution may be of interest to anybody who works with ROBDDs, but in particular to those who employ some kind of approximation of Boolean functions, as it happens for example in areas of program analysis, cryptography, machine learning and property testing.

The reader is assumed to be familiar with propositional logic and ROBDDs. Section 2 recapitulates ROBDDs and some standard operations, albeit mainly to establish our notation. In Section 3 we define several classes of Boolean functions and establish, for each, relevant properties possessed by their members. Section 4 presents new algorithms for approximating Boolean functions represented as ROBDDs. The aim is to give each method a simple presentation that shows its essence and facilitates a proof of correctness. Section 5 discusses related work and Section 6 concludes.

2 ROBDDs

We briefly recall the essentials of ROBDDs [4]. 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.

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 $\mathsf{vars}(R) = \{v \mid \mathsf{ite}(v, \_, \_) \text{ appears in } R\}$. The meaning of a BDD is given as follows.

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

A BDD is an *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 an *ROBDD* (Reduced Ordered Binary Decision Diagram, [3, 4]) iff for all BDDs R_1 and R_2 appearing in R, $R_1 = R_2$ when $[\![R_1]\!] = [\![R_2]\!]$. Practical implementations [2] 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 ROBDD in diagrammatic form. This ROBDD denotes the function $(x \leftarrow y) \rightarrow z$. An ROBDD $\mathsf{ite}(x, R_1, R_2)$ is depicted 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.

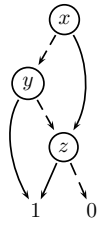

Fig. 1. $(x \leftarrow y) \rightarrow z$

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 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*. See Brace *et al.* [2] for details. We assume a computed table is used for all recursive ROBDD algorithms presented here.

3 Boolean function classes as closure operators

Let $\mathcal{B} = \{0, 1\}$. A Boolean function over variable set $\mathcal{V} = \{x_1, \ldots, x_n\}$ is a function $\varphi : \mathcal{B}^n \rightarrow \mathcal{B}$. We assume a fixed, finite set $\mathcal{V}$ of variables, and use $\mathbf{B}$ to 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, that is, the ordering relation is entailment, $\models$.

The class $\mathbf{C} \subset \mathbf{B}$ contains just the two constant functions. As is common, we use '*0*' and '*1*' not only to denote the elements of $\mathcal{B}$, but also for the elements of $\mathbf{C}$. The class $\mathbf{1} \subset \mathbf{C}$, contains only the element *1*.

A *valuation* $\mu : \mathcal{V} \rightarrow \mathcal{B}$ is an assignment of truth values to the variables in $\mathcal{V}$. Valuations are ordered pointwise. We will sometimes write a valuation as the set of variables which are assigned the value *1*. In this view, the *meet* operation on valuations is set intersection, and the *join* is set union.

A valuation μ is a *model* for φ, denoted $\mu \models \varphi$, if $\varphi(\mu(x_1), \ldots, \mu(x_n)) = 1$. In the "set" view, the set of models of φ is a set of sets of variables, namely

$$[\![\varphi]\!]_{\mathcal{V}} = \left\{ \{x \in \mathcal{V} \mid \mu(x) = 1\} \,\middle|\, \mu \models \varphi \right\}.$$

Again, we will often omit the subscript $\mathcal{V}$ as it will be clear from the context. We will also switch freely amongst the views of φ as a function, a formula, and as its set of models, relying on the reader to disambiguate from context.

We say that a Boolean function φ is *model-meet closed* if, whenever $\mu \models \varphi$ and $\mu' \models \varphi$, we also have $\mu \cap \mu' \models \varphi$. In other words, φ is model-meet closed if $[\![\varphi]\!]$ is closed under intersection. Similarly, φ is *model-join closed* if, whenever $\mu \models \varphi$ and $\mu' \models \varphi$, also $\mu \cup \mu' \models \varphi$. We likewise say that a Boolean function φ is *downward closed* if, whenever $\mu \models \varphi$, we also have $\mu \cap \mu' \models \varphi$ for all valuations μ', and similarly φ is *upward closed* if, whenever $\mu \models \varphi$, we also have $\mu \cup \mu' \models \varphi$ for all valuations μ'. Note that a downward closed function is necessarily model-meet closed, and an upward closed function is model-join closed.

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 $\mathcal{L} = \mathcal{V} \cup \overline{\mathcal{V}}$, that is, a variable or a negated variable. We say that φ is *independent* of literal x (and also of literal $\overline{x}$) when for all models μ of φ, $\mu \setminus \{x\} \models \varphi$ iff $\mu \cup \{x\} \models \varphi$.

The *dual* of a Boolean function φ is the function that is obtained by interchanging the roles of *1* and *0*. 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 φ° as the dual of $\overline{\varphi}$. Following Halmos [13], we call φ° the *contra-dual* of φ. Clearly, given a formula for φ, a formula for φ° is obtained by changing the sign of each literal in φ. As an example, $((x \leftrightarrow y) \rightarrow z)^\circ = (x \leftrightarrow y) \rightarrow \overline{z}$. Alternatively, given a truth table for a Boolean function, the truth table for its contra-dual is obtained by turning the result column upside down. Given an ROBDD R for φ, we can also talk about R's contra-dual, R°, which represents φ°. An ROBDD's contra-dual is obtained by simultaneously making all solid arcs dashed, and all dashed arcs solid.

Clearly the mapping $\varphi \mapsto \varphi^\circ$ is an involution, and monotone: $\psi \models \varphi$ iff $\psi^\circ \models \varphi^\circ$. Note that φ° is model-join closed iff φ is model-meet closed. For any class Δ of Boolean functions, we let Δ° denote the class $\{\varphi^\circ \mid \varphi \in \Delta\}$.

The classes of Boolean functions considered in this paper can all be seen as upper closures of $\mathbf{B}$. Recall that an *upper closure operator* (or just *uco*) $\rho : L \to L$ on a complete lattice L satisfies the following constraints:

- It is monotone: $x \sqsubseteq y$ implies $\rho(x) \sqsubseteq \rho(y)$ for all $x, y \in L$.
- It is extensive: $x \sqsubseteq \rho(x)$ for all $x \in L$.
- It is idempotent: $\rho(x) = \rho(\rho(x))$ for all $x \in L$.

As each class Δ under study contains *1* and is closed under conjunction (this will be obvious from the syntactic characterisations given below), Δ is a lattice. Moreover, the mapping $\rho_\Delta : \mathbf{B} \to \mathbf{B}$, defined by

$$\rho_\Delta(\psi) = \bigwedge \{\varphi \in \Delta \mid \psi \models \varphi\}$$

is a uco on $\mathbf{B}$. Since it is completely determined by Δ, and vice versa, we will usually denote ρ_Δ simply as Δ. The view of such classes (*abstract domains*) Δ as upper closure operators has been popular ever since the seminal papers on abstract interpretation [6, 7].

We list some well-known properties of closure operators [23, 6]. Let L be a complete lattice $(L, \bot, \top, \bigsqcap, \bigsqcup)$ and let $\rho : L \to L$ be a uco. Then $\rho(L)$ is a complete lattice $(\rho(L), \rho(\bot), \top, \bigsqcap, \lambda S.\rho(\bigsqcup S))$. It is a complete sublattice of L if and only if ρ is additive, that is, $\rho(\bigsqcup S) = \bigsqcup \rho(S)$ for all $S \subseteq L$. In any case,

$$\rho(\bigsqcap S) \sqsubseteq \bigsqcap \rho(S) = \rho(\bigsqcap \rho(S)) \tag{1}$$

$$\bigsqcup \rho(S) \sqsubseteq \rho(\bigsqcup S) = \rho(\bigsqcup \rho(S)) \tag{2}$$

Given two upper closure operators ρ and ρ' on the same lattice L, $\rho \circ \rho'$ need not be an upper closure operator. However, if $\rho \circ \rho' = \rho' \circ \rho$ then the composition is also an upper closure operator, and $\rho(\rho'(L)) = \rho'(\rho(L)) = \rho(L) \cap \rho'(L)$ [10, 19].

The Boolean function classes we focus on here are those characterised by combinations of certain interesting semantic properties: model-meet closure, model-join closure, downward closure, and upward closure. Nine classes are spanned, as shown in the Hasse diagram of Figure 2. These classes are chosen for their importance in program analysis, although our method applies to many other natural classes, as we argue in Section 5.

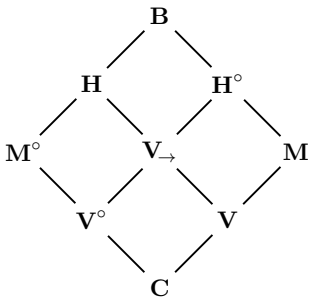

Fig. 2. Boolean function classes

We define $\mathbf{H}$ to be the set of model-meet closed Boolean functions, so $\mathbf{H}^\circ$ is the set of model-join closed functions. Similarly, we define $\mathbf{M}$ to be the upward-closed functions, with $\mathbf{M}^\circ$ the downward-closed functions. We define $\mathbf{V}_\rightarrow$ to be the set of Boolean functions that are both model-meet and model-join closed; *i.e.*, $\mathbf{H} \cap \mathbf{H}^\circ$. In what follows we utilise that $\mathbf{V}_\rightarrow$ is a uco, and therefore $\mathbf{H} \circ \mathbf{H}^\circ = \mathbf{H}^\circ \circ \mathbf{H} = \mathbf{V}_\rightarrow$. We also define $\mathbf{V} = \mathbf{H} \cap \mathbf{M} = \mathbf{V}_\rightarrow \cap \mathbf{M}$ and $\mathbf{V}^\circ = \mathbf{H}^\circ \cap \mathbf{M}^\circ = \mathbf{V}_\rightarrow \cap \mathbf{M}^\circ$, both of which are ucos, as well. Finally, we observe that $\mathbf{C} = \mathbf{M} \cap \mathbf{M}^\circ = \mathbf{V} \cap \mathbf{V}^\circ$ is also a uco. One can think of these elements as classes of functions, ordered by subset ordering, or alternatively, as upper closure operators, ordered pointwise (in particular, $\mathbf{B}$ is the identity function, providing loss-less approximation).

These classes (ucos) have a number of properties in common. All are closed under conjunction and existential quantification. None are closed under negation, and hence none are closed under universal quantification. All are closed under the operation of fixing a variable to a truth value. Namely, we can express instantiation using only conjunction and existential quantification. We write the fixing of x in φ to *0* as $\varphi^0_x \equiv \exists x : \varphi \wedge \overline{x}$ and the fixing of x in φ to *1* as $\varphi^1_x \equiv \exists x : \varphi \wedge x$. Finally, all of these classes enjoy a property that is essential to our algorithms: they do not introduce variables. For each uco Δ considered and each $x \in \mathcal{V}$ and $\varphi \in \mathbf{B}$, if φ is independent of x, then so is $\Delta(\varphi)$. In Section 5 we discuss a case where this property fails to hold.

We now define the various classes formally and establish some results that are essential in establishing the correctness of the algorithms given in Section 4.

3.1 The classes M and M°

The class $\mathbf{M}$ of *monotone* functions consists of the functions φ satisfying the following requirement: for all valuations μ and μ', $\mu \cup \mu' \models \varphi$ when $\mu \models \varphi$. (These functions are also referred to as *isotone*.) Syntactically the class is most conveniently described as the the class of functions generated by $\{\wedge, \vee, 0, 1\}$, see for example Rudeanu's [21] Theorem 11.3. It follows that the uco $\mathbf{M}$ is additive.

The class $\mathbf{M}^\circ = \{\varphi^\circ \mid \varphi \in \mathbf{M}\}$ consists of the *antitone* Boolean functions. Functions φ in this class have the property that, for all valuations μ and μ', if $\mu \models \varphi$, then $\mu \cap \mu' \models \varphi$. $\mathbf{M}^\circ$, too, is additive.

As ROBDDs are based on the idea of repeated Boolean development, we are particularly interested in characterising class membership for formulae of the forms $x \vee \varphi$ and $\overline{x} \vee \varphi$ (with φ independent of x).

Lemma 1. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$. Then

(a) $x \vee \varphi \in \mathbf{M}$ iff $\varphi \in \mathbf{M}$
(b) $\overline{x} \vee \varphi \in \mathbf{M}$ iff $\varphi \in \mathbf{1}$
(c) $x \vee \varphi \in \mathbf{M}^\circ$ iff $\varphi \in \mathbf{1}$
(d) $\overline{x} \vee \varphi \in \mathbf{M}^\circ$ iff $\varphi \in \mathbf{M}^\circ$

Proof: In all cases, the 'if' direction is obvious from the well-known syntactic characterisations of $\mathbf{M}$ and $\mathbf{M}^\circ$. We show the 'only if' direction for cases (a) and (b); the proofs for (c) and (d) are similar.

(a) Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$, such that $x \vee \varphi \in \mathbf{M}$. Consider a model μ of φ. Since φ is independent of x, we have that $\mu \setminus \{x\} \models x \vee \varphi$. Let μ' be an arbitrary valuation. Then $(\mu \setminus \{x\}) \cup (\mu' \setminus \{x\}) \models x \vee \varphi$, so $(\mu \cup \mu') \setminus \{x\} \models \varphi$. Thus $\mu \cup \mu' \models \varphi$, and since μ' was arbitrary, $\varphi \in \mathbf{M}$.

(b) Suppose $\overline{x} \vee \varphi \in \mathbf{M}$. We show that every valuation is a model of φ. For any valuation μ, $\mu \setminus \{x\} \models \overline{x} \vee \varphi$. But then, $\mu \cup \{x\} \models \overline{x} \vee \varphi$, as $\overline{x} \vee \varphi \in \mathbf{M}$. As φ is independent of x, $\mu \models \varphi$. But μ was arbitrary, so φ must be *1*. ∎

3.2 The classes H and H°

The class $\mathbf{H}$ of propositional Horn functions is exactly the set of model-meet closed Boolean functions. That is, every $\mathbf{H}$ function φ satisfies the following requirement: for all valuations μ and μ', if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \cap \mu' \models \varphi$. Similarly, $\mathbf{H}^\circ$ is the set of model-join closed Boolean functions, satisfying the requirement that for all valuations μ and μ', if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \cup \mu' \models \varphi$.

There are well-known syntactic characterisations of these classes. $\mathbf{H}$ is the set of functions that can be written in conjunctive normal form $\bigwedge(\ell_1 \vee \cdots \vee \ell_n)$ with at most one positive literal ℓ per clause, while $\mathbf{H}^\circ$ functions can be written in conjunctive normal form with each clause containing at most one *negative* literal.[1] It is immediate that $\mathbf{M} \subseteq \mathbf{H}^\circ$ and $\mathbf{M}^\circ \subseteq \mathbf{H}$.

The next lemma characterises membership of $\mathbf{H}$ and $\mathbf{H}^\circ$, for the case $\ell \vee \varphi$.

Lemma 2. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$. Then

(a) $x \vee \varphi \in \mathbf{H}$ iff $\varphi \in \mathbf{M}^\circ$
(b) $\overline{x} \vee \varphi \in \mathbf{H}$ iff $\varphi \in \mathbf{H}$
(c) $x \vee \varphi \in \mathbf{H}^\circ$ iff $\varphi \in \mathbf{H}^\circ$
(d) $\overline{x} \vee \varphi \in \mathbf{H}^\circ$ iff $\varphi \in \mathbf{M}$

[1] An unfortunate variety of nomenclatures is used in Boolean taxonomy. For example, Schaefer [22] uses "weakly negative" for $\mathbf{H}$ and "weakly positive" for $\mathbf{H}^\circ$. Ekin *et al.* [8] use the term "Horn" to refer to $\{\overline{\varphi} \mid \varphi \in \mathbf{H}\}$ and "positive" for $\mathbf{M}$, while we use the word "positive" to refer to another class entirely (see Section 5).

Proof: In all cases, the 'if' direction follows easily from the syntactic characterisations of the classes. We prove the 'only if' directions for (a) and (b), as (c) and (d) are similar.

(a) Assume $x \vee \varphi \in \mathbf{H}$ and x is independent of φ. Let μ be a model for φ and let μ' be an arbitrary valuation. Both $\mu \setminus \{x\}$ and $\mu' \cup \{x\}$ are models for $x \vee \varphi$. As $x \vee \varphi \in \mathbf{H}$, their intersection is a model as well, that is, $(\mu \cap \mu') \setminus \{x\} \models x \vee \varphi$. But then $(\mu \cap \mu') \models x \vee \varphi$, and as μ' was arbitrary, it follows that $\varphi \in \mathbf{M}^\circ$.

(b) Assume $\overline{x} \vee \varphi \in \mathbf{H}$ and x is independent of φ. Consider models μ and μ' for φ. As φ is independent of x, $\mu \setminus \{x\}$ and $\mu' \setminus \{x\}$ are models for $\overline{x} \vee \varphi$, and so $(\mu \setminus \{x\}) \cap (\mu' \setminus \{x\}) \models \overline{x} \vee \varphi$. But then $(\mu \setminus \{x\}) \cap (\mu' \setminus \{x\}) \models \varphi$, hence $\mu \cap \mu' \models \varphi$, so $\varphi \in \mathbf{H}$. ∎

3.3 The class $\mathbf{V}_{\rightarrow}$

We define $\mathbf{V}_{\rightarrow} = \mathbf{H} \cap \mathbf{H}^\circ$. Hence this is the class of Boolean functions φ that are both model-meet and model-join closed. For all valuations μ and μ', $\mu \cap \mu' \models \varphi$ and $\mu \cup \mu' \models \varphi$ when $\mu \models \varphi$ and $\mu' \models \varphi$. Since $\mathbf{H}$ and $\mathbf{H}^\circ$ commute as closure operators, we could equally well have defined $\mathbf{V}_{\rightarrow} = \mathbf{H} \circ \mathbf{H}^\circ$.

Syntactically, $\mathbf{V}_{\rightarrow}$ consists of exactly those Boolean functions that can be written in conjunctive normal form $\bigwedge c$ with each clause c taking one of four forms: 0, x, $\overline{x}$, or $x \rightarrow y$. Note that $\mathbf{V}_{\rightarrow}^\circ = \mathbf{V}_{\rightarrow}$.

Lemma 3. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$. Then

(a) $x \vee \varphi \in \mathbf{V}_{\rightarrow}$ iff $\varphi \in \mathbf{V}^\circ$ (b) $\overline{x} \vee \varphi \in \mathbf{V}_{\rightarrow}$ iff $\varphi \in \mathbf{V}$

Proof: (a) Since $x \vee \varphi \in \mathbf{V}_{\rightarrow}$, we know that $x \vee \varphi \in \mathbf{H}$ and $x \vee \varphi \in \mathbf{H}^\circ$. Then by Lemma 2, $\varphi \in \mathbf{M}^\circ$ and $\varphi \in \mathbf{H}^\circ$. Thus $\varphi \in \mathbf{V}^\circ$. The proof for (b) is similar. ∎

3.4 The classes V, V°, C, and 1

We define $\mathbf{V}$ to be the class of model-meet and upward closed Boolean functions. Syntactically, $\varphi \in \mathbf{V}$ iff $\varphi = 0$ or φ can be written as a (possibly empty) conjunction of positive literals. Dually, $\mathbf{V}^\circ$ is the class of model-join and downward closed Boolean functions — those that can be written as 0 or (possibly empty) conjunctions of *negative* literals. $\mathbf{C}$ is the set of Boolean functions that are both upward and downward closed. This set contains only the constant functions 0 and 1. Finally, $\mathbf{1}$ consists of only the constant function 1. The next lemma is trivial, but included for completeness.

Lemma 4. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$. Then

(a) $x \vee \varphi \in \mathbf{V}$ iff $\varphi \in \mathbf{C}$
(b) $\overline{x} \vee \varphi \in \mathbf{V}$ iff $\varphi \in \mathbf{1}$
(c) $x \vee \varphi \in \mathbf{V}^\circ$ iff $\varphi \in \mathbf{1}$
(d) $\overline{x} \vee \varphi \in \mathbf{V}^\circ$ iff $\varphi \in \mathbf{C}$
(e) $x \vee \varphi \in \mathbf{C}$ iff $\varphi \in \mathbf{1}$
(f) $\overline{x} \vee \varphi \in \mathbf{C}$ iff $\varphi \in \mathbf{1}$
(g) $x \vee \varphi \in \mathbf{1}$ iff $\varphi \in \mathbf{1}$
(h) $\overline{x} \vee \varphi \in \mathbf{1}$ iff $\varphi \in \mathbf{1}$

Proof: The proof is similar to that of Lemma 3. ∎

4 Algorithms for approximating ROBDDs

In this section we show how to find upper approximations within the classes of the previous section. Algorithms for lower approximation can be obtained in a parallel way. We assume input and output given as ROBDDs. In this context, the main obstacle to the development of algorithms is a lack of distributivity and substitutivity properties amongst closure operators. To exemplify the problems in the context of $\mathbf{H}$, given $\varphi = x \leftrightarrow y$ and $\psi = x \vee y$, we have

$$\begin{array}{rcl} \mathbf{H}(\varphi \wedge \psi) = \mathbf{H}(x \wedge y) = x \wedge y & \neq & (x \leftrightarrow y) \wedge 1 = \mathbf{H}(\varphi) \wedge \mathbf{H}(\psi) \\ \mathbf{H}(x \vee y) = 1 & \neq & x \vee y = \mathbf{H}(x) \vee \mathbf{H}(y) \\ (\mathbf{H}(x \vee y))^0_x = 1^0_x = 1 & \neq & y = \mathbf{H}(y) = \mathbf{H}((x \vee y)^0_x) \end{array}$$

Nevertheless, for a large number of commonly used classes, Boolean development gives us a handle to restore a limited form of distributivity. The idea is as follows. Let $\sigma = (x \wedge \varphi) \vee (\overline{x} \wedge \psi)$. We can write σ alternatively as

$$\sigma = (\overline{\psi} \to x) \wedge (x \to \varphi)$$

showing how the "subtrees" φ and ψ communicate with x. As we have seen, we cannot in general find $\rho(\sigma)$, even for "well-behaved" closure operators ρ, by distribution—the following does *not* hold:

$$\rho(\sigma) = \rho(\overline{\psi} \to x) \wedge \rho(x \to \varphi)$$

Suppose however that we add a redundant conjunct to the expression for σ:

$$\sigma = (\overline{\psi} \to x) \wedge (x \to \varphi) \wedge (\varphi \vee \psi)$$

The term $\varphi \vee \psi$ is redundant, as it is nothing but $\exists x(\sigma)$. The point that we utilise here is that, for a large number of natural classes (or upper closure operators) ρ, distribution is allowed in this context, that is,

$$\rho(\sigma) = \rho(\overline{\psi} \to x) \wedge \rho(x \to \varphi) \wedge \rho(\varphi \vee \psi)$$

The intuition is that the information that is lost by $\rho(\overline{\psi} \to x) \wedge \rho(x \to \varphi)$, namely the "$\rho$" information shared by φ and ψ is exactly recovered by $\rho(\varphi \vee \psi)$. Figure 3 shows, for reference, the ROBDD for a function, $(x \to z) \wedge (\overline{y} \to z)$, and the ROBDDs that result from different approximations.

Before we present our approximation algorithms, we need one more lemma.

Lemma 5. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$.

(a) **if** $x \vee \varphi \in \Delta \leftrightarrow \varphi \in \Delta'$ **then** $\Delta(x \vee \varphi) = x \vee \Delta'(\varphi)$
(b) **if** $\overline{x} \vee \varphi \in \Delta \leftrightarrow \varphi \in \Delta'$ **then** $\Delta(\overline{x} \vee \varphi) = \overline{x} \vee \Delta'(\varphi)$

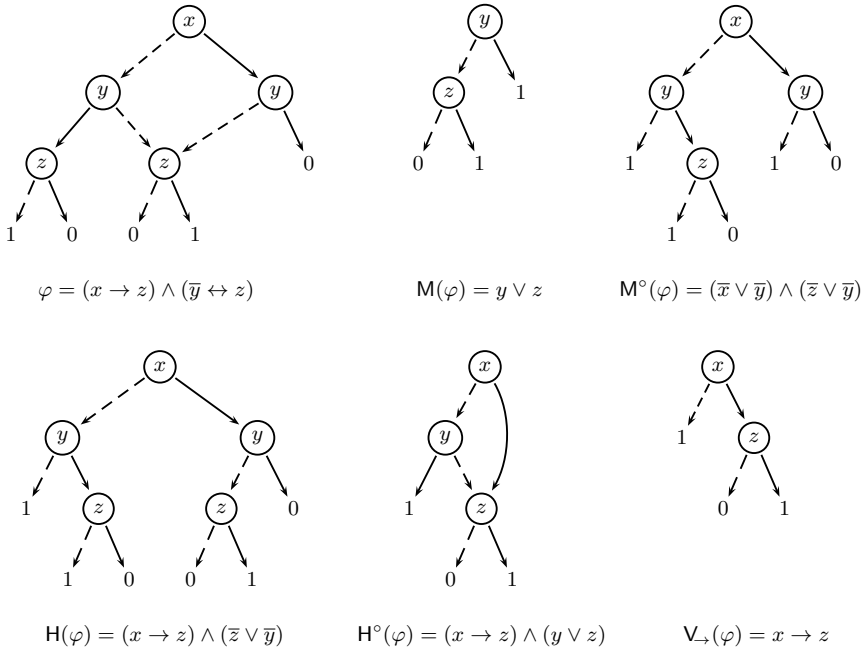

Fig. 3. ROBDDs for $(x \to z) \wedge (\overline{y} \leftrightarrow z)$ and some approximations of it

Proof: We show (a)—the proof for (b) is similar. Assume $x \vee \varphi \in \Delta \leftrightarrow \varphi \in \Delta'$.

$$
\begin{array}{lll}
\Delta(x \vee \varphi) = \bigwedge\{\psi' \in \Delta \mid x \vee \varphi \models \psi'\} & & \\
\quad = \bigwedge\{\psi' \in \Delta \mid x \models \psi' \text{ and } \varphi \models \psi'\} & & \\
\quad = \bigwedge\{x \vee \psi \mid x \vee \psi \in \Delta \text{ and } \varphi \models x \vee \psi\} & & \psi' \text{ is of the form } x \vee \psi \\
\quad = \bigwedge\{x \vee \psi \mid x \vee \psi \in \Delta \text{ and } \varphi \models \psi\} & & \varphi \text{ is independent of } x \\
\quad = \bigwedge\{x \vee \psi \mid \psi \in \Delta' \text{ and } \varphi \models \psi\} & & \text{premise} \\
\quad = x \vee \bigwedge\{\psi \mid \psi \in \Delta' \text{ and } \varphi \models \psi\} & & \\
\quad = x \vee \Delta'(\varphi) & & \blacksquare
\end{array}
$$

4.1 The upper closure operators H and H°

Algorithm 1 To find the strongest **H** consequence of a Boolean function:

$$
\begin{array}{lll}
\mathsf{H}(0) & = 0 & \\
\mathsf{H}(1) & = 1 & \\
\mathsf{H}(\mathsf{ite}(x, R_1, R_2)) & = \mathsf{mknd}(x, R^t, R^f) & \\
& \quad \textbf{where} \;\; R' = \mathsf{H}(\mathsf{or}(R_1, R_2)) & \\
& \quad\;\; \textbf{and} \;\; R^t = \mathsf{H}(R_1) & \\
& \quad\;\; \textbf{and} \;\; R^f = \mathsf{and}(\mathsf{M}^\circ(R_2), R') &
\end{array}
$$

To prove the correctness of this algorithm, we shall need the following lemma:

Lemma 6. Let $\varphi \in \mathbf{B}$ be independent of $x \in \mathcal{V}$. Then

(a) $\mathbf{H}(x \wedge \varphi) = x \wedge \mathbf{H}(\varphi)$ (c) $\mathbf{H}^\circ(x \wedge \varphi) = x \wedge \mathbf{H}^\circ(\varphi)$
(b) $\mathbf{H}(\overline{x} \wedge \varphi) = \overline{x} \wedge \mathbf{H}(\varphi)$ (d) $\mathbf{H}^\circ(\overline{x} \wedge \varphi) = \overline{x} \wedge \mathbf{H}^\circ(\varphi)$ ∎

Proposition 1. For any ROBDD R, $\mathbf{H}[\![R]\!] = [\![\mathsf{H}(R)]\!]$.

Proof: By induction on $\mathsf{vars}(R)$. When $\mathsf{vars}(R) = \varnothing$, R must be either 0 or 1; in these cases the proposition holds.

Assume $\mathsf{vars}(R) \neq \varnothing$ and take $R = \mathsf{ite}(x, R_1, R_2)$. $\mathsf{vars}(R) \supset \mathsf{vars}(\mathsf{or}(R_1, R_2))$, so the induction is well-founded. Let $R' = \mathsf{H}(\mathsf{or}(R_1, R_2))$. By the induction hypothesis, $[\![R']\!] = \mathbf{H}[\![\mathsf{or}(R_1, R_2)]\!] = \mathbf{H}([\![R_1]\!] \vee [\![R_2]\!])$.

We prove first that $\mathbf{H}[\![R]\!] \models [\![\mathsf{H}(R)]\!]$. Note that

$$\begin{array}{lll} x \vee \mathbf{H}(\psi) & \models \mathbf{H}(x \vee \mathbf{H}(\psi)) & = \mathbf{H}(x \vee \psi) \\ \overline{x} \vee \mathbf{H}(\varphi) & \models \mathbf{H}(\overline{x} \vee \mathbf{H}(\varphi)) & = \mathbf{H}(\overline{x} \vee \varphi) \\ \mathbf{H}(\varphi) \vee \mathbf{H}(\psi) & \models \mathbf{H}(\mathbf{H}(\varphi) \vee \mathbf{H}(\psi)) & = \mathbf{H}(\varphi \vee \psi) \end{array}$$

Since $\mathbf{H}$ and $\wedge$ are monotone, $\mathbf{H}[(x \vee \mathbf{H}(\psi)) \wedge (\overline{x} \vee \mathbf{H}(\varphi)) \wedge (\mathbf{H}(\varphi) \vee \mathbf{H}(\psi))] \models \mathbf{H}[\mathbf{H}(x \vee \psi) \wedge \mathbf{H}(\overline{x} \vee \varphi) \wedge \mathbf{H}(\varphi \vee \psi)]$. Hence, by (1),

$$\begin{array}{l} \mathbf{H}[(x \vee \mathbf{H}(\psi)) \wedge (\overline{x} \vee \mathbf{H}(\varphi)) \wedge (\mathbf{H}(\varphi) \vee \mathbf{H}(\psi))] \\ \quad \models \mathbf{H}(x \vee \psi) \wedge \mathbf{H}(\overline{x} \vee \varphi) \wedge \mathbf{H}(\varphi \vee \psi) \end{array} \tag{3}$$

Now we have

$$\begin{array}{lll} & \mathbf{H}[\![R]\!] & \\ & = \mathbf{H}[(x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!])] & \\ & = \mathbf{H}(\mathbf{H}(x \wedge [\![R_1]\!]) \vee \mathbf{H}(\overline{x} \wedge [\![R_2]\!])) & \text{uco property} \\ & = \mathbf{H}[(x \wedge \mathbf{H}[\![R_1]\!]) \vee (\overline{x} \wedge \mathbf{H}[\![R_2]\!])] & \text{Lemma 6} \\ & = \mathbf{H}[(x \vee \mathbf{H}[\![R_2]\!]) \wedge (\overline{x} \vee \mathbf{H}[\![R_1]\!])] & \text{distribution} \\ & = \mathbf{H}[(x \vee \mathbf{H}[\![R_2]\!]) \wedge (\overline{x} \vee \mathbf{H}[\![R_1]\!]) \wedge (\mathbf{H}[\![R_1]\!] \vee \mathbf{H}[\![R_2]\!])] & \\ & \models \mathbf{H}(x \vee [\![R_2]\!]) \wedge \mathbf{H}(\overline{x} \vee [\![R_1]\!])) \wedge \mathbf{H}([\![R_1]\!] \vee [\![R_2]\!]) & \text{Equation 3} \\ & = (x \vee \mathbf{M}^\circ[\![R_2]\!]) \wedge (\overline{x} \vee \mathbf{H}[\![R_1]\!]) \wedge [\![R']\!] & \text{Lemmas 2 and 5} \\ & = (x \wedge \mathbf{H}[\![R_1]\!] \wedge [\![R']\!]) \vee (\overline{x} \wedge \mathbf{M}^\circ[\![R_2]\!] \wedge [\![R']\!]) & \text{distribution} \\ & = (x \wedge \mathbf{H}[\![R_1]\!]) \vee (\overline{x} \wedge \mathbf{M}^\circ[\![R_2]\!] \wedge [\![R']\!]) & \mathbf{H} \text{ is monotone} \\ & = (x \wedge [\![\mathsf{H}(R_1)]\!]) \vee (\overline{x} \wedge [\![\mathsf{M}^\circ(R_2)]\!] \wedge [\![R']\!]) & \text{Ind. hyp., Prop 4} \\ & = [\![\mathsf{mknd}(x, \mathsf{H}(R_1), \mathsf{and}(\mathsf{M}^\circ(R_2), R'))]\!] & \\ & = [\![\mathsf{H}(R)]\!] & \end{array}$$

Next we show $[\![\mathsf{H}(R)]\!] \models \mathbf{H}[\![R]\!]$. From the development above, it is clear that this amounts to showing that

$$(x \wedge \mathbf{H}[\![R_1]\!] \wedge [\![R']\!]) \vee (\overline{x} \wedge \mathbf{M}^\circ[\![R_2]\!] \wedge [\![R']\!]) \models \mathbf{H}[(x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!])]$$

By Lemma 6, $x \wedge \mathbf{H}[\![R_1]\!] = \mathbf{H}(x \wedge [\![R_1]\!])$. So clearly $x \wedge \mathbf{H}[\![R_1]\!] \wedge [\![R']\!] \models \mathbf{H}(x \wedge [\![R_1]\!]) \vee \mathbf{H}(\overline{x} \wedge [\![R_2]\!])$, so $x \wedge \mathbf{H}[\![R_1]\!] \wedge [\![R']\!] \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. It remains to prove that $\overline{x} \wedge \mathbf{M}^\circ[\![R_2]\!] \wedge [\![R']\!] \models \mathbf{H}[(x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!])]$. If the left-hand side

is false, then the claim holds trivially. So let μ be a model of $\overline{x} \wedge \mathbf{M}^\circ[\![R_2]\!] \wedge [\![R']\!]$. Thus $\mu \models \overline{x}$, $\mu \models \mathbf{M}^\circ[\![R_2]\!]$, and $\mu \models [\![R']\!]$, and we must show μ entails the right-hand side. Let us consider three exhaustive cases.

First assume $\mu \models \mathbf{H}[\![R_2]\!]$. Then since $\mu \models \overline{x}$, and by Lemma 6, $\mu \models \mathbf{H}(\overline{x} \wedge [\![R_2]\!])$, so certainly $\mu \models \mathbf{H}(x \wedge [\![R_1]\!]) \vee \mathbf{H}(\overline{x} \wedge [\![R_2]\!])$. Then μ must entail the weaker $\mathbf{H}(\mathbf{H}(x \wedge [\![R_1]\!]) \vee \mathbf{H}(\overline{x} \wedge [\![R_2]\!]))$, which by uco properties is equivalent to $\mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$.

Next assume $\mu \models \mathbf{H}[\![R_1]\!]$ and $\mu \not\models \mathbf{H}[\![R_2]\!]$. Since $\mu \models \mathbf{M}^\circ[\![R_2]\!]$, we know that there is some μ' such that $\mu' \models [\![R_2]\!]$, and that $\mu \subseteq \mu'$. Then $\mu' \setminus \{x\} \models \overline{x} \wedge \mathbf{H}[\![R_2]\!] \vee (x \wedge [\![R_1]\!])$, so $\mu' \setminus \{x\}$ must entail the weaker $\mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. We have also assumed $\mu \models \mathbf{H}[\![R_1]\!]$, so by similar argument $\mu \cup \{x\} \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. Then $(\mu \cup \{x\}) \cap (\mu' \setminus \{x\}) \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. But since $\mu \subseteq \mu'$, and since $x \notin \mu$, $(\mu \cup \{x\}) \cap (\mu' \setminus \{x\}) = \mu$, and therefore $\mu \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$.

Finally, assume $\mu \not\models \mathbf{H}[\![R_1]\!]$ and $\mu \not\models \mathbf{H}[\![R_2]\!]$. Since $\mu \models [\![R']\!]$, μ must be the intersection models of $\mathbf{H}[\![R_1]\!]$ and $\mathbf{H}[\![R_2]\!]$. So let μ^+ and μ^- be interpretations such that $\mu^+ \models \mathbf{H}[\![R_1]\!]$ and $\mu^- \models \mathbf{H}[\![R_2]\!]$ and $\mu = \mu^+ \cap \mu^-$. Then, similar to the previous case, $(\mu^+ \cup \{x\}) \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$ and $(\mu^- \setminus \{x\}) \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. But since $\mu = \mu^+ \cap \mu^-$, we know $(\mu^+ \cup \{x\}) \cap (\mu^- \setminus \{x\}) = \mu$, and therefore $\mu \models \mathbf{H}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!]))$. ∎

Algorithm 2 To find the strongest $\mathbf{H}^\circ$ consequence of a Boolean function:

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

Proposition 2. For any ROBDD R, $\mathbf{H}^\circ[\![R]\!] = [\![\mathsf{H}^\circ(R)]\!]$.

Proof: Similar to Proposition 1. ∎

4.2 The upper closure operators M and M°

The algorithms and proofs for $\mathbf{M}$ and $\mathbf{M}^\circ$ are simpler, because these closure operators are additive.

Algorithm 3 To find the strongest $\mathbf{M}$ consequence of a Boolean function:

$$
\begin{aligned}
&\mathsf{M}(0) && = 0\\
&\mathsf{M}(1) && = 1\\
&\mathsf{M}(\mathsf{ite}(x, R_1, R_2)) && = \mathsf{mknd}(x, \mathsf{or}(R_1', R_2'), R_2')\\
&&& \quad \textbf{where}\ \ R_1' = \mathsf{M}(R_1)\\
&&& \quad\ \ \textbf{and}\ \ R_2' = \mathsf{M}(R_2)
\end{aligned}
$$

Proposition 3. For any ROBDD R, $\mathbf{M}[\![R]\!] = [\![\mathsf{M}(R)]\!]$.

Proof: By structural induction. For $R = 0$ and $R = 1$ the proposition clearly holds. Consider $R = \mathsf{ite}(x, R_1, R_2)$ and let $R'_1 = \mathsf{M}(R_1)$ and $R'_2 = \mathsf{M}(R_2)$.

$$\begin{aligned}
\mathbf{M}[\![R]\!] &= \mathbf{M}((x \wedge [\![R_1]\!]) \vee (\overline{x} \wedge [\![R_2]\!])) \\
&= (\mathbf{M}(x) \wedge \mathbf{M}[\![R_1]\!]) \vee (\mathbf{M}(\overline{x}) \wedge \mathbf{M}[\![R_2]\!]) && \mathbf{M} \text{ is additive} \\
&= (x \wedge \mathbf{M}[\![R_1]\!]) \vee \mathbf{M}[\![R_2]\!] \\
&= (x \wedge [\![R'_1]\!]) \vee [\![R'_2]\!] && \text{induction hypothesis} \\
&= (x \wedge ([\![R'_1]\!] \vee [\![R'_2]\!])) \vee (\overline{x} \wedge [\![R'_2]\!]) && \text{development around } x \\
&= (x \wedge [\![\mathsf{or}(R'_1, R'_2)]\!]) \vee (\overline{x} \wedge [\![R'_2]\!]) \\
&= [\![\mathsf{mknd}(x, \mathsf{or}(R'_1, R'_2), R'_2)]\!] \\
&= [\![\mathsf{M}(R)]\!]
\end{aligned}$$

∎

Algorithm 4 To find the strongest $\mathbf{M}^\circ$ consequence of a Boolean function:

$$\begin{aligned}
&\mathsf{M}^\circ(0) && = 0 \\
&\mathsf{M}^\circ(1) && = 1 \\
&\mathsf{M}^\circ(\mathsf{ite}(x, R_1, R_2)) && = \mathsf{mknd}(x, R'_1, \mathsf{or}(R'_1, R'_2)) \\
& && \quad \textbf{where} \;\; R'_1 = \mathsf{M}^\circ(R_1) \\
& && \quad \;\; \textbf{and} \;\; R'_2 = \mathsf{M}^\circ(R_2)
\end{aligned}$$

Proposition 4. For any ROBDD R, $\mathbf{M}^\circ[\![R]\!] = [\![\mathsf{M}^\circ(R)]\!]$.

Proof: Similar to Proposition 3. ∎

4.3 The upper closure operator $\mathbf{V}_\rightarrow$

Algorithm 5 To find the strongest $\mathbf{V}_\rightarrow$ consequence of a Boolean function:

$$\begin{aligned}
&\mathsf{V}_\rightarrow(0) && = 0 \\
&\mathsf{V}_\rightarrow(1) && = 1 \\
&\mathsf{V}_\rightarrow(\mathsf{ite}(x, R_1, R_2)) && = \mathsf{mknd}(x, \mathsf{and}(\mathsf{V}(R_1), R'), \mathsf{and}(\mathsf{V}^\circ(R_2), R')) \\
& && \quad \textbf{where} \;\; R' = \mathsf{V}_\rightarrow(\mathsf{or}(R_1, R_2))
\end{aligned}$$

Proposition 5. For any ROBDD R, $\mathbf{V}_\rightarrow[\![R]\!] = [\![\mathsf{V}_\rightarrow(R)]\!]$.

Proof: This follows from the fact that $\mathbf{V}_\rightarrow = \mathbf{H} \cap \mathbf{H}^\circ$. We omit the details. ∎

4.4 The upper closure operators C, V, and $\mathbf{V}^\circ$

The remaining algorithms are given here for completeness. Their correctness proofs are straightforward.

Algorithm 6 To find the strongest $\mathbf{V}$ consequence of a Boolean function:

$$\mathsf{V}(0) = 0 \qquad \mathsf{V}(1) = 1 \qquad \mathsf{V}(\mathsf{ite}(x, R_1, R_2)) = \mathsf{mknd}(x, R', \mathsf{and}(\mathsf{C}(R_2), R')) \;\; \textbf{where} \;\; R' = \mathsf{V}(\mathsf{or}(R_1, R_2))$$

Algorithm 7 To find the strongest $\mathbf{V}^\circ$ consequence of a Boolean function:

$$\mathsf{V}^\circ(0) = 0 \qquad \mathsf{V}^\circ(1) = 1 \qquad \mathsf{V}^\circ(\mathsf{ite}(x, R_1, R_2)) = \mathsf{mknd}(x, \mathsf{and}(\mathsf{C}(R_1), R'), R') \;\; \textbf{where} \;\; R' = \mathsf{V}^\circ(\mathsf{or}(R_1, R_2))$$

Algorithm 8 To find the strongest $\mathbf{C}$ consequence of a Boolean function:

$$\mathsf{C}(0) = 0 \qquad \mathsf{C}(1) = 1 \qquad \mathsf{C}(\mathsf{ite}(x, R_1, R_2)) = 1$$

5 Discussion and related work

The classes we have covered are but a few examples of the generality of our approach. Many other classes fall under the same general scheme as the algorithms in Section 4. One such is **L**. Syntactically, $\varphi \in \mathbf{L}$ iff $\varphi = 0$ or φ can be written as a (possibly empty) conjunction of literals. Slightly more general is the class **Bij** of *bijunctive* functions. Members of this class can be written in clausal form with at most two literals per clause.

A class central to many analyses of logic programs is that of *positive* functions [16, 17]. Let $\mu_\top$ be the unit valuation, that is, $\mu_\top = 1$ for all $x \in \mathcal{V}$. Then φ is positive iff $\mu_\top \models \varphi$. We denote the class of positive functions by **Pos**. This class is interesting in the context of ROBDDs, as it is a class which is easily recognisable but problematic to find approximations in. To decide whether an ROBDD represents a positive function, simply follow the solid-arc path from the root to a sink—the function is positive if and only if the sink is *1*. Approximation, however, can not be done in general without knowledge of the entire space of variables, and not all variables necessarily appear in the ROBDD. For example, if the set of variables is $\mathcal{V}$, then $\mathbf{Pos}(\overline{x_i}) = x_i \to \bigwedge \mathcal{V}$, which depends on every variable in $\mathcal{V}$. We should note, however, that this does not mean that our approximation algorithms are useless for sub-classes of **Pos**. On the contrary, they work seamlessly for the positive sub-classes commonly used in program analysis, discussed below, as long as positive functions are being approximated (which is invariably the case).

The classes we have discussed above are not sub-classes of **Pos**. (In both **M** and **V**, however, the only non-positive element is *0*.) Restricting the classes to their positive counterparts, we obtain classes that all have found use in program analysis. Figure 4 shows the correspondence. The classes on the right are obtained by intersecting those on the left with **Pos**. We mention just a few example uses. In the context of groundness analysis for constraint logic programs, **Pos** and **Def** are discussed by Armstrong *et al.* [1]. **Def** is used for example by Howe and King [15]. **2IMP** is found in the exception analysis of Glynn *et al.* [12]. We have also omitted characterizations and algorithms for $\mathbf{V}_{\leftrightarrow}$, the class of functions that can be written as conjunctions of literals and biimplications of the form $x \leftrightarrow y$ with $x, y \in \mathcal{V}$. This class corresponds to the set of all possible partitionings of $\mathcal{V} \cup \{0, 1\}$. Its restriction to the positive fragment is exactly Heaton *et al.*'s "*EPos*" domain [14].

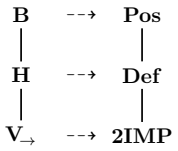

Fig. 4. Positive fragments

M is a class which is of considerable interest in many contexts. In program analysis it has a classical role: Mycroft's well-known two-valued strictness analysis for first-order functional programs [18] uses **M** to capture non-termination information.

The classes we have considered are of much theoretical interest. The classes **Bij**, **H**, $\mathbf{H}^\circ$, **Pos** and $\mathbf{Pos}^\circ$ are five of the six classes from Schaefer's dichotomy result [22] (the sixth is the class of affine Boolean functions). **M** plays a role in

Post's functional completeness result [20], together with the affine functions, **Pos** and its dual, and the class of self-dual functions. Giacobazzi and Scozzari provide interesting characterisations of domains including **Pos** in terms of domain completion using natural domain operations [11].

The problem of *approximating* Boolean functions appears in many contexts in program analysis. We already mentioned Genaim and King's suspension analysis [9] and the formulation of set-sharing using **Pos**, by Codish *et al.* [5]. Another possible application is in the design of widening operators for abstract interpretation-based analyses.

6 Conclusion

We have provided algorithms to find upper approximations for Boolean functions represented as ROBDDs. The algorithms all follow the same general pattern, which works for a large number of important classes of Boolean functions. They also provide a way of checking an ROBDD R for membership of a given class Δ: Simply check whether $R = \Delta(R)$.

In the design of our algorithms we have emphasised clarity rather than efficiency. We note that the critical term $\Delta(\varphi \vee \psi)$ is identical to the join $\Delta(\varphi) \sqcup_\Delta \Delta(\psi)$, so in many cases, efficient approximation algorithms may boil down to efficient computation of the join. Future research will include a search for appropriate data structures and associated complexity analyses, as well as attempts at a more general and abstract approach to the algorithms and proofs.

References

1. 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.
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. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, C–35(8):677–691, 1986.
4. R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
5. 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.
6. 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.
7. 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.
8. O. Ekin, S. Foldes, P. L. Hammer, and L. Hellerstein. Equational characterizations of Boolean function classes. *Discrete Mathematics*, 211:27–51, 2000.

9. 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*, pages 84–98. Springer, 2003.
10. R. Giacobazzi. *Semantic Aspects of Logic Program Analysis*. PhD thesis, University of Pisa, Italy, 1993.
11. R. Giacobazzi and F. Scozzari. A logical model for relational abstract domains. *ACM Trans. Programming Languages and Systems*, 20(5):1067–1109, 1998.
12. 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*, pages 98–109. ACM Press, 2002.
13. P. R. Halmos. *Lectures on Boolean Algebras*. Springer-Verlag, 1963.
14. 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.
15. J. M. Howe and A. King. Efficient groundness analysis in Prolog. *Theory and Practice of Logic Programming*, 3(1):95–124, 2003.
16. 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*, pages 152–188. Springer, 2004.
17. 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.
18. A. Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs*. PhD thesis, University of Edinburgh, Scotland, 1981.
19. O. Ore. Combinations of closure relations. *Ann. Math.*, 44(3):514–533, 1943.
20. 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.
21. S. Rudeanu. *Boolean Functions and Equations*. North-Holland, 1974.
22. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. Tenth Ann. ACM Symp. Theory of Computing*, pages 216–226, 1978.
23. M. Ward. The closure operators of a lattice. *Ann. Math.*, 43(2):191–196, 1942.