프로그램 합성 기법을 이용한 임베디드 소프트웨어의 모델 체킹 효율성 향상을 위한 사례 연구

김요엘, 최윤자
경북대학교 컴퓨터학부
kimyoel2305@gmail.com, yuchoi76@knu.ac.kr

A Case Study to Improve the Efficiency of Model Checking in Embedded Software Using Program Synthesis

Yoel Kim, Yunja Choi
School of Computer Science and Engineering, Kyungpook National University

요 약

C언어로 작성된 임베디드 소프트웨어를 엄밀히 검증하기 위해 모델 검증기 CBMC을 사용할 수 있으나, 소프트웨어가 복잡할수록 검증 효율성이 크게 감소하는 문제가 있었다. 이를 해결하기 위한 방법으로 프로그램 합성 기법을 사용해 검증 대상 함수를 좀 더 단순한 형태의 동일 행위를 갖는 함수로 합성함으로써 검증 시간을 단축하려는 시도가 있었으나, 합성 기법은 큰 사이즈의 배열을 가진 함수의 합성 정확도가 낮아 실제로 검증에 적용하기 어려웠다. 이를 해결하기 위해 본 연구에서는 배열 인덱스 계산식 및 배열 입력 범위 분석을 통해 검증에 필요 없는 입력 범위를 제거하여 합성 난이도를 줄였고, 기존보다 더 정확하면서 간결한 함수를 합성할 수 있었다. 이를 Object Follower 예제에 적용한 결과 CBMC 실행 시간이 10배 이상 단축되었으며, 검증 결과의 신뢰성을 높일 수 있었다.

1. 서 론

C언어로 작성된 임베디드 소프트웨어는 모델 검증기 CBMC(C Bounded Model Checker) [1]을 사용해 검증할 수 있으나, 큰 사이즈의 배열과 복잡한 실행 경로(반복문 등)가 포함된 소프트웨어의 경우 검증 효율성이 크게 감소하는 문제가 있었다 [2].

기존 연구 [3]에서는 프로그램 합성 기법을 통해 검증 대상 함수를 좀 더 단순한 형태의 동일 행위를 갖는 함수로 합성함으로써 검증 시간을 단축하려는 시도가 있었으나, 합성 기법은 큰 사이즈의 배열을 전역 변수로 사용하는 임베디드 소프트웨어의 특징 때문에 합성 결과의 정확도가 매우 낮아 실제 검증에 적용하기 어려웠다.

본 연구에서는 이러한 문제를 해결하기 위해 함수에서 사용되는 입력 배열의 인덱스 계산식과 입력 범위를 분석하여 검증에 필요 없는 배열 원소(element)를 제외하고, 전체 입력 범위를 대폭 감소시켜 합성 난이도를 줄임으로써 더 정확하고 간결한 함수를 합성하는 방법을 제시한다.

이를 위해선 먼저 프로그램 합성 기법에 대해 간단히 설명하고, 왜 검증에 도움이 되는지 설명한다. 또한 배열 인덱스 계산식 분석, 대상 함수의 스펙 생성 및 합성, CBMC 실행 및 반례 분석까지의 과정을 소개한다. 실험으로는 Object Follower [4]의 함수들을 합성한 결과와 이를 기반으로 CBMC을 실행했을 때의 검증 효율성과 그 영향을 보여준다.

2. 프로그램 합성 기법을 사용한 검증 효율 향상 방법

프로그램 합성 기법은 사용자가 원하는 프로그램의 스펙(Specification)을 입력으로 받아들여 주어진 스펙을 만족하는 프로그램을 찾아내는 기법으로, 대표적으로 입출력 예제를 스펙으로 받아들이는 Duet [5]이 존재한다. 또한 Duet은 SyGuS(Syntax-Guided Synthesis) [6]을 채택하는데, 이는 합성 스펙에 사용할 수 있는 문법을 제한하여 프로그램 탐색 공간을 효율적으로 줄여 합성 난이도를 감소시킬 수 있다.

합성 기법을 검증에 적용하면 기존 함수보다 정확도가 떨어지지만 복잡한 연산이나 변수를 덜 사용하는 함수로 대체하여 CBMC가 풀어내기 쉽도록 하며, 특히 반복문을 사용하지 않는(loop-free) 함수를 합성하여 unwinding을 줄일 수 있어 전체 검증 시간을 줄일 수 있을 것으로 기대된다.

하지만 후술할 표 1을 보면 큰 사이즈의 배열을 접근하는 함수의 경우 합성 정확도가 매우 감소하여 검증에 적용하기 어려운 문제가 발생했다. 본 연구에서는 배열 인덱스 계산식 분석 및 입력 범위 분석을 통해 검증에 사용될 필요가 없는 배열 원소를 제외함으로써 이 문제를 해결하고자 한다.

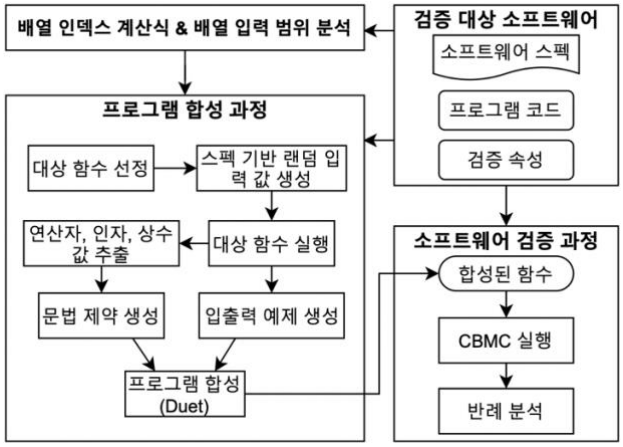

그림 1 대상 소프트웨어의 전체 검증 과정.

그림 1은 대상 소프트웨어의 구체적인 합성 및 검증 과정을 보여준다. 대상 함수를 선정하고, 유효한 입력 범위 내에서 랜덤으로 입력값을 생성하여 대상 함수를 실행하여 입출력 예제를 생성한다. 또한 함수 실행 중에 수행된 문장에서 사용된 연산자, 인자, 상수 값을 추출한다. 이는 SyGuS 합성 스펙 중 문법 제약을 생성하는 데 사용된다. 입출력 예제와 문법 제약을 하나로 묶어 합성 스펙을 생성하고 이를 Duet에 전달하여 대상 함수를 합성한다. 합성된 함수는 기존 함수를 대체하며, 주어진 검증 속성에 대해 CBMC을 실행한다.

그러나 CBMC가 정상적으로 종료되었다고 해도 합성된 함수는 기존 함수와 비교했을 때 정확도가 떨어지기 때문에 검증 결과를 그대로 신뢰할 수 없다. 설령 측정된 정확도가 100%이라도 모든 입력값에 대한 정확도가 아니므로 역시 신뢰할 수 없다. 그러므로 반례에서 사용된 합성된 함수의 입력값에 대한 출력값이 기존 함수와 일치하는지 확인하고, 이러한 출력값이 전부 일치해야 검증 결과를 신뢰할 수 있다.

3. 배열 인덱스 계산식 및 배열 입력 범위 분석

배열 인덱스 계산식 분석은 기본적으로 [7]에서 설명한 프로그램 슬라이싱(slicing) 개념을 사용한다. 다음은 Object Follower 함수의 사례로 설명한다.

Object Follower [4]은 카메라로 포착한 물체를 따라가는 로봇 자동차이다. Object Follower는 크게 카메라에서 측정된 물체와의 거리를 계산하는 함수와 자동차의 속도와 방향을 제어하는 함수로 구성된다. 그림 2는 Object Follower의 함수들을 호출하는 코드(라인 4~9)와 호출되는 함수 중 하나인 getX 함수의 코드(라인 1~3)을 나타낸다. 전체 코드는 [8]에서 확인할 수 있다.

코드의 의미를 설명하자면 먼저 nxtcamdata 배열은 외부 센서의 입력값으로, 주어진 소프트웨어 스펙에 따라 배열 원소별로 0~8, 0~88, 또는 0~144 사이의 임의의 값을 가질 수 있다. nxtcamdata는 최대 8개의 물체를 포착할 수 있으며, getbiggestrect 함수를 통해 포착한 물체 중 가장 큰(가까운) 물체의 데이터를 담고 있는 인덱스(rectindex)을 반환하고, 해당 인덱스가 가리키는 nxtcamdata의 데이터를 가지고 여러 과정(getArea, median_filter, fisqrt, getX 함수)을 거쳐 라인 8, 9의 size와 x가 자동차의 속도와 방향을 계산할 때 사용된다 (자동차의 속도와 방향은 그대로 검증 속성의 변수가 된다).

다음은 getX 함수에 대한 배열 인덱스 계산식 및 배열 입력 범위 분석 과정으로, getX 함수에서 같은 의미로 사용되는 nxtcamdata 배열 원소들을 분류 및 분석할 수 있다.

1. 먼저 합성 대상인 getX 함수의 return 문장을 사용하는 라인을 찾는다. 라인 3이 해당된다.
2. 라인 3에서 사용(reference)되는 변수(xlr, xul)들을 정의(define)하는 문장을 찾는다. 각각 라인 1, 2가 해당된다.
3. 라인 1, 2에서 사용되는 배열의 인덱스 계산에 사용되는 변수 (rectindex)가 그 외의 문장(출력값 계산, 조건식 등)에 사용되지 않아야 하므로, 이를 확인한다.
4. 3을 만족한다면 rectindex의 유효한 입력 범위(0~7)에 대해 실제로 접근된 인덱스를 계산한다.

```
 static int nxtcamdata[41];
 int getX(int rectindex) {
(1)  int xul = nxtcamdata[5 * rectindex + 2];
(2)  int xlr = nxtcamdata[5 * rectindex + 4];
(3)  return (xlr + xul) / 2;
   }
-----------------------------------------
(4)  rectindex = getbiggestrect(0, -1);
(5)  if (rectindex >= 0) {
(6)    area = getArea(rectindex);
(7)    area = median_filter(area);
(8)    size = fisqrt(area);
(9)    x = getX(rectindex);
     }
```

그림 2 Object Follower 함수 실행 코드.

5. getX 함수에 대한 rectindex, nxtcamdata의 입력 범위는 소프트웨어 스펙을 통해 알 수 있다고 가정한다.
6. 라인 1에서 모든 rectindex 값에 대해 인덱스 계산식을 실행한 결과 {2,7,12,…,37}가 접근되었고, 이 때 각 배열 원소의 입력 범위가 모두 0~88으로 동일함을 확인할 수 있다. 라인 2도 같은 방법으로 라인 2에서 접근된 배열 원소들의 입력 범위가 모두 동일함을 확인한다.
7. 6을 만족하는 경우, 어느 rectindex 값을 사용해도 접근된 배열 원소에서 동일한 입력 범위를 가지고 있으므로, getX 함수 역시 동일한 출력 범위를 가진다. 따라서 rectindex의 유효한 입력 값 중 하나(예: 0)만 사용해도 검증하기에 충분하다고 판단할 수 있다.
8. getX 함수를 호출한 라인(9)을 기준으로, rectindex을 가장 최근에 정의하는 라인을 찾아 rectindex = 0;으로 대체한다. 여기서는 라인 4의 getbiggestrect 함수 호출이 대체된다.
9. 마지막으로 입력 범위가 감소된 getX 함수에 대한 랜덤 입력값을 생성할 수 있으며, 이를 실행했을 때 사용되지 않는 배열 원소 39개를 합성 스펙에서 제외할 수 있다.

4. 실 험

4.1 함수 합성 결과

표 1 Object Follower 함수에 대한 합성 결과 비교.

대상 함수	전체 입력 범위		인덱스 계산식 분석 후	
	정확도	사이즈	정확도	사이즈
getX	1.5%	50	**100%**	**5**
getArea	1.1%	94	**100%**	**28**
getbiggestrect	**64.7%**	**16**	**return 0으로 대체**	
median_filter	21.8%	26	N/A	
fisqrt	9.7%	82	N/A	

표 1에서는 Duet을 사용하여 함수의 전체 입력 범위를 사용했을 때와, 인덱스 계산식 분석을 사용했을 때의 합성 결과를 보여준다. 실행 옵션은 max_size 128, init_comp_size 3으로 설정하였다. 입출력 예제 10개씩 랜덤으로 선택하여 10번씩 합성한 것 중에 가장 정확도가 높은 것을 선택하였다. 입출력 예제를 10개만 선택한 이유는 합성 결과의 사이즈를 최대한 줄여서 검증 시간을 최소화하기 위함이다. 정확도는 합성에 사용되지 않은 입출력 예제 1000개로 측정하였다.

표 2 Object Follower에 대한 속성별 CBMC 실행 결과.

속성	(1) 기존 함수		(2) getbiggestrect 함수만 대체			(3) 배열 인덱스 계산식 분석 후		
	시간 (초)	검증 결과	시간(초)	검증 결과	신뢰 여부	시간 (초)	검증 결과	신뢰 여부
p1	10120	TRUE	5621	TRUE	NO (0/0)	**5325**	TRUE	NO (0/0)
p2	57536	FALSE	6246	FALSE	NO (0/1)	**5848**	FALSE	**YES (2/2)**
p3	9578	TRUE	5632	TRUE	NO (0/0)	**5331**	TRUE	NO (0/0)
p4	68143	FALSE	6135	FALSE	NO (0/1)	**5651**	FALSE	**YES (2/2)**
p5	66315	FALSE	6171	FALSE	NO (0/1)	**6106**	FALSE	**YES (2/2)**
p6	76757	FALSE	6633	FALSE	NO (0/1)	**5759**	FALSE	**YES (2/2)**

Duet은 배열 접근 연산을 지원하지 않기 때문에, getX와 getArea 함수의 경우 큰 사이즈의 배열을 접근하는 경우가 많아 정확도가 매우 낮았다. median_filter와 fisqrt 함수는 복잡한 계산이 필요하기 때문에 정확도가 낮았다. 따라서 64.7%의 정확도를 보이는 getbiggestrect 함수만 기존 함수를 대체하여 CBMC 실험에 사용하였다. 반면 인덱스 계산식 및 입력 범위 분석 후에 getX와 getArea 함수의 입력 범위를 줄일 수 있었고, 이를 기반으로 사이즈가 더 작으면서 정확도 100%인 함수를 합성할 수 있었다. 추가로 getbiggestrect가 호출되는 문장은 0으로 대체되었다. median_filter와 fisqrt 함수는 입력 배열을 사용하지 않아 합성 대상에서 제외되었다.

4.2 CBMC 실험 결과

표 2는 표 1의 합성 결과를 기반으로 각각 (1) 기존 함수를 그대로 사용한 경우, (2) 정확도 64.7%, 사이즈 16의 getbiggestrect 함수만 대체한 경우, (3) 배열 인덱스 계산식 분석 후 getX, getArea 함수를 대체하고 getbiggestrect 함수 호출을 0으로 대체한 경우에 대해 CBMC을 실행한 결과이다.

검증 속성은 대표적으로 '일정 이상의 속력일 때 급 회전을 하면 안 된다' (p1,p2), '급정거 시 특정 힘 이상으로 감속하면 안 된다' (p3)이 있으며, 다른 속성들도 동일한 속성 변수를 사용하나 조건이 조금씩만 다르도록 설정했다. 모든 실험은 Ubuntu 18.04 LTS, 3.3-GHz Intel Xeon W-2155 CPU, 256GB 메모리에서 진행되었으며, CBMC 버전은 5.11, unwind는 모든 거짓 속성이 반례를 생성할 수 있도록 120으로 설정했다.

실험 결과 (2), (3)의 경우 공통적으로 검증 시간이 (1)에 비해 검증 결과가 참일 때 약 2배, 검증 결과가 거짓일 때 약 10배 정도 감소한 효과가 나타났다. 반복적으로 getArea 함수를 호출하여 복잡한 getbiggestrect 함수를 대체했기 때문에 달성할 수 있는 결과였다. 하지만 bounded model checking의 한계로 인해 참 속성인 경우 (p1) 검증 결과를 그대로 신뢰할 수 없고, 거짓 속성이나 bound가 부족한 경우에도 (p3) 기존 함수의 결과와 동일했다.

검증 결과가 거짓일(반례가 생성될) 때의 신뢰 여부는 반례에서 합성된 함수들을 호출한 횟수와, 그 중에서 정확한 출력값을 반환한 횟수가 같을 때 신뢰할 수 있고, 그 외의 경우는 신뢰할 수 없다. 예를 들어 반례에서 합성된 함수를 2번 호출했고, 그 중에서 하나의 출력값만 정확했을 경우 표 2의 신뢰 여부는 NO (1/2)로 적을 수 있다.

(2)의 경우 정확도 64.7%의 함수임에도 불구하고, 모든 반례에서 부정확한 값을 출력하여 모든 속성에서 검증 결과를 신뢰할 수 없었다. 반면 (3)의 경우 단 하나의 인덱스만 선택했고, 해당 인덱스의 물체 데이터를 사용하는 모든 합성된 함수의 정확도가 매우 높았기 때문에 반례 분석 결과 거짓 속성의 모든 검증 결과를 신뢰할 수 있었다.

5. 결론 및 향후 연구

배열의 인덱스 계산식을 분석했을 때 검증에 필요 없는 배열 원소를 제외하고 입력 범위를 줄임으로써 합성 결과의 성능 향상이 이뤄졌고, 기존 함수를 대체한 결과 CBMC의 검증 시간은 기존에 비해 최대 10배 이상 단축되었다. 하지만 인덱스 분석은 Object Follower 사례와 같이 배열 원소들이 서로 같은 의미(입력 범위)를 가지고 있어야 하기 때문에 다른 사례에 적용하기엔 제한적이다. 또한 분석 과정이 정형화되지 않았기 때문에, 이를 일반화하고 다양하게 적용될 수 있는 분석 기법을 만드는 것이 향후 연구의 목표이다.

Acknowledgement

이 논문은 2016, 2021 년도 정부(교육부, 과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2016R1D1A3B01011685, NRF-2021R1A5A1021944).

참고 문헌

[1]. E. Clarke, et al., "A tool for checking ANSI-C programs." TACAS 2004.

[2]. L. Cordeiro, et al., "SMT-based bounded model checking for embedded ANSI-C software." *IEEE Transactions on Software Engineering* 38.4, 957-974, 2011.

[3]. Y. Kim and Y. Choi, "A Case Study on the Performance of Program Synthesis in Embedded Software Domain." KCSE 2021.

[4]. Object Following Robot, https://devpost.com/software/object-follower

[5]. W. Lee, "Combining the Top-down Propagation and Bottom-up Enumeration for Inductive Program Synthesis," *Proceeding of the ACM on Programming Languages*, 5, 1-28, 2021.

[6]. R. Alur, et al., "Syntax-Guided Synthesis," FMCAD 2013.

[7]. Y. Choi, et al., "Efficient safety checking for automotive operating systems using property-based slicing and constraint-based environment generation." *Science of Computer Programming*, 103, 51-70, 2015.

[8]. Object Follower, https://github.com/addud/object-follower