정보흐름분석을 위한 BIRS의 확장

정지웅* 김제민* 김선태* 박준석* 유원희*
*인하대학교 컴퓨터정보공학부
jje0615@hanmail.net

The Expansion of BIRS for Information Flow Analysis

Ji-Woong Jung* Je-Min Kim* Seon-Tae Kim*
Joon-Seok Park* Weon-Hee Yoo*
*Dept. of Computer Science and Information Engineering, INHA University

요 약

프로그램에 대한 검증을 수행하기 위해서는 프로그램 안에서 여러 가지 분석이 이루어진다. 그 중에서 객체지향언어의 경우 분석을 위하여 객체지향적인 특성에 맞는 정보흐름분석을 사용해야 한다. 본 논문에서는 왜 객체지향언어의 특성에 맞는 정보흐름분석이 필요한지를 보이고, 객체지향적인 특성을 고려하지 않은 정보흐름분석 시에 어떤 문제가 생기는지 예시를 들어 설명한다. 그리고 자바의 검증 도구인 BIRS(Bytecode Intermediate Representation with Specification)언어를 대상으로 하여 객체지향언어의 특성을 고려한 정보흐름분석이 가능하도록 새로운 명세를 추가하여 확장한다.

1. 서론

소프트웨어의 안전한 실행을 보장하기 위하여 여러 가지 검증 도구들이 개발되고 있다. 검증 도구들에게 정확한 분석은 가장 중요한 요소 중 하나이다. 정확한 분석을 행하기 위해서는 각 특성에 따른 정보흐름분석이 필요한데, 그 중 객체지향언어에서는 객체지향적인 특성에 맞는 정보흐름분석이 이루어져야 한다.

정보흐름분석에서 주된 관심 중의 하나는 데이터의 기밀성을 보장하는데 있다. 데이터의 중요도가 높은 경우 외부로 정보가 유출되어서는 안 된다. 반대로 사용자가 필요로 하는 정보가 기밀로 취급되어 접근할 수 없는 경우가 생겨서도 안 된다. 정보흐름분석은 객체지향언어에서 잘못된 분석으로 인하여 데이터의 값이 혼동되지 않도록 막는다.

본 논문의 구성은 다음과 같다. 2장에서 정보흐름분석과 관련된 연구와 연구의 대상이 되는 BIRS(Bytecode Intermediate Representation with Specification)[1]에 대하여 간단히 소개하고, 3장에서 BIRS의 특징에 대해 알아본다. 4장에서는 객체지향언어의 특성을 고려한 정보흐름분석을 행하지 않을 경우의 문제를 제기한다. 그리고 5장에서 문제를 해결할 방법으로 특성에 맞는 정보흐름분석이 가능하도록 논리식을 세우고, 그에 따른 새로운 명세를 BIRS에 추가한다.

2. 관련연구

객체지향프로그램에서 정보흐름의 정책은 데이터의 기밀성에 관심을 갖는다[2].

전형적으로 데이터의 기밀성을 확인하는 설정은 보안 수준과 관계가 있다. 예를 들어, 민감하고 중요한 채널을 위한 높은 수준과 대중적인 채널을 위한 낮은 수준, 그리고 (다른 허가수준의)입력채널에 도달하는 데이터를 조작하고 (다른 허가수준의)출력채널로 흐르는 결과를 생성하는 프로그램이 있다고 가정하자. 이 설정에서 프로그램의 실행 중에 데이터가 높은 수준에서 낮은 수준으로 흐르지 않는다면 데이터의 기밀성은 확실하게 보장받는다. 즉, 기밀성 유지를 위해서는 낮은 수준을 가진 채널이 높은 수준을 가진 채널에 영향을 받아서는 안 된다. 이 개념을 객체지향언어의 문맥에서 '독립성(Independence)'이라고 명시한다.

BIRS는 자바 프로그램 검증만을 위한 중간 표현 언어로서 자바 바이트 코드를 입력받아 검증코드를 생성한다. BIRS코드는 스택 코드로 이루어진 자바 바이트 코드로부터 Sawja(Static analysis workshop for java)[3] 라이브러리를 이용해 검증이 용이하도록 변환된 스택리스 코드이다. 구현 언어로는 안정성과 신뢰성을 갖춘 OCAML(The Objective Caml language)[4]을 사용하였다.

3. BIRS 중간표현 언어

본 논문에서는 BIRS를 대상으로 한다. 관심을 가져야 할 부분은 LogicalExpr, ComputationFunction, Body, Command 4 부분으로 나눌 수 있다.

그림 1은 자바에서 표현되는 모든 논리식을 정의하고, 검증조건을 생성하기 위한 형태로 표현되어있는 LogicalExpr부분이다.

```
LogicalExpr ::==   T | F | "¬" LogicalExpr
                  |LogicalExpr RelOp LogicalExpr
                  |LogicalExpr BoolOp LogicalExpr
                  |LogicalExpr "⇒" LogicalExpr
                  |LogicalExpr "≡" LogicalExpr
                  |"forall"TypeIdList "." LogicalExp
                  |"exists"TypeIdList "." LogicalExpr
```

(그림 1) BIRS의 LogicalExpr

그림 2는 자바로 된 프로그램의 함수에서 수행되는 명령어에 대한 정보를 표현하는 ComputationFunction부분이다. 그림 2에서는 함수의 이름과 파라미터 타입, 변수 이름, 반환 타입 그리고 추후 검증에 사용될 필요조건과 보장조건을 표현한다.

```
ComputationFunctionDec ::==
        computationfunction Id : (ParamList)
        →Type{ [require LogicalExpr]
        [read (x,....,x)] [write (y,....,y)]
        Body [ensure LogicalExpr]}
```

(그림 2) BIRS의 ComputationFunction

그림 3은 함수의 주요 코드를 표현하고 명령어를 블록 단위로 표현하며, Flow를 통해 흐름을 나타내는 Body와 Body를 이루는 구성단위의 표현식이다.

```
Body     ::==   [LocalVariable] Block+ Flow+
                Return+
Block    ::==   block Label : Command+
Flow     ::==   from Label to Label
                [when BooleanExpression]
Return   ::==   returnblock Label
Label    ::==   Id
```

(그림 3) BIRS의 Body 및 구성단위

그림 4는 모든 자바코드의 명령어를 표현할 수 있는 Command의 BNF이다. Command는 Index를 기준으로 표현하며 판독성을 높이기 위해 자바 소스 코드의 문법을 참조하여 설계되었다.

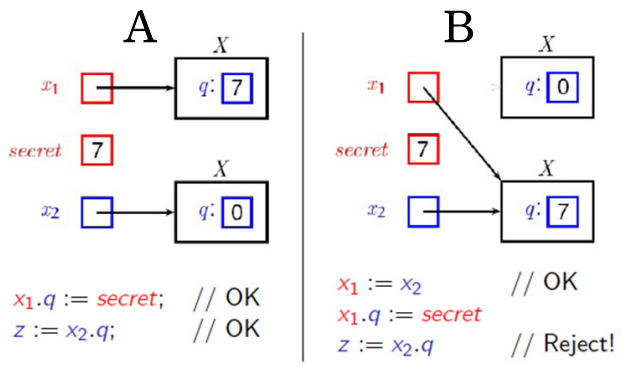
```
Command ::==   Index : Instr
Index   ::==   Id
Instr   ::==
        assignVal VariableId := Expression
        |referenceObj VariableID := Expression
        |modArray Expression [Expression]
          := Expression
        |nop
        |affectField (Expression . Fieldname :
        ClassId)   := Expression
        |affectStaticField (Fieldname : ClassId)
          := Expression
        |goto Index
        |ifd (Expression Op Expression) Index
        |Throw Expression
        |return
        |vreturn Expression
        |new VariableId
          := new ClassId (Expression)
        |newarray VariableId
          :=  Type Expression
        |invokeVirtual VariableId
          :=  ClassId . MethodId Expression
        |invokeVirtual
          ClassId . MethodId Expression
        |invokeNonVirtual VariableId :=
        Expression.ClassId.MethodId Expression
        |invokeNonVirtual
        Expression.ClassId.MethodId Expression
        |monitorEnter Expression
        |monitorExit  Expression
        |maylnit VariableId
        |assert LogicalExpr
        |assume LogicalExpr
```

(그림 4) BIRS의 Command

4. 객체지향언어에서의 참조와 별칭문제

그림 5는 서로 다른 객체와의 정보흐름을 나타내고, 객체사이에서 발생할 수 있는 경우에 대한 예제이다.

그림 5와 같은 상황에서 객체지향적인 특성을 고려한 정보흐름분석이 행해지지 않는다면, 아래와 같이 객체의 별칭에 따른 문제가 발생하게 된다.



(그림 5) 객체의 정보흐름분석 예제

예제 A(좌측)에서는 각각의 객체가 서로 다른 값을 참조하고 있다. 예제 A에서 x1이 가리키는 q에는 secret값이 들어있고, x2가 가리키는 q값은 x1.q값과는 관계가 없다. 따라서 정보흐름상 안전하다고 할 수 있다. 반면에 예제 B(우측)에서는 x1과 x2가 같은 q값을 참조하고 있다. 단순한 비교연산에 의한 분석을 하게 된다면, 예제 B의 x1.q와 x2.q가 같은 필드 값으로 인식되어 두 변수가 같은 값을 참조하는 것으로 혼동하게 된다. 두 예제의 상황을 구분해내기 위해서는 객체지향언어의 특성을 고려한 정보흐름분석이 필요함을 알 수 있다.

BIRS도 객체지향언어인 자바를 기반으로 만들어진 언어이기 때문에 객체지향적인 특성을 고려한 정보흐름분석이 필요하다.

5. 명세가 추가된 BIRS

BIRS에 명세를 추가함에 있어서 표현해야할 정보는 크게 4 가지로 볼 수 있다.

4 가지 정보는 각각의 객체가 서로 다른 필드를 참조함을 보증하는 정보, 보증된 필드의 구체적인 위치(메모리상의)로부터 정보의 누수가 발생하지 않음을 보증하는 정보와 이러한 필드와 필드의 위치를 참조하는 객체에 대한 정보들로 이루어진다.

필요한 표현정보를 문법으로 정의하는 과정은 **True, False** 같은 기본적인 문법은 남겨둔 상태에서 객체에 대한 참조와 별칭 정보를 포함하는 문법을 추가하는 방식으로 이루어졌다.

그림 6은 객체지향 특성을 고려하여 확장한 문법을 완성한 모습이다.

LogicalExpr ::== **T** | **F** | "¬" *LogicalExpr*
|*LogicalExpr RelOp LogicalExpr*
...
| **VariableId ⇝ L**
| **L.f ⇝ L'**
| **VariableId**$_{\ltimes}$
| **L.f**$_{\ltimes}$

(그림 6) 확장된 LogicalExpr

L은 추상적인 위치 l들의 집합이라고 정의하고, L.f는 l의 위치에 있는 필드 f라는 의미를 가지고 있다고 하자.

"VariableId ⇝ L"는 VariableId에 의해 명시된 구체적인 위치를 L이 추상화한다는 뜻이고, **"L.f ⇝ L'"**는 L''에 의해 추상화된 모든 구체적인 위치에 대하여, 만약 l.f가 l'를 포함한다면, l'는 L'에 의해 추상화된다는 뜻이다.

"VariableId$_{\ltimes}$**"**는 VariableId가 그림 5와 같이 값을 가지는 두 가지 상황에서, VariableId에 의해 정보가 누출되지 않음을 보장한다는 것을 뜻한다. 위의 명세를 사용하여 자바의 검증에 필요한 정보흐름분석이 이루어진다.

6. 결론 및 향후 연구과제

본 논문에서는 객체지향언어의 특성을 고려하여 정보흐름분석이 이루어져야 함을 예를 들어 설명함으로써 BIRS 언어에 새로운 명세를 추가해야 함을 보였다. 그리고 객체지향언어의 특성을 고려한 정보흐름분석이 가능하도록 BIRS의 명세를 추가하였다. 추가된 명세에 의해 정확성이 높아진 정보흐름분석이 가능해지고, 올바른 검증이 이루어지도록 프로그램의 안정성과 신뢰도에 기여했다고 생각한다.

이후에는 확장된 BIRS에 의한 실제적인 성능 변화를 실험해 보고 논리적 평가를 해나갈 계획이다.

논문 사사

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (No. 2011-0026495).

참고문헌

[1] 김선태, 김제민, 박준석, 유원희, "자바 프로그램 검증을 위한 스택 리스 중간 표현 언어 생성기 구현", 한국정보기술학회, 2011년 9월 30일.

[2] Torben Amtoft, Sruthi Bandhakavi, Anindya Banerjee, "A Logic for Information Flow in Object-Oriented Programs", POPL'06, January 11-13, 2006.

[3] Laurent Hubert, Nicolas Barré, Frédéric Besson, Delphine Demange, Thomas Jensen, Vincent Monfort, David Pichardie, Tiphaine Turpin, "Sawja: Static analysis workshop for java", Formal Verification of Object-Oriented Software (FoVeOOS), June 2010.

[4] The Objective Caml language, http://caml.inria.fr/.