이진 프로그램을 위한 CTL 모델 체킹*

이태훈 권기현
경기대학교 정보과학부
{taehoon,khkwon}@kyonggi.ac.kr

CTL Model Checking for Boolean Programs

Taehoon Lee , Gihwon Kwon
Computer Science Department, Kyonggi University

요 약

마이크로소프트의 SLAM 프로젝트는 C 코드를 입력 받아서 프로그램이 주어진 속성을 만족하는지를 자동으로 검사한다. 그 결과, 실제 윈도우 2000에 탑재된 디바이스 드라이버의 버그를 찾아낼 수 있었다. 그러나 SLAM에서 검사할 수 있는 속성은 안전성 속성에만 국한되었다. 사용자가 검사하기를 원하는 속성은 안전성뿐만이 아니라 궁극성도 있지만, SLAM에서는 궁극성을 검사하지 못한다. 본 논문에서는 주어진 프로그램을 이진 프로그램으로 추상화 한 후에, 이진 프로그램에 대해서 CTL 모델 체킹을 수행하는 방법을 제안하였다. 그 결과 SLAM 보다도 더 많은 속성을 검사할 수 있다.

1. 서 론

모델 체킹은 시스템이 가지고 있는 잠재적인 오류를 찾기 위한 기술이다. 모델 체킹은 유한 상태 모델 M과 시제 논리식ϕ를 받아서 만족성 관계 $M \models \phi$를 결정한다[1]. 모델이 갖는 상태 공간을 철저히 탐색하면서 만족성 여부를 결정하기 때문에, 테스팅이나 시뮬레이션으로는 찾을 수 없는 난해한 버그를 찾을 수 있다. 그리고 사용자의 개입 없이 자동으로 검증 과정이 진행되기 때문에, 정리 증명에 비해서 빠르다. 또한 모델이 논리식을 만족하지 않는 경우, 그 이유를 담은 반례를 제공함으로서 모델의 디버그 작업을 돕는다[2]. 이와 같은 이점들 때문에 모델 체킹은 지난 20년 동안 하드웨어 검증, 소프트웨어 검증, 프로토콜 검증등 다양한 분야에 적용되어 왔다.

그러나 모델 체킹이 소프트웨어 검증에는 아직까지 널리 사용되지 못하고 있다. 대부분의 모델 체킹 도구는 유한개의 상태만을 다룰 수 있는데 비해서, 프로그램의 상태 공간은 일반적으로 무한하다. 그러므로 현재의 모델 체킹 기술을 적용해서 프로그램을 검증하기는 힘들다. 유한 상태 공간만을 취급하는 모델 체킹 도구로 프로그램을 검증하기 위해서는, 무한 상태 공간을 유한 상태 공간으로 변환해야 한다. 술어 추상화는(Predicate Abstraction)[3] 무한 상태 공간을 유한 상태 공간으로 축소하는 기술이다. 술어 추상화를 기반으로 한 프로그램 코드 검증 도구들이 많이 연구되어 왔다. 대표적인 연구는 Bandera[4], SLAM[5], BLAST[6], MAGIC[7] 등이 있다. Bandera 는 캔사스 대학에서 개발 중인 자바 검증 도구이다. SLAM 은 마이크로 소프트에서 개발 중인 C 검증 도구이다. MAGIC는 CMU에서 개발 중인 C 검증 도구이다. 이중 SLAM은 실제 윈도우에서 사용 중인 디아비스 드라이버의 오류를 찾아내는 등 실제 소프트웨어 검증에 사용되고 있다.

SLAM에서 검증은 3단계로 나누어 진다. 첫 번째로 C 프로그램의 추상화된 이진 프로그램으로 추상화 시킨다. 이진 프로그램은 원래 C 프로그램의 제어 흐름을 가지고 있는 이진 변수만을 가지고 있는 프로그램이다. 이후 이진 프로그램에서 모델 체킹을 수행한다. 만일 검사결과 참이 나왔다면 원래 C 프로그램에서도 참이다. 하지만 만일 거짓이 나왔다면 원래 C 프로그램에서는 어떤 결과가 나올지 알 수 없다.

시스템에서 검증해야하는 속성에는 크게 안전성 속성(Safety Property)과 궁극성 속성(Liveness Property)이 있다. 안전성 속성은 시스템이 실행 중에 일어나서는 안 되는 일을 의미한다. 안전성 속성의 대표적인 예는 "데드락이 일어나선 안된다" 등이 있다. 궁극성 속성은 시스템이 종료되기 전에 언젠가는 발생해야 하는 일을 의미한다. 궁극성 속성의 대표적인 예는 "메시지를 보내면 언젠가는 응답을 받아야한다" 등이 있다. 이 두 가지 속성은 시스템이 안전하다는 것을 보장하기위한 중요한 속성들이다. 하지만 SLAM에서는 검증할 수 있는 속성이 안전성 속성에 대해서만 검사가능하다. 따라서 궁극성 속성에 대해서도 검증할수 있는 방법이 필요하다.

본 논문에서는 SLAM의 추상화 모델인 이진 프로그램에서 안전성 속성과 궁극성 속성을 검증할 수 있는 방법을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 이진 프로그램

* 본 연구는 한국과학재단 특정기초연구 (R01-2005-000-11120-0)지원으로 수행되었음.

과 CTL 모델 체킹 방법에 대해서 설명한다. 3장에서는 이진프로그램을 위한 CTL 모델 체킹 기법에 대해서 설명한다. 마지막으로 4장에서 결론을 맺는다.

2. CTL 모델 체킹과 이진 프로그램

모델 M의 모든 초기 상태에서 CTL 식 ϕ가 참인 경우를 $M \models \phi$로 표시하며 "M이 ϕ를 만족한다"라고 읽는다. 모델 검사는 M과 ϕ를 받아서 "M이 ϕ를 만족하는지를 결정"하는 문제이다. 이를 위해서 ϕ를 만족하는 상태들의 집합을 구한 후, 이 상태 집합에 초기 상태가 포함되어 있는지를 검사한다. CTL 식 ϕ를 만족하는 상태들의 집합을 $\lfloor \phi \rfloor$라고 표시하자. 모델 검사 알고리즘의 핵심은 상태 집합 $\lfloor \phi \rfloor$를 구한 후 초기 상태 집합이 $\lfloor \phi \rfloor$의 부분 집합인 것을 검사하는 것이다. 즉,

$$M \models \phi \quad \text{iff} \quad I \subseteq \lfloor \phi \rfloor$$

이다. 만족성을 검사하기 위해서는 모델이 갖는 상태 공간을 탐색해야 하는데, 여기에는 2가지 탐색 방법이 있다. 초기 상태에서 출발해서 목표 상태를 찾아나가는 정방향 탐색과 목표 상태에서 출발해서 거꾸로 초기 상태를 찾아가는 역 방향 탐색 방법이 있다. 다음과 같이

$$pre_{\exists}(Q) = \{s \in S \mid \exists_{s' \in Q} \cdot (s,s') \in R\}$$
$$post_{\exists}(Q) = \{s' \in S \mid \exists_{s \in Q} \cdot (s,s') \in R\}$$

함수 $pre_{\exists}$는 집합 Q로 도달할 수 있는 이전 상태들의 집합을 역방향으로 구하며, $post_{\exists}$는 현재 상태 Q로 부터 도달 가능한 다음 상태들의 집합을 정방향으로 찾는다. 모델 검사의 핵심 부분인 ϕ를 만족하는 상태 집합 $\lfloor \phi \rfloor$은 다음과 같이 구한다.

$$\lfloor p \rfloor = \{s \in S \mid p \in L(s)\}$$
$$\lfloor \bot \rfloor = \varnothing$$
$$\lfloor \neg\phi \rfloor = S \setminus \lfloor \phi \rfloor$$
$$\lfloor \phi_1 \wedge \phi_2 \rfloor = \lfloor \phi_1 \rfloor \cap \lfloor \phi_2 \rfloor$$
$$\lfloor \mathrm{EX}\phi \rfloor = pre_{\exists}(\lfloor \phi \rfloor)$$
$$\lfloor \mathrm{EF}\phi \rfloor = \mu \mathrm{Z}.(\lfloor \phi \rfloor \cup pre_{\exists}(\mathrm{Z}))$$
$$\lfloor \mathrm{EF}\phi \rfloor = \nu \mathrm{Z}.(\lfloor \phi \rfloor \cap pre_{\exists}(\mathrm{Z}))$$
$$\lfloor \mathrm{E}(\phi_1 \mathrm{U} \phi_2) \rfloor = \mu \mathrm{Z}.(\lfloor \phi_2 \rfloor \cup (\lfloor \phi_1 \rfloor) \cap pre_{\exists}(\mathrm{Z}))$$

여기서 μ,ν는 각각 최소 고정점과 최대 고정점이다[8]. 상태 집합 $\lfloor \phi \rfloor$을 계산할 때, 최소 고정점 μ는 공집합을 초기값으로 하여 계속해서 증가되다가 더 이상 증가하지 않는 집합을 구할 때 사용하며, 반대로 최대 고정점 ν은 전체 집합을 초기 값으로 하여 계속해서 감소하다가 더 이상 감소하지 않는 집합을 계산할 때 사용한다.

이진 프로그램은 SLAM에서 처음 제안한 프로그램이다. 원래의 C 프로그램을 바로 모델 체킹을 하기는 힘들기 때문에 C 프로그램을 이진 프로그램으로 변환한다. 이진 프로그램은 이진 변수만 가지고 있는 프로그램이다. 추상화된 프로그램은 원래 C 프로그램의 제어흐름을 포함하고 있다. 따라서 추상화된 이진 프로그램에서 검사를 수행했을 경우 원래 프로그램에서도 참이 나온다. 하지만 추상화된 프로그램에서 거짓이 나왔을 경우 원래 프로그램의 결과는 알 수 없다. 이 경우 추상화 단계를 좀더 개선하여 다시 검사를 수행하게 된다.

3. 이진 프로그램을 위한 CTL 모델 체킹 기법

이진 프로그램을 위한 CTL 모델 체킹을 위해, 이진 프로그램을 다음과 같이 모델링 한다. 이진 프로그램에서 하나의 상태에 대한 정의는 다음과 같다.

$$bp = (pc, VALUE)$$

여기서 pc 는 프로그램의 위치를 의미한다. 그리고 VALUE는 이진 변수들의 가능한 값들의 집합이다.

상태들의 전의는 다음과 같이 표현 가능하다.

$$T_{pc} = pc \times pc$$
$$T_{val} = VALUE \times VALUE$$

예를 들어, 다음과 같은 이진 프로그램이 있을 때

```
1: main(){
2: boolean a,b,c;
3: a=true;  b=false;  c=true;
4: L1:
5: b= a?false : true;
6: c= a?true : *;
7: if(b==false){a=!a; goto L1; }
8: }
```

우선, 도달 가능한 상태들을 얻을 수 있다.

(2,{(true,true,true)}),
(3,{(true,false,true)}),
(4,{(true,false,true),(false,false,true)})
(5,{(true,false,true),(false,true,true)})
(6,{(true,false,true),(false,true,true),(false,true,false)})
(7,{(true,false,true),(false,true,true),(false,true,false)})
(8,{(false,true,true),(false,true,false)})

도달 가능한 상태는 초기 상태부터 상태가 더 이상 추가되지 않을 때까지 탐색을 하여 이진 프로그램에서 도달 가능한 상태들을 찾는다. 만일 프로그램이 종료되었다면 전의 관계에서 현재 상태에 그대로 머무르고 있다고 표시한다.

2장의 CTL 모델 체킹 알고리즘을 살펴보게 되면 모델 체킹을 수행하기 위해서 $pre_{\exists}$ 연산자를 이진 프로그램에 알맞게 재정의 한다.

$$pre_{\exists}(i,j) = \{(x,y) | (x,i) \in T_{pc} \wedge (y,j) \in T_{val} \wedge (x,y) \in R\}$$

여기서 R 은 도달 가능한 상태들의 집합이다. 이렇게 재 정의된 $pre_{\exists}$ 연산자를 통하여 이진 프로그램에 대한 CTL 모델 체킹을 수행할 수 있다.

4. 결론

하드웨어 혹은 시스템의 모델의 오류를 찾는데 사용되었던 모델 체킹을 실제 시스템의 소스 코드를 검증하는데 사용하는 연구가 활발히 진행 중이다. 그중 대표적인 연구는 SLAM 프로젝트 이다. 사례연구로서 실제 윈도우 2000에서 사용중인 디바이스 드라이버의 오류를 찾아내었고 다음버전의 DDK에서 공개할 예정이다. 하지만 SLAM에서 검증할 수 있는 속성은 안전성 속성만을 검증할 수 있다. 따라서 본 논문에서는 안전성 속성 뿐만 아니라 궁극성 속성도 검증할 수 있도록 CTL 모델 체킹을 수행할 수 있는 방법을 제안한다. 이를 통해서 SLAM 와 같은 소프트웨어 모델 체킹 도구에서 검증할 수 없었던 속성도 검증 할 수 있다.

속성을 명세 하는 것은 원래의 C 프로그램을 대상으로 하지만 본 논문의 예제에서 추상화된 이진 프로그램을 대상으로 CTL로 속성을 기술 하였다. 따라서 원래 C 프로그램을 대상으로 CTL 속성을 기술하고 그것을 추상화된 이진 프로그램을 대상으로 하도록 변환하는 과정이 필요하다. 또한 이 방법을 이용해서 다양한 프로그램의 검증을 시도해야 한다.

참 고 문 헌

[1] E.M. Clarke, O.Grumberg and D.A. Peled Model Checking, The MIT Press, 1999.

[2] E.M. Clarke, O. Grumberg, K.L. McMillan and X. Zhao, "Efficient Generation of Counterexamples and Witness in Symbolic Model Checking," in Proceedings of Design Automation Conference, pp.427-432, 1995.

[3] S. Graf and H. Saidi, "Construction of Abstraction State Graphs with PVS," in Proceedings of Computer Aided Verification, pp.72-83, 1997.

[4] J. Corbett, et.al., "Bandera: Extracting Finite-state Models from Java Source Code," in Proceedings of International Conference Software Engineering, 2000.

[5] T. Bal, R. Majumdar, T. Millstein and S.K. Rajamani, "Automatic Predicate Abstraction of C programs," SIGPLAN Notices, Vol. 36, No.5, pp.203-213, 2001.

[6] T.A. Henzinger, R. Jhala, R. Majumdar and G. Sutre, "Lazy Abstraction," in Proceedings of Principles of Programming Languages, pp.58-70, 2002.

[7] S. Charki, E.M. Clarke, A. Groce, S. Jha and H. Veith, "Modular Verification of Software Components in C," IEEE Transactions on Software Engineering, Vol.30, No.6, pp.388-402, 2004.

[8] 이태훈, 권기현, "프로그램 모델 체킹을 위한 술어 추상화 기법의 비교", 2004년 소프트웨어 공학 합동 워크샵 발표자료, pp.193-199, 2004.

[9] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking,"In the Proceedings of CAV'02, 2002.

[10] T. Ball and S.K. Rajamani, "Generating Abstract Explanations of Spurious Counterexamples in C programs," Technical Report MSR-TR-2002-09, 2002.

[11] E.M. Clarke, O.grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," in Proceedings of Computer Aided Verification, pp154-169. 2000.