프로그램 분석 기초

최광훈

September 13, 2023

Chapter 1

프로그램 분석 개요

1.1 목적

프로그램을 개발하거나 프로그램의 보안 취약점을 탐지할 때 다양한 프로그램 분석 방법을 사용하여 개발 생산성을 높이거나 꼼꼼하게 보안 취약점을 탐지하고 있다. 특정 프로그램 분석을 지원하는 오픈소스 소프트웨어도 다양하게 개발되어 개발자나 보안 담당자가 이러한 프로그램을 사용고 있다. 하지만 프로그램 분석에 대한 기초 지식이 없다면 이 프로그램을 제대로 사용하는데도 한계가 있다. 프로그램 분석에 대한 기초 지식을 갖추고 있다면 이러한 프로그램을 사용하는 것뿐만 아니라 자신의 목적에 맞추어 수정하는 것도 가능할 것이다. 하지만 프로그램 분석을 직접 배울곳이 마땅치 않고 프로그램 분석 기법과 밀접하게 관련된 프로그래밍언어론과 컴파일러를 공부할 수 있는 곳도 제한되어 있다. 설사 그러한 과목들을 배울 수 있다하더라도 프로그램 분석 기법을 배우는데 한계가 있다.

프로그램 분석은 어려운 주제이다. 여러 이유를 생각해볼 수 있다. 우선, 모든 프로그램 분석 기법은 복잡한 수학에 기반을 두고 있기 때문이다. 그리고, 분석 대상이 되는 프로그램을 작성한 프로그래밍언어 자체가 복잡하기 때문이다. 마지막으로 우리가 분석하고 싶은 품질 속성이 다양하기 때문이다.

프로그램 분석 기법을 쉽게 익힐 수 있도록 이 강의 노트를 구성하고자 한다. 우선, 수학을 사용하여 프로그램 분석을 설명하는 것을 최소한으로 제한하고 실행 가능한 명세를 통해서 설명한다. 그리고 간단한 프로그래밍언어 WHILE을 정의

하여 이 언어로 작성된 프로그램을 대상으로 분석 기법을 설명한다. 마지막으로 어휘 분석, 구문 분석, 동적 의미, 타입 체킹, 정적 분석, 기호 실행으로 분석 기법을 한정하여 설명한다.

1.2 미리 알아두면 좋을 내용

프로그램 분석 기법에 대한 이 강의 노트를 공부하기 위해서 자료 구조와 기초 프로그래밍 지식이 필요하다. 리스트, 트리, 그래프와 같은 자료 구조를 이해하고 프로그래밍할 수 있어야 한다.

하스켈 프로그래밍언어로 프로그램 분석 기법에 대한 실행 가능한 명세를 작성한다. 이 언어를 사용한 이유는 수학 표기법을 사용하여 명세를 작성한 것에 비해 접근하기 쉽고, 하스켈 프로그램이 간결하여 명세로 삼기에 적합하기 때문이다. 표 1.1에 각 분석 방법과 그 방법을 구현한 하스켈 프로그램의 라인 수를 볼 수 있다. 88 라인에서 250 라인으로 여섯 가지 분석 방법을 작성하였다.

분석 기법 명세	라인
어휘 분석	88
구문 분석	228
동적 의미	123
정적 의미(타입 검사)	145
정적 분석	250
기호 실행	208

Table 1.1: 분석 방법과 하스켈로 작성한 실행 가능한 명세의 라인 수

하스켈 프로그래밍언어에 대한 공식 사이트는 https://haskell.org이다. 다양한 레퍼런스를 이 웹 사이트에서 찾을 수 있다. 하스켈 기초 프로그래밍을 배우기 위해서 헬싱키 대학에서 만든 무크 사이트를 이용할 수 있다. 이 무크 사이트 내용으로 구성된 유튜브 동영상 강의도 하스켈 프로그래밍을 배우기에 도움이 될 것이다.

여담이지만 하스켈과 같은 정통 함수형 프로그래밍언어를 한번 배우는 것은 설사 객체지향 프로그래밍언어를 주로 사용하는 사람들에게도 도움이 될 것이다. 최근 파이썬, C++, 자바, 자바스크립트, 코틀린, 스위프트와 같은 객체지향 프로그래밍언어에 람다, 다형 타입 (제네렉, 템플릿), 변경 불가능한 값(immutable value), 리스트 제시법(list comprehension)과 같은 함수형 프로그래밍언어의 핵

심 특징들을 도입하고 있는 추세이다. 객체지향 프로그래밍 언어의 구문에 함수형 프로그래밍언어 특징을 구겨넣다보니 구문도 복잡하고 그 의미도 이해하기 어려운 경우도 있다. 정통 함수형 프로그래밍언어에서 그 특징들을 경험한다면 파이썬 프로그래밍을 할때도 프로그래밍에 대한 관점이 달라질 것이다.

이 외에 집합(set)이나 릴레이션(relation)과 같은 기초적인 수학 개념을 알 필요가 있다.

1.3 이 강의 노트로 공부하는 방법

이 강의 노트를 공부하는 방법은, 하스켈 프로그램으로 미리 준비해놓은 실행 가능한 명세들을 독자가 사용하는 프로그래밍언어로 다시 작성함으로써 각 분석 방법을 주체적으로 이해하는 방법을 권한다. 이 강의 노트에서는 파이썬을 선택하여 설명한다.

강의 노트에서 사용하는 하스켈 프로그램의 내용은 다음과 같다.

- WHILE 프로그래밍 언어 예제 프로그램
- 어휘 분석(lexical analyzier)
- 구문 구조 분석(Parser)
- 동적 의미(semantics)
- 정적 의미-타입체킹
- 정적 분석-자료 흐름 분석
- 기호 실행

이 프로그램을 깃허브 저장소에서 내려 받고, 빌드한다.

기본적으로 하스켈 빌드 시스템 도구 스택(stack)을 설치해야 한다. 추가로 기호 실행 분석에서 사용하는 산술 복합 논리식을 해결하는 풀이 엔진 (SMT - Satisfiability Module Theories - solver) Z3 라이브러리가 필요하다.

- 리눅스

```
$ sudo apt-get install libz3-dev
$ git clone https://github.com/kwanghoon/Lecture_SAV
$ cd Lecture_SAV/whilelang
$ stack build
```

- 윈도우즈
  - z3-4.8.12 버전을 내려받기
  - (D:\z3-4.8.12-x64-win 디렉토리 아래 bin에 라이브러리, include 에 헤더 파일이 있다고 가정)

```
D:\> git clone https://github.com/kwanghoon/Lecture_SAV
D:\> cd Lecture_SAV/whilelang
D:\> stack build
       --extra-include-dirs=D:\z3-4.8.12-x64-win\include
       --extra-lib-dirs=D:\z3-4.8.12-x64-win\bin
```

하스켈 프로그램을 빌드한 다음 어휘 분석부터 기호 실행까지 실행하는 방법은 다음과 같다. 실행 파일을 통해 각 분석 방법을 다음과 같이 실행할 수 있다.

```
$ stack exec -- whilelang-exe --lex ./example/while2.while
$ stack exec -- whilelang-exe --parse ./example/while2.while
$ stack exec -- whilelang-exe --typecheck ./example/while2.while
$ stack exec -- whilelang-exe --dataflow ./example/while2.while
$ stack exec -- whilelang-exe --symexec ./example/while2.while
```

하스켈 프로그램을 read-eval-print 방식으로 실행할 수도 있다.

```
$ stack ghci --
ghic> let srcFile = "./example/while2.while"
ghci> doLexing srcFile
ghci> doParsing srcFile
ghci> doRun srcFile
```

```
ghci> doTypecheck srcFile
ghci> doAnalysis srcFile
ghci> doSymbolic srcFile
```

WHILE 프로그램을 하스켈 어휘 분석과 구문 구조 분석 결과로 얻은 추상 구문 트리(AST, Abstract Syntax Tree)를 다른 프로그래밍언어에서 사용하기 위해서 이 트리를 JSON 형식으로 출력하는 방법을 제공한다.

```
$ stack exec -- whilelang-exe --json ./example/while2.while
```

```
$ stack ghci --
ghic> let srcFile = "./example/while2.while"
ghci> doJson srcFile
```

파이썬 프로그램에서 이 JSON 형식의 분석 내용을 읽어서 원하는 분석을 진행할 수 있다.

Chapter 2

While 프로그래밍언어

```
-- Program
data Prog = Prog { progDecls :: Decls, progComms :: Comms }

type Decls = [ Decl ]
type Comms = [ Comm ]

type Decl  = (Type, VarName)

-- Type
data Type =
    TyInt
  | TyBool

type VarName  = String

-- Statement
data Comm =
    CSkip
  | CSeq Comm Comm
  | CAssign VarName Expr
  | CRead VarName
  | CWrite Expr
  | CIf Expr Comm Comm
  | CWhile Expr Comm
  | CAssert Expr

-- Expression
data Expr =
    ECst    Const
  | EVar    VarName
  | EBinOp Op Expr Expr
  | EUnaryOp Op Expr

data Const =
    CInt  Int
```

```
  | CBool Bool

data Op =
    OpAdd
  | OpSub
  | OpMul
  | OpDiv
  | OpMod
  | OpLessThan    -- x < y
  | OpEqual       -- x == y
  | OpAnd
  | OpOr
  | OpNot
```

Chapter 3

구문 분석

Chapter 4

의미 분석

Chapter 5

정적 분석

Chapter 6

기호 실행

Chapter 7

맺음말