SAT 사례 연구: 루빅 큐브

임형주, 권기현
경기대학교 컴퓨터과학과
e-mail : {limhome01, khkwon}@kgu.ac.kr

A Case Study on Satisfiability : Rubik's Cube

Hyoung-Joo Lim, Gihwon Kwon
Department of Computer Science, Kyonggi University

요 약

최근 정형 검증 분야에서 만족가능성 처리기를 사용한 연구가 많아지면서 이를 적용한 사례들이 많이 나타나고 있다. 만족가능성 처리기를 사용하기 위해서는 검증 되어야 할 시스템을 CNF 식으로 변환해야 한다. 우리는 루빅 큐브를 만족가능성 처리기를 통해 풀어보기 위해 이를 CNF 식으로 변환해 보았다. 루빅 큐브는 정육면체로 이루어진 퍼즐의 일종으로 모든 면이 각각 한가지 색으로만 채워져야 하는 문제이다. 우리는 본 논문에서 만족가능성 처리기를 사용한 사례 중에 아직 적용한 적이 없는 루빅 큐브를 CNF 식으로 변환해 풀었음을 보여준다.

1. 서론

바운디드 모델 검증[1]에 대한 유용성이 알려지면서 이 주제에 대한 연구가 많이 시도되고 있다. 디바이스 드라이버 오류 검출, 임베디드 소프트웨어 검증 등이 그 사례가 될 수 있다. 비록 바운디드 모델 검증에 대한 문헌이 많지만, 초보자들이 이러한 문헌을 읽고 이해하기 쉽지 않다. 특히, 바운디드 모델 검증의 핵심인 SAT(Satisfiability)변환에 대한 상세한 이해 없이는 바운디드 모델 검증을 이해하거나 활용하기 어렵다. 본 논문에서는 독자들이 바운디드 모델 검증을 쉽게 이해하는 것을 돕기 위해서 만족가능성 문제의 사례 연구를 상세히 기술코자 한다. 여기에서 사례로 선정한 것은 루빅 큐브[2]이다. 이것을 선정한 이유는 다음과 같다. 첫째, 우리가 알고 있는 한 루빅 큐브를 아직까지 SAT 문제로 다룬 연구는 없다. 둘째, 루빅 큐브는 인공지능의 계획 문제로서, 바운디드 모델 검증을 이해하는데 유용하다. 왜냐하면 비에르(Biere)는 인공지능의 결정적 계획 문제에서 바운디드 모델 검증의 아이디어를 얻었기 때문이다. 거듭 강조하지만, 본 논문에서는 새로운 아이디어를 제안하기보다는 사례 연구를 충실히 다루어, 독자들로 하여금 주어진 문제를 SAT 으로 변환하는 방법을 이해시키고자 한다.

논문의 구성은 다음과 같다. 2 장에서는 배경 지식을 다룬다. 여기서는 만족가능성 문제에 대한 개념을 다룬다. 그리고 3 장에서는 루빅 큐브 문제를 SAT 문제로 다루는 방법을 살펴본다. 4 장에서는 루빅 큐브의 풀이 시간을 단축시키기 위한 내용을 다루며, 5 장에서는 실험결과를 다룬다. 마지막으로 6 장에서 결론 및 향후 연구를 기술한다.

이 논문은 2007 년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(R01-2005-000-11120-0)

2. 배경 지식

주어진 문제를 SAT 으로 풀기 위해서는 문제를 명제 논리식으로 변환해야 한다. 다양한 명제 논리식 표현이 있지만, 여기서는 만족가능성 처리기[3]의 표준인 CNF(Conjunctive Normal Form) 형식을 사용한다.

정의 1 (논리식 구문). 이진 변수 집합 $X=\{x_1,x_2,\ldots\}$상에서 CNF 논리식의 구문을 정의한다.

$$L ::= x \mid \neg x$$

$$C ::= \bigvee_{1 \le i \le n} L_i$$

$$\phi ::= \bigwedge_{1 \le i \le n} C_i$$

이진 변수는 더 이상 하위 논리식으로 분해할 수 없기 때문에 변수 $x \in X$ 또는 그것의 부정 $\neg x \in X$ 을 리터럴 L 이라고 부른다. 절 C 는 리터럴의 이접(disjunctive, $\vee$)이며 $m = 1$ 인 경우는 단항 절, $m = 2$ 인 경우는 이항 절, 그리고 $m \ge 3$ 인 경우는 다항 절이라고 부른다. 논리식 ϕ 는 절의 연접(conjunctive, $\wedge$)으로 구성되며 n 은 절의 개수이다.

구문 규칙에 맞게 작성된 논리식의 의미를 평가하기 위해서는 각 변수에 값을 배정해야 한다. 변수는 참(⊤) 또는 거짓(⊥)을 가질 수 있기 때문에 값 배정은

함수 $\alpha : X \rightarrow \{ \top, \bot \}$로 표현할 수 있다. 만약 사용된 변수가 n 개이면 값을 배정할 수 있는 경우의 수는 모두 2^n 이다. 논리식의 의미는 배정에 좌우된다. 특히 배정 α 하에서 논리식 ϕ 가 참으로 평가되는 경우를 $\alpha \models \phi$ 로 표기하고 "배정 α 가 논리식 ϕ 를 만족한다"라고 읽는다. 사용된 기호 $\models$ 는 만족 관계를 나타내는 메타 기호이다.

정의 2(논리식 의미). 만족 관계를 바탕으로 논리식의 의미를 귀납적으로 정의한다.

$$\alpha \models x \quad iff \quad \alpha(x) = \top$$
$$\alpha \models \neg x \quad iff \quad \alpha(x) = \bot$$
$$\alpha \models \bigvee_{1 \le i \le m} L_i \quad iff \quad \exists_{1 \le i \le m} \cdot \alpha \models L_i$$
$$\alpha \models \bigvee_{1 \le i \le n} C_i \quad iff \quad \forall_{1 \le i \le n} \cdot \alpha \models C_i$$

리터럴 L 의 의미는 긍정 리터럴(x) 또는 부정 리터럴($\neg x$)에 따라서 다르다. 긍정 리터럴이 만족되기 위해서는 해당 변수의 값이 참이어야 하며, 부정 리터럴이 만족되기 위해서는 해당 변수의 값이 거짓이어야 한다. 절 C 는 리터럴의 이접이기 때문에 절이 만족되기 위해서는 최소한 하나의 리터럴이 만족되어야 한다. 비슷하게 논리식 ϕ 는 절의 연접이기 때문에 논리식이 만족되기 위해서는 모든 절이 만족되어야 한다.

만족 관계에 따라서 논리식을 세 가지 그룹으로 구분한다. 첫째, 선택한 배정에 무관하게 항상 만족되는 논리식을 타당(vailid)하다고 부른다. 즉 $\forall_\alpha \cdot \alpha \models \phi$ 이며, 줄여서 $\models \phi$ 로 표기한다. 둘째, 논리식을 만족하는 배정이 존재하는 경우 만족가능(satisfiable)하다고 부른다. 즉 $\neg\exists_\alpha \cdot \alpha \models \phi$ 이며, 줄여서 $\not\models \phi$ 로 표기한다.[4]

살펴본 바와 같이 논리식이 만족가능인 경우, 그 논리식을 만족하는 배정이 존재한다. 이러한 배정은 응용 분야에 따라서 다양하게 해석된다. 예를 들어 인공 지능에서 경로 찾기 문제를 표현한 논리식이 만족 가능인 경우, 해당 논리식을 만족하는 배정은 바로 찾고자 하는 경로이다[5]. 이러한 SAT 문제는 이론 전산학으로부터 인공지능, 소프트웨어 공학, 정형 검증 등 전산학의 많은 분야에서 활용되고 있다.

3. 루빅 큐브를 SAT 문제로 간주

1970 년 헝가리의 Erno Rubik 에 의해 발명된 루빅 큐브는 그 당시 가장 유명했던 조합 퍼즐이다. 루빅 큐브의 종류는 2x2x2, 3x3x3, 4x4x4 루빅 큐브등 많은 종류가 현존하지만 본 논문에서는 2x2x2 루빅 큐브를 SAT 문제로 간주한다.

2x2x2 루빅 큐브는 표 1 과 같이 깊이가 깊어질수록 12^n 은 경우의 수를 갖는다. 여기서 12 는 행위의 수이며 n 은 깊이를 의미한다.

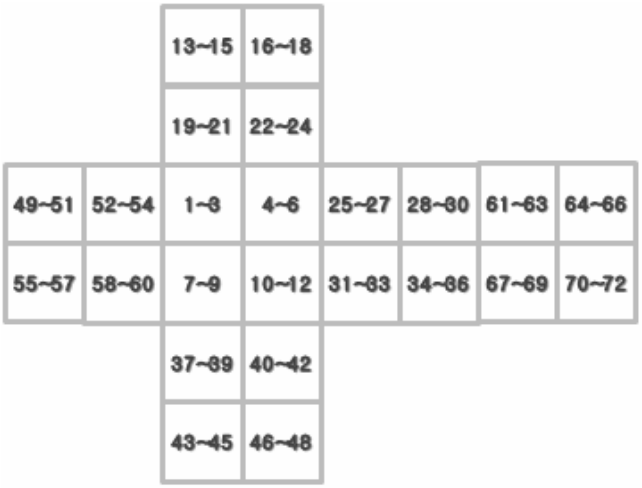
이번 장에서는 루빅 큐브를 CNF 로 변환한 방법을 설명한다. 먼저 루빅 큐브의 컬러를 표현한 방법과 변환 방법을 설명하고, 루빅 큐브의 행위를 CNF 로 변환한 방법을 설명한다. 마지막으로루빅 큐브의 초기상태와 목표상태를 변환한 방법을 설명한다.

<표 1> 2x2x2 루빅 큐브의 경우의 수

Depth	2x2x2 루빅 큐브
1	12
2	144
3	1,728
4	20,736
5	248,832
6	2,985,984
7	35,831,808
8	429,981,696
9	5,159,780,352
10	61,917,364,224
11	743,008,370,688
12	8,916,100,448,256
13	106,993,205,379,072
14	1,283,918,464,548,864

*현재 풀 수 있는 Depth 는 13 이다.

루빅 큐브는 서로 다른 6 개의 컬러로 이루어진 정육면체이다. 우리는 6 개의 컬러를 표현하기 위해서 6 개의 이진 변수를 사용하였다. 6 개의 변수 중에서 정확하게 한 개만이 참의 값을 가져야 하며, 나머지는 모두 거짓이 되어야 한다. CNF 기반의 만족가능성 처리기에서는 하나의 참값을 고르기 위해 "정확히 하나(exactly one)"라는 표현을 사용하며, 이는 "최소한 하나(at least one)"와 "최대한 하나(at most one)"의 연접으로 이루어진다.



(그림 1) 컬러 변환에 사용된 변수

그림 1 은 각 블록에 사용되는 변수들을 나타낸다. 그림과 같은 2x2x2 루빅 큐브의 컬러를 표현하기 위해서는 144(=6 컬러*4 블록*6 면)개의 이진 변수가 필요

하며 바운드 k 가 증가할 때마다 매번 144 개의 변수가 추가된다. 생성된 CNF 의 총 변수의 수를 구하는 식은 다음과 같다.

$$\text{총 변수의 수} = 144 * (k + 1) + 12 * k$$

여기서 12 는 행위의 개수이다. 행위는 스텝당 12 개씩 증가하며 매 스텝마다 한 개의 행위만이 실행된다. 행위를 기술하기 위해서는 선행 조건과 후행조건을 나타내야 하지만, 루빅 큐브의 행위에는 행위의 결과를 나타내는 후행조건만이 존재한다. 이는 매 스텝마다 모든 행위가 선택 가능함을 의미한다. 행위의 규칙은 다음과 같다.

- Operator encoding : 행위의 적용 효과를 나타낸다.
- Frame axiom : 행위의 후행 조건에 언급된 내용만 변하고 그 이외의 것은 원래의 값을 보존한다.
- Exclusion axiom : 한번에 오직 하나의 행위만이 일어난다.

루빅 큐브는 기본값으로 초기상태와 목표상태를 갖는다. 기본값은 값이 미리 배정되어있는 변수들의 집합을 말하며, 각 변수들은 CNF 의 단항 절(unit clause)로 나타낸다. 위의 내용은 다음과 같이 정의된다.

정의 3. 2x2x2 루빅 큐브를 CNF 로 변환하기 위해서 k 번째에 s 면, b 블록에 컬러 c 가 배정됨을 나타내는 이진 변수 $x_{k,s,b,c}$ 를 사용해서 루빅 큐브의 각 규칙들을 CNF 논리식으로 변환한다.

수식 1. 블록은 최소한 하나의 컬러를 갖는다.

$$\bigwedge_{1\le k\le n} \bigwedge_{1\le s\le 6} \bigwedge_{1\le b\le 4} \bigvee_{1\le c\le 6} x_{k,s,b,c}$$

수식 2. 블록은 최대한 하나의 컬러를 갖는다.

$$\bigwedge_{1\le k\le n} \bigwedge_{1\le s\le 6} \bigwedge_{1\le b\le 4} \bigwedge_{1\le c_i\le 5} \bigwedge_{c_i+1\le c_j\le 6} \left(\neg x_{k,s,b,c_i} \vee \neg x_{k,s,b,c_j}\right)$$

수식 3. 모든 행위에는 선행조건이 없으며, 행위가 일어난 후의 효과만 나타난다.

$$\bigwedge_{1\le k\le n} \bigwedge_{1\le i\le 12} \bigwedge_{x,y\in G_i} \bigwedge_{1\le c\le 6} \left(\neg A_{k,i} \vee \neg x_{k,j,c} \vee y_{k+1,j',c}\right)$$

이때 j 는 (s,b)이고, j'는 (s',b')이다. 또한 $y=A_i(x)$이며, y 는 A_i 행위에 의해서 x 의 값을 갖게 되는 변수이다. G_i 는 A_i 행위에 영향을 받는 변수들의 집합이다.

수식 4. 어느 행위가 일어나면, 행위의 후행 조건에 언급된 내용만 변하고 그 외의 것은 원래의 값을 보존한다.

$$\bigwedge_{1\le k\le n} \bigwedge_{1\le i\le 12} \bigwedge_{x\notin G_i} \bigwedge_{1\le c\le 6} \left(\neg A_{k,i} \vee \neg x_{k,j,c} \vee x_{k+1,j,c}\right)$$

수식 5. 매 차례에 행위는 오직 하나의 행위만이 일어난다.

$$\bigwedge_{1\le k\le n} \bigvee_{1\le i\le 12} A_{k,i}$$

$$\bigwedge_{1\le k\le n} \bigwedge_{1\le i_1\le 11} \bigwedge_{i_1+1\le i_2\le 12} \left(\neg A_{k,i_1} \vee \neg A_{k,i_2}\right)$$

수식 6. 미리 값이 배정된 초기상태는 단항 절로 나타낸다. I 는 시작상태를 나타낸다.

$$\bigwedge_{x_{1,s,d,c}\in I} x_{1,s,b,c}$$

수식 7. 미리 값이 배정된 목표상태는 단항 절로 나타낸다.

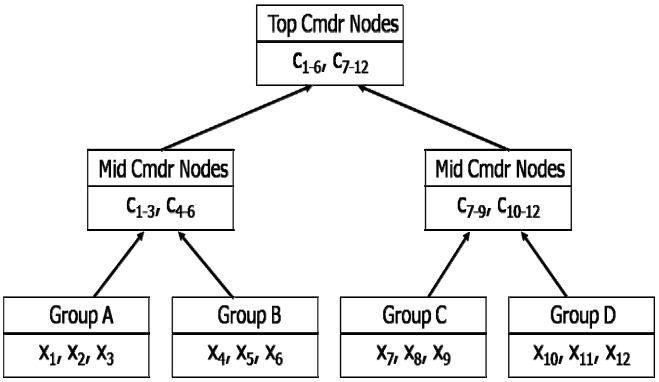
$$\bigwedge_{x_{n,s,d,c}\in E} x_{n,s,b,c}$$

여기서, n 는 실행된 행위의 수이며 E 는 목표상태이다. 따라서 루빅 큐브는 다음과 같이 위의 수식들의 연접으로 나타낸다.

수식 1∧ *수식 2*∧…… ∧*수식 6*∧ *수식 7*

4. 커멘더 인코딩과 새 제약조건

앞 장에서 설명한 *수식 1* 과 *수식 2* 는 m 개중에 1 개를 고르는 알고리즘으로 *NaÏve* 인코딩이라 부른다. 이 알고리즘은 m 의 개수가 커질수록 더 많은 절의 개수를 필요하며, 만족가능성 처리기에게 더 많은 비용을 요구한다. 우리는 본 장에서 *NaÏve* 인코딩의 성능을 개선한 커맨더 인코딩[6](그림 2 참조)과 새로 추가된 제약조건에 대해서 설명한다.



(그림 2). 커맨더 인코딩

커멘더 인코딩은 다음과 같은 수식을 갖는다.

수식 8. 그룹 내의 최대 하나의 변수가 참이다.

$$\bigwedge_{x_j\in G_i} \bigwedge_{x_k\in G_i, j<k} \left(\neg x_j \vee \neg x_k\right)$$

수식 9. 커멘더 변수가 참이면 그룹 내의 변수는 적어도 하나가 참이다.

$$\neg C_i \vee \bigvee_{x_j \in G_i} x_j$$

수식 10. 커멘더 변수가 거짓이면 그룹 내의 모든 변수가 거짓이다.

$$\bigwedge_{x_j \in G_i} (C_i \vee \neg x_j)$$

수식 11. 커멘더 변수 중의 하나는 반드시 참이다.

$$(C_1 \vee C_2 \vee \ldots \vee C_m) \wedge \bigwedge_{i \le m} \bigwedge_{j<i} (\neg C_i \vee \neg C_j)$$

원래 변수의 집합을 $X=\{x_1,\ldots,x_n\}$라 하면, X 를 m 개의 부분 집합 $G_1,\ldots,G_m$으로 분할하고, 각 그룹 G_i마다 커멘더 변수 C_i를 생성한다. *Naïve* 인코딩과 비교했을 때, 변수의 수는 증가하나 절의 수가 감소한다.

루빅 큐브는 표 1 에서 나타나듯이 많은 경우의 수를 갖는다. 이로 인해 만족가능성 처리기는 많은 시간을 필요로 한다.

다음은 루빅 큐브의 경우의 수를 줄여서 이 문제를 푸는 시간을 단축시키기 위해 추가된 제약조건들이다.

- 이전 상태와 같은 상태로 돌아가는 행위는 선택에서 제외한다.
- 시계방향으로 180 도와 반 시계방향으로 180 도는 같은 결과를 갖기 때문에 한 방향은 제거한다.
- 시계방향으로 270 도와 반 시계방향으로 90 도는 같은 결과를 갖기 때문에 한 행위가 세 번 연속 일어나는 경우를 제거된다.

커멘더 변수를 사용해서 루빅 큐브에 적용했을 때와 제약조건이 추가되었을 때의 실험 결과를 5 장에서 알아보도록 하겠다.

5. 실험결과

루빅 큐브를 풀기 위해 *Naïve* 인코딩과 커멘더 인코딩을 이용해 보았으며 또한 새로운 제약조건을 추가해 보았다. 실험 환경은 윈도우 XP 에서 3GHz 의 펜티엄 D 프로세서를 사용했고 메모리 용량은 2GHz 였다. 만족가능성 처리기로는 minisat 1.14 버전을 사용했다[3]. 실험을 위해 제한 시간을 600 초로 정했다. 제한 시간 내에 답을 얻지 못한 경우에 타임아웃 처리를 했고, 여기에서 언급된 시간은 SAT 처리기에 소요된 시간만을 의미할 뿐 인코딩에 소요된 시간은 제외 시켰다. 인코딩은 사용된 자료 구조와 사용된 언어에 따라서 실행 시간이 매우 다르기 때문이다. 이러한 이유로 대부분의 SAT 관련 논문에서는 평가 기준으로 SAT 실행 시간을 이용한다. 아래 표 2 에서 보는 바와 같이 *Naïve* 인코딩 방법에 비해 커멘더 인코딩 방법이 실행 시간 줄어들었음을 알 수 있으며, 제약조건이 추가되어서 더 짧은 시간내에 결과를 구한 것을 확인할 수 있다.

<표 2> 스텝에 따른 만족가능성 처리기 풀이 시간

	NAÏVE			COMMANDER		
step(k)	Non	Rule2,3	Rule1,2,3	Non	Rule2,3	Rule1,2,3
2	0.015	0.031	0.031	0.031	0.031	0.031
3	0.046	0.031	0.042	0.046	0.031	0.046
4	0.093	0.062	0.078	0.062	0.046	0.062
5	0.234	0.218	0.187	0.171	0.125	0.25
6	0.515	1.312	0.375	0.203	0.375	0.343
7	3.515	4.218	4.843	5.64	1.031	11.187
8	0.39	0.109	0.125	0.64	0.093	0.437
9	9.312	7.781	3.343	1.593	14.25	1.734
10	869.046	307.078	1153.72	624.203	89.062	4.578
11	18.218	5.562	37.671	3.875	2.437	2.171
12	23.859	44.343	17.984	15.812	5.062	166.39
13	1.843	0.734	4.984	4.296	0.765	5.812
Avg.	77.257	30.956	101.948	54.714	9.442	16.086

6. 결론 및 향후 연구 과제

최근 만족가능성 처리기의 성능이 향상되며 많은 문제들이 SAT 문제로 간주되고 있다. 많은 사례 중 퍼즐의 일종인 루빅 큐브를 SAT 문제로 다룬 사례가 없기에 우리는 이를 SAT 문제로 간주하고 CNF 로 인코딩했다. 루빅 큐브는 SAT 문제 중에서도 SAT Plan 문제로 초기상태에서 목표상태로 도달하는 경로를 찾는 문제이다. 우리는 일반적인 방법으로 루빅 큐브를 인코딩 했지만, 기존의 인코딩 방법으로는 루빅 큐브가 복잡해 질수록 너무 많은 시간이 소요되는 문제가 있음을 확인했다. 이런 문제를 해결하기 위해 여러 개 중에 한 개를 선택하는 문제에 강한 커멘더 인코딩 방법을 적용시켜 기존 인코딩에 비해 약간의 성능이 개선되었음을 확인할 수 있었다. 또한 새로운 제약조건을 넣어서 성능을 더 개선하였다. 향후 연구로는 기존의 인코딩 방법으로는 13 스텝 이상을 해결하지 못하는 위 문제를 해결하기 위해 새로운 방법이 적용하는 것으로 Planning 문제에 많이 쓰이는 휴리스틱 탐색 방법을 CNF 논리식에 적용시키는 것을 하나의 대안으로 보고 있다.

참고문헌

[1] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman and Y. Zhu, “Bounded Model Checking”, Advances in Computers, Vol.58, 2003

[2] http://www.rubiks.com

[3] N. Een and N. Sorensson, “An Extensible SAT-solver,” in proceedings of SAT’03, 2003

[4] 권기현, “SAT 처리기를 위한 수도쿠 퍼즐의 최적화된 인코딩”, 정보과학회논문지: 소프트 웨어 응용 제 34 권 제 7 호(2007.7)

[5] H. A. kauz, D. McAllester and B. Selman, “Encoing Plans in Propsitional Logic,” in the Proceedings of Principle of Knowledge Repres-entation and Reasoning, pp.374-384, 1996.

[6] G. Kwon and W. Klieber, “Efficient CNF Encoding for Selecting 1 from N Objects”, In the Proceeding of CFV07, 2007.