SMV를 이용한 정보 흐름 안전성의 모델검사 (Model Checking the Safety of Information Flow using SMV)

조 영 갑, 도 경 구[1]
한양대학교 컴퓨터공학과
{ykcho,doh}@cse.hanyang.ac.kr
신 승 철[2]
동양대학교 컴퓨터공학부
scshin@phenix.dyu.ac.kr

요 약

비밀변수에 저장되어 있는 데이터가 배정문을 통해 공개변수에 저장되어 외부로 알려지는 현상을 정보 누출이라고 하고, 정보 누출이 없는 프로그램을 "정보흐름이 안전한 프로그램"이라고 한다. 이 논문에서는 모델검사 기법을 사용하여 정보흐름의 안전성 검사할 수 있는 CTL 논리식을 정의하고, SMV 도구를 사용하여 모델 검사하는 방법을 제시한다.

1. 서론

컴퓨터가 인터넷을 통해서 서로 연결되고, 불특정 다수의 프로그램이 자신의 컴퓨터에서 자신도 모르는 사이에 실행되는 환경에서 자신의 비밀정보의 보호는 중요하다. 특히 프로그램을 서로 공유하는 경우 자신의 사적인 비밀 데이터가 고의이건 고의가 아니건 바깥 세상에 전부 또는 일부가 공개될 수도 있다. 비밀 정보가 저장되어 있는 비밀변수의 값이 프로그램이 실행되는 도중 공개변수에 저장이 되어 궁극적으로 타인에게 공개되는 것을 정보의 누출(information leak)이라고 하고, 정보의 누출이 없는 프로그램을

1 본 논문은 한국과학재단의 특정기초연구(과제번호: 2000-1-30300-010-3) 연구비 지원에 의한 것임
2 이 논문은 2000년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2000-003-E00250)

"정보흐름이 안전한 프로그램"이라고 한다[3,4]. 주어진 프로그램에 대하여 비밀변수와 공개변수 사이의 정보 누출이 없는지를 검사하는 정보흐름 안전성 검사(secure information flow analysis) 기법으로는 지금까지 데이터흐름분석[1], 요약 해석[8], 타입시스템[11] 등을 이용하는 방법이 연구되어 왔고, 최근에는 의미구조를 이용한 방법[6]도 발표되고 있다.

이 논문에서는 정형검증 기법으로 많이 사용되고 있는 모델검사(model checking)[2,9]로 정보흐름의 안전성을 검사할 수 있는 CTL 논리식을 정의하고, SMV[7]를 사용하여 모델검사하는 기법을 제시한다.

이 논문의 구성은 다음과 같다. 2절에서는 검사하고자 하는 정보흐름의 안전성이 무엇인지 간단히 설명하고, 3절에서는 정보흐름의 안전성을 CTL 논리식으로 정형적으로 정의하며, 4절에서는 정보흐름의 안전성을 모델검사하는 절차를 예를 통해서 보여주고, 5절에서는 SMV 도구로 모델검사하는 방법을 제시하며, 마지막으로 6장에서는 결론을 맺고 추후 연구 방향을 제시한다.

2. 정보 흐름의 안전성

이 논문에서 "정보의 흐름이 안전하다(information flow is safe/secure)"란 공개되어서는 안되는 비밀변수(private variable)에 저장된 데이터에 관한 정보가 어떠한 경로를 통해서도 공개변수(public variable)에 저장되어 외부로 누출되지 않음을 의미한다. 명령형 언어에서 비밀정보의 누출은 다음과 같이 두 가지 경우로 나누어 볼 수 있다. 비밀변수(x)에 저장된 정보가 배정문(assignment) `a := x` 를 통하여 공개변수(a)에 저장되어 누출될 수 있다. 이를 정보의 명시적 누출(explicit leak)이라고 한다. 또, `if x then a := 0 else a := 1` 와 같은 선택문의 경우 비밀변수(x)의 정보를 공개변수(a)의 값을 보고 알아낼 수 있다. 이를 정보의 묵시적 누출(implicit leak)이라고 한다. 이 논문에서는 이 두 가지 누출을 모두 고려한다.

누출은 전이적(transitive)으로 일어날 수도 있다. 예를 들어, 프로그램

```
a := y;
b := a;
```

의 경우 비밀변수(y)의 정보가 공개변수(a)로 누출되고, 전이적으로 공개변수(b)로도 누출된다.

경우에 따라서, 프로그램의 일부분만 보면 누출이 일어나는 것처럼 보이지만 그 부분을 포함하고 있는 프로그램과 같이 따져보면 누출이 일어나지 않는 경우가 있다. 예를 들어, x가 비밀변수이고 a와b가 공개변수일 때,

(P1) (1) `a := x;`
(2) `a := b;`

에서 (1)만 보면 누출이 일어난 것처럼 보이지만 (2)에서 a의 값이 공개변수의 값으로 즉각 바뀌었기 때문에 실질적으로 정보 누출이 일어나지 않은 셈이 된다. 또한,

(P2) (3) `x := a;`
(4) `a := x;`

에서 (4)만 보면 정보 누출이 일어난 것처럼 보이지만 (3)에서 비밀정보가 사실상 공개정보에 의해서 지워졌기 때문에 전체적으로는 누출이라고 볼 수 없다.

프로그램의 모든 비밀변수에 대해서 누출이 발생하지 않으면 그 프로그램은 정보흐름이 안전하다고 할 수 있다.

3. 정보 흐름 안전성의 정의

2절에서 정의한 정보 흐름의 안전성 분석을 Doh-Shin의 논문에서는 데이터흐름식(data flow equation)으로 정의하고 이를 모달 μ-논리식으로 표현함으로써 모델 검사를 통한 정보흐름 분석이 가능하게 되었다[5]. 그들의 연구결과에 따르면 그들이 고안한 순방향분석(forward analysis)과 역방향분석(backward analysis) 중에서 하나 만이라도 프로그램이 안전하다고 판정이 되면 그 프로그램의 정보 흐름은 안전하다고 한다. 본 장에서는 그들의 데이터흐름식을 모달논리식(modal logic)의 하나인 CTL(computational tree logic) 논리식으로 정의한다.

3.1 FlowGraph

명령형 언어의 정보흐름의 안전성을 분석하기 위하여 프로그램의 정보 흐름만을 표현하는 언어 FlowGraph를 정의한다. FlowGraph의 문법은 다음과 같다.

$$A ::= [x \leftarrow y]^l \mid A1 \Rightarrow A2 \mid A1|A2$$

$$x,y \in \text{Var}, \quad l \in \text{Label}$$

$[x \leftarrow y]^l$는 기본 블록으로 그래프의 마디라고 할 수 있는데, 변수 y의 값이 변수 x에 저장됨을 의미하고, 여기서 l은 자연수로서 블록의 번호를 나타낸다. A1⇒A2는 A1에서 A2로 흐름이 있음을 나타내며, A1|A2는 A1과 A2 중에서 선택하여 흐를 수 있음을 나타낸다. 다음은 두 배정문으로 이루어진 명령형 언어를 FlowGraph로 변환한 예이다.

```
a := x;
b := a;
```

의 FlowGraph 표현:

$[\ a \leftarrow x\ ]^1 \Rightarrow [\ b \leftarrow a\ ]^2$

3.2 CTL

CTL(Computational tree logic)은 분기시제 논리(branching-time, temporal logic)이다. CTL에서는 선형 시제 논리(Linear temporal logic)에서 사용되는 시제 연산자 G, F, X, U 앞에 경로 한정사(path quantifier) E, A가 선행되어 논리식을 표현한다. 예를 들어, 어느 시점(state)에서 CTL 논리식 AGφ는 현 시점으로부터 모든 가능한 계산 경로에서 φ가 만족함을 나타내고, EFφ는 φ를 만족하는 포인트를 갖는 경로가 존재함을 나타낸다.

3.3 정보누출의 정의

프로그램에 속해있는 모든 변수는 보안수준(security level)을 가지며, 이 논문에서는 비밀변수와 공개변수의 두 단계 수준만 가진다고 가정한다. 이 논문에서 공개변수의 집합은 Public으로, 비밀변수의 집합은 Secret으로 표시하고, Var = Public ∪ Secret 이다..

3.3.1 순방향 누출

FlowGraph 프로그램의 시작점에서 분석을 시작하여 그래프의 흐름 방향으로 분석을 진행하여 검사가 끝난 후 누출이 발생하면 “순방향으로 누출이 발생한다(forwardly leaked)”고 한다. 비밀변수 x에서 공개변수 a

로 순방향으로 누출이 블록 l에서 존재하는지 알아보는 논리식을 CTL로 표현하면 다음과 같다.

$leakForward\ (l,\ a,\ x)$ =

EU(¬$isLhs(a)$,

($isLhs(a) \wedge isRhs(x)$)

$\vee$($\exists b.isLhs(a) \wedge isRhs(b)$

$\wedge \exists l' \in past(l).leakForward(l,b,x)$))

where

$a,\ b \in$ Public, $x \in$ Secret, $a \neq b$

$[a \leftarrow x]^1 \models isLhs(a)$,

$[a \leftarrow x]^2 \models isRhs(x)$,

$past$: Label $\rightarrow \mathcal{P}$(Label)

$past(l)$ = 블록 l로 흘러 들어오는 모든 블록 Label의 집합

CTL에서 EU(φ,ψ)은 현재 ψ가 true이거나, φ가 true이면서 추후 언젠가 ψ가 true 되면, true가 된다는 의미이다. 따라서 위 논리식은 블록 l에 도달하기까지 언젠가는 좌변이 a가 되는 블록이 존재하고, 그 블록의 우변이 x가 되어 즉시 누출(immediate leak)이 발생하거나($isLhs(a) \wedge isRhs(x)$), 그 이전 블록에서 발생한 누출이 전이되어 누출(transitive leak)이 일어나게 되면($\exists b.isLhs(a) \wedge isRhs(b) \wedge \exists l' \in past(l).leakForward(l,b,x)$), x에서 a로 순방향 누출이 블록 l의 지점에서 존재한다는 의미이다. 따라서 순방향 누출을 검사하기 위해서는 다음과 같이 프로그램의 마지막 블록에서 위 논리식을 적용해야 한다. 순방향으로 검사를 하면, 2절에 제시한 프로그램 (P1)의 경우 (1)에서 일어난 정보 누출이 (2)에서 복구됨을 감지해 낼 수 있지만, 프로그램 (P2)의 경우 (3)에서 비밀 정보가 지워졌다는 사실을 알 수 없으므로 (4)에서 정보 누출이 발생했다고 밖에 판정할 수밖에 없다. 따라서 순방향으로만 검사하면 정보가 누출되지 않은 경우도 정보가 누출된다고 판정이 나올 수 있다. 따라서 역방향 누출 검사가 필요하다.

3.3.2 역방향 누출

FlowGraph 프로그램의 끝점에서 분석을 시작하여 그래프의 흐름 반대 방향으로 분석을 진행하여 검사가 끝난 후 누출이 발생하면 "역방향으로 누출이 발생한다(backwardly leaked)"고 한다. 비밀변수 x에서 공개변수 a로 역방향으로 누출이 블록 l에서 존재하는지 알아보는 논리식을 CTL로 표현해보면 다음과 같다.

$leakBackward\ (l,\ a,\ x)$ =

EU(¬$isLhs(x)$,

($isLhs(a) \wedge isRhs(x)$)

$\vee$($\exists z.isLhs(z) \wedge isRhs(x) \wedge$

$\exists l' \in future(l).leakBackward(l',a,z)$))

where

$a \in$ Public, $x,\ z \in$ Secret, $x \neq z$

$[a \leftarrow x]^1 \models isLhs(a)$,

$[a \leftarrow x]^2 \models isRhs(x)$,

$future$: Label $\rightarrow \mathcal{P}$(Label)

$future(l)$ = 블록 l에서 흘러 나가는 모든 블록 Label의 집합

위 논리식은 블록 l에 역방향으로 도달하기까지 언젠가는 반드시 블록의 우변이 x가 되는 블록이 존재하고, 그 블록의 좌변이 a가 되어 즉시누출이 발생하거나($isLhs(a) \wedge isRhs(x)$), 그 블록좌변의 어떤 변수 z가 그 이후 블록에서 전이되어 누출이 일어나게 되면($\exists z.isLhs(z) \wedge isRhs(x) \wedge \exists l' \in future(l).leakBackward(l,a,z)$), x에서 a로 역방향 누출이 블록 l에서 존재한다는 의미이다. 이때 좌변이 x인 블록을 만나게 되면, 그 블록에서 즉시 및 전이 누출이 일어나지 않으면 누출

이 없다($\neg isLhs(x)$). 따라서 역방향 누출을 검사하기 위해서는 다음과 같이 프로그램의 시작 블록에서 위 논리식을 적용해야 한다. 실질적으로 누출이 없으면 순방향 누출이 있다고 나온 경우라도 역방향 누출은 없다고 나온다. 역방향으로 검사를 하면 프로그램 (P2)의 경우 (4)번에서 발생한 정보 누출은 (3)번에 의해서 복구됨을 알 수 있다. 반면 순방향 검사로 정보 누출이 없다고 판정된 프로그램 (P1)의 경우, 역방향으로는 누출된 비밀 정보가 (2)에서 지워졌다는 사실을 알 수 없으므로 정보 누출이 발생했다고 밖에 판정할 수 없다.

3.4 프로그램의 안전성 정의

프로그램에서 비밀변수의 누출 여부를 효과적으로 검사를 하기 위해서는 양방향 검사가 반드시 필요하다. 순방향이나 역방향 둘 중 하나라도 누출이 없다고 확인되면 그 프로그램은 누출이 없으므로 안전하다고 판정할 수 있다. 결론적으로 프로그램의 안전성을 검사하는 식은 다음과 같다.

$$\forall a \in \text{Public}.\ \forall x \in \text{Secret}.\ \neg leakForward(l, a, x) \vee \neg leakBackward(l', a, x)$$

where l = 마지막 블록의 번호,
l' = 시작 블록의 번호

여기서, $m \notin$ Public 이거나, $n \notin$ Secret 인 경우에는 항상 다음과 같이 된다.

$$leakForward(l, m, n) = \text{false}$$
$$leakBackward(l, m, n) = \text{false}$$

모든 프로그램 분석이 그렇듯이 안전한 프로그램을 안전하지 않다고 판정할 수는 있다. 그러나, 안전하지 않은 프로그램을 잘 못 판정하는 경우는 절대 없어야 한다. 이 식의 안전성(soundness)는 Doh-Shin의 논문에 증명이 되어 있다[5].

4. 정보 흐름 안전성의 모델검사

이 절에서는 정보 흐름 안전성 점검을 위한 모델 검사가 어떻게 이루어지는 지 예를 통해서 살펴본다.

(예제 1)

아래 프로그램은 블록 1에서 누출이 일어나지만 블록 2에서 복구되므로 정보누출이 없는 안전한 프로그램이다.

$$[\ a \leftarrow x\ ]^1 \Rightarrow [\ a \leftarrow b\ ]^2$$

where $a, b \in$ Public, $x \in$ Secret

이 프로그램을 순방향 검사를 하면 정보 누출이 없다고 나오지만 역방향 검사를 하면 그렇지 않게 나온다. 위 프로그램이 안전함을 보이기 위해서는 x에서 a로의 누출이 없고 x에서 b로의 누출이 없음을 보이면 된다. 따라서 이 프로그램의 안전성을 표현하는 논리식은 다음과 같이 표현할 수 있다.

$$(\neg leakForward(2,a,x) \vee \neg leakBackward(1,a,x)) \wedge (\neg leakForward(2,b,x) \vee \neg leakBackward(1,b,x))$$

먼저 x에서 a로의 누출 여부를 양방향 모두 따져보자.

$leakForward(1,b,x)$
$= EU(\neg isLhs(b),\ isLhs(b) \wedge isRhs(x))$
$= \text{false}$
이므로

leakForward(2,*a*,*x*)
= EU(¬*isLhs*(*a*),
(*isLhs*(*a*)∧*isRhs*(*x*))
∨(*isLhs*(*a*)∧*isRhs*(*b*)
∧*leakForward*(1,*b*,*x*)))
= false
leakBackward (1, *a*, *x*) =
= EU(¬*isLhs*(*x*), *isLhs*(*a*)∧*isRhs*(*x*))
= true
결국 ¬false∨¬true = true 가 되어 *x*에서 *a*로의 누출이 없음을 알 수 있다.

다음으로 *x*에서 *b*로의 누출 여부를 양방향 모두 따져보자.
leakForward(2,*b*,*x*) =
= EU(¬*isLhs*(*b*),
(*isLhs*(*b*)∧*isRhs*(*x*))
∨(*isLhs*(*b*)∧*isRhs*(*a*)
∧*leakForward*(1,*a*,*x*)))
= false
leakBackward (1, *b*, *x*) =
= EU(¬*isLhs*(*x*), *isLhs*(*b*)∧*isRhs*(*x*))
= false
마찬가지로 ¬false∨¬false = true 가 되어 *y*에서 *a*로도 누출이 없음을 알 수 있다.
따라서, 두 경우 모두 누출이 없으므로, 이 프로그램은 안전하다.

(예제 2)

아래 프로그램은 블록 3에서 누출이 일어나지만, 블록 1에서 *y*의 비밀정보가 공개정보로 바뀌었고 그 값이 블록 2에서 전이되어 *x*의 비밀정보도 삭제되었으므로 정보누출이 없는 안전한 프로그램이다.

$[\ y \leftarrow a\ ]^1 \Rightarrow [\ x \leftarrow y\ ]^2 \Rightarrow [\ a \leftarrow x\ ]^3$

where $a \in$ Public, $x, y \in$ Secret

이 프로그램을 역방향 검사를 하면 정보 누출이 없다고 나오지만, 순방향 검사를 하면 그렇지 않게 나온다. 위 프로그램이 안전함을 보이기 위해서는 *x*에서 *a*로의 누출이 없고 *y*에서 *a*로의 누출이 없음을 보이면 된다. 따라서 이 프로그램의 안전성을 표현하는 논리식은 다음과 같이 표현할 수 있다.

(¬*leakForward*(3,*a*,*x*)∨¬*leakBackward*(1,*a*,*x*))
∧(¬*leakForward*(3,*a*,y)∨¬*leakBackward*(1,*a*,*y*))

먼저 *x*에서 *a*로의 누출 여부를 양방향 모두 따져보자.
leakForward(3,*a*,*x*)
= EU(¬*isLhs*(*a*),
(*isLhs*(*a*)∧*isRhs*(*x*))
∨(*isLhs*(*a*)∧*isRhs*(*y*)
∧*leakForward*(2,*a*,*y*)))
= EU(false,
true ∨(*isLhs*(*a*)∧*isRhs*(*y*)
∧*leakForward*(2,*a*,*y*)))
= true
leakBackward (1, *a*, *x*)
= EU(¬*isLhs*(*x*),
(*isLhs*(*a*)∧*isRhs*(*x*))
∨(*isLhs*(*y*)∧*isRhs*(*x*)
∧*leakBackward*(2, *a*, *y*)))
= false
결국 ¬true∨¬false = true 가 되어 *x*에서 *a*로의 누출이 없음을 알 수 있다.

다음으로 *y*에서 *a*로의 누출 여부를 양방향 모두 따져보자.
leakForward(2,*a*,*x*)
= EU(¬*isLhs*(*a*),
(*isLhs*(*a*)∧*isRhs*(*x*))
∨(*isLhs*(*a*)∧*isRhs*(y)
∧*leakForward*(1,*a*,*y*)))
= EU(true, false∨false)

= false
이므로
leakForward(3,*a*,*y*)
= EU(¬*isLhs*(*a*),
(*isLhs*(*a*)∧*isRhs*(*y*))
∨(*isLhs*(*a*)∧*isRhs*(*x*)
∧*leakForward*(2,*a*,*x*)))
= EU(false, false∨(true∧*leakForward*(2,*a*,*x*)))
= false
leakBackward (1, *a*, *y*)
= EU(¬*isLhs*(*y*),
(*isLhs*(*a*)∧*isRhs*(*y*))
∨(*isLhs*(*x*)∧*isRhs*(*y*)
∧*leakBackward*(2, *a*, *x*)))
= false
마찬가지로 ¬false∨¬false = true 가 되어 y에서 a로의 누출이 없음을 알 수 있다.

5. SMV를 사용한 모델검사

SMV는 하드웨어나 소프트웨어 검증을 하는데 쓰는 정형검증도구이다[7]. SMV는 주어진 대상 시스템의 모델과 명세(specification)가 주어지면 가능한 모든 경우를 검사하여 대상 시스템이 명세를 만족하는지를 검사하여 준다. 여기에서 명세는 대상 시스템이 유지하는 특성(property)을 열거한 것으로서, 간단한 불린 식으로부터 복잡한 시제 명제(temporal logic)에 이르기까지 다양한 방법으로 기술할 수 있다. SMV에서는 CTL(Computational Tree Logic)을 사용하여 특성을 표시한다. SMV는 고유의 입력언어를 가지고 있으며, 이 언어는 유한 Kripke 구조(finite Kripke structure)의 상태전이(transition relation)를 효과적으로 기술할 수 있도록 고안되었다.

4절에서와 살펴 본 모델 검사는 SMV로 구현할 수 있다. 본 논문에서 정의한 CTL 논리식이 상호 호출하도록 정의되어 있기 때문에 모델 검사를 위해서 4절에서 했던 것처럼 서로의 검사 결과를 상호 참조해야 한다. 블록의 개수가 n, 비밀변수의 개수가 s, 공개변수의 개수가 p일 때, 검사에 필요한 논리식의 개수는 순방향과 역방향을 모두 합쳐서 최악의 경우 $2\times n\times s\times p$가 된다. 이 식을 각각 하나의 모델검사 프로세서에 할당하여 모델검사를 하면 된다. 이때 식을 검사하는 각 프로세서는 서로 종속되어 있으므로, 빨리 검사가 끝나는 식의 검사결과를 먼저 얻어서, 다른 식을 검사하는데 사용하면 효과적이다. 따라서 각 식은 다음과 같이 쪼개서 3개의 모듈을 작성하여 이전 모듈에서 얻어진 결과를 다음 모듈에서 이용하도록 한다. *leakForward*의 경우 3개의 단계별 모듈에서 검사하는 식을 다음과 같이 분리한다.

leakForward (*l*, *a*, *x*) =
EU(A, B∨(C∧D))
where A = ¬*isLhs*(*a*)
B = *isLhs*(*a*)∧is*Rhs*(*x*)
C = ∃*b*.*isLhs*(*a*)∧*isRhs*(*b*)
D = ∃*l′*∈*past*(*l*).*leakForward*(*l′*,*b*,*x*)

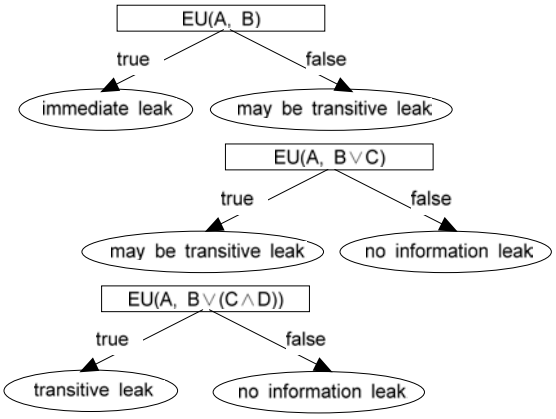
단계별로 검사하는 식과 다음 단계에서 사용할 결과는 [그림 1]에 요약되어 있다.

모듈 1 : 이 모듈에서는 EU(A,B)를 검사하여 직접적인 누출이 있는지를 검사한다. 이 경우 true가 나오면 직접적인 누출이 있다고 결론지을 수 있으며, 이 결과를 다음 단계에서 사용할 수 있다. false가 나오는 경우는

정보 누출이 없을 수도 있고, 전이에 의한 정보 누출이 있을 수도 있으므로 다음 모듈에서 계속 검사해야 한다.

모듈 2 : 다음 단계에서는 전이를 통한 누출의 검사에서 다른 식을 호출하는 부분을 제외한 논리식 EU(A, B∨C)를 검사하는데, 모듈 1에서 얻은 결과를 사용한다. 이 경우 false가 나오면 누출이 없다고 결론지을 수 있고, true가 나오면 정보 누출이 없을 수도 있고, 전이에 의한 정보 누출이 있을 수도 있으므로 결론은 다시 다음 모듈로 넘겨서 다른 식을 호출하는 부분까지 검사를 하여야 한다.

모듈 3 : EU(A, B∨(C∧D))를 검사하는데, 모듈 1과 2에서 얻은 결과를 사용한다. 이 경우 false가 나오면 누출이 없다고 결론지을 수 있고, true가 나오면 전이에 의한 정보 누출이 있다고 결론지을 수 있다.



[그림 1] 단계별 검증 결과의 의미

6. 결론 및 향후 과제

이 논문에서는 명령형 프로그램에서 정보 흐름의 안전성 분석을 모델검사 기법을 사용하여 검사하기 위해서 정보흐름의 안전성에 대한 CTL 논리식을 정의하고, SMV를 사용하여 프로그램의 정보 흐름의 안전성을 검사하는 방법을 제시하였다. 실제로 SMV로 검사 프로그램을 작성하여 실행시켜 보았고 예상했던 대로의 결과를 얻었다.

정보흐름의 안전성을 검사하기 위하여 검사해야 하는 CTL 논리식의 개수가 최악의 경우 2×블록개수×비밀변수개수×공개변수개수가 되어 비교적 많지만, 동적계획법(dynamic programming) 방식으로 단계별로 논리식을 검사하여 최종 결론에 도달하기 때문에 다항식시간(polynomial-time)에 검사를 끝낼 수 있다. 그러나 프로그램의 크기가 커지면 논리식의 개수가 많아지고, 따라서 자연히 수동으로 모두 처리하는 것은 현실적이지 않다. 따라서 프로그램이 주어지면 자동으로 SMV 검사 프로그램을 생성하는 도구가 필요하다. 이는 충분히 실현 가능하리라 생각되며, 다음 연구과제로 남겨둔다.

참고문헌

[1] J.-P.Banatre, C.Bryce, D. Le Metayer, Compile-time detection of information f low in sequential programs, Proc. Euro pean Symposium on Research in Comp uter Security, Lecture Notes in Comput er Science, vol.875, pp.55-73, 1994.

[2] E.M. Clarke, O. Grumberg and D.E. Lo ng, Model checking and abstraction, A CM Transactions on Programming Lan guages and Systems, 16(5):1512-1542, 1994.

[3] D.E. Denning, A lattice model of secur e information flow, Comm. ACM, 19(5), pp.236-243, 1976.

[4] D.E. Denning and P.J. Denning, Certific ation of programs for secure informatio n flow, Comm. ACM, 20(7), pp.504-513, 1977.

[5] K.-G. Doh and S.-C. Shin, "Analysis of secure information flow", draft, December 2001.

[6] R. Joshi and K.R.M. Leino, A semantic approach to secure information flow, S cience of Computer Programming, 37, p p.113-138, 2000.

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

[8] M. Mizuno and D. Schmidt, A security flow control algorithm and its denotatio nal semantics-based correctness proof, Formal Aspects of Computing 4, 727-7 54, 1992.

[9] M. M. Muller-Olm, D. Schmidt and B. Steffen, "Model-checking: A tutorial i ntroduction", In G. File and A. Cortesi, editors, Proc. 6th Static Analysis Symp osium, Lecture Notes in Computer Scie nce, Springer, 1999.

[10] D.A. Schmit, Data flow analysis is a model checking of abstract interpretatio n, In Proc. 25th ACM Symp. on Princi ples of Programming Languages, ACM Press, 1998.

[11] D. Volpano, G. Smith, C. Irvine, A so und type system for secure flow analy sis, J. Comp. Security 4(3), pp.1-21, 19 96.

조영갑

1990년～1996년 한양대학교 전자계산학과(학사)

2000년～현재 한양대학교 컴퓨터공학과(석사과정)

도경구

1976년~1980년 한양대학교 산업공학과(학사).
1984년~1987년 아이오와주립대 전산학(석사).
1987년~1992년 캔사스주립대 전산학(박사).
1993년~1995년 Univ. of Aizu 교수
1995년~현재 한양대학교 교수
2000년~현재 (주)스마트카드연구소 대표이사
관심분야는 프로그래밍언어, 프로그램 정확성 및 안전성 검증, 스마트카드

신승철

1983년~1987년 인하대학교 전자계산학과(학사).
1987년~1989년 인하대학교 전자계산학과(석사).
1992년~1996년 인하대학교 전자계산공학과(박사).
1999년~2000년 Kansas State Univ. 박사후과정 연구원
1996년~현재 동양대학교 컴퓨터공학부 조교수
관심분야는 프로그래밍 언어, 프로그램 분석, 프로그램 검증, 정형 기법.