CONSISTENCY AND OBSERVABILITY INVARIANCE IN MULTI-LEVEL LOGIC SYNTHESIS

by

Patrick C. McGeer and Robert K. Brayton

Memorandum No. UCB/ERL M89/88

21 July 1989

CONSISTENCY AND OBSERVABILITY INVARIANCE IN MULTI-LEVEL LOGIC SYNTHESIS

by

Patrick C. McGeer and Robert K. Brayton

Memorandum No. UCB/ERL M89/88

21 July 1989

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CONSISTENCY AND OBSERVABILITY INVARIANCE IN MULTI-LEVEL LOGIC SYNTHESIS

by

Patrick C. McGeer and Robert K. Brayton

Memorandum No. UCB/ERL M89/88

21 July 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Consistency and Observability Invariance in Multi-Level Logic Synthesis

Patrick McGeer, Robert K. Brayton,
Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley,
Berkeley, CA, 94720.*

May 4, 1989

Abstract

In this paper, we depict an m-function network on n primary inputs as forming an $n+m$ dimensional space. In this space there are points that can never occur due to mutual dependencies among the functions. This set has been called the the satisfiability don't-care (SDC) set of the network and can be viewed as a function over the extended space. We show that the size of the SDC is uniquely determined by the number of primary inputs to the network and the number of functions over it. We demonstrate a sharp criterion for determining which transformations of the network preserve the SDC, and show that most of the operations of the MIS-II synthesis system preserve the SDC. This has importance for "implications" which are used in a number of network manipulations. This analysis also clarifies how other operations change the SDC, but in very predictable ways. Finally, we show that most of the algebraic and some Boolean operations commonly used in logic synthesis preserve the testability of all but a single node in a network. An interesting example is algebraic division (or resubstitution) of one node into another. The testability of only the divisor is possibly changed. Surprisingly the testability of none of the divisor's inputs is changed.

1 Introduction

It has long been known that the use of various consistency and observability conditions in networks are useful in multi-level logic synthesis[2]. However, the use of these conditions has historically been impeded by the feeling that most transformations of the network modified these conditions, and so re-computation of the conditions was essential after each transformation. Since re-computation is potentially expensive, interest has historically centered on incremental updates.

The theme of this paper is to explore the consistency and observability conditions. The consistency conditions most often appear as the so-called *satisfiability don't-care set* or SDC. These arise because we depict an m-function network on n primary inputs as forming an $n+m$ dimensional space. In this space there are points that can never occur due to mutual dependencies among the functions. Occasionally a subset of the SDC is referred to by the name of a *forcing* or $\mathcal{F}-$ set [16] or implications. These have been found increasingly useful in many applications [9]. Observability conditions are related to testability, and have appeared as don't-cares in the form of the *fanout don't care set*[2][8].

In this paper we examine and answer the following questions:

*This research supported by the Semiconductor Research Corporation under Contract 87-DC-008

- In transforming a logic network using the various operations of logic optimization and synthesis, when and how are the consistency conditions changed? In particular, can we identify a set of transformations that either leave the consistency conditions invarient. or allow them to be easily updated in a regular incremental way?

- Given that the SDC is often used as a don't care set in logic optimization, what can we say about its size or its inverse? The SDC has been used for verification [10], without notable success. Can we explain this in terms of the size of the SDC?

- Are the consistency and observability conditions related and if so, in what way? Under what conditions on a transformation of the network can we guarantee that the observability of a node (and hence the conditions under which it is testable) is unchanged?

A principle result of this paper is that the consistency and observability conditions are related and can both be derived using the SDC or a subset. This leads to a precise statement under which the testability of a signal is preserved, and provides insight into an increasingly important issue in synthesis for testability - the generation and maintenance of a test set throughout the logic synthesis process[7].

2 Boolean Spaces, Don't Care Sets and Mapping Between Spaces

In general, a boolean function $f_j(v)$ is defined on the space of primary inputs, conventionally denoted as $\mathbf{B}^n$. Large functions, however, are often better (and sometimes only) realized as part of larger networks. These networks involve intermediate variables, and hence lead to the definition of the function on a space of combined intermediate variables and primary inputs; this space is conventionally represented as $\mathbf{B}^{n+m}$. This description is incomplete, however, inasmuch as mutual dependencies among the internal variables, and dependencies of the internal variables upon the primary inputs, guarantee that a large set of points on this extended space can never occur. A set of such points are collectively called the *Satisfiability Don't Care* (SDC) set, and, where the internal function f_j is represented by variable y_j, is defined as the set of the points in $\mathbf{B}^{n+m}$covered by the function:

$$\sum_j f_j \oplus y_j$$

In the sequel, we demonstrate that *every* point in $\mathbf{B}^{n+m}$which can't occur is covered by the SDC set. In this paper we demonstrate the exact size of the SDC set on *any* network of n primary inputs and m internal variables. We give an alternative view of forcing sets based on this kerneling process, discuss the effect of network transformations on this space, and in particular characterize the transformations which leave the homomorphism invariant.

3 The Boolean n-space and the Boolean n-cube

In general, one can derive a geometric picture of the boolean n-space as an n-dimensional cube as follows. Consider the n-dimensional Cartesian co-ordinate system. Since each variable can only assume the values 0 and 1, we are left with the immediate conclusion that the dimension of the space represented by the variable x can be restricted by planes at $x = 0$ and $x = 1$. Once this has been done in every dimension, the resulting object is an n-dimensional cube.

Now, one can represent a function on this cube in a variety of ways; in particular, one can consider a function as an arbitrary set of vertices on this cube, or as a partition of the vertices of

the cube into three sets, an *off-set*, an *on-set*, and an *external don't-care-set*. This formulation is awkward, however, for multiple-output functions. For such functions (say, in particular, functions with m outputs and n inputs), one can consider the space as a 2^{n+m}-dimensional cube in the obvious way. In this manner the extended space $\mathbf{B}^{n+m}$arises from the primary input $\mathbf{B}^n$.

3.1 Notation

There is a great deal of notation in this paper, largely because the focus continually shifts from the primary input space $\mathbf{B}^n$to the extended space $\mathbf{B}^{n+m}$. So that the inherent chaos may become some form of order, we detail the notation here.

Functions over $\mathbf{B}^{n+m}$will be denoted f_j, and associated with each f_j is a variable y_j. Further, each such f_j is a representation of a function over $\mathbf{B}^n$, the *global image* of y_j, $\tilde{f}_j$; in the sequel, we formally show the correspondence of f_j and $\tilde{f}_j$. If we refer to a function f without subscript, this will correspond to the variable y.

Cubes as products of literals have both a geometric and a set-theoretic interpretation. In the geometric interpretation, if a literal x appears in a cube w, then w lies wholly on the face x, and we write $w \subseteq x$. In the set theoretic-interpretation, where a cube is a set of literals, we have that $x \in w$. Further, the union and intersection operations have precisely opposite semantics in the two interpretations. Obviously this is confusing, and the only way to establish clear semantics is to pick one of the two interpretations and stick to it. In this paper, we adopt and maintain the geometric interpretation throughout.

If a cube is fully-specified, it is a single point, also called a *minterm* or a *vertex*. In the remainder of this paper, an arbitrary cube over the primary input space will be referred to by the symbol v; a cube over the extended space will be referred to by the symbol w. Occasionally w will be written uv; u will be understood to stand for the intermediate or function variables specified in w.

A table summarizing this notation will be given after the discussion of mappings between the spaces.

3.2 Maps Between Spaces

There are many maps from $\mathbf{B}^{n+m}$onto $\mathbf{B}^n$. The most obvious is the trivial projection operator $\mathcal{P} : \mathbf{B}^{n+m} \rightarrow \mathbf{B}^n$, which may be informally defined by partitioning each vertex of $\mathbf{B}^{n+m}$ into its primary input co-ordinates v and intermediate variable co-ordinates u. From this, we define:

$$\mathcal{P}(uv) = v$$

Now, $\mathcal{P}$ is one of the most trivial mapping operators from $\mathbf{B}^{n+m}$ onto $\mathbf{B}^n$.

The evaluation map $\theta : \mathbf{B}^n \rightarrow \mathbf{B}^{n+m}$ is defined in the obvious way. For each function f_j, we have (assuming f_j is completely specified) either $\tilde{f}_j(v) = 1$ or $\tilde{f}_j(v) = 0$. From this observation, we obtain $w = \theta(v)$, by writing $w = u_1 ... u_m v$, where $u_j = y_j$ if $\tilde{f}_j(v) = 1$, and $\overline{y_j}$ if $\tilde{f}_j(v) = 0$.

Now, it is immediately clear that:

Theorem 3.1 *For each vertex v of $\mathbf{B}^n$, there is exactly one vertex w of $\mathbf{B}^{n+m}$ such that $w = \theta(v)$*

Proof: Follows immediately from the observation that every variable in the extended space is assigned a value by fully specifying the variables in the input space. ∎

Notice immediately that there are 2^n points w of $\mathbf{B}^{n+m}$such that $w = \theta(v)$ for some v. Since these points must all be distinct, it follows that there are precisely $2^{n+m} - 2^n$ points w of $\mathbf{B}^{n+m}$for which there is *no* v such that $w = \theta(v)$. This set takes on added significance due to the next theorem, which identifies such points as the SDC set.

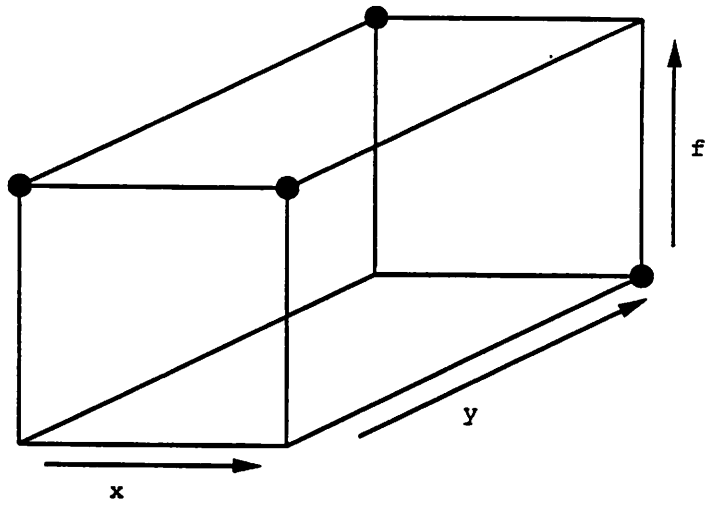


Figure 1: Cube Representation of $f = xy$

Theorem 3.2 $SDC = \{w | w \neq \theta(\mathcal{P}(w))\}$

For this proof, we will be using the trivial projection operator $\mathcal{P} : \mathbf{B}^{n+m} \rightarrow \mathbf{B}^n$, defined above. Note that if $w = \theta(v)$ for any v, then $w = \theta(\mathcal{P}(w))$.

Proof: $w \neq \theta(\mathcal{P}(w))$. Let $v = \mathcal{P}(w)$. Now, clearly there is some f_j such that $\tilde{f}_j(v) = 1$ and $\bar{y}_j$ is a literal of w, or $\tilde{f}_j(v) = 0$ and y_j is a literal of w (if not, $w = \theta(v)$, contradiction). Hence $w \in y_j \oplus f_j$, and so is in the SDC. Similarly, if w is in the SDC, $v = \mathcal{P}(w)$ then there is some f_j such that $w \in y_j$[1], and $\tilde{f}_j(v) = 0$ or $w \in \bar{y}_j$, $\tilde{f}_j(v) = 1$. In either case, $w \neq \theta(v)$. ∎

The cube corresponding to $f = xy$ is shown in figure 1. The darkened circles correspond to the elements of the SDC set. The two key facts to draw from this discussion are first, that the Satisfiability Don't Care set is really not a don't care set at all, since it is trivial (the empty set) when viewed as a set in the primary input space; and second, that the size of the SDC set is uniquely determined by the number of intermediate and output variables in the network. Both of these observations are immediate corollaries of the observation that the SDC set is the projection of the null cube of the primary input space onto the extended space.

Not only is the SDC set large, it is quite evenly spread on the cube. In particular, notice that every vertex in $\overline{SDC}$ has at least m neighbours in SDC[2], and at most n in $\overline{SDC}$. This fact gives us an intuitive feel for the well-known observation that the inverse of the SDC set is hard to write down: every vertex is on the border of the set.

This picture of the extended space and the SDC set will help us understand the effect of transformations that we make on a network in optimizing it. First, we need to discuss cofactors and don't-care sets on a network.

4 Don't Care Sets on the Extended Space

In the preceding discussion, it was assumed that the functions over the primary input space were completely specified. To some large extent this is valid; every network eventually realized is fully

[1] i.e. y_j appears in w

[2] for any $w \in \overline{SDC}$, simply hold the input part fixed and change an occurence of any y_j to $\bar{y}_j$, or vice-versa

Summary of Notation	
Notation	Definition
$\mathbf{B}^n$	Space of primary inputs to the network
v	Generic cube over $\mathbf{B}^n$
$\mathbf{B}^{n+m}$	Space of all variables in the network
w	Generic cube over $\mathbf{B}^{n+m}$(sometimes written vu)
y_j	A node in the network
f_j	Local function associated with node y_j over $\mathbf{B}^{n+m}$
$\tilde{f}_j$	Global function (also *global image*) associated with node y_j (function over $\mathbf{B}^n$)
θ	Evaluation map from $\mathbf{B}^n \rightarrow \mathbf{B}^{n+m}$; $\theta(v) \in y_j$ iff $\tilde{f}_j(v) = 1$
SDC	$\sum_j (y_j \oplus f_j)$; equal to the complement of the range of θ

specified in the sense that some value is computed on each output for each primary input combination. However, not all primary input combinations occur, and for those combinations of primary inputs which *do* occur, some primary outputs may not be meaningful. Both these sets of events are conventionally captured in *don't care sets*, which are vectors of the primary input space for which the values of some or all outputs are undefined.[3]

The question is how such combinations can be realized in this picture of a high-dimensional space, where the on- and off-sets of each function are each realized indirectly as points in $\overline{SDC}$ lying on the appropriate $n + m - 1$ dimensional face of the cube.

If we consider the don't cares, we realize that these are vertices for which some or all functions f_j could be either one or zero; further, in every implementation, each such function *will* be one or zero. Hence the obvious conclusion is that these don't-care sets represent cubes in the extended space of which exactly one is in $\overline{SDC}$ set. Hence these sets represent allowable shifts in the SDC set.

The size of the set of points in the extended space represented by each don't-care vertex on the input space is dependendent upon how many functions have a value of don't-care for this input combination. If r functions are don't-cares for some input vertex v, then this represents a total of 2^r vertices on the extended space, of which all but one must lie inside the SDC and of which any one will lie outside the SDC. Note, incidentally, that these points form a cube of size r in the extended space.

Consider, for example, the drawing in figure 2. This is the function $f = xy$, with a don't-care point selected at $\bar{x}y$ on the primary input space. Note this corresponds to two points on the shaded cube; every realization of this function must choose one of these points to lie in $\overline{SDC}$, and another to lie inside the SDC.

5 Cofactors in the Extended Space

The cofactor operator is generally regarded as an evaluation homomorphism; f_x is the function f evaluated on the face $x = 1$.

In the new formalism, however, we can adopt a more geometric picture. In general, since we distinguish between the cofactor of f with respect to x and the cofactor of $\bar{f}$ with respect to x, the set of points f_x can then be regarded as the space where $f = 1$ and $x = 1$ on the extended space; in particular, we can think of the cofactor as the set of points outside the SDC set on the $n + m - 2$

[3] In fact, don't-care sets are incomplete representations of such conditions; for a full discussion of this phenomenon, see [5]

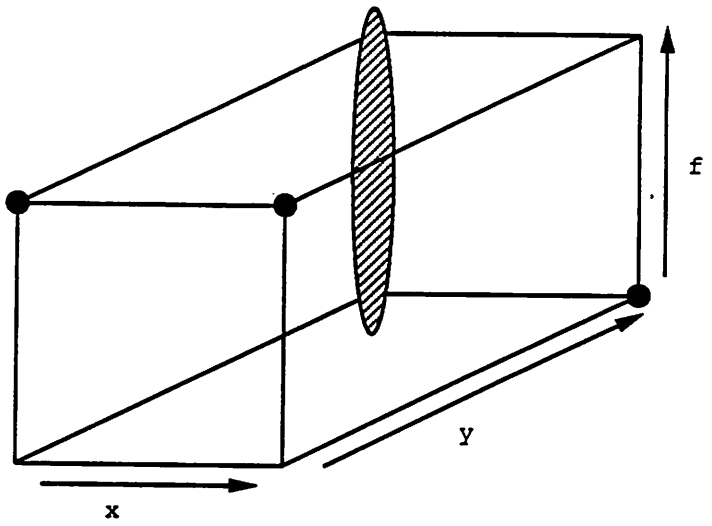


Figure 2: $f = xy$ with a Don't-Care point at $\bar{x}y$

dimensional-cube yx, where, as usual, y is the variable corresponding to f. (Technically, f_x is a function independent of x with the relation $xf = xf_x$ holding).

It is important to prove that this correspondence is exact. For the moment, assume that x is a primary input. We want to show that vertex v of the primary input space $\in \tilde{f}_x$ iff $\theta(xv)$ lies on the face y of the extended space. This gives the precise correspondence, since such vertices are precisely the vertices lying outside the SDC set and on the quarter-space yx.

Theorem 5.1 *For an input variable x:*

$$f_x = y\overline{SDC}_{yx}$$

Proof: $\overline{SDC}$ gives the set of consistent points in the extended space, i.e., the range of θ. Thus:

$$\begin{aligned} f &= y\{w_y | w \in \overline{SDC}\} \\ f_x &= y\{w_{yx} | w \in \overline{SDC}\} \\ &= y\overline{SDC}_{yx} \end{aligned}$$

■

Note that this proof relies on the fact that x was a primary input. If x is not a primary input, we need to modify things somewhat. First, however, we must ask the question of what distinguishes a primary input from an intermediate variable on the extended space. The answer is that an intermediate variable has associated with it a local function, and so a local satisfiability don't-care set. Now, when we cofactor f against a general x, we wish to treat this x as a primary input rather than an internal node; the reason for this is that cofactoring is intended to model the effect of a fault at x, i.e., x set to a value independent of the value on its inputs. Conceptually, we do this by reforming the network with the function associated with x removed, forming the satisfiability don't-care set for this network and applying the theorem of this section. The satisfiability don't-care set for this modified network is:

$$SDC(x) = \sum_{y_j \neq x} y_j \oplus f_j$$

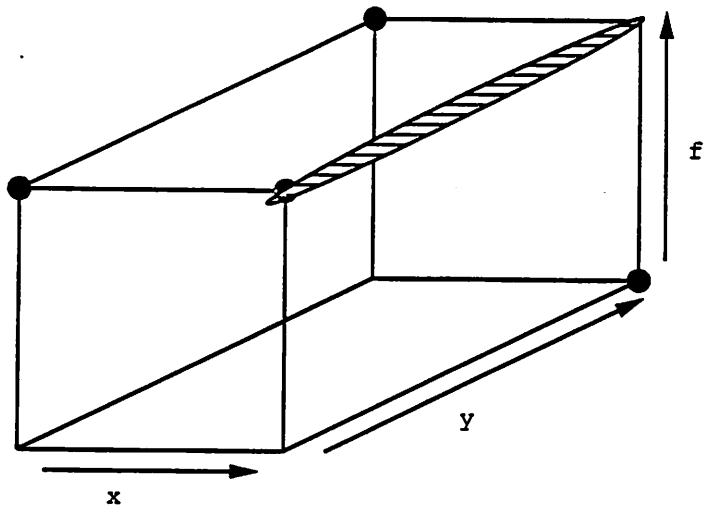


Figure 3: The Cofactor of f with respect to x

Since there are now 2^{n+1} inputs to this network, there are $2^{n+m} - 2^{n+1}$ points in $SDC(x)$, which can also be viewed as the set of points in $\mathbf{B}^{n+m}$outside the range of a new evalutation map denoted θ_x. θ_x is the evaluation map of the network with x disconnected from its function, and can be derived from the network by considering the modified global functions $\tilde{f}^{(x)}$.

For concreteness, we give an example of the SDC for the network $y_1 = xy, y_2 = x + y_1$. In figure 4(top), we show the SDC for the network with both y_2 and y_1 as internal variables. In figure 4(bottom), we show the SDC for the network treating y_1 as a primary input.

Note we can write the SDC in the first case as $y_1\bar{x} + y_1\bar{y} + \bar{y_1}xy + y_2\bar{x}\bar{y_1} + \bar{y_2}x + \bar{y_2}y_1$. For the second case, the SDC(which is $SDC(y_1)$) is written $y_2\bar{x}\bar{y_1} + \bar{y_2}x + \bar{y_2}y_1$.

Theorem 5.2 $f_x = y\overline{SDC(x)}_{yx}$

Proof: For x a primary input, $SDC = SDC(x)$ and the result follows from theorem 5.1. For x not a primary input, $SDC(x)$ is the set of consistency conditions for the network $N(x)$. f is the local function of y, and so is identical in both networks. Thus by theorem 5.1

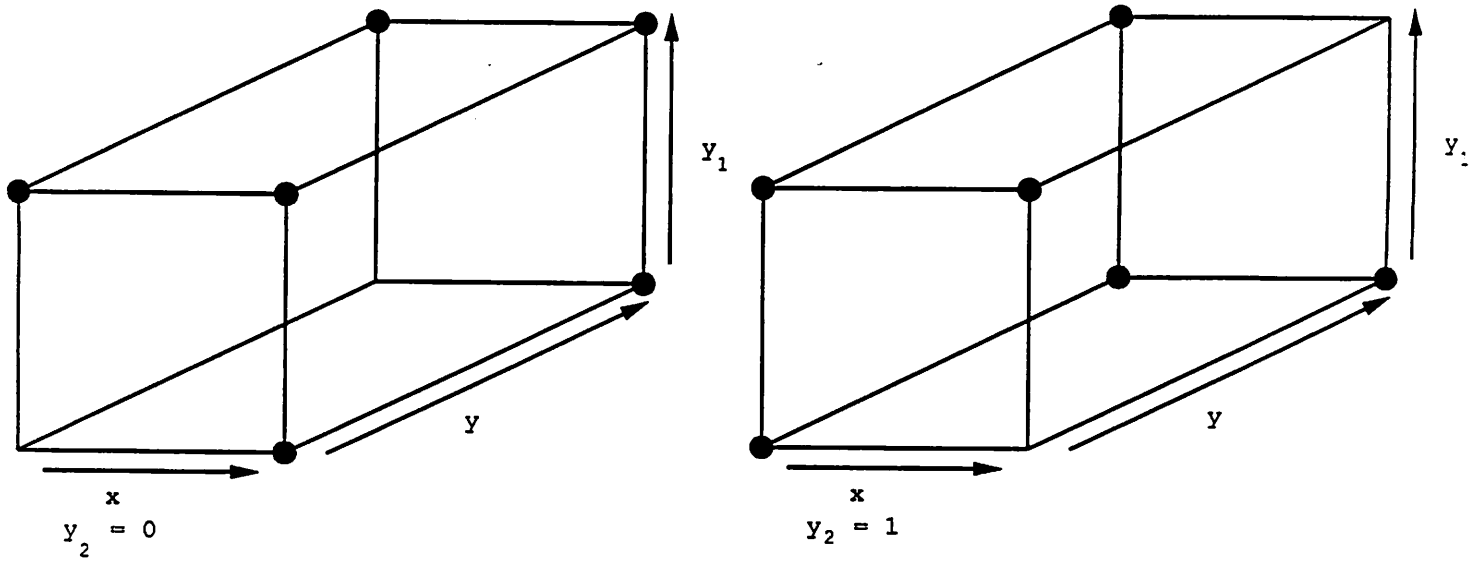
$$f_x = y\overline{SDC(x)}_{yx}$$

■

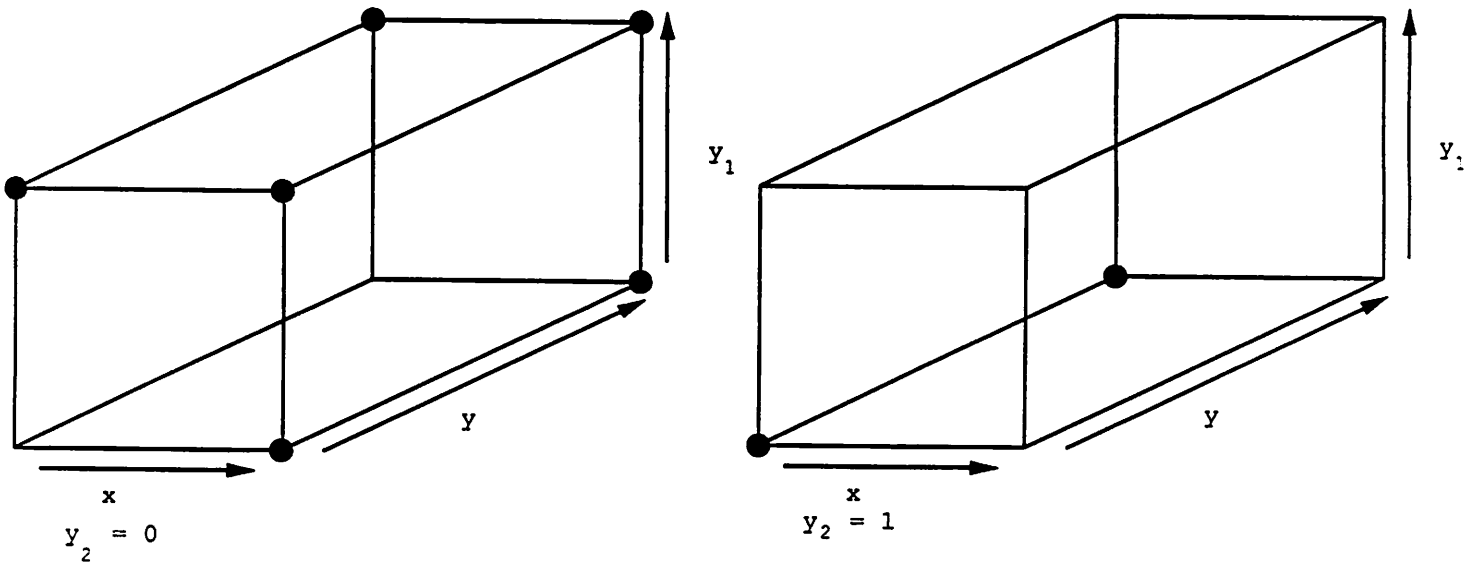
Corollary 5.3 $\tilde{f_x^{(x)}} = \mathcal{P}(\overline{SDC(x)}_{yx})$

Note that since $SDC(x) = SDC$ for a primary input x, this definition is valid for cofactors defined with respect to a primary input.

There is one more interesting thing to say. We can now provide an *exact* count of the number of vertices added to the SDC by any non-primary-input y_j, *regardless of the function at* y_j. The fact that the number of vertices in SDC which do not appear in $SDC(y_j)$ (the amount contributed by the local SDC of y_j, $y_j \oplus f_j$) is a constant is surprising. Nevertheless, it follows as an immediate consequence of the theory developed above.



$f_2 = y_1 + x, \; y_1 = x$



$f_2 = y_1 + x$

Figure 4: SDC for y_1 as a Function and as a Primary Input

Theorem 5.4 *For each f_j in a network of n primary inputs and m functions,*

$$|\overline{SDC(y_j)} \cap (y_j \oplus f_j)| = 2^n$$

Proof: The formula is just the number of vertices which appear in the LSDC that would not otherwise appear in the SDC, i.e., those vertices in the SDC only because y_j is a function. The fact that it is 2^n follows immediately from the fact that:

$$SDC = SDC(y_j) \cup (y_j \oplus f_j)$$

Now, $|SDC| = 2^{m+n} - 2^n$. $SDC(y_j)$ is the SDC of the network formed by considering y_j as a PI. This is a network of $n+1$ primary inputs and $m-1$ functions, and so its SDC, $SDC(y_j)$ is of size $2^{m+n} - 2^{n+1}$. The quantity we are after is the difference of these two sizes, which is:

$$2^{m+n} - 2^n - (2^{m+n} - 2^{n+1})$$

or:

$$2^{n+1} - 2^n = 2^n(2-1) = 2^n$$

as desired. ■

The theorem is somewhat more intuitive when it is considered that each primary input combination determines the value of x, and hence adds one vertex to the SDC.

6 The Observability Function and The Fanout Don't-Care Set

A final class of true don't-cares appears from the observation that not all functions are visible as output functions. Of the m functions realized in a network, perhaps r will appear as outputs, and only these are directly observable. The remainder are only of interest to the extent that they are observable at the outputs. When the results are unobservable, the result of these functions are don't-cares.

Unobservable functions are classically treated as those for whom one or another of the stuck faults is untestable. It has been shown[6,14]that the set of all stuck-at-0 tests for a node x through primary output f in a combinational network is the function:

$$x\frac{\partial F_j}{\partial x}$$

(i.e., we force x to be 1 in the good network and observe the difference at the outputs) and the set of all stuck-at-1 tests is the function

$$\bar{x}\frac{\partial F_j}{\partial x}$$

The observability function is obtained by summing $\frac{\partial F_j}{\partial x}$ over all outputs F_j[4]

$$O(x) = \sum_{j \in \text{outputs}} \frac{\partial F_j}{\partial x}$$

Now, the function $\frac{\partial f}{\partial x}$ is defined in terms of cofactors:

$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}$$

[4]Note that what is really meant here is that the faulty network is $N(x)$ with input x stuck-at. Hence the measurement of the difference should really be taken over the global image of $N(x)$; what is really meant here is $\frac{\partial F_j^{(x)}}{\partial x}$. In order to be consistent with the testing literature, we stick to the conventional notation

From the previous definition of the generalized cofactor in the preceding section, we can write each cofactor as the set of vertices of the appropriate face outside the appropriate SDC set. In particular,

$$\begin{aligned} \frac{\partial f}{\partial x} &= f_x \oplus f_{\bar{x}} \\ &= \mathcal{P}(\overline{SDC(x)}_{yx}) \oplus \mathcal{P}(\overline{SDC(x)}_{y\bar{x}}) \end{aligned} \tag{1}$$

To obtain the fanout-don't care (FDC) set associated with output f, we simply invert (1), denoting the inverse of the exclusive or function as boolean equality $\overline{\oplus}$, to obtain:

$$\mathcal{P}(\overline{SDC(x)}_{yx}) \overline{\oplus} \mathcal{P}(\overline{SDC(x)}_{y\bar{x}}) \tag{2}$$

Now, notice that the observability function is a function of the primary inputs, and so too must be its inverse, the fanout don't care set. This is a don't care set for one internal function x, and so each vertex maps to a point in $\overline{SDC}$. Thus the fanout don't-cares increase the don't-care vertices for this function over the space of the primary inputs.

7 Transformations on the Network

The purpose of logic synthesis is the redesign of the networks which implement these functions. We depict this general process as a sequence of well-defined *transformations* of the network, which yields a network realizing the same function which is hoped to be in some sense simpler. We are here concerned with demonstrating that certain of these transforms preserve certain properties of the network; to wit, the satisfiability don't-care set (and so the forcing sets), and the testability of nodes for stuck-at-faults. We summarize the relevant notions.

A *boolean network* N is a set of *nodes*, each of which is a pair (y_j, f_j) y_j is the unique variable name of the node; f_j is a function over some set of variables $(y_{j_1} \ldots y_{j_k})$, where each y_{j_i} is a node in the network. If $f_j = y_j$, then y_j is a primary input.

Each node has a *global image* $\tilde{f}_j$, which is the function of the node realized over the primary input space. $\tilde{f}_j$ is defined as follows:

$$\tilde{f}_j = \{v \in \mathbf{B}^n | \theta(v) \in y_j\}$$

The *consistency* condition on $\tilde{f}_j$ is that it truly represents the global function of the network, i.e. it represents the function that would be obtained by collapsing the network in the conventional sense.

We begin by considering classes of transformations.

At the lowest level, the set of transformations of the network may be divided into two categories: those that change the dimensionality of the extended space and those that do not. The reason for treating these classes of transform separately is that the objective of this section is to classify the effect of the various transforms on the SDC set. Clearly transforms which change the space change the SDC set. However, they do so in such a way that the set over the expanded (or contracted) space can be easily derived.

7.1 Changing the Dimensionality of a Network

When we say expansions or contractions of the space, we mean simply the addition or deletion of a variable, with its accompanying function. Substituting the variable into the network through the use of division, or collapsing the variable out of the network by elimination, are considered separate operations. Further, we make the strong assumption that either the addition or deletion operations

are *correct* in the obvious sense: that is, they do not affect the images of the existing or remaining functions.

The effect on the SDC of adding a new variable is clear. The function space expands by one dimension. Let y_j be the new variable. Every vertex w in the previous SDC set becomes two vertices in the new SDC set $\bar{y_j}w$ and y_jw, since adding the new variable will not change the fact that no vertex of the primary input space satisfies w. Now, for each vertex w not in the previous SDC set, one of the two new vertices y_jw and $\bar{y_j}w$ goes in the SDC set, according as to whether the new function is 0 or 1 on $\mathcal{P}(w)$.

The key thing to understand is that the addition of a dimension can be shown to require the addition to the SDC set of precisely those points contained in the *local* SDC of the function being added:

$$y_j \oplus f_j$$

and hence we can write, denoting the new SDC as SDC':

$$SDC' = SDC + (y_j \oplus f_j)$$

The effect of deleting a variable is similar. Let y_j be the variable under deletion. Each vertex w such that both y_jw and $\bar{y_j}w$ were in the previous SDC set goes into the new SDC set, SDC', and each other vertex w remains outside the new SDC set. Note that because the size of the SDC set is strictly a function of dimension, there are precisely 2^n of the latter vertices in the space. Hence in each dimension there are exactly 2^n vertices of the SDC distance 1 in that dimension from the complement of the SDC.

In sum, the deletion of the node represented by the variable y_j merely requires that the SDC be re-expressed by the Shannon expansion:

$$SDC' = SDC_{y_j} SDC_{\bar{y_j}}$$

Transformations that do not change the space can be further subdivided into those that do not change the SDC and those that do. The objective of this section is to arrive at a precise criterion for this determination.

7.2 Transformations Which Leave the Dimensionality Fixed

In general, there are don't-care sets (e.g., input, output, and fanout don't-cares), but many transformations do not use them. Formally, we say a transformation does not use a don't-care set if the transformation remains invariant when the don't-care sets are reduced to 0. In such a case, the transformation must leave the SDC set invariant, for the transformation must leave the SDC set invariant when the don't-care sets are 0. It is always possible to do this reduction, since each of the three true don't-care sets is entirely a matter of specification: the input and output don't-care sets are explicitly specified. The fanout don't-care set for each internal node is derived from the specification of some nodes of the circuit as primary outputs.

Since the don't-care sets for a boolean network N are thus specifications of a circuit, we can then formally define a *circuit* as an ordered pair (N, D) where N is a boolean network in the usual sense and D is a vector of don't care sets, where D_i is the don't-care set for node f_i of a network. D_i is equal to the union of the input, fanout, and output don't-care sets of node f_i.

A *transformation* $\mathcal{T}$ is then a mapping from a circuit (N, D) to a new circuit (N', D'); in general, $N \neq N'$ and $D \neq D'$; in this subsection, we are considering only transformations where the set of nodes is preserved by the transformation (though, perhaps, the *local* functions at the various nodes may change).

A transformation $\mathcal{T}(N, D) \rightarrow (N', D')$ is said to be *invariant* iff for each node y_j, the global image $\tilde{f}_j$ of y_j is preserved.

Theorem 7.1 *A transformation is invariant iff it leaves the SDC unchanged.*

Proof: If a transformation is invariant, it leaves the global image of each node unchanged. Hence the map $\theta : \mathbf{B}^n \rightarrow \mathbf{B}^{n+m}$ is left unchanged, and so to is its range, $\overline{SDC}$. Hence the SDC is left unchanged. Conversely, suppose $\mathcal{T}$ leaves SDC unchanged. For any function f,

$$f = \{w | w \in \overline{SDC} \text{and} w \in y\}$$

since SDC is unchanged, then so is f, and hence so is $\tilde{f} = \mathcal{P}(f)$. ∎

Corollary 7.2 *Every transform on a completely specified network where every node is a primary output is invariant.*

Proof: On such a network, for every node, the don't-care set is 0. Thus the global image of each node must be unchanged and, by theorem 7.1, the SDC is unchanged. ∎

This result gives the key insight into the basic theorem of invariant transforms. Since every transform on such a super-specified network is invariant, the next question is, which transforms behave the same way on a super-specified and incompletely-specified network? Such transforms must be invariant; further, *only* these transforms are invariant, as we see below.

Theorem 7.3 *Let $\mathcal{T}(N, D) = (N', D')$ be any transformation of a network. $\mathcal{T}$ is invariant iff for every D, $\mathcal{T}(N, 0) = (N', D'')$ (i.e., N' is obtained independent of D).*

Proof: For every D, let $\mathcal{T}(N, 0) = (N', D'')$ and $\mathcal{T}(N, D) = (N', D')$. Since $\mathcal{T}$ is unchanged even when the network is completely specified, and since the SDC can only be changed under a rotation specified by a don't-care set D, it follows that the SDC is left unchanged by $\mathcal{T}$, and so it follows that $\mathcal{T}$ is invariant. For the converse, $\mathcal{T}$ is invariant, and so all global images remain unchanged. Thus the don't-care set D_j was not used to change f_j. Hence $\mathcal{T}(N, D) = (N', D')$ and $\mathcal{T}(N, 0) = (N', D'')$. ∎

Interestingly, for all the abstraction of this theorem, one can say immediately that all the transformations within the MIS-II[3] synthesis system, with the sole exception of *node-simplify*, do not change the SDC set, and so the forcing sets remain invariant under this transformation. Further, *node-simplify* does not change the SDC or the forcing set of any node unless external or fanout don't-cares care used.

One can further say that *topological* optimizations, such as the global flow procedure, *implicitly* use the fanout don't-care set, and so potentially change the SDC set. However, this can be shown to occur in only a small set of dimensions; in particular, if the SDC set is formed of a series of local SDC sets, only the local SDC sets of an identifiable set of functions need be changed. We now elaborate this point.

Consider the global flow optimization procedure, especially as elaborated in [4]. This procedure encompasses two phases. In the first phase, a cutset of nodes between some node f and the primary outputs is identified, such that, if g is in the cutset, setting a value on f forces a value on g; the local function at g is then modified to make its dependence on f explicit. This phase is an invariant transformation, since the expansion is explicitly constructed to preserve the global image of g.

In the second phase, edges leading out from f are deleted from nodes h between f and the cutset. This explicitly changes the global functions of such nodes h, and, hence, the various sets

which depend upon these global functions must be recomputed. An immediate question that arises is whether the entire SDC and the forcing sets must be recomputed, a potentially expensive proposition. The answer to this question may be answered in the negative, due to the following result.

Theorem 7.4 *Let $\mathcal{T}(N, D) = (N', D')$ be any transformation. Let $f_j \in N$, $f_j \in N'$ (i.e., the local function on y_j is unchanged). Further, for each y_k a fanin of f_j, let $\tilde{f}_k$ be unchanged. Then $\tilde{f})_j$ is unchanged.*

Proof: $\tilde{f}_j$ can be obtained by collapsing $\tilde{f}_k$ into f_j in place of y_k. ∎

This immediately implies that the only change that need be made in the satisfiability don't-care set is the recomputation of the local SDC's of those functions that have *explicitly* changed, and in the forcing sets of those functions whose global image has changed. So even in the case when a transform is not invariant, the techniques of this section can be used to demonstrate that only a subset of the don't-care set has changed.

8 Testability Preservation

Every operation in a logic synthesis system has as its ultimate goal the minimization of the network. A vital component of this strategy is the removal of redundant edges. Hence we would not expect most operations on a network to preserve exactly the observability function for each node. However, we *can* attempt to discover which nodes and edges have had their observability functions *potentially* affected by each transformation.

We will first discuss the testability *output* stuck faults: that is, the condition that the output of a node is testable for stuck-at-faults. This is not the condition usually sought by testing algorithms or logic synthesists interested in redundancy removal; these seek whether every *edge* is testable for stuck-at faults. However, this is the property most strongly related to the fanout don't-care sets. Further, it is strongly related to edge testability, in the following sense. If y is an input to node f, then for each primary output g we have the identity:

$$\frac{\partial g}{\partial y} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial y}$$

Hence if we can show that $\frac{\partial g}{\partial f}$ and $\frac{\partial f}{\partial y}$ are preserved for a given node and input edge, we have shown that the test function for that edge is preserved. Further, we can artificially insert a buffer at each edge; thus the testability of the output of the buffer is equivalent to the testability of the edge.

Now, for the remainder of this discussion, we will only be concerned with invariant (or SDC-preserving) transforms. Clearly transforms which change the functions of nodes in the network have the potential to radically alter the test set. If we consider the set of SDC-preserving transforms we see that the condition of preservation of the SDC is not strong enough to maintain testability; maintaining the testability of node x requires that not only must SDC be maintained, but $SDC(x)$ must be maintained, since the observability function is defined in terms of $SDC(x)$. We state this intuition formally here.

A transform $\mathcal{T}(N, D)$ is said to be test-preserving for node y_j if the fanout don't-care set of node y_j is left unchanged by the transformation.

Clearly every transformation worth the name will affect the fanout don't-care set of at least some nodes, so the problem is not to characterize the transforms which leave the entire set of output don't-care sets invariant, but, rather, to characterize for a given transform which don't-care sets are left invariant. Fortunately, this set is easily characterized:

Theorem 8.1 *Let $\mathcal{T}(N, D)$ be an invariant transform. If $SDC(x)$ is left unchanged by $\mathcal{T}$, then the observability function of x is unchanged and hence the transform is test-preserving.*

Proof: Recall that the cofactor of f wrt x can be written (f corresponds to variable y, as before):

$$f_x = y\overline{SDC(x)}_{yx}$$

Hence we can write $\frac{\partial f}{\partial x}$ as:

$$\mathcal{P}(\overline{SDC(x)}_{yx}) \oplus \mathcal{P}(\overline{SDC(x)}_{y\overline{x}})$$

Since $SDC(x)$ is preserved, and since the observability function is a sum over the primary outputs F_j of $\frac{\partial F_j}{\partial x}$, it is preserved if $SDC(x)$ is preserved. ∎

Thus the question of the preservation of the observability function for node x is then the question of the preservation of $SDC(x)$. Recall the definition of $SDC(x)$:

$$SDC(x) = \sum_{y_j \neq x} y_j \oplus f_j$$

and recall that this is precisely equal to the SDC of a network which differs only from the original network in that node x is replaced by a primary input x; indeed, it was this observation that led to the definition of cofactor over the extended space. Recall also that $N(x)$ is the network N with node x replaced by a primary input labelled x. Note that if x is a primary input, then $N(x) = N$.

Theorem 8.2 *Let $\mathcal{T}(N, D) = (N', D')$ be any invariant transformation such that $\mathcal{T}(N(x), D(x)) = (N'(x), D'(x))$. Then $\mathcal{T}$ preserves $SDC(x)$.*

This theorem encodes a pair of informal observations. First, an invariant transform preserves the cofactors of every primary input, and, second, that if the function attached to node x is ignored by the transform, then x is indistinguishable from a primary input and so its cofactors and hence its observability function are preserved by the transform.

Proof: Since $\mathcal{T}(N(x), D(x)) = (N'(x), D'(x))$, $SDC(x)$ of N' is equivalent to the SDC of the network obtained by taking the transformation on N and then removing the function feeding x. This is equivalent to the SDC of $N(x)$, since $\mathcal{T}$ is invariant. But this is $SDC(x)$ of N, and hence $SDC(x)$ of N' is equal to $SDC(x)$ of N, and so $\mathcal{T}$ preserves $SDC(x)$. ∎

The corollaries of this theorem are immediate. One need not actually form the network $N(x)$ for each node x and then compute the transformation; it suffices to determine which functions f_j are used by the transformation. If the function attached to node y_j is unused, then it follows instantly that the transform operating on $N(y_j)$ returns the same result as that operating on N, and so by the theorem the testability of node y_j is unaffected.

As an example, consider the transformation given by the algebraic division procedure [11][3]. Algebraic division examines only two functions, those of the divisor and the dividend. Hence it follows immediately that at most the testability of the divisor and the dividend are affected. In fact, this observation can be generalized. Consider the classic form of the boolean division algorithm in figure 5, taken from [3]. Note that the only functions referred to in this procedure are those of f and g. *Hence, the classic boolean division procedure has no greater or lesser effect on testability than does the algebraic procedure.* This result seems counterintuitive, based on the usual intuition that boolean procedures are more powerful than algebraic procedures, and on the experience that boolean division seems to return better results than algebraic division. Nonetheless, the fact that the classic boolean division procedure uses only the local SDC associated with the

```
bool-divide(f, g)
    q, r <- alg-divide(f, g)
    minimize r with respect to the DC set (x ⊕ g) + qx
    (quotient, remainder) <- alg-divide(r, x)
    return (quotient + quot, remainder)
```

Figure 5: Classic Boolean Division

divisor means that in some sense it has no more information than the algebraic procedure. In an attempt to gain better results, subsequent boolean division procedures expand the don't-care set somewhat to gain a larger neighbourhood of information; these revised procedures can affect testability and redundancy more globally, but it is important to realize that one can analyze the effect on redundancy in precisely the same manner; loosely, an invariant transformation can only affect the observability of those nodes y_j such that the effect of the transformation is dependent upon the function f_j. We can state the effect of various transformations on the observability of various nodes in a series of corollaries. The omitted proof of each corollary is, formally, that the named transformation is dependent upon the functions of only the named nodes.

Corollary 8.3 *Algebraic and Classic Boolean division affect the observability function of (at most) the divisor and dividend.*

Corollary 8.4 *Collapsing node f into node g affects the observability function of (at most) f and g.*

Corollary 8.5 *Adding node f to or deleting node f from the network does not affect the observability function of any node $g \neq f$.*

Further, if one attempts to simplify a node using the subset-support filter [13] on the SDC and no external or fanout don't-cares, one can use this theorem to make strong statements concerning the preservation of testability of most of the network.

Earlier, we had occasion to observe the duality of node and edge testing. We now use this again, for another purpose. If the observability function for each edge out of a node is preserved, then the observability function for the node is obviously preserved. We can use this to say immediately:

Theorem 8.6 *Algebraic and the Boolean division of [3] do not affect the observability function of the dividend, f. Similarly, collapsing node g into node f does not affect the observability function of f.*

Proof: Note for f in both cases, neither the local function of any successor node of f was affected by the operation, nor the function on any input edge of such a successor node. Hence for each edge e out of f to successor node h, $\frac{\partial h}{\partial e}$ is unaffected, and, similarly, $\frac{\partial F_j}{\partial h}$ is unaffected for each output F_j, since h was neither the divisor, dividend, nor the node being collapsed. Hence we have that $\frac{\partial F_j}{\partial f}$ was unaffected, giving the result. ∎

This theorem has an interesting, and counterintuitive, consequence. Only the observability of the g, (respectively the divisor or the node being collapsed) is affected by the division or collapsing

computation. This runs deeply counter to one's intuition, which suggests that the inputs to g are similarly affected, since these inputs have paths through g to the output. In particular, one's intuition would argue that if g were made more or less observable by a transformation, then, at least potentially, its fanin should be made observable also. However, a little reflection obtains the following. If g in fact divides into f in a non-trivial way, then every fanin of g is a fanin of f, and hence the paths from the fanins of g are neither created nor destroyed.

In this section, we have shown that in any form of division, only the observability function of the divisor is affected: the observability function of every other node in the network is unaffected. Further, the operation of collapsing a node into a successor affects only the observability function of the node being collapsed.

9 Conclusions and Future Work

In this paper, we have demonstrated a technique by which the effect of transformations on the consistency and observability conditions in a network are preserved, and have used this technique to analyze a variety of procedures which have appeared in the current logic synthesis literature. In particular, we have demonstrated that no transformation can affect the consistency conditions in a network unless some form of external or fanout don't-care sets are used, and that observability of nodes and edges is affected by current transformation techniques in a very limited way. As we continue this research, we foresee two major applications. First, this technique can be used to evaluate synthesis heuristics to get some quantitative idea of their power; and, second, that we may use this technique to prove that a great deal of computation is unnecessary following transformations: for example, we have used this technique to show that most transformations preserve the $\mathcal{F}$-sets of a network in most cases, and that in other cases only a small, identifiable subset of the $\mathcal{F}$-sets have changed.

References

[1] S. B. Akers. On a theory of Boolean functions. *J. SIAM*, 1959.

[2] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Multi-level logic minimization using implicit don't cares. *IEEE Transactions on CAD*, 1988.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Mis: a multi-level logic optimization system. *IEEE Transactions on CAD*, 1987.

[4] R. K. Brayton, E. M. Sentovich, and F. Somenzi. Don't cares and global flow analysis of boolean networks. In *IEEE International Conference on Computer-Aided Design*, 1988.

[5] R. K. Brayton and F. Somenzi. Boolean relations and the incomplete specification of logic networks. In *Proceedings of VLSI '89*, 1989.

[6] H. Fujiwara. *Logic Testing and Design for Testability.* The MIT Press, 1985.

[7] G. Hachtel, R. Jacoby, K. Keutzer, , and C. Morrison. On the relationship between area optimization and multifault testability of multilevel logic. In *International Workshop on Logic Synthesis*, 1989.

[8] G. Hachtel, R. Jacoby, and P. Moceyunas. On computing and approximating the observability don't-care set. In *International Workshop on Logic Synthesis*, 1989.

[9] G. Hachtel, R. Jacoby, P. Moceyunas, and C. Morrison. Performance enhancements in BOLD using "implications". In *IEEE International Conference on Computer-Aided Design*, 1988.

[10] G. D. Hachtel and R. M. Jacoby. Algorithms for multilevel tautology checking. *IEEE Transactions on CAD*, 1988.

[11] P. C. McGeer and R. K. Brayton. Efficient, stable algebraic operations on logic expressions. In *Proceedings of VLSI '87*, 1987.

[12] J. P. Roth. Diagnosis of automata failures: a calculus and a method. *IBM J. Res. Develop*, 1966.

[13] A. Saldanha, A. Wang, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-level logic simplification using don't-cares and filters. In *Design Automation Conference*, 1989.

[14] Frederick F. Sellers, Jr., M. Y. Hsiao, and L. W. Bearnson. Analyzing errors with the Boolean difference. *IEEE Transactions on Computers*, 1968.

[15] L. Trevillyan and L. Berman. Improved logic optimization using global flow analysis. In *IEEE International Conference on Computer-Aided Design*, 1988.

[16] L. Trevillyan, W. Joyner, and L. Berman. Global flow analysis in automated logic design. *IEEE Transactions on Computers*, 1986.