DNF 를 이용한 SAT 의 효율적 적용

남명진° , 최진영
고려대학교 컴퓨터학과
{mjnam°, choi}@formal.korea.ac.kr

Efficient Application to SAT Using DNF

Myoung-Jin Nam° , Jin-Young Choi
Dept. of Computer Science and Engineering, Korea University

요 약

하드웨어 검증과 모델 체킹 등의 분야에서, SAT(satisfiability problem)나 항진 명제 검사(tautology checking)는 매우 중요한 문제이다. 그러나 이들은 모두 NP-complete 문제이므로 그 복잡도가 매우 크다. 이를 해결하기 위해 여러 가지 연구가 이루어져 왔으며, 여러 효율적인 알고리즘이 존재한다. 이러한 알고리즘은 대부분 일반 표현식을 CNF(conjunctive normal form)로 바꾸어 입력 형식으로 사용한다. 이 논문에서는 일반 표현식을 입력으로 받아 DNF로 변환한 뒤 DNF의 특성을 이용하여 SAT를 검사하는 효율적인 방법을 제시한다.

1. 서 론

명제 논리(propositional logic)의 이진 표현식(boolean expression)은, 그 속의 각 변수가 참과 거짓 중 어느 값을 가지느냐에 따라 표현식의 true/false 값이 결정된다. 변수가 n 개 일 때 가능한 값의 조합은 2^n 개인데, 이 중에 주어진 표현식을 " true" 로 만드는 경우가 존재하는지 여부를 알아보는 것을 satisfiability problem(SAT) 이라고 한다.

SAT 문제는 어떠한 전자 회로가 언제나 주어진 속성을 만족하는지를 알아보는 하드웨어 검증[1] 등에 사용되며, 또한 어떤 모델이 특정 속성을 만족하는지를 검사하는 모델 체킹[2] 등에도 사용된다. 이와 같은 방식으로 하드웨어나 소프트웨어 시스템의 정확성(correctness)을 확인하여 보다 안정된 시스템을 구현하는데 도움이 된다.SAT 문제는 모두 전산학에서 매우 중요한 문제들인데, 이들은 모두 NP-Complete 문제이다. 따라서 최악의 경우에 지수 증가 복잡도(exponential complexity)를 가진다. 그러나 항상 최악의 경우만 존재하는 것은 아니므로, 평균적으로 또는 특정한 경우에 DPLL, DP[3] 등의 효율적인 여러 알고리즘이 존재한다. 이러한 알고리즘들은 이진 표현식을 입력으로 받아서, 이 표현식이 satisfiable 인지를 검사하여, yes/no 로 출력해준다.

이 알고리즘들은 대부분은 CNF(conjunctive normal form)[4][5]를 입력 형식으로 받는다. 특별한 형식을 지니지 않은 일반 표현식보다 형식을 가진 입력 형태가 SAT 임을 검사하는데 더 효율적이다. 이 논문에서는 일반 표현식을 DNF 로 변환하여 SAT 를 검사하는 방법을 제시한다.

2. Satisfiability Problem(SAT)

SAT 는 satisfiability problem 의 줄임말이다. 어떠한 표현식이 있을 때, 그 표현식을 " true" 가 되게 하는 값의 조합이 존재하면 그 표현식은 " satisfiable" 이라고 얘기하고, 없으면 " unsatisfiable" 이라고 얘기한다. 그리고 " true" 가 되도록 하는 변수값의 조합은 모델(*model*)이라고 한다. 예를 들어 (p + q)란 표현식이 있을 때, 변수값 조합은 4 개가 있다.(진리표 이용) 그 중 p = false 그리고 q = false 인 경우를 제외한 나머지 3 가지 경우는 표현식 전체값이 " true" 이다. 만약, 표현식을 " true" 로 만들어주는 변수값 조합의 경우의 수가 한가지 밖에 없더라도 " satisfiable" 이다. 이러한 변수값 조합의 경우의 수가 없으면 " unsatisfiable" 이다. 그런데, 특별한 형식이 없는 일반 표현식은 SAT 임을 알아내기 힘들다. 그래서 일반 표현식은 특별한 형태로 바꾸어서 SAT 임을 증명하는데, 보통 CNF 를 많이 사용한다.

CNF 는 다음과 같은 형식을 가진다.

X = A1 & A2 & A3 & A4 ···
A1 = (a1 + a2 + a3 +···)
A2 = (b2 + b2 + b3 +···)
···

A1, A2..An 을 *clause* 라고 하고, clause 내의 a1, a2, b1,b2..를 *literal* 이라고 한다. 여기서 literal 은 positive/negative 형 모두를 가리킨다. 여기서 X 가 " true" 값을 가지려면 모든 clause 들이 " true" 값을 가져야 하고, 각 clause 내의 적어도 하나의 literal 이 " true" 값을 가져야 한다. 대부분의 SAT 알고리즘들은 이러한 CNF 의 특성을 이용한다.

3. 효율적 DNF 변환 알고리즘

대부분의 SAT 알고리즘과 Solver 들이 CNF 을 입력 형식으로 사용하는데 반해, 이 논문에서는 일반 표현식을 입력받아 DNF 로 변환한 뒤 DNF 의 특성을 이용하여 SAT 를 검사하는 방식을 사용한다. 이때, 일반 표현식을 equivalent 한 DNF 로 변환하지 않고, " triple" 을 이용하여 새로운 DNF 를 생성하는 방식을 제시한다. 원래의 표현식의 모델들은 변환된 DNF 의 모델들로 확장될 수 있다. 원래의 표현식이 SAT 이면 변환된 DNF 도 SAT 이다. 여기서 주의할 것은 원래의 표현식이 항진 명제이면, 변환된 DNF 는 단지 SAT 일수도 있다. 만약 이 변환 방법을 이용해서 항진 명제임을 증명하려면, 표현식의 negation 이 모순(contradiction)임을 밝히면 된다.

임의의 표현식에 대해서 이 표현식이 변수 하나로만 이루어진 표현식이 아닐 때 이 표현식을 새로운 변수 A 로 치환한 뒤, 그림[1]과 같은 방식으로 DNF 를 만든다.

$$A \wedge \bigwedge_{\varphi \in Sub(A),\ \varphi \equiv \varphi 1 \oplus \varphi 2} (P_{\varphi} \Leftrightarrow (P_{\varphi 1} \oplus P_{\varphi 2})) \wedge \bigwedge_{\varphi \in Sub(A),\ \varphi \equiv \neg \varphi 1} (P_{\varphi} \Leftrightarrow P_{\neg \varphi 1})$$

[그림 1] 변환

Sub(A) 는 A 의 하위 표현식(subformula)들의 집합이고, ⊕ 은 논리연산자이다. 예를 들어, (p & q) + r 인 표현식이 있다고 하자. 우선, 각 하위 표현식을 가리키는 다른 변수를 두어 치환한다. 하위 표현식을 한 변수로 치환한 식을 " triple" 이라고 한다.

A <-> p & q
B <-> A + r

그 후에 전체 표현식을 치환한 변수(여기서는 B) 와 생성된 triple 들을 and-연산자로 묶는다.
B & (A <-> p & q) & (B <-> A + r)

위와 같은 형태가 나오면, 우선 equivalence-연산자를 implication 의 형태로 바꾼다.

B & (A -> (p&q)) & ((p&q) -> A) & (B -> (A+r)) & ((A+r) -> B)

Implication 은 negation 과 or-연산자 형태를 이용하여 바꿀 수 있다.

(A -> p & r) & ((p & q) -> A) 는
(¬ A +(p & r)) & (¬(p & q) + A) 로,
(B -> (A+r)) & ((A+r) -> B) 는
(¬ B +(A +r)) & (¬(A +r)+B)로 변환된다.

드모르간 법칙을 이용하여 negation 을 없앤다. 즉 NNF(implication 과 negation 이 존재하지 않는 형식)형으로 만든다. 그리고 마지막 형태인 DNF 로 변환한다.

4. DNF 의 SAT 검사 알고리즘

DNF 는 각 clause 가 or-연산자로 연결된 표현식 형태이다. 그리고 clause 내에 literal 들은 and-연산자 로 연결되어 있다.

X = A1 + A2 + A3 +A4 ···
A1 = a1 & a2 & a3 &···
A2 = b2 & b2 & b3 &···
···

X 의 값이 " true" 가 되기 위해서는 (satisfiable), clause 들이 or-연산자로 연결되어 있으므로, 적어도 하나의 clause 가 " true" 값을 가져야 한다. 위에서 A1 을 " true" 라고 하자. A1 내에 각 literal 들은 and-연산자로 연결되어 있다. A1 이 " true" 값을 가지려면 A1 내의 모든 literal 들은 " true" 값을 가져야 한다. 즉, DNF 형의 표현식이 SAT 인지를 검사하려면 clause 의 모든 literal 에 " true" 값을 할당할 수 있는지를 검사하면 된다.

먼저 모든 clause 를 방문하여 unit clause(내부에 하나의 literal 만 존재하는 clause) 가 존재하는지를 검사한다. unit clause 가 존재한다면, 그 표현식은 무조건 SAT 이다. 예를 들어, x + (¬ x & y & z) + (¬ y + ¬ z) 이 있을 때, 3 개의 clause 중에 하나의 clause 만 " true" 가 되어도 그 표현식은 SAT 인데 이 표현식은 x(unit clause)에 바로 " true" 값을 할당할 수 있다.

모든 clause 를 방문하여 unit clause 가 존재하지 않는다면 다음은 clause size 가 2 인 clause(literal 이 2 개인 clause)를 방문하여 " true" 값을 할당할 수 있는지 여부를 검사한다. Clause 내의 모든 literal 에 " true" 값을 할당할 수 없는 경우는 단 한가지 경우 밖에 없다. 어떤 변수에 대해서 positive/negative 가 한 clause 안에 같이 존재하는 경우이다. (x & ¬ x)일 경우는 그 변수에 true 또는 false 의 어떤 값을 할당해도 clause 값이 " true" 가 될 수 없다. 만약 한 변수의 positive/negative literal 이 한 clause 에 존재하지

않는다면 각 literal 을 " true" 값으로 할당할 수 있으므로 전체 표현식은 SAT 임을 알 수 있다.

size 가 3 미만(1 또는 2)의 clause 가 존재하지 않으면 size 가 3 이상인 clause 내의 literal 을 서로 비교하여 각 literal 에 " true" 값을 할당할 수 있는지 여부를 검사해야 한다. 모든 literal 에 " true" 를 할당할 수 없는 경우는 단 한가지 경우 밖에 없다. 어떤 변수에 대해서 positive/negative 가 한 clause 안에 같이 존재하는 경우이다. 따라서, DNF 가 주어졌을 때 SAT/UNSAT 여부를 알기 위해서는 한 clause 내의 변수에 대해서 positive/negative 형이 둘 다 존재하는지를 검사하여, 만약 한 clause 내에 동시에 존재하지 않으면 전체 표현식은 SAT 이다. 우선 한 clause 내에 임의의 literal x 를 선택한다. 그리고 그 clause 내의 나머지 모든 literal 과 각각 비교하여, ㄱ x 이 존재하는지 검사한다.

a) 선택한 literal x 에 대해서 " ㄱ x" 가 존재하지 않으면 선택한 literal x 에 " true" 값을 할당한다. " true" 값이 할당되면 그 clause 내에서 더 이상 비교할 필요가 없으므로 그 literal 은 삭제한다. 그리고 다른 literal 을 선택하여 같은 검사를 한다.

b) 만약 " ㄱ x" 가 존재하면 x 에 어떠한 값을 할당해도 이 clause 는 " true" 값을 가질 수 없으므로 다음 clause 를 방문한다. 더 이상 방문할 clause 가 없으면 그 표현식은 UNSAT 이다.

CNF 를 이용한 SAT 검사 방법은 모든 clause 가 " true" 값을 가져야 하므로 literal 의 값에 따라 clause 상태가 달라진다. 반면 DNF 를 이용한 방법은 한 clause 내의 각 literal 에 " true" 값을 할 수 있는지 여부만 검사한다. 그리고 CNF 에서 unit clause 가 존재할 때 DNF 와 마찬가지로 unit clause 에 " true" 값을 할당한다. 하지만 CNF 에서는 unit clause 가 " true" 값이 되어도 전체 표현식이 SAT 라고 할 수 없다. 그러나 DNF 인 경우에는 unit clause 가 존재하고, 그 clause 에 " true" 값을 할당하면 전체 표현식이 SAT 이다. 이렇게 DNF 를 이용한 방법은 전체 clause 의 상태를 검사하지 않고 한 clause 내에서 literal 들의 관계를 보기 때문에 더 간단하다. Clause size 가 작을수록 SAT 검사는 더욱 간단해진다.

5. 결론 및 향후 연구 방향

SAT 는 하드웨어/소프트웨어 검사에 광범위하게 사용되고, NP-complete 문제(problem)로 효율적으로 풀기 위한 연구가 활발히 진행되고 있다. SAT 를 풀기 위한 많은 알고리즘들이 존재하는데 대부분 CNF 를 이용하여 문제를 해결하고 있지만, 이 논문은 DNF 를 이용하여 DNF 의 특성을 이용하여 SAT 를 검사하는 방식을 사용한다.

이 논문에서는 일반 표현식을 변환할 때 equivalent 한 DNF 로 변환하지 않고, 새로운 변수로 하위 표현식(subformula)을 치환하여 변환하였다. 이 변환방법은 표현식을 DNF 로 변환하는데 equivalent 하게 변환시키지 않으나 SAT 여부만 검사하기에는 효율적이다. DNF 가 SAT 이기 위해서 한 clause 만 " true" 값으로 할당할 수 있는지 검사하는 방식인데, 한 clause 내에 literal 들을 모두 true 로 할당할 수 있는지 여부만 검사한다. 이는 literal 의 값에 따라 다른 달라지는 다른 clause 의 값을 검사해야 하는 CNF 보다 효율적이다. 특히 이 방법은 clause 의 size 가 작을수록 더 간단해진다. size 가 1 인 clause, 즉 unit clause 가 존재하는 경우에는 바로 전체 표현식이 SAT 임을 알 수 있다. 이 방식은 complete algorithm 이다. incomplete algorithm 은 효율성이 더 좋을 수 있으나 모델이 하나일 경우에는 SAT/UNSAT 여부를 모를 수도 있다. Incomplete algorithm 과는 달리 complete algorithm 은 모든 모델을 찾아낼 수 있고 contradiction 을 이끌어내거나, 증명이 가능하다. SAT 여부를 검사하기에도 효율적이다.

그런데 DNF 를 이용한 SAT 검사는 변환에 많은 시간이 걸린다. 그리고 DNF 에서 size 가 1 또는 2 인 clause 는 존재하는 경우가 드물다. 일반 표현식을 DNF 로 변환하는 방법에 좀더 보완을 한다면 더 효율적인 SAT 검사를 할 수 있다.

6. 참고 문헌

[1] A.Bierre and W.Kunz, SAT and ATPG : Boolean engines for formal hardware verification, Proceedings of 20th IEEE/ACM international Conference on Computer Aided Design(ICCAD' 02), San jose CA. USA. Novemver 2002.

[2] A.Biere, A.Cimatti, E.M.Clarke, M.Fujita, Y.Zhu, Symbolic Model Checking using SAT procedures instead of BDDs (1999) proceedings of Design Automaton conference(DAC' 99)

[3] M. Davis, G. Logemann, and D. Loveland, A machine program for theorem proving., Communications of the ACM, (5):394-397, 1962

[4] A.V.Gelder, A satisfiability tester for non-clausal prepositional calculus. Information and computation, 79(1):1-21, Oct.1988

[5] J.Peysakh, A fast algorithm to convert Boolean expressions into CNF, IBM Computer Science RC 12913, No.57971, T.J.Watson, New York, 1987