A Theory of Non-Deterministic Networks

Alan Mishchenko and Robert K. Brayton
Department of EECS, University of California at Berkeley
{alanmi, brayton}@eecs.berkeley.edu

Abstract

Both non-determinism and multi-level networks compactly characterize the flexibility allowed in implementing a circuit. A theory for representing and manipulating non-deterministic (ND) multi-level networks is developed. The theory supports all the network manipulations commonly applied to deterministic binary networks, such as node minimization, elimination, and decomposition. It is shown that an ND network's behavior can be interpreted in three ways, all of which coincide when the network is deterministic. Operations performed on an ND network are analyzed under each interpretation for changes in a network's behavior. Modifications of a few operations are given which must be used to guarantee that a network's behavior does not violate its external specification. These modifications depend on which behavior is being used and the location of related non-determinism. This theory has been implemented in a system, MVSIS. We provide comparisons among the uses of the various behaviors.

1 Introduction

A non-deterministic (ND) network is similar to a Boolean network, except that, in general, each node has a multi-valued (MV) output and is represented by a non-deterministic relation. The familiar don't cares used in logic synthesis are a special form of non-determinism. For example, don't cares specify for some input minterms, the output can take *any* of the values in its range, while more generally, non-determinism occurs when, for an input minterm, the output can take values from a *subset* of values in the range of the output.

Non-determinism arises naturally in a synthesis setting. For example, a system's specification may be given by an ND network or automaton. Part of the system may be given also. To be synthesized is an unknown sub-component. The set of all possible behaviors for the unknown can be derived as an ND relation or ND automaton, using complementation and composition operations 0.

In logic synthesis, an initial network representation can be given with "compatible" don't cares at the primary outputs. Some RTLs allow incomplete behavior to be specified at internal nodes. This is interpreted as don't care, i.e. for the unspecified inputs, the output can take any value allowed for the variable. Don't cares can also be derived from a network's functionality in terms of observability (ODC) and satisfiability (SDC) don't cares. Generalization of these concepts to MV networks leads to the more general notion of non-determinism.

Starting from the initial specification, synthesis consists of operating on a Boolean network to obtain a smaller, faster, more efficient one, which finally is mapped into a set of logic gates for implementation in hardware. When these operations are generalized to account for non-determinism, an analogous network and set of operations is desired. The use of such networks can lead to final more efficient deterministic binary implementations, since the generalization to MV ND networks allows a larger space for optimization algorithms to explore [3].

We define three network simulation models (SS, NS, NSC) for ND networks, which lead to three types of network "behaviors". A behavior is defined to be the set of all primary-input primary-output pairs of vectors that can be simulated for the network. All three newly defined behaviors reduce to the same unique behavior if the network is deterministic. In the binary case, one of these simulation models (SS) is analogous to ternary simulation with the three values, {0,1,X} [1]. We analyze how the corresponding ND network behaviors can change under various common network operations, such as decompose, substitute, eliminate, collapse, node minimize, and merge [8]. It was found that some of the classical operations need to be modified to account for the effects of non-determinism. We also study the limits (flexibility), within which the functionality of a node in an ND network can be changed without violating the external specification. For all behaviors, we derive a formula for computing the complete (maximum) flexibility (CF) allowed at the node.

The paper is organized as follows. In Section 2, we define an ND network and give some notation. Section 3 discusses the three methods for interpreting the behavior of an ND network. In Section 4, we give for each behavior type, methods for computing the complete flexibilities (CFs) at a node and show that these cannot increase any of the respective network behaviors when any "well-defined" sub-relation of the CF is used to replace the old relation at the node. Section 5 discusses the node elimination process, Section 6 extraction and decomposition, and Section 7 merging. In each case, we analyze how the respective operations can change the three types of network behaviors. Section 8 discusses the relative merits of the two computationally more viable behaviors, NSC and SS. Section 9 discusses how each of these behaviors can be made to fit into a hierarchical theory where a network can be partitioned and sub-parts can be optimized separately. Section 10 discusses modifications on the two operations (one for NSC and one for SS), which could increase the corresponding behavior, to ensure that the network always satisfies its specification. Section 11 discusses some experimental results which compare the use of the different behaviors in terms of the relative sizes of the flexibilities allowed. Section 12 concludes, summarizing the contributions and listing some longer-term goals for the application of this theory and its implementation.

Because of restricted space, no proofs are given in this paper. However, all results have been proved and the proofs tested against a number of readers. For the proofs, please refer to a more extensive report [5]. In addition, the theory has been implemented in a system, MVSIS, (http://www-cad.eecs.berkeley.edu/mvsis/) and experimental results are consistent with the theory. Our implementation indicates that runtimes penalties incurred for the generalization to MV and non-determinism are minimal. Because most algorithms had to be completely re-implemented, we used this opportunity to improve the efficiency of the algorithms and data-structures. Experiments indicate that runtimes are about 5 times faster than SIS, even though all algorithms have been generalized. The quality of results (when run on binary deterministic examples) is equal to or better than for SIS.

2 ND Networks

An ND network is an acyclic directed graph. A node represents an ND relation between the node's inputs and its one output. An edge is directed from node i to j if the relation at node j depends syntactically on the variable y_i, associated with the output of node i. The output of node i is multi-valued and takes values from the domain $D_i = \{0,\cdots,n_i - 1\}$.

Primary inputs (PI) are nodes with no inputs. Primary output nodes (PO) deliver the functionality of the network to its environment. Single input and output storage element nodes have the next state (NS) variables as inputs, and the present state (PS) variables as outputs. Since this paper is concerned only with the combinational portion of the network, the set (PI, PS) is called the combinational inputs (CI) and represented by the vector X, and the set (PO, NS) is called the combinational outputs (CO) and represented by the vector Z.

An external specification of a network, $R^{spec}(X,Z)$, is the set of all acceptable (CI, CO) minterm pairs, (m_X, m_Z), such that $R^{spec}(m_X, m_Z) = 1$ if and only if the pair (m_X, m_Z) is allowed.

Definition: *A relation* $R(X,Z)$ *is* well-defined *if for each input minterm, there exists at least one allowed output minterm in the relation:* $\forall_X \exists_Z (R(X,Z)=1)$.

Definition: *Let* . $R(X,Z)$ *is* output-symmetric *if for any* m_X ,

$$m_Z \in S_1(m_X)\times\cdots\times S_m(m_X) \Rightarrow (m_X, m_Z) \in R(X,Z),$$

where $S_i(m_X) \equiv \{ v \mid \exists_{z_j, j\neq i} R(m_X, z_1,\cdots, z_{i-1}, v, \cdots, z_N) = 1\}$.

Example. Consider a network with two binary outputs, z_1 and z_2. Suppose, for some minterm, the values that the outputs can take are {00, 01}. The relation $R(X, Z)$ is output-symmetric for this minterm, because S_1={0}, S_2={0,1}, and every combination from the product set {0}×{0,1}={00, 01} belongs to the relation. If the same outputs were to take values {00, 01, 11} for this minterm, it would not be output-symmetric because S_1 ={0,1}, S_2 ={0,1}, and there exists a combination {10} in product set, {0,1}×{0,1}={00, 01, 10, 11}, which does not belong to the relation.

Output symmetry has been used to define "compatible" external don't cares in binary networks. The primarily reason its use is that the choice of value made at one output is independent of the choice made at any other output. For a general relation, a choice made at one output, can restrict the choices allowed at another output, and this makes it much harder to deal with.

An ND relation giving the functionality of a node in a network can be specified by the characteristic function relating the inputs and output of the node. The relation at a node j in the network is denoted $R_j(Y_j, y_j)$ where Y_j is the set of fanin variables, and y_j is the single output variable of the node. For ease of notation, sometimes the arguments of a relation will be used to identify it, e.g. $R(X, Y_j)$ and $R(Y_j, y_j)$ denote different relations even though each is named R.

A relation with a single output is often stored as a *set* of deterministic multi-valued input, binary output functions, the ith of which is 1 for those fanin minterms that can produce value i at the output. These are called the i-sets of the relation and each can be represented in SOP (MV) form or as a MDD. Binary-output, MV-input functions can be minimized using a program like Espresso-MV [2][7], resulting in a minimized MV sum-of-products (MVSOP) expression. A product term in an MVSOP is the conjunction of MV-literals. An MV-literal of a variable y, for example, y^S, is the binary function, which is 1 if and only if y has a value in the set of values S.

A smaller MVSOP representation of the relation can be obtained by designating one of the i-sets as a default, which is defined as the complement of the other i-sets.[1] For binary relations, any overlap between the 0-set and the 1-set is called a don't care set, which is typically represented as a separate binary function. In a Boolean network, the 0-set is usually taken as the default and don't cares are derived from the network structure (SDCs and ODCs).

The notation $R^{spec}(X,Z)$ will be used to represent the specification of the network. An *output symmetric* specification has the advantage that it can be represented by a set of individual single-output relations, one for each CO, i.e. $R^{spec}(X, z_i), i = 1,\cdots, N$.

In the next section, we define three types of simulations for an ND network, {NS, NSC, SS}, all of which are the same as the usual notion of simulation when the network is deterministic.

Definition: *The* B-behavior *of an ND network is the set of all CI/CO pairs that can be simulated using the simulation of type*

[1] Note that a general ND relation cannot be fully represented this way because there may be no i-set that is disjoint from the union of the others.

$B, B \in \{NS, NSC, SS\}$. *The B-behavior of a network is denoted as* $R^B(X,Z)$.

Definition: *An ND network B*-conforms to, *or B*-complies with, *its external specification if* $R^B(X,Z) \subseteq R^{spec}(X,Z)$.

3 Behaviors of ND Networks

Each interpretation of the behaviors to be defined for an ND network is associated with a particular type of simulation model. We define three, all of which yield the same behavior if the network is deterministic. The interpretations to be defined are listed in the order of increasing amount of behavior:

1. Normal simulation (NS-behavior).
2. Normal simulation made compatible (output-symmetric) for all outputs (NSC-behavior).
3. Set simulation (SS-behavior).

We define each of the simulation models and discuss their relative merits. In most applications, it is usually appropriate to view NS as the “real” behavior, and the others as easier-to-compute over-approximations.

In manipulating a network, it is important to use only one interpretation of a network’s behavior consistently. This is because in some operations, a network’s behavior is periodically compared with its external specification. Changes are allowed provided they do not cause an increase beyond the specification. Since an ND network can satisfy its specification under one interpretation but not another, switching between different interpretations could lead to a final network that does not conform to its external specification.

3.1 Behavior by Normal Simulation (NS)

NS is the most intuitive type of simulation of an ND network. Proceeding in topological order, each ND node non-deterministically selects one output value allowed by the current fanin minterm, and transmits this value to all of its fanouts. For this type of simulation, it is easy to obtain single pairs (m_X, m_Z) of (CI, CO) minterms of the behavior. However, it is difficult to obtain *all* pairs, which is most often required; in fact, of the three methods, NS is the most computationally complex.

The complete NS-behavior can be obtained by the following computation,

$$R^{NS}(X,Z) = \underset{y_i \in \text{ internal nodes}}{\exists} \prod_j R_j(Y_j, y_j) \qquad (3.1)$$

A pair (m_X, m_Z) is in the MV multi-output relation $R^{NS}(X,Z)$ precisely if m_X is given at the CI, and at each node there exists a **choice** that is propagated to its fanouts, such that finally the vector m_Z appears at the COs.

A more efficient method for computing $R^{NS}(X,Z)$ is to use “early” quantification of a conjunctive relation as it is done in some formal verification applications. Even so, this computation is still problematic since, in general, there is one final relation, which must relate all CIs with all COs. In contrast, the other two types of behaviors to be discussed can be represented by N independent relations, each relating CI vectors, m_X, with one CO, $z_k, k \in \{1, \cdots, N\}$. In these cases, the set of CO vectors related to m_X is the cross product of the sets of values at the individual COs related to m_X. Thus, these two behaviors produce output-symmetric relations.

3.2 Behavior by NS made Compatible (NSC)

In NSC, each CO is simulated independently with NS, obtaining a set of relations:

$$R^{NSC}(X, z_k), \quad k = 1, \ldots, N.$$

Thus $R^{NSC}(X,Z) \equiv \prod_{k=1}^{N} R^{NSC}(X, z_k)$ is output-symmetric (the set $\{R^{NSC}(X, z_k)\}$ is compatible). This increases the behavior over NS since each node that has more than one CO in its TFO is treated independently in each of the simulations for the different COs. This is called *NSC-behavior,* since it represents the operation of making the NS-behavior *compatible.* If each ND node has only one CO in its TFO, then NS and NSC are the same.

Collapsing denotes the process of eliminating[2] all the internal nodes in a network, one by one, in some unspecified order. After a network is collapsed, only the output nodes remain and their relations will depend only on the CI variables, *X*.

Theorem 3.1: *The NSC-behavior is equivalent to collapsing the network in* reverse *topological order.*

It is easy to show that collapsing in reverse topological order yields the smallest set of output-symmetric relations which *contains* the NS behavior of the network. In this sense, it is the smallest easy-to-compute output-symmetric over-approximation of the NS behavior.

The following is a useful observation.

Theorem 3.2: *The NS and NSC behaviors of a network are not changed by eliminating any deterministic node.*

Thus for a deterministic network, the order of elimination during collapsing is not important.

3.3 Behavior by Set Simulation (SS)

Set simulation is performed as follows. Given a minterm m_X, each CI has a single value (singleton set). However, in general, an internal node can have a subset of the allowed values for that node. The simulation proceeds in a topological order. When a node is to be simulated, each of its fanins has been assigned a *set* of values. The node’s output is the *set* of all values possible for that node given its fanin sets; each fanin minterm can be taken from the

[2] A more detailed discussion of the elimination operation can be found in Section 5.

product set of the fanin sets. For example, suppose each input has a set of values, S_{i_k}. The output of a node i is evaluated as the following set:[3]

$$S_i = \{ v \mid R_i(V_i, v) = 1,\ V_i \in S_{i_1} \times S_{i_2} \times \cdots \times S_{i_{|Y_i|}} \}.$$

Each fanout edge $i \rightarrow j$ then receives the set S_i. When all CO nodes have been computed, the cross product of the CO output sets forms the set of minterms { m_Z } allowed for m_X. Any such a pair (m_X, m_Z) is in the SS-behavior of the network[4], i.e.

$$(m_X, m_Z) \in R^{SS}(X, Z).$$

It is easy to observe that the SS-behavior is an output-symmetric relation and, hence, can be represented by a set of independent relations, one for each output. Similar to NSC, a key advantage of SS is that the network can be manipulated as a network of single-output MV nodes. In contrast, the use of NS-behavior would lead to multi-output nodes and MV multi-output relations at these nodes (see [5]).

SS-behavior can be shown to be the same as considering the ND network as a set of deterministic binary nodes, one for each *i*-set of each MV node. For example, consider a ternary node *j*. The *i*-sets of this node (0-set, 1-set, and 2-set) are represented by MV-input binary-output functions. In this binary interpretation, each internal MV signal and each CO is replaced by a set of binary signals and each corresponding literal in any MVSOP is converted to a sum of binary literals, e.g. $y^{\{1,3,5\}} = b_1^y + b_3^y + b_5^y$, where b_i^y is the binary output of the j^{th} *i*-set of *y*. The resulting network is deterministic and can be manipulated like any such network[5]. Basically, this conversion is like representing MV signals using positional notation which allows for the representation of sets. The only signals that are multi-valued are the CIs, which do not have to be converted since they only carry singleton sets.

Theorem 3.3: *The SS-behavior of an ND network can be obtained by treating each i-set as a separate binary function, collapsing the network (in any order), and merging each set of binary outputs associated with a CO to form the i-sets of that MV output.*

Another method for computing the SS-behavior is the following.

Theorem 3.4: *The SS-behavior of an ND network is exactly that obtained by eliminating all internal nodes in* topological *order.*

The same effect can be obtained by unfolding the network into a tree (using duplication), resulting in a network where each node has exactly one fanout. It turns out that the SS-behavior is unchanged. An ND node in this tree has a unique path to one CO, so the effect that an ND node in the original network can have on the SS-behavior is directly related to the set of all paths from the node to a CO. Each time the output of an ND node branches to several fanouts, the effect is as if independent "copies" are made of the ND node are made. As discussed in subsequent sections, any network operation that increases (decreases) the number of paths from an ND node to a CO can increase (decrease) the SS-behavior of the network.

[3] Note that even if the node relation is deterministic, the output set can have more that one element if some of the inputs are sets with more than one value.

[4] Set simulation is similar to what is done in ternary simulation when values 0,1,X are propagated. X stands for the set {0,1}.

[5] In fact, the network is unate.

3.4 Comparison and Representation of Behaviors

A network's external specification gives the upper bound on the allowed network behavior. The specification can be output-symmetric (independent relation for each output) or a Boolean relation relating all outputs. An output-symmetric specification is analogous to giving compatible external don't cares for a binary network. Operations on an ND network can change its *B*-behavior, $B \in \{NS, NSC, SS\}$. An increase in behavior is allowed only if it is still contained in the specification.

Output-symmetric specifications have the advantage that they can be stored individually for each output, e.g. as a set of binary-output *i*-set functions. Other specifications may require a single global multi-output relation, relating all inputs and outputs, which can easily become too large. If the specification is not output-symmetric, one option is to *under*-approximate it with an output symmetric one; this leads to a correct but conservative approach.

The node minimization operation (as discussed in Section 4) uses the external specification directly to test how much a node's behavior (any of the *B*-behaviors) can be increased without violating the specification. In Section 4, an ND relation at a node is computed to describe the maximum flexibility (complete flexibility CF) allowed in implementing the node. Different interpretations of a network's behavior will lead to different flexibilities (*B*-CFs) allowed. Node minimization is the process of solving for a well-defined sub-relation of the *B*-CF, which gives the smallest representation of the node [4]. Using an ND sub-relation of the *B*-CF allows for smaller representations.[6]

Another aspect is the ease of performing network manipulations using the different behaviors. SS-behavior is the most efficient because it is related to collapsing the network in topological order. This allows building global MDDs of each node, where only CI variables are needed at any stage in the collapsing process. NSC is also relatively easy because collapsing in reverse topological order can be used, but building global MDDs is slightly more difficult since internal variables (but only those representing the outputs of ND nodes) must be used temporarily in the MDDs. NS-behavior requires either the use of multi-output relations or input determinization using pseudo-inputs.

[6] A minimum deterministic sub-relation can never be smaller that a minimum ND sub-relation.

It is obvious that NS behavior is contained in NSC-behavior. Also, NSC is a subset of the SS-behavior. One way to see this is that in NSC some "copies" of ND relations, which lead to the same CO, are kept synchronized (interdependent) during the collapsing process. In contrast, with SS, all correlations between different fanouts of an ND node are lost when the node is eliminated (since elimination is done each fanout at a time). As a result, we have,

$$R^{NS}(X,Z) \subseteq R^{NSC}(X,Z) \subseteq R^{SS}(X,Z). \qquad (3.3)$$

In Section 4, it is shown that this ordering has the reverse effect on the optimization potentials (flexibilities) computed using these behaviors, because the computation is based on comparing (by containment) against the external specification. For example, if SS-behavior is used, containment is more restrictive since SS-behavior is the largest. Thus the use of SS behavior will lead to less flexibility allowed in implementing a node. On the other hand, SS-behavior is easier to compute with.

4 Node Flexibilities

The computation of the complete flexibility, CF, at a node y_i in an ND network can be described somewhat generically for the different behaviors $B \in \{NS, NSC, SS\}$. However, for each behavior, certain modifications need to be done.

Cut the network at the output of node *i* and consider the new network (the *cut* network), which has an additional independent primary input y_j. Require that the *B*-behavior of the cut network, $R^B(X, y_j, Z)$, complies with the network specification $R^{spec}(X,Z)$:

$$R^B(X, y_j) \equiv \forall_Z (R^B(X, y_j, Z) \Rightarrow R^{spec}(X,Z)), \qquad (4.1)$$

which simply says that for all outputs, the cut-network behavior should be contained in the specification. Note that for both NSC and SS, the behavior and the specification can be stated in terms of the individual outputs, z_k, in which case the computation becomes

$$R_k^B(X, y_j) \equiv \forall_{z_k} (R^B(X, y_j, z_k) \Rightarrow R^{spec}(X, z_k)) \qquad (4.2)$$

which makes the computation for $B \in \{NSC, SS\}$ much more efficient.

It turns out that if the flexibility for SS-behavior were computed by Equation 4.2, then Theorem 4.2 below, about how it can be used, would not hold. We need to modify the computation of $R_k^{SS}(X, y_j)$ as follows. When $R^{SS}(X, y_j, z_k)$ is computed for the cut network, it needs to be changed such that it can have *set* inputs at the y_j input node in the cut network, since that is what can happen at the output of the y_j node when SS-simulation is done on the uncut network. This can be done by introducing new binary variables b^j, which encode subsets of D_j, the domain of y_j.

For example, if $D_j = \{0,1,2\}$, there would be 3 binary signals $\{b_0^j, b_1^j, b_2^j\}$ as inputs to a modified cut network (for example, (0,1,1) would encode the subset $\{1,2\} \subset \{0,1,2\}$). A new node, h_j is introduced in place of node *j*. Its inputs are $\{b_0^j, b_1^j, b_2^j\}$ and its output y_j fans out to the same nodes as in the original network. The node relation at h_j is denoted $R^{set}(b^j, y_j)$ and serves to translate between the binary inputs and the MV set outputs. Thus, in the example, (0,1,1,1) and (0,1,1,2) are in the relation $R^{set}(b_0^j, b_1^j, b_2^j, y_j)$, and (0,1,1,0) is not.

Then, $R^{SS}(X, b^j, z_k)$ is computed for the modified cut network and this is used in Equation (4.2) to obtain $R_k^{SS}(X, b^j)$, which relates X to allowed subsets of D_j.

Relations $R_k^B(X, y_j)$ express the Observability Partial Care (OPC) for the node at output *k*, which is related to observability don't cares computed for a node in a binary network. Note that the relations depend on the CIs, *X*.

Next we bring in what is analogous to "satisfiability don't cares" (SDC) to derive a local "complete" flexibility (CF). Define $M^B(X, Y_j)$ as the relation between CI minterms and vectors of values that the fanin variables, Y_j, of node *j* can take **during** *B*-simulation of the entire network[7]. The *B*-CF is computed by the formula

$$R^B(Y_j, y_j) = \prod_{k=1}^{N} \forall_X (M^B(X, Y_j) \Rightarrow R_k^B(X, y_j)).^{8} \qquad (4.3)$$

This simply says that for all input minterms, those fanin minterms, m_{Y_j}, that can be produced by *B*-simulation must be related to the corresponding output values of the global flexibility. It can be shown that

$$R^{SS}(Y_j, y_j) \subseteq R^{NSC}(Y_j, y_j) \subseteq R^{NS}(Y_j, y_j). \qquad (4.4)$$

We claim that each of these is maximal, i.e. no additional pair of minterms can be included in any of the relations while maintaining a valid flexibility relation.

[7] A subtle point is that in general for NSC, this is **not** the same as the NSC-behavior $R^{NSC}(X, Y_j)$ of the cut sub-network whose COs are Y_j, but in fact $M^{NSC}(X, Y_j) = R^{NS}(X, Y_j)$. Roughly, this is because internal nodes and output nodes are treated differently in ND networks. This makes it more difficult to compute.

[8] For NS, there is no product over all outputs since $R^{NS}(X, y_j)$ takes all outputs into account. For SS, we obtain $R^{SS}(Y_j, b^j)$ which is a (multiple output) Boolean relation. In general, to convert this to an ND multi-valued relation (single output), we need to choose, for each m_{Y_j}, one of the allowed sets as indicated by b^j. For a given m_{Y_j}, there may be several such sets.

In general, the CFs, $R^B(Y_j, y_j)$ for $B \in \{NS, NSC, SS\}$, are non-deterministic relations. Since the current relation at node j, $R(Y_j, y_j)$, is well defined and $R_j(Y_j, y_j) \subseteq R^B(Y_j, y_j)$, then also $R^B(Y_j, y_j)$ is well-defined (assuming that the current network conforms to the specifications).

Theorem 4.1: *The B-CF for node j is well-defined if and only if there exists a relation for node j such that the resulting network B-conforms* $\forall_{z_i}, z_i \in TFO(j)$.

By *B*-conforms $\forall_{z_i}, z_i \in TFO(j)$ we mean that the containment of relations (behavior is in the spec.) holds for those outputs in the TFO of node *j*. However, conflicts are possible at the outputs not in *TFO(j)*. The importance of Theorem 4.1 is that the CF can tell us if it is possible to correct the network to meet its specifications at the *TFO(j)* by changing the relation at node *j* only. The main import of the CF is the following.

Theorem 4.2: *If **any** well-defined ND sub-relation contained in* $R^B(Y_j, y_j)$ *is inserted at node j, then the new network,* $\tilde{N}$*, is B-compliant, i.e.* $\tilde{R}^B(X, Z) \subseteq R^{spec}(X, Z)$*, at least for those outputs in the TFO(j).*

It could be that the initial network is not compliant. Then the use of a well-defined sub-relation can only correct those outputs that it can influence. If the initial network is compliant, then it remains so after using any well-defined sub-relation contained in its CF. In practice, one wants to find the well-defined sub-relation with the smallest representation. This is normally measured in terms of the total number of cubes in the non-default *i*-sets. In [4], a Quine-McCluskey type algorithm is given for finding a sub-relation with the exact minimum number of cubes. Generally, the solution is ND. The corresponding problem for finding an optimum deterministic sub-relation is not solved.

5 Elimination

Elimination is the process of substituting the relation of a node into all the relations of its fanouts. Substitution of the relation at node *k* into a fanout *i* is defined as replacing relation $R_k(Y_k, y_k)$ with $\exists_{y_i} R_i(Y_i, y_i) R_k(Y_k, y_k)$[9]. After R_i has been substituted into all its fanouts, it can be removed (eliminated) from the network, since y_i is no longer used anywhere. The impact of eliminating a node on the behavior of the resulting network is summarized below.

Theorem 5.1: *Eliminating a node can increase the NS and NSC behaviors of a network only if the node being eliminated is ND and has more than one fanout.*

Theorem 5.2: *Eliminating a node can increase a network's NSC behavior if and only if the node is ND and has reconvergent fanout.*

[9] $y_i \in Y_k$ since *k* is a fanout of *i*.

Theorem 5.3: *Eliminating a node cannot increase the SS-behavior of the network.*

The original reason for considering SS-behavior was that elimination effectively substitutes a copy of the eliminated node into each fanout. Each copy acts independently of the other copies and effectively broadcasts an independent *set* of values to its fanout. Since SS effectively does the same thing, elimination can not increase the SS-behavior of a network. However, elimination can *decrease* the SS-behavior if the nodes are not eliminated in topological order (the number of paths to an output can decrease in this case).

6 Extraction and Decomposition

Extraction and decomposition are similar; the latter operates on a single node at a time, while the former operates on a set of nodes. With decomposition, a new node (divisor) is created, which has only a single fanout; with extraction there are two or more fanouts. The objective is the same, to find a good divisor. There are two forms of extraction/decomposition, disjoint and non-disjoint. It is disjoint if the fanins of the new node are **not** fanins of its fanouts.

Theorem 6.1: *Extraction and decomposition cannot increase the NS and NSC behaviors of an ND network.*

Theorem 6.2. *The SS-behavior of a network is not changed if, in a node decomposition/extraction, the non-disjoint variables have no ND nodes in their TFIs.*

The SS-behavior is related to the number of paths from an ND node to the outputs. A non-disjoint decomposition can increase this number. As an example, consider the network in Figure 2.

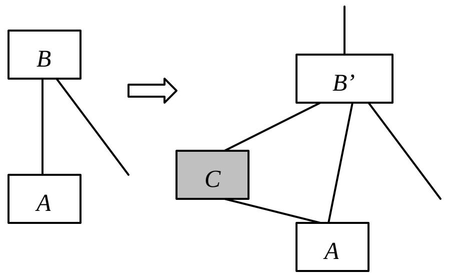

Figure 2. Non-disjoint decomposition of B

The decomposition of *B* is non-disjoint because the inputs of *C* are not disjoint from the inputs of *B'*. Thus the number of paths from *A* to *B* has increased. If *A* is non-deterministic or there is an ND node in TFI(*A*), then the SS-behavior could increase.

7 Merging

Merging is the process of combining two or more nodes (the merging set) into a single node with more values [6]. A constraint on the merging set is that the network should remain acyclic. The *i*-sets of the new node are composed of intersections of the *i*-sets of the set of nodes being merged.

Example: Consider the merging of two nodes with value ranges 3 and 5, respectively. Then, the 0-set of the new node is the intersection of the 0-sets of the two relations, the 1-set is the intersection of the 0-set and the 1-set, the 2-set the intersection of the 0-set and 2-set, etc, and the 14-set the intersection of the 2-set and the 4-set.

The second step of merging involves substituting the new node into the union of the fanouts of the merging set by replacing literals of the merging set of each cube in the *i*-set covers of a fanout by a single literal of the new variable.

Example: In the above example, if a fanout cube in some *i*-set involves the product $x^{\{0,2\}}y^{\{1,3,4\}}$ (*x* and *y* form the merging set and are three-valued and five-valued MV variables, respectively), this product is replaced by the single literal of the new 15-valued variable, say *z*, $z^{\{1,3,4,11,13,14\}}$. A cube with the literal $x^{\{1\}}$ but no y is replaced by $z^{\{5,6,7,8,9\}}$ since the absence of a literal of y implies all (five) values of *y*.

Thus, the number of *i*-set cubes in the fanouts cannot increase, but most likely will decrease (which is one point in doing a merge) after the resulting *i*-sets are made prime and irredundant.

Example: An example of the reduction is given by the binary XOR gate with inputs *x* and *y*: $x^{\{0\}}y^{\{1\}} + x^{\{1\}}y^{\{0\}}$. It has two cubes and four literals. If *x* and *y* are merged into a single node *z*, the MV-SOP of the gate becomes $z^{\{1\}} + z^{\{2\}}$, which, when made prime and irredundant, becomes one cube and one literal, $z^{\{1,2\}}$.

Theorem 7.1. *Merging of nodes cannot change the NS or NSC behaviors and cannot increase the SS behavior of the network.*

Merging can decrease the SS-behavior since the number of paths to an output may decrease.

8 Comparing NSC and SS

Since NSC and SS behaviors are computationally easier than NS, they are the likely candidates for implementation. Comparison leads to the following statements.

1. Both lead to output-symmetric relations at the COs. This allows the handling of each CO separately.
2. The computational process for SS is made easier since collapsing in topological order allows for building global MDDs using only the CI variables. However, it is made slightly more difficult by the need to handle sets at the cut variable. Since NSC is computed in reverse topological order, intermediate variables at the ND nodes must be part of the computation. We found that this makes it much more difficult to compute.
3. Both lead to network operations, similar to those for binary networks, allowing operations on single nodes and creating only single output nodes.
4. One operation in each case can increase behavior. It has been characterized when this happens and, in both cases, the operations can be easily modified to ensure that the behavior can't increase. The problematic operation is elimination for NSC and is extraction for SS.
5. NSC can provide more flexibility at a node, but from experiments, the improvement is only about 1%.

The theory based on SS-behavior was the one implemented initially in MVSIS, being the most computationally efficient. NSC implementation has just been completed.

Table 1 summarizes the comparison between NSC behavior and SS-behavior in terms of possible changes of the network behavior after the corresponding operation. An increase in behavior could cause non-compliance.

Operation	**SS-behavior**	**NSC-behavior**
elimination	can't increase	**may increase (see Theorem 5.2)**
extraction	**may increase (see Theorem 6.2)**	can't increase
node minimization	can't increase	can't increase
node flexibility	less	more
merging	can't increase	can't change

Table 1. Comparing two computationally viable theories.

9 Hierarchical Theory

An interesting question is what type of simulation has been assumed if the specification is given by the initial ND network[10]. This occurs in the following situations:

1) Network N has been cut out of a larger network, $\tilde{N}$, and N acts as its own specification. This might happen if $\tilde{N}$ is so large that optimization algorithms cannot be applied to it. Thus, sub-networks N are cut out; their inputs and outputs are treated as PIs and POs. No external don't cares are given for N because these would have to be derived from $\tilde{N}$. The objective is to re-synthesize N to obtain a smaller sub-network whose behavior is contained in N. The result is then stitched back into $\tilde{N}$. It is important to **guarantee** that the network containing the optimized sub-network still satisfies the specifications for $\tilde{N}$, because it is time consuming to check containment after each step.

[10] If the specified network is deterministic, this question does not arise.

2) A sub-network N is cut out of a larger network, but its contents are ignored. The specification N is derived only from the surrounding environment of N in $\tilde{N}$ and the specification. This is similar to computing the CFs, except that, in general, the cut-out sub-network may have several outputs.

The second type of optimization in a hierarchical theory is problematic, since a type of CF would need to be derived for multiple output nodes. This is similar to what has been done for deterministic binary networks [1] in terms of Boolean relations. We leave this type of optimization for another paper.

For the first type, we can state the following results:

1. If the *NS*-behavior of N is not increased, then the *NS*-behavior $\tilde{N}$ is not increased.
2. For NSC, a small modification needs to be done in treating groups of outputs (those that have paths to the same output in $\tilde{N}$) of N as dependent.
3. To guarantee compliance for SS-behaviors, the inputs of the carved-out network N should be considered as general set inputs, rather than singleton-set inputs as it is done in the case of NS and NSC. This modification is similar to the change in the computation of flexibility using SS-behavior, discussed in Section 4, where set inputs were required.

Hence, under these modifications, if the original network $\tilde{N}$ has its {NS, NSC, SS}-behavior is contained in the specification, then the modified network containing the modified carved network would also {NS, NSC, SS}-conform. Thus {NS, NSC, SS}-behaviors can be made suitable for sub-network optimization. This observation is important since it allows for partitioning a large network, optimizing the sub-parts separately, and composing the results, leading to a valid modified network.

10 Modifying Two Network Operations

In Table 1, two operations are problematic. one for NSC and one for SS. These need to be modified to ensure that the network behavior is not increased to possibly make it non-conform.

The only network operation that could cause the NSC-behavior to increase is elimination, and then only if an ND node with reconvergent fanout is eliminated. Thus, the following modification to the elimination operation can be made when NSC behavior is used.

Check a node to be eliminated for being both ND and having reconvergent fanout; if both conditions hold, then the node relation is **determinized** *(replaced by a well-defined deterministic sub-relation) before elimination.*

Since all other network eliminations can't increase the NSC behavior, the ND network can never get out of NSC-compliance if this modification is used.

For SS, the only operation that could cause non-compliance is extraction/decomposition. By Theorem 6.2, this can happen only if there is a non-disjoint variable in the decomposition with an ND node in its TFI. The following modification can be made.

During extraction, check the new divisor for inputs that are not disjoint. If found, then either determinize all ND nodes in the TFI of this divisor, or look for another divisor.

Additionally, one could accept the new divisor and then check its SS-CF. If it is well defined, then according to Theorem 4.1, the divisor can be modified to correct this non-conformance.

11 Experimental Results

We compared the amount of flexibility obtained by using NSC versus SS. Ten benchmarks were selected that were multi-level and had multi-valued variables.

	Statistics				
Name	I/O	N	Nnd	Rec	ND, %
4ac	9/8	48	20	0.25	10.39
bpds	12/6	79	23	0.29	10.97
cc	15/10	97	68	0.17	12.91
comp	4/2	11	6	0.00	16.12
ep	7/4	32	27	0.03	12.93
sort	8/8	25	20	15.36	9.89
9sym	9/1	230	97	0.30	8.15
clip	9/5	333	125	0.40	7.31
cordic	23/2	68	35	0.10	11.30
c432	36/7	117	44	4.05	9.78
Aver.					10.97

Table 2: Statistics of benchmark examples.

These benchmarks were modified by inserting at each node about 10% more non-determinism than originally present. The average amount of non-determinism per node is shown in Column 6 of Table 2, where N is the number of nodes, Nnd the number of ND nodes, and Rec the average number of reconvergent paths per node.

	Behavior			Flexibility	
Name	SS, %	NSC, %	NS, %	SS, %	NS, %
4ac	1.39	1.29	1.04	45.74	47.30
bpds	4.53	4.53	4.22	72.59	72.59
cc	0.54	0.53	0.49	64.40	64.46
comp	100.00	100.00	85.41	100.00	100.00
ep	20.83	20.83	20.83	81.76	81.76
sort	1.08	1.07	0.62	54.92	57.13
9sym	57.61	57.61	57.61	87.12	87.12
clip	41.58	41.58	41.58	84.23	84.31
cordic	45.63	45.63	44.71	84.74	84.74
c432	46.28	44.83	16.95	54.21	54.64
Aver.	31.94	31.79	27.34	72.97	73.40

Table 3: Comparison of NSC and SS flexibilities.

The amount of global non-determinism was computed for the three behaviors for the resulting network shown in Columns 2, 3, and 4 of Table 3. The SS behavior was taken as the external specification, and was used in calculating the average, over all nodes, of the amount of non-determinism in the resulting NSC and SS flexibilities. These are shown in Columns 5 and 6 of Table 3.

12 Conclusions

A theory of non-deterministic networks has been developed and implemented to merge the two concepts of multi-level networks and non-determinism. Space limitations allowed for only stating the results of this investigation. For a more complete understanding, it is necessary to refer to [5].

The legality of various classical network manipulations was analyzed under three different definitions of behavior, which correspond to three methods of simulation: normal (NS), normal compatible (NSC), and set simulation (SS). SS is the same as that obtained by eliminating all internal nodes in the network in topological order and NSC is equivalent to elimination in reverse topological order. It was shown that $NS \subseteq NSC \subseteq SS$.

For $B \in \{NS, NSC, SS\}$, algorithms were given to compute corresponding complete flexibilities (*B*-CFs) at a node. It is claimed that these flexibilities are maximum, and all have the property that any well-defined sub-relation contained in them maintains compliance.

Two operations could cause non-compliance when non-determinism is present; extraction is problematic for SS, while elimination is problematic for NSC. However, easy specific modifications were given, which alter the way extraction or elimination is performed in those cases where some non-determinism is related to the node under operation.

The use of NS behavior seems too computationally expensive for larger circuits since it is equivalent to computing a Boolean relation for the entire network. In contrast, NSC and SS behaviors have reasonable computational costs (as verified by our current implementations), since they can be formulated in terms of a relation at each of the separate outputs. All behaviors can be fit into a hierarchical theory, which can handle very large networks.

The manipulation of ND networks using SS-behavior was implemented in the second generation of the MVSIS system [6]. Recently, NSC behavior has been implemented in MVSIS. The experimental results show that the flexibility computed using both types of behavior are significantly larger than the amount of non-determinism present at the node (73% vs. 10%). The additional flexibility can improve the quality of the optimization algorithms.

Surprisingly, experimental results indicate that NSC yields only about 1% more flexibility than for SS. Since we also found that computing NSC flexibility was much more difficult that we first thought, we conclude that the use of the more efficient computation (SS) does not loose much in flexibility.

Initial experience with modification of those operations that could cause the network to become non-compliant, show that it is easy to modify these on-the-fly. Efficiency of handling ND networks is on a par with the manipulation of comparable deterministic binary networks; thus the penalty for including the generalizations to multi-valued non-deterministic nodes is small[11].

A near-term future goal is to experiment with enhancing the well-known binary operations by allowing various optimization algorithms to search in a larger (MV) space. A longer-range goal is to open up, to many applications, the possibility of manipulating non-deterministic problems. For example, we can use ND network manipulations to operate on ND regular automata by developing efficient operations of complementation and composition. These could be applied directly to multi-level network representations of the automata yielding possible improvements in efficiency over existing techniques. Many applications, such as protocol synthesis, cryptography, discrete control problems, solving games etc., could benefit from this capability.

References

[1] D. Brand, "Verification of large synthesized designs", Proc. ICCAD '93, pp. 456-459.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Dordrecht, 1984.

[3] A. Mishchenko, and R. Brayton, "A Boolean paradigm for multi-valued logic synthesis", Proc. *IWLS'02*, pp. 173-177.

[4] A. Mishchenko and R. Brayton, "Simplification of non-deterministic multi-valued networks", *Proc. ICCAD'02*, pp.557-562.

[5] A. Mishchenko, and R. Brayton, "A Theory of Non-Deterministic Networks", UC Berkeley Technical Report, ERL, EECS Department, Feb. 2003.

[6] MVSIS Group. *MVSIS*. UC Berkeley. http://www-cad.eecs.berkeley.edu/mvsis/

[7] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization". *IEEE Trans. CAD*, Vol. 6(5), pp. 727-750, Sep. 1987.

[8] 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.

[9] Y. Watanabe, L. Guerra and R. K. Brayton, "Logic optimization with multi-output gates", Proc. ICCD '93, pp. 416-420.

[10] N. Yevtushenko, T. Villa, R. K. Brayton, A. Petrenko, and A. L. Sangiovanni-Vincentelli, "Solution of parallel language equations for logic synthesis", Proc. ICCAD '01, pp. 103-111.

[11] In fact, the new theory gave us the motivation to re-implement all the technology independent operations of SIS in this more general setting. The result is a logic synthesis capability which is much more efficient – there was little loss because of the generalization while the re-thinking of the algorithms and data-structures improved the efficiency greatly.