Finding Composition Trees for Multiple-Valued Functions

E. V. Dubrova[*] J. C. Muzio
VLSI Design and Test Group
University of Victoria, P.O.Box 3055
Victoria, B.C., Canada, V8W 3P6
elena@shannon.uvic.ca
jmuzio@csr.uvic.ca

B. von Stengel[†]
Theoretical Computer Science
ETH Zürich
CH-8092 Zürich, Switzerland
stengel@inf.ethz.ch

Abstract

The composition tree of a given function, when it exists, provides a representation of the function revealing all possible disjunctive decompositions, thereby suggesting a realization of the function at a minimal cost. Previously and independently, the authors had studied the class of multiple-valued functions that are fully sensitive to their variables. These functions are useful for test generation purposes, and almost all m-valued n-variable functions belong to this class as n increases. All functions in this class have composition trees. This paper presents a recursive algorithm for generating the composition tree for any function in this class. The construction proceeds top-down and makes immediate use of any encountered decomposition, which reduces the (in general exponential) computation time.

1. Introduction

Most approaches to the logic synthesis of digital systems consist of two phases: a technology-independent phase that manipulates and optimizes functions, and a technology-mapping phase that maps functions onto a set of gates in a specific target technology. The technology-independent phase for two-level synthesis, resulting in two-level devices such as programmable logic arrays, is based on minimization techniques [2]. For multi-level synthesis, *decomposition* is the essential step in the technology-independent phase, leading to devices with multi-level structure such as field programmable gate arrays [3].

Generally, the problem of decomposition of functions can be formulated as follows. Given a function f, express it as a composite function of some set of new functions. Sometimes, a composite expression can be found in which the new functions are significantly simpler than f. Then the design of a logic circuit realizing f may be accomplished by designing circuits realizing the simpler functions of the composite representation, thus reducing the overall cost of implementing f.

However, the problem of selecting the "best" decomposition minimizing the overall cost of realization of a given function appears to be far too difficult to be solved exhaustively. Therefore, all previous efforts to apply decomposition theory to the design of Boolean and multiple-valued logic circuits restrict the decomposition to be obtained to a particular type. In our paper we consider *disjunctive* decompositions only. The basis for the different types of disjunctive decomposition is the *simple disjunctive* decomposition where a function $f(x_1, x_2, \ldots, x_n)$ is expressed as a composite function of two functions g and h, namely

$$f(x, y) = g(h(x), y) \tag{1}$$

with x and y being sets of variables forming a partition of the set of variables $\{x_1, x_2, \ldots, x_n\}$ of f. If f, g and h are m-valued functions, then in (1) the original function f specifying an n-input, 1-output m-valued circuit is replaced by the specifications of two m-valued circuits, one having p inputs and one output, and the other having $1 + n - p$ inputs and one output (see Figure 1). Every set of variables x such that f has a decomposition (1) is called a *bound set* for f. Such a decomposition exists *trivially* for x given by any singleton set $\{x_i\}$ or the all-set $\{x_1, x_2, \ldots, x_n\}$.

If $C_{n,m}$ is an upper bound on the cost of realizing an m-valued function of n variables, then the total cost of realizing these two circuits is bounded above by $C_{p,m} + C_{(1+n-p),m}$. Because the cost bound $C_{n,m}$ increases nearly exponentially with n [12], the discovery of any nontrivial decomposition of the form (1) greatly reduces the cost of realizing f. Once such a decomposition has been selected, either g, h, or both may be similarly decomposed, giving one of the following

*Supported in part by Research Grant No. 5711 from the Natural Sciences and Engineering Research Council of Canada and by an equipment loan from the Canadian Microelectronic Corporation.

†Supported by a Heisenberg grant from the Deutsche Forschungsgemeinschaft (DFG).

In: Proc. 27th IEEE International Symposium on Multiple Valued Logic (1997) 19–26.

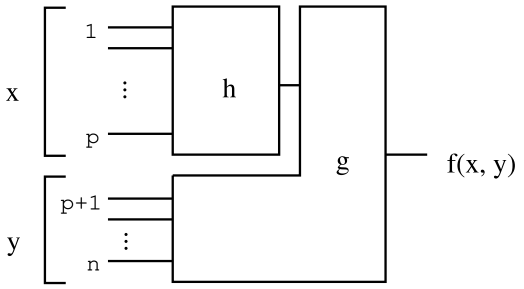

Figure 1. Simple disjunctive decomposition

complex disjunctive decomposition types [7]:

$$\begin{array}{ll} \text{multiple :} & f(x,y,z) = g(h(x), k(y), z) \\ \text{iterative :} & f(x,y,z) = g(h(k(x),y),z) \end{array} \qquad (2)$$

or more generally tree-like decompositions as in $f(x,y,z,w) = g(h(k(x),y), l(z), w)$.

Clearly, since each decomposition of type (1) reduces the overall cost of implementing f, the more f is decomposed, the more the cost is reduced. However, sometimes a function can be decomposed in several different ways, depending on the bound set chosen, e.g. when two decompositions of the same function $f(x,y) = g(h(x),y)$ and $f(z,v) = k(l(z),v)$ exist such that $x \cap z \neq \emptyset$. We call such decompositions *conflicting*. Therefore, at this point, a theory is needed to decide which bound sets should be chosen to obtain the "most decomposed" representation of f. Such a theory was developed by Ashenhurst [1] for the case of Boolean functions, and generalized by the third author of the present paper to a certain class Σ of general n-ary operations on (not necessarily finite) sets [13]. The decomposition properties of the functions in Σ are as useful as possible in the sense that there is a final decomposition that represents *any* bound set of variables. The simple case is that two such bound sets are disjoint or that one contains the other, which gives rise to a multiple or iterative decomposition, respectively. The interesting third possibility is that two bound set are *overlapping*, i.e. given by (x,y) and (y,z) where x, y, and z are nonempty sets of variables. Remarkably, this implies a representation of f in the form

$$f(x,y,z,w) = g(a(x) \bullet b(y) \bullet c(z), w) \qquad (3)$$

with an m-valued associative binary operation $\bullet$, where the two ways of writing parentheses, $(a(x) \bullet b(y)) \bullet c(z)$ and $a(x) \bullet (b(y) \bullet c(z))$, show that (x,y) and (y,z) are bound sets (as assumed), as well as (in consequence) their intersection y, union (x,y,z), set-theoretic differences x and z, and possibly (x,z) if $\bullet$ is commutative (as always for Boolean functions).

Starting from these prototypical cases for two bound sets, the disjunctive decomposition theory in [13] defines a *composition tree* of the function f which gives a representation of the function reflecting any bound set of variables, thus a "most decomposed" one. Hence, the realization of the given function in correspondence with the composition tree (with suitable assumption about the cost of logic elements) should have a cost that is close to minimal. In the sixties it was even conjectured that such an implementation must be a minimal one. However, Paul [11] found a counterexample demonstrating a circuit derived by other than decomposition techniques that has smaller cost than the one implementing the composition tree. Nonetheless, this seems to be the exception.

This paper summarizes the results on composition trees from [13] restricted to the case of m-valued n-variable functions, defining composition trees and specifying a class of functions Σ for which the composition tree exists. We show the close connection between the class Σ and *full sensitivity*, the discrete difference introduced in [5] for test generation purposes, which allows us to apply some of the results for full sensitivity to the class Σ and vice versa. For example, one of the results from [6], restated in the terminology of the present paper, says that the percentage of m-valued n-variable functions which are not in Σ tends to zero as n increases.

The results of [13] prove the existence of the composition tree for a particular class of functions and give a description of this tree. However, this is not algorithmic and does not describe how to build the composition tree from a specification of the given function. In this paper we describe a recursive algorithm for generating composition trees of functions in Σ. It is a combined top-down and depth-first construction of the tree. In parts, it generalizes Curtis' approach [4] for Boolean functions to the m-valued case. A bottom-up construction that could be adapted to our situation is described in [10].

If all functions – or at least a large class of m-valued functions – were disjunctively decomposable, the algorithm presented in our paper would have been more than adequate for obtaining highly economical multi-level m-valued circuits. However, the fraction of all Boolean functions of n variables possessing nontrivial disjunctive decompositions of type (1) approaches zero as n approaches infinity [12, p. 90]. It is straightforward to generalize this result to m-valued functions for $m > 2$. However, this does not mean that the disjunctive decomposition theory developed in [1] and [13] is of no practical value. First, the "practical" functions are not randomly distributed through the space of all functions. Second, in the Boolean case, Ashenhurst's disjunctive decomposition theory led to the formulation in [4] of the general theory of nondisjunctive decompositions, a theory encompassing all switching functions on n variables regardless of the size of n. We hope that the theory developed in [13] can serve as a base for more general investigations of m-valued functions. Several directions for the extension of the theory are discussed in the final section.

The paper is organized as follows. Section 2 recalls the definition of bound sets, which we study for the class Σ of fully sensitive functions. This class is defined in [5], [6], and independently considered for decomposition in [13]. The main results on composition trees are summarized in Section 3. For proofs we refer to [1], [10], [13]. Section 4 describes the new recursive algorithm. Section 5 contains conclusions and discusses possible directions for further research.

2. Function class Σ and full sensitivity

Let $f(x_1, x_2, \ldots, x_n)$ be a completely specified multiple-valued function. In general, the multiple-valued functions may be *heterogeneous* where the variables of the function do not take values in the same set. However, in our paper we consider only the case of *homogeneous* functions of type $M^n \to M$ on a fixed set $M := \{0, 1, \ldots, m-1\}$, and restrict all functions participating in the decomposition to this type.

Let the set of indices of the variables be $N = \{1, \ldots, n\}$. Any subset A of N defines a vector $(x_i)_{i \in A}$ of such variables, so that $x_N = (x_1, \ldots, x_n) \in M^n$. Let $M^A = \{(x_i)_{i \in A} \mid x_i \in M\}$. By arranging the variables as it is suitable, we can postulate

$$M^N = M^A \times M^{\overline{A}},$$

where $\overline{A} = N - A$. A *simple disjunctive decomposition* of f is given by a nonempty subset A of N and suitable functions $h\colon M^A \to M$ and $g\colon M \times M^{\overline{A}} \to M$ so that

$$f(x, y) = g(h(x), y) \tag{1}$$

for all $x \in M^A$, $y \in M^{\overline{A}}$. Recall that every set A of (indices of) variables such that f has such a decomposition is called a *bound set* for f (equivalently, we may call the set $\{x_i \mid i \in A\}$ or the vector $x = (x_i)_{i \in A}$ a bound set of variables).

The classical method for recognizing a bound set is based on representing the function by a *decomposition chart* [1], [4]. The decomposition chart for $f(x, y)$ is a two-dimensional table where the columns represent the elements x of M^A and the rows the elements y of $M^{\overline{A}}$. Then A is a bound set if and only if the chart has *column multiplicity* at most m, i.e. there are at most m distinct columns in the chart [7]. The reason is that each such column is a function $y \mapsto f(x, y)$ for fixed x (called a *subfunction* of f of the variables y) which by (1) is equal to $y \mapsto g(h(x), y)$, so there cannot be more of these columns than h takes values. Figure 2 shows such a chart for the $\{x_3\}|\{x_1, x_2\}$ partitioning of the variables of $f(x_1, x_2, x_3)$, where the set $\{x_1, x_2\}$ is indeed a bound set.

$x_1 x_2$	00	01	02	10	11	12	20	21	22
x_3 0	1	0	0	0	0	0	1	0	1
1	0	1	2	2	1	1	0	1	0
2	1	1	1	1	1	1	1	1	1

Figure 2. Decomposition chart for an f in Σ

Using these charts, Ashenhurst [1] showed representations like (3) and constructed the composition tree for any n-variable Boolean function that is nondegenerate, i.e. which actually depends on all n variables to determine its output. We may call such a function *sensitive* to all its inputs. Call a function f *fully sensitive* to a variable x_i if there are fixed values for the other variables such that *any* change in x_i causes a change in the value of f.

Let Σ be the class of functions $f\colon M^n \to M$ for some $n \geq 1$ that are fully sensitive to all their variables. In [13], it is shown (using different terminology) that the functions in Σ have composition trees, generalizing this property of nondegenerate Boolean functions. Note, that for $m = 2$, sensitivity is the same as full sensitivity. Independently, full sensitivity is introduced in [5] for test generation purposes, where this name denotes an algebraic expression that indicates where the function is fully sensitive to a given input.

Let x_i be a variable of $f(x_1, \ldots, x_n)$ and $z = (x_j)_{j \neq i}$ be the vector of remaining variables. Full sensitivity of f to x_i means that for some fixed z, the subfunction $x_i \mapsto f(x_i, z)$ is injective, in fact bijective (a permutation of M) since x_i and f have only m possible values. For $f(x_1, x_2, x_3)$ in Figure 2, the subfunctions $x_1 \mapsto f(x_1, 2, 1)$, $x_2 \mapsto f(0, x_2, 1)$, and $x_3 \mapsto f(0, 2, x_3)$ are such bijections, so this function f is fully sensitive to all its variables. In the decomposition chart for $f(x_i, z)$ where the possible values for z and x_i denote rows and columns, this means that at least one row (fixed z) has m distinct elements. This is not necessarily the case, even if this chart has m distinct columns. However, a result from [6] states that the percentage of m-valued n-variable functions which are not fully sensitive to all their variables tends to zero as n increases (intuitively, because it becomes unlikely not to find a bijection among the m^{n-1} rows of the chart).

Subfunctions of f of one variable are used to define full sensitivity. Subfunctions of several variables (like the rows and columns in any decomposition chart) can be used for representing g and h in (1), which is the central idea of the decomposition theory in [13]:

Lemma 1. *Let $f \in \Sigma$, $f\colon M^N \to M$, and $\emptyset \neq A \subseteq N$. Then the following are equivalent:*

(a) A is a bound set for f,

(b) f has a decomposition (1) where $h(x)$ is a subfunction

of f of the variables $x \in M^A$,

(c) f has a decomposition (1) where $g(x_i, y)$ is a subfunction of f of the variables $x_i \in M$ and $y \in M^{\overline{A}}$, for any $i \in A$.

Here, (b) means that $h(x) = f(x, \hat{y})$ for some fixed $\hat{y} \in M^{\overline{A}}$. That is, $\hat{y}$ is a suitable row of the decomposition chart for $f(x, y)$. In Figure 2 where $A = \{1, 2\}$, we can choose $\hat{y} = \hat{x}_3 = 1$ and observe $f(x_1, x_2, x_3) = g(f(x_1, x_2, 1),\, x_3)$ so the second row $f(x_1, x_2, 1)$ of the chart is a suitable representation of $h(x_1, x_2)$.

Condition (c) is slightly more complicated. In (1), g has $1+|\overline{A}|$ variables since one variable is substituted by $h(x)$. To obtain $g(x_i, y)$ as a subfunction of f, any variable x_i of the vector x can be chosen, $i \in A$. That is, $x = (x_i, v)$ where $v = (x_j)_{j \in A,\, j \neq i}$, so that (c) asserts $g(x_i, y) = f((x_i, \hat{v}), y)$ for some fixed $\hat{v}$. In Figure 2, we can choose $x_i = x_2$, $\hat{v} = \hat{x}_1 = 0$ and let $g(x_2, y) = g(x_2, x_3) = f(0, x_2, x_3)$ (the left of the three squares of the chart, which represents a sample of the m columns that can occur).

As an extension to Lemma 1, one can show that the fixed values $\hat{y}$ and $\hat{v}$ in these representations can be any vector $(\hat{v}, \hat{y})$ such that $x_i \mapsto f((x_i, \hat{v}), \hat{y})$ is a bijection (as used for full sensitivity), like $x_2 \mapsto f(0, x_2, 1)$ in Figure 2. Hence, if $f \in \Sigma$ and we know a fixed vector $\hat{z} = (\hat{x}_j)_{j \neq i}$ such that $x_i \mapsto f(x_i, \hat{z})$ is bijective (as a "witness" for the full sensitivity to x_i), then the functions g and h in a decomposition (1) of f are found very simply by just fixing some variables of f at the levels in $\hat{z}$. Furthermore, Lemma 1 implies the following.

Corollary 2. *Σ is closed under composition and decomposition: If g and h in $f(x, y) = g(h(x), y)$ belong to Σ, then so does f, and vice versa.*

The class Σ of fully sensitive functions does not include *all* functions for which the composition tree can be built. However, we have not yet found a simple weaker condition. In general, arbitrary m-valued n-ary functions may not have composition trees, contrary to a conjecture in [7]. When looking for functions that do not have a composition tree, full sensitivity must be violated. Such counterexamples exist but are not easy to find. This is no coincidence since, as mentioned, n-variable functions not in Σ are rare for large n [6]. The connections of full sensitivity as defined in [5] and [6] and the structural decomposition theory in [13] are fruitful in many other respects. For example, the procedure for algebraic calculation of full sensitivity from [5] can be used for checking whether a given f is in Σ. Apart from the existence of the composition tree, Corollary 2 is a nice result in terms of circuit testing since, as shown in [6], a circuit implementing a function fully sensitive to all its variables can be tested for all single stuck-at faults on primary inputs with two tests only.

3. Composition Trees

Throughout, let $f \in \Sigma$, $f\colon M^N \to M$, $N = \{1, \ldots, n\}$. A function f that has only the *trivial bound sets* N and $\{i\}$ for $i \in N$ (so that g or h in (1) is unary) is called non-decomposable or *prime*, as for example if $n \leq 2$. If A is a nontrivial bound set and (1) holds, then other bound sets that are subsets or supersets of A or disjoint to A relate to decompositions of h and g:

Lemma 3. *Let A be a bound set with decomposition (1), where $h\colon M^A \to M$ and $g\colon M^{\{i\} \cup \overline{A}} \to M$ for some $i \in A$. Then for all sets $C \subseteq A$, $D \subseteq \overline{A}$:*

(a) C is bound for f $\iff$ C is bound for h,

(b) $A \cup C$ is bound for f $\iff$ $\{i\} \cup D$ is bound for g,

(c) D is bound for f $\iff$ D is bound for g.

Several bound sets for f, as in Lemma 3, lead to iterative or multiple decompositions of f as in (2). Call a bound set A for f *strong* if any other bound set is either a subset or superset of A or disjoint to A. For example, the trivial bound sets N and $\{i\}$ are strong. Clearly, the partial order of inclusion among these strong bound sets defines a tree. This tree with strong bound sets as nodes (suitably labeled) is called the *composition tree* of f.

Each node of the composition tree is labeled with a function that has as many variables as the node has children. Leaves are labeled with unary functions, which may be the identity. The hierarchical term of these functions represents f. For a tree like in Figure 3 (which will be described more fully below), this term may be

$$f(x_1, \ldots, x_6) = g(h(a(x_1, x_2),\, x_3, x_4),\, x_5, x_6). \qquad (4)$$

If all these functions are prime, then all bound sets are strong and the composition tree is fully described. The interesting case is therefore that some bound sets are overlapping with others.

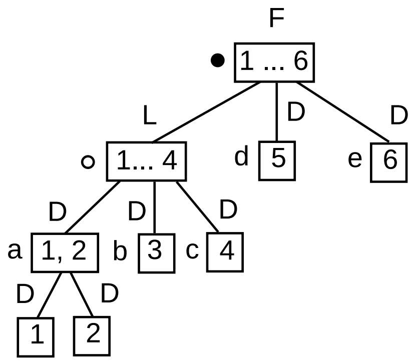

Figure 3. Example of a composition tree

Theorem 4. *Let $f \in \Sigma$ and A, B be bound sets for f that overlap, i.e. $C = A - B$, $D = A \cap B$, and $E = B - A$ are not empty. Then*

(a) $A \cup B$ and C, D, E are bound sets,

(b) f has a representation (3) for $x \in M^C$, $y \in M^D$, $z \in M^E$ and remaining variables w where $\bullet$ is an associative function in Σ, which is commutative if and only if $C \cup E$ is a bound set.

In Theorem 4, A and B are not strong bound sets, but $A \cup B$ and its partition classes C, D, E may be. In that case, the node $A \cup B$ of the composition tree is labeled with the function $a \bullet b \bullet c$ of the three variables a, b, c (with values in M). In fact, it suffices to store with the node $A \cup B$ the binary function $\bullet$ (with m^2 values) and to specify a *linear order* among its children showing how to take the product with $\bullet$, like $a \bullet b \bullet c$ since the order $b \bullet a \bullet c$ is not allowed if $\bullet$ is not commutative. If $\bullet$ is commutative, then the order of taking the product is irrelevant. For $m > 2$, Σ may have non-commutative associative operations $\bullet$ like

$$a \bullet b = \begin{cases} a & \text{if } b = 0 \\ b & \text{if } b > 0. \end{cases}$$

For example, f in (4) may also have the overlapping bound sets $\{1, 2, 3\}$ and $\{3, 4\}$, which are bound sets for h by Lemma 3(a). Then Theorem 4 asserts

$$\begin{aligned} h(a(x_1, x_2), x_3, x_4) &= \hat{h}(a(x_1, x_2), b(x_3), c(x_4)) \\ &= a(x_1, x_2) \bullet b(x_3) \bullet c(x_4) \end{aligned} \tag{5}$$

with an associative binary operation $\circ$. The unary functions b and c are bijections which in general are necessary for this representation (but would be unnecessary if h was prime). The functions h and $\hat{h}$ in (5) are called *isotopic* since they are identical except for such bijections $M \to M$ of variable or function values. Isotopy leaves decompositions *invariant*: If f, g, or h in (1) is replaced by an isotopic function, then the other functions can be replaced by isotopic functions such that (1) still holds. As a stronger notion, two, say, binary operations $*$ and $\bullet$ are called *isomorphic* if there is a bijection $\phi \colon M \to M$ such that $\phi(a * b) = \phi(a) \bullet \phi(b)$.

A *maximal* bound set is an inclusion-maximal bound set not equal to N. Then either

$$\text{all maximal bound sets are pairwise disjoint,} \tag{6}$$

or

$$\text{two maximal bound sets } A, B \text{ overlap.} \tag{7}$$

In (7), $A \cup B = N$ because of Theorem 4(a). Starting with this case distinction, one can show the following.

Theorem 5. *Let $T(f)$ be the composition tree of f given by the strong bound sets as nodes, related by inclusion. Any node of the tree can be labeled "disjoint" (which holds for all nodes with at most two children) or "full" or "linear", such that A is a bound set for f if and only if*

(a) A is a node of the tree, or

(b) A is the arbitrary union $\bigcup_{i \in L} B_i$ of the children $B_1, \ldots, B_k$ of a "full" node, $\emptyset \neq L \subseteq \{1, \ldots, k\}$, or

(c) A is the union $\bigcup_{i=j}^{l} B_i$ of an interval of the children $B_1, \ldots, B_k$ (specified in that linear order) of a "linear" node, $1 \leq j \leq l \leq k$.

The children of a "disjoint" node A are the maximal, pairwise disjoint bound sets contained in A as in (6). We assume $k \geq 3$ in (b) and (c) to distinguish these cases from a "disjoint" node. The number of bound sets may be exponential, as in (b), but they are efficiently coded by this labeled tree, which has size linear in n [10].

Theorem 6. *Let $T(f)$ be the composition tree of $f \colon M^N \to M$ and $B_1, \ldots, B_k$ be the children of the root N. Then*

$$f(y_1, \ldots, y_k) = g(h_1(y_1), \ldots, h_k(y_k)) \tag{8}$$

for functions $h_i : M^{B_i} \to M$ $(1 \leq i \leq k)$ and $g \colon M^k \to M$ in Σ where

(a) g is prime if N is labeled "disjoint",

(b) $g(a_1, \ldots, a_k) = a_1 \bullet \cdots \bullet a_k$ (for $a_i \in M$, $1 \leq i \leq k$) with an associative and commutative operation in Σ if N is labeled "full",

(c) $g(a_1, \ldots, a_k) = a_1 \bullet \cdots \bullet a_k$ with an associative and non-commutative operation in Σ if N is labeled "linear".

(d) In (a), g is unique up to isotopy. In (b) and (c), $\bullet$ is unique up to isomorphy.

This is the main representation theorem of [13]. It is applied by induction, which ends if f is prime. Using the trees $T(h_1), \ldots, T(h_k)$ with the children $B_1, \ldots, B_k$ of N as roots, (8) gives the fully decomposed representation of f. By Theorem 5, the resulting hierarchical term contains any decomposition (1) as a subterm, where $h(x)$ is possibly obtained by suitable arrangement of "products" with an operation $\bullet$.

An example of a composition tree is shown in Figure 3. Abbreviations "D", "F" and "L" stand for labels "disjoint", "full" and "linear", respectively. The linear order among the children of the "L" node is from left to right. Letters a, b, c, d, e denote the functions associated with the nodes, and $\bullet$ and $\circ$ denote the operations. In accordance with the tree, the complete decomposition of the function f is

$$f(x_1, \ldots, x_6) = (a(x_1, x_2) \circ b(x_3) \circ c(x_4)) \bullet d(x_5) \bullet e(x_6)$$

with $\bullet$ being an associative and commutative operation and $\circ$ being associative and non-commutative.

4. Constructing Composition Trees

Assume an m-valued n-variable f is specified in some way, and $f \in \Sigma$. We want to find its composition tree $T(f)$. The problem of finding a simple disjunctive decomposition, i.e. determining a bound set for an m-valued function f and obtaining suitable functions g and h in $f(x,y) = g(h(x),y)$, is widely studied with a number of algorithms developed for its solution including those in [8], [9], [14]. So we assume the existence of the following function:

IsBoundSet(A, f, N)
input: $f\colon M^N \to M$ in Σ, $A \subseteq N$
output: "true" if A is a bound set for f,
"false" otherwise.

We assume this function has worst-case time complexity m^n. One may hope that the output "false" is produced faster, if more than m columns in the decomposition chart are detected early. As long as all arguments of f have to be evaluated, m^n is the worst-case running time [10]. It might be interesting to analyze the *expected* time needed to recognize that a "random" function is prime, with all caveats that such functions are not those used in practice.

We check successively all sets A with cardinality $n-1$, $n-2, \ldots, 2$ until a bound set is found. If f is prime, this requires $2^n - n - 2$ many calls to **IsBoundSet** and overall running time $O(2^n m^n)$. However, this is the worst-case time complexity. It will become apparent that there is a significant speedup as soon as decompositions are found.

If f is prime, then we return

TrivialTree(f, N)

which is just one node if N is a singleton, otherwise root N with children $\{i\}$ for $i \in N$. The root is labeled "disjoint" and with the function f to take the place of g in (8), and the children $B_i = \{i\}$ are labeled with $h(x_i) = x_i$.

If a bound set A is found, then it is maximal since we examine the largest subsets of N first. We construct a specification of h and g in (1), for example by Lemma 1. Recall that for that purpose, it is useful to know explicitly where f is fully sensitive to all its variables.

If A is not overlapping with any other bound set, then A is by definition a node of the composition tree $T = T(f)$. Using Lemma 3, we can recursively compute

$$T_1 = T(h), \qquad T_2 = T(g) \tag{9}$$

and return

$$T = \textbf{Append}(T_1, A, T_2, \{i\}, \overline{A}).$$

The function **Append** takes two trees, T_1 with root A and T_2 with root $\{i\} \cup \overline{A}$, as input and returns T obtained by replacing the leaf $\{i\}$ of T_2 by T_1, and new root $A \cup \overline{A}$. The labels of the nodes are not changed. Then Lemma 3 and Theorem 6 imply $T = T(f)$.

However, this is not correct if another bound set B overlaps with A, as in (7). Then $A \cup B = N$ because A is maximal, hence $\overline{A} = N - A = B - A$ and Theorem 4(a) implies:

$A - B$ and $A \cap B$ are bound sets for f,
$\overline{A}$ is a bound set for f.

By Lemma 3(a) and (c), these necessary conditions are equivalent to

$A - B$ and $A \cap B$ are bound sets for h,
$\overline{A}$ is a bound set for g.

As before, these conditions can be verified from the composition trees T_1 and T_2 of h and g in (9) if these are computed recursively first. The conditions fail if the root of either tree is labeled "disjoint" and has three or more children, by (6). Otherwise, we invoke

PossiblyMerge$(T_1, A, T_2, \{i\}, \overline{A})$
input: composition trees $T_1 = T(h)$ with root A and $T_2 = T(g)$ with root $\{i\} \cup \overline{A}$. Assume (1).
output: If no bound set B for f overlaps with A, then the same as **Append**$(T_1, A, T_2, \{i\}, \overline{A})$.
Otherwise, $T = T(f)$ with the roots of T_1 and T_2 merged, the leaf $\{i\}$ of T_2 omitted, and new root $A \cup \overline{A}$ labeled "full" or "linear".

We describe this procedure in more detail. The root of $T_2 = T(g)$ is labeled "disjoint" since $\{i\}$ is a maximal bound set for g (by Lemma 3(b), since A is maximal for f). Suppose the root of T_2 has two children $\{i\}$ and $\overline{A}$, and the root of $T_1 = T(h)$ has two children C, D. Then the functions G and H for these roots (which are stored with the trees) show

$$\begin{aligned} g(h,y) &= G(h, c(y)), & h &\in M,\ y \in M^{\overline{A}} \\ h(u,v) &= H(a(u), b(v)), & u &\in M^C,\ v \in M^D \end{aligned} \tag{10}$$

so that $f(u,v,y) = G(H(a(u),b(v)),\ c(y))$. We have to check for the possibility that G and H are isotopic to an operation $\bullet$. Rather than trying out the bijections $M \to M$ for verifying such an isotopy, we test if $B = D \cup \overline{A}$ (i.e. the variables v, y) and $B = C \cup \overline{A}$ (i.e. the variables u, y) are bound sets for $f(u,v,y)$. It is not necessary to apply this test to f. Equivalently, we test if $\{b,c\}$ and $\{a,c\}$ are bound sets for the function $F(a,b,c) = G(H(a,b),\ c)$ of the three variables a, b, c with values in M. This can be done quickly in time m^3. Then if

(i) $\{b,c\}$ and $\{a,c\}$ are bound: merge the roots and label the new root $A \cup \overline{A}$ "full", with children $C, D, \overline{A}$.
(ii) $\{b,c\}$ is bound, $\{a,c\}$ is not: merge, with label "linear" and children $C, D, \overline{A}$.
(iii) $\{a,c\}$ is bound, $\{b,c\}$ is not: merge, with label "linear" and children $D, C, \overline{A}$.

(iv) Otherwise, just append T_1 to T_2.
In cases (i)–(iii), the root obtained by merging is labeled with the operation $\bullet$, which can be found similar to Lemma 1 [13]. Furthermore, it may be necessary to apply a bijective transformation to the values of the functions of the children, as when changing from h to $\hat{h}$ in (5).

PossiblyMerge is also applied if the root of T_1 is labeled "full" or "linear" with children $B_1, \ldots, B_k$. In that case, we let $C = B_1$ and $D = B_2 \cup \cdots \cup B_k$, and let H be the operation $\bullet$ at the root of T_1, so that (10) holds, and proceed as before. In case (ii), the children of the new root are $B_1, \ldots, B_k, \overline{A}$. In case (iii), they are $B_k, \ldots, B_1, \overline{A}$. In any case, **PossiblyMerge** has time complexity $O(m^3)$.

The great advantage of the recursion (9) is that it saves computation time. If $|A| = p$ as in Figure 1, each test of a bound set for h or g with **IsBoundSet** requires up to m^p or m^{1+n-p} many steps, much fewer than the m^n steps for f.

For the computation of $T_2 = T(g)$, some care is necessary to avoid duplicate computations. First, $\{i\}$ is a leaf of T_2, so only subsets of $\overline{A}$ have to be checked. Second, subsets D of $\overline{A}$ with $|D| > p$ can also be disregarded since they have already been checked for f and Lemma 3(c) holds. (Even certain subsets D of $\overline{A}$ with $|D| = p$ may have been checked before A, and can be disregarded by a careful implementation of the algorithm.) The second point is relevant only when $p \le |\overline{A}| = n - p$, i.e. $p \le n/2$. In that case (which includes $p = |\overline{A}|$) it is also unnecessary to invoke **PossiblyMerge** since then $|B| > p$ for the bound set B of f that is sought there, which is not possible.

In order to compute T_2 efficiently, we therefore pass $S := \overline{A}$ and p as additional parameters to the algorithm, which looks as follows. Preconditions and assertions at various stages are given in $\langle \ldots \rangle$. The procedure terminates at each **return** statement with the indicated output.

CompositionTree(f, N, S, p)
input: $f\colon M^N \to M$ in Σ, $S \subseteq N$, integer p
assumption: $\langle A \subseteq S$ and $|A| \le p$ for any bound set A of f, $A \ne N \rangle$
output: the composition tree $T = T(f)$
initial call: **CompositionTree**($f, N, N, n-1$).

1. **if** $|N| \le 2$ **then**
    return $T :=$ **TrivialTree**(f, N);
2. **while** $p > 1$
    for all $A \subseteq S$ with $|A| = p$
        if IsBoundSet(A, f, N) **then** goto 3;
    end for;
    $p := p - 1$;
  end while;
  $\langle f$ is prime $\rangle$
  return $T :=$ **TrivialTree**(f, N);
3. $\langle A$ is a bound set for f, $|A| = p \rangle$
  let $f(x, y) = g(h(x), y)$ as in (1), $i \in A$
    $T_1 :=$ **CompositionTree**($h, A, A, p-1$);
    $T_2 :=$ **CompositionTree**($g, \{i\} \cup (N - A)$,
        $S - A$, $\min\{p, |S - A|\}$);
4. **if** $N \ne S$
    or $p \le |N| - p$
    or T_1 or T_2 has a "disjoint" root with
        more than two children
  then
    $T :=$ **Append**($T_1, A, T_2, \{i\}, N - A$)
  else
    $T :=$ **PossiblyMerge**($T_1, A, T_2, \{i\}, N - A$);
  return T.

When computing T_2, we exploit that T_2 has a "disjoint" root when we invoke **CompositionTree** with third parameter $S := S - A$ rather than $N - A$, since all elements of $N - A$ (as singletons) will be children of the root. For example, suppose $N = 12345678$ (as shorthand for $\{1, \ldots, 8\}$ with disjoint maximal bound sets 123, 45, 67, 8. Assume $i \in A$ in step 3 is the first element of A. Then the parameters N, S of **CompositionTree** for computing T_2 are
  after 123 is found: $N = 145678$, $S = 45678$,
  after 45 is found: $N = 14678$, $S = 678$,
  after 67 is found: $N = 1468$, $S = 8$.
Similarly, the test $N \ne S$ in step 4 reveals if the current computation is for some tree T_2.

To illustrate the recursive calls for computing T_1, consider Figure 3 where $N = 123456$. In succession, we find the bound sets 12345, 1234, 123, 12. (In this example, T_2 is therefore always the trivial tree by step 1.) Let $k(x_1, x_2, x_3)$ be the function with root 123, $k(x_1, x_2, x_3) = K(a(x_1, x_2), x_3)$. Because K and a are binary functions, **PossiblyMerge**($T(a), \{1, 2\}, T(K), \{1\}, \{3\}$) is called, which is the same as **Append** since (case (iv) above) neither $\{2, 3\}$ nor $\{1, 3\}$ are bound sets for k. After this recursion terminates, the function l for node 1234 is checked with, say, **PossiblyMerge**($T(k), \{1, 2, 3\}, T(L), \{1\}, \{4\}$) which merges 123 into 1234 (case (ii) above), making this a "linear" node. Next, via **PossiblyMerge**, case (iv), 1234 is just appended to 12345, and then 12345 is merged via case (i) into 123456 which is labeled "full".

The algorithm exploits the structure of bound sets as stated in Theorems 4 and 5, and there seems to be no obvious way to do this better. Furthermore, it is readily adapted to bound sets A that are apparent from a modular specification of the function, where h and g in (1) are explicitly given. After computing $T_1 = T(h)$ and $T_2 = T(g)$, these trees could possibly be merged. The above procedure **PossiblyMerge** can easily be modified for that purpose so that it works with arbitrary composition trees T_2.

Making use of decompositions that are found along the way reduces the running time substantially. If f is composed only of binary functions, for example, then f has a bound set of size $n - 1$, which is found after nm^n or fewer steps, and the same holds for any function h (with correspondingly smaller number n of variables). Thus, the complete tree is computed in $\sum_{i=3}^{n} im^i = O(nm^n)$ time. The expensive cost m^n of finding a bound set is still there, but with a factor of n rather than 2^n for a non-decomposable function.

5. Conclusions

This paper summarizes the results on composition trees from [13] restricted to the case of m-valued n-variable functions, defining composition trees and specifying a class of functions Σ for which the composition tree exists. The composition tree provides a representation for the function showing all its disjunctive decompositions, which substantially reduce the overall cost of realizing the function. The class Σ is connected to full sensitivity, a concept introduced independently in [5] as the discrete difference for test generation purposes. This shows the practical importance of the class Σ, and allows to apply the results obtained for test generation as well as the decomposition theory.

Some extension of the algorithm seems desirable. First, since the class Σ does not include all functions for which the composition tree can be built, the present algorithm is incapable of constructing the composition tree for the decomposable functions which are not in Σ. An open problem remains how to generalize this class so that it includes all decomposable functions. Second, the functions included in class Σ are restricted to homogeneous functions only. If the theory developed in [13] can be extended to the case of heterogeneous functions, then it would have a direct application to Boolean circuit synthesis, since it would cover as a special case the decomposition of type

$$f(x, y) = g(h(x), y)$$

with $f: \{0,1\}^n \to \{0,1\}$, $h: \{0,1\}^p \to \{0,1,...,m-1\}$ and $g: \{0,1,...,m-1\} \times \{0,1\}^{n-p} \to \{0,1\}$. In such a decomposition the m-valued function h of 2-valued variables can be coded by $k = \lceil \log_2 m \rceil$ Boolean functions $h_1, h_2, \ldots, h_k$, giving a decomposition of the form

$$f(x, y) = g(h_1(x), h_2(x), \ldots, h_k(x), y) \qquad (11)$$

with all functions being Boolean. The decomposition of type (11) includes as a subclass simple disjunctive decompositions ($k = 1$) as well as nondisjunctive decompositions. As long as f is a function of more than three variables, such a decomposition can always be found with $h_1(x), h_2(x), \ldots, h_k(x)$ and g each having fewer arguments than f, for there always exists a decomposition of the form

$$f(x_1, \ldots, x_n) = f(z, x_n) = g(h_1(z), h_2(z), x_n)$$

with $z = (x_1, \ldots, x_{n-1})$. Thus, a decomposition (11) allows simplifying *any* Boolean function. Therefore, a theory of composition trees for this extended case would be a base for systematic synthesis of multi-level Boolean circuits.

References

[1] R. L. Ashenhurst. The decomposition of switching functions. *Proc. Int. Symp. Theory of Switching*, Part I, Ann. Comput. Lab. Harvard Univ. 29:74–116, 1959.

[2] M. Bolton. *Digital Systems Design with Programmable Logic*. Addison-Wesley Pub. Co., 1990.

[3] S. D. Brown, R. J. Francis, J. Rose and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[4] H. A. Curtis. *A New Approach to the Design of Switching Circuits*. Van Nostrand, Princeton, 1962.

[5] E. V. Dubrova, D. B. Gurov and J. C. Muzio. Full sensitivity and test generation for multiple-valued logic circuits. *Proc. 24th Int. Symp. on MVL*, 284–289, 1994.

[6] E. V. Dubrova, D. B. Gurov and J. C. Muzio. The evaluation of full sensitivity for test generation in MVL Circuits. *Proc. 25th Int. Symp. on MVL*, 104–109, 1995.

[7] R. M. Karp. Functional decomposition and switching circuit design. *J. Soc. Indust. Appl. Math.* 11:291–335, 1963.

[8] D. M. Miller. *Decomposition in Many-Valued Logic Design*. Ph.D. Thesis, University of Manitoba, March 1976.

[9] D. M. Miller and J. C. Muzio. Decomposition and the synthesis of many-valued switching circuits. *Proc. 1976 Int. Symp. on MVL*, 164–168, 1976.

[10] R. H. Möhring. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and Boolean functions. *Annals of Operations Research* 4:195–225, 1985.

[11] W. Paul. Realizing Boolean functions on disjoint sets of variables. *Theoretical Computer Science* 2:383–396, 1976.

[12] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical J.* 28:59–98, 1949.

[13] B. von Stengel. *Eine Dekompositionstheorie für mehrstellige Funktionen*. Mathematical Systems in Economics, Vol. 123, Anton Hain, Frankfurt, 1991.

[14] K. M. Waliuzzaman and Z. G. Vranesic. Decomposition of multiple-valued switching functions. *Computer Journal* 13(4):359–362, 1970.