Don’t Cares and Multi-Valued Logic Network Minimization  *

Yunjian Jiang    Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{wjiang,brayton}@eecs.berkeley.edu

Abstract

We address optimizing multi-valued (MV) logic functions in a multi-level combinational logic network. Each node in the network, called an MV-node, has multi-valued inputs and single multi-valued output. The notion of don’t cares used in binary logic is generalized to multi-valued logic. It contains two types of flexibility: incomplete specification and non-determinism. We generalize the computation of observability don’t cares for a multi-valued function node. Methods are given to compute (a) the maximum set of observability don’t cares, and (b) the compatible set of observability don’t cares for each MV-node. We give a recursive image computation to transform the don’t cares into the space of local inputs of the node to be minimized. The methods are applied to some experimental multi-valued networks, and demonstrate reduction in the size of the tables that represent multi-valued logic functions.

1 Introduction

Multi-valued (MV) logic synthesis is becoming important in various applications. It can be used in hardware synthesis as a higher level representation before the circuit is encoded into binary. There are optimization opportunities at this stage that cannot be discovered in a binary domain. It can also be used in control dominated software compilation, where control variables are computed and tested just as variables in multi-valued logic. This explores logical relations between variables to restructure the control flow of a program. Such optimizations are usually not considered by traditional software compilers.

We address the optimization problem of multi-valued input, multi-valued output logic functions in a combinational logic network. The notion of don’t cares used in binary logic [1] is generalized to multi-valued logic. These contain two types of flexibility, incomplete specification and non-determinism. We define multi-valued don’t cares and partial cares to capture this flexibility. Multi-valued functions combined with multi-valued partial cares are similar to Boolean relations in binary logic, where a set of compatible functions are to be explored for the optimization. The algorithms developed for Boolean relation minimization, [2] [3] [4] [5], can be applied in partial care minimization for a multi-valued function.

We give algorithms for generating a compatible set of don’t cares for MV-nodes. Observability don’t cares (ODC) for an MV-node is the set of minterms, which, when applied in the network, blocks the output values of that node, i.e. the primary outputs do not depend on the values of that node. Compatible ODC’s (CODCs) are don’t cares which do not depend on how the don’t cares at other MV-nodes in the network are used. The methods to compute ODC’s and CODC’s are extended from the binary case [6] [7]. Observability partial cares (OPC) are the set of the minterms that blocks a subset of the output values, i.e. the primary output can not distinguish any pair of values in that subset. OPC’s provide additional flexibility for the implementation of an MV-node. This is a generalization of observability Boolean relations for binary networks. In order to use the OPC’s, a multi-valued relation minimizer needs to be applied. We give one method for generating OPC’s.

In Section 2, we give the definition of multi-valued partial cares. Section 3 gives the algorithms to compute compatible don’t cares for a multi-valued node in a combinational network, and Section 4 discusses image computation for a MV-network. We give some experimental results in Section 5 and conclude in Section 6.

2 Multi-valued Functions

Consider a multi-valued function with multiple multi-valued inputs and single multi-valued output. A multi-valued relation is, like a binary relation, a one to many mapping. Let \( P_1 = \{0, 1, ..., |P_1| - 1\}, P_2 = \{0, 1, ..., |P_2| - 1\}, P_n = \{0, 1, ..., |P_n| - 1\} \) be the input space, and \( Q = \{0, 1, ..., |Q| - 1\} \) be the output space. The multi-valued relation \( R : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q \) maps each minterm in the input space, \( P_1 \times P_2 \times ... \times P_n \), to a set of values in \( Q \), i.e. \( m \in P_1 \times P_2 \times ... \times P_n, R(m) \subseteq Q \). We assume that \( R \) is complete, i.e. \( R(m) \neq \emptyset \), for all \( m \). Associated with \( R \) is a set of multi-valued functions \( \{f_i\} \) compatible with \( R, f_i \prec R \). The multi-valued minimization problem is to find an optimal implementation of \( f \) that is compatible with \( R \).

Example 1 Multi-valued relation \( R_1 \) is defined in the following sum-of-products table.

---

*This work was supported by SRC under contract 684.004 and the California Micro program.
does not affect the functionality of the network. In the hardware implementation, a function needs to be deterministic and produce one value for each input minterm. Therefore, the synthesis process needs to be deterministic and produce a deterministic function satisfying some optimality criteria. If the target application is software, however, the functionality of an MV-node need not be determined for the purpose of output evaluation.

**Definition 3 (Compatible)** A multi-valued function \( f : P_1 \times P_2 \times \ldots \times P_n \rightarrow Q \) is compatible with a multi-valued relation \( R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q \) if \( \forall m \in P_1 \times P_2 \times \ldots \times P_n, f(m) \in R(m) \).

Given a multi-valued relation \( R \) with a set of don’t cares and partial cares, the minimization problem for hardware implementation consists of two steps: (1) find a multi-valued function \( f \) compatible with \( R \); (2) find an optimal implementation for \( f \), in terms of the number of product terms and/or the number of multi-valued literals. For instance, the example given in Figure 1 can be optimized into:

| \{0,1\} × \{0,1,2\} × \{0,1,2\} → \{0,1,2,3\} |
|---|---|---|---|---|
| 0 | 1 | - | - | \{0,1\} |
| 1 | - | 1 | 1 | \{0,2\} |
| - | 0 | 1 | 1 | \{1,2\} |
| 0 | - | 1 | 2 | \{1,3\} |

The mapping is shown graphically in Figure 1. As can be seen, two types of flexibility exist, namely incomplete specification and non-determinism. There are unspecified minterms in the table, e.g. \( \{0\} \times \{0\} \times \{0\} \), which can take any value in \( Q \) for the mapping. This represents the traditional don’t care minterms in the binary domain. There are also minterms that can take a subset of the values in \( Q \), e.g. \( \{0\} \times \{1\} \times \{2\} \rightarrow \{0,1,3\} \).

We call these partial care minterms. This situation is not present in binary logic, where each minterm can take either value 0 or value 1 if it is not a don’t care.

**Definition 1 (Don’t Care)** A minterm \( m \in P_1 \times P_2 \times \ldots \times P_n \) for multi-valued relation \( R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q \) is a don’t care, if \( R(m) = Q \).

**Definition 2 (Partial Care)** A minterm \( m \in P_1 \times P_2 \times \ldots \times P_n \) for multi-valued relation \( R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q \) is a partial care, iff \( |R(m)| > 1 \) and \( R(m) \subset Q \).

Partial cares can result from observability relations in a logic network, or special requirements given by the designer. Consider an MV-node \( n_i \) in an MV-network. There exists a set of minterms, which when applied at the primary input space, allow the output values of \( n_i \) to be within a subset of the values \( v_j \in [x_i] \) for \( n_i \), where \( x_i \) is the output variable for node \( n_i \) and \( |x_i| \) is the number of values specified for \( x_i \). This set of minterms, when mapped into the local input space of \( n_i \), can be used to produce any value in the subset \( v_j \) for node \( n_i \). This provides additional flexibility in the implementation of \( n_i \), which does not affect the functionality of the network. In the hardware implementation, a function needs to be deterministic and produce one value for each input minterm. Therefore, the synthesis process needs to be deterministic and produce a deterministic function satisfying some optimality criteria. If the target application is software, however, the functionality of an MV-node need not be determined for the purpose of output evaluation.

**Definition 3 (Compatible)** A multi-valued function \( f : P_1 \times P_2 \times \ldots \times P_n \rightarrow Q \) is compatible with a multi-valued relation \( R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q \) if \( \forall m \in P_1 \times P_2 \times \ldots \times P_n, f(m) \in R(m) \).

If the multi-valued output variable is encoded into binary variables, the multi-valued relation becomes a Boolean relation. Exact and heuristic algorithms for Boolean relation minimization [2] [3] [4] [5], etc., can thus be used to minimize multi-valued relations.

### 3 Observability in multi-valued logic networks

Observability Boolean relations have been studied for Boolean networks, e.g. [7][8][9], but computational intensity has prevented the methods from being practical. Multi-valued observability partial cares are a generalization of binary observability relations.

A multi-valued combinational logic network, or MV-network, is a network of nodes. Each node represents a multi-valued function with a single multi-valued output and multi-valued inputs. There is a directed edge from node \( i \) to node \( j \), if the function at node \( j \) explicitly depends on the output variable at node \( i \). We use \( |x_i| \) to denote the number of values specified for the MV-variable \( x_i \) at node \( i \). Algebraic methods like [10] [11] can be used to derive an appropriate structure for the MV-network. Once the structure has been decided, the multi-valued function at each node can be optimized according to the maximal permissible behavior allowed for this node. The flexibility is given by satisfiability don’t cares (SDC), observability don’t cares (ODC) and observability partial cares (OPC). For the definition and application of binary SDC and ODC refer to [7].

#### 3.1 Maximal set of observability don’t cares

Let \( y_i \) be the output variable, and \( \{x_1, ..., x_r\} \) be the input variables of node \( i \). Let \( y_i \in \{0, ..., n\} \), and \( x_j \in \{0, ..., t_j\} \). The MODC for the input edge \( x_j \), is the set of minterms in the primary input space, such that the output MV-function \( y_i \) is insensitive to all values of \( x_j \). This set of minterms can be used as don’t cares for the minimization of the MV-function \( x_j \), if \( y_i \) is the only fanout of \( x_j \). We first compute the set of don’t care minterms MODC in the local input space of \( y_i \), under which the values of \( x_j \) are indistinguishable. This gives the maximal

---

Figure 1: Multi-valued functions
ODC for edge \( x_j \rightarrow y_i \). MODC can be defined as follows:

\[
MODC(y_i, x_j) = \{ m | f(m[x_j = 0]) = ... = f(m[x_j = t_j]) , \ 
\quad m \in P_1 \times ... \times P_r \}
\]

We use \( m[x_j = k], k \in \{0, ..., t_j\} \) to denote setting the value of \( x_j \) in minterm \( m \) to \( k \). The value of \( y_i \) does not change, if we arbitrarily flip the value of \( x_j \) within the range \( \{0, ..., t_j\} \) and keep the other parts of minterm \( m \) fixed. This gives the condition that the function produced by \( x_j \) is totally blocked by the other inputs and can not be observed at \( y_i \). \( x_j \rightarrow y_i \) becomes a redundant wire for this set of minterms. It can be shown that if \( f \) is specified as a function, i.e. deterministic, MODC can be computed using the following formula:

\[
MODC(y_i, x_j) = f_{m1} \cdot f_{m2} \cdot ... \cdot f_{mk} + \sum_{l=0}^{m} \prod_{k=0}^{n} f_{kl}
\]

\( f^j \) is a binary function, \( P_1 \times ... \times P_r \rightarrow B \), which defines the set of minterms in \( P_1 \times ... \times P_r \) that produce output \( l \) for \( y_i \). Function \( f_{kl}^j \) is the cofactor of binary function \( f^j \) with respect to literal \( x_j \). It is independent of \( x_j \) and preserves the onset of \( f^j \) whenever \( x_j = k \), i.e. \( x_j \cdot f^{j}_{x_j} = x_j \cdot f^{j} \). Function \( f_{m1}^{j} \cdot f_{m2}^{j} \cdot ... \cdot f_{mk}^{j} \) defines the set of minterms in \( P_1 \times ... \times P_r \) such that the output value for \( y_i \) is always \( l \) no matter what value \( x_j \) takes, i.e., the universal quantification over the values of \( x_j \). Formula (2) represents the complement of the multi-valued logic difference (\( \partial f/\partial x_j \)) when \( f \) is deterministically specified.

Theorem 1 (MODC) The binary function (2) computes the set of MODC minterms for input edge \( x_j \) in the input space of \( y_i \) as defined in (1), if the functionality of \( y_i \) is deterministically specified in \( f \).

If \( f \) is specified as a relation, i.e. nondeterministic, more complicated methods, which use multi-valued logic difference, need to be applied.

3.2 Compatible set of observability don’t cares

The validity of MODC’s for a particular input edge requires other input edges to produce certain values. Cyclic dependencies in this relationship cause incompatibility. Consider node \( i \), with input edges: \( x_1, ..., x_j, ..., x_r \). Let \( x_j \in \{0, ..., t_j\} \), and \( x_j \in \{0, ..., t_l\} \). Let MODC\(^j \) and MODC\(^i \) be the maximal set of observability don’t cares for the input edges \( x_j \) and \( x_l \) respectively. Let \( m_q \in P_1 \times ... \times P_r \) be a minterm in the local input space of \( \{ y_i \} \), such that \( m_q \in MODC^j \) and \( m_q \in MODC^i \). The primary input minterms that produce \( m_q \) will be used as don’t cares for both \( x_j \) and \( x_l \). The optimization of \( x_j \) as a result of \( m_q \) may destroy the validity of MODC\(^i \) and vice versa. The minterm in MODC\(^j \), for input edge \( x_j \), is said to be compatible with \( x_l, l \neq j \), if it is not a minterm in MODC\(^i \), or if MODC\(^i \) does not depend on the value of \( x_l \), i.e.

\[
CODC^i = \{ m | (m \notin MODC^i) \lor (\forall x_l(m) \in \text{MODC}^i), \quad m \in \text{MODC}^j \}
\]

CODC\(^i \) is the subset of MODC\(^j \) that is compatible with MODC\(^i \). By “\( \forall x_l \)”, we denote the computation of universal quantification over all values of \( x_l \). The interpretation of this formulae is that: of all the minterms, \( m \in MODC(f, x_j) \), where \( f \) is insensitive to \( x_j \), \( m \) is said to be compatible with another input edge \( x_j \), if (1) either \( m \) is not a don’t care for \( x_j \), or (2) no matter what value is chosen for \( x_l \), \( f \) is still insensitive to \( x_j \) under \( m \).

The input edges are implicitly ordered and the CODC for each input edge is computed by making the associated MODC compatible with all the preceding edges in the ordering. Given an ordering \( x_1 \prec ... \prec x_j \prec ... \prec x_r \), the CODC for edge \( x_j \) can be defined as follows:

\[
CODC(y_i, x_j) = \{ m | \forall l < j, (m \notin \text{CODC}^i) \lor (\forall x_l(m) \in \text{MODC}^j), \quad m \in \text{MODC}^j \}
\]

This approach gives the first edge in the ordering the most flexibility. Successive edges are more restricted in order to be compatible with previous ones. The ODC set for each successive node is thus reduced for compatibility.

In practice, the set of minterms in MODC can be represented symbolically using MDD’s [12] [13]. Also, the CODC set for each node can be inherited by all input edges. The formula thus can be constructed with MDD operations:

\[
\begin{align*}
CODC(y_i, x_j) &= P_l(P_2(...P_{r-1}(\text{MODC}(y_i, x_j)))) + \text{CODC}^i_x \\
CODC_x &= P_k(F) = \text{CODC}^i_x \cdot F + \forall x_k(F) \\
\text{CODC}^i_x &= \prod_{i \in \text{fanin}(x_i)} \text{CODC}(y_i, x_j)
\end{align*}
\]

CODC\(^i \) is the edge-CODC for edge \( x_j \rightarrow y_i \). CODC\(^i \) is the node-CODC for the fanout node \( y_i \). CODC\(^i \) is computed for each fanout edge of node \( y_i \) and they are intersected to give the node-CODC for \( x_j \). This is passed to all fanin nodes of \( x_j \). \( P_k(F) \) is the compatibility operation which is applied to each fanin node of \( y_i \) that precedes \( x_j \) in the pre-assigned order.

Theorem 2 (CODC) The set of minterms computed by (3) are don’t cares for node \( x_j \) and they remain to be don’t cares if the functions of all other nodes change within their computed CODC’s.

3.3 Maximal set of observability partial cares

CODC’s do not capture all the flexibility for MV-networks. For the input edge \( x_j \) of node \( i \), there are minterms such that \( y_i \) is insensitive to only a subset of the values for \( x_j \). In other words, minterms under which a subset of the values for \( x_j \) is
indistinguishable at $y_j$. These minterms are, by the definition given in Section 2, partial cares for the function implemented at node $x_j$, and can also be used in the minimization of $x_j$.

The maximal set of partial cares for edge $x_j \rightarrow y_j$ can be defined on the power set, $2^P$, of the values for $x_j$. Let $x_j \in P_j = \{0, \ldots, m\}$ and $v = \{r_1, \ldots, r_v \} \in 2^P$. For each subset of the values $v_j \in 2^P$ for $x_j$, there exists a set of minterms $S_i$ in the local input space, such that the values in the subset $v_j$ can not be distinguished at the output $y_j$. We compute these sets of minterms $S_i$ for each such subset of values $v_j$. When mapped into the primary input space, $S_i$ represents the set of partial care minterms for node $x_j$, where $x_i$ can take any value in $v_i$.

\[
OPC(f, v, x_j) = \{m|f(m[x_j = r_1]) = \cdots = f(m[x_j = r_v]), m \in P_1 \times \cdots \times P_j\}
\]

\[
MOPC(y_j, x_j) = \{(m, v)|m \in OPC(y_i, v, x_j), v \in 2^P\}
\] (4)

$OPC(f, v, x_j)$ gives the set of minterms in the local input space of $f$, such that a subset $v$ of the values for $x_j$ are indistinguishable at $y_j$, i.e. the output function of $y_j$ does not change, if we arbitrarily change the value of $x_j$ within the subset of values $v = \{r_1, \ldots, r_v\}$, while keeping the other parts of $m$ fixed. Therefore, MOPC by definition is a set of pairs, $(m, v)$, where $m \in P_1 \times \cdots \times P_j$, is a minterm in the local input space of $y_j$, and $v = \{r_1, \ldots, r_v\}$ is a subset of the values for $x_j$. Similar to (1) MOPC’s can be computed by the following formula if $f$ is deterministically specified.

\[MOPC(y_j, x_j) = \sum_{v \in 2^P} \left( v_1 \left( f_{x_1}^0 \cdot f_{x_2}^0 \cdots f_{x_j}^0 + \cdots + f_{x_1}^m \cdot f_{x_2}^m \cdots f_{x_j}^m \right) \right)\]

\[= \sum_{v \in 2^P} \left( v_1 \prod_{i=0}^n f_{x_i}^i \right)\] (5)

This computation can be expensive due to the power set summation. OPC needs to be computed for every subset of $|P_j|$, which is exponential. In practice multi-valued variables usually have a small set of values to choose from, thus suggesting the feasibility of (5). However algorithmic trade-offs need to be explored to assess the practicality of MOPC.

**Algorithm [CODC-based MV-network minimization]:**

**Input:** MV-network ntk

**Input:** external don’t care XDC, at each primary output $j$

**Local CODC:** CODC set for node $i$

**Local DC:** complete don’t care set for node $i$

Traverse each node $j$ in ntk in reverse DFS order

If $j$ is primary output

CODC$ _j = $ external don’t care (XDC$ _j$)

Continue;

For each fanout node $k$

$D = MODC(f_x, y_j)$

For each fanin node $i$ of $k$ that is already visited

$D = D + CODC_i$

CODC$ _j = CODC_j \cap D$

Collapse CODC$ _j$ into primary input space

Quantify out the variables not in the TFI cone of $y_j$

DC$ _j = \neg$ image (CODC$ _j$)

MINIMIZE (ONSET$ _j$, DC$ _j$)

End

Figure 3: CODC-based MV-network minimization

Like observability don’t cares, observability partial cares may also become invalid if the functions of related nodes are changed. Similar to the approach used in CODC, we can order the input edges for each node, and make the MOPC set compatible with each preceding edge. The advantages of COPC for MV-network minimization needs to be evaluated further and are not explored here.

4 Implementation

We give an algorithm to compute a set of CODC’s for each node in a multi-level MV-network. The algorithm is an extension of the binary CODC computation from [7]. The computation of MODC and CODC is implemented in a multi-valued logic synthesis infrastructure called MVSIS. MOPC and COPC computations are expensive and require a multi-valued relation minimizer for the optimization of each node; it is not implemented in the current version.

4.1 MV-network optimization

All computation is carried out using MDD operations. A heuristic depth-first search MDD variable ordering is implemented. The logic function of each node in the network is represented by a multi-valued table structure, as defined in VIS [14]. A table is a sum-of-products representation of a multi-valued function. Each row is partitioned into an input part and an output part, and represents a multi-valued cube.

The algorithm traverses the MV-network in a reverse topological order from primary outputs to primary inputs. Each node
in the network is traversed once. At each node, the CODC set is computed for each fanout edge, and then intersected to give the approximated CODC set for this node. Once calculated, the CODC set for each node is inherited by each of its own fanin nodes. The CODC set is mapped into the primary input space by variable substitution, and then mapped into the local input space by image computation. Given the DC set, the logic minimization of a multi-valued node is carried out using ESPRESSO-MV [15].

4.2 Multi-valued image computation

Two methods exist for image computation, transition relation and recursive range computation. We extend the recursive range computation from the binary domain to the multi-valued domain. For binary image computation, refer to [7]. Multi-valued cofactoring is used to reduce the computation in a recursive fashion.

In the local input space of node $y_i$, each input variable is cofactored by the complement of the don’t care set $A(x)$, which is an MDD in terms of primary input variables. This array of cofactored functions gives the transition functions that map the entire primary input space $PI$ into the local care set of node $y_i$.

$$F_{A(x)} = \left\{ (f_1)_{A(x)}, (f_2)_{A(x)}, \ldots, (f_r)_{A(x)} \right\}$$

$$A(x) = \text{CODC}_{y_i}^\text{local}(x)$$

Each $f_k$ is the multi-valued function for one of the fanins, $x_k$, of $y_i$. Each $f_k$ is represented by an array of MDD’s; each MDD represents the onset for one of the values of $f_k$. The cofactor $(f_k)_{A(x)}$ is obtained by constraining the MDD function for each value of $f_k$ against $A(x)$. This is called the constraining computation. Once we have the array of range functions, we apply output cofactoring to carry out the recursive image computation:

$$\text{CODC}_{y_i}^\text{local} = \text{IMAGE}\left(\text{CODC}_{y_i}^\text{local}\right) = \text{RANGE}\left(F_{A(x)}\right)$$

$$= \bigoplus_{k=0}^{\left|P_1\right|-1} x_k \cdot \text{RANGE}\left([f_2, \ldots, f_r]_{f_k}\right)$$

In the above formula, $x_k$ denotes the literal for the intermediate variable $x_i$ that takes the $k^{th}$ value. $\left|P_1\right|$ is the total number of values specified for $x_i$. The range computation is recursively applied to the list of functions, until every one has been cofactored. The final result is a set of cubes in the local input space of $y_i$, which can then be used in the minimization of node $y_i$.

<table>
<thead>
<tr>
<th>examples</th>
<th>#cubes</th>
<th>#literals</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>91</td>
<td>168</td>
</tr>
<tr>
<td>matmul</td>
<td>72</td>
<td>240</td>
</tr>
<tr>
<td>sort</td>
<td>594</td>
<td>2296</td>
</tr>
<tr>
<td>bakery-proc</td>
<td>79</td>
<td>208</td>
</tr>
<tr>
<td>coherence-ch</td>
<td>99</td>
<td>337</td>
</tr>
<tr>
<td>eisenberg-proc</td>
<td>57</td>
<td>140</td>
</tr>
<tr>
<td>slider-nsf</td>
<td>321</td>
<td>1210</td>
</tr>
<tr>
<td>car</td>
<td>23</td>
<td>92</td>
</tr>
<tr>
<td>iris</td>
<td>25</td>
<td>107</td>
</tr>
<tr>
<td>mm3</td>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td>mm4</td>
<td>27</td>
<td>109</td>
</tr>
<tr>
<td>mm5</td>
<td>55</td>
<td>231</td>
</tr>
<tr>
<td>balance</td>
<td>61</td>
<td>244</td>
</tr>
<tr>
<td>nursery</td>
<td>53</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td></td>
<td>208</td>
</tr>
</tbody>
</table>

Table 1: Results for fullsimp

The CODC-based MV-network minimization is implemented in MVIS as command fullsimp. As a comparison, we implemented command simp, which calls ESPRESSO-MV for each node directly without computing CODC’s. Table 1 shows the comparison in the number of multi-valued cubes and literals. There is about 20% gain by computing CODC’s. We combine fullsimp with algebraic decomposition as in [11] and form heuristic network optimization scripts like script.rugged in SIS. The script repetitively applies decomposition, full simplification and node elimination until no improvement. Table 2 shows the optimization results for the same set of examples. Note that only deterministic examples can be minimized and produce meaningful results. The results are verified by the combinational verification package in VIS. Also note that only $\left|P\right|-1$ values are represented in the sum-of-products form if a node has $\left|P\right|$ values. The value with the most cubes is treated as default and is not counted.

The experiments are performed on an Intel 500MHz machine with 128MB memory. The run time ranges from 1-10 minutes depending on the size of the example. The reduction in cube count and literal count is significant compared with the original specification. This translates into implementation cost savings whether in software or in hardware.
**Table 2: Results for optimization scripts**

<table>
<thead>
<tr>
<th>examples</th>
<th>cubes</th>
<th>literals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>orig</td>
<td>script</td>
</tr>
<tr>
<td>max</td>
<td>196</td>
<td>91</td>
</tr>
<tr>
<td>matmul</td>
<td>120</td>
<td>48</td>
</tr>
<tr>
<td>sort</td>
<td>594</td>
<td>158</td>
</tr>
<tr>
<td>bakery-vc</td>
<td>788</td>
<td>69</td>
</tr>
<tr>
<td>coherence-cch</td>
<td>120</td>
<td>66</td>
</tr>
<tr>
<td>eisenberg-vc</td>
<td>1341</td>
<td>57</td>
</tr>
<tr>
<td>slider-nsf</td>
<td>1359</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>518</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>598</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>2055</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>337</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>8640</td>
<td>40</td>
</tr>
</tbody>
</table>

6 Conclusion

We generalized the notion of don’t cares used in binary logic to multi-valued logic, and defined partial cares for multi-valued relations. We gave a method to construct multi-valued observability don’t cares in a combinational network. We showed how observability partial cares can be generated, and discussed their potential complexity. Experimental results were given to show the effectiveness of using CODC’s for node minimization.

Some future research directions are: (a) Devise heuristic algorithms to generate observability partial cares efficiently; (b) Apply a multi-valued relation minimizer in the optimization process, which uses partial cares; (c) Apply multi-valued logic minimization to other CAD areas, e.g. state encoding, software compilation and asynchronous hardware synthesis.

Acknowledgement

We are grateful for the support of the SRC under contract 684.004 and the California Micro program and industrial sponsors, Fujitsu, Cadence, Motorola and Synopsys.

References


