Logic decomposition of incompletely specified functions

Jordi Cortadella*
Department of Software
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
jordic@lsi.upc.es

Abstract

The logic decomposition of incompletely specified functions (ISFs) is studied. A theoretical support based on ternary algebras is provided. This support aims at the logic decomposition of ISFs. A partial order is defined on the ISFs determined by the "definedness" of the functions, in which the completely specified functions are the maximal elements. The proposed methods aim at preserving the ISFs as undefined as possible during logic decomposition. Incompletely specified diagrams (IDDs) are proposed as a graph-based formalism to efficiently explore different decompositions.

1 Introduction

Logic synthesis of Boolean functions aims at finding implementations that minimize some target cost. In many situations, the functions are incompletely specified due to the don't care information extracted from their environment. Don't care sets are crucial in Boolean minimization to reduce the cost of the implementations.

However, it is often the case that Boolean minimization cannot be performed on large circuits. Decomposition techniques must be used to manage the complexity of such task. Algebraic methods have been often used for decomposition, but the quality of the result is highly dependent on the initial circuit and don't care information cannot be used effectively.

Other methods closer to the Boolean nature of the circuits have also been used for decomposition, such as Boolean relations [CF93] or SPDFs [YSH00]. Even though the quality of the results is superior to that of algebraic methods, the high computational cost of such methods make their use limited.

In this work, a framework for Boolean decomposition based on incompletely specified functions is studied. The framework sits in between algebraic methods and Boolean methods. On one hand, the main aim is the decomposition into well-known structures: AND/OR, XOR and MUX decompositions. On the other hand, the Boolean properties of the functions and the don't care information are exploited and mantained during decomposition, in such a way that the assignment of don't care values to **0** or **1** is delayed as much as possible.

Incompletely specified diagrams (IDDs) are prosposed as a graph-based formalism to efficiently manipulate and decompose Boolean functions.

This work has been inspired by the methods proposed in [YSC99, YCS00] for the decomposition of completely specified functions.

*This work has been partially funded by a grant from Intel Corporation, CICYT TIC 98-0410, CICYT TIC 98-0949 and GRQ 1999SGR-150.

2 Definitions

2.1 Ternary logic

This section presents some basic concepts on ternary logic and incompletely specified functions (ISFs). ISFs are represented as 3-valued functions, where the value $\perp$ represents the undefinedness of the function. In the scope of Boolean minimization, $\perp$ corresponds to the don't care value. A crucial task in minimization is to assign $\mathbf{0}$ or $\mathbf{1}$ values to $\perp$ in such a way that a low-cost completely specified function is obtained.

Definition 2.1 (Boolean domain)
The Boolean domain *is defined as the set* $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$. *The* ternary domain *is defined as* $\mathbb{B}^\star = \mathbb{B} \cup \{\perp\}$.

Definition 2.2 (Operations on $\mathbb{B}^\star$)
The operations complement, disjunction *and* conjunction *are defined on* $\mathbb{B}^\star$ *according to the following tables:*

x	$\overline{x}$
$\mathbf{0}$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$
$\perp$	$\perp$

$+$	$\mathbf{0}$	$\mathbf{1}$	$\perp$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	$\perp$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$
$\perp$	$\perp$	$\mathbf{1}$	$\perp$

$\cdot$	$\mathbf{0}$	$\mathbf{1}$	$\perp$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	$\perp$
$\perp$	$\mathbf{0}$	$\perp$	$\perp$

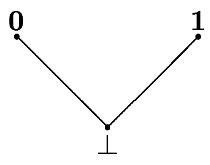
Other operations tipically used in Boolean algebra can also be defined on $\mathbb{B}^\star$ in terms of the previous operations, e.g.

$$\begin{aligned} x \oplus y &= x\overline{y} + \overline{x}y \\ x \Longrightarrow y &= \overline{x} + y \end{aligned}$$

Note that the previous operations coincide with the corresponding Boolean operations on the domain $\mathbb{B}$. Kleene introduced the previous operations in his *strong ternary logic* [Kle52], where the value 0.5 was used to represent ambiguity or undefinedness. Complement, conjunction and disjunction were performed by the operations $1 - x$, min and max, respectively. Multiple-valued Kleenean functions taking values in the unit interval $[0, 1]$ have also been used to represent fuzzy functions [TKNM98].

Property 2.3 (Ternary algebra)
The system $T = \langle \mathbb{B}^\star, +, \cdot, \bar{\ }, \mathbf{0}, \mathbf{1}, \perp \rangle$ *is a ternary algebra [Muk83].*

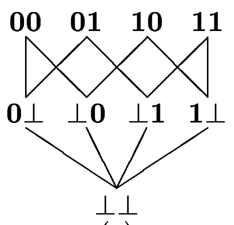
Property 2.4 (Properties in ternary algebra)
The following properties hold in ternary algebra for any a, b and c in $\mathbb{B}^\star$*:*



(a)

x	$f(x)$								
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\perp$	$\perp$	$\perp$
$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	$\perp$	$\mathbf{0}$	$\mathbf{1}$	$\perp$	$\mathbf{0}$	$\mathbf{1}$	$\perp$

(b)



(c)

Figure 1: (a) semilattice for truth values, (b) 1-variable functions, (c) semilattice for 1-variable functions (the pairs $\langle f(\mathbf{0}) f(\mathbf{1})\rangle$ are depicted in the Hasse diagram).

- $a+b=b+a; \quad ab=ba$ *(commutativity)*
- $(a+b)+c=a+(b+c); \quad (ab)c=a(bc)$ *(associativity)*
- $a(b+c)=ab+ac; \quad a+(bc)=(a+b)(a+c)$ *(distributivity)*
- $\mathbf{0}+a=a; \quad \mathbf{1}\cdot a=a$ *(identities)*
- $a+a=a; \quad aa=a$ *(idempotence)*
- $a+\mathbf{1}=\mathbf{1}; \quad a\cdot\mathbf{0}=\mathbf{0}$
- $a+ab=a; \quad a(a+b)=a$ *(absorption)*
- $\overline{\overline{a}}=a$ *(involution)*
- $\overline{a+b}=\overline{a}\,\overline{b}; \quad \overline{ab}=\overline{a}+\overline{b}$ *(De Morgan's Laws)*
- $a+\overline{a}b=a+b; \quad a(\overline{a}+b)=ab$
- $a+\overline{a}+\perp=a+\overline{a}; \quad a\,\overline{a}\perp=a\,\overline{a}$

Property 2.5 *The system $T=\langle \mathbb{B}^\star,+,\cdot,^-,\mathbf{0},\mathbf{1},\perp\rangle$ is not a Boolean algebra, since the postulate for the complement does not hold:*

$$a=\perp \quad\Longrightarrow\quad (a+\overline{a}\neq\mathbf{1} \quad\wedge\quad a\cdot\overline{a}\neq\mathbf{0})$$

Definition 2.6 ($\sqcup$-semilattice on $\mathbb{B}^\star$)
The following binary relation is defined on $\mathbb{B}^\star\times\mathbb{B}^\star$ (see Figure 1(a)):

$$\sqsubseteq \;=\; \{(\mathbf{0},\mathbf{0}),(\mathbf{1},\mathbf{1}),(\perp,\perp),(\mathbf{0},\perp),(\mathbf{1},\perp)\}$$

$\sqsubseteq$ is reflexive, antisymmetric and transitive, and every pair of elements in $\mathbb{B}^\star$ has a lower bound. Therefore, $(\mathbb{B}^\star,\sqsubseteq)$ is a $\sqcup$-semilattice.

Property 2.7 (Monotonicity of operations)
The operations in Definition 2.2 are monotone, i.e.

$$\begin{aligned}
x\sqsubseteq y \quad&\Longrightarrow\quad \overline{x}\sqsubseteq\overline{y}\\
x_1\sqsubseteq y_1 \;\wedge\; x_2\sqsubseteq y_2 \quad&\Longrightarrow\quad (x_1\cdot x_2)\sqsubseteq(y_1\cdot y_2)\\
x_1\sqsubseteq y_1 \;\wedge\; x_2\sqsubseteq y_2 \quad&\Longrightarrow\quad (x_1+x_2)\sqsubseteq(y_1+y_2)
\end{aligned}$$

2.2 Incompletely specified functions

Definition 2.8 (Incompletely specified function (ISF))
An n-variable ISF is a function $f : \mathbb{B}^n \to \mathbb{B}^\star$. The set of n-variable ISFs is called $\mathcal{F}_n^\star$. The constant functions $f(x) = \mathbf{0}$, $f(x) = \mathbf{1}$ and $f(x) = \bot$, for any $x \in \mathbb{B}^n$, will be denoted by $\mathbf{0}$, $\mathbf{1}$ and $\bot$, respectively.

Definition 2.9 (Completely specified function)
A function $f \in \mathcal{F}_n^\star$ is said to be completely specified if $f(x) \neq \bot$ for any $x \in \mathbb{B}^n$. The set of n-variable completely specified functions is called $\mathcal{F}_n$.

Definition 2.10 ($\sqcup$-semilattice on $\mathcal{F}_n^\star$)
The following binary relation is defined on $\mathcal{F}_n^\star \times \mathcal{F}_n^\star$:

$$f \sqsubseteq g \iff \forall x \in \mathbb{B}^n, f(x) \sqsubseteq g(x)$$

$\sqsubseteq$ is reflexive, antisymmetric and transitive, and every pair of elements in $\mathbb{B}^\star$ has a lower bound. Therefore, $(\mathcal{F}_n^\star, \sqsubseteq)$ is a $\sqcup$-semilattice.

Property 2.11 *The following properties hold in $(\mathcal{F}_n^\star, \sqsubseteq)$:*

- *$\bot$ is the minimal element.*
- *$\mathcal{F}_n$ is the set of maximal elements.*
- *f and g have an upper bound $\iff \forall x \in \mathbb{B}^n \ (f(x) \sqsubseteq g(x) \ \vee \ g(x) \sqsubseteq f(x))$*

Figures 1(b) and 1(c) show all 1-variable ISFs with the partial order induced by their definedness.

Property 2.12 (Meet and join operators in $(\mathcal{F}_n^\star, \sqsubseteq)$)
The meet and join operators give the least upper bound and greatest lower bound of two elements of a lattice, respectively. Given f and g in $\mathcal{F}_n^\star$, the meet ($\sqcap$) and join ($\sqcup$) operators in $(\mathcal{F}_n^\star, \sqsubseteq)$ can de defined in terms of the same operators in $(\mathbb{B}^\star, \sqsubseteq)$ as follows:

$$\begin{aligned}(f \sqcap g)(x) &= f(x) \sqcap g(x) \\ (f \sqcup g)(x) &= f(x) \sqcup g(x)\end{aligned}$$

Since $(\mathbb{B}^\star, \sqsubseteq)$ and $(\mathcal{F}_n^\star, \sqsubseteq)$ are not $\sqcap$-semilattices, the meet of two elements is not always defined. For convenience, the fact that no meet exists for x and y will be denoted by $x \sqcap y = \top$.

Definition 2.13 (Realization and complete realization)
Given two functions f and g, f is said to be a realization of g if $g \sqsubseteq f$ (also denoted by $f \sqsupseteq g$). f is said to be a complete realization of g if it is a realization of g and a completely specified function.

Definition 2.14 (Operations on $\mathcal{F}_n^\star$)
The Boolean operations in Definition 2.2 are extended to $\mathcal{F}_n^\star$ as follows:

$$\begin{aligned}\overline{f}(x) &= \overline{f(x)} \\ (f + g)(x) &= f(x) + g(x) \\ (f \cdot g)(x) &= f(x) \cdot g(x)\end{aligned}$$

Definition 2.15 (Cofactor)
Given an ISF $f(x_1, \dots, x_n)$, *the* positive cofactor *of f with respect to x_i, denoted by f_{x_i} is defined by:*

$$f_{x_i} = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

The negative cofactor *of f with respect to x_i, denoted by $f_{\overline{x}_i}$, is defined by:*

$$f_{\overline{x}_i} = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Theorem 2.16 (Boole's expansion for ISFs)
If f is an ISF *then*

$$f = x_i \cdot f_{x_i} + \overline{x}_i \cdot f_{\overline{x}_i}$$

Proof: The proof is similar to the one for Boolean functions (see [Bro90]). □

2.3 Logic decomposition

This section presents some results that support the approaches for decomposition proposed in Sections 5 and 6. The letters f, g and h are used to denote functions in $\mathcal{F}_n^\star$.

Theorem 2.17 (Conjunctive decomposition)
Let $f = f_1 f_2 + f_3 f_4$, $f_1 \sqsubseteq g$ and $f_3 \sqsubseteq g$. Then $f \sqsubseteq g(f_2 + f_4)$.

Proof: By property 2.7, $f \sqsubseteq g f_2 + g f_4$, and the theorem immediately follows by applying the distributive law. □

Corollary 2.18 (Disjunctive decomposition)
Let $f = (f_1 + f_2)(f_3 + f_4)$, $f_1 \sqsubseteq g$ and $f_3 \sqsubseteq g$. Then $f \sqsubseteq g + f_2 f_4$.

Proof: The corollary holds by the duality between $+$ and $\cdot$. □

Theorem 2.19 (XOR decomposition)
Let $f = f_1 f_2 + f_3 f_4$, $f_1 \sqsubseteq g$, $f_3 \sqsubseteq \overline{g}$, $f_2 \sqsubseteq \overline{h}$ and $f_4 \sqsubseteq h$. Then $f \sqsubseteq g \oplus h$.

Proof: We have that $f \sqsubseteq g\overline{h} + \overline{g}h = g \oplus h$. □

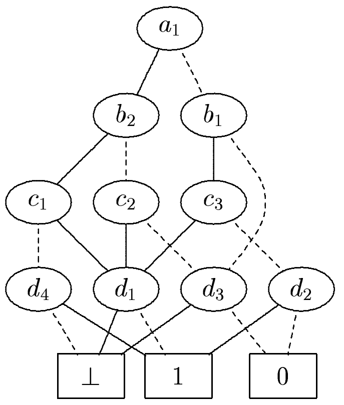
3 Representation of ISFs with IDDs

An ISF can be represented by a *ternary-valued BDD* [MF89]. In this work, we will call them IDDs (incompletely specified decision diagrams). An IDD is a 3-terminal decision diagram[1], each terminal corresponding to an element of $\mathbb{B}^\star$. We will often used the following notation to represent ISFs:

$$f = \mathbf{0} \cdot f^{\mathbf{0}} + \mathbf{1} \cdot f^{\mathbf{1}} + \perp \cdot f^{\perp} = f^{\mathbf{1}} + \perp \cdot f^{\perp}$$

where f^0, f^1 and $f^\perp$ are the characteristic functions of the of the assignements that give the value $\mathbf{0}$, $\mathbf{1}$ and $\perp$, respectively. Figure 2 depicts the representation of an ISF. Node labels denote variables, whereas the indices distinguish nodes with the same variable. Solid and dashed arcs denote T

[1] IDDs are different from *ternary decision diagrams* [Sas96], that have three children for each non-terminal node.



		ab			
		00	01	11	10
	00	**0**	**0**	$\perp$	**0**
cd	01	$\perp$	**1**	**1**	$\perp$
	11	$\perp$	$\perp$	$\perp$	$\perp$
	10	**0**	**1**	**1**	**1**

Figure 2: IDD and the corresponding Karnaugh map.

and E arcs, respectively. Henceforth, we will assume that all IDDs are reduced and ordered, as commonly assumed in most of the works that deal with decision diagrams [Bry92]. The function is characterized by:

$$\begin{aligned} f^0 &= (\overline{a}+\overline{b})\overline{c}\,\overline{d}+\overline{a}\,\overline{b}\,\overline{d} \\ f^1 &= (a+b)c\overline{d}+b\overline{c}d \\ f^\perp &= (c+\overline{b})d+ab\overline{c}\,\overline{d} \end{aligned}$$

Definition 3.1 (Paths)
Given and IDD, a path *is a set of literals that identify a path from the root node to one of the terminals. The set of all paths of an IDD is denoted by* Π. **0**-, **1**- *and* $\perp$*-paths are paths that go to the terminal nodes* **0**, **1** *and* $\perp$*, respectively. The sets of* **0**-, **1**- *and* $\perp$*-paths are denoted by* $\Pi_\mathbf{0}$, $\Pi_\mathbf{1}$ *and* $\Pi_\perp$*, respectively. Therefore,* $\Pi = \Pi_\mathbf{0} \cup \Pi_\mathbf{1} \cup \Pi_\perp$. *For convenience, a path will be often interpreted as a conjunction of literals. An empty path (with no literals) will be interpreted as the function* **1**.

In the example of Figure 2, $a\overline{b}cd$ and $\overline{a}\,\overline{b}\,\overline{d}$ are paths. However, $\overline{a}\,\overline{b}c\overline{d}$ is not a path. Thus, an ISF represented by an IDD can be represented as follows:

$$f = \mathbf{0}\cdot\sum_{p\in\Pi_\mathbf{0}} p + \mathbf{1}\cdot\sum_{p\in\Pi_\mathbf{1}} p + \perp\cdot\sum_{p\in\Pi_\perp} p = \sum_{p\in\Pi_\mathbf{1}} p + \perp\cdot\sum_{p\in\Pi_\perp} p$$

Definition 3.2 (Cut)
Given and IDD, a cut *is a set of nodes such that every path crosses the nodes in the cut at most once.* **0**-, **1**- *and* $\perp$*-cuts are cuts that cut all* **0**-, **1**- *and* $\perp$*-paths, respectively. The fact that a path* p *crosses (does not cross) a cut* C *will be denoted by* $p \rightsquigarrow C$ ($p \not\rightsquigarrow C$).

In Figure 2, the set $\{b_2, c_3\}$ is a cut, whereas the set $\{b_2, c_1, c_3\}$ is not a cut. The set $\{c_1, c_2, c_3\}$ is a **1**-cut, and the set $\{c_2, c_3\}$ is a **0**-cut.

Definition 3.3 (Prefix and suffix)
Given a cut C *and a path* p*, the* prefix *of* p *with respect to* C*, denoted by* p^C*, is the subpath that leads from the root to one of the nodes of* C*. The sufix, denoted by* p_C*, is the rest of the path. In case* p *does not cross* C*, then* $p^C = p$ *and* $p_C = \emptyset$ *(or* $p_C = \mathbf{1}$*, when interpreted as a product).*

```
meet (f,g) {
    if ($f = g$) return $f$;
    if ($f = \bot$) return $g$;
    if ($g = \bot$) return $f$;
    if ($f, g$ are constant $\wedge$ $f \neq g$) return $\top$;
    $v$ = topVar($f$);
    $h_t$ = meet($f_t$,$g_t$); if ($h_t = \top$) return $\top$;
    $h_e$ = meet($f_e$,$g_e$); if ($h_e = \top$) return $\top$;
    if ($h_t = h_e$) return $h_t$;
    return buildIdd($v$,$h_t$,$h_e$);
}
```

```
join (f,g) {
    if ($f = g$) return $f$;
    if ($f = \bot$) return $\bot$;
    if ($g = \bot$) return $\bot$;
    if ($f, g$ are constant $\wedge$ $f \neq g$) return $\bot$;
    $v$ = topVar($f$);
    $h_t$ = join($f_t$,$g_t$);
    $h_e$ = join($f_e$,$g_e$);
    if ($h_t = h_e$) return $h_t$;
    return buildIdd($v$,$h_t$,$h_e$);
}
```

Figure 3: Algorithms for IDD *meet* and *join* operations (no cache operations included).

Given a cut C, a function f can be represented as follows:

$$f = \sum_{p \in \Pi} p^C \cdot f_{p^C} \tag{1}$$

where f_{p^C} is an ISF representing the cofactor of f with respect to the prefix of p. The function f_{p^C} is represented by the IDD node reached from the root by the prefix p^C. Note that for any path p not cut by C, we have that $p = p^C$ and f_{p^C} is the constant function corresponding to the terminal reached by p. Therefore, a function f can also be represented by the following equation:

$$f = \sum_{p \rightsquigarrow C} p^C \cdot f_{p^C} + \sum_{p \not\rightsquigarrow C} p \cdot f_p \tag{2}$$

4 IDD operations

IDD operations for conjuntion, disjuntion and complement are easily implemented by the recursive paradigm provided by Boole's expansion and the terminal cases derived from the tables in Definition 2.2.

Figure 3 shows the algorithms for the *meet* and *join* operations. The meet algorithm returns $\top$ in the case that the two functions are not comparable. An exponential reduction in the efficiency of the algorithms can be achieved by caching the intermediate results obtained during the recursive calculation of the operations, as it is typically done in BDD packages [Bry92].

Figure 4 shows an algorithm for IDD minimization. Once the terminal cases have been solved, two possibilities are considered for the recursive cases. Let f be represented by the triple (v, f_t, f_e), where v is the top variable and f_t and f_e the cofactors with respect to v. The minimization can be calculated as:

$$\mathit{minimize}((v, f_t, f_e)) = \begin{cases} g = (v, \mathit{minimize}(f_t), \mathit{minimize}(f_e)) \\ h = \mathit{minimize}(f_t \sqcap f_e) \end{cases}$$

In case $h = \top$, i.e. no upper bound exists for f_t and f_e, then g is the selected solution, otherwise the solution with minimum cost is selected. In the proposed algorithm, the selection is done according to the IDD size. Note that the solution h removes the variable v.

```
minimize ($f$) {
   /* Pre-condition:  $f \neq \bot$ */
   if ($f = \mathbf{0}$ $\vee$ $f = \mathbf{1}$) return $f$;
   if ($f_t = \bot$) return minimize($f_e$);
   if ($f_e = \bot$) return minimize($f_t$);
   $v$ = topVar($f$); $g_t$ = minimize($f_t$); $g_e$ = minimize($f_e$);
   if ($g_t = g_e$) $g$ = $g_t$; else $g$ = buildIdd($v$,$g_t$,$g_e$);
   $f_{meet}$ = $f_t \sqcap f_e$; if ($f_{meet} = \top$) return $g$;
   $h$ = minimize($f_{meet}$);
   if (IddSize($g$) < IddSize($h$)) return $g$;
   return $h$;
}
```

Figure 4: Algorithm for IDD minimization (no cache operations included).

5 Conjunctive and disjunctive decomposition of IDDs

Theorem 5.1 *Let f be an ISF represented by and IDD and C a $\mathbf{1}$-cut of the IDD. Then,*

$$f \sqsubseteq \sum_{p \leadsto C} p^C \cdot f_{p^C}$$

Proof: Let us take equation (2) for f. Since C is a $\mathbf{1}$-cut, for all paths that do not cross C we have that $f_p \in \{\mathbf{0}, \bot\}$. By the monotonicity of $\cdot$ and $+$, we can substitute $\bot$ by $\mathbf{0}$ for all f_p and remove the second summand of f, thus obtaining a more specified function. □

Theorem 5.2 *Let C be a cut of the IDD representing a function f, and let g be an ISF such that $f_{p^C} \sqsubseteq g$ for any path p crossing C. Then,*

$$f \sqsubseteq g \cdot \sum_{p \leadsto C} p^C + \sum_{p \not\leadsto C} p \cdot f_p$$

Proof: It immediately follows by considering the monotonicity of $\cdot$ and $+$, and by substituting f_{p^C} by g in equation (2). □

Corollary 5.3 (Factorization)
Let C be a cut of the IDD representing a function f, and let $g = \bigsqcap_{p \leadsto C} f_{p^C} \neq \top$. Then,

$$f \sqsubseteq g \cdot \sum_{p \leadsto C} p^C + \sum_{p \not\leadsto C} p \cdot f_p \tag{3}$$

Proof: By the definition of meet, $f_{p^C} \sqsubseteq g$ for all paths crossing C. The corollary immediately follows from theorem 5.2. □

Theorem 5.4 (Conjunctive decomposition)
Let C be a $\mathbf{1}$-cut of the IDD representing a function f, and let $g = \bigsqcap_{p \leadsto C} f_{p^C} \neq \top$. Then,

$$f \sqsubseteq g \cdot \left(\sum_{p \leadsto C} p^C + \sum_{p \not\leadsto C} p \cdot f_p \right)$$

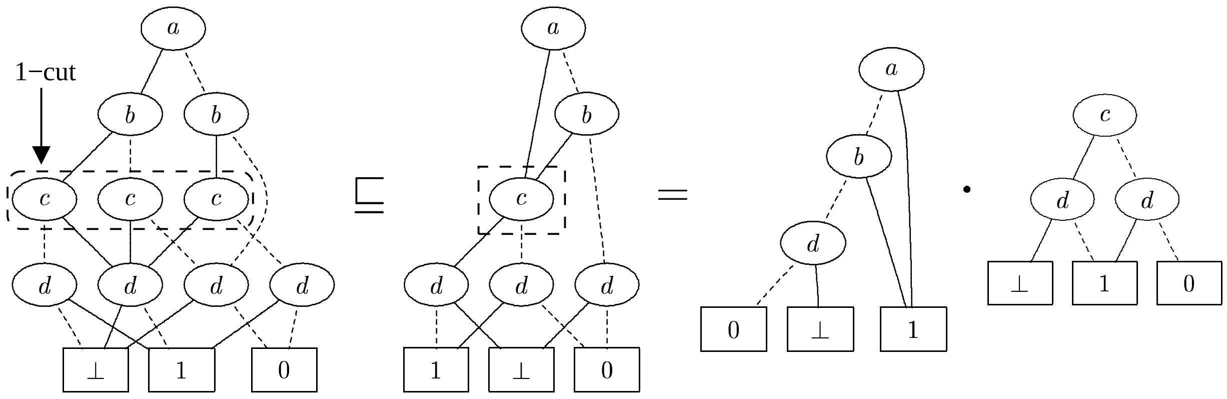


Figure 5: Conjunctive decomposition.

Proof: From corollary 5.3, equation (3) holds. The equation can be rewritten as follows:

$$f \sqsubseteq g \cdot \sum_{p \rightsquigarrow C} p^C + \mathbf{0} \cdot \sum_{\substack{p \not\rightsquigarrow C \\ f_p = \mathbf{0}}} p + \perp \cdot \sum_{\substack{p \not\rightsquigarrow C \\ f_p = \perp}} p$$

Since $g \cdot \mathbf{0} = \mathbf{0}$ and $\perp \sqsubseteq g \cdot \perp$, we also have

$$f \sqsubseteq g \cdot \sum_{p \rightsquigarrow C} p^C + g \cdot \mathbf{0} \cdot \sum_{\substack{p \not\rightsquigarrow C \\ f_p = \mathbf{0}}} p + g \cdot \perp \cdot \sum_{\substack{p \not\rightsquigarrow C \\ f_p = \perp}} p$$

and the proof of the theorem immediately follows by factoring g out. □

Corollary 5.5 *Under the same conditions of theorem 5.4,*

$$f \sqsubseteq g \cdot \sum_{p \rightsquigarrow C} p^C$$

Proof: It can be easily proved by substituting $\perp$ by $\mathbf{0}$ in all paths that do not cross C. □

Example 5.6 *Figure 5 illustrates the conjunctive decomposition of the ISF in Figure 2. The* **1***-cut is substituted by the meet of the nodes in the cut. The disjunctive decomposition is obtained by substituting the cut by the constant* **1** *in the upper part of the cut. Note that the realization $(a+b)(c+d)$ is one of the completely specified functions (maximal upper bounds) of the conjunctive decomposition. However, theorem 5.4 gives the least specified decomposition. For example, the decomposition $(a+b)(c \oplus d)$ is another valid realization that can be obtained by substituting $\perp$ by* **0** *in the rightmost IDD of Figure 5.*

A similar approach can be proposed for disjunctive decomposition, given the duality of the operations $\cdot$ and $+$. A disjunctive decomposition of f can be obtained by deriving a conjunctive decomposition of $\overline{f}$ and complementing the disjuncts[2]. Alternatively, an approach similar to the one for conjunctive decomposition, but using **0**-cuts instead of **1**-cuts, can also be applied.

[2]Note that the complement of an IDD can be simply obtained by interchanging the constants **0** and **1**.

6 XOR and MUX decompositions of IDDs

Theorem 6.1 (XOR decomposition)
Let C be a cut of the IDD such that all **0**- *and* **1**-*paths are cut by C. Let $C = C_0 \cup C_1$ and $C_0 \cap C_1 = \emptyset$. Let $X_0 = \bigsqcap_{p \rightsquigarrow C_0} f_{p^{C_0}}$, $X_1 = \bigsqcap_{p \rightsquigarrow C_1} f_{p^{C_1}}$ and $X = X_0 \sqcap \overline{X_1} \neq \top$. Then*

$$f \sqsubseteq X \oplus \left(\sum_{p \rightsquigarrow C_1} p^{C_1} + \bot \cdot \sum_{p \not\rightsquigarrow C} p \right)$$

Proof: Let us first define

$$P_0 = \sum_{p \rightsquigarrow C_0} p^{C_0}; \quad P_1 = \sum_{p \rightsquigarrow C_1} p^{C_1}; \quad P_\bot = \sum_{p \not\rightsquigarrow C} p$$

We also know that P_0, P_1 and $P_\bot$ define a partition of the set of paths of the IDD, i.e.

$$P_0 + P_1 + P_\bot = \mathbf{1}; \quad P_0 \cdot P_1 = P_0 \cdot P_\bot = P_1 \cdot P_\bot = \mathbf{0}$$

We can also prove that the following equality holds: $P_0 + \bot P_\bot = \overline{P_1 + \bot P_\bot}$

$$\begin{aligned}
\overline{P_1 + \bot P_\bot} &= \overline{P_1} \cdot \overline{\bot P_\bot} \\
&= (P_0 + P_\bot) \cdot (\bot + P_0 + P_1) \qquad \text{[since } P_0, P_1 \text{ and } P_\bot \text{ is a partition of the paths]} \\
&= \bot P_0 + P_0 + P_0 P_1 + \bot P_\bot + P_\bot P_0 + P_\bot P_1 \\
&= P_0 + \bot P_\bot
\end{aligned}$$

We can represent f as

$$f = \sum_{p \rightsquigarrow C_0} p^{C_0} f_{p^{C_0}} + \sum_{p \rightsquigarrow C_1} p^{C_1} f_{p^{C_1}} + \bot \sum_{p \not\rightsquigarrow C} p$$

For all paths cut by C_0 and C_1 we have $X_0 \sqsubseteq f_{p^{C_0}}$ and $X_1 \sqsubseteq f_{p^{C_1}}$, respectively. Therefore,

$$f \sqsubseteq X_0 P_0 + X_1 P_1 + \bot P_\bot$$

Since $\overline{X}_1 \sqsubseteq X$, we also have that $X_1 \sqsubseteq \overline{X}$. Moreover, we have that $X_0 \sqsubseteq X$. Then[3],

$$\begin{aligned}
f &\sqsubseteq XP_0 + \overline{X}P_1 + \bot P_\bot \sqsubseteq \\
&\sqsubseteq XP_0 + \overline{X}P_1 + (X + \overline{X})\bot P_\bot = \\
&= X(P_0 + \bot P_\bot) + \overline{X}(P_1 + \bot P_\bot) = \\
&= X \cdot \overline{P_1 + \bot P_\bot} + \overline{X}(P_1 + \bot P_\bot) = \\
&= X \oplus (P_1 + \bot P_\bot)
\end{aligned}$$

□

Example 6.2 *Figure 6 shows an example of XOR decomposition. The cut $C = C_0 \cup C_1$ is substituted by X and $\overline{X}$. The prefixes and sufixes of the cut derive the new ISFs, shown at the rightmost part of the figure.*

[3]Note that $X + \overline{X} = 1$ does not always hold in ternary algebra.

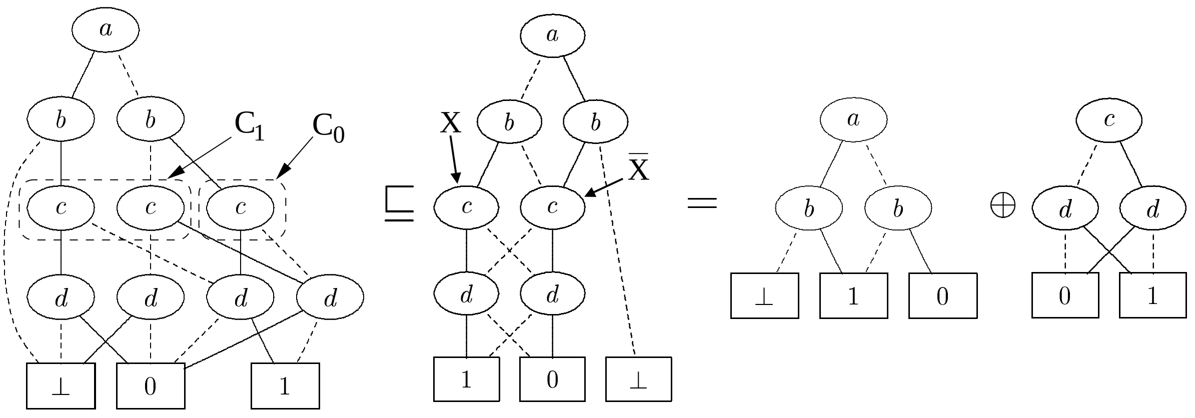


Figure 6: XOR decomposition.

The MUX decomposition can be applied when the same conditions of the XOR decomposition hold, except for the fact that $X_0 \sqcap \overline{X_1} \neq \top$. In that case, a less stringent decomposition can be done.

Corollary 6.3 (MUX decomposition)
Let C be a cut of the IDD such that all $\mathbf{0}$- and $\mathbf{1}$-paths are cut by C. Let $C = C_0 \cup C_1$ and $C_0 \cap C_1 = \emptyset$. Let $X_0 = \prod_{p \rightsquigarrow C_0} f_{p^{C_0}} \neq \top$ and $X_1 = \prod_{p \rightsquigarrow C_1} f_{p^{C_1}} \neq \top$. Let

$$Y = \sum_{p \rightsquigarrow C_1} p^{C_1} + \bot \cdot \sum_{p \not\rightsquigarrow C} p$$

Then

$$f \sqsubseteq Y X_1 + \overline{Y} X_0$$

Proof: The proof is similar to the one for Theorem 6.1. □

7 Decomposition by function approximation

When the decompositions presented in the previous sections cannot be applied, other methods must be sought. In this section, a method based on function approximations is described. The method is illustrated by the example in Figure 7. Using approximations to decompose BDDs has already been proposed in the context of formal verification [RMSS98].

Imagine that we would target at a conjunctive decomposition, $f = g \cdot h$, of the function represented in Figure 7. Clearly, each conjunct must be an overapproximation of f, i.e. $f(x) = \mathbf{1} \implies g(x) = h(x) = \mathbf{1}$. By overapproximating f by g (see Figure 8), a new function accepting a XOR decomposition, as shown in Figure 9, would be derived. However, g is not a valid realization for f. The least specified function h for the other conjunct can be obtained as follows:

$$\begin{aligned}
f(x) = \mathbf{1} \quad &\implies \quad h(x) = \mathbf{1} \\
f(x) = \bot \quad &\implies \quad h(x) = \bot \\
f(x) = \mathbf{0} \ \wedge \ g(x) = \mathbf{0} \quad &\implies \quad h(x) = \bot \\
f(x) = \mathbf{0} \ \wedge \ g(x) \neq \mathbf{0} \quad &\implies \quad h(x) = \mathbf{0}
\end{aligned}$$

cd \ ab	00	01	11	10
00	**0**	**0**	**0**	**1**
01	**1**	**0**	**1**	**0**
11	**0**	**1**	$\perp$	$\perp$
10	**1**	**0**	**1**	**0**

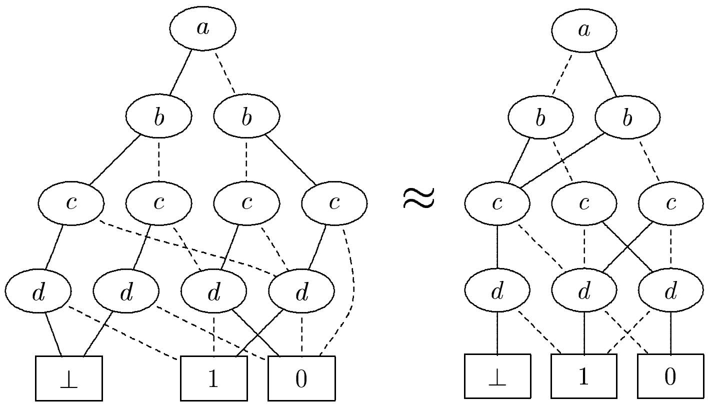


Figure 7: Approximation of an IDD without disjoint decomposition.

0	**0**	**0**	**1**
1	**0**	**1**	**0**
0	**1**	$\perp$	$\perp$
1	**0**	**1**	**0**

f

$=$

0	**1**	**0**	**1**
1	**0**	**1**	**0**
0	**1**	$\perp$	$\perp$
1	**0**	**1**	**0**

g

$\cdot$

$\perp$	**0**	$\perp$	**1**
1	$\perp$	**1**	$\perp$
$\perp$	**1**	$\perp$	$\perp$
1	$\perp$	**1**	$\perp$

h

Figure 8: Overapproximation of f by g.

Needless to say that the accuracy of g in approximating f affects the number of assignments to $\perp$ in h. Therefore, there is a tradeoff between the accuracy of g and the complexity of h. In the example, g only differs from f in one assignment.

This approximation can also be done at the level of IDD by using techniques similar to those proposed in [RMSS98] to obtain dense approximations.

After having factored g out of f, a new node implementing g in the Boolean network is created. This new node can also be used to implement other nodes in the Boolean network. Moreover, the implementation of g also incorporates new don't care conditions. Let us assume that g has finally been implemented as

$$g = a \oplus b \oplus c \oplus d.$$

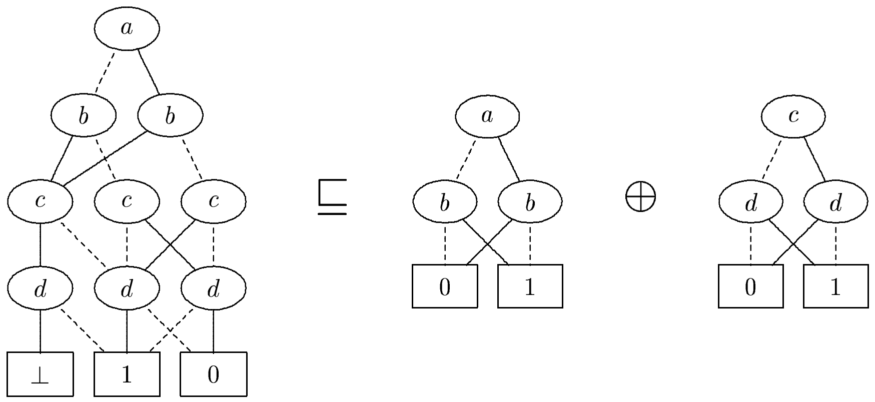


Figure 9: XOR decomposition of function g.

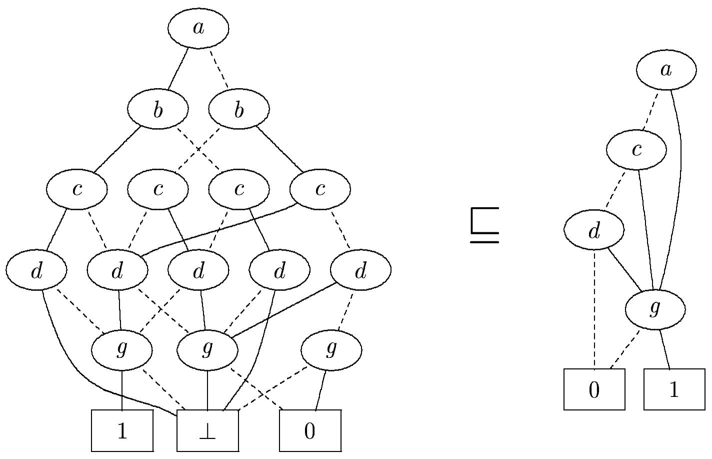


Figure 10: Function f' and its simplification after applying the algorithm in Figure 4.

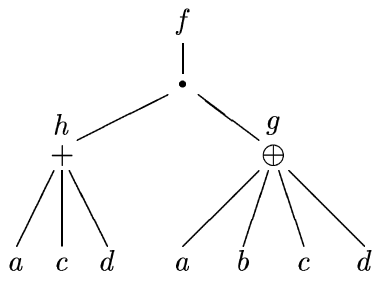


Figure 11: Netlist for function f after logic decomposition

Thus, the condition $g \neq a \oplus b \oplus c \oplus d$ is now a new don't care condition that can be incorporated into f as follows:

$$f' = f \cdot (\overline{g} \oplus a \oplus b \oplus c \oplus d) + \perp \cdot (g \oplus a \oplus b \oplus c \oplus d)$$

where g is a new variable representing the node implementing function g in the Boolean network. The previous equation evaluates to $\perp$ when the variable g is different from the value of the function g. The representation of f' as an IDD is shown in Figure 10.

Now function f' can be decomposed by some of the techniques previously presented. For example, a cut including the two leftmost nodes with label g would be a **1**-cut for conjunctive decomposition.

Figure 10 depicts the IDD obtained after minimizing f'. The result can be decomposed as follows (see Figure 11):

$$\begin{aligned} h &= a + c + d \\ f &= h \cdot g \end{aligned}$$

8 Conclusions

This paper has presented some theoretical background for the Boolean decomposition of ISFs. The main feature of the theory is the preservation of the undefinedness of ISFs during decomposition.

While being more powerful than algebraic decomposition, the proposed theory does not have as much flexibility as Boolean relations or SPDFs. However, the complexity can be kept manageable for moderate-size functions if IDDs are used for the manipulation of ISFs.

An algorithmic framework for the decomposition of Boolean functions is a near-future task that will enable to evaluate the effectiveness of this theory in practice.

References

[Bro90] F.M. Brown. *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[Bry92] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[CF93] K.-C. Chen and M. Fujita. Network optimization using don't-cares and Boolean relations. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 87–108. Kluwer Academic Publishers, 1993.

[Kle52] S.C. Kleene. *Introduction to Metamathematics.* Wolters-Noordhoff, Noth-Holland Publishing, 1952.

[MF89] Y. Matsunaga and M. Fujita. Multi-level logic optimization using binary decision diagrams. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 556–559, November 1989.

[Muk83] M. Mukaidono. Regular ternary logic functions – ternary logic functions suitable for treating ambiguity. In *Proc. of the 13th Annual Symposimum on Multiple-Valued Logic*, pages 286–291. IEEE Press, 1983.

[RMSS98] K. Ravi, K.L. McMillan, T.R. Shiple, and F. Somenzi. Approximation and decomposition of binary decision diagrams. In *Proc. ACM/IEEE Design Automation Conference*, 1998.

[Sas96] T. Sasao. Ternary decision diagrams and their applications. In T. Sasao and M. Fujita, editors, *Representations of Discrete Functions.* Kluwer Academic Publishers, 1996.

[TKNM98] N. Takagi, H. Kikuchi, K. Nakashima, and M. Mukadiono. Identification of Incompletely Specified Multiple-Valued Kleenean Functions. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 28(5):637–647, September 1998.

[YCS00] C. Yang, M. Ciesielski, and V. Singhal. BDS: a BDD-based logic optimization system. In *Proc. ACM/IEEE Design Automation Conference*, pages 92–97, 2000.

[YSC99] C. Yang, V. Singhal, and M. Ciesielski. BDD decomposition for efficient logic synthesis. In *Proc. International Conf. Computer Design (ICCD)*, pages 626–631, 1999.

[YSH00] S. Yamashita, H. Sawada, and A. Hagoya. SPFD: a new method to express functional flexibility. *IEEE Transactions on Computer-Aided Design*, 19(8):840–849, August 2000.