**Chapter 16**

**Cube Calculus**

**Marek Perkowski**

There are three basic representations to describe Boolean Functions: Binary Decision Diagrams, Cube Calculus and AND/NOT graphs. Many efficient modern logic synthesis programs use cube calculus to represent and process Boolean functions. This representation is used in U.C.Berkeley programs, including the well-known Espresso [Brayton84], MIS II and SIS, and many others [Perk88, Perk89, Song93, Song93b]. Cube calculus has been also used to represent Multiple-Valued logic functions. This calculus has been used for Boolean minimizing programs (minimizers), tautology and satisfiability checkers, logic verification programs (verifiers) and other software tools [Perk92].

In this chapter, the concept of a set is presented first because it is used as a fundament of cube calculus; then the concepts of cube and the array of cubes are presented. The last part of this chapter presents positional notation of cubes and how to carry out the cube calculus operations in positional notation. This chapter is a link between theoretical cube calculus as an algebra and a hardware of a specific machine to realize cube calculus.

**16.11. Sets**

A set is a collection of objects called elements or members. As listed in Table 16.1, we use curly braces to indicate sets.

For instance, the set of all integers between 0 and 5 is written as:



the infinite set of all positive, odd integers can be described by:



The membership of a element ***a*** in a set ***A*** is denoted by ***a∈A*** to mean “*a is an element of A*”. A set which has no element is called an empty set (denoted by ***φ***). The empty set is a subset of all sets. The elements contained in a set are either listed explicitly or described by their properties (implicit description).

The following relations between two sets are used in cube calculus:

• Two sets ***A*** and ***B*** are equal, or identical, if they contain precisely the same elements. It is denoted by ***A=B.***

• A set ***A*** is said to be a subset of ***B*** if every element of ***A*** is also an element of ***B***. It it denoted by ***A  B.***

• If **A ⊆ B**, and B contains at least one element which is not contained in ***A***, then ***A*** is said to be proper subset of ***B***. It is denoted by ***A B***.

The elements of the sets are taken from universal set (***U***). The following basic set operations are used in cube calculus:

• The complement of ***A*** in universal set ***U*** (denoted by  ***A***) is the set of all elements of ***U*** that are not elements of ***A***.

• The intersection of A and B (denoted by ***A*** ***B***) is the set containing the elements that are in both A and B.

• The union of A and B (denoted by A B) is the set containing the elements that are in either A or B.

**Example 16.1.** Assuming that the universal set U is {0,1,2,3,4,5}, a set A is {0,1,2,3} and another set B is {2,3,4}. Then A={4,5}, A B={2,3}, and A B={0,1,2,3,4}.

The universal set U of possible values of a binary variable is {0, 1}. For a binary variable a, literal $\overbar{a}$ means that literal is true when variable a is 0, and can be described by $a^{\{0\}}$ where {0} is the true set of literal $\overbar{a}$. More detailed discussion on sets can be found in [35,36].

**16.2. The Concept of a Cube**

The basic concepts of cube calculus are a cube and an array of cubes. A cube is a product of literals. For example, product $\overbar{a}$bc$\overbar{d}$ is a cube. An array of cubes is a sum of cubes. A logic function can be represented by a cube or by an array of cubes. For instance, a simple 2-input binary logic function AND can be described by a cube as: fAND (a,b) = ab; another simple 2-input binary logic function XOR (exclusive OR) can be described by an array of cubes as 

In a binary logic, a literal is a binary variable with negation or without negation (x or $\overbar{x}$). In a multi-valued logic a literal  is a variable  with its set of values  for which the variable is true.

A multi-valued input, two-valued output, incompletely specified switching function (multi-valued function for short) is a mapping:

 **16.1.**

where  is a multi-valued (pi-valued) variable, is the set of values that variable xi may assume, B={0,1,x} ( x denotes a don't care value). Value n denotes the number of variables (number of positions). For any subset  representing the function such that:

**16.2.**

 is called true values set (true set for short) of literal . For example, a four-valued input logic of, which means if x=1 or x=2 or x=3; otherwise, =0. We always assume that the set of possible values of a n-valued logic variable is.

A product of literals, , is referred to as a product term (also called product or term for short), and can be represented by a cube. A product term that includes literals for all function variables  is called a full term. Any literal of the form  is always equal to 1 since the literal is true for all possible values of  can be written as.

A sum of products is denoted as a *Sum-Of-Products (SOP) Expression* while a product of sums is called a *Product-Of-Sums (POS) Expression*. An EXOR of products will be called a *Exclusive Sum Of Products expression ESOP*. A product of EXORs will be called a *Product Of Exclusive Sums expression (POES)*. SOPE, POSE, ESOP and POES are all represented as an array of cubes. Products of SOPEs (PSOPEs) are also used in *Generalized Propositional Formulas*. They are represented as arrays of arrays of cubes.

The degree of a cube is the number of literals in the cube that are not equal to one (in other word, ).

**Example 16.2.** The degree of binary cube is 3 (assuming a, b, c and d are binary variables).

The difference of two cubes is the number of variables for which the corresponding literals have different true sets. The distance of two cubes is the number of variables for which the corresponding literals have disjoint true sets.

**Example 16.3.** Given two cubes A=, B=. The difference of cubes A and B is 2 because they have different true sets on variables a and b. The distance of cubes A and B is 1 because they have disjoint true sets on variable b.

**16.3. Cube Calculus Operations**

The cube calculus operations presented in this book can be categorized into three groups:

1. Simple combinational operations,

2. Complex combinational operations, and

3. Sequential operations.

Each cube operation has one or two operand cube(s). Cubes A and B are used to represent these arguments and they can be described by:

16.3.



16.4.

where  are the true sets of literal , respectively. Value n is the number of variables. In this chapter, S represents the true set of a literal, the subscript of S represents the index of the literal (index of a variable corresponding to this literal), the superscript of S represents the operand cube (A or B).

**16.3.1. Simple Combinational Cube Operations**

Simple combinational cube operations are defined as single set operations. Such a set operation is applied on all pairs of true sets of corresponding literals of operand cubes. A simple combinational cube operation produces one resultant cube. The intersection and the *supercube* are simple combinational cube operations presented in this section.

**Intersection**

The intersection of two cubes A and B is the cube that is included in both A and B. The intersection operation of cubes A and B is defined as follows:

**16.5.**

where  is a set intersection operation. φ denotes an empty set, and ∈ denotes a contradiction.

**Example 16.4.** Assuming two cubes A = ab and B = b$\overbar{c}$, where a, b and c are binary variables. The intersection of two cubes A and B is the following:





 

**Figure 16.1. Intersection operation example.**

Example 16.4 is illustrated in Figure 16.1 by Karnaugh map. The intersection operation can be used in Ashenhurst/Curtis functional decomposition [40, 41].

**16.3.1.2 Supercube**

The supercube of two cubes A and B is the smallest cube that includes cubes A and B. The supercube operation of cubes A and B is defined as follows:

**16.6.**

Where  is a set union.

**Example 2.5** The supercube on two cubes A and B used in Example 2.4 follows:





*Figure 16.2. Example of Supercube Operation*.

Example 16.5 is illustrated in Figure 16.2 by Karnaugh map. The supercube operation can be used in graph coloring.

**16.3.2 Complex combinational cube operations**

Complex combinational cube operations are defined by two set operations and one set relation. These two set operations are called before (***bef*** for short) and active (act for short). All variables whose pair of true sets satisfy relation are said to be special variables. The active set operation is applied on all pairs of true sets of special variables. The before set operation is applied on the others. A complex combinational cube operation produces one resultant cube like a simple combinational cube operation. To this group of operations belong ***Prime, Cofactor*** and

Prime is an example of a complex combinational cube operation presented in this section.

**Prime**

The prime operation of two cubes A and B is defined as:

**16.7.**

Where the relation for the prime operation is . The active set operation is , and the before set operation is . In the above equation, variables  and  are the special variables.

**Example 2.6** Assuming two cubes A =  and B = , where  and  are binary variables. The prime of  is defined as follows:



Because,



Variable x2 and x4 are special variables. Therefore,





*Figure 16.3. Prime operation example*

Example 16.6 is illustrated in Figure 16.3 by Karnaugh map. The prime operation is used in the ESOP minimization program EXORCISM, developed by Perkowski and his former students [29, 30].

**Consensus**

The consensus operation on cubes A and B is defined as follows:



16.8

 16.9

where . For basic consensus operation, the before set operation is , the active set operation is , and the relation is always true.

**Example 16.7.** Assuming two cubes A=  and B = , where  and  are binary variables. Because the distance of cubes A and B is 1, then cubes A and B have consensus as follows:



Because:



Variable x2 is a special variable. Therefore,





*Figure 16.4. Prime operation example*

Example 16.7 is illustrated in Figure 16.4 by Karnaugh map. The consensus operation is used for finding prime implicants, and finding prime implicants is a basic step of the well-known Quine-McCluskey algorithm that is used for two-level logic minimization. Consensus is also used in its variants [23], as well as many other basic algorithms for two-level, three-level and multi-level logic minimization and machine learning [33]. We should have example of multiple-valued consensus, as in binary consensus the variable for which the argument cubes are disjoint is removed as union of two values is unity. But in multiple-valued consensus the union of disjoint values can be not a unity, so variable will still exist with a union of values of arguments in its value.

**Cofactor**

The cofactor operation of two cubes A and B is defined as:

16.10



16.11

where the relation for cofactor operation is . The result of the active set operation is always U (universal set), and the before set operation is . In the above equation, variables xk and xl are special variables.

Assuming two cubes A=  and B= , where  and  are binary variables. The cofactor of A|B is defined as follows:



Because:



variable x1 and x4 are special variables. Therefore,



This example is illustrated in Figure 16.5 by a Karnaugh map. In the Karnaugh map, first we calculate the intersection of cubes A and B, the intersection result is shown in Figure 16.5(b), then we remove variable x1 which means in the result cube, variable x1 can be either 1 or 0 (don't care); the result cube is shown in Figure 16.5(c). If there is no intersection of two operand cubes, then the cofactor is an empty cube. The cofactor operation is an important operation used in functional decomposition [40, 41], creation of decision diagrams, and many other applications.

  

*Figure 16.5. Cofactor operation example.*

**16.3. Sequential cube operations**

All sequential cube operations are defined as a single formula that has three set operations and one set relation. These three set operations are called before (bef for short), active (act for short) and after(aft for short), respectively. All variables whose pair of true sets S{i}{A} and S{i}{B} satisfy relation are said to be special variables.

The sequential cube operations produces m resultant cubes, where m is the number of special variables for a given operation. The sequential cube operations can be generally described by the following fundamental equation:



16.12

where  is a special variable, bef, act and aft are set operations. Every special variable produces a resultant cube according to Equation 16.12. This equation is a general pattern of all cube calculus operations and it was mentioned in the introduction.

**Crosslink**

The crosslink operation on cubes A and B produces an array of cubes defined as:



  16.13

For crosslink operation, the before set operation is , the active set operation is , the after set operation is , and the relation is . The crosslink operation can only be applied to two cubes when the two operand cubes are of the same degree, and X (don't care) must be in same position(s).

**Example 16.9.** Assuming variables  are binary, two cubes  and the crosslink operation  follows:

****

Because:



The variables x1 and x2 are special variables. And the two resultant cubes are:



Therefore,





*Figure 16.6. Crosslink example*

The Example 16.9 is illustrated in Figure 16.6 by Karnaugh map. The crosslink operation can be used in the minimization of logic functions in some canonical forms based on EXOR logic, for instance, in the Generalized Reed Muller form [29], as well as the general-purpose AND/EXOR expression (non-canonical), called ESOPs. The function.



*Figure 16.7. A complex crosslink example*

Can be realized using exor gates as



Another more complex example is shown in Figure 16.7. As shown in Figure 16.7(a), we have a function, where A, B, C and D are four cubes. First, we calculate C  D and obtain cubes E and F, and the function becomes as shown in Figure 16.7(b). Second we calculate  and obtain cubes G and H; therefore, the function is simplified as.

The crosslink operation is used in ESOP minimization program EXORCISM, developed at PSU by Martin Helliwell in 1988/89 [29,30]. A more powerful cube operation, called exorlink, and a new ESOP minimization programs EXORCISM-MV-2 and EXORCISM-MV-3 based on variants of exorlink operation were developed also at PSU by Song Ning and Alan Mishchenko, respectively [16, 32, 43].

**Sharp**

The (non-disjoint) sharp operation on cubes A and B is defined as follows:

 16. 14



16.15

For sharp operation, the before set operation is , the active set operation is , the after set operation is , and the relation is .

**Example 16.10.**

Assuming variables x1, x2, x3 and x4 are binary, two cubes A = $\overbar{ x\_{3}}$ and B = x2x4, the sharp operation A # B follows:



Because:



variables x2 and x4 are special variables. Thus, 2 resultant cubes are:



Therefore,





*Figure 16.8. Sharp example*

Remember, universal set U of possible values of a binary variable is {0,1}, therefore, . This example is also illustrated in Figure 16.8 by a Karnaugh map. The sharp operation can be used in the tautology problem [33].

**Disjoint sharp**

The disjoint sharp operation on cubes A and B is defined as follows:



16.17

For disjoint sharp operation, the before set operation is , the active set operation is , the after set operation is , and the relation is .

**Example 16.11.** The disjoint sharp operation A #d B, where A and B are used in Example 16.5, is calculated as follows:

Since the relation of disjoint sharp is the same as sharp, therefore variables x2 and x4 are still special variables. Thus, two resultant cubes are:





Therefore,





*Figure 16.9. Disjoint sharp example*

The Example 16.11 is also illustrated in Figure 16.9 by a Karnaugh map. The disjoint sharp operation can be used in tautology problem [hach96] and in conversions between SOP and ESOP representations.

**2.3.4. Summary of cube calculus operations**

From the above formulas (16.3 to 16.15), it is can be seen that sequential cube operations are the most complex operations in three groups of cube operations. Every sequential cube operation is defined by three set operations and one set relation. For the consistency of description, all cube operations in these three groups can be generally described by three set operations and one set relation. For simple combinational cube operations, only one set operation is used (called before); For complex combinational cube operations, two set operations and one set relation are used.

All cube operations (some of them are basic operations) described in this chapter are summarized in Table 16.1. Every row describes one cube operation. For each operation, its name, notation, set relation and three set operations (called output functions in the Table) are listed from left to right, respectively. The consensus operation is still to be investigated as it is similar to sequential operations in the table but it belongs to complex combinational operations such as prime and cofactor.

*Table 16.1. Cube Calculus Operations*



S iA ∩ S i B = ∅

**16. 4. Positional Notation and Cube Operations in Positional Notation**

From the above section, it can be seen that all cube operations are broken down into several set relations and set operations, and it is easy to carry out these set relations and set operations by hand. Now, the problem is how to represent sets in some way that they can be processed most efficiently by computers. Our answer to this problem is the positional notation.

**16.4.1. Positional Notation**

In *Positional notation*, every possible value of a variable (binary or multi-valued) is represented by one bit, 0 or 1. Thus, a p-valued variable is represented by a string of p-bit; The i-th possible value is represented by the i-th bit. If the literal of this variable is true for a specific possible value (say the i-th possible value), the corresponding bit (the i-th bit) is set to 1, otherwise, it is set to 0.

For example, a four-valued variable x is represented by a string of 4 bits. Literal x{0,2} is represented by 1010 because the first and third possible values let the literal be true.

The positional notation for binary literals is shown in Table 16.2. The don't care means the variable can be either 0 or 1, so both bits are set to 1. The contradiction means that the literal is not true for any possible value of variable, so both bits are set to 0. The last two cases, don't care and contradiction, can be extended to multi-valued variables. For p-valued variable, the string of 1's (the number of 1's is p) presents a don't care, and the string of 0's (the number of 0 is p) presents a contradiction.

*Table 16.2. Positional Notation for binary literals*



**16.4.2. Set operations in positional notation**

As listed in Table 16.1, all set operations used in cube operations are based on three basic set operations: intersection, union and complement. These three set operations can be executed using bitwise operations in positional notation:

* The set intersection operation can be executed using bitwise AND on two strings of bits that represent two true sets of literals in positional notation.

**Example 16.12.** Assume two literals x{0,1,2} and x{0,2,3}, where x is a 4-valued variable. Thus two true sets of these two literals are {0, 1, 2} and {0, 2, 3}, respectively. The intersection of these two true sets is {0,1,2}{0,2,3}={0,2}. In positional notation, set {0, 1, 2} is represented by 1110, and set {0, 2, 3} is represented by 1011. The bitwise AND of 1110 and 1011 is 1010, which means set {0, 2}, and this is just what we want. Therefore, the set intersection operation is executed by bitwise AND in positional notation.

* The set union operation can be executed using bitwise OR on two string of bits that represent two true sets of literals in positional notation.

**Example 16.13.** Assume two literals x{0,2} and x{3}, where x is a 4-valued variable. Thus two true sets of these two literals are {0, 2} and {3}, respectively. The union of these two true sets is {0, 2}{3}={0, 2, 3}. In positional notation, set {0, 2} is represented by 1010, and set {3} is represented by 0001. The bitwise OR of 1010 and 0001 is 1011, which means set {0, 2, 3}, and this is just what we want. Therefore, the set union operation is executed by bitwise OR in positional notation.

* The set complement operation can be executed using bitwise NOT on the string of bits that represents the true set of literal in positional notation.

**Example 16.14.** Assume a literal x{0,2}, where x is a 4-valued variables. Thus the true sets of the literal is {0, 2}. The complement of the true set is  (the U = {0, 1, 2, 3} for 4-valued variable). In positional notation, set {0, 2} is represented by 1010. The bitwise NOT of 1010 is 0101, which means set {1, 3}, and this is just what we want. Therefore, the set complement operation is executed by bitwise NOT in positional notation.

All other set operations can be done by combining these three basic set operations.

**Example 16.15.** Assume two literals S{A} = x{0,2} and S{B} = x{2,3}, where x is a 4-valued variables. Thus two true sets of these two literals are S{A} = {0, 2} and S{B} = {2, 3}, respectively. The set operation is: S{A} (S{B})=1010 AND (NOT 0011)=1010 AND 1100=1000 where AND and OR are bitwise operations. The result 1000 represents set {0}, which is correct result. This kind of set operation is called set difference, and is used in sharp and disjoint sharp cube operations.

**16.4.3. Set relations in positional notation**

The result of set relation is true or false and can be represented by one bit, 1 presents true and 0 presents false. The set relation cannot be done by bitwise function because it is the function of all bits of two operand sets in positional notation.

Set relation is broken down into two parts in positional notation, partial relation and relation type. The partial relation determines whether or not a pair of the same possible value of two literals satisfy the relation “locally". The relation type determines the method of combining partial relations.

Assuming there are two literals x{A} and x{B}, where x is p-valued variable x, A is positional notation of true set of literal x{A}, and A=[a0,a2,…..ap-1], where ai presents the (i+1)-th possible value of the literal (Note: the possible value starts with 0, ends with p-1), and . Similarly,  ] , where .

For the crosslink operation, the set relation is . Thus partial relation is  (from De Morgan's theorem). If and only if all pairs of possible values satisfy this partial relation, then the set relation is satisfied. This can be written as: relation

Therefore, the partial relation is , and the relation type of crosslink operation is AND type because AND function is used to combining all partial relations.

An example of OR type relation is the one used in the sharp operation, where the relation is. Thus partial relation is, where Ai is the subset of the true set SA. If the set SA includes the possible value i-1, then the set Ai has one element that is the possible value i-1 and is represented by ai=1; otherwise, the set Ai is an empty set and is represented by ai=0. It can be seen that ai is the i-th bit of the bit string that represents the set SA in positional notation. The same thing is with Bi and bi.

*Table 16.3. The partial relation of sharp operation*



Table 16.3 shows how to find the partial relation function for the sharp operation. The first column shows two bits ai and bi. The next two columns show the value of the negated partial relation and the partial relation itself, the last column shows the bitwise function used to determine the partial relation. In the first two rows, ai = 0 means that the set Ai is an empty set, and it is subset of all sets, thus  are 1's (true). In the third row, ai = 1 means that the set Ai includes one element that is possible value i-1, and bi=0 means that the set Bi is an empty set, thus  is 0 (false). In the fourth row, ai = bi = 1 means that the sets Ai and Bi include one same element that is possible value i-1, thus sets Ai and Bi are equal, and  is 1 (true). Therefore, the set relation of sharp operation can be determined by:



where the partial relation function is ai $\overbar{.b\_{i}}$, and the relation type is OR.

**16.4.4. Summary of cube operations in positional notation**

From the above discussion, before, active, after set operations and partial (set) relation can be defined by bitwise functions on bits. Therefore, all cube operations (some of them are basic operations) can be completely specified by 4 bitwise functions and relation type. Table 16.4 summarizes all cube operations described in this chapter in bitwise functions and relation type.

*Table 16.4. Cube Calculus Operations in bitwise function and relation type*



 ai‘ + b i‘ and

The following examples show the entire procedure to carry out cube calculus operations in positional notation.

**Example 16.16.** Variables x1, x2 and x3 all have 3 possible values, thus the sets of possible values are P1 = P2 = P3 = {0,1,2}. A cube of is denoted as  in positional notation; and it is written as [110-010-111] for simplifying.

**Example 16.17.** Assuming cubes A = x1x2 and B = x2x3$\overbar{x4}$, where variables x1, x2, x3 and x4 are binary, the intersection of cubes A and B follows:



 

 

 

 

 

where `.' is a bitwise AND operation.

When two opposite literals are multiplied, if the contradiction is generated from the bitwise operation, then there is no resultant cube, which means that there is no intersection between two operand cubes.

**Example 16.18.** Assuming cubes A = ab and B = a$\overbar{b}$, where a and b are binary variables. Then the intersection of cubes A and B is:

****

where 00 is a contradiction symbol for a binary variable, and the contradiction is denoted by ∈ (see Table 16.2).

**Example 16.19.** Redo the crosslink operation shown in Example 16.8 in positional notation as follows:





where the header of bit is the name of the bit. The subscript of bit name has two parts separated by comma (`,'), the first part represents the index of the variable, and the second part represents the possible value. This notation will also be used in the next chapters. Because:









The variables x1 and x2 are special variables. The two resultant cubes are :





Where ‘+’ is a bitwise OR operation.

**16.5. Questions and Problems for students**