論文2002-39SD-2-10

BDD 최소화 문제에 적용하기 위한 μO 알고리즘의 최적화

(Optimization of μO Algorithm for BDD Minimization Problem)

李 珉 那 *, 曺 常 榮 *

(Min-Na Lee and Sang-Young Cho)

요 약

BDD는 부울 함수를 간결하고 유일하게 나타낼 수 있기 때문에 다양한 컴퓨터 지원 설계 분야에 널리 활용되고 있다. BDD 크기는 변수 순서에 따라 매우 민감하게 변하므로, BDD의 크기를 최소화할 수 있는 변수 순서를 구하는 것은 매우 중요한 문제이다. 그러나 최적의 변수 순서를 찾는 것은 NP-complete 문제이므로, 근사 최적 변수 순서(near-optimal variable ordering)를 결정하기 위한 여러 가지 휴리스틱 알고리즘이 제안되었다. 본 논문에서는 최근에 제안된 μO (Microcanonical Optimization) 알고리즘에 기반하여 BDD 최소화 문제에 더욱 적합하도록 보완한 Faster-μO 알고리즘을 제안한다. Faster-μO 알고리즘은 빠르고 더 나은 해를 찾기 위한 국부 탐색 방법으로 임의의 해를 반복적으로 생성하는 초기화 단계 대신에 시프팅 알고리즘을 실행하는 시프팅 단계로 대체한다. 제안된 알고리즘의 튜닝을 위하여 실험적으로 알고리즘 파라메타를 구하였으며 제안된 알고리즘은 많은 벤치마크 회로에 대하여 실험되었으며 기존의 μO 알고리즘 보다 빠르고 좋은 성능을 보인다.

Abstract

BDD have become widely used for various CAD applications because Boolean functions can be represented uniquely and compactly by using BDD. The size of the BDD representation for a function is very sensitive to the choice of orderings on the input variables. Therefore, it is very important to find a good variable ordering which minimize the size of the BDD. Since finding an optimal ordering is NP-complete, several heuristic algorithms have been proposed to find good variable orderings. In this paper, we propose a variable ordering algorithm, Faster-μO, based on the μO(microcanonical optimization). In the Faster-μO algorithm, the initialization phase is replaced with a shifting phase to produce better solutions in a fast local search. We find values for algorithm parameters experimentally and the proposed algorithm has been experimented on well known benchmark circuits and shows superior performance compared to various existing algorithms.

* 正會員, 韓國外國語大學校 情報産業工科大學 컴퓨터工學科

(Hankuk University of Foreign Studies College of Information Industrial Engineering Computer Science and Engineering Department)

接受日字:2001年9月19日, 수정완료일:2002年1月2日

Ⅰ. 서 론

이진결정도(BDD : Binary Decision Diagrams)는 부울 함수의 표현과 조작에 효과적인 자료구조이다[1-5]. BDD의 변수 순서에 제약조건을 준 "순서화 BDD

(OBDD : Ordered Binary Decision)"와 OBDD에 축약 규칙을 적용한 "축약된 순서화 BDD(ROBDD : Reduced Ordered Binary Decision Diagrams)"가 소개되면서 ROBDD는 하나의 함수에 대해 변수 순서가 일정할 때 유일한 형태로 표현되는 정준형(canonical form)을 갖는 특성과 효과적인 부울 함수의 조작 기술로 인하여 논리합성(logic synthesis), 형식검증(formal verification) 등과 같은 컴퓨터 지원 설계(CAD : Computer Aided Design) 분야에 널리 활용되기 시작했다[3].

BDD의 크기는 변수 순서에 매우 민감하며, 최악의 경우, 변수 개수에 비례하여 그 크기가 지수적으로 증가한다. 따라서 변수의 개수가 많을 경우 변수 순서에 따라서 BDD의 크기는 컴퓨터로 다룰 수 없을 만큼 매우 커질 수 있다. BDD의 효율적인 조작을 위하여 적절한 변수 순서 선택에 의한 BDD 크기 최소화는 모든 BDD 응용 분야에서 매우 기본적이고도 중요한 문제이다[6].

그러나 BDD 크기가 최소인 최적의 변수 순서를 구하는 문제는 NP-complete 문제이다[7]. 따라서 변수 순서를 구하는 다양한 휴리스틱 알고리즘들이 제안되었다. 초기에 제안된 BDD 변수 순서 알고리즘으로 시프팅(Sifting) 알고리즘[8]이 있으며, 이는 각각의 변수의 최적 위치를 찾기 위해 인접 변수와 위치를 서로 교환하며 모든 위치를 순회하는 방법으로 그중 크기가 가장 작은 BDD의 변수 순서를 찾아내는 알고리즘이다. 이러한 시프팅 알고리즘보다 더욱 향상된 해를 찾기 위해 유전자(Genetic) 알고리즘[9-11]과, SA(Simulated Annealing) 알고리즘[12]을 이용한 BDD 최소화 기법이 제안되었다. 그러나 이와 같은 휴리스틱 알고리즘은 최적에 근접한 해를 구하지만 실행 시간이 매우 오래 걸리는 단점을 가지고 있다.

최근에 μO(Microcanonical Optimization)[13] 알고리즘을 이용한 BDD 최소화 기법[19]이 제안되었으며 이 알고리즘은 우수한 해를 구하며 수행시간이 짧은 장점이 있다. μO 알고리즘은 초기화 단계와 샘플링 단계를 반복 실행하는 알고리즘으로, 초기화 단계는 국부 최적해를 찾는 단계로 임의의 해를 제안하여 향상된 해만을 받아들이는 단계이며, 샘플링 단계는 전체 최적 해를 찾기 위하여 초기화 단계에서 찾은 국부 최적 해를 벗어나기 위한 탐색과정으로 총 에너지가 같은 일정한 탐색 범위 내에서 나쁜 결과의 해도 받아들이는 단계이다. 이와 같은 초기화 단계와 샘플링 단계를 종료 조건이 만족할 때까지 반복 실행한다.

본 논문은 기존의 μO 알고리즘을 BDD 최소화 문제에 적합하도록 더욱 최적화된 Faster-μO 알고리즘을 제안한다. 기존의 μO 알고리즘의 초기화 단계에서는 임의의 해를 생성하여 향상된 해만을 받아들이는 국부 탐색(Local Search)을 실행한다. 그러나 Faster-μO 알고리즘은 초기화 단계를 시프팅 알고리즘으로 대체한다. 임의의 해를 생성하는 것은 BDD 문제에서는 매우 오랜 시간이 걸리며, 또한 임의의 해가 최악의 경우이면 BDD로 생성되지 못할 수도 있다. 그러므로 임의의 해를 반복적으로 생성하는 초기화 단계 대신에 시프팅 방법을 이용하여 더욱 빠르고 나은 해를 구하도록 한다.

본 논문의 실험은 잘 알려진 IWLS91 벤치마크 데이터를 통해 제안된 알고리즘이 기존의 제안된 다른 휴리스틱 알고리즘에 비해 우수함을 보여준다. 또한 Faster-μO 알고리즘의 파라메타는 외부 반복횟수, 데몬 크기, 샘플링 단계의 반복횟수가 있다. 이러한 파라메타를 실험을 통해 상관관계를 분석하고, 최적의 파라메타 값을 결정한다.

본 논문의 구성은 다음과 같다. 2장에서는 BDD의 개념을 설명하고, 3장에서는 μO 알고리즘의 개념을 설명한다. 4장은 BDD 최소화 문제에 적합한 Faster-μO 알고리즘을 설명하고, 5장에서는 실험결과를 통해 알고리즘의 파라메타를 분석하며, 제안된 알고리즘의 성능 및 효율성이 우수함을 보이고, 6장에서 결론을 맺는다.

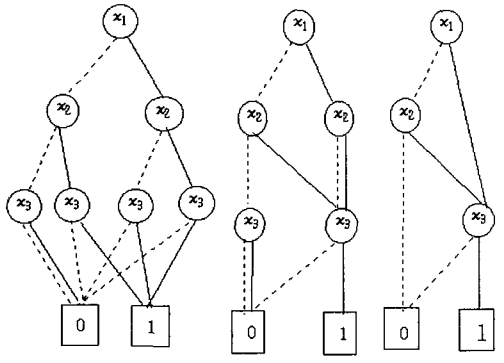
Ⅱ. BDD의 개념

BDD는 부울 함수를 비순환 방향 그래프((DAG : Directed Acyclic Graph)로 나타낸 것을 말한다. BDD는 n개의 입력변수, $X_n=\{x_1, x_2, \cdots, x_n\}$에 대한 루트 노드(root node)가 존재하며, 노드(node)와 에지(edge)로 구성되는 그래프 $G=(V, E)$의 형태로 표현된다. 노드는 비단말 노드(nonterminal node)와 단말 노드(terminal node) 두 가지 형태로 구분된다. 변수를 나타내는 비단말 노드(ν)는 두 개의 후손 노드인 low(ν), high(ν) $\in$V를 가진다. 이것은 각각 수식 (1)처럼 샤논 확장(Shannon expansion) 형태로서 변수 x_i에 관한 함수 f로 표현하게 된다. 부울 함수의 논리 값을 나타내는 단말 노드는 0 과 1을 나타내는 두 개의 노드로 구성되며 후손 노드를 가지지 않는다.

$$f = \bar{x}_i \cdot f|_{xi=0} + xi \cdot f|_{xi=1}$$
$$= \bar{x}_i \cdot f(x_0, x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$$
$$+ x_i \cdot f(x_0, x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \quad (1)$$

Bryant는 BDD의 변수 순서에 제약조건을 추가한 "순서화 BDD(OBDD : Ordered BDD)"를 제안하였다[2-3]. OBDD란 그래프의 모든 경로(path) 상의 변수 순서가 고정되어 있고, 변수는 각 경로에서 최대한 한번만 나타나는 BDD를 의미하는 것으로 부울 함수에 대한 연산은 OBDD상의 그래프 알고리즘으로 표현 할 수 있다.

OBDD에 축약 규칙을 적용한 "축약된 순서화 BDD (ROBDD : Reduced Ordered BDD)"는 하나의 부울 함수에 대해 변수 순서가 일정할 때 정준형의 BDD를 갖는 특성을 갖는다. 본 논문에서의 BDD란 ROBDD를 의미한다. 축약규칙은 유일성(Uniqueness)과 비 중복성 시험(Non-redundant tests)이 있다. 유일성은 서로 다른 비단말 노드 u와 v가 그 변수명이 같고, 각각의 후손이 서로 동일하다면, 두 노드중 한 노드는 삭제하고, 삭제한 노드로 연결되는 에지는 남은 다른 노드로 연결하는 것이다. 비 중복성 시험은 비단말 노드 v의 두 후손 노드가 서로 동일한 노드이면, 노드 v는 삭제하고 v 노드로 연결되는 에지는 v 노드의 후손 노드로 연결한다. 그림 1은 $f = x_1x_3 + \bar{x}_1x_2x_3$ 함수의 OBDD를 축약 규칙을 적용하여 ROBDD로 만들어 가는 과정을 나타낸 그림이다.



(a) OBDD (b) 유일성적용 (c) 비중복성 시험 적용

그림 1. $f = x_1x_3 + \bar{x}_1x_2x_3$ 함수의 OBDD의 축약과정
Fig. 1. OBDD reduction processing of function $f = x_1x_3 + \bar{x}_1x_2x_3$.

BDD의 크기는 그래프의 노드 개수를 의미한다. 그 크기는 변수 순서에 직접적으로 의존하므로 적절한 변수 순서의 선택은 매우 중요한 문제이다. 그러나 BDD의 최적 변수 순서를 결정하는 것은 NP-complete 문제이므로, 좋은 변수 순서를 결정하기 위하여 많은 변수 순서 최적화 기법들이 제안되었다[9-12]. 초기에 제안된 알고리즘은 Rudell이 제안한 시프팅(Sifting) 알고리즘으로, 시프팅은 인접변수의 순서 교환 기법(Swapping)을 기반으로 하는 동적 변수 순서화 알고리즘이다[8]. 이 알고리즘은 각각의 변수가 가능한 모든 변수 위치를 순회하여 자신의 최적 위치를 찾는 방법으로 한번 자신의 최적 위치를 찾은 변수는 그 위치가 고정된다. 이러한 방법으로 전체 BDD 크기가 최소화 된 변수 순서를 찾는다.

$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	initial
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	swap (x_4, x_5)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	swap (x_4, x_6)
$x_1, x_2, x_3, x_5, x_6, x_7, \mathbf{x_4}$	swap (x_4, x_7)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	swap (x_7, x_4)
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	swap (x_6, x_4)
$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	swap (x_5, x_4)
$x_1, x_2, \mathbf{x_4}, x_3, x_5, x_6, x_7$	swap (x_3, x_4)
$x_1, \mathbf{x_4}, x_2, x_3, x_5, x_6, x_7$	swap (x_2, x_4)
$\mathbf{x_4}, x_1, x_2, x_3, x_5, x_6, x_7$	swap (x_1, x_4)
$x_1, \mathbf{x_4}, x_2, x_3, x_5, x_6, x_7$	swap (x_4, x_1)
$x_1, x_2, \mathbf{x_4}, x_3, x_5, x_6, x_7$	swap (x_4, x_2)
$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	swap (x_4, x_3)
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	swap (x_4, x_5)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	swap (x_4, x_6)
$x_1, x_2, x_3, x_5, x_6, x_7, \mathbf{x_4}$	swap (x_4, x_7)

그림 2. 시프팅 알고리즘의 예
Fig. 2. Example of Sifting algorithm.

그림 2는 시프팅 알고리즘이 변수 x_4에 적용될 때 나타나는 변수교환의 실행을 보여준다. 전체 변수는 총 7개이므로, 최적 위치를 찾기 위해 선택된 변수 x_4의 가능한 위치는 7개의 위치이다. 각 위치로의 이동을 위해 9번의 인접 변수 교환(swap)을 실행하여야 하며, 다시 최적의 위치로 이동하기 위해 6번의 인접 변수 교환이 실행된다. 즉, 선택된 변수는 먼저 변수 순서의 하단 쪽으로 이동한 후 다시 최 하단에서 최 상단으로 이동하면서 해당 변수의 각 위치에서의 BDD 크기를 조사한다. 그림 3의 예에서는 x_4 변수가 x_7의 위치에 있을 때 BDD 크기가 최소화되는 곳이다.

그러나 이와 같은 시프팅 알고리즘은 국부 최적(local optimum)의 변수 순서를 구하는 방법으로 실행 시간은 빠르나 전체 최적(global optimum)에 가까운 변수 순서를 구하지 못하는 단점이 있다.

Ⅲ. Microcanonical Optimization(μO)

NP-complete 문제를 더욱 빠르고 효과적으로 해결하기 위하여 Creutz 시뮬레이션을 기반으로 한 μO 알고리즘이 제시되었다[14]. Creutz 시뮬레이션은 통계 물리학을 기반으로 하며, 외부 환경과 상호작용이 없는 독립된 시스템을 고려한다. 그러므로 시스템의 상태는 외부 환경의 변화와 상관없이 일정한 에너지를 가지며, 시스템의 모든 가능한 상태들은 동일한 확률로 분포되어 있다. 이러한 외부 환경과 독립되고 동일 확률 분포를 갖는 시스템의 어느 한 상태를 찾기 위한 시뮬레이션 알고리즘이 Creutz 알고리즘이다[13].

Creutz 알고리즘에는 여분의 자유도(extra degree of freedom)를 나타내는 데몬(demon)이 존재한다. 데몬은 항상 양수(positive) 값을 가지며, 시스템과 에너지를 서로 교환하면서 시스템과 상호작용 한다. 데몬은 $E_D(E_D < E_S)$로 표기하고, 데몬의 최대 용량을 E_{DMAX}로 표기하며, E_S는 시스템 에너지를 의미한다. 시스템의 총 에너지는 데몬과 시스템 에너지의 합이며 ($E_{TOTAL}=E_D+E_S$), 알고리즘의 샘플링 단계에서 데몬과 시스템의 에너지가 변하더라도 같은 샘플링 단계에서는 시스템의 총 에너지가 항상 일정한 값을 유지한다.

현재 시스템 상태에서 새로운 상태로의 임의 변환(random transition)의 수락 여부는 현재 상태와 새로운 상태의 에너지 차이 ($\triangle E = new\,E_S - current\,E_S$)에 따라 결정된다. 즉, $\triangle E$가 데몬에 의해 받아 들여 지거나, 또는 거절되어진다. 만약 $\triangle E > 0$ 일 경우는 $\triangle E \le E_D$ 일 때만 새로운 변환이 받아 들여 지고, $\triangle E \le 0$ 일 경우는 $E_D - \triangle E \le E_{DMAX}$ 일 때만 새로운 변환이 받아 들여 진다. 새로운 임의 변환이 받아 들여지면 데몬은 $E_D = E_D - \triangle E$ 으로 수정되어진다.

μO 알고리즘은 초기화 단계(Initialization Phase)와 샘플링 단계(Sampling Phase)의 2가지 단계로 구성되어 종료조건이 만족할 때까지 반복 실행한다. 초기화 단계의 목적은 빠른 국부 탐색(local search)을 구현하여 국부 최적 해를 찾는 것이며, 샘플링 단계의 목적은 초기화 단계에서 획득한 국부 최적 해를 벗어나기 위해 Creutz 알고리즘으로 같은 에너지 수준을 갖는 탐색 범위 안에서 대체 해(alternative solution)를 생성하는 것이다. 그러므로 탐색할 이웃 해의 영역은 $[E_S - E_{DMAX} + E_D, E_S + E_D]$ 이다.

μO 알고리즘을 기반으로 하여 BDD 최소화 문제를 해결하는 기법이 최근에 제안[19]되었다. 이 기법은 기존의 가장 최적의 근사해를 구하는 것으로 알려진 유전자 및 SA 알고리즘과 유사한 해를 구하면서도 그 실행 시간을 많이 단축시켰다. 그러나 BDD 최소화 문제 특성상 가장 많은 실행시간을 갖는 임의의 해 제안 부분은 개선되지 않았다.

Ⅳ. Faster-μO 알고리즘

1. Faster-μO 알고리즘 정의

BDD 최소화 문제 특성상 임의의 해 제안은 기존의 BDD를 제안된 변수 순서에 맞게 그래프를 변환시켜야 함으로 많은 시간이 걸린다. μO의 초기화 단계 목적은 빠른 시간에 국부 최소해를 찾는 것인데 임의의 해를 제안하는 방법은 이러한 목적에 부합되지 못한다. 따라서 더욱 빠른 국부 최적해를 찾기 위해 Faster-μO에서는 초기화 단계를 시프팅 단계로 대체한다. 시프팅의 실행시간이 임의의 국부 탐색의 실행 시간보다 더 빠르게 실행되기 때문이다.

그림 3은 BDD의 크기 최소화를 위해 제안된 Faster-μO 알고리즘이다. Faster-μO 알고리즘은 샘플링 단계와 시프팅 단계로 구분되며, 이 두 단계의 사이클이 종료조건을 만족할 때까지 반복 실행한다. BDD의 초기 크기를 계산하기 위해 임의의 변수 순서를 발생하여 BDD를 생성하지 않는다. BDD의 크기는 변수 개수와 순서를 가지고 구할 수 있는 평가함수가 존재하지 않으므로 실제로 BDD를 해당 변수 순서에 의해 생성하여 그 노드의 개수를 세어보아야 한다. 그러나 임의의 변수 순서를 발생하여 BDD를 생성하면 최악의 경우 BDD의 생성이 불가능 할 수 있다. 그러므로 좀더 효율적인 알고리즘 실행을 위하여 초기 BDD는 시프팅 방법을 이용하여 생성하고 그 크기를 계산한다.

샘플링 단계는 이전 시프팅 단계에서 얻어진 국부 최소해를 벗어나기 위한 작업이다. 샘플링 단계를 시작

```
Procedure Faster-μO
begin
  Sifting;
  Demon_Decide ;
  EDMAX = ED * 2 ;
  while(Stop-criterion) do
  begin
      iter = 0 ;
      while ( iter < SAMP_NUM) do          // Sampling
      begin
         iter++ ;
         Propose_Move ;
         if ( (ΔE ≤ ED) and (ED - ΔE ≤ EDMAX) ) then
         begin
            Accept_Move ;
            ED = ED - ΔE ;
         end if
      end while
      Sifting;
  end while
end
```

그림 3. BDD 최소화를 위한 Faster-μO 알고리즘
Fig. 3. Faster-μO algorithm for BDD minimization.

하기 위해 데몬과 데몬의 최대 값을 결정한다. 데몬은 초기 BDD 크기의 작은 일부 값으로 지정한다. 시프팅 단계에서 구한 해의 이웃해 중에서 임의의 새로운 해를 제안하고, 임의 해와 현재 해의 크기 차이를 데몬과 비교한다. △E가 양수일 경우는 $E_D - \triangle E \geq 0$ 일 때 임의 해를 현재 해로 받아들이고, △E가 음수일 경우는 $E_D - \triangle E \leq E_{DMAX}$ 일 때 임의 해를 현재 해로 받아들인다. 임의 해를 현재 해로 받아들일 때는 데몬을 $E_D = E_D - \triangle E$ 으로 수정한다.

2. μO와 Faster-μO의 실행시간 비교

임의의 해 제안은 현재 해의 이웃해 중에서 임의의 해를 선택하여야 한다. 임의의 해를 선택하는 방법은 일반적으로 3가지 방법을 혼용하여 사용된다. 현재 변수 순서에서 임의의 두 변수 위치를 서로 교환하는 Exchange transition, 임의의 한 변수를 앞쪽의 임의의 변수 위치로 옮겨놓는 Jump-up transition, 그리고 임의의 한 변수를 뒤쪽의 임의의 변수 위치로 옮겨놓는 Jump-down transition 방법이 있다[12, 17]. 이 방법 모두 두 인접 변수를 교환하는 스와핑 방법을 여러번 실행하여 구현한다. 그러므로 μO의 초기화 단계와 Faster-μO의 시프팅 단계의 스와핑 횟수를 비교하면 어느 방법이 더 빠르게 실행되는지 알 수 있다. 먼저 시프팅 방법의 스와핑 횟수는 변수의 개수가 n개 일 때, 수식 (2)와 같다.

$$\text{Sifting\_loop} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \qquad (2)$$

초기화 단계의 임의의 해 제안 방법은 Exchange transition을 40%, Jump-up/down transition을 60% 실행하고, 초기화 반복횟수를 변수 개수의 1.5배로 하여 실행하면 비교적 좋은 해를 구한다[12, 17]. 초기화 반복 횟수가 크면 탐색 횟수가 증가하여 더 좋은 해를 찾을 수도 있지만 실행시간이 너무 오래 걸린다. 그러므로 이러한 값으로 지정하였을 때 스와핑 횟수는 다음 수식 (3)과 같다.

$$\begin{aligned} Init-loop &= 15n\left\{0.4\left(2\left(\frac{n-1}{2}\right)-1\right)+0.6\left(2\left(\frac{n-1}{2}\right)\right)\right\} \\ &= 1.5n(0.7n-1.1) = 1.05n^2 - 1.65n \end{aligned} \qquad (3)$$

이와 같이 스와핑 횟수를 비교해보면 초기화 단계가 시프팅 단계보다 훨씬 많은 스와핑을 실행하는 것을 알 수 있다. 그러므로 시프팅 단계가 초기화 단계보다 실행시간이 더 빠름을 알 수 있다. 또한 결과 값은 Rudell에 의해 시프팅 방법이 임의의 국부 탐색 방법보다 더 나은 해를 구한 다는 것은 증명되었다[8]. 이로써 초기화 단계를 시프팅 단계로 대체하는 것이 실행 시간과 BDD 크기 면에서 더 나은 알고리즘인 것으로 예상된다.

V. 실험결과

1. 파라메타 분석

제안된 Faster-μO 알고리즘은 3가지의 파라메타가 존재한다. 첫째, 샘플링 단계와 시프팅 단계의 사이클을 몇 번 반복 실행하여야 하는지 결정하는 외부 반복횟수이다. 외부 반복 횟수는 초기 BDD 크기와 비례한다. 변수 개수에 비해 BDD 크기가 크다는 것은 함수가 더욱 복잡하다는 것을 의미한다. 그러므로 복잡한 함수일수록 BDD 크기의 범위가 더욱 넓으므로 더 많은 반복 실행이 필요하다. 외부 반복 횟수 n을 구하는 수식은 아래 수식 (4)와 같다. 초기 BDD 크기에 반복 상수(A) 0.88, 0.90, 0.92, 0.95를 n번 곱하여 *loop_func*이 처음으로 1 보다 작아질 n을 구해 반복 횟수로 결정한다. 비교적 반복 횟수가 클수록 최적에 가까운 근사 값을 구한다.

$$loop\_func = size * \sum_{i=1}^{n} A$$

A : 반복상수(0.88, 0.90, 0.92, 0.95) (4)

두 번째 파라메타는 샘플링 단계를 몇 번 반복 실행하여야 하는지 결정하는 샘플링 반복횟수이다. 이 샘플링 단계는 이전 시프팅 단계의 결과인 국부 최소 값에서 벗어나기 위한 단계이므로 많은 반복은 필요하지 않다. 또한 반복 횟수를 크게 증가 시켜도 BDD 크기에는 크게 영향을 주지 않는다. 그러므로 초기 BDD 크기와 비례하여 10 ~ 50 반복 횟수 안에서 실행한다.

세 번째 파라메타는 BDD 크기에 가장 큰 영향을 주는 데몬이다. 일반적으로 데몬은 시스템 에너지의 작은 일부 값으로 지정한다. 데몬이 작다고 항상 실행 시간이 단축되는 것이 아니며, 데몬이 크다고 항상 좋은 해를 찾는 것도 아니다. 각 회로마다 특성에 맞는 데몬을 결정하면 더욱 좋은 해를 구할 수 있으나 본 실험에서는 모든 회로에 동일한 계산식을 이용하여 데몬을 결정하였다. 본 실험에서는 데몬 값을 초기 BDD 크기에 데몬 상수(B) 0.1, 0.2, 0.3, 0.4, 0.5를 곱하여 결정하였다.

표 1은 BDD 크기측면에서 외부 반복 상수(A)와 데몬 상수(B)를 비교한 것이다. 각각의 값은 데몬 상수와 반복 상수에 대해 표 3의 각 회로의 크기를 구해 그 합을 구한 것이다. 실험 결과에서 알 수 있듯이 반복 상수(A)는 그 값이 클수록 더 좋은 결과를 찾아내지만 데몬 상수(B)는 그 값이 크다고 항상 좋은 결과를 찾는 것은 아니다. 그리고 BDD 크기에 더욱 영향을 미치는 상수는 반복 상수(A) 보다 데몬 상수(B)임을 알 수 있다. 이로서 데몬의 크기가 BDD 크기에 가장 큰 영향을 주는 것을 알 수 있다. BDD 크기 최소화 문제에서는 데몬의 크기가 초기 BDD 크기의 30%일 때 그 결과가 가장 좋음을 알 수 있다.

표 1. BDD 크기의 A, B 비교
Table 1. A, B comparison of BDD size.

데몬상수B / 반복상수A	0.1	0.2	0.3	0.4	0.5
0.88	68,784	68,460	68,319	68,363	68,425
0.90	68,783	68,459	68,138	68,174	68,300
0.92	68,653	68,460	68,090	68,137	68,300
0.95	68,591	68,459	68,088	68,133	68,173

표 2는 실행 시간측면에서 외부 반복 상수(A)와 데몬 상수(B)를 비교한 것이다. 각각의 값은 데몬 상수와 반복상수에 대해 표 3의 각 회로의 실행 시간을 구해 그 합을 구한 것이다. 실험 결과에서 알 수 있듯이 반복 상수(A)가 커질수록 그 실행시간은 커지며, 데몬 상수(B)는 그 값이 크다고 항상 실행 시간이 오래 걸리는 것은 아니다. 데몬이 너무 클 경우는 국부 최소값을 벗어날 때 너무 나쁜 값으로 되돌아가 근사 최적값에 도달하지 못하고 실험이 조기 종료 될 수 있기 때문이다. 또한 반복 상수가 너무 크면 BDD 크기에는 크게 영향을 미치지 않지만, 실행 시간을 너무 오래 걸리게 한다. 그러므로 항상 반복 상수를 크게만 지정하는 것은 좋지 못하다. 실험결과 그 값이 0.95 이상이 되면 표 2에서와 같이 실행시간이 급격히 증가한다. 그러나 크기가 좋아지는 비율은 아주 적으므로 반복 상수 값은 0.92로 지정하는 것이 적당하다.

표 2. 실행시간의 A, B 비교
Table 2. A, B comparison of runtime.

데몬상수B / 반복상수A	0.1	0.2	0.3	0.4	0.5
0.88	5,072.42	4,410.48	4,940.42	7,011.88	5,700.23
0.90	5,405.93	5,028.84	7,390.74	9,162.04	6,739.66
0.92	6,531.27	5,866.63	8,484.75	8,937.71	6,918.19
0.95	9,007.65	10,477.84	11,158.44	11,695.36	17,688.05

2. BDD 크기와 실행시간

제안된 Faster-μO 알고리즘의 구현은 C언어를 사용하였으며, 콜로라도 주립대학의 CUDD 패키지[17]를 이용하였다. 제안된 알고리즘은 IWLS91 벤치마크[18] 회로에 적용되었으며 실험에 사용된 기종은 256MB 메모리를 갖은 248 MHz의 2*SUN SPARC-II 이다.

표 3은 제안된 Faster-μO 알고리즘의 실험결과이다. circuit name열은 실험 회로의 이름을 나타내며, in과 out 열은 회로의 입출력 단자의 수를 각각 나타낸다. 수행된 알고리즘은 근사 최적 변수 순서를 찾아내는 SA 알고리즘[12], 유전자 알고리즘[9], μO 알고리즘[19]과 본 논문에서 제안하는 Faster-μO 알고리즘이 수행되었다. 실행된 모든 알고리즘은 CUDD 패키지를 이용하였으며, 유전자 및 SA 알고리즘의 파라메타는 CUDD에서 최적화된 값으로 실행하였다. Faster-μO의 외부

반복횟수의 상수 값은 0.92이며, 샘플링 반복 횟수는 회로의 초기 크기에 따라 10~50으로 지정하고, 데몬의 상수 값은 0.3이다. 실험결과로 볼 때 Faster-μO 알고리즘은 유전자와 SA 알고리즘, 그리고 μO 알고리즘과 비교하여 BDD 크기와 실행 시간에서 성능 향상을 보였다.

표 3. Faster-μO 알고리즘 실험결과
Table 3. Faster-μO algorithm experimental result.

circuit			SA		Genetic		μO		Faster-μO	
Name	In	Out	Node	Time	Node	Time	Node	Time	Node	Time
C1355	41	32	2586	9089.79	25866	24792.05	2586	3959.29	25866	3741.85
C3540	5(	22	2383	6385.48	23828	8981.15	2382	3393.04	23828	2435.90
C1908	3:	25	565	874.68	5832	1843.09	583	907.85	5526	1065.65
C432	3(	7	120	96.22	1095	452.40	118	49.82	1164	40.11
C880	6(	26	405	1674.86	4065	3344.00	405	893.79	4053	303.95
apex6	13:	99	55	113.96	511	168.93	51	128.47	500	55.89
cm85a	11	3	31	1.76	31	1.12	2	0.71	28	0.71
des	25	245	303	665.90	2969	1220.61	295	515.71	2947	506.30
i8	13:	81	130	215.78	1277	316.48	127	220.83	1277	107.81
k2	4:	45	126	133.73	1256	170.10	125	103.33	1259	74.67
mux	21	1	3	11.28	33	16.04	3	7.30	33	7.52
my_adder	3:	17	8	23.34	82	53.77	8	12.45	82	11.09
too_large	3:	3	31	66.62	306	130.71	30	48.25	304	41.61
vda	17	39	481	15.53	478	12.15	47	18.48	478	12.19
x1	51	35	40	72.67	423	100.87	40	83.04	409	44.33
x4	94	71	36	62.63	340	97.39	341	62.32	336	35.17
Total			68,48	19,504.23	68,392	41,700.86	68,43	12,404.68	68,090	8,484.75

먼저 C1355, C3540, C1908, C432, C880 회로는 비교적 규모가 크고 복잡한 회로로써, 그중 C432 회로는 크기에서 유전자 알고리즘이 가장 좋은 결과를 보인다. 그러나 Faster-μO의 크기는 μO의 크기보다 24개 감소하였으며 실행시간도 19.50% 감소되었다. 나머지 회로들은 Faster-μO가 그 크기 면에서 유전자, SA 그리고 μO와 동일하거나 더 향상되었다. 실행시간 면에서는 μO 알고리즘보다도 크게 감소하였으며, 그 중 C3540과 C880 회로는 실행 시간이 각각 54.84%, 66.00% 가 향상되었다. 이로써 규모가 크고 복잡한 회로일 때 Faster-μO 알고리즘의 실행시간이 더욱 감소된다는 것을 알 수 있다.

비교적 규모가 작고, 덜 복잡한 회로는 apex6, cm85a, i8, k2, mux, my_adder, too_large, vda, x1, x4, des 회로이며, 그중 k2, too_large 회로는 μO 알고리즘이 가장 좋은 결과 값을 구하고, apex6, des, x4 회로는 Faster-μO가 가장 좋은 값을 구한다. 또한 실행시간은 Faster-μO가 전체적으로 모두 감소되었으며 그 중 apex6, i8, x4 회로는 실행 시간이 각각 56.50%, 51.18%, 43.57%가 향상되었다. 이로써 규모가 작은 회로에도 Faster-μO 알고리즘의 적용이 효율적인 것을 알 수 있다.

전체적인 BDD의 크기는 Faster-μO 알고리즘을 SA 알고리즘과 비교하면, BDD 크기 면에서 392 감소하였으며, 실행시간은 56.40% 감소되었다. 유전자 알고리즘과 비교하면, BDD 크기 면에서 302 감소하였으며, 실행시간은 79.65% 감소되었다. μO 알고리즘과 비교하면 BDD 크기 면에서 348 감소하였으며, 실행시간은 31.60% 감소하였다. 이와 같은 결과로 제안된 Faster-μO 알고리즘은 기존의 μO 알고리즘보다 BDD 크기 성능은 비교적 비슷하거나 약간 향상되었으며, 실행 시간 성능 면에서 큰 향상을 보여 효율적인 알고리즘이라는 것을 보여준다.

VI. 결 론

BDD는 회로 합성과 형식 검증에 널리 사용되는 회로 표현 기법으로서 BDD 크기의 최소화 문제는 매우 중요한 문제이다. 본 논문에서는 μO 알고리즘을 BDD 최소화 문제에 더욱 적합하도록 보완한 Faster-μO 알고리즘을 제안하고, IWLS91 벤치마크 데이터 실험을 통해 그 효율성이 검증되었다.

BDD 크기 최소화를 위해 많은 알고리즘들이 제안되었다. 가장 최근에 제안된 μO 알고리즘은 기존의 휴리스틱 알고리즘인 유전자 알고리즘과 SA 알고리즘이 BDD 크기 면에서는 좋은 성능을 보이지만, 실행 시간이 매우 오래 걸리는 단점을 해소시킨 알고리즘이다. 본 논문에서 제안한 Faster-μO 알고리즘은 기존의 μO 알고리즘의 실행 시간을 더욱 단축시키기 위한 알고리즘이다. 실험결과 Faster-μO 알고리즘이 μO 알고리즘보다 BDD 크기와 실행 시간 면에서 크게 향상된 성능을 나타내고 있음을 보여준다.

참 고 문 헌

[1] S. B. Akers, "Binary Decision Diagrams," IEEE Transaction on Computers, vol. C-27, No. 6, pp. 509-516, August 1978.

[2] Randal E. Bryant "Symbolic Manipulation of Boolean Functions Using a Graphical

Representation," 22nd Design Automation Conference, pp. 688-694, June 1985.

[3] Randal E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transaction on Computers, vol. C-35, pp. 667-691, August 1986.

[4] Randal E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," ACM Computing Surveys, vol. 24, no. 3, September 1992.

[5] Randal E. Bryant, "Binary Decision Diagrams and Beyond : Enabling Technologies for Formal Verification," International Conference on Computer Aided Design, pp. 236-243, November 1995.

[6] Rolf Drechsler, Bernd Becker, "Binary Decision Diagrams : Theory and Implementation," Kluwer Academic Publishers, 1998.

[7] S.J. Fredman, K.J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," IEEE Transaction on Computers, vol. C-39, no. 5, pp. 710-713, May 1990.

[8] Richard Rudell, "Dynamic Variable Ordering of Ordered Binary Decision Diagrams," International Conference on Computer-Aided Design, pp. 43-47, November 1993.

[9] Rolf Drechsler, Bernd Beckern, Nicole Gockel, "A Genetic Algorithm for Variable Ordering of OBDDs," International Workshop on Logic Synthesis, pp. 5C:5.55-5.64, 1995.

[10] Rolf Drechsler, Nicole Gockel, "Simulation Based Minimization of large OBDDs," Preprint at the Institute of Computer Science, Albert-Ludwigs-University Freiburg im Breisgau, germany, January 1997.

[11] Rolf Drechsler, Nicole Gockel, "Minimization of BDDs by Evolutionary Algorithms," International Workshop on Logic Synthesis(IWLS'97), Lake Tahoe, 1997.

[12] B. Bolling, M. Lobbing, I. Wegener, "Simulated Annealing to improve variable orderings for OBDDs," International Workshop on Logic Synthesis, pp. 5b:5.1-5.10, 1995.

[13] M. Creutz, "Microcanonical Monte Carlo simulation," Phys, Rev, Lett, vol. 50, pp. 1411-1413, 1983.

[14] A. Linhares, J. A. Torreao, "Microcanonical optimization applied to the traveling salesman problem," Int. J. Modern Phys. C, vol. 9, pp. 133-146, 1998.

[15] Alexandre Linhares, Horacio H. Yanasse, Jose R. A. Torreao, "Linear Gate Assignment : A Fast Statistical Mechanics Approach," IEEE Transaction on Computer-Aided Design of Integrated circuits and system, vol. 18, no. 12. December 1999.

[16] S.C S. Porto, A. M. Barrose, J.R.A. Torreao, "A Parallel Microcanonical Optimization Algorithm for the Task Scheduling Problem," Technical Report, Applied Computing & Automation Universidado Federal Fluminense, September 1999.

[17] Fabio Somenzi, "CUDD : CU Decision Diagram Package Release 2.3.0," University of Colorado, April 12, 1997.

[18] Saeyang Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Distributed as part of the IWLS91 benchmark distribution, January 15, 1991.

[19] 이민나, 조상영, "Microcanonical Optimization을 이용한 BDD의 최소화 기법," 한국정보처리학회, 제8-A권, 제1호, 48-55쪽, 2001년 3월

저 자 소 개

李 珉 那(正會員)

1987년 : 성신여자대학교 수학과 졸업(학사). 1991년 : 한국외국어대학교 경영정보대학원 응용전산학과(이학석사). 2000년 : 한국외국어대학교 대학원 컴퓨터공학과 박사과정 수료. 1991년~1995년 : 내무부 전산지도과 근무. <관심분야> 컴퓨터아키텍쳐, 병렬처리, CAD

曺 常 榮(正會員)

1988년 : 서울대학교 제어계측학과(공학사). 1990년 : 한국과학기술원 전기 및 전자공학과(공학석사). 1994년 : 한국과학기술원 전기 및 전자공학과(공학박사). 1994~1995 : KAIST 정보전자 연구소 연구원. 1995~1996 : UC, Irvine Post Doc. 1996~1997 : 삼성전자 선임연구원. 1997~현재 : 한국외국어대학교 컴퓨터공학과 조교수. <관심분야> 컴퓨터 아키텍쳐, 내장형 실시간 시스템