

# Higher-Order Flexibilities in Multi-Valued Networks

**Alan Mishchenko**

Department of ECE  
Portland State University, Portland, OR  
alanmi@ee.pdx.edu

**Robert K. Brayton**

Department of EECS  
University of California, Berkeley, CA  
brayton@eecs.berkeley.edu

## Abstract

*Computation of complete flexibility (CF) of a node in the non-deterministic MV network is introduced in [10]. CF describes the freedom to modify a node without violating the functionality of the network. It is defined for a node assuming a fixed representation of other nodes. CF is a first-order flexibility because it allows for optimization of one node at a time.*

*In this paper, we extend the concept of CF to a group of nodes. The higher-order flexibility expresses the freedom to modify several nodes simultaneously without violating the functionality of the network. We show how this flexibility can be used to solve optimization problems arising in binary and multi-valued logic synthesis. In particular, it leads to a more efficient merging algorithm, an improved *full\_simplify* procedure, and certain advantages in decomposition.*

*A preliminary experimental evaluation in the environment of MVSIS [12] suggests that the computation and use of higher-order flexibilities is feasible for a variety of benchmarks.*

## 1 Introduction

Multi-level logic optimization is computationally hard. Exact methods are known to work only for very small networks [4]. Thus, all currently used methods are heuristic. They are based on incremental improvements to the Boolean network performed by applying operations such as node elimination, common logic extraction, resubstitution, and simplification using don't-cares [2].

Optimization of logic networks using don't-cares has a long history starting from the use of subsets of satisfiability don't-cares in [2], through the use of compatible observability don't-cares [13] implemented in SIS [14], to recent extension of compatible [6] and complete [10] don't-cares to MV networks.

Although don't-care computation methods take into account the context of a node in the network, they simplify

only one node at a time. However, simultaneous changes to the functionalities of two or more nodes could increase the optimality of results.

One of the known approaches to simultaneous optimization of several nodes uses "Sets of Pairs of Functions to be Distinguished" (SPFDs) [15]. However, this approach has a high computational complexity inherent in the processing of SPFDs.

The main contribution of the present work is the extension of the first-order flexibility (called "complete flexibility (CF) at a node" in [10]) to higher-order flexibilities. A higher-order flexibility expresses the freedom to modify several nodes simultaneously without violating the functionality of the network.

We show how higher-order flexibilities impact a number of problems arising in binary and multi-valued logic synthesis. In particular, they lead to a more efficient merging algorithm, to an improved *full\_simplify* procedure, and to certain advantages in decomposition. The new approach appears to be computationally efficient because it is based on the BDD representation of Boolean relations, efficient heuristic MV-SOP minimization methods, and also has links to Boolean satisfiability.

The present work considers the general case of multi-valued networks. We also show that the proposed methods can have an immediate bearing on binary circuits resulting in more efficient logic synthesis, compared to purely binary optimization methods [2][13].

The theoretical foundations of this work are related to the research in Boolean relations [17]. However, previous formulations did not result in computationally efficient procedures, in contrast to those proposed in this paper.

The rest of the paper is organized as follows. Section 2 supplies necessary background. Section 3 presents an algorithm to compute higher-order flexibilities. Section 4 explains the connections to logic optimization. Section 5 presents improved *merge* and *full\_simplify* procedures. Section 6 gives experimental results (not yet). Section 7 concludes the paper and outlines future work.

## 2 Background

The material of Sections 2.1-2.3 is borrowed from [10]. The reader familiar with [10] may skip to Section 2.4.

### 2.1 Multi-Valued Relations

**Definition.** Let the domain of a multi-valued variable  $a_i$  be denoted as  $D_{a_i}$ . An *MV relation*  $R$  with MV input variables  $\{a_i\}$  and an MV output variable  $z$  relates input minterms to output values:

$$R(a_1, a_2, \dots, a_n, z): D_{a_1} \times D_{a_2} \times \dots \times D_{a_n} \times D_z \rightarrow \{0, 1\}.$$

**Definition.** The set of minterms of the combined input domains of relation  $R$ , where the output takes only one value, is the *care set* of  $R$ . The set of minterms, where the output is the set of all possible values, is the *don't-care set* of  $R$ . The set of other minterms is the *partial care set* of  $R$ .

**Definition.** If the total domain of  $R$  is the care set,  $R$  is a *completely specified* MV function. If the total domain of  $R$  consists of the care set and the don't-care set, with no partial care set,  $R$  is an *incompletely specified* MV function. If the domain of  $R$  contains at least one partial care minterm,  $R$  is a *partially specified (non-deterministic)* MV relation.

**Example.** Shown in Figure 1 are three ternary MV relations depending on binary variable  $a$  and ternary variable  $b$ . Relation  $R_1$  is completely specified. Relation  $R_2$  is incompletely specified. Relation  $R_3$  is partially specified, or non-deterministic.

	$R_1$		$R_2$		$R_3$
$b \backslash a$	0 1	$b \backslash a$	0 1	$b \backslash a$	0 1
0	0 2	0	0,1,2 2	0	0,1 1,2
1	0 0	1	0 0	1	0,1 0
2	1 0	2	0,1,2 0	2	0,1,2 0,1

Figure 1. Example illustrating types of MV relations.

**Definition.** The *i-set* of MV relation  $R$  is the MV-input binary-output function  $F_i$  defined over the input domain of  $R$  and taking value 1 for those minterms where the set of output values of  $R$  **contains** value  $i$ .

Note that the *i-sets* of a completely specified MV function are pair-wise disjoint. The *i-sets* of an incompletely or partially specified relation can overlap.

**Definition.** MV relation  $R_1$  is *contained in* MV relation  $R_2$  (denoted  $R_1 \leq R_2$ ) if, for each minterm of the input domain, the output values of  $R_1$  are a subset of the output values of  $R_2$ .

**Example.** In Figure 1,  $R_1 \leq R_3$  is true, but  $R_2 \leq R_3$  does not hold, because for the minterm (0,0), the value set of  $R_2$ , {0,1,2}, is not a subset of the value set of  $R_3$ , {0,1}.

**Definition.** A *multi-valued sum-of-products* (MV-SOP) is the representation of an MV function, in which each *i-set* of

the MV function is represented by a set of MV-input binary-output cubes. The *i-set* having the most cubes is marked as the *default i-set* (or the *default value*).

To reduce the total number of cubes needed to represent the MV-SOP, one *i-set* is selected as the default *i-set* and is not stored but is computed on demand by complementing the sum of the cubes belonging to the other *i-sets*.<sup>1</sup>

### 2.2 Multi-Valued Networks

**Definition.** An *MV network*  $N$  is a directed acyclic graph with nodes represented by MV relations.<sup>2</sup> The sources of the graph are the *primary inputs* of the network. There is one dummy sink whose inputs are the *primary outputs*.

We typically name the nodes and their output signals the same. The output of a node may be an input to any number of other nodes called its *fanouts*. The inputs of a node are called its *fanins*.

**Definition.** An MV network is *non-deterministic* if any of its outputs is non-deterministic as a function of the primary inputs. If all internal nodes are functions, then the network is deterministic.

The Cartesian product of the MV domains of the fanin variables is the *local space*<sup>3</sup> of a node. The MV relations of the nodes expressed in the local space are the *local relations*. The Cartesian product of the MV domains of the primary input variables is the *primary input space*. The MV relations of the nodes expressed in the primary input space are the *global relations*.

To compute the global relations, the networks can be traversed in a depth-first manner and each local relation of a node is composed with the global relations of its fanins. The global relation of a primary input is the single-variable function representing that input.

**Definition.** A transformation *changes the functionality* of the MV network if the global relation of any primary output is not contained in the original specification. Note that this definition does allow the set of behaviors represented by the network to decrease, as long as all the behaviors of the new network are contained in the original set of behaviors.

In this paper, transformations are not allowed to change the functionality of the network. Removing a node that does not fanout and is not a primary output is a trivial example of such a transformation. In general, it is possible to consider transformations that do change the functionality if later followed by a transformation, which brings the network back into conformity.

<sup>1</sup> To generally be able to represent non-determinism at a node, all *i-sets* have to be represented. Even for binary networks, non-determinism has been used for the initial specification. For example, in some RTLs, it is possible to specify don't cares at internal nodes.

<sup>2</sup> Normally, a network is represented by functions at the internal nodes.

<sup>3</sup> The terms "domain" and "space" are used interchangeably.

## 2.3 Flexibility at a Node

**Definition.** A *flexibility* at node  $Z$  of an MV network  $N$  is an MV relation  $R_Z^f$  such that replacing the current relation  $R_Z$  of node  $Z$  by any relation contained in  $R_Z^f$  does not change the functionality of  $N$ .

**Definition.** A flexibility  $R_Z^f$  at a node  $Z$  is *complete* if it is impossible to add another output value to any input minterm while preserving the flexibility property of  $R_Z^f$ .

In other words, adding a value to any input minterm of  $R_Z^f$  creates too much freedom for the modification of the node relation; modifying the node using this freedom leads to a network not functionally contained to the original; thus by definition,  $R_Z^f$  is no longer a flexibility at node  $Z$ .

**Definition.** A set of flexibilities at a set of nodes is *compatible* if performing simultaneous replacement of the node relations by any relations contained in the respective flexibilities does not change the functionality of the network.

A compatible flexibility at a node is a subset of the complete flexibility at that node.

The flexibilities introduced and computed in [6] are compatible. They can be pre-computed and used independently at each node, but they are not complete and therefore may result in networks with inferior results.

## 2.4 Flexibility of a Set of Nodes

The complete flexibility (CF) at a node is a first-order flexibility, which takes into account the context of one node in the network. The definition of the CF of one node can be extended to a set of nodes.

**Definition.** The *product* of relations  $R_1, R_2, \dots, R_k$  is the relation  $R$ , containing the union of all the input-output combinations that belong to all of the relations  $R_1, R_2, \dots, R_k$ . In terms of the BDDs representing the relations, it is the product of the BDDs.

**Definition.** A *flexibility* of a set of nodes  $S = (Z_1, Z_2, \dots, Z_k)$  in an MV network  $N$  is an MV relation  $R(Y, Z)$  over the fanin variables,  $Y$ , and output variables,  $Z$ , of the set of nodes  $S$ . Relation  $R(Y, Z)$  is such that replacing any set of current relations  $\{R_{Z_i}\}$  of the nodes in  $S$  by any set of relations  $\{R_{Z_i}'\}$ , whose product is contained in  $R(Y, Z)$ , does not change the functionality of  $N$ .

**Definition.** The flexibility  $R(Y, Z)$  of a set of nodes  $S$  is *complete* if it is impossible to add another output value to any input minterm of the relation while preserving the flexibility property of  $R$ . The complete flexibility of the set of nodes is called the *complete mutual flexibility* (CMF).

**Definition.** Given a set of MV nodes  $S$ , the *merged node* is a single MV node, whose input and output spaces are the Cartesian products of the input and output spaces of the original nodes, and whose relation is the product of the

relations present at the individual nodes. If the nodes have shared fanins, these fanins are not duplicated. If among the nodes in  $S$ , there are outputs that are also inputs to other nodes in  $S$ , the corresponding inputs are existentially quantified from the relation.

The merging operation is considered well defined only if replacing the original nodes by the merged node does not produce a combinational loop in the network. A necessary and sufficient condition for this is that whenever a node,  $i$  in  $S$ , is in the transitive fanin of another node,  $j$  in  $S$ , all nodes between  $i$  and  $j$  are also in  $S$ .

## 3 Flexibility Computation

Computation of the CMF of a set of nodes,  $S$ , in an MV network is similar to the computation of the CF for one node [10]. The main steps of this computation are:

- Step 1.** Compute the global relations  $R_i(X, y_i)$  of each PO,  $y_i$ , of the network where  $X$  is the set of PIs.<sup>4</sup>
- Step 2.** Introduce auxiliary variables  $z_i$  at the outputs of the nodes in  $S$  (see Figure 3, left) and compute the global relations  $R(X, Z, y_i)$  of the POs in terms of the PIs and the auxiliary variables  $Z = \{z_i\}$ .
- Step 3.** Express the condition that the relation  $R(X, Z, y_i)$  is contained in the relation  $R(X, y_i)$  for all POs.<sup>5</sup> This condition, seen as a relation with the input variables  $X$  and the output variables  $Z$  of  $S$ , represents the complete mutual flexibility  $CMF^S(X, Z)$  of the nodes from the set  $S$  expressed in the PI space.
- Step 4.** Image  $CMF^S(X, Z)$  from the PI space of the network into the local fanin space of the nodes in  $S$ . This yields the  $CMF^S(Y, Z)$  expressed using the variables,  $Y$ , of the local fanin space of  $S$ .

The proof that these steps produce the CMF of the nodes is essentially the same as the proof of Theorem 1 in [10].

The following theorem relates the  $CMF^S$  of a set of nodes and the  $CF^m$  of the node,  $m$ , created by merging the set  $S$ .

**Theorem 1.** Let  $S = (Z_1, Z_2, \dots, Z_k)$  be a set of MV nodes in network  $N$ , such that the operation of merging is well-defined. Let  $m$  be the node resulting from merging nodes in  $S$ . The CMF of nodes from  $S$  in the network  $N$  is equal to the CF of the node  $m$  in  $N'$  obtained by substituting  $m$  into  $N$  replacing the nodes of  $S$ , i.e.

$$CF^m(Y, z) = CMF^S(Y, D_{Z_1}(z), \dots, D_{Z_k}(z)),$$

<sup>4</sup> More generally, we can start with the functionality of the network given as an I/O specification by a relation  $R(X, Y)$  where  $Y$  is the set of PO variables.

<sup>5</sup> The condition,  $R(X, Z, y_i) \leq R(X, y_i)$  for all PO  $i$ , yields a relation  $R(X, Z)$  where for a minterm in  $X$ , only those outputs  $Z$  are allowed which do not change the functionality of the network, given by the  $\{R(X, y_i)\}$ .

where  $D_{z_i}(z)$  is the decoder converting the auxiliary variable,  $z$ , of the MV node,  $m$ , into the auxiliary variable,  $z_i$ , of a node in the set  $S$ .

**Proof.** Assume that the set of nodes consists of two nodes,  $x$  and  $y$ . The generalization to more nodes is straightforward.

We introduce  $m$  and two decoders,  $d_x$  and  $d_y$ , into the network instead of  $x$  and  $y$ . Their role is to transform the output of  $m$  (also called  $m$ ) into the outputs of  $x$  and  $y$ . Due to the decoders, the rest of the modified network is structurally and functionally identical to the original network (Figure 2).

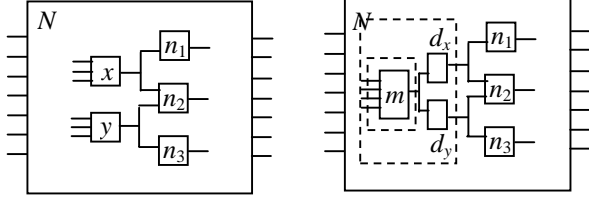


Figure 2. Network before and after merging.

The number of values of the merged MV node  $m$  is equal to the product of the number of values of nodes  $x$  and  $y$ . The  $i$ -sets of the decoders are MV literals transforming  $m$  to produce the signals  $d_x$  and  $d_y$  equivalent to  $x$  and  $y$  in the original network. For example, if  $x$  is ternary and  $y$  is quaternary,  $m$  is 12-valued, has the  $i$ -sets:  $m^0 = x^{(0)}y^{(0)}$ ,  $m^1 = x^{(0)}y^{(1)}$ , ...,  $m^{12} = x^{(3)}y^{(4)}$ , while the decoder  $d_x$  has the  $i$ -sets:  $d_x^0 = m^{(0,3,6,9)}$ ,  $d_x^1 = m^{(1,4,7,10)}$ ,  $d_x^2 = m^{(2,5,8,11)}$ .

The CMF of  $\{x, y\}$  is computed by replacing the outputs of decoders  $d_x$  and  $d_y$  by variables  $z_x$  and  $z_y$ . The CF of node  $m$  is computed by replacing the output of  $m$  by  $z$ . In both cases, the computation is performed by the functionally and structurally equivalent part of the original network. The two computations differ in the part of the network that is between the dashed lines, which contains the decoders (Figure 2, right).

Suppose we computed the CMF of the small network (the outer dashed box of Figure 2, right),  $CMF^{xy}(Y, z_x, z_y)$ , where  $Y$  stands for the inputs and  $\{z_x, z_y\}$  stands for the outputs of the small network. Now, we remap this relation using the functions of the decoders  $D_x$  and  $D_y$  defined for all  $z^0$ :

$$CF^m(Y, z) = \exists z_x z_y [(z_x = D_x(z)) \wedge (z_y = D_y(z)) \wedge CMF^{xy}(Y, z_x, z_y)] = CMF^{xy}(Y, D_x(z), D_y(z)).$$

#### Q. E. D.

The CMF of nodes in  $S$  can lead to a more substantial optimization of the nodes in  $S$  compared to the optimization achieved through the CFs of individual nodes in  $S$ . Using

<sup>6</sup> This formula is just the result of applying the CF computation to the small network (the outer dashed box of Figure 2) consisting of  $Y$  as inputs and  $\{z_x, z_y\}$  as outputs with the external specification given by  $CMF^{xy}(Y, z_x, z_y)$ . We use the fact that  $D_x(z)$  and  $D_y(z)$  are functions and not relations.

the CFs of individual nodes limits the optimizing transformations to those applicable to one node at a time without ever changing the functionality of the network. On the other hand, using the CMF, results in simultaneous modifications of several nodes without changing the functionality of the network. Applying exactly the same modifications to one node at a time would change the functionality of the network and therefore would not be possible if only individual CFs are used.

Simultaneous changes to several nodes in the case of CMF are such that the change in one node violates the functionality of the network, but the error is fixed by changing the second node in such a way that the two modifications taken together satisfy the CMF and therefore preserve the functionality of the network.

To summarize, optimization based on the CMFs leads to "deeper" local minima compared to other don't-care simplification methods [6][13][10]. Optimization based on SPFDs [15] has a similar aspect of changing several nodes at a time but these are not strictly comparable to CMFs. Also, SPFDs involve manipulating relations with a doubled variable space, which is less computationally efficient.

## 4 Flexibility-Based Optimization

The optimization procedures in this section exploit CMFs to achieve simultaneous optimization of several nodes.

### 4.1 MME Algorithm

The Merge-Minimize-Encode (MME) algorithm treats the CMF of a group of nodes as the CF of the merged MV node, which is warranted by Theorem 1. The merged node is minimized using a heuristic MV-SOP minimization method [10]. Finally, the minimized node is encoded to produce new nodes, which can replace the original nodes. The network before and after applying the MME algorithm is shown in Figure 3 for the case of two nodes,  $x$  and  $y$ , in the set replaced by two encoded nodes,  $x_1$  and  $y_1$ . Only the logic inside the dotted box is changed. The intermediate step with the merged MV node is shown in Figure 2 (right).

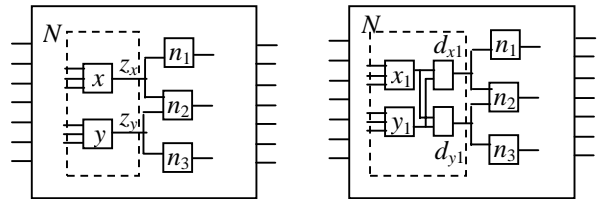


Figure 3. Network before and after applying the MME algorithm.

Note that after re-encoding of the merged node to produce new nodes  $x_1$  and  $y_1$ , the functionalities of the decoders,  $d_x$  and  $d_y$ , originally introduced to accommodate the merged node (Figure 2, right), change to reflect the new encoding.

In particular, the  $i$ -sets of the decoders may no longer be literals but in general, MV functions depending on variables  $x_1$  and  $y_1$ . If the encoding is the same as the original, then the decoders,  $d_x$  and  $d_y$ , disappear.

As an example, if initially we had two binary nodes,  $x$  and  $y$ , the merged node  $m$  is quaternary. After encoding the minimized quaternary node, the new binary nodes,  $x_1$  and  $y_1$ , are created while the decoders,  $d_{x1}$  and  $d_{y1}$ , are in general two-variable Boolean functions,  $d_{x1} = D_{x1}(x_1, y_1)$  and  $d_{y1} = D_{y1}(x_1, y_1)$ .

After minimization, the decoders may be collapsed into their fanouts, or left as additional nodes in the network. The latter solution may be preferable if there are many fanouts. In this case, the decoders may represent good common divisors.

In the above example, both the original and the resulting set of nodes consist of two nodes. In general, the MME algorithm can change the number of nodes as well as the number of their output values. These choices reflect the additional freedom inherent in this type of MV network optimization.

After applying the MME algorithm, we can choose any of the following possibilities: (1) leaving the original nodes unchanged; (2) using the minimized merged node; (3) partially encoding the merged node; (4) completely encoding the merged node (i.e. into binary nodes). The choice may be based on the cost function or on the desired outcome, such as the need to have only binary nodes in the resulting network.

## 4.2 Encoding for the MME Algorithm

To perform a partial or a complete encoding, it may be possible to apply one of the following encoding techniques.

### (1) Bypassing an input to achieve “value reduction”

This technique performs a partial encoding of the ND MV relation  $R$  in such a way that one of the encoding functions is equal to an input  $x$  of the relation (Figure 4). In terms of the original network, this technique corresponds to replacing one of the encoded nodes by a wire. An algorithm for this kind of partial encoding can be found in [11]. The objective is to reduce the number of values that  $v_2$  has compared to  $v$ .

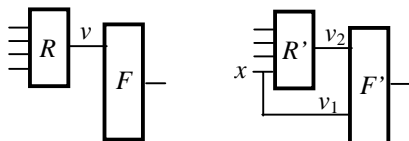


Figure 4. The case when the encoding function is a wire.

## (2) Formulating Encoding as SAT Problem

The encoding problem, in which the resulting cover  $C$  has an upper bound on the number of cubes (see Figure 5), can be formulated as an MV-SAT problem. The problem has two types of variables and two types of clauses.

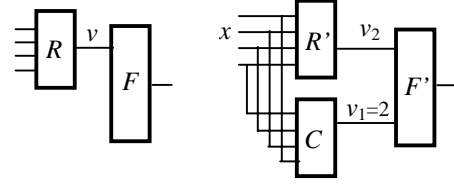


Figure 5. The case when the encoding function is a cube.

SAT variables represent: (a) the literals of the cubes (because of the upper bound on the number of cubes, there is a fixed number of these variables); (b) the occurrence of the output values in each minterm of the truth table representing the CMF.

The SAT clauses are (i) the covering constraints and (ii) the closure constraints. The covering constraints specify that each cell in the map should be covered by at least one cube in the cover. The closure constraints make sure that the ND relation representing the CMF is not violated in any cell. For relatively small encoding problems, this SAT formulation can provide an exact solution.

## 4.3 Combined Minimization of Several MV-SOPs

The heuristic minimization of MV relations presented in [10] minimize the MV-SOP of each  $i$ -set, one at a time starting from the CF at a node. It may be possible to extend this to perform simultaneous minimization of several MV-SOPs bounded by the CMF. One idea for such an extension is to start by computing those minterms of the CMF, for which only one combination of the output values is possible. These minterms can be used as seeds for the essential covers. The remaining minterms are covered greedily using an algorithm similar to [10].

## 5 Applications

### 5.1 Improvements to merge

The operation of merging is defined in Section 2.4. This operation is often used to combine a set of binary nodes into a single MV node, but it can also be applied to set of MV nodes.

Merging involves constructing all possible intersections of the  $i$ -sets of the nodes to be merged. Each intersection represents one value of the merged node. The values corresponding to empty intersections are removed from the value set of the merged node. In the current implementation

of *merge* in MVSIS [12], the *i*-sets of the nodes are taken from the current node representations.

Merging would benefit from using the partial or complete flexibility instead of the current representation of the nodes, because this would give more freedom to optimize the representation of the merged node. Of course, the merge can be done first, and then the CF for the resulting node can be used subsequently to minimize the *i*-set representations of the new node.

Alternately, Theorem 1 allows us to use the CMF of the nodes to be merged as the flexibility of the merged node. This would circumvent having to deal with the default *I*-sets of the nodes to be merged, since they could be very large and are not represented in memory, but computed on demand by complementing the union of the other *i*-sets. The heuristic MV-SOP minimization methods [10] can use the CMF to derive directly, a more compact representation of the merged node, compared to the methods based on the current node representation.

## 5.2 Improvements to *full\_simplify*

The *full\_simplify* procedure based on using the CF [10] iterates through the nodes of the network. For each node, it computes the CF and derives a minimal MV-SOP. If the resulting representation has less SOP literals than the current one, the latter is updated. In the same pass over the nodes, this procedure also performs resubstitution and phase assignment if these cause improved reduction. The number of CF computations performed by *full\_simplify* is linear in the number of nodes. We can think of the *full\_simplify* as a first-order procedure.

A second-order *full\_simplify* considers node pairs. It computes the CMF for each, and tries to improve the representation of both nodes simultaneously using the MME algorithm presented above, or other generalized MV-SOP minimization procedures.

Computing the CMF for a node pair is only marginally more complex than computing the CF for a single node because approximately 75% of runtime is typically spent updating global BDDs. The updating is essentially the same for any number of auxiliary variable  $z_i$  in the network.

For the exhaustive second-order *full\_simplify*, the number of CF computations is quadratic in the number of nodes. Filters can be applied to remove those pairs that are not related, and therefore would not benefit from mutual simplification. Network partitioning can simplify the computation of global relations and further reduce the number of considered node pairs.

Instead of limiting attention to node pairs, we can consider arbitrary sets of nodes to be optimized simultaneously. A potential problem here is a large size of the BDD representation of the CMF. Considering sets of

nodes, as opposed to pairs of node would correspond to applying a higher-order *full\_simplify* procedure.

A higher-order *full\_simplify* can be helpful when the user is willing to spend more time optimizing the network in order to achieve a deeper local minimum. This kind of "intensive treatment" could be used to reduce the logic density along a critical path or in a congested region of the network. In these cases, the number of node pairs to consider is much less than the total number of all pairs.

This method can also be helpful in the process of decomposition, because the CMF establishes a "functionality flow" between the nodes. So, if we cannot decompose a node as it is, we can "channel" some of its functionality to another node, which in some cases can improve decomposability.

## 6 Experimental Results

We will first study binary networks, pre-optimized by *script.rugged*. Then by applying the MME algorithm to subsets of nodes with identical or very similar support sets, we hope to show that this leads to a reduction in the number of literals. This will prove the usefulness of the higher-order flexibilities.

## 7 Conclusions

In this paper, we extended the concept of first-order flexibilities involving one node to higher-order flexibilities involving a set of nodes. We showed the connection between the CMF for a set of nodes and the CF of the corresponding merged node. We demonstrated the possible impact of the new flexibilities on the *merge* and *full\_simplify* procedures and outlined links between encoding and Boolean satisfiability.

The optimization methods developed in this paper are fully applicable to binary networks. Applying the MME algorithm with complete encoding back to a set of binary nodes results in a set of new nodes, which can be simpler than the original nodes. The intermediate merged MV node links binary network synthesis to MV optimization methods. Searching for solutions in a larger MV solution space can facilitate logic optimization for purely binary networks.

## Acknowledgements

The first author has been partially supported by a research grant from Intel Corporation. The second author would like to thank the SRC, the GSRC and the California Micro program and associated industrial sponsors, Cadence and Synplicity, for their support.

## References

- [1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli. Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Dordrecht, 1984.
- [2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, MIS: A Multiple-Level Logic Optimization System, *IEEE Trans. CAD*, Vol. CAD-6, No. 6, pp.1062-1082, November 1987.
- [3] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Comp*, Vol. C-35, No. 8 (August, 1986), pp. 677-691.
- [4] R. Drechsler and W. Gunther. Exact Circuit Synthesis. *Proc. IWLS '98*.
- [5] J. -H. Jiang, Y. Jiang, and R. Brayton. An Implicit Method for Multi-Valued Network Encoding. *Proc. of IWLS'01*, pp. 127-131.
- [6] Y. Jiang and R. Brayton. Don't-Cares and Multi-Valued Logic Network Optimization. *Proc. ICCAD'00*. pp. 520-525.
- [7] A. A. Malik, R. Brayton, A. R. Newton and A. Sangiovanni-Vincentelli. Reduced Offsets for Two-Level Multi-Valued Logic Minimization. *Proc. of DAC' 90*, pp. 290-296.
- [8] S. Minato. Fast generation of irredundant sum-of-products forms from binary decision diagrams. *Proc. of, Kobe, Japan*, pp. 64-73.
- [9] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. *Proc. of DAC '93*, pp. 272-277.
- [10] A. Mishchenko, R. K. Brayton. Optimization of Non-Deterministic Multi-Valued Networks. *Submitted to IWLS 2002*.
- [11] A. Mishchenko, T. Sasao. Encoding of Boolean Functions and Its Application to LUT Cascade Synthesis. *Submitted to IWLS 2002*.
- [12] MVSIS Group. *MVSIS*. UC Berkeley.  
<http://www-cad.eecs.berkeley.edu/mvsis/>
- [13] H. Savoj. R. K. Brayton. The use of observability and external don't-cares for the simplification of multi-level networks. *Proc. of DAC' 90*. pp. 297-301.
- [14] E. Sentovich, et al. "SIS: A System for Sequential Circuit Synthesis", *Tech. Rep. UCB/ERI, M92/41*, ERL, Dept. of EECS, Univ. of California, Berkeley, 1992.
- [15] S. Sinha, R. K. Brayton. Improved Robust SPFD Computation. *Proc. IWLS' 01*.
- [16] F. Somenzi. *CUDD Package, Release 2.3.1*.  
<http://vlsi.Colorado.EDU/~fabio/CUDD/cuddIntro.html>
- [17] Y. Watanabe, L. Guerra and R. K. Brayton. Logic Optimization with Multi-Output Gates. *Proc. ICCD*, 1993, pp. 416-420.