Functional Abstraction and Partial Specification of Boolean Functions

Karem A. Sakallah

CSE-TR-255-95

August 9, 1995

THE UNIVERSITY OF MICHIGAN

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109-2122
USA

Functional Abstraction and Partial Specification of Boolean Functions

Karem A. Sakallah

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2122

August 9, 1995

Abstract

We define functional abstraction as the process of deliberately ignoring the dependence of a Boolean function on a subset of its variables. Functional abstraction causes a completely specified function to become partially specified. We propose function sets as a theoretical model for partially specified functions and function intervals as a practical approximation to them. We develop an interval Boolean algebra suitable for the symbolic manipulation of function intervals and highlight the relationship between functional abstraction and universal and existential quantification.

Notational Conventions and Glossary of Symbols

We will generally use lower-case symbols to denote scalar quantities and upper-case symbols to denote aggregates (vectors and sets.) Calligraphic type will denote the carriers (universal sets) of algebraic structures. Unless explicitly stated otherwise, when we speak of Boolean variables and functions we mean variables and functions in the 2-element Boolean (switching) algebra. Thus, $x_1, x_2, \ldots, x_n$ refer to switching variables, f, g, h denote switching functions, $X = (x_1, x_2, \ldots, x_n)$ represents a vector of switching variables, and F, G, H denote sets of switching functions.

Notation	*Meaning*	*Example*
$\mathcal{B}$	2-element Boolean algebra $\{0, 1\}$	—
$\mathcal{S}$	Set of intervals on $\mathcal{B}$; $\mathcal{S} = \{[0,0], [0,1], [1,1]\}$. When clear from context, these three intervals may be relabeled 0, U and 1 respectively	—
$x_1, x_2, \ldots, x_n, y, z$	2-element Boolean variables	—
$\bar{x}, x', \neg x$	Complement	—
$xy, x \cdot y, x \wedge y$	AND	—
$x \vee y$	OR	—
$x \oplus y$	Exclusive-Or (XOR)	—
$x \odot y$	Equivalence; exclusive-NOR (XNOR)	—
$X = (x_1, x_2, \ldots, x_n)$	Vector of n Boolean variables	—
$f(X), g(X), h(X)$	n-variable Boolean functions	$f(x_1, x_2) = x_1 \vee \bar{x}_2$
$\Vert f(X) \Vert$	Number of minterms for which $f(X) = 1$	$\Vert x_1 \oplus x_2 \Vert = 2$
$\mathcal{U}_n(X)$	Set of all n-variable Boolean functions	$\mathcal{U}_1(x_1) = \{0, \bar{x}_1, x_1, 1\}$
$F(X), G(X), H(X)$	Partially specified n-variable Boolean functions; subsets of $\mathcal{U}_n(X)$	$F(x_1, x_2) = \{x_1, x_1 \vee \bar{x}_2, \bar{x}_1 x_2\}$
$\lvert F(X) \rvert$	Size (cardinality) of $F(X)$	$\lvert\{x_1, x_1 \vee \bar{x}_2, \bar{x}_1 x_2\}\rvert = 3$
$F \cup G$	Set union	$\{x_1\} \cup \{\bar{x}_1 x_2\} = \{x_1, \bar{x}_1 x_2\}$
$F \cap G$	Set intersection	$\{x_1, x_2\} \cap \{x_1, \bar{x}_1\} = \{x_1\}$
$\lfloor F(X) \rfloor$	Greatest lower bound (glb) of $F(X)$	$\lfloor \{x_1, x_1 \vee \bar{x}_2, x_1 x_2\} \rfloor = x_1 x_2$
$\lceil F(X) \rceil$	Least upper bound (lub) of $F(X)$	$\lceil \{x_1, x_1 \oplus x_2, x_2\} \rceil = x_1 \vee x_2$
$U(x)$	Uncertainty set of Boolean variable x	$U(x) \equiv \{0, \bar{x}, x, 1\}$

Notation	*Meaning*	*Example*
$F(X)_{U(x_i)}$	Abstraction of $F(X)$ with respect to x_i	$\{x_1x_2\}_{U(x_1)} = \{x_1x_2, 0, \bar{x}_1x_2, x_2\}$
$[l(X), u(X)]$	Interval in $\mathcal{U}_n(X)$ between $l(X)$ and $u(X)$	$[x_1x_2, x_1 \vee x_2]$
$[F(X)]$	Smallest interval containing $F(X)$	$[\{x_1 \oplus x_2, x_1 \odot x_2\}] = [0, 1]$
$F(X)_{[U(x_i)]}$	Conservative abstraction of $F(X)$ with respect to x_i	$\{x_1x_2\}_{[U(x_1)]} = [0, x_2]$
$F(X) \lozenge G(X)$	Expand operator; interval containing $F(X) \cup G(X)$	—

1 Introduction

In this report we address the following question: “Given a completely specified *n*-variable Boolean function $y = f(x_1, x_2, \ldots, x_n)$, what is the effect of ignoring its dependence on one or more of its variables?” We refer to this process as *functional abstraction*. Intuitively, functional abstraction introduces *uncertainty* in our knowledge of the function and causes the function to become *partially specified*. Partial specification of Boolean functions has traditionally been examined in the context of logic synthesis by way of exploiting the inherent flexibility in a partially specified function to produce more efficient implementations than would otherwise be possible from a complete specification. Partial specification can also be viewed as a vehicle for approximate circuit analysis. In this scenario, the objective is to establish efficiently-computed *bounds* on circuit behavior by judicious injection of ambiguity in a circuit’s functional specification.

A particularly important example of this type of approximation is timing analysis where we seek to determine the earliest and latest event times for circuit signals under all possible input stimuli. Theoretically, these times can be determined exactly by exhaustive simulation. In practice, though, this approach is infeasible except for trivially small circuits. “Incomplete” simulation may be acceptable in certain cases if the input vectors can reasonably be expected to stimulate extreme circuit behavior (longest path, for example.) Generally, however, incomplete simulation does not provide a coverage guarantee and carries the risk of false negatives. Alternatively, functional abstraction allows us to err conservatively by bounding extreme behavior; it guarantees complete coverage by embedding the sought-after behavior in a larger set of behaviors that are easier to determine. The inevitable incidence of false positives in such an approach can be controlled by appropriate choice of which variables to abstract away. The loosest bounds are found by structural (topological) analysis of the circuit and correspond to total abstraction of all functional information. Tighter bounds can be obtained by abstracting away only some but not all functional dependencies.

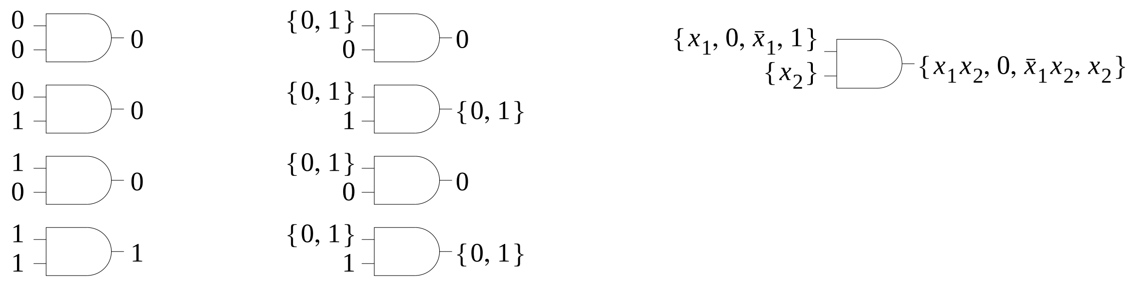
The use of uncertain signal values in models of logic gate circuits has a long history that started with the introduction, by Muller [13], of a third value to the two-element switching algebra. This third value—which Muller wrote as $\frac{1}{2}$—was meant to model digital signals in transition between the binary values 0 and 1. Muller also pointed out that the resulting ternary algebra was formally equivalent to the strong 3-valued logic of Kleene [10]. Since that time, this third value has become a standard feature of logic simulation models and has found additional applications such as circuit initialization, hazard detection, race detection, etc. [4]. More recently, Hayes [8, 9] examined uncertainty in the context of multiple-valued logics and showed how logics that incorporate more than one uncertain value can be systematically generated. Examples, other than timing analysis, of approximate modeling through careful introduction of uncertainty have been studied by Harkness [6, 7] and include switch-

x_1	x_2	$f(x_1, x_2)$
		$x_1 x_2$
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$f(x_1, x_2)$
		$x_1 x_2$
$\{0, 1\}$	0	0
$\{0, 1\}$	1	$\{0, 1\}$
$\{0, 1\}$	0	0
$\{0, 1\}$	1	$\{0, 1\}$

$\Leftrightarrow$

$g(x_1)$				x_2	$f(x_1, x_2)$			
x_1	0	$\bar{x}_1$	1		$x_1 x_2$	0	$\bar{x}_1 x_2$	x_2
0	0	1	1	0	0	0	0	0
0	0	1	1	1	0	0	1	1
1	0	0	1	0	0	0	0	0
1	0	0	1	1	1	0	0	1



(a) AND function specification (b) "Numeric" model of AND function when x_1 is unknown (c) Symbolic model of AND function when x_1 is unknown

Figure 2-1: Modeling uncertainty for AND gate

level simulation with uncertain signal strength, delay in RC networks with uncertain parameter values, and placement using uncertain costs. Zukowski [16] also demonstrated that tight bounds on voltage waveforms in circuit simulation can be efficiently obtained when appropriate bounds on the circuit excitations are assumed.

The remainder of this report is divided into four sections. In Section 2 we motivate the use of function sets for modeling functional uncertainty and establish the computation rules required for their manipulation. In Section 3 we formally define functional abstraction and establish some of its important properties. While providing the theoretical basis for functional abstraction, explicitly enumerated function sets are unwieldy and have little, if any, practical value. We address this deficiency in Section 4 by introducing function intervals as approximations to unrestricted function sets. We also develop a suitable interval Boolean algebra for function intervals, illustrate its use with several examples, and conclude by revealing the relationship between functional abstraction and universal and existential quantification. Section 5 summarizes the main points of the report and highlights ongoing and future applications.

2 Modeling Uncertainty with Function Sets

Consider the functional behaviors that can be observed at the output of a two-input AND gate $f(x_1, x_2) = x_1 x_2$ when the value of the first input x_1 is unknown. As shown in Figure 2-1, uncertainty about the value of x_1 can be modeled by replacing each entry in the x_1 column of the AND truth table with the set $\{0, 1\}$. Application of the AND function to x_2 and this set-valued x_1 yields, in turn, an uncertain set-valued result. This *numeric* model of uncertainty can be converted to an equivalent *symbolic* model by transforming the uncertainty in the *values* of x_1 and *f* to uncertainty about their *functional* dependence on x_1. In this view, the first input of the AND gate is treated as a 1-variable Boolean function $g(x_1) = x_1$. Thus, when the value of x_1 is known, the AND gate "sees" the function x_1 at its first input and produces the expected function x_1x_2 at its output. However, when the value of x_1 is unknown, the AND gate is unable to distinguish among the four possible single-variable Boolean functions of x_1, namely the functions in the set $\{x_1, 0, \bar{x}_1, 1\}$. Consequently, the gate output $f(x_1, x_2)$ becomes uncertain since, in addition to the original function x_1x_2, it can also be any of the functions 0, $\bar{x}_1x_2$, or x_2. Symbolically, these observations are captured by the equation:

$$f(\{x_1, 0, \bar{x}_1, 1\}, \{x_2\}) = \{x_1, 0, \bar{x}_1, 1\} \wedge \{x_2\} = \{x_1x_2, 0, \bar{x}_1x_2, x_2\} \tag{2.1}$$

where the inputs and output of the AND gate are considered to be *function sets* rather than single functions.

This example suggests that function sets arise naturally in the context of modeling functional uncertainty. Indeed, function sets provide the most general framework for modeling uncertainty in the functional behavior of logic circuits. Other models, such as don't-cares, Boolean relations [3], or function intervals [5, p. 45], are easily shown to be special cases of function sets.[1] In the remainder of this section, we provide a formal definition of partially specified Boolean functions as function sets and examine several of their properties.

Definition 2.1 (Boolean Function Space) *The function space, or universe, of n-variable Boolean functions is the* 2^{2^n} *-element set denoted by* $\mathcal{U}_n(x_1, x_2, \ldots, x_n)$.[2] □

Definition 2.2 (Partially Specified Boolean Functions) *A partially specified n-variable Boolean function* $F(x_1, x_2, \ldots, x_n)$ *is a nonempty subset of the function space* $\mathcal{U}_n(x_1, x_2, \ldots, x_n)$. *The cardinality of the set F, denoted by* $|F|$, *is the number of n-variable Boolean functions it contains and indicates the degree of its uncertainty. When* $|F| = 1$, *F is said to be completely specified. When* $F = \mathcal{U}_n$, *F is said to be completely unspecified.* □

Unless stated otherwise, the phrases "function" and "partially specified function" in this report should be interpreted as abbreviations for "partially specified *n*-variable Boolean function."

Example 2.1 The function space of 2-variable Boolean functions is the sixteen-element set

$$\begin{aligned}\mathcal{U}_2(x_1, x_2) = \{&0, \bar{x}_1\bar{x}_2, \bar{x}_1x_2, x_1\bar{x}_2, x_1x_2, \bar{x}_1, \bar{x}_2, x_1, x_2,\\ &x_1 \odot x_2, x_1 \oplus x_2, \bar{x}_1 \vee \bar{x}_2, \bar{x}_1 \vee x_2, x_1 \vee \bar{x}_2, x_1 \vee x_2, 1\}\end{aligned}$$

1. Don't-cares, Boolean relations, and function intervals yield identical function sets when used for the partial specification of single-output Boolean functions. A natural extension of don't-cares to *m*-output Boolean functions is to view them as subsets of two or more elements of the power set of $\mathcal{B}^m$. The function sets generated by such don't-cares are identical to those produced by a Boolean relation from $\mathcal{B}^n$ to $\mathcal{B}^m$. In contrast, function intervals result when the *m* functions are individually treated as single-output partially specified functions.
2. Brown [5, p. 47] denotes this set by $F_n(B)$ where B is, in general, a *k*-element Boolean algebra.

The sets $F(x_1, x_2) = \{0, 1\}$, $G(x_1, x_2) = \{x_1\bar{x}_2, x_1 \oplus x_2, \bar{x}_1 \vee x_2\}$, and $H(x_1, x_2) = \{x_1 \vee x_2\}$ correspond to partially specified 2-variable functions whose uncertainties are equal to 2, 3, and 1, respectively. Since $|H| = 1$, H is completely specified. □

Partially specified functions have two facets. On the one hand, *they are sets* and can thus be related by set inclusion and manipulated by set operations such as union, intersection, and difference. On the other hand, *their elements are Boolean functions* that can be manipulated by Boolean operators such as AND, OR, and NOT and that, more generally, can be composed with other Boolean functions. It is, therefore, appropriate to manipulate partially specified functions algebraically. Such algebraic manipulation is based on the following pair of functional composition rules [9].

Definition 2.3 (Functional Composition Rules for Partially Specified Boolean Functions)

- Composition Rule #1*: Let $\varphi(y_1, y_2, \ldots, y_k)$ be a k-variable Boolean function, and let $G_1(X), G_2(X), \ldots, G_k(X)$ be k partially specified n-variable Boolean functions. Then $\varphi(G_1(X), G_2(X), \ldots, G_k(X))$ is a partially specified n-variable Boolean function defined by the following rule:*

$$\varphi(G_1(X), G_2(X), \ldots, G_k(X)) \equiv \bigcup_{\substack{v(X) \in G_1(X) \\ w(X) \in G_2(X) \\ \ldots \\ z(X) \in G_k(X)}} \{\varphi(v(X), w(X), \ldots, z(X))\} \tag{2.2}$$

- Composition Rule #2*: Let $F(X), G_1(X), G_2(X), \ldots, G_n(X)$ be partially specified n-variable Boolean functions. Then $F(G_1(X), G_2(X), \ldots, G_n(X))$ is a partially specified n-variable Boolean function defined by the rule:*

$$F(G_1(X), G_2(X), \ldots, G_n(X)) \equiv \bigcup_{f(X) \in F(X)} f(G_1(X), G_2(X), \ldots, G_n(X)) \tag{2.3}$$

Note that the function F can be viewed as a mapping from $\left(2^{\mathcal{U}_n}\right)^n = 2^{\mathcal{U}_n} \times 2^{\mathcal{U}_n} \times \ldots \times 2^{\mathcal{U}_n}$ to $2^{\mathcal{U}_n}$ where $2^{\mathcal{U}_n}$ is the power set of $\mathcal{U}_n$. □

Example 2.2 Let

$$\begin{aligned} F(x_1, x_2) &= \{x_1x_2, 0, \bar{x}_1x_2, x_2\} \\ G_1(x_1, x_2) &= \{x_1, \bar{x}_1 \vee x_2\} \\ G_2(x_1, x_2) &= \{0, \bar{x}_2, x_2, 1\} \\ G_3(x_1, x_2) &= \{x_1\} \end{aligned}$$

Then,

$$
\begin{aligned}
&G_3(x_1, x_2) \subseteq G_1(x_1, x_2)\\
&F(x_1, x_2) \cap G_2(x_1, x_2) = \{0, x_2\}\\
&F(x_1, x_2) - G_2(x_1, x_2) = \{x_1x_2, \bar{x}_1x_2\}
\end{aligned}
$$

$$
\begin{aligned}
F(G_3, G_2) &= F(\{x_1\}, \{0, \bar{x}_2, x_2, 1\})\\
&= (\{x_1\} \wedge \{0, \bar{x}_2, x_2, 1\}) \cup \{0\} \cup (\{\bar{x}_1\} \wedge \{0, \bar{x}_2, x_2, 1\}) \cup \{0, \bar{x}_2, x_2, 1\}\\
&= \{0, x_1\bar{x}_2, x_1x_2, x_1\} \cup \{0\} \cup \{0, \bar{x}_1\bar{x}_2, \bar{x}_1x_2, \bar{x}_1\} \cup \{0, \bar{x}_2, x_2, 1\}\\
&= \{0, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, \bar{x}_1x_2, x_1x_2, \bar{x}_1, \bar{x}_2, x_1, x_2, 1\}\\
F'(x_1, x_2) &= \{\bar{x}_1 \vee \bar{x}_2, 1, x_1 \vee \bar{x}_2, \bar{x}_2\}\\
F_{x_1} = F(x_1, x_2)\big|_{x_1 = 1} &= \left\{x_1x_2\big|_{x_1 = 1}\right\} \cup \left\{0\big|_{x_1 = 1}\right\} \cup \left\{\bar{x}_1x_2\big|_{x_1 = 1}\right\} \cup \left\{x_2\big|_{x_1 = 1}\right\} = \{0, x_2\}
\end{aligned}
$$

where the last operation was the cofactor of F with respect to x_1 [2]. □

Finally, let us note that the function space $\mathcal{U}_n(x_1, x_2, \ldots, x_n)$ forms a 2^{2^n}-element Boolean algebra [5, p. 48], i.e. it is a distributive and complemented lattice [11, p. 64]. Thus, each subset of $\mathcal{U}_n$ has a unique greatest lower bound (glb) and a unique least upper bound (lub). The following definition prescribes how these bounds are calculated.

Definition 2.4 (*glb* and *lub* of Partially Specified Functions) *The greatest lower bound and least upper bound of the partially specified n-variable Boolean function $F(X)$ are n-variable Boolean functions denoted, respectively, by $\lfloor F(X) \rfloor$ and $\lceil F(X) \rceil$, and determined according to*:

$$
\begin{aligned}
\lfloor F(X) \rfloor &= \bigwedge_{f(X) \in F(X)} f(X)\\
\lceil F(X) \rceil &= \bigvee_{f(X) \in F(X)} f(X)
\end{aligned}
\qquad (2.4)
$$

In general, $\lfloor F(X) \rfloor$ and $\lceil F(X) \rceil$ are not necessarily members of $F(X)$. □

Example 2.3 Let $F(x_1, x_2) = \{x_1, x_1\bar{x}_2, x_1 \oplus x_2\}$. Then,

$$l(x_1, x_2) \equiv \lfloor F(x_1, x_2) \rfloor = x_1 \wedge x_1\bar{x}_2 \wedge (x_1 \oplus x_2) = x_1\bar{x}_2$$

and

$$u(x_1, x_2) \equiv \lceil F(x_1, x_2) \rceil = x_1 \vee x_1\bar{x}_2 \vee (x_1 \oplus x_2) = x_1 \vee x_2$$

Note that $l(x_1, x_2) \in F(x_1, x_2)$ whereas $u(x_1, x_2) \notin F(x_1, x_2)$. □

3 Functional Abstraction

As mentioned in the introduction, partial specification of Boolean functions has been mainly used in synthesis. In this report, however, we are primarily concerned with characterizing the partially specified functions that arise when we choose to ignore the functional dependence on certain variables. In this section we formally define functional abstraction and investigate its properties.

Definition 3.1 (Uncertainty Sets of Boolean Variables) *The uncertainty set of a Boolean variable x_i is the partially specified n-variable Boolean function $F(X) = \{0, \bar{x}_i, x_i, 1\}$. The shorthand notation, $U(x_i)$, will be used to denote this set and may be read as "x_i is unspecified." Further, the notation $U(V)$, where V is a set of variables, will be used to indicate that each of the variables in V is individually unspecified.* □

Definition 3.2 (Functional Abstraction) *A variable x_i is said to be functionally abstracted from a function $F(X)$ if every occurrence of x_i in $F(X)$ is replaced by $U(x_i)$. The result of abstracting x_i from F is a partially specified n-variable function $F(x_1, \ldots, U(x_i), \ldots, x_n)$ called the abstraction of F with respect to x_i and denoted as $F_{U(x_i)}$.*[3] *Functional abstraction of both x_i and x_j from F will be denoted by $F_{U(x_i, x_j)}$ and evaluated according to $F_{U(x_i, x_j)} \equiv F(x_1, \ldots, U(x_i), \ldots, U(x_j), \ldots, x_n)$. Abstraction of larger sets of variables is handled similarly.* □

It is important to emphasize that functional abstraction of a variable does not eliminate functional dependence on it. The difference between the functions $F(X)$ and $F(X)_{U(x_i)}$ is that the latter's dependence on x_i may be more ambiguous than the former's. The exact relationship between these two functions is provided by the following theorem.

Theorem 3.1 $F(X) \subseteq F(X)_{U(x_i)}$

Proof: The following expression for $F(X)_{U(x_i)}$ follows directly from definitions 3.1 and 3.2 and from application of the functional composition rule in (2.3):

$$\begin{aligned} F(X)_{U(x_i)} &= F(x_1, \ldots, U(x_i), \ldots, x_n) \\ &= F(x_1, \ldots, \{0, \bar{x}_i, x_i, 1\}, \ldots, x_n) \\ &= F(x_1, \ldots, 0, \ldots, x_n) \cup F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \cup \\ &\quad\ F(x_1, \ldots, x_i, \ldots, x_n) \cup F(x_1, \ldots, 1, \ldots, x_n) \end{aligned} \tag{3.1}$$

Thus $F(X)$ is a subset of $F(X)_{U(x_i)}$. □

Example 3.1 Let $F(x_1, x_2) = \{x_1x_2, 0, \bar{x}_1x_2, x_2\}$. Functional abstraction of x_2 yields:

3. This notation is similar to the cofactor notation [2] and can be read as "the function F with x_i unspecified." It suggests the evaluation of the function at the values indicated in the subscript.

$$\begin{aligned} F(x_1, U(x_2)) &= \{x_1\} \wedge U(x_2) \cup \{0\} \cup \{\bar{x}_1\} \wedge U(x_2) \cup U(x_2) \\ &= \{0, x_1\bar{x}_2, x_1x_2, x_1\} \cup \{0\} \cup \{0, \bar{x}_1\bar{x}_2, \bar{x}_1x_2, \bar{x}_1\} \cup \{0, \bar{x}_2, x_2, 1\} \\ &= \{0, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, \bar{x}_1x_2, x_1x_2, \bar{x}_1, \bar{x}_2, x_1, x_2, 1\} \end{aligned}$$

Note that $F(x_1, x_2) \subseteq F(x_1, U(x_2))$. □

Example 3.2 Let $f(x_1, x_2) = x_1x_2$. Functional abstraction of both x_1 and x_2 yields:

$$\begin{aligned} f_{U(x_1, x_2)} &= f(U(x_1), U(x_2)) = \{0, \bar{x}_1, x_1, 1\} \wedge \{0, \bar{x}_2, x_2, 1\} \\ &= \{0, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, \bar{x}_1x_2, x_1x_2, \bar{x}_1, \bar{x}_2, x_1, x_2, 1\} \end{aligned}$$ □

It is interesting to note that the partially specified function resulting from the abstraction of x_1 in Example 3.1 is identical to the function resulting from the abstraction of both x_1 and x_2 in Example 3.2. This is not coincidental since the starting functions in the two examples are related: $F(x_1, x_2) = f(U(x_1), x_2)$. Indeed, as stated in the following theorem, variables can be abstracted from a function in any order without affecting the result.

Theorem 3.2 $F(X)_{U(x_i, x_j)} = (F(X)_{U(x_i)})_{U(x_j)} = (F(X)_{U(x_j)})_{U(x_i)}$

Proof: Immediate from expansion of each side using composition rule #2 in (2.3). □

Corollary 3.3 $(F(X)_{U(x_i)})_{U(x_i)} = F(X)_{U(x_i)}$ □

Note also that the abstraction of both x_1 and x_2 from the two-variable AND function yields a ten-element partially specified function rather than the sixteen-element completely unspecified function $\mathcal{U}_2(x_1, x_2)$. This may seem odd considering that we are ambiguating the functional dependence on *all* input variables. This apparent paradox disappears, though, when we recall from definitions 3.1 and 3.2 that multiple variables are abstracted *individually.* Thus, while we may now no longer know the precise functional dependence on each of x_1 and x_2, we do know that they are still combined by an AND function. The maximum uncertainty set $\mathcal{U}_2(x_1, x_2)$ would have resulted had the definition of functional abstraction allowed for the *simultaneous* abstraction of both x_1 and x_2.

While variables can be abstracted away in any order, the following example demonstrates that abstracting a variable from a function may not produce the same result as abstracting the variable from a *circuit implementation* of that function.

Example 3.3 Let $f(x_1, x_2) = x_1 \oplus x_2$. The functional abstraction of x_1 yields the following partially specified function:

$$f_{U(x_1)} = \{0, \bar{x}_1, x_1, 1\} \oplus \{x_2\} = \{x_2, x_1 \odot x_2, x_1 \oplus x_2, \bar{x}_2\}$$

Consider next the abstraction of x_1 from the four-NAND implementation of XOR shown in Figure 3-1. The resulting partially specified functions at the intermediate nodes *g*, *v* and *w* as well as that at the output node *f* are easily shown to be:

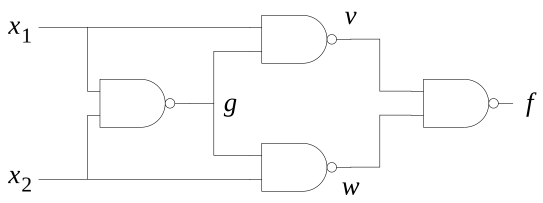


Figure 3-1: Four-NAND Circuit Implementation of XOR

	$U(x_1)$			
	0	$\bar{x}_1$	x_1	1
$g = \overline{x_1 x_2}$	1	$x_1 \vee \bar{x}_2$	$\bar{x}_1 \vee \bar{x}_2$	$\bar{x}_2$
$v = \overline{x_1 g}$	1	$x_1 \vee x_2$	$\bar{x}_1 \vee x_2$	x_2
$w = \overline{x_2 g}$	$\bar{x}_2$	$\bar{x}_1 \vee \bar{x}_2$	$x_1 \vee \bar{x}_2$	1
$f = \overline{vw}$	x_2	$x_1 \odot x_2$	$x_1 \oplus x_2$	$\bar{x}_2$

Figure 3-2: Solution of Example 3.3 taking signal correlations into account

$$\begin{aligned} g(U(x_1), x_2) &= \{\bar{x}_2, \bar{x}_1 \vee \bar{x}_2, x_1 \vee \bar{x}_2, 1\} \\ v(U(x_1), x_2) &= \{0, \bar{x}_1 x_2, x_1 x_2, \bar{x}_1, x_1, x_2, \bar{x}_1 \vee x_2, x_1 \vee x_2, 1\} \\ w(U(x_1), x_2) &= \{\bar{x}_2, \bar{x}_1 \vee \bar{x}_2, x_1 \vee \bar{x}_2, 1\} \\ f(U(x_1), x_2) &= \mathcal{U}_2(x_1, x_2) \end{aligned}$$

Thus, abstraction of x_1 from the circuit implementation of the XOR function yields the maximally uncertain function of two variables indicating a complete loss of information. □

The above example illustrates an undesirable characteristic of uncertainty models, namely the unavoidable introduction of *apparent* uncertainty when *correlations* that exist among the model variables are ignored. As shown in Figure 3-2, the correct answer for this example can be obtained when such correlations are properly taken into account. To accomplish that, however, requires that we distinguish among individual members of sets and amounts to an enumerative "case analysis." Since the abstraction of *m* variables gives rise to 4^m independent cases, such an approach is infeasible except for small values of *m*. The only other way of eliminating apparent uncertainty is to express the model only in terms of independent variables such as those corresponding to primary inputs or head lines [1, p. 208] in the circuit implementation. Unfortunately, for many functions such representations are impractical since their sizes can be exponential in the number of independent variables. Realistically, therefore, the injection of apparent uncertainty must be accepted as a necessary consequence of abstraction, and the result of abstraction should be viewed as a conservative approximation to the sought-after partially specified function.

4 Function Intervals

Function sets represent uncertainty *explicitly* by enumerating all the completely specified functions that are obtained as a result of abstraction. Their importance, thus, is primarily theoretical since they may become too large for any practical application, even when apparent uncertainty is eliminated. The only recourse for containing the inevitable exponential growth in set sizes is to bound such sets by sets that can be represented *implicitly*, i.e. without enumeration. Intervals in the function space $\mathcal{U}_n$ are function sets that can be described compactly by specifying two distinguished members. In this section we define function intervals, derive some of their properties, and show how they can be used for functional abstraction.

Definition 4.1 (Function Interval [5, p. 45]) *The partially specified function $F(X)$ is a function interval if it can be expressed as*

$$F(X) = \{f(X) | l(X) \le f(X) \le u(X)\}$$

where $l(X)$ and $u(X)$ are two n-variable Boolean functions such that $l(X) \le u(X)$. A function interval $F(X)$ can be viewed as a mapping $\mathcal{B}^n \to S$, where $S = \{[0,0],[0,1],[1,1]\}$ is the set of intervals on $\mathcal{B}$, and can thus be equivalently expressed as $F(X) = [l(X), u(X)]$. □

Several examples of function intervals in $\mathcal{U}_2(x_1, x_2)$ are shown in Figure 4-1.

Function intervals have many useful properties most of which derive from the partial ordering (inclusion) relation "≤" and its various equivalents [5, p. 28]. In particular, given $F(X) = [l(X), u(X)]$ and $f(X) \in F(X)$, we can establish the following two sets of equivalent identities:

$$\begin{array}{ll} l(X) \le f(X) & f(X) \le u(X) \\ l(X) \wedge f(X) = l(X) & u(X) \vee f(X) = u(X) \\ l(X) \wedge f'(X) = 0 & u(X) \vee f'(X) = 1 \\ f'(X) \le l'(X) & u'(X) \le f'(X) \\ l'(X) \vee f'(X) = l'(X) & u'(X) \wedge f'(X) = u'(X) \\ l'(X) \vee f(X) = 1 & u'(X) \wedge f(X) = 0 \end{array} \tag{4.1}$$

An immediate consequence of these identities is that $l(X)$ and $u(X)$ are the glb and lub of $F(X)$, i.e.

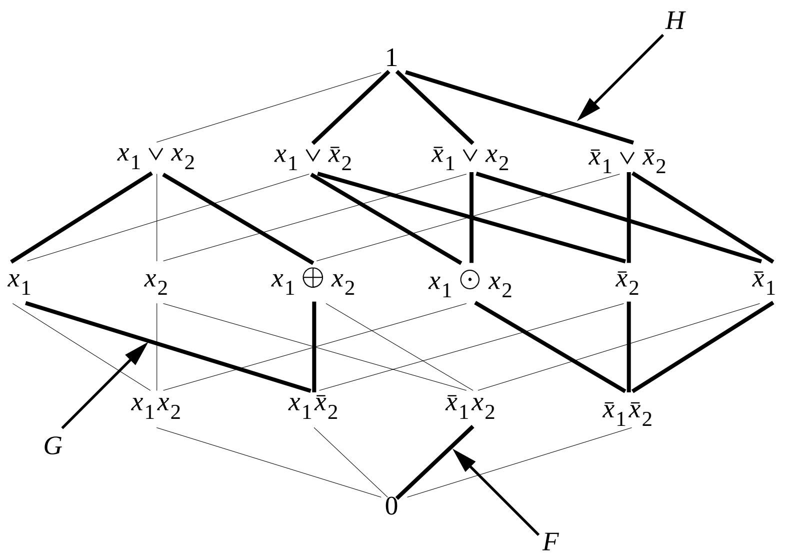
$$\lfloor F(X) \rfloor = l(X) \tag{4.2}$$

$$\lceil F(X) \rceil = u(X) \tag{4.3}$$

Thus, if $F(X)$ is a function interval we can denote it most succinctly as

$$F(X) = [\lfloor F(X) \rfloor, \lceil F(X) \rceil] \tag{4.4}$$

The size of a function interval $F(X)$ is 2^m where *m* is the number of minterms for which the function value is $[0, 1]$. For each one of those *m* combinations, the glb $\lfloor F(X) \rfloor$ is 0 whereas the lub $\lceil F(X) \rceil$ is 1. Thus, *m* is equal to the number of minterms for which the function $\lfloor F(X) \rfloor' \lceil F(X) \rceil$ is equal to 1 yielding the following expression for the size of $F(X)$:



x_1	x_2	F	G	H
0	0	[0, 0]	[0, 0]	[1, 1]
0	1	[0, 1]	[0, 1]	[0, 1]
1	0	[0, 0]	[1, 1]	[0, 1]
1	1	[0, 0]	[0, 1]	[0, 1]

(b)

Figure 4-1: (a) Sixteen-element lattice $\mathcal{U}_2(x_1, x_2)$ of two-variable Boolean functions. The highlighted lines correspond to the three intervals $F = [0, \bar{x}_1x_2]$, $G = [x_1\bar{x}_2, x_1 \vee x_2]$ and $H = [\bar{x}_1\bar{x}_2, 1]$. (b) Truth tables for highlighted intervals.

$$|F(X)| = 2^{\|\lfloor F(X) \rfloor ' \lceil F(X) \rceil\|} \tag{4.5}$$

where $\|f(X)\|$ is the number of minterms for which a Boolean function $f(X)$ is equal to 1. Using (4.5) it is easy to show that the size of the completely unspecified n-variable function $[0, 1]$ is 2^{2^n}, whereas that of any completely specified function $[f(X), f(X)]$ is 1. It is worth noting that, while identical in appearance, the interval $[0, 1]$ in the n-variable function space is distinct from the interval $[0, 1]$ in the 2-element Boolean algebra $\mathcal{B}$. The interval bounds in the former case represent the n-variable constant functions 0 (*inconsistency*) and 1 (*tautology;*) in the latter case the bounds denote the constant elements of $\mathcal{B}$. Furthermore, the partially specified n-variable functions $[0, 1]$ and $\{0, 1\}$ are different; the latter consists of just two elements, namely the constant functions 0 and 1, whereas $[0, 1]$ denotes the entire function space $\mathcal{U}_n$.

The interval operator introduced next is useful in "converting" unrestricted partially specified functions into intervals. It can be viewed as a *type cast* [15] that converts objects of type "function set" to objects of type "function interval".

Definition 4.2 (Interval Operator [ ]) *The interval operator* $[\ \ ]$ *is a unary operator that returns the smallest function interval containing a given partially specified function. It is defined by the formula*:

$$[F(X)] \equiv [\lfloor F(X) \rfloor, \lceil F(X) \rceil] \tag{4.6}$$

Note that F itself is an interval whenever $\lfloor F \rfloor \in F$ *and* $\lceil F \rceil \in F$*; in such a case* $[F] = F$. □

Example 4.1 Let $F(x_1, x_2) = \{x_1x_2, \bar{x}_2\}$. Then,

$$[F] = [\lfloor F \rfloor, \lceil F \rceil] = [0, x_1 \vee \bar{x}_2] = \{0, \bar{x}_1\bar{x}_2, x_1\bar{x}_2, x_1x_2, x_1, \bar{x}_2, x_1 \odot x_2, x_1 \vee \bar{x}_2\} \qquad \square$$

4.1 Operations on Function Intervals

The main advantage of function intervals over unrestricted partially specified functions is that they do not require an explicit enumeration of their member functions. Instead, the elements of a function interval are implicitly defined by the interval's glb and lub along with the partial ordering relation "≤". This advantage may not, in general, be preserved when function intervals are combined by set operators. Specifically, the union, intersection, and difference of two function intervals are not necessarily function intervals. On the other hand, applying Boolean operators to function intervals will always yield function intervals. This property allows us to develop an *interval Boolean algebra* that can be used to manipulate and simplify expressions involving function intervals.

The extension of the three basic Boolean operators NOT, AND, and OR to function intervals is provided by the following theorem.

Theorem 4.1 (Boolean Operations on Function Intervals) *Let $F(X)$ and $G(X)$ be two function intervals. Then $F'(X)$, $F(X) \wedge G(X)$, and $F(X) \vee G(X)$ are also function intervals that are given by the following identities:*

$$F'(X) = [\lceil F \rceil', \lfloor F \rfloor'] \tag{4.7}$$

$$F(X) \wedge G(X) = [\lfloor F \rfloor \wedge \lfloor G \rfloor, \lceil F \rceil \wedge \lceil G \rceil] \tag{4.8}$$

$$F(X) \vee G(X) = [\lfloor F \rfloor \vee \lfloor G \rfloor, \lceil F \rceil \vee \lceil G \rceil] \tag{4.9}$$

Proof: We demonstrate the proof procedure for (4.7); the other two identities can be proved similarly.

$$H(X) \equiv F'(X) = \bigcup_{f(X) \in F(X)} \{f'(X)\} \qquad \text{from (2.2) (functional composition rule \#1)}$$

$$\begin{aligned}
\lfloor H(X) \rfloor &= \bigwedge_{h(X) \in H(X)} h(X) = \bigwedge_{f(X) \in F(X)} f'(X) && \text{from (2.4) (definition of glb)}\\
&= \lceil F(X) \rceil' \wedge \bigwedge_{\substack{f(X) \in F(X)\\ f(X) \neq \lceil F(X) \rceil}} f'(X) && \text{factor out lub}\\
&= \lceil F(X) \rceil' && \text{from (4.1) (inclusion relation)}
\end{aligned}$$

$$\begin{aligned}
\lceil H(X) \rceil &= \bigvee_{h(X) \in H(X)} h(X) = \bigvee_{f(X) \in F(X)} f'(X) && \text{from (2.4) (definition of lub)}\\
&= \lfloor F(X) \rfloor' \vee \bigvee_{\substack{f(X) \in F(X)\\ f(X) \neq \lfloor F(X) \rfloor}} f'(X) && \text{factor out glb}\\
&= \lfloor F(X) \rfloor' && \text{from (4.1) (inclusion relation)}
\end{aligned}$$

Table 4-1: Some properties of interval Boolean algebra (F, G, and H are assumed to be function intervals)

Commutative	$F \vee G = G \vee F$	$F \wedge G = G \wedge F$
Distributive	$F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$	$F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$
Identities	$[0, 0] \vee F = F$	$[1, 1] \wedge F = F$
Associative	$F \vee (G \vee H) = (F \vee G) \vee H$	$F \wedge (G \wedge H) = (F \wedge G) \wedge H$
Idempotence	$F \vee F = F$	$F \wedge F = F$
Null Elements	$F \vee [1, 1] = [1, 1]$	$F \wedge [0, 0] = [0, 0]$
Absorption	$F \vee (F \wedge G) = F$	$F \wedge (F \vee G) = F$
Involution	$(F')' = F$	
De Morgan's	$(F \vee G)' = F' \wedge G'$	$(F \wedge G)' = F' \vee G'$
Complement	$F \vee F' = [\lfloor F \rfloor \vee \lceil F \rceil', 1]$	$F \wedge F' = [0, \lfloor F \rfloor' \wedge \lceil F \rceil]$

Thus, $\lfloor H(X) \rfloor \in H(X)$ and $\lceil H(X) \rceil \in H(X)$, i.e. $H(X)$ is an interval (see Definition 4.2). This immediately leads to $F'(X) = [\lceil F(X) \rceil', \lfloor F(X) \rfloor']$. □

The identities in (4.1) allow us to derive many useful properties that facilitate the algebraic manipulation of expressions involving function intervals. A few of these properties are listed in Table 4-1. It is interesting to note that function intervals obey most, but not all, of the laws of Boolean algebra. In particular, they do not obey the complement laws or any properties derived from them. Thus, unlike the case for completely specified functions, the conjunction and disjunction of a function interval F and its complement do not yield, respectively, $[0, 0]$ and $[1, 1]$. Instead, there is some residual uncertainty whose size is equal to $|F|$.

Equality of function intervals derives from their interpretation as function sets. Thus, the function intervals $F(X)$ and $G(X)$ are equal if and only if $F(X) \subseteq G(X)$ and $G(X) \subseteq F(X)$. Equivalently:

$$(F = G) \Leftrightarrow (\lfloor F \rfloor = \lfloor G \rfloor) \wedge (\lceil F \rceil = \lceil G \rceil) \tag{4.10}$$

i.e., $F(X)$ and $G(X)$ are identical if their respective bounds are equal.

Example 4.2 Assume that $x_1, x_2, x_3 \in \{0, 1\}$ are Boolean variables and that F and G are function intervals. The following equations illustrate the application of interval Boolean algebra.

$$[x_1 x_2 \vee x_2 x_3, x_1 x_2 \vee x_3]' = [\bar{x}_1 \bar{x}_3 \vee \bar{x}_2 \bar{x}_3, \bar{x}_1 \bar{x}_3 \vee \bar{x}_2]$$

$$[\bar{x}_1 x_2 \vee x_1 \bar{x}_2, x_1 \vee x_2] \wedge [x_1 x_2, \bar{x}_1 \bar{x}_2 \vee x_1 x_2] = [0, x_1 x_2]$$

$$[\bar{x}_1 x_2 \vee x_1 \bar{x}_2, x_1 \vee x_2] \vee [x_1 x_2, \bar{x}_1 \bar{x}_2 \vee x_1 x_2] = [x_1 \vee x_2, 1]$$

$$F \oplus G = F'G \vee FG' = [\lceil F \rceil' \lfloor G \rfloor \vee \lfloor F \rfloor \lceil G \rceil', \lfloor F \rfloor' \lceil G \rceil \vee \lceil F \rceil \lfloor G \rfloor'] \qquad \square$$

It is useful to note that the above 2-valued interval Boolean algebra is isomorphic to the ternary algebra commonly used in logic simulators. Specifically, the three intervals on $\{0, 1\}$ are renamed 0, 1, and *U*, are considered to be totally ordered according to $[0, 0] \le [0, 1] \le [1, 1]$ and are assumed to obey the following truth tables:

F	F'
[0, 0]	[1, 1]
[0, 1]	[0, 1]
[1, 1]	[0, 0]

$\vee$	[0, 0]	[0, 1]	[1, 1]
[0, 0]	[0, 0]	[0, 1]	[1, 1]
[0, 1]	[0, 1]	[0, 1]	[1, 1]
[1, 1]	[1, 1]	[1, 1]	[1, 1]

$\wedge$	[0, 0]	[0, 1]	[1, 1]
[0, 0]	[0, 0]	[0, 0]	[0, 0]
[0, 1]	[0, 0]	[0, 1]	[0, 1]
[1, 1]	[0, 0]	[0, 1]	[1, 1]

As mentioned earlier, set operations on function intervals do not, in general, yield function intervals. In such cases, the interval operator $[\ \ ]$ can be used to convert the result of a set operation to the smallest enclosing function interval. The function interval containing the union of two function sets is particularly useful and we define a special operator for it as follows:

Definition 4.3 (Expand Operator) *The expand operator for partially specified functions is a binary operator defined by the formula*:

$$F(X) \Diamond G(X) \equiv [F(X) \cup G(X)]$$

where $F(X)$ and $G(X)$ are arbitrary partially specified functions (i.e. not necessarily function intervals.) $\square$

The next theorem establishes the relation between the bounds of the interval produced by the expand operator and the corresponding bounds of its two arguments.

Theorem 4.2 $F(X) \Diamond G(X) = [\lfloor F \rfloor \wedge \lfloor G \rfloor, \lceil F \rceil \vee \lceil G \rceil]$

Proof:

$$\begin{aligned}
F \Diamond G &= [F \cup G] && \text{from Definition 4.3}\\
&= [\lfloor F \cup G \rfloor, \lceil F \cup G \rceil] && \text{from (4.6) (definition of interval operator)}\\
&= \left\lfloor \bigwedge_{h \in F \cup G} h, \bigvee_{h \in F \cup G} h \right\rfloor && \text{from (2.4) (definition of glb and lub)}\\
&= \left\lfloor \bigwedge_{f \in F} f \wedge \bigwedge_{g \in G} g, \bigvee_{f \in F} f \vee \bigvee_{g \in G} g \right\rfloor &&\\
&= [\lfloor F \rfloor \wedge \lfloor G \rfloor, \lceil F \rceil \vee \lceil G \rceil] && \text{from (2.4) (definition of glb and lub)} \qquad \square
\end{aligned}$$

Example 4.3 Let $F(x_1, x_2) = [x_1 \oplus x_2, x_1 \vee x_2]$ and $G(x_1, x_2) = [x_1 x_2, x_1 \odot x_2]$. Then,

$$F \cup G = \{x_1 \oplus x_2, x_1 \vee x_2\} \cup \{x_1 x_2, x_1 \odot x_2\} = \{x_1 x_2, x_1 \odot x_2, x_1 \oplus x_2, x_1 \vee x_2\}$$

$$F \Diamond G = [(x_1 \oplus x_2) \wedge (x_1 x_2), (x_1 \vee x_2) \vee (x_1 \odot x_2)] = [0, 1] \qquad \square$$

4.2 Abstraction of Variables from Function Intervals

Another operation on a function interval that may not necessarily yield a function interval is functional abstraction. We therefore extend Definition 3.2 so that the result of functional abstraction from a function interval is another function interval.

Definition 4.4 (*Conservative* Functional Abstraction) *The conservative functional abstraction of a partially specified n-variable Boolean function $F(X)$ with respect to a variable x_i is a function interval denoted by $F_{[U(x_i)]}$ and defined according to:*

$$F(X)_{[U(x_i)]} = [F(X)_{U(x_i)}] \tag{4.11}$$

In other words, the conservative functional abstraction of $F(X)$ with respect to x_i is the smallest function interval containing the (exact) functional abstraction of $F(X)$ with respect to x_i. □

The conservative functional abstraction has a particularly simple form in terms of the cofactors of the function bounds. To derive this form, it is convenient first to establish the following two identities:

Lemma 4.3 *The glb/lub and cofactor operators commute, i.e.* $\lfloor F_{x_i} \rfloor = \lfloor F \rfloor_{x_i}$ and $\lceil F_{x_i} \rceil = \lceil F \rceil_{x_i}$.

Proof:

$$F_{x_i} = \bigcup_{f \in F} f_{x_i} \qquad \text{from (2.3)}$$

$$\lfloor F_{x_i} \rfloor = \bigwedge_{f \in F} f_{x_i} \qquad \text{from (2.4)}$$

$$\lfloor F \rfloor = \bigwedge_{f \in F} f \qquad \text{from (2.4)}$$

$$\lfloor F \rfloor_{x_i} = \left(\bigwedge_{f \in F} f \right)_{x_i} = \bigwedge_{f \in F} f_{x_i} = \lfloor F_{x_i} \rfloor$$

The other identity is proved similarly. □

Theorem 4.4 *The conservative abstraction of $F(X)$ with respect to x_i is given by the following expression:*

$$F(X)_{[U(x_i]} = [\lfloor F \rfloor_{\bar{x}_i} \wedge \lfloor F \rfloor_{x_i}, \lceil F \rceil_{\bar{x}_i} \vee \lceil F \rceil_{x_i}] \tag{4.12}$$

Proof:

$$
\begin{aligned}
[F_{U(x_i)}] &= [F_{\bar{x}_i} \cup F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \cup F(X) \cup F_{x_i}] && \text{from (4.11) and (3.1)} \\
&= [\lfloor F_{\bar{x}_i} \cup F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \cup F(X) \cup F_{x_i} \rfloor, \\
&\qquad \lceil F_{\bar{x}_i} \cup F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \cup F(X) \cup F_{x_i} \rceil] && \text{from (4.6) (interval operator)} \\
&= [\lfloor F_{\bar{x}_i} \rfloor \wedge \lfloor F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \rfloor \wedge \lfloor F(X) \rfloor \wedge \lfloor F_{x_i} \rfloor, \\
&\qquad \lceil F_{\bar{x}_i} \rceil \vee \lceil F(x_1, \ldots, \bar{x}_i, \ldots, x_n) \rceil \vee \lceil F(X) \rceil \vee \lceil F_{x_i} \rceil] && \text{from (2.4)} \\
&= [\lfloor F_{\bar{x}_i} \rfloor \wedge \lfloor F_{x_i} \rfloor, \lceil F_{\bar{x}_i} \rceil \vee \lceil F_{x_i} \rceil] && \text{by perfect induction} \\
&= [\lfloor F \rfloor_{\bar{x}_i} \wedge \lfloor F \rfloor_{x_i}, \lceil F \rceil_{\bar{x}_i} \vee \lceil F \rceil_{x_i}] && \text{from Lemma 4.3} \qquad \square
\end{aligned}
$$

It is interesting to note that the conservative abstraction of $F(X)$ with respect to x_i is the interval whose glb is the *consensus* operator $C_{x_i}\lfloor F \rfloor$ and whose lub is the *smoothing* operator $S_{x_i}\lceil F \rceil$ [12]. Brown [5, pp. 106-110] refers to these bounds, respectively, as the conjunctive and disjunctive eliminants, and views what we have called conservative abstraction as the "elimination" of x_i from the function interval $F(X)$. Finally, these two bounds can be obtained by universal and existential quantification of x_i from $\lfloor F \rfloor$ and $\lceil F \rceil$:

$$F(X)_{[U(x_i)]} = [\forall x_i \cdot \lfloor F(X) \rfloor, \exists x_i \cdot \lceil F(X) \rceil] \tag{4.13}$$

This last form is particularly helpful because it demonstrates that abstraction is essentially a process of quantification.

Example 4.4 Let $f(x_1, x_2) = x_1 x_2$ and $g(x_1, x_2) = x_1 \oplus x_2$. Then $f_{\bar{x}_1} = 0$, $f_{x_1} = x_2$, $g_{\bar{x}_1} = x_2$, $g_{x_1} = \bar{x}_2$ and the following functional abstractions are easily established:

$$F(x_1, x_2) \equiv f_{[U(x_1)]} = [f_{\bar{x}_1} f_{x_1}, f_{\bar{x}_1} \vee f_{x_1}] = [0, x_2] \tag{4.14}$$

$$f_{[U(x_1, x_2)]} = F_{[U(x_2)]} = \left[0\big|_{x_2 = 0} \wedge 0\big|_{x_2 = 1}, x_2\big|_{x_2 = 0} \vee x_2\big|_{x_2 = 1} \right] = [0, 1] \tag{4.15}$$

$$g_{[U(x_1)]} = [g_{\bar{x}_1} g_{x_1}, g_{\bar{x}_1} \vee g_{x_1}] = [x_2 \bar{x}_2, x_2 \vee \bar{x}_2] = [0, 1] \tag{4.16}$$

Recalling (2.1), note that the conservative abstraction in (4.14) does not involve additional pessimism. On the other hand, the conservative abstractions in (4.15) and (4.16) are more pessimistic than their "exact" counterparts in examples 3.2 and 3.3.□

5 Conclusions and Future Work

In this report we sought to understand the transformation to a Boolean function that results from functionally abstracting its dependence on some of its variables. We argued for the use of function sets as a basis for capturing functional uncertainty, and indicated how function intervals serve as a practical approximation to unrestricted function sets. We also developed a complete interval Boolean algebra that helps us in the practical manipulation of function intervals.

The purpose of this report was to lay the algebraic foundation for both functional abstraction and partial specification of Boolean functions. In related work, we describe a comprehensive model of the dynamic behavior of logic gate circuits that builds on this foundation [14]. We are also developing a general model for the timing of logic gate circuits that is derived through careful functional abstraction. Unlike most existing timing models, the proposed model handles state-dependent component and wire delays and replaces the plethora of local sensitization criteria with a single universal criterion that captures the functional dependence of component delays and signal arrival times.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*. Electrical Engineering, Communications, and Signal Processing, ed. R.L. Pickholtz. 1990, Computer Science Press.

[2] R. K. Brayton, J. D. Cohen, G. D. Hachtel, B. M. Trager and D. Y. Y. Yun, "Fast Recursive Boolean Function Manipulation," *in Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 58-62, 1982, Rome, Italy.

[3] R. K. Brayton and F. Somenzi, "An Exact Minimizer for Boolean Relations," *in Digest of IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 316-319, 1989, Santa Clara, California.

[4] M. A. Breuer, "A Note on Three-Valued Logic Simulation," IEEE Transaction on Computers, vol. C-21, pp. 399-402, April 1972.

[5] F. M. Brown, *Boolean Reasoning*. ed. J. Allen. 1990, Kluwer Academic Publishers.

[6] C. L. Harkness, *An Approach to Uncertainty in VLSI Design*, Ph.D. Thesis, Brown University, 1991.

[7] C. L. Harkness and D. P. Lopresti, "Interval Methods for Modeling Uncertainty in RC Timing Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 11, pp. 1388-1401, November 1992.

[8] J. P. Hayes, "Uncertainty, Energy, and Multiple-Valued Logics," *IEEE Transactions on Computes*, vol. C-35, no. 2, pp. 107-114, February 1986.

[9] J. P. Hayes, "Digital Simulation with Multiple Logic Values," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-5, no. 2, pp. 274-283, April 1986.

[10] S. C. Kleene, *Introduction to Mathematics*. 1952, Princeton, N.J.: D. Van Nostrand, Inc.

[11] Z. Kohavi, *Switching and Finite Automata Theory.* Second ed. Computer Science Series, 1978, McGraw-Hill.

[12] P. C. McGeer, *On the Interaction of Functional and Timing Behavior of Combinational Logic Circuits*, Ph.D. Thesis, University of California, Berkeley, November 1989.

[13] D. E. Muller, "Treatment of Transition Signals in Electronic Switching Circuits by Algebraic Methods," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 401, 1959.

[14] K. A. Sakallah, "Dynamic Modeling of Logic Gate Circuits," Technical Report CSE-TR-253-95, The University of Michigan, July 1995.

[15] B. Stroustrup, *The C++ Programming Language*. Addison-Wesley Series in Computer Science, ed. M.A. Harrison. 1986, Addison-Wesley.

[16] C. A. Zukowski, *The Bounding Approach to VLSI Circuit Simulation*. The Kluwer International Series in Engineering and Computer Science, ed. J. Allen. 1986, Kluwer Academic Publishers.