요약 해석의 모델 검사를 이용한 정보흐름 제어
(Information Flow Control using Model-Checking of Abstract Interpretation)

조순희[1], 신승철[1], 도경구[2]
(Soon Hee Cho, Seung Cheol Shin, Kyung-Goo Doh)

요 약 본 논문은 명령형 언어 **While**의 요약해석을 모델검사기 SMV에서 구현하고 정보흐름 안전성을 검사하는 CTL 논리식을 적용하는 방법을 설명한다. 주어진 **While** 프로그램의 요약 프로그램을 SMV 프로그램으로 변환하는 방법과 정보흐름 안전성 검사를 위한 CTL 논리식의 유도과정을 보여준다. 요약해석기를 직접 구현하는 것보다 모델검사를 이용하는 것이 다양한 안전성 검사를 수행하기에 더욱 적합하다.

Abstract In this paper, implements the abstract interpretation of the imperative language While in SMV model-checker and explain how to apply the logic of CTL which example the security of information flow. And show the way to translate the abstract program of While into SMV program and explain the derive process of CTL logic to test the security of the information flow. For the various security test, it is suitable to use the model-checking than to implements the abstract interpretation.

1. 서 론

요즘처럼, 인터넷이 발달한 시대에서는 필요한 정보를 검색하면 누구나 쉽게 찾아볼 수가 있다. 이러한 과정에서 대부분의 사람들은 본인의 정보가 다른 사람에게 공개되지 않기를 바란다. 이렇게 인터넷 상에서 본인의 정보가 유출되는 것처럼 프로그램에서도 작성자의 의도와 상관없이 정보의 유출이 발생할 수 있다.

프로그램에서 나타나는 변수들은 각각 공개변수와 비밀변수로 구분할 수 있다. 공개변수란 그 정보가 외부로 공개되어도 좋은 변수이고 비밀변수란 이와 반대로 정보가 외부로부터 보호되어야 하는 변수이다. 프로그램 실행 중 비밀변수의 내용이 공개변수로 저장이 되면 결과적으로 정보가 외부로 유출이 된 것이라고 볼 수 있다. 이런 정보유출이 없는 프로그램을 "정보흐름이 안전한 프로그램" 이라고 한다.

본 논문에서는 공개변수와 비밀변수사이에 정보유출이 없음을 보이기 위해 모델검사의 하나인 SMV(Symbolic Model Verification)[1]를 이용하는 방법을 이용하였다. 기본적인 명령형 언어의 하나인 **While** 언어[2]로 작성된 프로그램을 요약해석 하여 생성된 요약프로그램을 모델검사기인 SMV 프로그램으로 변환하는 방법과 정보흐름 안전성을 검사하기 위한 CTL 논리식을 보여준다. 2절에서는 대상 프로그램인 **While** 프로그램의 문법과 의미구조를 보여주고 3절에서는 정보흐름 보안성이 무엇인지를 보여주고, 4절과 5절에서는 모델검사의 하나인 SMV를 이용하여 정보흐름의 안정성을 검증하는 방법을 설명한다.

1 동양대학교 컴퓨터공학부
2 한양대학교 컴퓨터공학과

2. While 프로그램

본 논문에서 정보흐름의 안전성을 검증하기 위한 대상이 되는 프로그램은 기본적인 명령형 언어인 **While** 의 프로그램이며 그 문법의 구조는 배정문, 조건문, while문 등을 정의한 것으로 다음과 같다[2].

$$S ::= x := e \mid \text{if } e \text{ then } S \text{ else } S \mid S ; S \mid \text{while } e \text{ do } S \mid \text{skip}$$
$$e ::= c \mid x \mid e \; op \; e$$

여기에서 c는 상수이며 여기에 해당하는 값은 정수와 불리언값이다. x는 변수, op는 이진연산자이며, op에 해당하는 연산자는 산술연산자와 불리언연산자이다. e는 표현식이고 S는 명령문을 나타낸다.

동작적 의미구조 표현방법(operational semantics)에는 구조적 의미구조방법(structural semantics)과 자연적 의미구조 표시방법(natural semantics)이 있다. 본 논문에서는

While 프로그램의 의미동작을 자연적 의미구조표시방법을 이용하였다.

여기에서의 프로그램의 의미는 $\langle S, \sigma\rangle \rightarrow \sigma'$와 같이 전이로 표현한다. 이 의미는 명령문 S가 초기상태함수 σ에서 수행하고 나면 σ'의 상태함수로 바뀌게 됨을 의미한다. 여기에서 상태함수 $\sigma \in$ State = Var → Val로 나타낼 수 있고, Val = Int + Bool이고 $[e]\sigma$는 상태함수 σ하에서 표현식 e를 계산한 결과를 말하는 것이며 $\sigma[v/x]$는 상태함수 σ에서 변수 x의 값을 v로 갱신함을 의미한다. 이렇게 정의된 자연적 의미구조 표시방법을 나타내면 다음과 같다[3].

$$\langle x := e, \sigma\rangle \rightarrow \sigma[[e]/x]$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma\rangle \rightarrow \sigma'} \text{ if } [e]\sigma = \text{true}$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma\rangle \rightarrow \sigma'} \text{ if } [e]\sigma = \text{false}$$

$$\frac{\langle s, \sigma\rangle \rightarrow \sigma' \quad \langle \text{while } e \text{ do } s, \sigma'\rangle \rightarrow \sigma''}{\langle \text{while } e \text{ do } s, \sigma\rangle \rightarrow \sigma''} \text{ if } [e]\sigma = \text{true}$$

$$\langle \text{while } e \text{ do } s, \sigma\rangle \rightarrow \sigma \text{ if } [e]\sigma = \text{false}$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow \sigma' \quad \langle s_2, \sigma'\rangle \rightarrow \sigma''}{\langle s_1; s_2, \sigma\rangle \rightarrow \sigma''}$$

$$\langle \text{skip}, \sigma\rangle \rightarrow \sigma$$

3. 정보흐름 보안성

정보흐름의 보안성이란 주어진 While 프로그램에서 변수들간에 배정문으로 인하여 보안수준이 높은 변수의 정보가 보안수준이 낮은 변수로 누출되는지를 검사하는 것이다. 변수들은 모두 초기에 보안수준이 정해져 있다.

초기에 부여된 변수들의 보안수준의 집합을 Sec라 하면 Sec={public, secret}으로 나타낼 수 있고 public은 공개되어도 좋은 공개변수이고 secret은 공개되지 말아야 하는 비밀변수이다. 그리고 이들 사이는 부분순서 (Sec, ⊑)가 형성되며 public⊑secret의 순서를 만족한다. 즉, public의 보안수준보다 secret의 보안수준이 높거나 같음을 의미한다. sec를 변수를 받아 그 변수의 초기보안수준 값을 주는 함수라 생각하면 sec∈Var→Sec라 정의할 수 있다. sec함수를 이용하여 변수 x와 y의 보안수준은 각각 $sec(x)$, $sec(y)$를 이용하여 구할 수가 있고, 이 두 변수를 연산한 보안수준을 구하기 위해선 join 연산을 하면 되고 다음과 같다. $sec(x) \sqcup sec(y) = \sqcup\{sec(x), sec(y)\} = sec(y)$ ***iff*** $sec(x) \sqsubseteq sec(y)$[3].

주어진 프로그램에서 정보유출은 크게 두 가지로 나눌 수 있다[4].첫 번째는 즉시 누출(immediately leak)이다. 이는 다시 $sec(a) \sqsubseteq sec(x)$일 때 $a := x$ 와 같은 배정문에서 발생하는 명시적 누출(explicit leak)과 $sec(a) \sqsubseteq sec(x)$일 때 if $x>0$ then $a=1$ else $a=0$ 과 같은 선택문에서 발생하는 묵시적 누출(implicit leak)로 나뉜다. 이는 얼핏보면 누출이 발생하지 않았다고 생각할 수도 있지만 비밀변수 x의 값에 따라 공개변수 a의 값이 정해지므로 a의 값을 보면 x의 정보를 유추해낼 수 있으므로 누출이 발생한 것이다.

두 번째는 전이적 누출(transitive leak)이다. 이는 이미 다른 변수로부터 누출된 정보를 가진 변수가 다시 다른 변수로 재 누출하는 경우를 말하는 것으로 $sec(x) \sqsubseteq sec(y) \sqsubseteq sec(z)$ 일 때 $x := z$; $y := x$와 같이 z의 정보가 x를 통해서 y로 누출이 되는 경우이다.

본 논문에서는 주어진 **While** 프로그램에서 위와 같이 초기의 보안수준에 비해 최종 보안수준이 높아지면 정보누출이 일어남을 알고 정보누출이 발생하지 않는 정보흐름이 안전한 프로그램임을 보이기 위해 요약해석을 이용하고 요약해석 프로그램을 모델검사기의 하나인 SMV 프로그램으로 바꾸어 정보의 누출이 있는지를 검증한다.

4. 모델검사기

하드웨어나 소프트웨어를 검증하는 방법에는 시뮬레이션(simulation), 테스팅(testing), 연역적 검증(deductive verification), 모델검사(model-checking) 등이 있다[5]. 본 논문에서는 이러한 방법 중 모델검사를 이용한다.

모델검사는 모델링, 명세화, 검증의 세 가지 과정으로 나뉜다. 모델링이란 주어진 시스템을 검증하기 위해 그 시스템을 대표할 수 있는 정형화된 모델을 만드는 것이다. 명세화란 주어진 시스템이 가져야할 특징들을 기술하는 것으로 보통 논리식으로 표현이 된다. 논리에는 무엇에 대해 중점을 둘 것인가에 따라 각각의 상태에 대해 생각하는 branching time logic 과 각각의 path에 대해 생각하는 linear-time logic이 있다[6]. 검증은 주어진 모델이 명세를 만족하는지를 자동으로 체크하는 것으로 에러가 발생하면 그 명세를 만족하지 않음을 뜻하며 그 경로를 보여준다. 모델링이 불완전하다던가 명세화가 부적절 하다면 검증이 결과값 또한 우리가 원하는 결과를 보여주지 못할 것이다. 그래서 모델검사에서는 모델링을 하는 것도 중요하지만 명세화 하는 것도 중요하다.

본 논문에서는 모델검사 도구중 SMV를 사용하여 정보흐름의 안전성을 검사하려고 한다. SMV는 유한한 상태 시스템에서 temporal logic의 하나인 CTL(computation tree logic)로 작성된 명세를 만족하는지를 검사하는 도구이다. 여기에서 사용되는 CTL 구문을 정의해 보면 다음과 같다[7].

$$\varnothing := \bot \mid \top \mid p \mid (\neg\varnothing) \mid (\varnothing \wedge \varnothing) \mid (\varnothing \vee \varnothing) \mid (\varnothing \rightarrow \varnothing) \mid AX\ \varnothing \mid EX\ \varnothing \mid A[\varnothing U \varnothing] \mid E[\varnothing U \varnothing] \mid AG\ \varnothing \mid EG\ \varnothing \mid AF\varnothing \mid EF\ \varnothing$$

여기에서 AX, EX, AG, EG, AU, EU, AF, EF는 시간연결어(temporal connectives)라 한다. A와 E가 한 쌍이며 A는 'along All paths'를 의미하고 E는 'along at least one path'를 의미하며 AU와 EU는 상반되는 개념이다. 또 다른 한 쌍은 X, Fm G, U 이다. X는 'neXt'를 F는 'some Future state'를 G는 'all furure states(Globally)'를 U는 'Until'을 의미한다. 이것들은 A나 E없이 홀로 사용될 수 없다.

SMV로 작성된 프로그램의 실행결과는 SPEC의 결과값이다. SPEC은 그 모델이 만족해야 할 명세를 CTL formula로 적어 놓는 부분으로 주어진 명세에 만족을 하게 되면 true를 그렇지 않으면 false를 출력하면서 어디에서 만족되지 않는지 그 경로를 보여준다.

5. SMV에서의 정보흐름보안성 검증

5. 1 **While** 프로그램의 요약해석

주어진 **While** 프로그램을 요약해석(abstract interpretation)하여 요약 프로그램을 만들 수 있다. 요약해석이란 프로그램을 실행시킬 때 실제 값(concrete values)들을 사용하지 않고 요약한 값(abstract values)들을 가지고 요약연산자(abstract operator)를 이용하여 계산하여 값을 구하는 것이다.

예를 들어 자연수의 연산을 한다고 가정하면 자연수의 값들 즉, 1이나 -1을 가지고 연산을 하는 것이 아니라 요약값(여기에서는 부호를 요약값으로 하자)을 가지고 연산을 하는 것이다. 부호값으로 요약한 새로운 정의역은 Sign={NONE, POS, NEG, ANY}라 정의할 수 있다. 이렇게 되면 자연수끼리의 연산이 아니라 부호끼리의 연산이 되고 연산도 일반 사칙연산이 아닌 POS + POS = POS 라든가 NEG + NEG = NEG 등으로 새로 정의된 사칙연산이 될 것이다. 또한, 그 결과값도 어떤 자연수가 아니라 Sign의 원소가 될 것이다.

While 프로그램의 요약을 통해 정보흐름을 나타내면 다음과 같은 프로그램을 요약된 프로그램으로 변환할 수 있다.

$[z := 0]^1$;
while $x > 0$ do ($[z := z + 1]^2$; $[y := y + z]^3$)

위의 프로그램은 3개의 배정문을 가지고 있는데 각 배정문에서 좌변의 변수에 영향을 주는 변수들의 집합이 다음과 같이 정해진다.

$[z := 0]^1$: {}
$[z := z + 1]^2$: {z, x}
$[y := y + z]^3$: {y, z, x}

이 집합을 이용하면 다음과 같은 요약된 프로그램을 얻을 수 있다.

$z := 0$;
$z := z \parallel x$;
$y := y \parallel z \parallel x$;

이 프로그램은 배정문을 통해 각 변수에 어떤 변수들이 영향을 주는 지를 나타낼 수 있고 특히 영향을 주는 변수중에 '1' 값을 갖는 비밀변수가 포함되어 있는지에 따라 좌변의 변수에 비밀 정보가 유입되는 지를 확인할 수 있다.

5. 2 SMV에서 정보유출 여부 검증

요약 프로그램을 모델링하여 CTL formula를 사용한 모델검사기인 SMV 시스템으로 옮겨 표현할 수 있다. 본 논문에서는 Sec의 값이 secrete인 변수를 1로 정의하고 public인 변수를 0으로 정의하였다. 그리고 두 변수의 Sec는 *sec* 함수에 적용시켜 구할 수 있다.

최초의 보안수준이 정해져 있다면 최종 보안수준은 or 연산하여 구할 수 있다. 예를 들면 $sec(x) = 0$, $sec(y) = 0$, $sec(z) = 1$ 이라 할 때 $x := z$와 같은 배정문이 있다고 하자. 초기 x의 Sec는 public이다. 그러나 배정문 실행 후 x의 Sec는 0 or 1이 되어서 secrete가 된다. 이렇게 초기 낮은 보안수준을 가진 변수가 최종적으로 초기보다 높은 보안수준을 갖는지를 검사하면 정보의 누출이 있었는지를 알아낼 수가 있다.

다음 SMV 프로그램은 요약 프로그램으로부터 변환한 것이다.

```
MODULE main

VAR
   x : boolean;
   y : boolean;
   z : boolean;
   pc : integer;

ASSIGN
   init(x) := 1;
   init(y) := 0;
   init(z) := 0;
   init(pc) := 1;

   next(y) := case
              pc = 3 : y | z | x;
              1      : y;
              esac;
   next(z) := case
              pc = 1 : 0;
              pc = 2 : z | y | x;
              1      : z;
              esac;
   next(pc) := case
               pc < 3 : pc + 1;
               1      : pc;
              esac;

SPEC
AG((y=0)&(z=0))
```

로 나타낼 수 있고 프로그램 실행결과는 다음과 같다.

-- specification AG (y = 0 & z = 0) is false

즉, 결과값이 false 가 되어 정보유출이 있음을 알 수가 있다. 위의 프로그램에서 최초 *y*, *z*의 보안수준은 0이다. 프로그램을 실행시키고 난 후에 *y*나 *z*의 값이 1로 바뀌었다면 정보유출이 있어났다고 볼 수가 있는 것이다. 그래서 SPEC을 AG(*y* = 0) & (*z* = 0)을 주었다. 이 말은 항상 모든 곳에서 *y*와 *z*의 값이 변하지 않고 0이면 참이고 그렇지 않고 1이 되면 정보유출이 일어난 것이고 결과는 거짓이 됨을 나타내는 것이다.

2절에서 언급했듯이 정보유출은 배정문과 선택문에서 일어난다. 배정문에서는 보안수준이 높은 변수의 정보를 보안수준이 낮은 변수가 가지게 되면서 정보의 유출이 발생하는 것이다. 이런 배정문에서의 정보유출 검사를 위해 모델검사에서는 변수초기의 보안수준값과 배정문 우변의 연산결과값을 비교하여 최종 보안수준 값이 더 높은 값으로 바뀌었다면 정보유출이 있었다고 본다.

선택문에서의 정보유출은 비교문의 값에 따라 결정되는 결과값을 보고 비교문에 있는 변수의 정보를 파악할 수 있다. 이런 선택문에서의 정보유출 검사를 위해서는 선택문의 결과에 따라 실행하는 실행문의 변수들을 연산한 결과값에 비교문에 쓰인 변수의 값을 연산한 최종 보안수준값과 최초 보안수준값을 비교하여 정보유출의 유무를 알아낸다. 예를 들면 *sec*(*x*) = 0, *sec*(y) = 0, *sec*(*z*) = 1 경우 if *z*>0 then *x*=*x*+1 else *y*=*y*-1 이라는 비교문이 있다고 보면 실행문의 표현식에서는 정보유출이 없는 것처럼 보인다. 그러나 이들 표현식의 값으로 유추해 볼 때 변수 *z*의 정보를 알아낼 수가 있는 것이다. 그렇기 때문에 선택문에서의 보안수준을 알아내기 위해선 실행문의 보안수준을 연산한 값과 비교문에 쓰인 변수의 보안수준의 값을 연산하여 구해야 한다.

6. 결론

본 논문은 주어진 **While** 프로그램을 실제값으로 해석하는 것이 아니라 요약해석을 이용하여 정보의 유출이 있는지의 여부를 검증하는 것이다. 요약된 값을 가지고 SMV System을 구성하고 CTL 논리식의 값으로 정보유출 여부를 검사하였다. 즉, SPEC을 보안수준이 낮은 변수의 보안수준 값이 언제든지 변하지 않을 때 참이 되게 만들어 그 결과값이 false 가 되면 정보유출이 있다고 보고 있다. 새로이 요약해석기를 구현하는 것이 아니라 모델검사기에서 돌려봄으로써 정보유출의 여부를 가려내는 것이다.

이렇게 되면 다양한 분야의 안전성을 검사하기 위해 SMV System에서의 표현식들과 CTL formula를 바꾸면 되기 때문에 새롭게 요약해석기를 구현하는 것보다 훨씬 효율적이라고 본다.

본 논문에서는 기본적인 명령형 언어의 하나인 **While** 프로그램의 요약해석 프로그램의 모델링을 하였으나 앞으로는 모든 언어에 가능한 방법으로 확장하는 것과 요약프로그램을 수작업으로 모델링 하는 것이 아니라 하나의 프로그램을 작성하여 그 프로그램을 이용하여 자동으로 SMV 모델링하는 것이 더 연구되어야 할 것이다.

참고문헌

[1] K L. McMillan, The SMV system for SMV version 2.5.4

[2] H.R. Noelson and R. Nielson, Semantics with Application : A Formal Introduction, Wiley, 1992

[3] 도경구 and 이수호, "요약해석을 이용한 정보흐름 제어", 정보과학회 학술발표대회, 2002, pp.355-357

[4] K.-G.Doh and S.C.Shin, "Analysis of Secure Information Flow by Model-Checking(extend abstract)", In Proceedings of APLAS2001, The Second Asian Workshop on Programming Languages and Systems, 2001, pp.225-236

[5] Edmund M. Clarke. Jr and Orna Grumberg and Doron A. Peled, Model Checking, MIT, 1999

[6] M. M. Muller-Olm, D. Schmidt and B. Steffen, "Model-checking : A tutorial introduction", In G. File and A. Cortesi, editors, Proc. of the 6th Static Analysis Symposium, Lecture Notes in Computer Science, Springer, 1999.

[7] M. Huth and M. Ryan, Logic in Computer Science : Modeling and Reasoning about Systems, Cambridge, 2000