반례를 이용한 프로그램의 오류 원인 탐지 기법

신모범°, 방호정, 김태효, 차성덕
한국과학기술원
{mbshin°, hjbang, taihyo, cha}@dependable.kaist.ac.kr

Isolating Cause of Error in a Counterexample

Mo Bum, Shin°, Ho Jung Bang, Tai Hyo Kim, Cha Sung Deok
Div. of Computer Science, Dept. of EECS, KAIST and AITRC/IITRTC/SPIC

요 약

모델 체킹(model checking)은 자동으로 소프트웨어의 속성을 검증하는 기법으로 그 필요성이 꾸준히 증가하고 있다. 시스템이 특정 속성(property)을 만족하지 않는 경우 모델 체커는 반례(counterexample)를 생성하게 된다. 반례는 오류가 발생한 원인을 담고 있는 정보로서 오류를 이해하고 수정하는 작업에 많은 도움을 준다. 하지만 반례가 너무 길거나 이해하기 어려운 경우에는 분석에 많은 시간과 자원이 소요되기도 한다. 따라서 자동적으로 반례 안의 오류를 찾아내고 설명을 제공하는 기법의 필요성이 대두되고 있다. 본 논문에서는 추상모델(abstract model)에서 생성된 반례의 오류의 원인을 밝히는 자동화 기법을 제시한다.

1. 서 론

모델체킹(model checking)은 자동으로 소프트웨어의 속성(property)을 검증하는 정형 검증 기법 중 하나이다.[1] 최근에는 소프트웨어의 안정성을 중요하게 생각하는 시스템이 늘고 있기 때문에, 모델체킹의 사용이 꾸준히 증가하고 있다. 모델체킹은 다른 분석 기법과는 달리 사용자가 입력해야 하는 정보의 양이 상대적으로 적은 편이며, 소프트웨어의 오류가 발생할 경우에 반례(counterexample)를 생성하여 효과적으로 오류를 찾아낼 수 있다.

모델체킹 검증기에서 생성된 반례를 분석하면 오류의 찾아내어 시스템의 오류 수정을 할 수 있다. 하지만 반례는 프로그램이 오류 상태(error state)로 도달하는 중에 방문하게 되는 상태(state)만 나열하고 있다. 따라서 반례가 길거나 분석하기에 어려운 경우에는 분석에 상당한 시간이 필요하다. 이러한 어려움을 감소시키기 위해서 반례를 효과적으로 분석하는 자동화 기법이 요구되고 있다.

오류의 분석 기법은 크게 오류설명(error explanation)과 오류국지화(fault localization)으로 구분된다. 오류 국지는 반례에서 오류가 발생한 지점을 찾아내는 기법이다. 오류설명기법은 오류가 발생한 이유를 설명하는 기법으로서 성공적으로 실행된 경우와 반례를 비교하여 그 원인을 설명하고 있다. 본 논문에서는 프로그램의 오류를 찾는 오류국지화 기법에 초점을 두고 있으며 술어추상화(Predicate Abstraction)를 사용한 모델의 반례를 대상으로 하고 있다.

추상화된 모델의 반례는 프로그램이 성공적으로 실행한 경우의 상태를 포함하고 있다. 모델체킹 알고리즘은

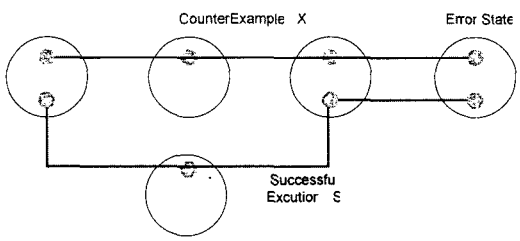

그림1: 추상화된 모델에서의 반례

대상의 상태가 많은 경우 상태 폭발(state explosion)을 일으켜 검증이 불가능한 경우가 있다. 이런 문제점을 극복하기 위해서 추상화(abstraction)기법이 사용된다. 술어(predicate)의 집합으로 상태를 표시하며 주어진 진리값(truth assignments)에 의해서 상태가 구분된다. 위의 그림 1에서는 추상화된 모델에서의 반례를 보여주고 있다. 큰 원은 추상상태(abstract state)이며 작은 원은 실제 상태(concrete state)를 의미한다. 반례 X는 오류 상태에 도달하나 추상화된 상태이기 때문에 오류가 없는 상태를 포함하고 있다. 따라서 오류를 일으키지 않으면서 같은 오류 상태를 지나는 trace를 구할 수 있으며 이 정보를 이용하여 반례에서 오류를 원인을 찾을 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 이전에 연구되었던 오류의 원인을 찾는 기법을 소개하고 3장에서는 본 논문에서 제안한 기법을 기술한다. 마지막으로 4장에서는 결론을 내린다.

2. 관련연구

이 장에서는 모델 체커에서 생성된 반례에서 오류의 원인을 찾아내는 여러 기법에 대해서 설명한다.

2.1 오류의 국지화(Error Localization)

오류가 없는 프로그램의 경로(correct trace)를 찾아 반례의 오류를 찾는 기법이다. 모델체킹 알고리즘이 수행 중에 저장하고 있는 정보를 사용하여 오류가 없는 경로를 빠르게 찾을 수 있으며 또한 실행 중에 다른 원인을 가지는 여러 반례를 생성할 수 있다. [2]

2.2 거리 메트릭을 이용한 오류의 설명

반례에 가장 가까운 성공적인 실행(successful execution)을 찾아내어 오류에 대한 정보를 제공하는 기법이다. Lewis의 우연적인 종속(casual dependency) [3] 개념과 거리 메트릭(distance metric)을 도입하여 오류와 거장 가까운 실행을 찾아내었다. 이를 반례와 비교하여 프로그램 상의 오류의 위치를 발견하고 오류에 관한 자세한 정보를 제공한다.

```
#include <assert.h>
main()
{
  SKIP  int x, y;
  L0  x = 1;
      y = 1;
  L1  if ( x == y )
  L2      y = 1;
  L3  else
  L4      y = 2;
  L5  assert( y == 2 );
}
```

예제 1: 프로그램의 안전속성 확인

3. 오류의 원인 계산

이 장에서는 반례에서 오류의 원인을 찾는 기법을 제안한다.

3.1 반례의 추상 모델 구축

오류의 원인을 찾기 위해선 우선 반례를 추상상태(abstract state)로 구성된 CFA(Control Flow Automata)로 변환한다.

CFA 다름과 같이 정의된다.

CFA A= (Q_{CF}, I_{CF}, T_{CF}, L_{CF})

Q_{CF}: 반례를 구성하는 상태의 유한 집합

I_{CF} : 의 유시작 상태(I_{CF} Q_{CF})

T_{CF}:

L_{CF}:

반례 CFA의 edge에는 기본블럭(basic block)과 프로그램의 분기점에 나타나있는 술어가(predicate)가 표시되게 되며 각 상태는 추상화된 상태, 즉 여러 상태를 대표하는 하나의 상태로 표기된다.

3 .2 오류 원인 계산

그림 1의 예제에서는 프로그램의 종료 시점에서 y 변수의 값이 2를 가지고 있는지 검사하고 있다. x의 값과 y의 값이 다른 경우에는 else 문으로 분기가 일어나서 정상적으로 프로그램이 종료한다. 하지만 위의 예제에서는 x와 y의 값이 같기 때문에 if문으로 분기가 일어나 L5에 기술되어 있는 안정속성(safety property)을 만족시키지 못한다. BLAST 모델체커[4]로 위의 프로그램을 실행하면 다음과 같은 반례가 생성된다.

```
L5  Pred(y@main  !=  2) :: 12
L2  Block(y@main = 1 ) :: 12
L1  Pred(x@main  ==  y@mair) :: 8
L0  Block(x@mair = 1 y@mair = 1 ) :: 7
-1 :: -1     Skip :: 5
0 :: 0       Block(Return(0) ) :: -1
0 :: 0       FunctionCall(__BLAST_initialize_rechable ()) :: -1
```

그림 1: BALST 검증기에서 출력된 반례

오류를 원인을 찾는 알고리즘에서는 WP(weakest precondition)[5]함수를 이용한다. C 프로그램의 할당문(assignment statement) v = e 을 a라하고 표현식(expression)을 ϕ 이라 할 때 WP(ϕ , a)는 ϕ 안에 모든 v를 e로 치환하여 구할 수 있다.

본 논문에서 제안한 기법은 프로그램이 속성을 위반하는 속성, 즉 assert문의 술어를 역(negation)으로 한 술어에서 시작한다. 반례의 전이(transition)을 따라서 시작 상태로 이동을 하면서 경로상의 상태에 WP 함수를 적용한다. 함수의 결과를 SAT solver를 사용하여 만족도(satisfiability)를 조사하는 도중 만족하지 못하는 상태(unsatifiable)에 도달한 경우 그 상태를 오류의 원인이라 한다.

CFA로 표현된 반례는 프로그램의 실제 상태를 표현하는 것이 아니라 추상화된 상태를 의미한다. 따라서 프로그램이 정상적으로 실행될 경우 즉 속성을 만족하는 경우에 지나게 되는 상태도 반례에서 속성을 위반하는 상태에 포함된다.

추상화된 상태에서의 반례를 α 라 할 때, 반례는 α_0, α_1, $\cdots\alpha_n$ 의 상태로 구성된다. 반례에서 최종 상태 α_n은 속성(Property) P를 만족하지 않는 상태 $\sim P(\alpha_n)$이다. α_n을 지나는 오류가 없는 경로가 반례와 가장 가깝다고 가정할 경우, 오류가 없는 경로 x, 즉 $x_0, x_1, \cdots x_n$에서 x_n은 속성을 만족하기 때문에 $P(x_n)$이다. x_n에 추상화 함수(abstract function)를 적용하면 그 결과값은 α_n이 된다. 따라서 α_n에서 속성을 역(negation)으로 한 술어를 적용하여 추상화를 수정(refine)하면 속성을 만족하지 않는 상태를 α_n에서 제거할 수 있다.

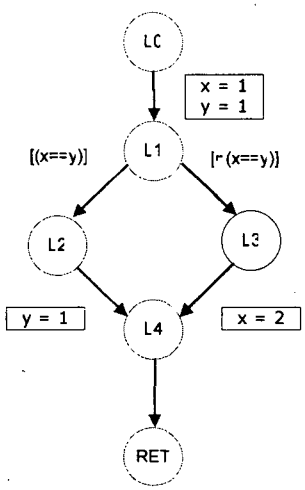

그림 3: 예제 1의 CFG

그림 3의 CFG는 예제1의 CFG로써 반례를 포함하고 있다. L0→L1→L2→L4→RET 경로를 지날 경우 RET 상태의 안전속성을 위반하게 된다. 위 프로그램의 오류가 없는 경로는 L0→L1→L3→L4→RET이다. 반례에서 오류가 없는 경로와 중복되는 상태를 제거하면 L2 만 남는다. 따라서 L2를 오류의 원인이라 할 수 있다. 본 논문에서 제안한 기법으로도 동일한 결과를 얻을 수 있다.

예제

오류의 원인을 밝히기 위해선 안전속성에 기술되어 있는 술어를 역으로 하여 시작한다. PRET부터 WP함수를 적용한 결과는 다음과 같다. P_{RET}={(y=!2)} P_{L4} = { WP((y!=2), y =1)} = {1 != 2}. 각 노드 WP 함수를 적용한 술어들을 SAT solver에 입력하면 오류의 원인을 찾을 수 있다. P_{L4}의 WP을 SAT solver에 입력한 결과는 UNSAT(unsatisfiable)이 된다. 따라서 L2를 오류의 원인이라 부른다.

제안된 기법으로 오류의 원인을 구한 경우 약간의 문제점이 있었다. 우선 와 끝 상태가 사용자가 원하는 속성을 만족시키기 위한 문장을 빠뜨린 경우에는 특정 상태가 오류의 원인이라 밝히지 못한다. 하지만 특정 영역을 오류의 원인이라 할 수 있기 때문에 큰 문제점은 아니라고 생각되었다. 다른 문제점은 *WP* 함수를 적용하여 구한 오류가 없는 경로가 시작 상태에서 도달할 수 있는가 라는 문제이다. (recheability) 이 문제는 추상 상태를 실제 상태로 변경하는 함수(concretize function)를 적용하며 경로의 첫 상태와 마지막 상태가 실제 프로그램의 존재하는지를 확인하면 해결될 것으로 예상된다.

4. 결론 및 향후 연구 과제

본 논문에서는 반례에서 오류의 원인을 찾아내는 기법을 제안하였다. 반례는 오류를 찾아내고 수정하기 위한 정보를 제공하지만 분석에 많은 시간이 들기도 한다. 제안된 기법을 사용하여 반례에서 오류의 원인을 쉽게 찾을 수 있었으며 또한 분석에 드는 시간과 노력을 줄일 수 있었다.

향후에는 오류의 원인과 관계가 없는 술어를 찾아 제거하는 기법을 개발할 여정이다. 어떤 표현식이 만족하지 않는 경우 SAT solver는 unsatisfiable core를 출력한다. Unsatisfiable core는 표현식이 만족하지 않는 이유를 출력함으로써 오류를 찾아내기 위해 검사해야 할 반례내의 술어 개수를 줄일 수 있을 것으로 예상된다. 또한 제안된 기법의 증명을 제공하여 타당성을 검증할 예정이다.

참고 문헌

[1] Edmund M. Clarke, Orna Grumberg, Doron A. Peled, " Model Checking" , MIT Press

[2] Ball T, Naik M, Rajamani S, " From symptom to cause: Localizing errors in counterexample traces" , ACM POPL 2003

[3] Alex Groce, Sagar Chaki, Daniel Kroeningm, Ofer Strichman, " Error explanation with distance metrics" , LNCS TACAS 2005

[4] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Gregoire Sutre " LAZY Abstraction" , ACM POPL 2002

[5] Thomas Ball, Sriram K. Rajamani, " Automatically Validating Temporal Safety Properties of Interfaces" , LNCS Spin workshop 2001