SAT Preprocessor의 구현 및 실험

남명진[0+] 최진영[+] 곽희환[++]
[+]고려대학교 컴퓨터학과
{mjnam[0], choi}@formal.korea.ac.kr
[++]Synopsys.Inc.
hkwak@synopsys.com

Implementation and Experiments of SAT Preprocessor

Myoung-Jin Nam[0+] , Jin-Young Choi[+] Hee Hwan Kwak[++]
[+]Dept. of Computer Science and Engineering, Korea University
[++]Synopsys.Inc.

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 위한 여러 연구가 진행되고 있고, 그 결과 성능이 좋은 solver들이 개발되었다. 하지만 문제가 커질수록 solver의 처리 시간이 급격하게 증가한다. 이 논문에서는 solver가 복잡한 문제를 더 효율적으로 풀기 위해 논문 " Local search for Boolean relations on the basis of unit propagation" [5]에서 제안된 preprocessor(전처리기), P_EQ의 개념을 설명하고, 실험을 통한 결과를 제시한다.

1. 서 론

SAT(satisfiability) 문제는 어떠한 전자 회로가 언제나 주어진 속성을 만족하는지를 알아보는 하드웨어 검증 등에 사용되며, 또한 모델 체킹 등에도 사용된다. 이와 같은 방식으로 하드웨어나 소프트웨어 시스템의 정확성(correctness)을 확인하여 보다 안정된 시스템을 구현하는데 도움이 된다. 이러한 SAT 문제를 효율적으로 풀기 위한 DP, DPLL[1] 등의 알고리즘들이 존재한다. 이 알고리즘들은 이진 표현식을 입력으로 받아서, 이 표현식이 SAT 인지 검사한다.

현재 SAT 를 자동으로 검사하는 SAT solver 가 많이 개발되어 사용되고 있다. zchaff[2], Berkmin[3], GRASP[4] 등이 그 예이다. 이 solver 들은 CNF 를 입력으로 받아서, SAT/UNSAT 여부를 출력해주고, solver 에 따라서 satisfying assignment 를 출력하기도 한다.

앞에 열거된 solver 들은 CNF 의 크기가 클수록, 그리고 복잡할수록 시간과 용량이 크게 늘어난다. 그래서 SAT solver 로 풀기 전에 proprocessor 를 이용하여 문제를 간단하게 만든 다음에 SAT solver 를 이용하기도 한다. 오직 solver 로만 푸는 시간보다 preprocessor 로 우선 푼 후에 solver 로 푸는 시간이 더 줄어드는 경우가 많다. 대표적인 것이 앞에서 언급된 BerkMin 의 proprocessor 인 P_EQ[5]이다. 이 논문에서는 P_EQ 의 개념을 설명하고, 두개의 실험 결과 및 분석을 제시한다.

2. Satisfiability Problem(SAT)

satisfiability problem(SAT)는 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*) 또는 satisfying assignment 라고 한다. 그런데, 특별한 형식이 없는 일반 표현식은 SAT 임을 검사하기 힘들다. 그래서 일반 표현식은 특별한 형태로 바꾸어서 SAT 임을 증명하는데, 보통 CNF 를 많이 사용한다.

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

위와 같은 CNF 가 있을 때 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. 대부분의 SAT 알고리즘은 CNF 을 입력으로 하며, SAT solver 도 마찬가지이다.

2.1 Preprocessor P_EQ 의 목적과 기본 개념

P_EQ 는 sat solver 이전에 strong relation(unit clause, equivalencies 등)을 찾아서 기존의 CNF 에

relation 을 추가하는 역할을 한다. 현재까지는 unit clause 와 equivalence 만을 체크하고 있다. unit clause 의 경우, 한 literal 이 unit clause 가 된다는 것은 그 literal 이 속한 clause 에 나머지 literal 들이 false 가 된다면 남은 하나의 literal 은 무조건 unit clause 로써 true 가 할당된다. 그리고 두 literal 의 값이 같다면 한 literal 을 다른 literal 로 치환 시킬 수 있다. P_EQ 는 이러한 경우가 있는지 보기 위하여 몇 개의 변수를 선택하여 이 변수값의 모든 경우를 대입해보는 것이다. 할당된 값으로부터 unit propagation 을 적용해 나가면서, 모든 branch 에서 unit clause 가 되는 literal 이 있는지, 또는 모든 경우에서 equivalence 관계에 있는 literal 들이 있는지 찾는다. 단, 여기서 strong relation 을 찾는 대상은 conflict 가 일어나지 않는 search space 에 한정된다. 이를 **partition of Boolean space** 라 한다. P_EQ 는 search space 를 conflict 가 발생하는 space 와 일어나지 않는 space 로 partitioning 하여, conflict 가 발생하지 않는 space 의 strong relation 만을 찾는다. 그리고 P_EQ 는 **local search** 를 이용하여 CNF 내의 모든 clause 가 아닌 부분만을 검사한다.

즉, P_EQ 는 CNF 를 받아 unit clause 를 추론하여 원래 CNF 에 unit clause 를 추가하고, equivalence 관계에 있는 literal 들을 찾아서 하나의 literal 을 다른 literal 로 치환하여 SAT solver 의 입력이 되는 새로운 CNF 를 생성한다. 여기서, P_EQ 의 coverage 가 전체가 아니기 때문에 P_EQ 가 찾지 못하는 unit clause 가 존재할 수 있지만 P_EQ 는 preprocessor 이기 때문에 모든 경우를 다 찾을 필요는 없다. 그리고 unit clause 로 추론된 literal 의 negation 이, P_EQ 가 검사하지 않은 CNF 의 나머지 부분에 unit clause 로 이미 존재한다면, 오히려 sat solver 는 conflict 를 도출하기 쉬워진다.

2.2 Inductive stage

CNF F 의 변수 집합의 subset 을 observable set, Z 라 하고, Z 의 subset 을 branch variable set, Y 라고 하자. Y 의 변수에 대해서 branch 하여 unit propagation(또는 Boolean constraint propagation)을 적용해 나간다. 그러던 Z 의 원소가 unit clause 인 경우에 그 값을 **ternary matrix T(Z)** 에 저장한다. 그리고 이 matrix 를 observable 하다고 한다. 예를 들어 Z = {x,y,z,a,d,e,f,g}라 하고, Y = {x,y,z}라 하면 matrix 는 다음과 같다.

C0		C2		C4	C5	C6		
0		0		1	1	1		x
0		1		0	0	1		y
0		0		0	1	0		z

1	1	1	-	-	0	-	1	a
0	0	-	-	1	-	-	-	d
1	0	0	1	1	1	0	0	e
0	0	-	0	-	1	1	-	f
0	-	0	-	0	0	0	1	g
0		0		0	0	0		B

[표 1] observable matrix

C0 에서 C7 은 cube 라고 하고, 각 search space 를 나타낸다. " 0" 은 true, " 1" 은 false, " -" 은 unknown(한 clause 에서 한 literal 이 true 로 할당되었을 때, 나머지 literal 의 값을 unknown 으로 한다.)을 나타낸다. 그리고 마지막 행의 B 는 *conflict vector* 로써, i 번째 원소가 1 이면 그 cube 는 conflict 가 일어났음을 뜻하고, solution 이 존재하지 않는다.

여기서, unit clause 와 equivalencies 만 고려한다면 binary matrix 로 하는 것이 성능이 더 좋다.

2.3 Deductive stage

observable matrix 를 바탕으로 각 변수에 대한 vector 를 생성한다. t_w 는 변수 w 에 대한 vector 이다. [표 1]에서 t_g 는 (0,-,0,-,0,0,0,1)가 된다. 그리고 *R* 을 observable variable 에 대한 논리 표현(logical expression)이라고 하자. R 내의 각 변수를 vector 로 변환하면, vector expression R_*을 얻을 수 있다. R = (¬a → d) ∨ f 일때 R_*은 (¬t_a → t_d) ∨ t_f 이다.

Theorem. $R \Leftarrow F$ if $R\_* \vee b = 1\_*$ [5]

여기서 F 는 초기의 CNF 이고, ⇐은 R ← F = 1, b 는 conflict vector, 1_*은 b 와 size 가 같고 1 만 원소로 갖는 vector 를 뜻한다. 그럼 다음과 같은 결과를 얻을 수 있다.

$p \Leftarrow F$ if $t\_p \vee b = 1\_*$ (1) unit clause
$p \leftrightarrow q \Leftarrow F$ if $(t\_p \leftrightarrow t\_q) \vee b = 1\_*$ (2) equivalence

[표 1]에서 ¬t_g ∨ b = 1_* 이다. 따라서 " ¬g" 는 CNF F 에 추가될 수 있다. 그리고 (t_y ↔ ¬t_e) ∨ b = 1_* 이므로 y 는 F 에서 ¬e 로 치환된다. 여기서 b1=b3=1 이기 때문에 y ↔ ¬e 는 C1 에서 성립되지 않음에도 불구하고 equivalence 관계라고 말할 수 있다. 이것을 conflict 의 **masking effect** 라고 한다.

2.4 실험 결과 및 Observable Set 의 선택

논문 [Local search for Boolean relations on the basis of unit propagation]에서는 임의적인 observable

set 을 선택한 후 두 번의 traverse 를 통하여 observable set 을 확장하고, branch variable 의 Set 을 선택한다. traverse 방법에 대한 자세한 내용은 [5]논문에 자세하게 나와 있다.

우선 논문[5]의 저자가 구현한 P_EQ 의 8fvp-UNSAT[6]에 대한 실험 결과는 다음과 같다.

Name	BerkMin	P_EQ			P_EQ+ BerkMin
	Time	Eq_lit	Units	Time	Time
6pipe	701.7	72	1713	89.7	**455.9**
7pipe	3237.5	71	3007	220.8	**1201.5**
Total	4194.7			618.2	**2035.2**

[표 2] 8fvp-UNSAT2.0

위에서 Eq_lit 은 equivalence 인 literal 들의 개수이고, units 생성된 unit clause 의 개수이다. 6pipe 의 경우에 BerkMin 으로만 풀면 701.7 초가 소요되지만 P_EQ 로 preprocessing 을 한 후에 BerkMin 으로 풀면 BerkMin 시간 + P_EQ 시간보다 작은 455.9 초가 나옴을 알 수 있다.

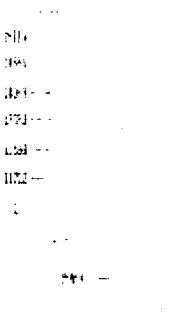
이 실험에서 쓰인 8fvp-UNSAT2.0[6]은 equivalence checking 에 관련된 CNF 이다. 논문[5]는 회로에 관련된 CNF 에 대해서는 다음과 같은 방법을 제시하였다. branch variable 의 set 을 선택할 때 circuit 의 primary input 을 선택한다. intermediate variable 로 branch 하는 것보다 primary input 으로 branch 하는 것이 다른 variable 들의 값의 할당에 더 많은 영향을 주기 때문이다. 하지만 primary input 에 대한 정보가 없을 경우에는 변수 이름의 숫자 크기로 결정한다. 변수 이름의 숫자가 작을수록 input 에서 가깝다고 가정한다.

하지만, 논문[5]의 제안대로 구현하고, observable set 을 정한 결과 unit 개수는 논문[5]에서 제시한 결과와 같지 않다. 6pipe 의 경우 unit clause 가 생성되지 않는다. 논문[5]에서는 첫번째 traverse 전에 1 에서 150 까지의 변수를 observable set, 즉 primary input 으로 가정하는데 실제로 6pipe 의 경우 변수 이름의 숫자가 작다고 하더라도 반드시 primary input 이 아니다. 예를 들어, 6pipe.cnf 내의 변수 " 1" 이 속하는 clause 의 set 과 그것을 회로로 바꾼 결과는 [표 3]에 나와 있다.

[표 3]에서 변수 " 1" 이 속하는 clause 들의 set 을 분석하면 다음과 같다. (1,15469), (-1, 15469)은 1XOR 15469 이고, 나머지 부분은 input 이 15119, 14650, 14069, 13383, 12599, 11722,2 이고 output 인 1 인 OR-gate 로 만들 수 있다. 6pipe 의 변수 15469 는 unit clause 로써 6pipe 의 output 이다. 따라서 변수 1 은 output 에 더 가까움을 알 수 있다.

그리고 6pipe 의 분석 결과 변수 2 는 1 을 output 으로 하는 gate 의 input 으로, 변수 3 은 다시 그 것의 input 이 되는 식이다. 이 예제에서 보듯 논문[5]에서 제안된 Primary input 결정 방법이 모든 CNF 에 해당되는 것이 아니다.

(1, 15469) , (-1, -15469)
(15119, 14650, 14069,
13383, 12599, 11722, 2, -
1)
(-15119, 1), (-14650, 1)
(-14069, 1), (-13383, 1)
(-12599, 1), (-11722, 1) ,
(-2, 1)



[표 3]6pipe 의 변수 1 와 관련된 clause 와 회로

3. 결론

P_EQ 는 SAT solver 로 풀기 전의 preprocessor 로써 크고 복잡한 문제들을 solver 가 쉽게 풀도록 CNF 에 정보를 추가하여 준다. 그 중 unit clause 와 equivalencies 는 sat solver 의 처리 시간을 현저히 낮춰준다. 하지만, preprocessor 는 주로 큰 문제들을 푸는데 사용되는데, 특히 회로에 관련된 문제에서 structural information, 그 중에서도 primary input 에 대한 정보가 없다면 preprocessing 의 효과를 얻을 수 없다. 따라서 CNF 에서 회로의 정보를 찾아내어 이용하는 것이 중요하다.

4. 참고 문헌

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

[2]M.W.Moskewicz, C.F. Madigan, Y.Zhao, L.Zhang, and S.Malik, Chaff:Engineering an Efficient SAT Solver,38th Design Automation Conference(DAC), pp.530-535, 2001.

[3]E.Goldgerg and Y.Nivikov, BerkMin : A Fast and Robust SAT solver, Design Automation and Test in Europe(DATE), pp.142-149,2000.

[4]J.P. Marques-Silva, K.A. Sakallah, GRASP:A Search Algorithm for prepositional Satisfiability, IEEE Transaction on computers, vol.48,pp.506-521,1999.

[5] Y.Nivikov, Local Search for Boolean Relations on the Basis of Unit Propagation, IEEE , 1530-1591, 2003

[6]M.Velev. CMU benchmark suite.available from http://www.ece.cmu.edu/~mvelev.