

ECE 510 OCE BDDs and Their Applications

Lecture 15. Functional Decomposition

May 16, 2000
Alan Mishchenko

Overview

- The concept of functional decomposition
- Two uses of BDDs for decomposition
 - as a computation engine to implement algorithms
 - as a representation that helps finding decompositions
- Two ways to direct decomposition using BDDs
 - bound set on top (Lai/Pedram/Vardhula, DAC'93)
 - free set on top (Stanion/Sechen, DAC'95)
 - other approaches
- Disjoint and non-disjoint decomposition
- Implicit support minimization

Applications of Decomposition

- Multi-level FPGA synthesis
- VLSI design
- Finite state machine design
- Machine learning and data mining

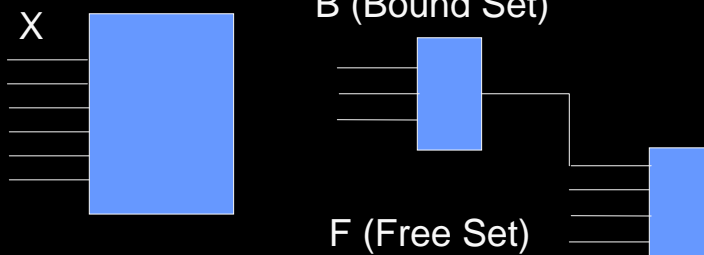
May 16, 2000

ECE 510 OCE: BDDs and Their Applications

3

Two-Level Curtis Decomposition

$$F(X) = H(G(B), F), \quad X = B \cup F$$



if $B \cap F = \emptyset$, this is disjoint decomposition

if $B \cap F \neq \emptyset$, this is non-disjoint decomposition

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

4

Column Multiplicity μ

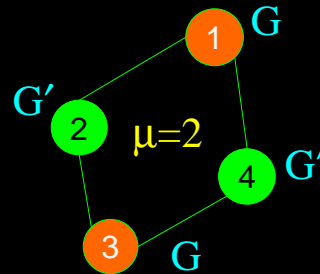
Bound Set = {a,b}

Free Set = {c,d}

	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	1	0	1
10	0	0	0	0

1 2 3 4

Incompatibility Graph



$$F(a,b,c,d) = (a'b' + ab)c' + (a'b + ab')(cd + c'd')$$

$$G(a,b) = a'b' + ab \quad H(G,c,d) = Gc' + G'(cd + c'd')$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

5

Multi-Level Curtis Decomposition

Two-level decomposition is iteratively applied to new functions H_i and G_i , until smaller functions G_t and H_t are created, that are not further decomposable.

One of the possible cost functions is **Decomposed Function Cardinality**. It is the total cost of all blocks, where the cost of a binary block with n inputs and m outputs is $m * 2^n$.

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

6

Typical Decomposition Algorithm

- Find a set of partitions (B_i, F_i) of input variables (X) into bound set variables (B_i) and free set variables (F_i)
- For each partition, find decomposition $F(X) = H_i(G_i(B_i), A_i)$ such that the column multiplicity is minimal, and compute DFC
- Repeat the process for all partitions until the decomposition with minimum DFC is found.

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

7

Uses of BDDs for Decomposition

- Whatever is the decomposition algorithm, BDDs can be used to **store data and perform computation** (using cubes, partitions, etc.)
- Alternatively, the algorithm may exploit the BDD structure of the function F to **direct the decomposition** in the bound set selection, column multiplicity computation, and deriving the decomposed functions G and H
- Finally, the process of decomposition may be made **fully implicit**

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

8

BDD-Based Decomposition

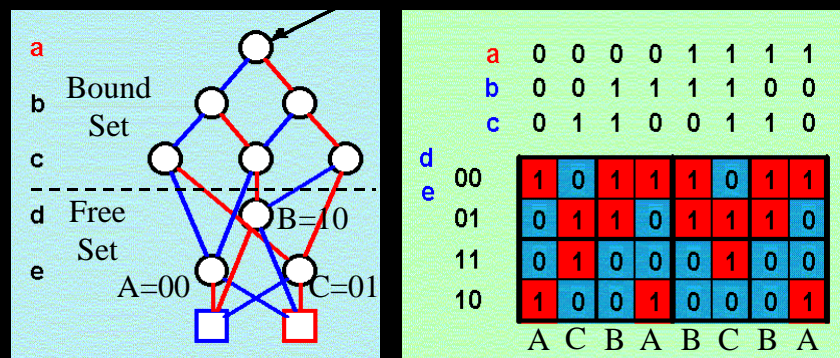
- Bound set on top (Lai/Pedram/Vardhula, DAC'93)
- Free set on top (Stanion/Sechen, DAC'95)
- Bi-decomposition using 1-, 0-, and EXOR-dominators (Yang/Ciesielski, ICDD'99)
- Recursive decomposition (Bertacco/Damiani, ICCAD'97)
- Implicit decomposition (Wurth/Eckl/Legl, DAC'95)

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

9

Bound Set on Top (Function G)



$$G = \{g_1, g_2\}, A = g_1'g_2', B = g_1g_2', C = g_1'g_2$$

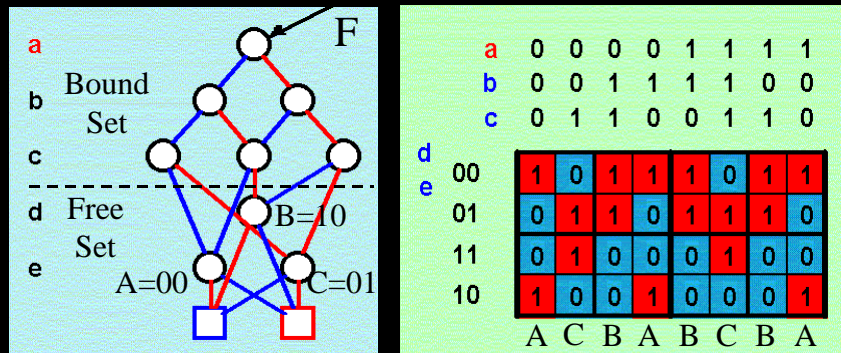
$$g_1 = a'bc + ab'c + abc', g_2 = a'b'c + abc$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

10

Bound Set on Top (Function H)



$$F(a,b,c,d,e) = H(g_1(a,b,c), g_2(a,b,c), d, e)$$

$$H = g_0'g_1'e' + g_0g_1'd' + g_0'g_1e$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

11

"Bound Set on Top" Algorithm

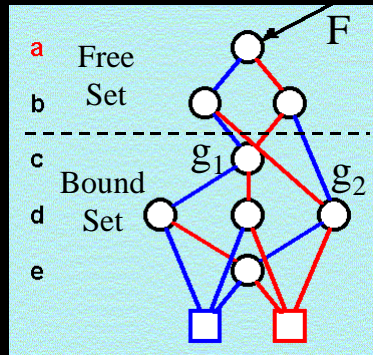
- Reorder variables in BDD for **F** and check column multiplicity for each bound set
- For the bound set with the smallest column multiplicity, perform decomposition (derive functions **G** and **H**)
- Iteratively, repeat the process for functions **G** and **H** (typically, only **H**)
- *This algorithm can be modified to work for non-disjoint decompositions but does not work with DCs*

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

12

Free Set on Top (Function G)



a	0	0	0	0	1	1	1	1
b	0	0	1	1	1	1	0	0
c	0	1	1	0	0	1	1	0
d	0	0	0	0	0	0	0	0
e	0	0	1	1	0	0	1	1
11	1	1	1	1	1	1	1	1
10	0	1	1	1	0	1	1	1

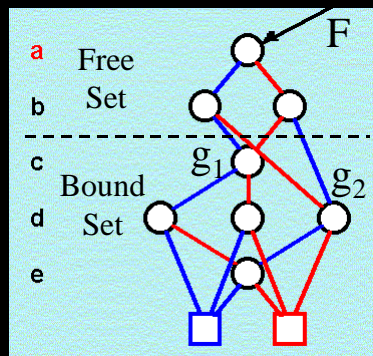
$$G = \{g_1, g_2\}, g_1 = c'de + cd, g_2 = d + e$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

13

Free Set on Top (Function H)



a	0	0	0	0	1	1	1	1
b	0	0	1	1	1	1	0	0
c	0	1	1	0	0	1	1	0
d	0	0	0	0	0	0	0	0
e	0	0	1	1	0	0	1	1
11	1	1	1	1	1	1	1	1
10	0	1	1	1	0	1	1	1

$$F(a,b,c,d,e) = H(a, b, g_1(c,d,e), g_2(c,d,e))$$

$$H = (a'b' + ab)g_1 + (a'b + ab')g_2$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

14

"Free Set on Top" Algorithm

- Find good variable order
- Derive implicit representation of all feasible cuts on the BDD representing **F**
- Use a cost function to find the best bound set and perform decomposition
- Repeat the process for functions **G** and **H**
- This algorithm is faster than "bound set on top" but it does not work for non-disjoint decompositions and with DCs

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

15

Non-Disjoint Decomposition

- Non-disjoint decomposition can be reduced to disjoint decomposition by adding variables
- Bound Set = $\{a,b,c\}$, Free Set = $\{c,d\}$
Disjoint decomposition can be generated by introducing variables $c_1=c_2=c$ instead of c
- In terms of the Karnaugh map, it is equivalent to introducing two variables instead of one in such a way that $c_1c_2' + c_1'c_2$ is a don't care set

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

16

Non-Disjoint Decomposition Example

a	0	0	1	1
b	0	1	1	0

c	00	1	1	0	1
d	01	0	0	0	0
	11	0	1	1	1
	10	1	0	0	0
		A	B	C	B

c1	0	0	0	0	1	1	1	1
a	0	0	1	1	1	1	0	0
b	0	1	1	0	0	1	1	0

c2	00	1	1	0	1	-	-	-	-
d	01	0	0	0	0	-	-	-	-
	11	-	-	-	-	1	1	1	0
	10	-	-	-	-	0	0	0	1
		A	A	B	A	B	B	B	A

There is no disjoint decomposition with any bound set;
 there is non-disjoint decomposition with bound set
{a,b,c}

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

17

Relation Between Decomposition Types

- Decomposition of completely specified functions is the simplest case
- Decomposition of incompletely specified functions is closely related to non-disjoint decomposition
- Decomposition of relations is the generalization of functional decomposition
- Decomposition of multi-valued relations is the most general decomposition considered so far

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

18

Implicit Support Minimization

- An incompletely specified function over n variables $F(x_1, x_2, \dots, x_n): B^n \rightarrow B$ can be represented by an interval $L \leq F \leq U$
- The **problem of support minimization** is to find a function $f(x_{i_1}, x_{i_2}, \dots, x_{i_k})$ depending on the minimum number of variables from (x_1, x_2, \dots, x_n) such that $L \leq f \leq U$
- **Lemma:** Variable x_i can be removed from the support of $F(x_1, x_2, \dots, x_n)$ iff it is true that
$$[\exists x_i L(x_1, x_2, \dots, x_n) \Rightarrow \forall x_i U(x_1, x_2, \dots, x_n)] \equiv 1$$

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

19

Pseudo-Code for FindSupports()

```
bdd FindSupports(bdd L, bdd U)
{
  if ( L = 0 || U = 1 ) return 1;
  if ( ( L >> U ) != 1 ) return 0;
  // check cache for results
  xi is the top variable in L and U
  // solve subproblems
  P0 = FindSupports(  $\exists x_i L$ ,  $\forall x_i U$  );
  P1 = FindSupports( Lxi', Uxi' ) & FindSupports( Lxi, Uxi );
  bdd P = ITE( xi, P1, P0 );
  // insert into cache
  return P;
}
```

May 16, 2000

ECE 510 OCE: BDDs and Their Applications

20

Source Code for FindSupports()

```
bdd FindSupports( const bdd L, const bdd U )
{
  if ( L == bddfalse || U == bddtrue ) return bddtrue;
  if ( ( L >> U ) != bddtrue ) return bddfalse;
  // check cache for results
  int TopVarL = bdd_var( L ); int TopVarU = bdd_var( U );
  int TopVar = ( TopVarL < TopVarU )? TopVarL: TopVarU;
  bdd TopVarBdd = bdd_ithvar( TopVar );
  // find cofactors
  bdd L0 = bdd_restrict( L, !TopVarBdd ); bdd L1 = bdd_restrict( L, TopVarBdd );
  bdd U0 = bdd_restrict( U, !TopVarBdd ); bdd U1 = bdd_restrict( U, TopVarBdd );
  // solve subproblems
  bdd P0 = FindSupports( bdd_exist( L, TopVarBdd ), bdd_forall( U, TopVarBdd ) );
  bdd P1 = FindSupports( L0, U0 ) & FindSupports( L1, U1 );
  bdd P = bdd_ite( g_XVars[TopVar], P1, P0 );
  // insert into cache
  return P; }
```