# CBMC를 이용한 트리플 DES의 검증

강미영°, 유희준, 최진영
고려대학교
{mykang°, hyoo, choi}@formal.korea.ac.kr

# The Verification Using CBMC about Triple DES

Miyoung Kang°, Hee-Jun Yoo, Jin-Young Choi
Dept of Computer Science & Engineering, Korea University

**요 약**

CBMC는 ANSI-C 프로그램과 베릴로그 서킷사이의 일치성을 검증하는 툴이다. 입력된 서킷과 코드를 CNF로 변환 과정에서 C 코드는 중첩 루프, pointer, dynamic memory allocation등에 대한 변환의 문제점이 있다. 본 논문에서는 CBMC에서 C 코드의 CNF로 변환하는 과정의 문제점들을 동일한 식(equation)의 변환 과정에 대하여 설명하고 상용적인 트리플 DES를 CBMC로 검증하는 과정을 제시한다.

## 1. 서 론

초기 컴퓨터 과학에서 정형 기법(formal methods)과 프로그램 검증(program verification)의 목적은 프로그램의 정확성(correct)을 명확하게 증명하는 기술을 제공하는 것이다. 그러나 이 목적은 전체적으로 비현실적으로 남겨진 반면에, 많은 연구가들은 덜 모호한 것에 초점을 두고 연구하고 있다. 프로그램의 행동(behavioral)의 부분적 명세을 기술하고 그 명세에 대한 정확성(correct)을 체크하기 위한 방법을 증명하는 것은 여전히 중요한 목적이다. 프로그램의 정확성에 대한 관심이 증가하고 소프트웨어 성능(quality)의 문제에 대한 연구 분야는 type checking, model checking, program analysis, 그리고 automated deduction의 기술적인 발전의 성과를 가져오게 되었다[1].

이러한 발전의 성과는 다양한 embedded 시스템을 검증 하기 위한 툴을 개발로 이어지고 특히 C 코드 검증의 방법으로 연구되어 지고 있다. C 코드 검증을 위한 툴들은 SLAM[2], BLAST[3], MAGIC[4], CBMC[5]등이 개발되어 사용되고 있다.

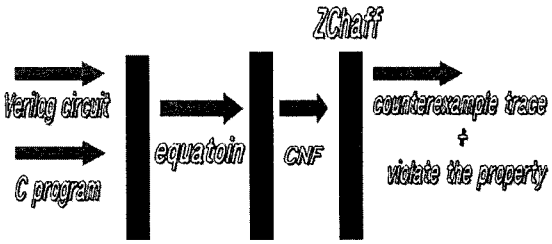
CBMC(Bounded Model Checking for ANCI-C)는 BMC(Bounded Model Checking)[6]을 이용하여 ANSI-C 프로그램과 베릴로그 서킷 (Verilog circuit)사이의 일치성(behavioral consistency)을 체크하는 툴이다. 검증을 위하여 베릴로그 서킷과 C 코드는 CNF (Conjunction Normal Form)로 변환하고 SAT solver인 ZChaff을 사용하여 체크한다. 그러나 C 코드는 CNF로 변환에 몇 가지 문제점을 가지고 있다. 즉 side effect, pointer, dynamic memory allocation, 그리고 중첩 루프등의 문제 때문에 CNF로 변환하는 것은 어렵다. 본 논문에서는 이러한 문제점을 해결하는 방법을 기술하고 트리플 DES(Data Encryption Standard)[7] 코드를 CBMC로 검증하는 방법을 제시한다.

2장에서는 CBMC에서 C 코드 분석하는 기법에 대한 설명을 하고 3장에서는 트리플 DES을 검증하는 방법을 제시한다. 4장에서 결론 및 향후 연구 방향을 제시한다.

## 2. CBMC에서의 C 코드 분석

CBMC는 BMC을 이용하여 ANSI-C 프로그램과 베릴로그 서킷(Verilog circuit)사이의 일치성(behavioral consistency)를 체크하는 툴이다. [그림 1]과 같이 입력된 베릴로그 서킷과 프로그램은 수식로 전환된다. 만약 서킷과 C 코드가 일치하지 않는다면 satisfiable한 수식이다. 다음과정은 생성된 수식을 SAT solver인 ZChaff[8]를 사용하여 체크된다. 이 과정에서 베릴로그는 CNF로 변환이 쉽다. 그러나 C 코드의 변환은 Side effect, pointer, dynamic memory allocation, 그리고 중첩 루프등을 가지고 있으므로 CNF로 변환하는 것은 어렵다.

그러므로 비트 벡터의 동일한 식(Bit vector equation)의 타당성을 결정하는 C 프로그램에 대한 Model Checking 문제를 해결하는 방법을 설명한다. 먼저, 프로그램을 While, if, goto, 그리고 배정문을 변환




[그림 1] CBMC의 검증 과정

하는데 사용하는 동일한 프로그램으로 전처리를 한다.

모든 While 루프를 아래[1]에 처럼 n번 변환을 사용하여 프로그램을 펼쳐준다.

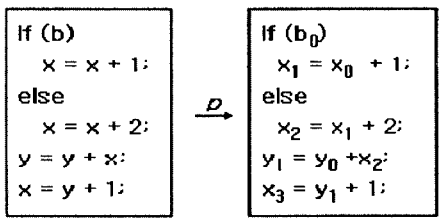
While (e) inst → if (e) { inst; while(e) inst } [1]

마지막 while 문은 프로그램이 더 이상 반복되지 않는 조건 **assertion !e** 을 사용한다. 이 조건(unwinding assertion)은 정의된 조건(assertions)과 함께 검증된다. 만약 프로그램의 범위(bound)가 크면 이 루프에 많은 반복이 증가한다.

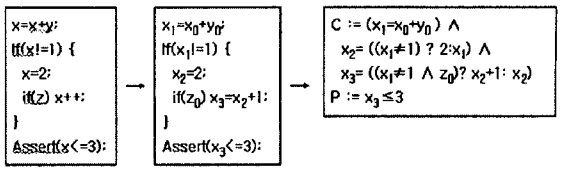
프로그램은 unwind되면 if 문, 배정문, assertions, 그리고 goto 문으로만 구성된다. 이 간단한 프로그램의 구성을 비트 벡터의 동일한 식 C(제한의 집합)와 P(assertions의 집합)으로 바꾼다. 이 과정은 [그림 2]처럼 변수의 이름은 순차적 번호가 부여되면서 바뀐다. 각각 변수는 단지 한번만 배정된다. 마지막 변환은 제약의 함수인 C(p, g)와 P(p,g)을 계산한다. C와 P는 인자(argument)로써 프로그램 p와 가드(guard) g 가지고 이것을 동일한 식(equation)으로 전환한다. 조건문은 프로그램 p 와 조건 c를 가진 if문과 여러줄의 명령문을 가진 I과 I`로 기술하고, functions는 코드 블록 둘 다에 대해 재귀적으로 사용된다.

I에 대해, p(c)는 guard를 추가하고 I`에 대해, ¬p(c) 는 guard가 추가된다.

C( if(c ) I else I`, g) := C(I, g∧p(c)) ∧ C(I`, g ∧ ¬p(c))
P(if(c ) I else I`, g) := P(I, g∧p(c )) ∧ P(I`, g ∧ ¬ p(p))




[그림 2] 변수명 변환 함수 P




[그림 3] 동일한 식으로의 변환 과정

배열 v[a] = e 에 대하여 아래의 제약을 추가한다. index i에서 array Va의 새로운 값은 If의 오른쪽에 가드(guard)을 둔다. 그리고 i는 a와 같다. 동일한 식은 function과 람다(lamda) 표현을 사용하여 나타낸다.

C( v= e, g) := va = λ i : (g∧ i = p(c)) ? p(e): va-1[i]

C와 P를 계산 한 후에, 우리는 C ⇒ P 가 타당한지 검증한다. [그림 3]은 변환 과정의 간단한 예를 보여준다. 첫번째 박스는 제한을 포함한 펼쳐진 프로그램이고 두번째 박스는 변수의 이름을 순서대로 변환하였다. 세번째 박스는 C(constraints)와 P(property)를 비트 벡터의 동일한 식으로 변환하였다.

## 3. 트리플 DES의 검증

DES는 개인키를 사용하여 데이터를 암호화하는 방법으로 사용되며, 72,000,000,00 0,000 (72천조)개 이상의 암호키가 사용되는 것이 가능하다. 각 64 비트 데이터 블록에, 56 비트 길이의 키를 적용하며 세개의 키가 잇달아 적용되는 트리플 DES를 상용적으로 사용한다. 그러므로 상용되는 트리플 DES을 CBMC로 코드 검증한다.

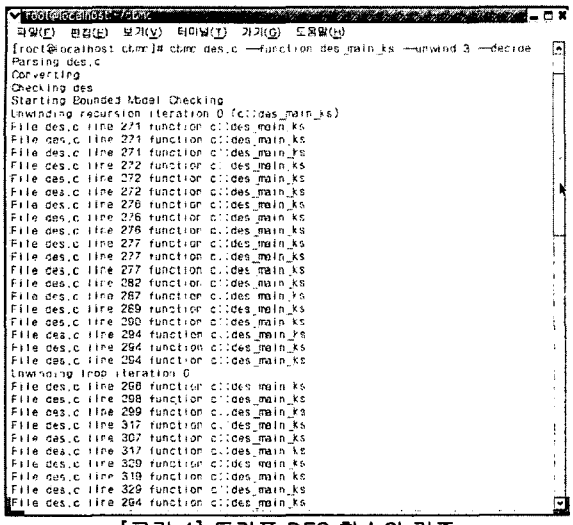
트리플 DES 코드를 검증하기 위한 환경은 래드햇 리눅스 8.0과 커널 2.4.28인 운영체계와 CBMC가 설치된 환경에서 트리플 DES의 코드를 CBMC의 입력으로 실행한다. 본 연구에서는 전체 코드 중에서 함수부분으로 나누어서 체크를 한다.

```
cbmc des.c --function des_main_ks --unwind 3 --
decide
```

이 명령은 des.c 파일에서 des_main_ks 함수를 검증하고 함수에는 반복문이 포함되어 있으므로 무한 반복을 3번으로 제한하여 검증을 마칠 수 있도록 범위를 제한한다. des.c 파일은 gcc 컴파일러로 이용하여 컴파일하면 에러없는 좋은 프로그램의 결과를 보여준다. 그러나 cbmc을 이용하여 체크를 하면 아래의 메시지가 출력된다.

**pointer dereference(invalid pointer) file des.c line 271 function C:des_main_kr**

271 라인의 코드에는 변수 각각에 대하여 초기화를




[그림 4] 트리플 DES 함수의 검증

하고 있다. Gcc 컴파일러에서 찾지 못한 프로그램의 문제를 CBMC에서 체크 한것이다. 왜냐하면 모든 변수를 비트 벡터의 동일한 식을 변환하고 C 조건에 대한 제약을 두어 CNF로 변환하여 체크를 하기에 때문에 변수의 초기화 문제을 찾아 낼수 있다

[그림4]은 함수 des_main_ks를 3번 반복하여 펼치는 과정을 보여준다.

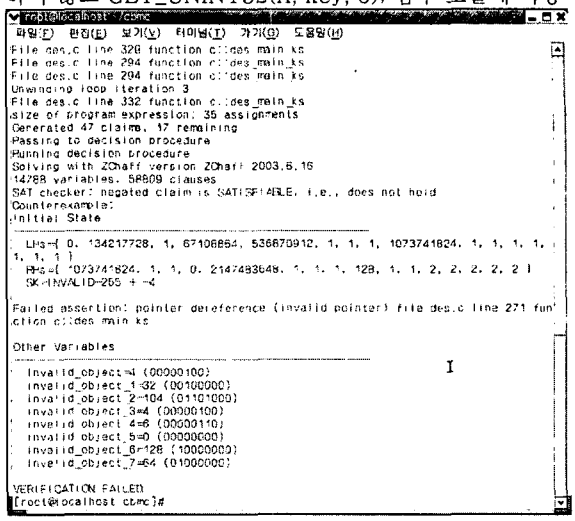
다른 모델 체커는 반복문이나 코드의 사이즈가 커짐에 따라 상태 폭발의 문제점을 가지고 있다. 그러나 BMC는 범위를 한정하여 체커를 할 수 있다는 장점을 가진다. 그리고 CBMC도 반복문을 효과적으로 체크할수 있는 장점을 가진다.

[그림 5]에서는 트리플 DES의 des_main_ks 함수의 검증의 마지막 과정을 보여준다. ZChaff에서 14788의 변수들과 58809의 절로 체크되어진다. C 컴파일러에서 체크하지 못한 변수의 초기화 문제점을 체크해준다.

## 4. 결론

급속하게 개발되어져야 하는 임베이디드 시스템에서 C 코드의 검증 방법이 절실히 요구된다. 본 논문에서는 지금까지 개발된 C 코드 검증 도구들 중에서 CBMC가 C 코드 검증을 위해 사용하는 기법을 설명하고 상용으로 사용되고 있는 트리플 DES의 검증 과정을 제시하였다. CBMC는 BMC의 장점을 수용하여 C 프로그램과 베릴로그 서킷의 행위적 일치를 검증하는 툴이다. 그러나 베릴로그는 CNF로 전환이 간단하나 C 코드는 CNF의 전환에 몇몇 어려움을 가지고 있다. 본 논문은 CBMC에서 C 코드 변환에 대하여 설명하고 트리플 DES를 코드를 CBMC로 검증하는 방법을 제시하였다. 향후 연구 방향은 CBMC을 이용하여 임베이드 시스템의 명세에 사용된 C 코드의 검증하고 나아가 새로운 코드 검증의 방법에 대한 연구가 필요하다.

하지 않고 GET_UNINT32(X, key, 0); 함수 호출에 사용




[그림 3] 트리플 DES 함수의 검증 결과

## 5. 참고문헌

[1] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft, Technical Report MSR-TR-2004-08, 2004

[2] T. Ball, R. Majumdar, Automatic Predicate Abstraction of C Programs, http://research.Microsoft.com/slam/

[3] http://www-cad.eecs.berkeley.edu/~rupak/blast/

[4] http://www-2.cs.cmu.edu/~chaki/magic/

[5] E. Clarke, D. Kroening, K. Yorav, Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking,

[6] A.Biere, A.Cimatti, E.E.Clarke, and Y. Yhu. Symbolic model Checking without BDDs, In Tools and Algorithms for Construction and Analysis of Systems, pp. 193-270,

[7]http://www.opencores.org/projects.cgi/web/des/overview

[8] M.W.Moskewicz, C.F.Medigan, and S. Malik, Chaff:Engineering and efficient SAT solver. In Proceedings of the 38th Design Automation Conference, 2001