About Binary Decision Diagram

BDD(Binary Decision Diagram)은 Boolean expression 을 Tree 형식으로 표현하는 데이터 구조입니다. Pre-decide 된 Variable 의 order 에 따라 각 variable 이 0 일 때를 low, 1 일 때를 high 로 두고 트리를 전개합니다. 마지막 트리 노드(leaf node)에서는 각 variable 이 해당 값을 가질 때, output 이 어떤 값을 가지는지 표현합니다. BDD 의 예시는 다음과 같습니다.

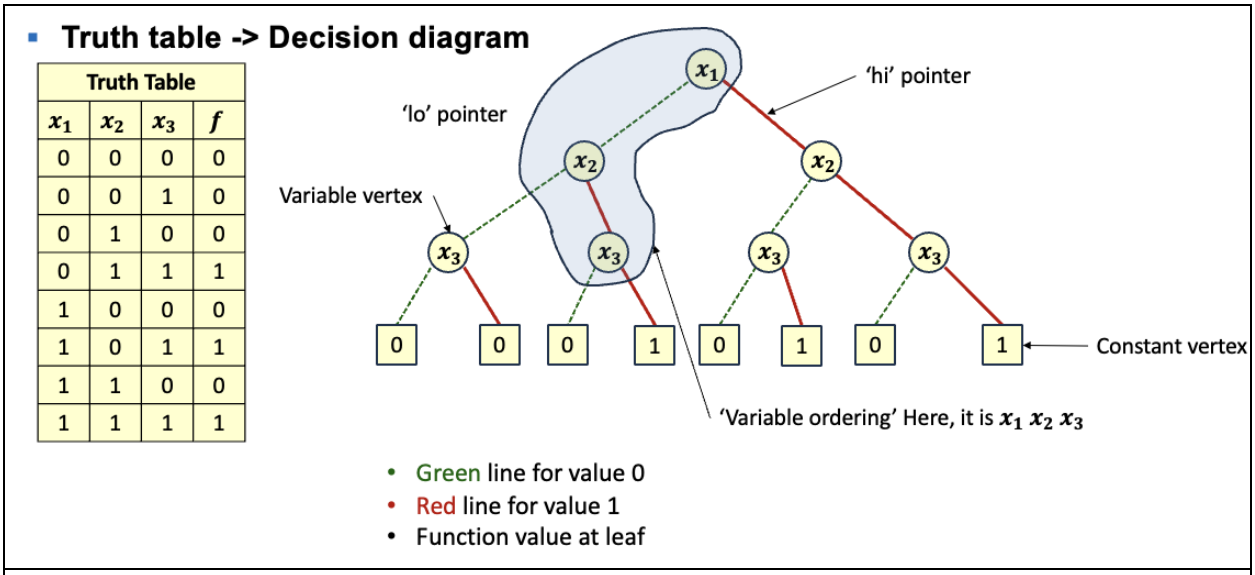

Fig.1 Truth Table to BDD

이때, 이 문제에서는 runtime 은 고려하지 않고, BDD 의 노드의 개수로만 평가하고자 합니다. BDD 의 노드의 개수에 영향을 미치는 알고리즘 및 요소는 다음과 같습니다.

(1) Reduction Rules:

노드로 인해 분기가 되었지만, 분기된 결과가 동일하다면 해당 변수는 표현할 필요가 없습니다. 위의 예시에서는 다음과 같이 생각할 수 있습니다.

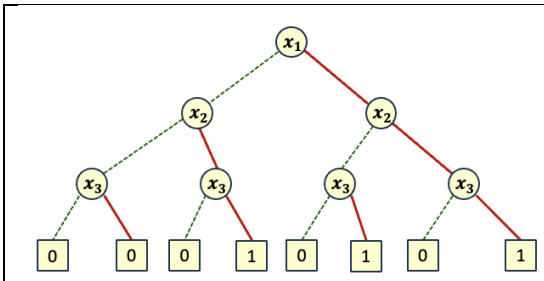

Fig.2 Initial BDD

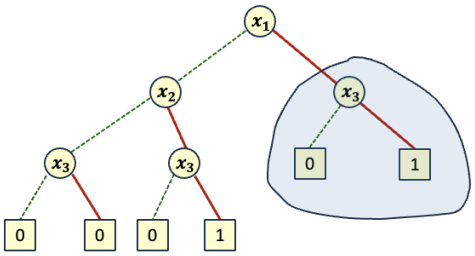

Fig.3 Reduced BDD

X1 의 high-child node(오른쪽으로 분기된 부분)에서, x2 로 인해 분기되는 두 갈래는 아래에서 결과값이 동일합니다. 따라서 x2 로 분기할 필요가 없습니다. X2 를 제거한 그림은 Fig.3 와 같습니다.

1) Reduction 을 진행하는 방법 중 하나는 다음과 같습니다. Constant leaf node 를 하나만 남깁니다.

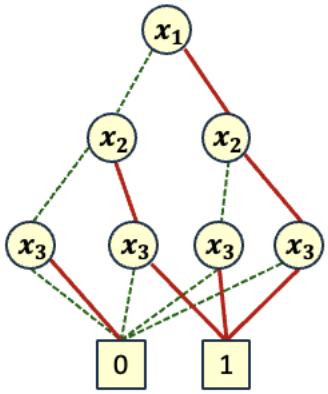

Fig4. Keep one copy of each constant leaf

2) 같은 노드를 병합합니다. 아래의 그림처럼 동일한 매핑이 되는 노드를 합치면 됩니다.

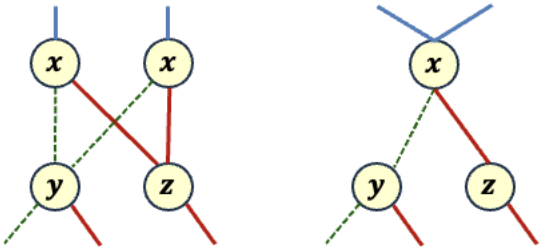

Fig5. Merge Isomorphic nodes

3) 이후 같은 edge 가 있으면 제거합니다.

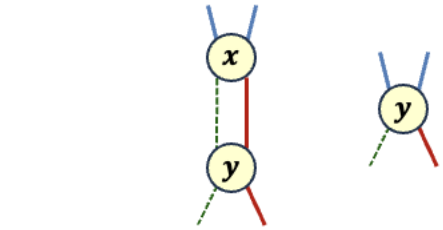

Fig 6. Eliminate Redundant Test

(2) Variable Ordering:

Variable(변수)가 어떤 순서로 전개되는지도 트리의 크기에 영향을 미칩니다. Reduction 여부 및 규모가 달라지기 때문입니다. 아래의 그림과 같이 $\boldsymbol{a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3}$에 대해 서로 다른 ordering 에 기인한 Tree 의 크기 차이가 발생합니다.

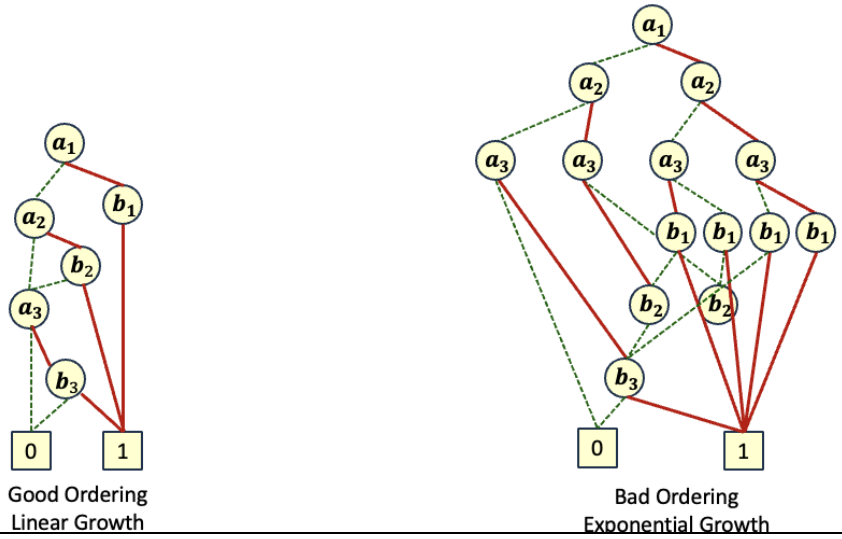

Fig. 7 Variable Ordering Matters

(3) Other Reduction Algorithms:

그 외에도 여러가지 Reduction Algorithms 가 존재합니다. Don't Care Condition, Dynamic Reordering, Function Composition 등의 방법이 존재하며, 이 방법을 활용하셔도 됩니다. 단, BDD 의 정확한 표현을 잃거나, 표현력을 낮추는 방법은 사용하시면 안됩니다.

(4) Sharing in BDDs:

두 개 이상의 output 에 대해서 각각 BDD 를 만들 때, 겹치는 노드와 엣지에 대해서 BDD 를 공유할 수 있습니다. 이를 통해 BDD 가 차지하는 memory 의 size 를 감소할 수 있습니다. 아래와 같은 예시를 생각할 수 있습니다.

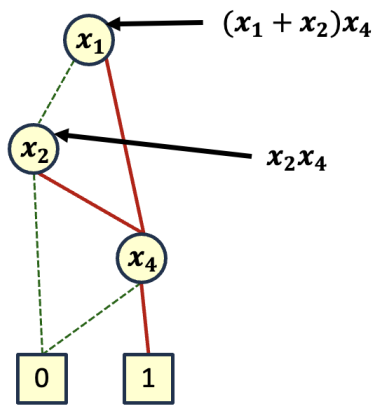

Fig. 8 Sharing in BDDs

Problem

1. 4-bit full adder Truth Table [3pt]

4-bit full adder 에 대해 C_{out}과 S_0, S_1, S_2, S_3의 truth table 을 아래와 같은 형식의 텍스트 파일(.txt)로 작성하시오. [제출 파일명: 'Carry.txt', 'S0.txt', 'S1.txt', 'S2.txt', 'S3.txt']

(파이썬 또는 C 언어 코드를 이용하셔도, 다른 방법을 이용하셔도 무관합니다.)

Fig.1 txt format

```
X0 x1 x2 x3 y
0 0 0 0
0 0 1 1
0 1 0 1
0 1 1 1
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 0
```

2. 4-bit full adder BDD & Visualization [5pt]

작성한 텍스트 파일을 읽어와 각각의 output 에 대해 BDD 를 구현하고, 시각화하시오. 구현 및 시각화하는 C 언어 또는 파이썬 코드와, 시각화 결과 이미지 파일을 제출하시면 됩니다. [제출 파일명 : 'pb2.c' or 'pb2.py', 'BDD.jpeg']

3. General BDD& Visualization [10pt]

텍스트 파일에 포함된 변수의 개수와, 회로의 종류에 무관하게, Truth Table 이 주어지면 BDD 를 만들고 시각화하는 코드를 만드시오. 채점은 4-bit full adder 가 아닌 다른 회로의 Truth Table 에 대해 채점이 이루어질 예정입니다. [input text file name : 'input.txt'][제출 파일명 : 'pb3.c' or 'pb3.p']

4. Reduction of 4-bit full adder BDD [20pt]

4-bit full adder 에 대해 *2.4-bit full adder BDD& Visualization* 에서 만든 BDD 를 Reduction 하는 알고리즘을 작성해주세요. 채점은 4-bit full adder 에 대해

이루어지고, Reduction 의 QoR(Quality of Result)는 node 의 개수로 평가될 예정입니다. 따라서 Node 의 개수를 count 하여 출력하는 알고리즘도 포함되어야 합니다. [제출 파일명 : 'pb4.c' or 'pb4.p', 'BDD_Reduction.jpeg']
(단, sharing in BDD 를 사용할 시, 겹치는 노드는 한 개로 카운트합니다.)

5. Reduction of General BDD [20pt]

텍스트 파일에 포함된 변수의 개수와, 회로의 종류에 무관하게, *3. General BDD& Visualization* 에서 만든 BDD 를 Reduction 하는 알고리즘을 작성해주세요.
채점은 4-bit full adder 가 아닌 회로에 대해 이루어지고, Reduction 의 QoR(Quality of Result)는 node 의 개수로 평가될 예정입니다. 따라서 Node 의 개수를 count 하여 출력하는 알고리즘도 포함되어야 합니다. [제출 파일명 : 'pb5.c' or 'pb5.p']
(단, sharing in BDD 를 사용할 시, 겹치는 노드는 한 개로 카운트합니다.)