Minimal Decomposition of Boolean Functions Using Non-Repeating Literal Trees

Joram Pe'er
National Semiconductor (I.C.)
Herzelia B. 46104, Israel
Tel. +972-9-594293
e-mail: joram@taux01.nsc.com

Ron Y. Pinter
IBM Science and Technology
Haifa 31905, Israel
Tel. +972-4-8296252, Fax: +972-4-8296114
e-mail: pinter@haifasc3.vnet.ibm.com

November 23, 1995

Abstract

We define and study the class of *non repeating tree* (NRT) representable Boolean functions. Our investigation provides new insights on Boolean function decomposition with applications to logic synthesis, as follows:

- We prove that NRT representable Boolean functions have a unique decomposition that is minimal in the number of edges in the Boolean graph.
- We present a top down, connectivity based decomposition technique which avoids division and generation of kernels.
- We investigate the abundance of NRT representable Boolean functions and show that for many functions, found in real world designs, this decomposition can be applied, and a minimal decomposition obtained.

Finally we show how this new technique can be extended to a heuristic for general Boolean functions.

1 Introduction and Overview

A common scheme for Multi-Level Logic (MLL) synthesis is to flatten the logic functions to a two level representation and then perform some heuristics to decompose this representation. Current techniques for decomposition are based on a recursive approach in which a function f is decomposed into $p \cdot q + r$, where p, q, and r are Boolean functions, and then the same decomposition heuristic is applied to them. During decomposition, a candidate divisor p is chosen out of a set of candidate divisor functions; the process of evaluating q and r is called *division.* If p and q have disjoint support then the division is called *algebraic,* otherwise it is *Boolean.* Algebraic division is faster to evaluate, but it may lead to results that are farther from the optimal [2].

In this paper we present a new, efficient approach to the problem of Boolean function decomposition. On our way to devise and analyze a new decomposition algorithm, we first confine the problem space to the class of *non repeating tree* (NRT) representable Boolean functions: these functions have the property that they can be represented by a Boolean computational tree in which every variable appears only once. After proving correctness and minimality of the algorithm in the confined space, we generalize the results to any Boolean function. Thus, the rest of this paper is organized as follows:

In Section 2 we define the NRT structure and prove some fundamental properties of NRT representable Boolean functions. Primarily, we show that the NRT representation of these functions is minimal in the number of edges in any Boolean graph representation of the given Boolean function. We also define the *cube connectivity graph* for a Boolean function's two level representation. The *clustering* properties of this graph are the key to NRT-based decomposition algorithms.

In Section 3 we develop a decomposition algorithm for retrieving the NRT representation of an NRT representable Boolean function. This algorithm is based on the reversal of a flattening algorithm, which in turn uses transformations from a sum of products representation to a product of sums representation and vice versa. We define several operators that are required by the flattening algorithm, and prove that all of them are *reversible.* We use this property to construct a new decomposition algorithm which does not rely on the division operation [2] and on generating the kernels of the Boolean function.

In Section 4 we investigate the abundance of NRT representable Boolean functions in a large sample space of Boolean functions gathered from the ISCAS benchmark [11, 12]. We conclude that many Boolean functions, found in real world designs, are NRT representable and thus have a minimal decomposition.

Finally, in Section 5, we present a heuristic that generalizes the NRT decomposition algorithm to general Boolean functions. We demonstrate its operation on several examples [14] and show that the results are comparable to those obtained by Boolean decomposition algorithms. Section 6 concludes by summarizing our results and putting them in the appropriate perspective.

2 Definition and properties of Non Repeating Tree (NRT) Representable Boolean Functions

In this section we define the notion of a non repeating tree (NRT) and the class of NRT representable Boolean functions. These functions have the property that they can be represented by a Boolean computational tree in which every literal appears only once. We also define the cube connectivity graph, the S2P and P2S operators, and examine the properties of NRT representable Boolean functions.

Definition D1.

- A *commutative node Boolean graph* is a connected directed acyclic graph $G(V, E)$ in which each node is labelled with one commutative operator (+ or *) or with a literal.
- Nodes with 0 fan-in are the *primary inputs* of the function; one node with 0 fan-out is the *primary output* of the function.
- A *concise commutative node Boolean graph* is a commutative node Boolean graph $G(V, E)$ where for each $e_{ij} = (v_i, v_j) \in E$, $op(v_i) \neq op(v_j)$, and for each $v_i \in V$, fan-in$(v_i) > 1$

unless v_i is a leaf representing a literal.

- A *non repeating tree* (NRT) is a concise commutative node Boolean graph which is a tree and in which any variable appears in no more than one leaf node.
- An *NRT representable Boolean function* is a Boolean function that can be represented as a non repeating tree (NRT).

Property P1.

If a Boolean function f is NRT representable, then f has an essential prime implicant representation.

Proof. Let $T_f(V,E)$ be an NRT representation of f. By recursive application of the distributive law, a two level representation is obtained. Each application of the distributive law involves terms which have no common variables as $T_f(V,E)$ is an NRT. Thus, after the removal of single cube containments, any two terms in the result differ by at least one variable. For any term, concatenating the variables which are not in it, in polarity opposite to how they appear in all the terms, yields a minterm that is covered by this term, and not covered by any other term. Thus f has an essential prime implicant representation. ∎

This property can also be proven by arguing that after the recursive application of the distributive law, a *unate* [3] cover is obtained; in fact, P1 is a corollary of Proposition 3.3.7 in [3].

Property P2.

If a Boolean function f is NRT representable, then f has two unique essential prime implicant representations: one is a product of sums, the other — a sum of products.

Proof. P1 implies that f has an essential prime implicant representation. If f has a sum of products prime implicant representation then it also has a product of sums prime implicant representation (and vice versa) [11]. ∎

Definition D2. Let $G_f(V,E)$ be a two level concise node Boolean graph. We define $C_f(V',E')$, the *cube connectivity graph*, as follows. For each Level 1 node $v_i \in V$, let $v_i' \in V'$. Define $e_{ij}' = (v_i', v_j') \in E'$ if the cubes rooted at v_i, v_j share a literal (*i.e.* $Support(v_i) \cap Support(v_j) \neq \emptyset$).

Definition D3. Let $G_f(V,E)$ be a two level concise node Boolean graph. $G_f(V,E)$ is a *cluster* if $C_f(V',E')$, the cube connectivity graph corresponding to $G_f(V,E)$, is a connected graph; otherwise, $G_f(V,E)$ is said to be a *multi-cluster* (*i.e.* $C_f(V',E')$ can be partitioned to several connected subgraphs which induce a partition of $G_f(V,E)$ into several clusters).

Definition D4. We define the following transformations:

- S2P: converts a sum of products representation to an irredundant product of sums.
- P2S: converts a product of sums representation to an irredundant sum of products.

Lemma L1.

Let $G_f(V,E)$ be a two level concise node Boolean graph that is a multi-cluster; then, applying S2P (P2S) to $G_f(V,E)$, yields a cluster.

Proof. We prove by choosing two arbitrary nodes in the cube connectivity graph of S2P($G_f(V,E)$) and showing there exists a path between them [14].

Property P3.

If a Boolean function f is NRT representable, then f has two unique essential prime implicant representations, one is a cluster and the other is a multi-cluster.

Proof. Let $T_f(V,E)$ be an NRT representations of f. Assume, without loss of generality, that the root operator of $T_f(V,E)$ is '+'. $T_f(V,E)$ is an NRT, so its Level 1 subtrees are also NRTs. From P2 we conclude that these subtrees can be flattened to a sum of prime implicants representation. As $T_f(V,E)$ is an NRT, the subtrees have disjoint support. The roots of the subtrees can be removed as they carry the same operator as the root of $T_f(V,E)$. So a two level prime implicant representation that is a multi-cluster is obtained. From L1 it follows that applying S2P to this representation yields a cluster. ∎

Property P4.

The NRT representation of an NRT representable Boolean function f is unique.

Proof. Assume $T1_f(V,E)$ and $T2_f(V,E)$ are two NRT representations of f. By applying the flattening procedure of P3 we obtain two prime implicant representations of f which are both multi-clusters. From P2 and P3 it follows that these two representations are equal. Thus the functions of the Level 1 subtrees of $T1_f(V,E)$ and $T2_f(V,E)$ are equal. Assume P4 is correct for functions with smaller support then f, then the Level 1 subtrees of $T1_f(V,E)$ and $T2_f(V,E)$ are equal and thus $T1_f(V,E)$ and $T2_f(V,E)$ are equal. ∎

2.1 Minimality of the NRT Representation

We now state that retrieving the underlying NRT representation for an NRT representable Boolean function guarantees a minimal decomposition in the sense of the number of edges in the Boolean graph.

Theorem 1. Let f be an NRT representable Boolean function and let $T_f(V,E)$ be its NRT representation. If the fan-in of all nodes in $T_f(V,E)$ which have operator nodes as sons is greater than 2, then $T_f(V,E)$ is a minimal representation of f in the sense of the number of edges in the Boolean graph.

Proof. By induction on the number of variables in the support of f; the details [14] are omitted for lack of space.

3 Decomposition of NRT Representable Boolean Functions

In this section we develop a top down algorithm for retrieving the underlying tree structure of an NRT representable Boolean function. This algorithm is based on the reversal of a flattening algorithm. To construct these algorithms, we define the following transformation:

Definition D5.

CONTRACT: given a commutative node Boolean graph $G_f(V,E)$ in which the operator of the root node (Level 0 node) is the same as the operators of nodes pointing to the root (Level 1 nodes), CONTRACT produces $G'_f(V', E')$ which is a commutative node Boolean graph in which the Level 1 nodes are eliminated; Level 2 nodes (which pointed to Level 1 nodes) now point to the root node.

3.1 The 2_Level_Of Flattening Algorithm

The following procedure returns a two level commutative node Boolean graph given a multi level concise commutative node Boolean graph:

```
2_Level_Of(Gf(V,E)):   /* Gf(V,E) is a concise commutative
                                           node Boolean graph */

1.      if LEVEL(Gf(V,E)) <= 3 then return Gf(V,E)

2.      else  foreach Gfi(Vi,Ei)
              such that ROOT(Gfi(Vi,Ei)) is a Level 1 node of Gf(V,E)
              {
2.1            Gf'(V',E') = 2_Level_Of( Gfi(Vi,Ei) ) /* flatten Gfi */

2.2            if (OP(ROOT(Gf'))=='*') Gf"(V",E"):=P2S(Gf'(V',E'))
               if (OP(ROOT(Gf'))=='+') Gf"(V",E"):=S2P(Gf'(V',E'))
               /* convert two level representation to sum of products
                  or product of sums so that
                             OP( ROOT (Gf) ) == OP( ROOT (Gfi) ) */
2.3            Gfi(Vi,Ei) := Gf"(V",E")
                             /* replace Gfi by flattened graph */
2.4            return CONTRACT( Gf(V,E) )
              }
```

2_Level_Of flattens $G_f(V, E)$ by flattening its Level 1 subgraphs $G_{f_i}(V_i, E_i)$ into a two level $G'(V', E')$. Depending on the operator at the root, G'_f is transformed from product of sums (sums of products) to sum of products (product of sums) using P2S (S2P). Since $G_f(V, E)$ was concise it follows that after the above transformation

$$OP(ROOT(G_f(V, E))) = OP(ROOT(G_{f_i}(V_i, E_i))).$$

The last step is CONTRACT which eliminates the redundant Level 1 nodes and returns a two level concise commutative node Boolean graph representing f.

Figure 1 illustrates the steps performed by 2_Level_Of() on an NRT representation of the NRT representable Boolean function depicted in (a). The bottom recursion step of 2_Level_Of() performs an S2P transformation at node `B`, which is then contracted into node `A` as shown in (b). P2S is then applied at node `A` and node `A` is contracted into node `C` as depicted in (c). Finally, S2P is applied to node `C` and contracted into node `D`, yielding the resultant two level representation shown in (d).

In case 2_Level_Of() is applied to an NRT representation, it returns a two level essential prime representation, as stated in the following theorem:

Theorem 2. Let $T_f(V,E)$ be an NRT representation of an NRT representable Boolean function f. Then 2_Level_Of($T_f(V,E)$) returns an irredundant two level sum (product) of essential prime implicants representation of f that is a multi-cluster. If OP(ROOT($T_f(V,E)$)) = '+' then 2_Level_Of($T_f(V,E)$) returns a sum of products. If OP(ROOT($T_f(V,E)$)) = '*' then 2_Level_Of($T_f(V,E)$) returns a product of sums.

Proof. By induction on the levels of $T_f(V,E)$; details [14] are omitted for lack of space. ∎

3.2 Reversability of the 2_Level_Of() Algorithm

Let us examine one invocation in the recursion of 2_Level_Of(). Following the reasoning of P3, 2_Level_Of() returns a multi-cluster. At step 2.1 a two level, essential prime implicant multi-cluster is returned. According to L1, step 2.2 transforms this multi-cluster into a cluster. As all subtrees of $G_f(V,E)$ have disjoint support, the clusters obtained at step 2.2 have disjoint support as well. Thus the CONTRACT operation creates a multi-cluster, but does not combine clusters. The CONTRACT operation can be reversed by breaking this multi-cluster into the clusters it encompasses. The reversability of Step 2.2 follows from P2.

3.3 Decomposition by Tree_of()

The following procedure retrieves the underlying Boolean tree structure of an NRT representable Boolean function f. Its input is a two level irredundant representation of f. Note that according to P4 this representation is unique. We also define the following operation:

Definition D6.

DECONTRACT: a procedure that partitions a two level Boolean graph into clusters and adds a root node to each cluster; the roots of the all clusters become the Level 1 nodes of the resultant graph.

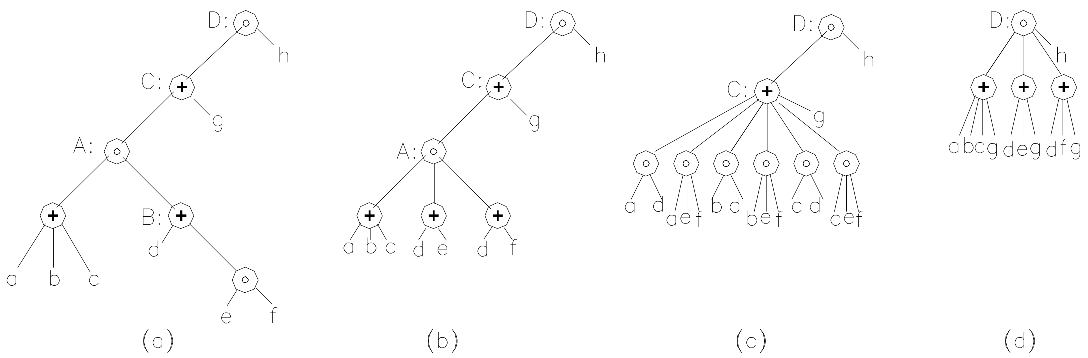

Figure 1: Example of flattening by 2_Level_of().

```
Tree_of( Gf(V,E) ):

/* Gf(V,E) is an irredundant two level Boolean graph of an NRT
                               representable Boolean function */

1    if (Gf(V,E) is a cluster)
1.1     then if ( OP(ROOT(Gf(V,E)))=="+" )
                then Gf"'(V"',E"') = S2P( Gf(V,E) )
                else Gf"'(V"',E"') = P2S( Gf(V,E) )
        else Gf"'(V"',E"') = Gf(V,E)

2    if (Gf(V,E) is an NRT)
        then return Gf"'(V"',E"')
        else {
2.1           Gf"(V",E") = DECONTRACT( Gf"'(V"',E"') )
2.2           foreach ( Gfi(Vi,Ei) s.t Gfi(Vi,Ei) is a subgraph
                          of Gf"(V",E")
                           && LEVEL( (ROOT(Gfi(Vi,Ei) ) == 1 )
              {
2.2.1          Gfi'(V',E') = Tree_of( Gfi"(V",E")
              }
3    return Gf"'(V"',E"')
```

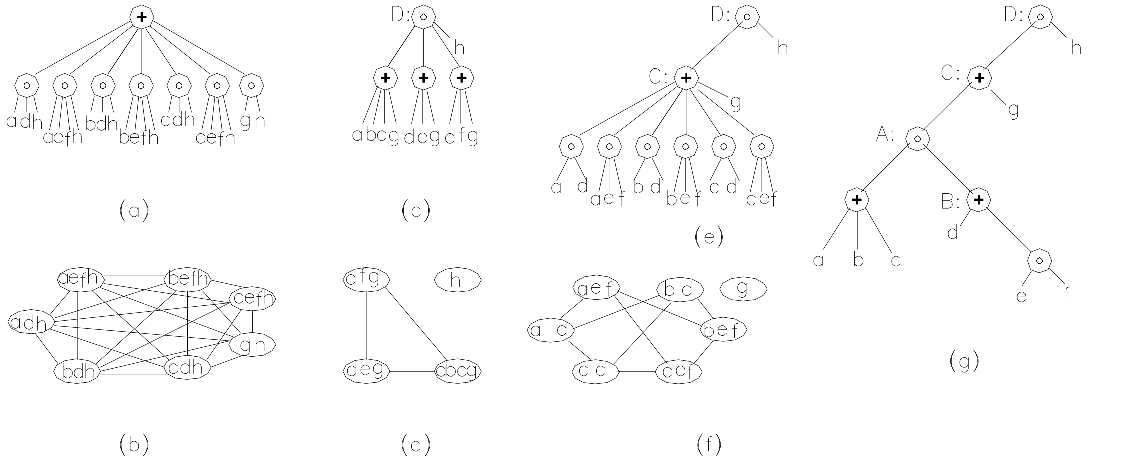

Figure 2: Example of decomposition by Tree_of().

Figure 2 illustrates the steps performed by Tree_of() on a two level representation of an NRT representable Boolean function depicted in (a). As seen from the cube connectivity graph depicted in (b), the two level representation (a) is a cluster. Thus the top recursion step of Tree_of() performs an S2P transformation at node `D`. The resultant representation is depicted in (c) and its corresponding cube connectivity graph is shown in (d). (d) suggests that (c) is a multi-cluster and thus DECONTRACT is performed at node `D`. The recursion continues on the

	Level 2			Level 3			Level 4			Level 5		
Circuit	No.	NRTs		No.	NRTs		No.	NRTs		No.	NRTs	
name	eqns	No.	%	eqns	No.	%	eqns	No.	%	eqns	No.	%
C432	142	79	55	142	91	64	114	19	17	111	18	16
C499	162	0	0	146	0	0	138	0	0	52	0	0
C880	430	312	73	278	157	57	257	103	40	247	52	21
C1355	496	326	66	409	168	41	377	96	25	350	50	14
C1908	822	810	98	734	569	77	648	283	43	361	0	0
C2670	1087	979	90	994	644	64	873	282	32	837	168	20
C3540	1588	1404	88	1550	1119	72	1464	764	52	1334	449	34
C5315	2108	1789	84	1844	1174	64	1805	396	21	1700	204	12
C6288	2159	1696	79	2147	603	28	2114	92	4	2143	109	5
C7552	3404	3290	97	3290	2410	73	3121	328	11	2919	367	14
Total	12398	10685	86	11534	6935	60	10911	2363	22	10054	1417	14

Table 1: The abundance of Non Repeating Tree representable Boolean functions in a sample of Boolean functions gathered from the ISCAS benchmarks.

subgraph rooted at node `C`. The subgraph rooted at node `C` is a cluster (its cube connectivity graph is the 3-node cluster in (d)) so P2S is applied at node `C`. The resultant representation is depicted in (e). The cube connectivity graph corresponding to the subgraph rooted at node `C` is depicted in (f). (f) suggests that the subgraph rooted at node `C` is a multi-cluster, thus DECONTRACT is performed at node `C`, producing node `A` and its subgraph. The recursion continues on the subgraph rooted at node `A` until the decomposed NRT representation is reached, as in (g).

Theorem 3. Let $G_f(V, E)$ be an irredundant two level representation of an NRT representable Boolean function f. Then Tree_of($G_f(V, E)$) returns the NRT representation of $G_f(V, E)$.

Proof. By induction on the levels of $T_f(V, E)$; details [14] omitted for lack of space.

4 The Abundance of NRT Representable Boolean Functions

Our objective is to asses the practicality of the presented algorithms in the context of a real world logic synthesis system. To do so, we examine the abundance of NRT representable Boolean functions in a large sample space of more than 10,000 Boolean functions generated from the ISCAS benchmarks. The abundance of NRT representable Boolean functions may also explain empirical results about the quality of level zero kernel decomposition [4, 8, 9].

As logic synthesis systems first do some flattening of the Boolean network before they start decomposition, we performed different levels of flattening on all available functions in the benchmarks and checked each function if it is NRT representable.

We used MISII [2, 8] to perform the flattening, and to perform decomposition used both GOOD_FACTOR and QUICK_FACTOR (we had to use both since GOOD_FACTOR failed in many cases at flattening levels 4 and 5; this also indicated to us that these were reasonable flattening levels). The results are summarized in Table 1.

From the results obtained we conclude that at flattening levels 2 and 3 most of the Boolean functions are indeed NRT representable. At deeper flattening levels the practicality of NRT decomposition degrades.

5 NRT Decomposition of General Boolean Functions

In this section we outline how the NRT decomposition algorithm can be generalized to work on general Boolean functions. A rigorous treatment can be found in [14]; here we present the algorithm, demonstrate its operation, and summarize its properties.

5.1 The Xtree_of() Algorithm

The difference between an NRT representable Boolean function and a general Boolean function is that the conditions of P3 are not met. Both $G_f(V, E)$ and S2P($G_f(V, E)$) (P2S($G_f(V, E)$)) may be clusters and DECONTRACT cannot always be performed.

DECONTRACT is replaced by the procedure Xdecontruct which partitions $G_f(V, E)$ into subgraphs such that the subgraphs are tightly connected and the number of edges between subgraphs is minimal. A root node is generated for each such subgraph.

The partition can be performed on either $G_f(V, E)$ or S2P($G_f(V, E)$) (P2S($G_f(V, E)$)). The decision is based on the *separability* function that is defined on the cube connectivity graph of $G_f(V, E)$. If $G_f(V, E)$ is a multi-cluster, then the separability of $G_f(V, E)$ equals 1; otherwise it is smaller. The partitioning is performed where the separability is the highest.

The procedure Xtree_of() presented here returns a Boolean tree in which literals may appear more than once. It is a generalization of the Tree_of() procedure. Its input is a concise two level Boolean graph representation of a Boolean function f.

```
Xtree_of (Gf(V,E)) :

/* Gf(V,E) is an irredundant two level Boolean graph
            representing a general Boolean function     */

1.  SEP = Separability (Gf(V,E))

    if (OP (ROOT (Gf(V,E)) == "+")
      then Gf"'(V"',E"') := S2P (Gf(V,E))
      else Gf"'(V"',E"') := P2S (Gf(V,E))

    SEP_INV := Separability (Gf"'(V"'))

1.2 if ( SEP s SEP_INV  )  Gf"'(V"',E"') := Gf(V,E)

    if ( Gf"'(V"',E"') is a Non Repeating  Tree )
      then return Gf"'(V"',E"')
      else  {
```

```
2.1          Gf"(V",E") := Xdecontract( Gf"'(V"',E"') )

2.2          foreach ( Gfi(Vi,Ei) s.t. Gfi(Vi, Ei)
                   is subgraph of Gf"(V",E")
               and LEVEL((ROOT(Gfi(Vi,Ei)) == 1) )

                {
2.2.1           Gfi(Vi,Ei) := Xtree_of( Gfi(Vi,Ei) )
                }

3.           return Gf"(V",E")  ;
            }
```

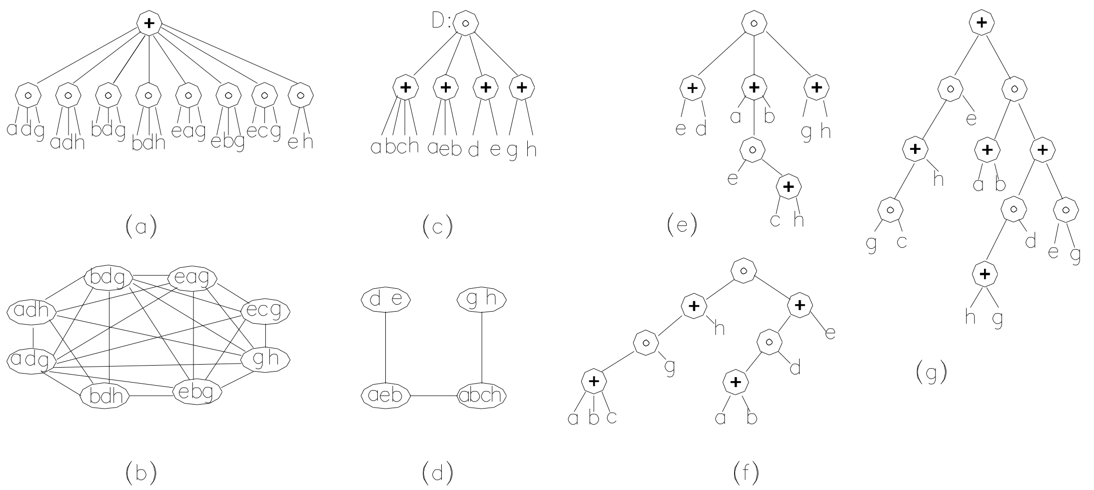

Figure 3: Example of decomposition by Xtree_of().

Figure 3 illustrates the steps performed by Xtree_of() on a two level representation of a general Boolean function `f = ((a+b+c)*g+h)*(e+d*(a+b))`. The two level input representation presented to Xtree_of() is depicted in (a). The corresponding cube connectivity graph is depicted in (b). Xtree_of() performs the S2P transformation in order to compute the separability of the product of sums representation of `f`. The product of sums representation obtained is depicted in (c). The corresponding cube connectivity graph is depicted in (d). Notice that both (b) and (d) are connected graphs so (a) and (c) are clusters. However, it is easier to partition (d) so the (c) representation is chosen to be acted upon by Xdecontract. (e) and (f) are two final decompositions obtained per different partitions of (d): (e) is obtained by a {de,aeb} {gh,abch} partition, whereas (f) is obtained by a {de} {aeb,abch} {gh} partition. The number of edges in the resultant graphs are 14 and 15, respectively, and the number of levels (in both) is 4. In comparison, the result obtained by kernel decomposition is shown in (g), which has 20 edges and is 5 levels deep.

It can be shown that the above procedure returns the NRT representation when applied to an NRT representable Boolean function [14].

6 Conclusions

We showed that the NRT representation of an NRT representable Boolean function is a minimal representation in the sense of the number of edges in the Boolean network. The number of edges in the Boolean graph is a good minimality criterion because in certain VLSI technologies [1] it linearly relates to the number of transistors in the physical implementation.

The algorithms presented have the virtue that they guarantee finding a minimal representation while existing algorithms do not. As the minimal representation is a tree, it also allows for optimal physical layout [13]. The drawback is that NRT decomposition handles only a subset of the Boolean space. However, we showed that in a large sample space of Boolean functions, these functions are quite abundant.

We further showed that the algorithms developed for NRT representable Boolean functions can be extended to a heuristic that handles general Boolean functions while they also guarantee finding a minimal representation when applied to NRT representable Boolean functions.

In [14] we show that applying the extended heuristic to several examples can produce considerably better results than algebraic decomposition. These results are comparable in quality to the results obtained by Boolean decomposition.

References

[1] Principles of CMOS VLSI design — a Systems Perspective; Neil Weste, Kamran Eshraghian, Addison-Wesley, 1985.

[2] Multi-Level Logic Synthesis; R. K. Brayton, A. L. Sangiovanni-Vincentelli, G. D. Hachtel, Proceedings of the IEEE, Vol. 78, No. 2 (February 1990), pp. 264–300.

[3] Logic Minimization Algorithms for VLSI Synthesis; Brayton, Hachtel, McMullen, Sangiovanni-Vincentelli, Kluwer Academic Publishers, 1984.

[4] Graph Based Algorithms for Boolean Function Manipulation. Randal E. Bryant, IEEE Transactions on Computers, Vol. 8, 1986.

[5] Logic Synthesis for VLSI. R. Rudell, Ph.D. thesis, University of California, Berkeley, 1989.

[6] The Decomposition and Factorization of Boolean Expressions. R. K. Brayton, Curt McMullen. IEEE Conference on Circuits and Systems, Rome, 1982.

[7] Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment; S. Malic, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, IEEE International Conference on Computer Aided Design, 1989.

[8] Factoring Logic Functions; R. K. Brayton. IBM Journal of Research and Development, Vol. 31, No. 2 (March 1987), pp. 187-198.

[9] Multi-Level Logic Optimization Systems; R.K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, ICCAD86, November 1986.

[10] Switching and Finite Automata Theory; Zvi Kohavi, McGraw-Hill, 1978, pp. 88-112.

[11] Accelerated ATPG and Fault Grading Testability Analysis; F. Brglez, P. Pownall, R. Hum, Proceedings of ISCAS'85, Vol. 2, pp. 695-698.

[12] A Fanout oriented Test Pattern Generation Algorithm; H. Fujiwara, Proceedings of ISCAS'85, Vol. 2, pp. 671-674.

[13] A Polynomial Algorithm for the Min-cut Linear Arrangement of Trees; M. Yannakakis, Journal of the Assoc. of Computing Machinery, Vol. 32, No. 4 (October 1985), pp. 950-988.

[14] Minimization of Boolean Computational Trees for Multi Level Logic Implementation of Combinational Networks. J. Peer, M.Sc. thesis, Technion — Israel Institute of Technology, Haifa, 1993.