論文 98-23-8-26

결정 다이어그램의 최적화를 위한 탐색공간 축소 기법

비회원 송 문 배*, 동 균 탁*, 장 훈**

Search Space Pruning Technique for Optimization of Decision Diagrams

Moon-Bae Song*, Gyun-Tak Dong*, Hoon Chang** *Regular Members*

※본 연구는 서울대학교 반도체 공동 연구소의 교육부 반도체분야 학술연구 조성비(과제번호 : ISRC96-E-2022)에 의해 수행되었습니다.

요 약

BDD의 최적화 문제는 논리합성과 형식검증 영역에서 필수적인 것으로 인식되고 있다. 변수 순서화 문제는 BDD의 크기와 형태에 직접적인 영향을 미치므로, 적절한 변수 순서를 구하는 문제는 매우 중요한 문제이다. 본 논문에서는 점진적 시프팅이라 부르는 새로운 변수 순서화 알고리듬을 소개한다. 제안된 알고리듬은 기존의 시프팅 알고리듬에서의 탐색공간을 절반이하로 줄이며, 성능의 저하없이 계산시간을 크게 감소시킬 수 있다. 더욱이 점진적 시프팅 알고리듬은 시프팅 알고리듬을 비롯한 다른 변수 순서화 알고리듬에 비해 매우 단순하다. 제안된 알고리듬은 많은 벤치마크 회로를 이용한 실험에서 그 효율성이 입증되었다.

ABSTRACT

The optimization problem of BDDs plays an important role in the area of logic synthesis and formal verification. Since the variable ordering has great impacts on the size and form of BDD, finding a good variable order is very important problem. In this paper, a new variable ordering scheme called incremental optimization algorithm is presented. The proposed algorithm reduces search space more than a half of that of the conventional sifting algorithm, and computing time has been greatly reduced without depreciating the performance. Moreover, the incremental optimization algorithm is very simple than other variable reordering algorithms including the sifting algorithm. The proposed algorithm has been implemented and the efficiency has been shown using many benchmark circuits.

Ⅰ. 서 론

부울(Boolean) 함수의 보다 효과적인 표현과 조작은 컴퓨터 지원 설계 (CAD : computer-aided design) 분야에서 필수적으로 요구된다. 이러한 예로는 조합 회로의 검증 (combinational logic verification), 순차 회로의 동일성 여부 검사 (sequential-machine equivalence checking), 조합 회로의 논리 최적화 (logic optimization of combinational circuits), 테스트 패턴 생성 (test pattern generation) 등이 있다. 이러한 목적으로, 부울

* 숭실대학교 대학원 전자계산학과
** 숭실대학교 컴퓨터학부
論文番號 : 97381-1021
接受日字 : 1997年 10月 21日

함수의 효과적인 표현과 조작을 위한 자료 구조로서 최근에 BDD (*binary decision diagrams*)에 관한 많은 연구가 진행되고 있다[1, 2].

BDD의 최적화를 위한 기법으로 많은 방법들이 연구되어 왔다. 이중 변수 순서화(variable ordering) 문제는 가장 널리 알려진 최적화 문제이다. BDD의 크기는 변수 순서에 의존하여 선형적 혹은 지수적으로 증가하므로, BDD 조작 이전에 적절한 변수 순서를 구하는 문제는 매우 중요하다. Bryant는 덧셈기나 곱셈기와 같은 산술회로들이 양질의 순서(*good order*)에서는 입력의 수에 선형적으로 증가하고, 그렇지 않은 경우 입력의 수에 지수적으로 증가하는 것을 보였다[1]. BDD에서 최적의 순서(*best order*)를 찾는 문제는 근본적으로 NP-complete 문제이며, 모든 최적의 순서 탐색 방법은 소모적 탐색 (exhaustive search) 기법에 의존한다[3, 4]. Friedman과 Supowit가 제안한 exact 알고리듬은 입력변수 n에 대하여 $O(n^2 3^n)$의 복잡도를 갖는다[3]. 따라서, 많은 변수 순서화 문제에 대한 연구가 양질의 순서를 구하는 휴리스틱 개발에 초점이 맞추어져 있다[4, 5, 6]. 이러한 휴리스틱들은 어떤 회로에는 좋은 결과를 제공하지만, 그렇지 않은 경우도 있다. 대부분의 휴리스틱은 빠른 실행 속도와 최적화 정도 사이의 trade-off 의 결정에 크게 의존한다.

R. Rudell에 의해서 제안된 동적 (dynamic) 변수 순서화 방법은 변수 순서화 알고리듬을 garbage 수집 시간에 수행하는 것으로서, 현재 모든 변수 순서화 알고리듬의 기본적인 접근 방법이다[5]. 하지만 동적 변수 순서화 방식은 garbage 수집 시간에 변수 순서화 알고리듬을 수행하지 않는 경우 보다 약 6-10 배 정도의 시간이 소요된다. 본 논문에서는 이러한 동적 변수 순서화 방식의 시간 오버헤드를 줄이기 위한 알고리듬으로 점진적 시프팅 알고리듬을 제안한다. 본 연구에서는 제안된 알고리듬을 기존의 BDD 패키지를 이용하여 구현 하였으며 IWLS'91 벤치마크 회로를 이용하여 BDD를 구성하는 과정에 적용하였다[7].

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 이진 결정 다이어그램, 변수 순서화의 효과의 기본개념에 대해 설명하고, 3장에서는 기존의 시프팅 (sifting) 알고리듬에 대해 기술한다. 4장에서는 제안된 알고리듬인 점진적 시프팅 알고리듬에 대해서 기술하고, 5장에서는 실험 결과를 보인다. 그리고 6 장에서는 결론 및 향후 연구 방향에 대해 논의한다.

Ⅱ. 기본개념

1. 결정 다이어그램

S. B. Akers는 디지털 함수를 정의, 분석, 검사 및 구현하는 방법으로서 BDD를 정의하였다[8]. BDD는 formal method 로서 부울 함수를 *if-then-else* 형태로 표현하고 조작하는 기술이다. 부울 함수 $X_n = \{x_1, x_2, \ldots, x_n\}$에 대한 BDD는 루트 (root) 노드가 존재하는 DAG (directed acyclic graph) 형태로서 $G = (V, E)$의 형태로 표현된다. V는 nonterminal과 terminal 두가지 형태의 vertex를 갖는다. X_n에 대한 nonterminal vertex V는 두 개의 후손 vertex인 $low(v)$, $high(v) \in V$를 가지며, 이것은 각각 수식 (1)처럼 샤논(Shannon)의 확장된 표현 형태를 가진다. Terminal vertex는 0과 1로 불리는 두개의 노드로 구성되며 후손을 갖지 않는다. 또한 그래프 G는 BDD의 루트 노드라 불리는 하나의 소스 노드를 갖는다.

$$f = \bar{x} \cdot f|_{x \leftarrow 0} + x \cdot f|_{x \leftarrow 1} \tag{1}$$

R. E. Bryant는 BDD에 변수 순서화 제약조건을 추가하여 부울 함수 조작을 위한 그래프 기반의 알고리듬을 제안하였다[1]. 순서화된 BDD (OBDD: Ordered BDD)는 그래프의 모든 경로 상의 모든 입력변수가 고정되어 있고 각 입력변수는 단 한번만 나타나는 BDD를 말한다. 부울 함수에 대한 연산은 OBDD 상의 그래프 알고리듬으로 표현할 수 있다. 또한 BDD의 유일한 표현 형태인 축약된 OBDD (ROBDD: Reduced OBDD)를 동일한 형태의 부그래프 (sub-graph)가 없고, 같은 노드로의 절선이 없는 형태의 OBDD를 말한다[1]. ROBDD는 하나의 부울 함수에 대해서 그 형태가 유일하므로 형식검증 (formal verification)에 많이 사용된다. 본 논문에서는 표현의 단순함을 위해서 ROBDD를 단순히 BDD라고 표현하였다. 이외에도 BDD의 변형된 형태로서 FDD, MTBDD, BMD, ZBDD, ADD 및 EVBDD 등의 변종이 존재하지만 변수 순서화 문제는 본질적으로 동일하다.

2. 변수 재순서화의 효과

BDD상의 노드 수는 변수 순서화에 따라서 선형적 혹은 지수적으로 변화한다. 그림 1은 부울 함수 $f =$

$a_1b_1+a_2b_2+a_3b_3$에 대한 변수 순서화의 효과를 보여주고 있다 [2].

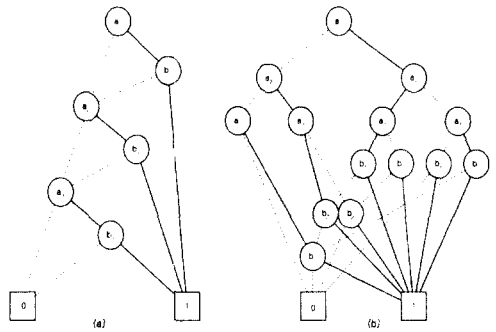


그림 1. 부울 함수에 대한 서로 다른 변수 순서화의 OBDDs
Fig. 1 OBDDs of the Boolean function $f = a_1b_1+a_2b_2+a_3b_3$ for two different variable orderings

그림 1의 (a)는 a_1, b_1, a_2, b_2, a_3, b_3의 변수 순서를 가지고 있으며 (b)는 a_1, a_2, a_3, b_1, b_2, b_3의 변수 순서를 가지고 있다. 그림 (b)에서와 같이 입력 변수의 순서를 다르게 했을 때 BDD의 크기 변화가 큼을 알 수 있다.

따라서, BDD에서 적절한 변수 순서화를 찾는 문제는 매우 중요한 문제이다. 많은 휴리스틱과 exact 알고리듬 [3, 5, 9] 이 이러한 문제를 해결하기 위해 개발되었다. 깊이 우선 탐색 (depth-first heuristic) 기법과 같은 대부분의 휴리스틱은 회로에 대한 분석을 통하여 생성될 BDD에 대한 변수 순서를 결정하는 것이다. 이러한 접근은 단순히 BDD의 초기생성시에만 사용되어 질 수 있다. 뿐만 아니라 이러한 휴리스틱도 좋지 못한 결과를 가져올 수 있다. 이러한 이유로 주어진 변수 순서에서 보다 나은 변수 순서로 향상시키는 변수 재순서화 알고리듬이 필요하다.

변수 재순서화 알고리듬은 국부 탐색 (local search) 기법을 이용한 연속적인 향상을 기반으로 한다. 시프팅 알고리듬은 개개의 변수에 대한 최적 위치를 결정하는 것으로 기본적으로 국부탐색 방법에 의존한다. 본 논문에서 제안하는 점진적 시프팅 알고리듬은 최적화된 임의의 영역을 확장시키는 것을 기본으로, 최적화된 영역 내에서만 국부탐색을 수행한다. 다음 장에서는 대표적인 변수 재순서화 알고리듬인 시프팅 알고리듬에 대해서 설명한다.

Ⅲ. 시프팅 알고리듬

시프팅 알고리듬은 R. Rudell에 의해서 제안되었다 [5]. 시프팅 알고리듬은 인접한 변수들 사이의 반복된 교환으로 국부탐색을 수행하는 알고리듬이다. 시프팅 알고리듬은 각각의 변수에 대한 최적의 위치를 찾는 것을 기반으로 한다. 그림 2는 시프팅 알고리듬의 한 예를 보여준다. 시프팅 알고리듬은 $O(n^2)$ 개의 인접 변수 교환이 필요하다.

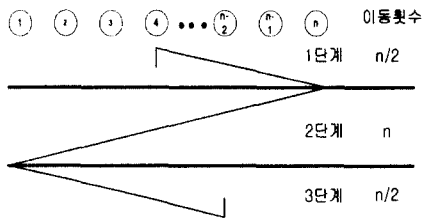


그림 2. 시프팅 알고리듬의 예
Fig. 2 An example of sifting algorithm

그림 2에서 4번 위치의 노드는 1, 2 단계의 사선 방향으로 이동하면서 n개의 새로운 순서열을 만들게 되고 각각의 경우에 다양한 BDD 크기를 얻을 수 있게된다. 이러한 과정에서 4번 위치의 노드가 어느 위치에 있을 때 제일 작은 BDD 크기를 갖는지에 대한 정보를 얻을 수 있다. 이렇게 얻어진 최적 위치로의 복귀 작업이 3단계에서 이루어지면 4번 위치의 노드에 대한 시프팅 작업은 끝나게 되고 모든 노드들에 대하여 순서적으로 위의 과정을 반복하여 최종적으로 간략화된 형태의 BDD를 얻게된다. 최종 결과를 얻기까지 각 노드의 평균 이동 횟수는 $n/2+n+n/2 = 2n$ 이다.

Ⅳ. 점진적 시프팅 알고리듬

점진적 시프팅 알고리듬은 BDD 크기를 최소화 하기 위한 새로운 국부탐색 알고리듬이다. 이 알고리듬의 기본 접근방식은 최적화된 영역을 확장시키는 것이다. 이 방법의 장점은 탐색공간을 크게 줄일 수 있다는 것이다. 이 방식에서 변수에 대한 최적 위치의 판

별은 BDD 전체 변수가 아닌 최적화된 영역 안의 변수에 대해서만 수행된다. 제안된 알고리듬은 단순한 형태로 그림 3에 표현되어 있다.

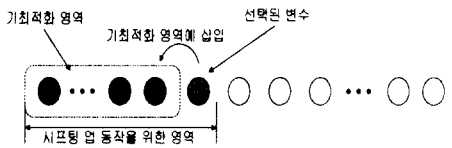


그림 3. 점진적 시프팅 알고리듬
Fig. 3 Incremental Sifting algorithm

제안된 알고리듬은 초기 입력변수 배열을 최상위 혹은 최하위 변수로부터 시작하여, 한 쪽 방향으로 하나씩 증가하면서 새로운 변수를 선택하게 된다. 그림 3에서 각 원은 BDD 상의 변수를 나타낸다. 검은색 원으로 이루어진 집합은 이미 최적화 되어 있는 영역을 나타내며, 회색으로 표시된 원은 최적화 되어있는 영역으로 삽입되기 위해서 선택되어진 변수 (chosen variable) 를 뜻한다. 선택된 변수는 최적화된 영역으로 시프팅되면서 최적화된 영역 안에서의 최적의 위치를 결정하게 되고 파악된 최적의 위치로 이동하면서 최적화된 영역이 확장되어진다. 이러한 동작이 모든 변수들에 대해 수행된다. 그림 4는 점진적 시프팅 알고리듬의 간단한 예이다. 그림에서, 전체 변수는 7개이고, 초기의 기최적화된 영역은 진하게 표현된 {x3, x1, x4, x2, x5}이다. 여기에서 최적화된 영역의 인접한 변수인 x6가 새롭게 선택되어진다. 최적화된 영역의 크기 만큼인 5 번의 인접 변수 교환으로 영역내의 모든 위치에 대해 최적의 위치를 파악할 수 있게된다. 또한 별표 (*) 로 표시된 곳이 최소의 크기를 갖는 최적의 위치로서 이곳으로의 이동은 3 번의 교환 (평균적으로 최적화된 영역의 크기의 절반)을 필요로 한다. 그러면, 새로운 최적 영역인 {x3, x1, x4, x6, x2, x5}가 생성된다.

이러한 접근방식에서 각 변수의 최적 위치는 기존 방식에서 사용되었던 전체 변수상위 위치가 아니라, 최적화된 영역 내에서의 위치이다. 이 방식은 기존의 방법과 비교하여 성능면에서는 비슷하며, 속도면에서는 훨씬 우수하다. 이와 같은 결과는 탐색공간의 축소로부터 예측이 가능하다. 제안된 알고리듬의 전반적인 프로시져는 그림 5와 같다.

x3, x1, x4, x2, x5, x6, x7	initial
x3, x1, x4, x2, x6, **x5,** x7	swap(x5, x6)
x3, x1, x4, x6, **x2, x5,** x7	swap(x2, x6) *
x3, x1, x6, **x4, x2, x5,** x7	swap(x4, x6)
x3, x6, **x1, x4, x2, x5,** x7	swap(x1, x6)
x6, **x3, x1, x4, x2, x5,** x7	swap(x3, x6)
x3, x6, **x1, x4, x2, x5,** x7	swap(x6, x3)
x3, x1, x6, **x4, x2, x5,** x7	swap(x6, x1)
x3, x1, x4, x6, **x2, x5,** x7	swap(x6, x4)

그림 4. 점진적 시프팅 알고리듬의 예
Fig. 4 An example of incremental sifting algorithm

```
Foreach (all variables in the BDD)
begin
    Initialize current_size_of_BDD, minimum_size_of_BDD, and optimal_index;

    while explore all possible positions in the optimized region do
    begin
        swap(previous_variable, variable);
        Update current_size_of_BDD;
        if current_size_of_BDD is smaller than minimum_size_of_BDD then
        begin
            Update minimum_size_of_BDD;
            Update optimal_index;
        end;
    end;

    repeat
        swap(variable, next_variable);
    until move to optimal_index;
end;
```

그림 5. 점진적 시프팅 알고리듬의 프로시져
Fig. 5 The procedure of incremental sifting algorithm

그림 5에 제안된 알고리듬의 의사코드 (pseudo code)를 나타 내었다. 제안된 알고리듬의 프로시져 구성은 크게 두 부분으로 구성되는데 첫 번째는 선택변수를 최적화 영역에 삽입하여 최적의 위치를 찾는 것이며 두 번째는 구한 최적의 위치로 선택변수를 이동시키는 과정이다.

그림 4와 5에서 볼 수 있듯이, 제안된 알고리듬은 시프팅 알고리듬을 비롯한 기존의 알고리듬에 비해 매우 단순하며 빠르다. 기존 방식은 임의로 여러개의 순서열을 검사하는 방식을 통해 BDD의 크기 축소가 이루어지는데 반해, 제안된 알고리듬에서는 최적화 과정이 부분적으로 최적화 된곳에서만 이루어지므로 최적화 정도가 기존 방식과 다르다. 제안된 알고리듬의 효율성은 다수의 벤치마크 회로를 이용하여 검증

하였다. 시프팅과 마찬가지로 [6, 10] 제안된 알고리듬은 여러 가지 휴리스틱과 결합하여 확장이 가능하다.

1. 연산량 분석

기존의 시프팅 알고리듬은 각 변수에 대해서 시프팅-다운, 시프팅-업, 최적위치로의 이동의 세 가지 동작을 수행한다. 각 단계에서 평균적으로 $\frac{n}{2}$, n, 그리고 $\frac{n}{2}$의 연산량을 가지며, 전체적으로는 $2n$의 연산량을 갖는다. 따라서, 전체 변수 n개에 대한 연산량은 $2n^2$이다.

그러나, 제안된 점진적 시프팅 알고리듬은 각 변수에 대해서 양끝 단으로의 이동이 불필요하다. 또한 최적화된 영역 내에서만 최적위치가 결정되므로, 평균적으로 연산량을 절반으로 줄일 수 있다.

따라서 각 변수에 대한 연산량을 알아보면 노드가 n개일 때 수행도중 최적화 영역의 평균은 n/2이고 선택된 노드를 최적화 영역안에 삽입하여 최적화 영역 안의 모든 노드와 위치를 바꾸는데 n/2의 연산량을 가지며 최적의 위치로 복귀하는데 평균적으로 최적화 영역의 절반인 n/4의 연산량이 필요하므로 하나의 노드에 대하여 (n/2+n/4) $\frac{3}{4}n$ 만큼의 연산량을 갖는다. 따라서 전체 변수에 대해서는 $\frac{3}{4}n^2$의 연산량을 가지며 결과적으로 점진적 시프팅 알고리듬은 평균적인 연산량 분석에서 2.67 배의 속도증가를 얻을 수 있다.

시프팅 알고리듬과 점진적 시프팅 알고리듬은 본질적으로 $O(n^2)$의 연산량을 갖지만, 후자 쪽이 보다 작은 상수항을 갖는 장점이 있으며, 따라서 시프팅 알고리듬보다 빠른 시간내에 최적화를 얻을 수 있다. 이는 다수의 벤치마크 회로를 이용하여 실험으로서 검증되었다.

V. 실험 결과

이 절에서는 제안된 알고리듬의 효율성을 검증하기 위한 실험결과를 보인다. 실험을 위하여, CUDD 패키지 [11]와 64MB 메모리의 SUN SPARC20 기종과 ISCAS'85 회로를 포함하는 IWLS'91 벤치마크 회로를 사용하였다. 표 1은 시프팅 알고리듬과 제안된 알고리듬과의 비교 결과를 보여준다.

표 1. 실험결과: 시프팅 알고리듬과 점진적 시프팅 알고리듬의 비교

Table 1. Experimental Results: Comparison of the incremental sifting with the conventional sifting

회로명	입력 개수	출력 개수	초기 크기	시프팅		점진적시프팅		비율	
				크기	시간	크기	시간	크기	시간
alu2	10	6	231	160	0.12	191	0.07	1.19	0.58
alu4	14	8	1,182	732	0.47	753	0.17	1.03	0.36
apex6	135	99	2,760	532	1.97	616	1.55	1.16	0.79
apex7	49	37	1,660	306	0.72	312	0.34	1.02	0.47
b9	41	21	178	109	0.30	167	0.15	1.53	0.50
c1355	41	32	45,922	31,439	88.53	33,560	44.26	1.07	0.50
c1908	33	25	36,072	7,580	54.62	6,865	39.12	0.91	0.72
c432	36	7	1,733	1,210	2.42	1,226	0.95	1.01	0.39
c499	41	32	45,922	31,439	87.13	33,560	44.10	1.07	0.51
c8	28	18	136	82	0.22	129	0.11	1.57	0.50
cc	21	20	101	60	0.12	65	0.06	1.08	0.50
cht	47	36	150	90	0.20	127	0.12	1.41	0.60
cm150a	21	1	131,071	33	25.68	33	9.08	1.00	0.35
cm151a	12	2	511	17	0.12	17	0.06	1.00	0.50
cm152a	11	1	383	16	0.10	16	0.05	1.00	0.50
cm163a	16	5	55	33	0.09	36	0.05	1.09	0.56
cm82a	5	3	16	16	0.04	8	0.03	0.50	0.75
cm85a	11	3	38	36	0.09	21	0.05	0.58	0.56
cordic	23	2	45	42	0.26	41	0.09	0.98	0.35
i8	133	81	4,366	2,104	5.42	2,281	2.50	1.08	0.46
k2	45	45	28,336	1,425	5.60	1,573	2.44	1.10	0.44
lal	26	19	165	119	0.19	104	0.08	0.87	0.42
mux	21	1	131,071	33	25.55	33	9.12	1.00	0.36
my_adder	33	17	327,677	84	218.68	170	12.10	2.02	0.06
parity	16	1	17	17	0.17	10	0.04	0.59	0.23
rot	135	107	166,674	9,782	141.01	6,242	54.39	0.64	0.39
term1	34	10	580	203	0.44	163	0.19	0.80	0.43
vda	17	39	4,345	504	0.67	510	0.22	1.01	0.33
x2	10	7	69	37	0.07	33	0.04	0.89	0.57
x3	135	99	2,760	532	1.93	616	1.54	1.16	0.80
x4	94	71	891	549	1.18	646	0.55	1.18	0.47
z4ml	7	4	47	17	0.05	16	0.03	0.94	0.60
Totals	-	-	935,194	89,338	664.16	90,140	223.65	1.01	0.34

첫번째 열은 회로의 이름을 알파벳순으로 기술하였다. BDD 의 초기 크기, 시프팅에 의한 결과, 점진적 시프팅에 의한 결과는 각각 초기 크기, 시프팅의 크기, 그리고 점진적시프팅의 크기란에 나타나있다. 시간으로 기술된 모든 열은 CPU 시간을 의미한다. 마지막 두 열은 제안된 알고리듬과 기존의 시프팅 알고리듬의 크기와 시간 면에서의 비율을 나타낸다.

표에서 보듯이, 점진적 시프팅 알고리듬은 기존의 시프팅 알고리듬과 비교하여 거의 동일한 성능을 내며, 매우 빠른 속도의 시간 향상을 갖는 것을 알 수 있다. 평균적인 CPU 시간의 향상은 각 결과에 대한 정규화된 평균으로서 294%이다. BDD 크기는 오직 1%만이 증가하였다. 32 개의 실험 결과 중 9 개는

크기면에서도 효과적인 것으로 나타나 있다. 특별한 경우로서 my_adder와 comp 회로는 CPU 시간의 감소가 각각 0.06과 0.01로서 매우 큰 것을 알 수 있다. 제안된 알고리듬은 일반적으로 회로가 크고 복잡할수록 속도증가 폭이 큰 것을 확인할 수 있었다. 이러한 장점은 실제 회로를 다루는 데에 있어 매우 중요하게 사용될 것이다.

Ⅵ. 결 론

BDD의 최적화는 검증과 합성 단계에서 매우 중요하다. 본 논문은 점진적 시프팅이라 부르는 새로운 변수 순서화 알고리듬을 제안하고, 효율성을 검증하였다. 제안된 알고리듬은 제한된 영역 내에서만 탐색이 이루어지므로, 탐색공간을 크게 감소시키는 장점을 갖는다. 더욱이 제안된 알고리듬은 매우 간단한 구조를 가지며, 따라서 구현이 용이하다. 점진적 시프팅 알고리듬은 기존의 알고리듬에 비하여 이론적으로는 2.67배, 실험적으로는 2.94배가 빠르게 수행됨을 알 수 있었다.

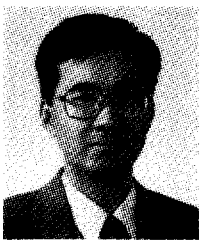
향후 연구과제로는 제안된 점진적 시프팅 알고리듬의 개선에 대한 연구와 그룹 시프팅 [6]과 같은 다른 알고리듬이나 선형 변환 [10, 12]과 같은 기법과의 조합을 통한 성능 향상등이다.

참 고 문 헌

1. R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transaction on Computers*, vol. C-35, pp. 667-691, August 1986.
2. R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293-318, September 1992.
3. S. J. Friedman, K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *IEEE Transactions on Computers*, Vol. C-39, No. 5, pp. 710-713, May 1990.
4. N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variables," *International Conference on Computer Aided Design*, pp. 472-475, November 1991.
5. R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *International Conference on Computer Aided Design*, pp. 42-47, November 1993.
6. S. Panda, F. Somenzi, "Who are the variable in your neighborhood," *International Conference on Computer Aided Design*, pp. 74-77, November 1995.
7. Moon-Bae song, Hoon Chang, "A Variable Reordering Algorithm for the fast optimization of Binary Decision Diagrams," Asian Test Symposium '97, pp. 228-233, 1997.
8. S. B. Akers, "Binary Decision Diagrams," *IEEE Transaction on Computers*, vol. C-27, 6 (August), pp. 509-516, 1978.
9. R. Drechsler, B. Becker, and N. Gockel, "A genetic algorithm for variable ordering of OBDDs," *International Workshop on Logic Synthesis*, Granlibakken, CA, May 1995.
10. C. Meinel, F. Somenzi, "Linear Sifting of Decision Diagrams," In *Proceeding of Design Automation Confernece*, pp. 202-207, June 1997.
11. F. Somenzi, CUDD : CU Decision Diagram Package. Release 2.1.2, University of Colorado, April 12, 1997.
12. C. Meinel, T. Theobald, "Local encoding transformations for optimizing OBDD-representations of finite state machines," In *Proceedings of FMCAD*, pp. 404-418, November 1996.

송 문 배(Moon-Bae Song) 비회원
1973년 9월 28일생
1996년 : 군산대학교 컴퓨터과학과 (이학사)
1998년 : 숭실대학교 대학원 전자계산학과 (공학석사)
※주관심분야 : VLSI/CAD, binary decision diagrams

동 균 탁(Gyun-Tak Dong) 비회원
1971년 4월 20일생
1995년 : 서울 산업대학교 전자계산학과 (공학사)
1998년 : 숭실대학교 대학원 전자계산학과 (석사과정)
※주관심분야 : 시스템 프로그래밍, 컴퓨터구조, Formal verification



장 훈(Chang Hoon) 정회원
1987년 : 서울대학교 전자공학과 졸업 (B.S.).
1989년 : 서울대학교 전자공학과 졸업 (M.S.).
1993년 : University of Texas at Austin 박사학위 취득.
1991년 : IBM Inc.
1993년 : Motorola Inc. Senior Member of Technical Staff.
1994년~현재 : 숭실대학교 컴퓨터학부 조교수.
※주관심분야 : 컴퓨터 시스템, VLSI 설계, VLSI 테스팅 등임
e-mail : hoon@computing.soongsin.ac.kr