OPKFDD를 이용한 불리안 함수 표현의 최적화

정 미 경[†] · 이 혁[†] · 이 귀 상[††]

요 약

DD(Decision Diagrams)는 불리안 함수의 최적화된 표현을 위한 효율적인 자료구조이다. DD를 이용한 그래프 기반 합성에서 불리안 함수의 최적화 목적은 함수를 DD로 표현하는데 있어서 표현공간을 줄이는 것이다.

본 논문에서는 불리안 함수의 그래프 기반 합성을 위해 OPKFDD(Ordered Pseudo-Kronecker Functional Decision Diagrams)를 사용하고, DD 크기의 기준을 노드 수로하고 있다. OPKFDD는 각 노드마다 각기 다른 확장 방법을 선택할 수 있는 특징 때문에 입력 노드에 대한 확장 방법과 입력 변수 순서의 결정에 의해서 OPKFDD의 크기가 좌우된다. 본 논문에서는 기존의 BDD(Binary Decision Diagram) 자료구조에서 OPKFDD를 효율적으로 유도해내는 방법을 제시하고 이를 최소화하기 위한 알고리즘을 제시한다. OPKFDD의 전체 노드 수에 영향을 미치는 것은 다중 출력함수의 경우 각 부분 함수의 관계를 들 수 있다. 이러한 사항들을 고려한 입력변수 순서 결정 방법을 제안하고 기존의 표현 방법 및 변수 순서 결정 방법과의 실험 결과를 제시한다.

An Optimization of Representation of Boolean Functions Using OPKFDD

Mi-Gyoung Jung[†] · Hyuck Lee[†] · Guee-Sang Lee[††]

ABSTRACT

DD(Decision Diagrams) is an efficient operational data structure for an optimal expression of boolean functions. In a graph-based synthesis using DD, the goal of optimization decreases representation space for boolean functions.

This paper represents boolean functions using OPKFDD(Ordered Pseudo-Kronecker Functional Decision Diagrams) for a graph-based synthesis and is based on the number of nodes as the criterion of DD size. For a property of OPKFDD that is able to select one of different decomposition types for each node, OPKFDD is variable in its size by the decomposition types selection of each node and input variable order. This paper proposes a method for generating OPKFDD efficiently from the current BDD(Binary Decision Diagram) Data structures and an algorithm for minimizing one. In the multiple output functions, the relations of each function affect the number of nodes of OPKFDD. Therefore this paper proposes a method to decide the input variable order considering the above cases. Experimental results of comparing with the current representation methods and the reordering methods for deciding input variable order are shown.

※ 이 논문은 1997년도 학술진흥재단 신진연구인력 연구 장려금 지원에 의하여 연구되었음.

† 정 회 원 : 전남대학교 대학원 전산통계학과
†† 종신회원 : 전남대학교 전산학과 교수

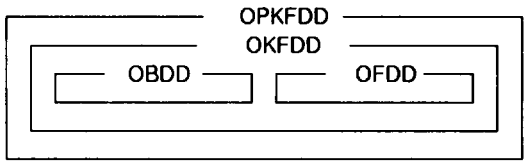
논문접수 : 1998년 12월 15일, 심사완료 : 1999년 1월 29일

1. 서 론

최근에 불리안 함수로 대변되는 논리회로의 표현을 위해 그래프를 이용한 표현 방법인 DD(Decision Diagrams)가 제시되어 논리합성, test 그리고 회로검증(verification)과 같은 여러 가지 설계자동화문제에 활발히 적용되어 왔다. 여러 가지 종류의 DD중에서 실제로가장 많이 사용하고 있는 것은 OBDD(Ordered Binary Decision Diagrams)이다[1,2]. 현재 OBDD의 응용범위는 더욱 넓어져가고 있으며 이는 불리안 함수 외에도 여러 가지 이산함수(discrete function) 즉, 다치함수, 큐브 집합(cube set), 산술함수(arithmetic function) 등의 경우에도 적용되며, 이와 같은 논리함수의 그래프표현은 전산, 전자분야뿐 아니라 다른 많은 분야에서도 매우 유용하게 쓰이고 있다. 또한 장기적으로 이러한 그래프 표현은 그 특성인 정규적(canonic) 표현임과 동시에 사용의 편리함 등으로 인하여 더욱 그 사용이 늘어날 전망이다.

따라서 주어진 불리안 함수를 DD로 표현하는데 있어서 표현 공간을 줄이기 위해서 그래프 기반 합성에 대한 연구가 활발히 이루어지고 있다. DD의 표현 공간을 최소화하기 위해서는 두 가지 관점에서의 연구가 필요하다. 첫째, 입력 변수가 n개 일 때, $n!$가지의 입력 변수 순서 중에서 DD의 크기를 작게 하는 입력 변수 순서를 결정하는 것이다. 둘째, Shannon 확장, Positive Davio 확장 그리고 negative Davio 확장 중에서 각 입력 노드에 대한 확장 방법을 선택하는 것이다.

입력변수 확장 방법에 관한 연구는 다음과 같다. 각 입력 노드의 확장 방법을 정하는 문제는 DD의 종류에 따라 달라진다[3]. DD들의 상관 관계는 (그림 1)과 같다[3]. 현재 가장 많이 사용되고 있는 DD는 OBDD이고, OBDD을 이용하여 불리언 함수를 효율적으로 나타내지 못하는 경우는 XOR-게이트를 기초로 하는 논리합성의 응용에 사용되는 OFDD(Ordered Functional Decision Diagrams)를 사용한다. OKFDD(Ordered Kronecker Functional Decision Diagrams)는 OBDD와 OFDD에서 파생되었고[4], 다단계 XOR-게이트를 기초로 하는 회로를 위한 기술 매핑(technology mapping)을 위해서 사용된다. 또한 각 변수에 대해서 적절한 확장 방법을 선택함으로써 OBDD와 OFDD(Ordered Functional Decision Diagrams)보다 주어진 같은 함수를 훨씬 적은 노드 수를 가지고 표현할 수 있다[5].



(그림 1) DD들간의 관계
(Fig. 1) Relations among decision diagrams

본 논문에서 사용하는 OPKFDD는 BDD, OFDD 그리고 OKFDD[6,7]를 일반화시킨 DD의 한 종류이다. 기존의 다른 DD와는 다르게 각 노드에 대해서 Shannon 확장, Positive Davio 확장 그리고 Negative Davio 확장 중에서 한가지를 선택하는 DD이며, 기존의 BDD 패키지에서 비교적 쉽게 OPKFDD를 생성할 수 있다. OPKFDD의 경우 n개의 입력변수에 대해서 $3^{2^{n-1}}$개의 서로 다른 확장을 할 수 있다. 따라서 본 논문에서는 OPKFDD를 이용하여 불리안 함수를 표현하고, 비용 함수를 제안하여 최소의 노드를 갖는 확장 방법을 선택한다.

입력 변수 순서를 결정하는 문제는 NP-complete임이 증명되어 있으며 따라서 이를 해결하기 위한 방법들은 여러 가지 휴리스틱(heuristic)에 의존하고 있는 실정이다[8,9,10,11]. 많은 휴리스틱 알고리즘들은 큰 벤치마크회로에 대해서는 효율적인 입력변수 순서(ordering)를 제공하지 못하는 단점을 가지고 있다. 현재는 휴리스틱 알고리즘을 더욱 개선한 방법들이 제안되고 있다.

변수순서를 결정하는 방법은 DD를 생성하기 전에 그 순서를 정하는 부분과 이미 생성된 DD에서 연산과정과 함께 이의 변수순서를 적절히 재조정하는 dynamic reordering[12,13]의 두 부분으로 구분하여 생각할 수 있다. 이 문제는 매우 중요한 연구테마로 인식되고 있으며, 이에 대하여 매우 활발한 연구가 진행되고 있다. 현재까지 이 문제에 대하여 여러 가지 해법들이 제시되었으나 앞으로 발전의 여지가 많이 남아있다고 보여진다. 입력 변수 순서 결정하는 문제에 대한 가장

최근의 연구 중 sifting[12]이라 하는 동적 재배치 방법이 가장 큰 주목을 받는 부분이다. 따라서 본 논문은 sifting 방법을 기반으로 하여 다중 출력 함수의 경우 각 부분 함수의 관계를 고려한 입력변수의 순서를 결정하는 방법을 제시한다. 또한 이 논문에서 제안한 방법과 CUDD(Colorado University Decision Diagram) Package[14]에서 제공하는 reordering 방법을 비교한 실험결과를 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 OPKFDD의 정의와 dynamic reordering 방법 중에 하나인 sifting 방법을 소개하고, 3장에서는 OPKFDD에서의 확장 방법과 입력변수 순서 결정 방법을 제안한다. 그리고 4장에서는 3장에서 제안한 알고리즘의 실험결과를 제시하며, 마지막으로 결론과 향후 연구 방향을 언급한다.

2. 관련연구

2.1 OPKFDDs(Ordered Pseudo-Kronecker Functional Decision Diagrams)

DD, OFDD, OKFDD 그리고 OPKFDD의 정의는 다음과 같다[3]. OFDD, OKFDD 그리고 OPKFDD는 확장된 DD의 개념을 갖는다.

[정의 1] 입력변수 $X_n := \{x_1, x_2, \ldots, x_n\}$에 대한 DD는 루트가 있는 방향성 비 순환 그래프이다. 이 그래프는 2가지 타입의 노드를 갖는데 비-단말(non-terminal)과 단말(terminal) 노드로 구성된다. 비-단말 노드 v는 입력 변수 x_n으로 분류되고, low(v)와 high(v) 두 개의 자 노드를 갖는다. 그리고 단말 노드 v는 0 또는 1의 값을 갖고 자 노드가 없다.

[정의 2] DD의 각 패스에서 각 입력 변수가 최소한 한번 있을 때 DD를 free라고 한다. 그리고 DD가 free이고 각 패스에서 입력변수의 순서가 동일하다면 DD를 ordered하다고 한다.

DD는 아래와 같은 확장 타입을 사용하여 불리안 함수를 표현한다.

$$\overline{x_i} f_{x_i=0} + x_i f_{x_i=1} \quad : \quad d_i = S$$
(*Shonnan Decomposition*)

$$f_{x_i=0} \oplus x_i \cdot g \quad : \quad d_i = pD$$
(*Positive Davio Decompositin*)

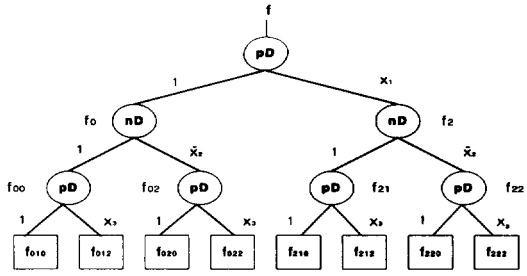
$$f_{x_i=1} \oplus \overline{x_i} \cdot g \quad : \quad d_i = nD$$
(*Negative Davio Decomposition*)

$$g = f_{x_i=0} \oplus f_{x_i=1}$$

여기서 $f_{x_i=0}$와 $f_{x_i=1}$는 각각 $x_i=0$와 $x_i=1$에 대한 f의 공통인자(cofactor)이고, +는 OR 연산자이고, ⊕은 Exclusive OR연산자이다.

[정의 3] OFDD는 입력변수 x_n에 대한 ordered DD이고 DD에서 같은 레벨에 있는 입력변수는 확장 타입 중 *pD*와 *nD*로 구성된다.

예를 들면, 입력 변수가 n개이고 각 입력변수가 확장된 경우 (그림 2)와 같고 OFDD는 각 노드가 *pD*와 *nD*중 한 가지로 확장되기 때문에 2^n개의 서로 다른 확장 방법이 있을 수 있다.

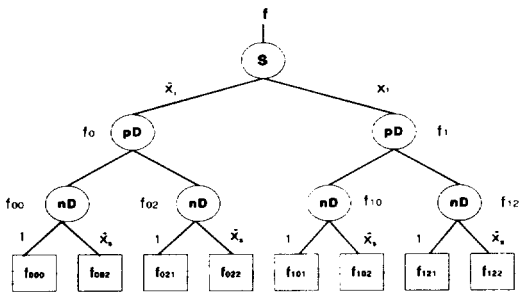


(그림 2) n변수 함수에 대한 OFDD의 확장 방법
(Fig. 2) The choice Decomposition of OFDD for n-variable function

[정의 4] OKFDD는 입력변수 x_n에 대한 ordered DD이고 DD에서 같은 레벨에 있는 입력변수는 *S*, *pD* 그리고 *nD* 중에서 한가지의 확장 타입으로 구성된다.

예를 들면, 입력 변수가 n개이고 각 입력변수가 확장된 경우 (그림 3)와 같고 OKFDD는 각 노드가 *S*,

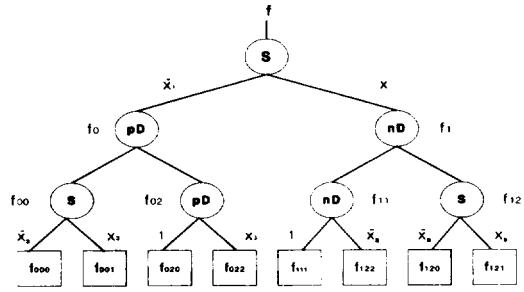
pD 그리고 *nD* 중에서 한가지로 확장되기 때문에 3^n개의 서로 다른 확장 방법이 있을 수 있다.



(그림 3) n변수 함수에 대한 OKFDD의 확장방법 선택
(Fig. 3) The choice Decomposition of OKFDD for n-variable function

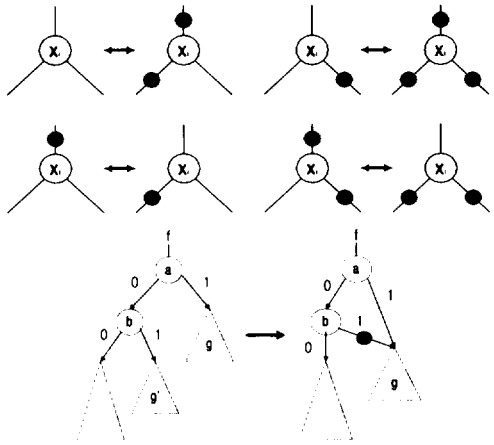
[정의 5] OPKFDD는 입력변수 x_n에 대한 ordered DD이고 각 노드마다 각기 다른 확장 타입으로 구성된다.

예를 들면, OPKFDD가 입력변수가 n개인 경우 (그림 4)와 같이 각 노드는 서로 다른 확장 방법을 선택할 수 있으므로 OPKFDD는 3^{2^n-1}개의 서로 다른 확장 방법이 있을 수 있다.



(그림 4) n변수 함수에 대한 OPKFDD의 확장방법 선택
(Fig. 4) The choice Decomposition of OPKFDD for n-variable function

OPKFDD의 표현은 보수간선(Complemented Edges (CEs))을 사용함으로써 그 크기를 감소시킬 수 있다. (그림 5)와 같이 만약 함수 f의 OPKFDD가 부 그래프 g와 g′를 갖는다면, g와 g′의 보수관계를 이용하여 DD에서 두 개의 부 그래프를 유지하지 않고 보수간선을 이용하여 하나의 부 그래프만을 유지함으로써 DD의 전체 크기를 줄일 수 있다[2,3].



(그림 5) 보수간선의 항등쌍과 보수간선 사용 예제
(Fig. 5) Complement edges : equivalent pairs and example

2.2 Dynamic Reordering

본 논문에서는 dynamic reordering 방법들[12,13] 중에서 sifting 방법을 응용하여 변수 입력 순서를 결정하는 알고리즘을 제시한다.

2.2.1 Sifting 알고리즘[12]

이 알고리즘은 다른 변수들의 위치가 고정되어 있다는 가정 하에 변수의 최적 위치를 찾는 방법이다. DD에 n개의 변수가 있다면 한 변수에 대해 n개의 잠정적인 위치(현재 자신의 위치도 포함)가 존재할 수 있다. n개의 위치 중에서 DD의 크기를 가장 적게 하는 위치를 찾아내는 것이 sifting 알고리즘의 목적이다.

알고리즘 수행 과정을 보면 변수들은 DD의 각 레벨에서 노드의 수에 따라 내림차순으로 정렬한다. 각 변수들은 다른 모든 변수들이 고정되어 있다는 가정 하에 국부적으로 최적의 위치에 이동된다. 이와 같은 과정에서 각 변수들의 위치가 한번 고정되면 변하지 않는다. 이 방법은 $O(n^2)$시간 정도의 교환 작업이 필요하다. DD의 크기가 원래의 두 배 이상으로 커지면 그쪽 방향으로의 탐색 작업은 끝난다. 다음 (그림 6)은 변수 x_4의 최적의 위치를 찾아가는 sifting 알고리즘의 예를 설명하고 있다. 각 변수들의 위치는 고정되었다는 가정 하에 변수 x_4을 가능한 모든 위치에 교환하면

서 최적의 위치를 찾아간다. 본 논문에서는 sifting 알고리즘을 적용하여 그룹 내에 있는 변수들의 국부적 변수들의 순서를 결정하였다. 이에 관한 자세한 내용은 3절에서 다루도록 하겠다.

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	*initial*
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	$swap(x_4, x_5)$
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	$swap(x_4, x_6)$
$x_1, x_2, x_3, x_5, x_6, x_7, x_4$	$swap(x_4, x_7)$
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	$swap(x_7, x_4)$
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	$swap(x_6, x_4)$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	$swap(x_5, x_4)$
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	$swap(x_3, x_4)$
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	$swap(x_2, x_4)$
$x_4, x_1, x_2, x_3, x_5, x_6, x_7$	$swap(x_1, x_4)$
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	$swap(x_4, x_1)$
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	$swap(x_4, x_2)$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	$swap(x_4, x_3)$
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	$swap(x_4, x_5)$
$x_1, x_3, x_3, x_5, x_6, x_4, x_7$	$swap(x_4, x_6)$
$x_1, x_2, x_3, x_5, x_6, x_7, x_4$	$swap(x_4, x_7)$

(그림 6) Sifting 알고리즘의 예
(Fig. 6) Example of sifting algorithm

3. OPKFDD 생성 방법과 입력변수 순서 결정 방법

3.1 OPKFDD의 생성 방법

OPKFDD는 DD의 특성상 각기 다른 확장 방법을 매 노드마다 결정하여 사용한다. 본 논문에서는 이러한 확장 방법의 결정을 위해 노드의 수를 기준으로 이용하고 있다. DD의 표현에 있어서 표현공간을 줄이는 것은 큰 이슈가 되고 있는 문제이다. 따라서 본 논문에서는 기본적으로 하위 노드의 개수를 줄이는 방향으로 DD를 구성해 나간다. OPKFDD의 생성 방법은 불리안 함수 표현으로 초기에 생성된 BDD의 각 노드마다 XOR-간선을 생성하여 ELSE-간선, THEN-간선 그리고 XOR-간선 중에서 최소의 노드 수를 갖는 2개의 간선을 선택한다. 단 매 노드마다 3가지 하위노드의 비용함수를 계산하는 것은 연산 시간을 증가시키므로 몇 가지 경우에 대해서는 고정된 확장 방법을 적용한다. 각 노드의 자세한 확장 방법 결정은 다음의 2단계 과정을 갖는다.

- 1 단계 : 각 노드에 대해서 XOR-간선이 생성되는 경우를 판단한다.
- 2 단계 : 최소의 노드 수를 갖는 두개의 간선을 선택하여 확장 방법을 선택하여 OPKFDD을 생성한다.

1) 각 노드에 대해서 XOR-간선이 생성되는 경우

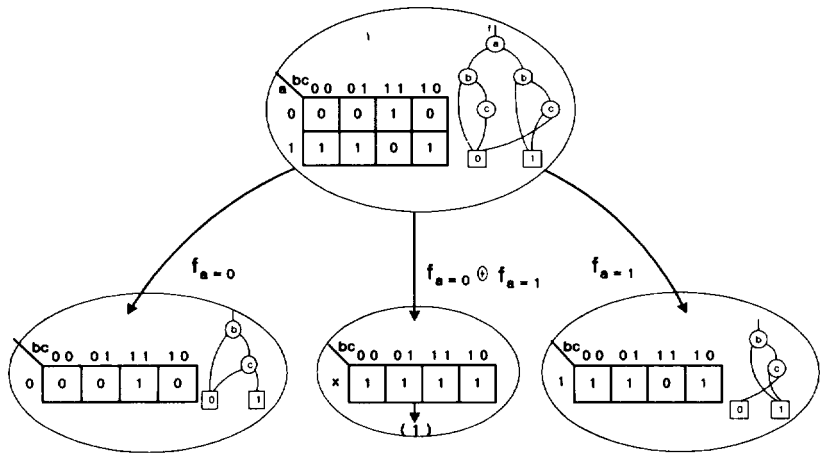
BDD에서 각 노드의 ELSE-간선 ($f_{x_{i=0}}$)과 THEN-간선 ($f_{x_{i=1}}$)을 XOR ($f_{x_{i=0}} \oplus f_{x_{i=1}}$)시킨 간선을 갖도록 수정한다. 여기서 각 노드마다 XOR-간선을 다 생성하지 않고 아래의 두 가지 조건을 만족하는 경우만 생성한다.

- 조건 1 : XOR-간선이 단말 노드인 경우
- 조건 2 : n +2 단계의 부 함수까지 공유정도가 2이상인 경우

첫째, XOR-간선이 단말 노드인 경우라는 의미는 상수 값을 갖는다는 것이다. 즉, ELSE-간선과 Then-간선의 XOR 연산의 결과가 0 혹은 1값을 갖는다는 것이다. 0 값을 갖는다는 것은 ELSE-간선의 cofactor와 THEN-간선의 cofactor가 같은 경우를 의미하고, 1값을 갖는다는 것은 ELSE-간선의 cofactor와 THEN-간선의 cofactor가 서로 보수(complement)관계에 있다는 것을 의미한다. 이 경우는 shannon 확장보다는 negative davio나 negative davio로 확장하는 것이 노드 수를 줄일 수 있다. (그림 7)과 같이 함수 f를 변수 a에 대해서 확장할 경우 $f_{a=0}$과 $f_{a=1}$을 XOR하면 함수 값이 1이 된다. 이것은 XOR 간선이 입력변수 b와 c의 노드 확장 없이 단말 노드 1로 되기 때문이다. sub 함수 $f_{a=0} \cdot_{a=1}$의 노드 수는 0이 된다. 즉, 이 경우의 ELSE-간선과 THEN-간선보다 노드 수가 작기 때문에 XOR-간선은 생성된다.

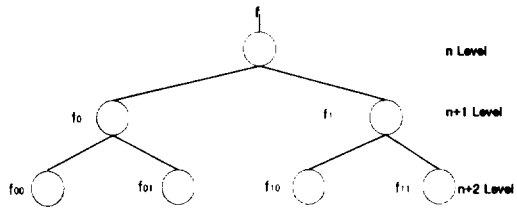
둘째, n +2 단계의 부 함수까지 공유정도를 계산하여 공유정도가 2개 이상인 경우는 n 단계 노드의 XOR-간선을 생성한다.

(그림 8)과 같이 $f_{00} = f_{10}$, $f_{00} = f_{11}$, $f_{01} = f_{10}$ 그리고 $f_{01} = f_{11}$를 계산하여 부 함수간의 공유정도를 파악한다. 여기서 같은 수가 0 혹은 1이면 공유정도가 적기 때문에 f_0와 f_1을 XOR 연산을 했을 때 나오는 결과가 노드 수면에서 크거나 같다. 즉, f_0와 f_1의 서브 노드들이 같지 않다는 경우는 연산을 했을 때 노드



(그림 7) 조건 1에 의해서 XOR-간선이 생성되는 경우
(Fig. 7) The case of generation XOR-edge by case 1

가 줄어드는 경우는 없기 때문에 서브 노드의 개수가 줄어들 확률이 적다. 그래서 부 노드의 공유가 2개 이하인 경우는 XOR-간선이 확장 방법에서 선택되는 경우가 적기 때문에 매 노드마다 3가지 하위노드의 비용함수를 계산하는 것은 연산 시간을 증가시키므로 이 경우에 XOR-간선을 확장하지 않고, 고정된 확장 방법을 적용한다.



(그림 8) 부 함수간의 공유 정도
(Fig. 8) The degree of share between subfunctions

2) 각 노드의 확장 타입 선택 방법

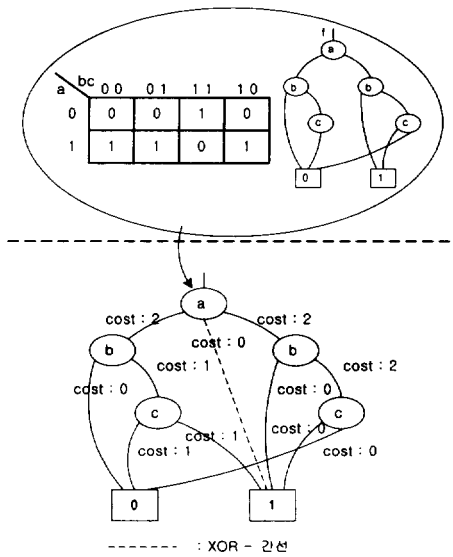
위의 조건에 의해서 XOR-간선을 갖는 수정된 BDD로부터 OPKFDD를 생성하기 위해서 (그림 9)의 비용함수를 이용하여 각 간선의 부 노드 수를 계산한다. 이 알고리즘은 각 노드의 각 노드의 ELSE-간선과 THEN-간선의 부 노드 수를 계산하고 XOR-간선의 부 노드 수는 XOR-간선이 확장된 경우에만 계산하는 비용 함수이다. 여기서 계산되는 노드 중 단말 노드는 제외된다. 그래서 계산된 각 간선의 부 노드의 수를 근거로 하여 3가지 간선 중에서 두 개를 선택하여 노드의 확장 타입을 결정하도록 한다.

```
/* OPKFDD의 각 간선에서 생성되는 노드 수 계산하는 함수 */
OPKFDD_GetCost(func)
{
  if (func의 비용이 이미 계산되어있으면) return func의 비용;
  lcost = OPKFDD_GetCost(OPKFDD_E(func));
                    /* ELSE-간선의 노드 수 계산 */
  rcost = OPKFDD_GetCost(OPKFDD_T(func));
                    /* THEN-간선의 노드 수 계산 */
  if (isCalcXor(func))  /* XOR-간선이 확장되는 경우를 판단*/
  {
     xcost = OPKFDD_GetCost(OPKFDD_X(func));
                    /* XOR-간선의 노드 수 계산 */
     ans = min2Sum(lcost,rcost,xcost);
   }
   else ans = lcost + rcost;
   return ans;
}
```

(그림 9) 노드 수를 계산하는 비용함수
(Fig. 9) The cost function of computing number of nodes

(그림 9)의 비용함수를 적용한 예제는 (그림 10)과 같다. 입력변수 a는 조건 1에 의해서 sub 함수 $f_{a=0} \bullet_{a=1}$의 함수 값이 1로 단말 노드 1을 갖기 때문에 XOR-간선을 생성하고, 입력변수 b, c는 조건 2에 의해서 XOR-간선을 생성하지 않고 고정된 확장 방법을 선택한다. 즉, b와 c는 shannon 확장이 된다. 그리고 매 노드마다 하위 노드 수를 계산하여 각 간선의 비용을 구한다.

(그림 10)의 예제에서 계산된 결과를 이용해서 다음 (그림 11)의 알고리즘을 적용한다. 이 알고리즘은 각 노드의 ELSE-간선, THEN-간선 그리고 XOR-간선 중에서 최소의 노드 수를 갖는 2개의 확장 방법을 선택한다. 즉, 노드가 만약 최소의 노드 수를 갖는 ELSE-간선과 THEN-간선이 선택되면 그 노드는 shannon으로 확장되고, ELSE-간선과 XOR-간선이 선택되면 negative davio로 확장된다. 그리고 THEN-간선과 XOR-간선 선택되면 positive davio로 확장된다. 그래서 결국 각 노드마다 최소의 노드 수를 갖는 확장 방법을 선택하여 OPKFDD를 생성한다.



(그림 10) 노드의 비용
(Fig. 10) node cost

결국 노드는 (그림 11)의 알고리즘을 적용하면 (그림 12)과 같이 *s*, *pD*, 그리고 *nD* 확장 방법중의 하나를 선택하여 OPKFDD를 구성한다. (그림 12)에서 입력변수 a는 비용이 적은 ELSE-간선과 XOR-간선을 선택하여 negative davio 확장하고 입력변수 b, c는 shannon 확장을 한다.

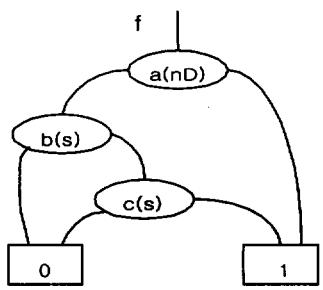
```
OPKFDD_Decomp_Choice(func) /* Decomposition 방법 선택 */
{ if(func가 터미널 노드) {
    근 노드로부터 노드 func까지의 path 생성;
    return;
    }
    lcost = OPKFDD_GetCost(OPKFDD_E(func));
    rcost = OPKFDD_GetCost(OPKFDD_T(func));
    xcost = OPKFDD_GetCost(OPKFDD_X(func));

 /* lcost, rcost, xcost 중에서  최소의 노드 수를
                    갖는 두 개의 간선 선택 */
   if( lcost과  rcost 선택)  {
                    /* shannon 확장 사용 */
    synLevel(OPKFDD_E(func));
    synLevel(OPKFDD_T(func));
    }

   if( rcost과 xrcost 선택) {
                    /* positive davio 확장 사용 */
    synLevel(OPKFDD_T(func));
    synLevel(OPKFDD_X(func));
    }

   if( lcost과  xcost 선택) {
                    /* negative davio 확장 사용 */
    synLevel(OPKFDD_E(func));
    synLevel(OPKFDD_X(func));
    }
   return;
 }
```

(그림 11) 확장 방법 선택 알고리즘
(Fig. 11) Algorithm of the decomposition type choice



(그림 12) 노드의 확장 방법
(Fig. 12) node decomposition

3.2 다 출력 함수의 입력변수 순서 결정하는 방법

다 출력 함수의 입력변수 순서 결정은 단일 함수들의 각각을 가지고 결정하기보다는 가능한 함수간의 포함 관계를 고려하여 입력변수 순서를 결정하는 것이 DD의 크기를 최소화할 수 있다.

논 논문에서 제시하는 다 출력 함수의 입력변수 결정방법에 대한 알고리즘은 (그림 13)과 같다. 이 알고리즘은 다 출력 함수의 입력변수 순서를 결정하기 위

해서 함수 각각을 개별적으로 취급하는 것이 아니라 함수의 포함 관계를 고려하여 입력 변수들을 그룹으로 나눈다 그리고 그룹의 종속관계에 따라 그룹의 순서를 결정한 다음 다시 그룹 내에 있는 변수들을 국부 순서를 결정하기 위해 수정된 sifting 방법을 적용하여 최종적인 입력변수 순서를 결정한다.

```
/* 각 함수의 포함관계를 고려하여 변수들의 그룹과
        그룹내의 변수 순서를 결정 */
Group_Shift(func)
{
  for(hFOrderCount) {
        /*함수의 순서를 이용하여 그룹내의 변수결정 */
   for(j=0; j<gInputs; j++) {
       if(hFOrder[i] & (1 << j)) {
          gOrder[ind++] = j;
        }
    }
   gGroup[gGroupCount++] = ind;
   switch(gOrdering) {
        /* 그룹내의 변수들의 국부적 순서 결정 */
       case OPKFDD_SUPPORT_RANDOM:
       SUPPORT_RandomGen(gOrder,last,ind);
       break;
       case OPKFDD_SUPPORT_SIFT:
       default:            /* OPKFDD_SUPPORT */
       SUPPORT_Sort(gOrder,hLiteral,last,ind);
       break;
    }
  }
}
```

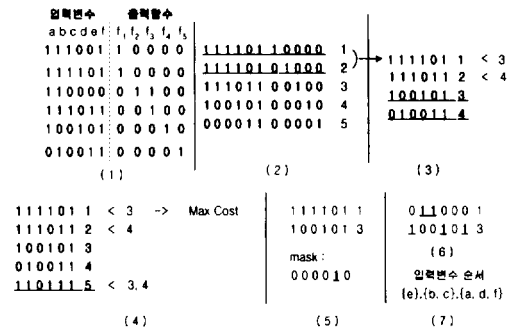
(그림 13) 다 출력 함수의 입력변수 결정 알고리즘
(Fig. 13) Algorithm of variable ordering for multi-function

(그림 13)의 알고리즘을 이용한 입력변수 순서 결정은 (예제)와 같다.

(예제) 함수 F는 $f1$, $f2$, $f3$, $f4$ 그리고 $f5$로 구성되고, 입력변수는 a, b, c, d, e, f이고, 각각의 함수는 다음과 같다.

$$f1 = a\,b\,c\,d'e'\,f + a\,b\,c\,d\,e'\,f$$
$$f2 = a\,b\,c\,d\,e'\,f + a\,b\,c'\,d'e'\,f'$$
$$f3 = a\,b\,c'\,d'e'\,f' + a\,b\,c\,d'e\,f$$
$$f4 = a\,b'c'\,d\,e'\,f$$
$$f5 = a'\,b\,c'\,d'e\,f$$

위의 각 함수를 (그림 14)과 같이 입력변수 순서를 결정한다.



(그림 14) 입력변수 순서 결정 방법
(Fig. 14) The method of decision input variable order

(그림 14)에서 (1)는 (예제)의 함수 $f1$, $f2$, $f3$, $f4$ 그리고 $f5$를 PLA(Programmable Logic Array) 형태로 나타낸 것이고, (2)는 각 함수의 변수 집합을 나타낸다. (3)는 (2)에서 동일한 변수 집합이 있는 함수 1과 2를 하나로 합한다. 그리고 각 함수가 포함하고 있는 변수 집합을 가지고 함수간의 종속성을 구한다. 여기에서 함수 1의 변수 집합과 함수 3의 변수 집합을 비교해보면 함수 1은 함수 3을 포함하게 된다. (4)는 종속성이 없는 함수 3과 4를 조합하여 새로운 함수 5와 변수 집합 110111을 추가한다. 그리고 각 변수 집합의 비용을 구한다. 여기서 비용은 bit수 + 포함된 product수 + max (종속함수 비용)로 계산된다. 이 수식을 적용해 보면 함수 1이 가장 큰비용을 갖게 된다. (5)는 max 비용에 따라 함수의 순서를 재 정의하고, 함수 1로부터 그룹의 mask를 구한다. 즉 입력변수 e가 그룹 1이 된다. (6)는 (5)의 함수 순서와 중복되지 않는 입력변수 집합이다. 이것을 기준으로 하여 함수 1에서 입력변수 b와 c가 그룹 2가 되고, 함수 3에서 입력변수 a, d, 그리고 f가 그룹 3이 된다. 그래서 결국 (7)과 같이 각 그룹의 order는 {e}, {b, c} 그리고 {a, d, f}가 되고 각 그룹 안에 있는 변수 집합은 dynamic reordering 방법 중 sifting 방법을 적용하여 최종적인 입력변수 순서를 결정한다.

4. 구현 및 실험결과

본 논문에서 제시한 방법은 C 언어로 구현했고, PC 팬티엄에서 실행했다. 실험 결과는 주어진 벤치마크

(benchmark) 회로에 대하여 본 논문에서 제안한 방법과 CUDD에서 제공하는 reordering 방법을 노드 수로 비교하였다. 그리고 실행 시간은 user time으로 하고 단위는 초(second)이다.

<표 1>의 결과는 각 노드의 확장 방법에 따른 BDD OKFDD 그리고 OPKFDD의 노드 수를 비교한 것이고, <표 2>는 CUDD에서 제공하는 reordering 방법과 본 논문에서 제안한 방법을 비교하여 제시한다. 각각의 열은 다음을 나타낸다.

<표 1>에서 알 수 있는 것은 확장 방법에 따라서 DD의 크기가 다름을 알 수 있다. OPKFDD는 OBDD나 OKFDD와 비교해서 노드 수가 작음을 알 수 있다. 특히 OBDD와 비교해서 50% 이상 노드 수가 감소함을 알 수 있다. 즉, 각 노드의 확장 방법을 잘 선택함으로써 DD의 크기를 최소화할 수 있다. 그리고 실행 시간은 OBDD나 OKFDD와 비교해서 1.6배 빠르다.

〈표 1〉 DD 종류에 따른 노드 수 비교
〈Table 1〉 comparison of the number of nodes for a class of DD

Benchmark 회로	OBDD		OKFDD		[illegible] DD	
	#node	time	#node	time	[illegible]	[illegible]
5xp1	142	0.33	142	0.27	[illegible]	0.27
card4	151	0.30	151	0.29	[illegible]	0.26
clip	671	11.14	570	11.18	[illegible]	5.52
clog8	632	11.15	642	12.56	[illegible]	6.54
cmlp4	434	6.10	414	4.53	[illegible]	3.67
cnrm	403	2.54	405	2.25	[illegible]	1.84
cu	574	3.41	100	1.86	[illegible]	1.47
f5lm	151	0.37	141	0.34	[illegible]	0.33
inc	191	1.46	172	1.55	[illegible]	1.39
mlp3	87	0.31	75	0.27	[illegible]	0.25
rd53	66	0.16	66	0.17	[illegible]	[illegible]
rd73	289	0.34	137	0.29	[illegible]	[illegible]
sao2	236	1.41	341	1.63	[illegible]	[illegible]
t481	1680	0.75	32	0.52	[illegible]	0.41
total	4027	39.77	3356	37.71	[illegible]	23.51

〈표 2〉 reordering 방법에 따른 노드 수 비교
〈Table 2〉 comparison of the number of nodes for reordering methods

Benchmark 회로	#2		#4		#6		#8		#14		A		B	
	#node	time	#node	time	#node	time	#node	time	#node	time	#node	time	#node	time
5xp1	147	10.38	88	5.37	88	5.76	137	2.58	100	6.59	137	1.93	[illegible]	0.27
card4	77	9.14	88	5.36	88	5.86	91	1.26	82	6.19	115	0.63	[illegible]	0.26
clip	279	11.31	301	6.74	301	7.44	297	3.98	307	8.47	409	8.34	[illegible]	5.52
clog8	408	15.07	436	11.49	436	12.13	367	5.06	445	13.83	335	2.56	[illegible]	6.54
cmlp4	274	14.93	283	12.59	283	12.98	249	5.98	275	14.88	260	8.58	[illegible]	[illegible]
cnrm	403	12.04	448	10.88	448	11.54	420	5.35	435	11.92	372	4.17	[illegible]	[illegible]
cu	98	9.33	96	5.15	96	5.65	106	1.39	102	6.43	106	0.54	[illegible]	[illegible]
f5lm	79	9.37	78	5.14	78	5.97	78	1.26	79	6.18	78	0.30	[illegible]	[illegible]
inc	163	9.98	172	6.71	172	6.98	123	1.53	172	7.42	123	0.62	[illegible]	[illegible]
mlp3	61	8.76	58	5.07	58	5.83	55	1.19	56	6.11	57	0.23	[illegible]	[illegible]
rd53	31	8.75	31	4.93	31	5.73	31	1.08	31	6.65	31	0.20	[illegible]	[illegible]
rd73	101	8.83	101	5.27	101	5.88	101	1.22	101	6.52	101	0.19	[illegible]	[illegible]
sao2	327	10.23	307	6.15	307	6.78	312	2.67	301	7.71	332	2.40	[illegible]	[illegible]
t481	39	10.51	39	6.44	39	7.04	39	2.07	39	8.15	39	0.38	[illegible]	[illegible]
total	2448	148.63	2487	97.29	2487	105.57	2367	36.62	2486	117.05	2456	31.07	[illegible]	[illegible]

A : OPKFDD로 노드 확장
Ordering 방법, 함수간의 포함관계 그리고 중복 term 제거가 고려되지 않았음.
B : 함수간의 포함관계를 이용하여 그룹 shifting 적용, 중복 term을 제거한 결과.

- CUDD의 Reordering Method[14]

#2 : CUDD_REORDER_RANDOM
#4 : CUDD_REORDER_SIFT
#6 : CUDD_REORDER_SYMM_SIFT
#8 : CUDD_REORDER_WINDOW2
#14 : CUDD_REORDER_그룹_SIFT

<표 2>의 실험결과를 보면, CUDD에서 제공하는 reordering 방법과 비교해서 본 논문에서 제안하고 있는 변수 순서 결정 방법이 clip와 clog8에서 다소 노드 수가 많으나 대체적으로 다른 벤치마크 회로에 대해서는 노드 수가 8% 감소함을 알 수 있다. 그리고 실행시간을 B와 비교해 보면, #2 방법보다 6.3배 빠르고 #8 방법보다 1.5배 빠르다.

5. 결론 및 연구방향

본 논문에서는 OPKFDD을 이용한 불리안 함수의 최적화 표현을 위해서 각 노드의 확장 방법과 입력변수 순서를 결정하는 방법을 제안하였다. 각 노드의 확장 방법은 BDD의 ELSE 간선과 THEN 간선을 exclusive-OR 시킨 XOR 간선을 추가한 후에 최소의 노드 수를 갖는 두 개의 간선만을 선택함으로써 결국에는 최소의 크기를 갖는 OPKFDD로 불리안 함수를 표현하여 최적화 한다. 또한 입력변수 결정 방법은 다출력 함수의 경우 함수 각각을 개별적으로 취급하는 것이 아니라 함수의 포함 관계를 고려하여 입력 변수들을 그룹으로 나누고, 그룹의 종속관계에 따라 그룹의 순서를 결정한 다음 다시 그룹 내에 있는 변수들을 local order를 결정하기 위해 sifting 방법을 적용하여 최종적인 입력변수 순서를 결정하는 알고리즘을 제시했고, 이것을 CUDD 패키지에 적용하여 보았다.

본 논문에서 제시한 방법은 <표 1>과 같이 각 노드의 확장방법을 다른 DD와 다르게 각기 다른 확장 방법을 선택하는 OPKFDD로 불리안 함수를 표현함으로써 OBDD보다 50% 이상 노드 수가 줄어들었음을 알 수 있다. 또한 기존의 CUDD에서 제공한 reordering 방법과 비교해 볼 때 <표 2>와 같이 개선된 결과를 보이고, 실행시간을 줄인다.

향후 연구 방향으로는 불리안 함수의 최적화 표현을 위해서 노드 수를 감소시킬 수 있도록 입력 변수의 순서를 찾는 알고리즘에 대한 연구와 본 논문에서 제안한 방법이 최적의 해에 얼마나 근사한지 평가, 보완하는 연구가 이루어져야한다.

참 고 문 헌

[1] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trean. on Computer, pp.677-691, 1986.

[2] K. S. Brace, R. C. Rudell, and R. E. Bryant, Efficient Implementation of a BDD Package, In Proceedings 27th Design Automation Conf., pp. 40-46, 1990.

[3] T. Sasao and M. Fujita, Representations of Discrete Functions, Kluwer Academic Publishers, 1996.

[4] R. Drechsler, M. Theobald, and B. Becker, Fast OFDD based minimization of Fixed Polarity Reed-Muller expressions, In European design Automation Conf., pp.2-7, 1994.

[5] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams, In Proceedings 31th Design Automation Conf., pp.415-419, 1994.

[6] R. Drechsler and B. Becker, Dynamic minimization of OKFDDs, In Int'l Conf. on Comp. Design, pp.602-607, 1995.

[7] R. Drechsler, B. Becker and N. Gockel, Minimization of OKFDDs by genetic algoritnm, In International Symposium on Soft Computing, Reading, pp.B : 271-277, 1996.

[8] T. Sasao and J. T. Butler, A design method for look-up table type FPGA by pseudo-kronecker expansion, In Proceedings International Symposium on Multiple-value Logic, pp.97-106, May 1994.

[9] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagram. In Proceedings International Conference on Computer-Aided Design, pp.2-5, November, 1988.

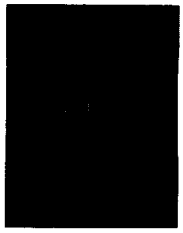
[10] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni Vincentelli, Logic Verification Using Binary Decision Diagrams in Logic Synthesis Environment, In Proceedings International Conference on Computer-Aided Design, pp.6-9, November 1988.

[11] S. Minato, N. Ishiura, and S. Yajima, Shared Binary Decision Diagram with attributed Edges for Efficient Boolean Function Manipulation, In Proceedings 27th Design Automation Conf., pp.52-57, June 1990.

[12] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, In IEEE International Conference on Computer-Aided Design, pp.42-47, 1993.

[13] M. Fujita, Y. Masunga, and T. Kakuda, On variable ordering of binary decision diagrams for application of multi-level synthesis, In European design Automation Conf., pp.50-54, 1991.

[14] *ftp://vlsi.colorado.edu/pub/cudd-2.2.0.tar.gz*



정 미 경

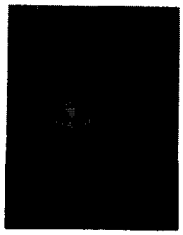
e-mail : mgjung@chonnam.chonnam.ac.kr

1987년 전남대학교 전산통계학과 졸업(학사)

1989년 전남대학교 대학원 전산통계학과 석사

1995년~현재 전남대학교 대학원 전산통계학과 박사수료

관심분야 : VLSI/CAD, 논리합성, 인공지능



이 혁

e-mail : hergilee@mail.posdata.co.kr

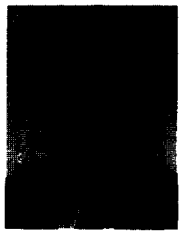
1994년 전남대학교 금속공학과 졸업(학사)

1996년 전남대학교 대학원 전산통계학과 석사

1996년~현재 전남대학교 대학원 전산통계학과 박사과정

1996년~현재 포스데이타 광양 SM부 재직

관심분야 : VLSI/CAD, 인공지능, 멀티미디어 시스템



이 귀 상

e-mail : gslee@chonnam.chonnam.ac.kr

1980년 서울대 공대 전기공학과 졸업(학사)

1982년 서울대 대학원 전자계산기공학과 석사

1982년 금성통신 연구소 근무

1991년 Pennsyivania 주립대학 박사

1984년~현재 전남대 전산학과 부교수

관심분야 : VLSI/CAD, 멀티미디어 시스템, 테스팅, 논리합성