Model Accumulation 을 이용한 SAT Hard Instance 의 해결 방법 연구

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

Research of model accumulation to solve SAT hard instance

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

요 약

SAT 문제는 하드웨어/소프트웨어 검증과 모델 체킹 등 다양한 분야에서 유용하게 사용되고 있으나 복잡도가 NP-complete 라는 어려움을 가지고 있다. 다양한 알고리즘과 휴리스틱, 도구들이 개발되었지만 그럼에도 불구하고 해결할 수 없는 hard instance 들이 존재한다. 이 논문에서는 그러한 hard instance 를 해결하기 위한 방법의 하나로 model accumulation 을 제안한다.

1. 서론

SAT(SATisfiability) 문제는 propositional formula 가 주어졌을 때, 이를 만족하는 모델이 존재하는지 여부를 알아보는 문제이다. 이는 하드웨어/소프트웨어 검증과 모델 체킹 등의 다양한 분야에 적용되어 안전한 시스템을 만드는 데 이용된다. 이 문제는 Cook 에 의해 최초로 NP-complete 임이 증명되었다. 따라서 최악의 경우 복잡도가 지수증가를 보이지만, 평균적으로 혹은 특정한 대상에 대해 효율적인 알고리즘, 휴리스틱, 도구들이 계속해서 개발되어 왔다. 그러나 많은 발전에도 불구하고 검증 대상이 점차 복잡해짐에 따라서 아무리 효율적인 방법을 사용하여도 결과가 나오지 않는 경우가 종종 있다. 이 논문에서 제안하는 model accumulation 은 기본 SAT 알고리즘을 변형하여 잘 풀리지 않는 문제들, 다시 말해 hard instance 를 해결하고자 하는 방법이다.

2. 관련 연구

먼저 SAT 문제에 관한 기본 개념과 알고리즘들에 대해서 알아보겠다. SAT 문제는 propositional formula 를 대상으로 하는데, formula 는 다음과 같이 이루어진다. 먼저 0 과 1 의 값을 가질 수 있는 변수($x_1,...x_n$)가 있고, 변수와 변수의 negation 을 literal(x_n, $\sim x_n$)이라 한다. 하나 이상의 literal 들을 논리합(OR, disjunction)으로 연결한 것을 clause 라 한다. Formula 는 하나 이상의 clause 를 논리곱(AND, conjunction)으로 연결한 것이다. 이러한 형태를 CNF(Conjunctive Normal Form)이라 하는데, SAT 문제를 해결하는 데는 대부분 이 형태를 사용한다.

변수가 n 개인 경우, 변수에 값을 할당할 수 있는 방법은 2^n 개의 조합이 가능하다. 이러한 조합 중에 formula 의 값을 1 로 만드는 것이 존재하면 그 formula 를 satisfiable 하다고 하며, 하나도 없는 경우에는 unsatisfiable 하다고 한다. 그리고 formula 의 값을 1 로 만드는 변수 값의 조합을 모델이라고 한다. SAT 문제란 어떠한 propositional formula 가 주어졌을 때, 그 formula 가 satisfiable 한지 여부를 가리는 것이다.

이 문제를 해결하기 위해서 DPLL[1], stålmarck [2] 등 여러 알고리즘과 Grasp[3], Chaff[4], BerkMin[5] 등 많은 도구가 개발되었다. 이러한 많은 연구의 결과로 SAT solver 는 성능이 많이 향상되었지만, 검증하고자 하는 대상의 복잡성이 커짐에 따라 풀 수 없는 hard instance 들이 종종 나타난다.

이 논문에서는 현재의 SAT solver 로 풀리지 않는 hard instance 를 풀기위한 방법으로 model accumulation 을 제안한다. 다음 장에서는 그 알고리즘을 설명하고 예제를 제시한다.

3. Model Accumulation

SAT 알고리즘은 크게 complete 알고리즘과 incomplete 알고리즘으로 나눌 수 있다. 전자는 언제나 satisfiable /unsatisfiable 의 결과를 내는 것이고, 후자는 satisfiable 한 경우는 답을 주지만 unsatisfiable 하다는 결론은 내릴 수 없는 것이다. Incomplete 알고리즘이 많은 경우 satisfiable 함을 밝히는 데는 더 빠르지만, 대신 unsatisfiable 함을 밝힐 수는 없다. 대부분의 complete 알고리즘은 satisfiable 함을 안 경우에 입력 formula 를 1 로 만드는 모든 변수의 할당 값, 즉 모델을 알려준다.

Model accumulation 은 기본적으로 complete 알고리즘을 이용한다. 이는 검증하고자 하는 formula 를 두개로 나눈 다음, 하나의 sub-formula 의 모든 모델을 또 다른 sub-formula 에 적용하여 전체 formula 의 satisfiability 를 검증하는 방식이다.

3.1. Model Accumulation 알고리즘

먼저 검증하고자 하는 대상인 formula 를 F 라 하고, 이를 두 개의 sub-formula 로 나누어 F1 과 F2 라 한다. 즉 F1 과 F2 는 F 의 clause 들의 부분 집합이며 partition 을 이룬다. F 의 모든 변수 중에서 F1 과 F2 에 모두 나타나는 변수를 전역 (global) 변수라 한다. [그림 1]은 기본 SAT 알고리즘을 바탕으로 한 model accumulation 알고리즘을 간단히 나타낸 것이다.

이 알고리즘의 입력은 검증을 원하는 formula 를 나눈 두 개의 sub-formula 이고, 결과는 SATISFIABLE 과 UNSATISFIABLE 중의 하나이다. 먼저 F1 과 F2 의 전역 변수

```
model_accumulation(F1, F2) {
  globals = global_variables_of(F1, F2);
  models =∅;
  while(1) {
    a_model = find_a_model(F1, globals);
    if (a_model != NULL) {
      models.add(a_model);
      add_a_clause(F1, negate(a_model));
    }
    else
      break;
  }
  if (models == ∅)
    return UNSATISFIABLE;
  else {
    models = minimize(models);
    for (i=0; i<size_of(models); i++) {
      F2' = apply_model(F2, models.get(i));
      if (find_a_model(F2') != NULL)
        return SATISFIABLE;
    }
    return UNSATISFIABLE;
  }
}
```

[그림 1] Model Accumulation 알고리즘

를 구한 다음, F1 의 모든 모델을 찾는다. F1 의 모델을 찾을 때 F 에 있는 모든 변수의 할당 값을 찾는 것이 아니라, 전역 변수만을 대상으로 한다. 따라서 F 의 변수가 n 개이고 전역 변수가 m 개이면 가능한 모델의 개수는 2^n이 아니라 2^m이 된다.

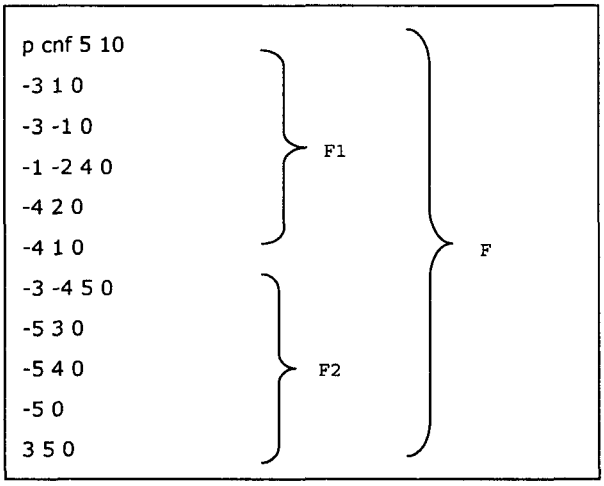
하나의 모델은 각 변수에 값을 할당한 것이므로 각 변수나 그 변수의 negation 을 모두 AND 로 연결한 DNF(Disjunctive Normal Form)로 나타난다. 이 모델을 negation 하면 하나의 clause 가 되는데, 이를 F1 에 추가하여 모델을 다시 찾으면, 추가한 모델을 제외한 다른 모델을 찾아주게 된다.

이러한 방법으로 F1 의 모든 모델을 찾고 나면 while 루프를 빠져 나오는데, 만약 모델이 하나도 없으면 F1 이

unsatisfiable 하므로 F 역시 unsatisfiable 하다. 모델이 존재하는 경우, 먼저 minimize 하여 개수를 줄인다. 그런 다음 각 모델을 F2 에 적용하는데, 이는 모델에 있는 대로 변수에 값을 할당하는 것을 의미한다. 다시 말하면, 값이 할당된 변수의 개수만큼 unit clause 가 추가되는 것이다. 이 때 한번이라도 모델을 찾으면, 즉 결과가 satisfiable 이면 F 는 satisfiable 하다. 모든 모델을 적용했을 때 항상 F2 가 모델을 찾지 못하면, 즉 unsatisfiable 하면 F 는 unsatisfiable 하다.

3.2. 예제

Model accumulation 에 대한 간단한 예제를 들어보겠다. 먼저 target formula 는 [그림 2]와 같다.



[그림 2] target formula

[그림 2]는 formula 를 DIMACS 표준에 의한 CNF 로 나타낸 것이다. 전체 formula 가 F 이고, 열 개의 clause 중 위의 다섯 개를 F1, 아래의 다섯 개를 F2 라 하자. [그림 1]의 알고리즘에 따라 먼저 전역변수를 구하면 {3,4,5}가 된다. 그런 다음 F1 의 모델들을 구하면 (-3,4,5), (-3,4,-5), (-3,-4,5), (-3,-4,-5) 네 개가 나온다. 이를 Espresso[6]를 이용하여 minimize 하면 (-3)이 된다.

이제 이 모델을 F2 에 적용해보자. 다시 말해 이는 (-3)이라는 unit clause 를 추가하는 것이다. 그런 다음 모델을 찾아보면 나오지 않는다. 즉, 모델을 적용한 F2 는 unsatisfiable 하다. 이 예제에서는 F1 의 모델이 하나밖에 존재하지 않으므로 이 결과를 통하여 F 가 unsatisfiable 함을 알 수 있다.

4. 결론 및 향후 과제

지금까지 기존의 SAT solver 로 풀리지 않는 hard instance 를 해결하기 위한 시도인 model accumulation 에 대해서 설명하였다. 어떤 instance, 다시 말해 어떤 formula 가 SAT solver 로 풀리지 않는 것은 보통 두 가지 경우인데, 첫째는 시간이 너무 오래 걸리는 것이고 두 번째는 메모리가 부족해서 해결하지 못하는 것이다. Model accumulation 은 두 번째 경우를 해결하기 위한 시도로서 하나의 formula 를 두개로 나누어 SAT solver 를 여러 번 실행시킴으로써 결과를 얻고자 하는 것이다.

현재의 알고리즘을 바탕으로 차후에 구현을 통해 model accumulation 의 실효성을 검증해볼 계획이다. 또한 하나의 formula 를 두 개로 나눌 때, 어떻게 하는 것이 결과가 좋은 지도 더 연구해야 할 점이다.

5. 참고 문헌

[1] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory", J. ACM, vol. 7, pp. 1,015-1,028, 1960.

[2] M. Sheeran and G. Stålmarck, "A Tutorial on Stalmarck's Proof Procedure for Propositional Logic", FMCAD, 1998.

[3] J.P. Marques-Silva, K.A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability", IEEE Transaction on Computers, vol. 48, pp. 506-521, 1999.

[4] 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.

[5] E. Goldberg and Y. Nivikov, "BerkMin: A Fast and Robust SAT Sovler", Design Automation and Test in Europe(DATE), pp. 142-149, 2000.

[6] Espresso, http://www-cad.eecs.berkeley.edu/Software/software.html