

10.7.2 Selective Node Elimination

The key operation in the process of node elimination is the replacement of a literal y_j which appears in a cover F_i by the algebraic SOP expression F_j , and similarly for the literal y'_j and the cover complement F'_j . The elimination process is driven by a quantity known as the elimination value, e_value_j , of node j . This is defined as the increase in the total number of literals in the network if F_j were substituted into its fanout to replace literal y_j , and the ensuing product expanded to SOP form (and similarly for F'_j). An approximation to this, valid when all products are algebraic, may be given as follows.

We consider a function F_i in the fanout of F_j to be represented by

$$F_i = \tilde{F}_i y_j + \hat{F}_i y'_j + R_i. \quad (10.9)$$

We are interested in the increase in the literal count when we substitute the SOP representation of F_j into this representation of F_i . If we do this for every node in the direct fanout of F_j we will have eliminated node j from the network. Because a small savings of literals can come at the cost of increased levels of logic and thus increased delay, we need to be able to eliminate nodes with small value. Also, because the process of factoring a two-level network into a multi-level network is heuristic, it is useful to be able to iterate. In order to iterate we will have to partially collapse the function in order to try a different factorization/decomposition strategy. This partial collapsing is done by node elimination.

Thus we now consider a single node in the fanout of node j and determine the increase in the number of literals associated with node elimination. In order to proceed with this computation we must agree as to the measure or value of a node in a Boolean network. There are two possible choices. The number of literals in the SOP representation of the node or the number of literals in the factored form representation of the node. The factored form literal count is useful for two reasons. First, it more closely approximates the actual transistor count for the final implementation of the node. Second, the literal count for the factored form is the same as the literal count for the complement of the function. Since we use both the function and its complement in node elimination having both values readily available is useful.

Therefore we first let n_i be the number of times that either y_j or its complement appears in the factored form representation of F_i . Further, we let L_i be the number of literals in the factored form for F_i . Now we see that an approximation to the number of literals in the factored form of F_i after elimination is $n_i L_j - n_i$, where the first term approximates the increase by substituting in F_j and the second term accounts for the elimination of y_j and its complement from the expression for F_i . Now, to find the complete e_value (cost) of the elimination for the entire network we need to sum over the fanout of node j and then subtract out the literal count of node j which will be eliminated. This leads to the following expression

$$e_value_j = \left(\sum_{i \in \text{fanout of } j} n_i (L_j - 1) \right) - L_j \quad (10.10)$$

For example, suppose a Boolean network contained nodes i and j with covers

$$\begin{aligned} F_i &= abd + cd + a'c'gh + b'c'gh + ef + ah, \\ F_j &= ab + c \\ F'_j &= a'c' + b'c'. \end{aligned}$$

Resubstitution of F_j into F_i would first obtain the quotient $Q = d$ and remainder $R = c'a'gh + c'b'gh + ef + ah$ and then $Q_C = gh$ and $R_C = ef + ah$, leading to the reduced expression

$$F_i = dF_j + (ghF'_j + ef + ah).$$

In the above resubstitution example, if we consider reversing the process and eliminating node j into node i the elimination value of node j is computed as follows.

$$\begin{aligned} L_j &= 3 \\ n_i &= 2 \end{aligned}$$

so, if i is the only node in the fanout of j , we have

$$\mathbf{e_value}_j = 2(3 - 1) - 3 = 1.$$

Note that the first equation accurately portrays the literal savings in the Boolean network accountable to node j , only if all the indicated products are algebraic. While this is indeed the case in the example given, this will not necessarily be the case in the context of multilevel logic optimization, in which Boolean operations may be freely intermixed with algebraic ones. In the more general case, it may be regarded as an approximation to the true elimination value.

As typically implemented, the elimination operation is an iterative process, carried out with respect to a specified value threshold, v . Thus the sis command “eliminate v ”, applied to a given Boolean network, would consider all nodes in the network in a certain order. For each node j encountered, if $\mathbf{e_value}_j < v$, F_j would be substituted into its fanout, and then node j eliminated from the network. On each pass through the network, a flag is initially lowered and then raised if any node is thus eliminated.

If the flag is still lowered at the end of any pass, then we know that no further elimination is possible, and the whole process terminates.

Iteration is necessary because subsequent node eliminations may affect the $\mathbf{e_value}$ of nodes already processed in the current pass. In the SIS program, the processing of nodes is done in depth first search (postorder), where the search is initialized at the primary inputs, and the elimination value of node j is attempted only after the conditional elimination of all nodes in the fanout of j has already been completed.

In the above example, node j would not be affected if $v \leq 0$, but if $v > 0$, the elimination operation on the Boolean network would put F_i back into its original form and eliminate node j , assuming node j has no other fanout.