정적 분석기를 골탕 먹이는 입력 프로그램 합성 기술

(Program Synthesis for Attacking Static Analyzers)

지도교수 : 이광근

이 논문을 공학학사 학위 논문으로 제출함.

2025년 10월 29일

서울대학교 공과대학

컴퓨터공학부

정원준

2026년 2월

정적 분석기를 골탕 먹이는 입력 프로그램 합성 기술

(Program Synthesis for Attacking Static Analyzers)

지도교수 : 이광근

이 논문을 공학학사 학위 논문으로 제출함.

2025년 10월 29일

서울대학교 공과대학

컴퓨터공학부

정원준

2026년 2월

국문초록

프로그래밍 언어와 그 실행 의미, 정적 분석기가 주어졌을 때 정적 분석기가 무의미한 결론을 내리게 하는 입력 프로그램을 합성한다. 무의미한 결론이라 함은, 프로그램의 특정 위치에서 특정 변수가 가질 수 있는 값에 대한 정보가 아무것도 없는 것을 의미한다. 본 연구에서는 적은 종류의 규칙을 가지는 G 언어와 그에 해당하는 한 정적 분석기를 고정하여 분석기에 대한 공격을 시도한다. 이러한 공격이 왜 항상 가능한지 증명하고, 전탐색에 기반한 방법으로 실제 공격 예시를 제시한다. 이후 프로그램 합성 기법을 이용하여 유의미한 실행 시간 안에 공격을 시도한다. 이 공격의 결과 통해 정적 분석기의 불완전한 부분을 파악하여 분석기를 보완하는데 도움을 줄 수 있다. 실행 의미는 같지만 정적 분석기가 다른 결론을 내리는 프로그램을 만들어 난독화 기법으로 사용될 수 있다.

주요어: 정적 분석, 프로그램 합성, 요약 해석, 난독화

목 차

제 1 장 배경 지식

제 1 절 정적 분석

정적 분석은 프로그램을 실행시키지 않고(혹은 실행시키기 전에) 프로그램의 행동 혹은 성질을 예측하는 것이다. 프로그램의 실행은 무한할 수 있지만, 정적 분석의 실행은 무한하지 않다. 프로그램이 입력을 받지 않고 실행되는 경우만 생각하자. 프로그램의 성질로 정할 수 있는 것은 다양하다. 실행 중 에러가 나진 않는지, 프로그램이 종료되는지, 특정 줄이 실행되는지, 변수 x가 값 v를 가질 수 있는지 등이 있다. 만약 정적 분석이 프로그램의 모든 성질을 완벽하게 예측할 수 있다면, 프로그램을 실행시킬 필요가 없어진다. 무한한 시간에 걸친 계산을 유한 시간 내에 끝낼 수 있게 되는 것이다. 그러나 정지 문제는 계산 불가능한 문제이기에, 프로그램이 종료된다는 성질 하나조차 완벽하게 예측할 수 없다.

제 2 절 안전함과 완전함

프로그래밍 언어 $\mathbb{L}$과 프로그램의 성질 $\mathscr{P} \subseteq \mathbb{L}$, $\mathscr{P}$를 분석하는 정적 분석기 $\mathcal{A}$에 대해 안전함$_{soundness}$과 완전함$_{completeness}$는 다음과 같의 정의된다:

$$\mathcal{A}\text{가}\mathscr{P}\text{에 대해 안전하다} : \forall p \in \mathbb{L}, \mathcal{A}(p) = true \implies p \in \mathscr{P}$$
$$\mathcal{A}\text{가}\mathscr{P}\text{에 대해 완전하다} : \forall p \in \mathbb{L}, \mathcal{A}(p) = true \impliedby p \in \mathscr{P}$$

제 3 절 Rice의 정리

정리 3.1 (Rice's Theorem, Rice 1951) φ를 부분 계산 가능한 함수들의 허용 가능한 넘버링$_{admissible\ numbering}$이라고 하자. $P \subseteq \mathbb{N}$을 어떤 성질을 만족하는 프로그램들의 인덱스 집합이라고 하자. 다음 두 조건을 만족한다고 가정하자:

1. P는 **비자명(*non-trivial*)**하다. 즉, $P \neq \varnothing$ 이고 $P \neq \mathbb{N}$이다.
2. P는 **확장적(*extensional*)**이다. 즉, 모든 $m, n \in \mathbb{N}$에 대해 $\varphi_m = \varphi_n$이면 $m \in P \iff n \in P$이다.

그렇다면, P는 **결정 불가능(*undecidable*)**하다.

이 정리를 우리 문제 상황에 맞게 가공하면 다음과 같다:

따름정리 3.2 $\mathbb{L}$을 튜링 완전한 프로그래밍 언어라고 하자. $\mathscr{P} \subseteq \mathbb{L}$을 프로그램의 부분집합, 혹은 성질이라 하자. 다음 두 조건을 만족한다고 가정하자:

1. $\mathscr{P}$는 **의미 있는 성질**이다. 즉, $\mathscr{P} \neq \varnothing$ 이고 $P \neq \mathbb{N}$이다.
2. $\mathscr{P}$는 **실행 의미에 딱맞는 성질**이다. 즉, $\forall p_1, p_2 \in \mathbb{L}$에 대해 p_1과 p_2의 실행 의미가 같으면 $p_1 \in \mathscr{P} \iff p_2 \in \mathscr{P}$이다.

그러면 다음과 같은 분석기 $\mathcal{A}$는 **존재하지 않는다**:

$$\forall p \in \mathbb{L}, \mathcal{A}(p) = true \iff p \in \mathscr{P}$$

다른 말로, 안전$_{sound}$하면서 완전한$_{complete}$ 분석기를 만들 수 없다.

제 4 절 프로그램 합성

중간보고서 작성 이후 작성 예정이다

제 2 장 문제 정의

Rice의 정리에 의해, 정적 분석기는 일반적으로 안전하면서 완전할 수 없다. 다른 말로 만약 어떤 정적 분석기가 안전하다면, 그 분석기는 완전하지 못하게 된다. 그 분석기가 어떤 프로그램을 입력으로 받았을 때 어떤 결함이 있는지 찾는 것이 목표이다.

일관된 공격 목표를 위해 성질 $\mathscr{P}$와 안전한 분석기 $\mathcal{A}$의 결함을 다음과 같이 정의한다.

$\mathscr{P}$:= '변수 x가 가질 수 있는 값은 $\mathbb{Z}$ 전체가 아니다.' 실제로 변수 x가 가질 수 있는 값이 $\mathbb{Z}$ 전체일 경우, $\mathcal{A}(p) = false$가 성립한다.(안전함의 대우) 그러나 $\mathcal{A}(p) = false$라고 해서 실제로 변수 x가 가질 수 있는 값이 $\mathbb{Z}$ 전체인 것은 아니다. 이런 경우 $\mathcal{A}$는 x가 가질 수 있는 값의 집합에 대해 아무 말도 하지 못한다. 즉, $\mathcal{A}$는 x에 대해 $\top$ 이라는 결과를 내놓는다. 어떤 정수 $n \in \mathbb{Z}$에 대해 $x = n$일 때 에러가 나게끔 프로그램을 짰다고 생각해 보자. 실제로는 이 프로그램에 에러가 나지 않지만 $\mathcal{A}$는 가짜 경보를 울릴 것이다. 이것이 $\mathcal{A}$의 결함이다.

이 결함의 정의에 딱 맞게 공격하기 위해서 언어 $\mathbb{L}$에 몇 가지 제약 조건을 준다.

1. 튜링 완전해야 한다. Rice's Theorem을 적용하기 위해서다.
2. 메모리와 함께 동작하고, 그 주소는 변수 이름이어야 한다. 'x가 가질 수 있는 값'을 잘 다루기 위해서다.
3. 부수적인 효과*side effect*가 없어야 한다.
4. 무엇 의미구조*denotational semantics*로 그 실행 의미가 정의되어야 한다.

이런 조건 하에서 목표를 다음과 같이 정하였다.

위 조건을 만족하는 언어 $\mathbb{L}$과 안전한*sound* 정적 분석기 $\mathcal{A}$가 주어졌을 때, 변수 x를 사용하는 프로그램 $p \in \mathbb{L}$을 만들어 $\mathcal{A}$의 결함을 밝혀낸다.

이때 정적 분석기의 내부 구조에 접근 가능한 상태에서 화이트박스 공격을 수행한다.

제 3 장 가능성

정적 분석에 대한 공격의 가능성은 Rice Theorem에 기반한다.

결함을 밝혀내는 프로그램 $g \in \mathbb{L}$ 이 존재하는 것과 실제로 그 프로그램을 찾는 일반적인 알고리즘을 만드는 것은 다른 문제이다. 다행히도, 일반적인 알고리즘이 존재한다.

프로그램은 결국 문자열이고, 문자열의 집합은 셀 수 있는 집합이므로 그 부분집합인 프로그램의 집합도 셀 수 있다. $\mathbb{L}$에 속하는 프로그램 모두에 1부터 번호를 붙여 일대일 대응 $\mathcal{I} : \mathbb{L} \to \mathbb{N}$을 만들면, $\mathcal{I}(g) \in \mathbb{N}$ 이다. 1번째 프로그램부터 순서대로 만들어나가면서

어? 안되네? i번째 프로그램을 분석기에 돌려보는건 되는데 실제로 top이 맞는지 아닌지 알 수가 없잖아.

제 4 장 기대효과

제 1 절 분석기 보강

제 2 절 난독화

제 5 장 G 언어

제 1 절 문법

G 언어의 문법은 다음과 같다:

$$
\begin{array}{rcl}
P & \to & C \\
\\
C & \to & x := E \\
 & | & C; C \\
 & | & \text{ifp } E\ C\ C \\
 & | & \text{while } E\ C \\
\\
E & \to & n \quad (n \in \mathbb{Z}) \\
 & | & x \\
 & | & E{+}E \\
 & | & E{*}E \\
 & | & {-}E
\end{array}
$$

이 때 P는 프로그램, C는 커맨드, E는 식을 의미한다. 모든 식은 정수값을 갖는다. (3) **가능성** 에서 사용한 전략들을 적용하기 용이한 언어로 구성하였다. while 문을 통해서 튜링 완전을 확보하였다. 이 내용은 G 언어의 실행의미를 정의한 후에 더 자세히 다루겠다.

제 2 절 실행의미

G 언어의 실행 의미는 무엇 의미구조$_{denotational\ semantics}$로 정의한다. 실행의미가 정의되는 규칙은 다음과 같다:

각 문자의 타입

$$
\begin{aligned}
Val &= \mathbb{Z} \\
n &\in Val \\
M \in Mem &= Var \xrightarrow{\text{fin}} Val \\
x &\in Var \\
P, C &\in Cmd \\
E &\in Exp \\
[\![\cdot]\!] &: Cmd \to Mem \to Mem \\
&+ Exp \to Mem \to Val
\end{aligned}
$$

$[\![E]\!]$의 의미

$$
\begin{aligned}
[\![n]\!]M &= n \\
[\![x]\!]M &= M(x) \\
[\![E_1{+}E_2]\!]M &= [\![E_1]\!]M + [\![E_2]\!]M \\
[\![E_1{*}E_2]\!]M &= [\![E_1]\!]M \times [\![E_2]\!]M \\
[\![\text{-}E]\!]M &= -[\![E]\!]M
\end{aligned}
$$

$[\![C]\!]$의 의미

$$
\begin{aligned}
[\![x := E]\!]M &= M\{x \mapsto [\![E]\!]M\} \\
[\![C_1; C_2]\!]M &= [\![C_2]\!]([\![C_1]\!]M) \\
[\![\text{if}\ E\ C_1\ C_2]\!]M &= \mathit{if}\ [\![E]\!]M > 0\ \mathit{then}\ [\![C_1]\!]M\ \mathit{else}\ [\![C_2]\!]M \\
[\![\text{while}\ E\ C]\!]M &= \mathit{lfp}\ (\lambda f.\mathit{if}\ [\![E]\!]M > 0\ \mathit{then}\ f([\![C]\!]M)\ \mathit{else}\ M)
\end{aligned}
$$

이 때, 전체 프로그램의 실행의미는 $[\![P]\!]\emptyset$으로 정의된다.

$[\![\text{if}\ E\ C_1\ (C_2\ C_3)]\!]M = [\![\text{if}\ E\ C_1\ C_2]\!]M$

G 언어는 튜링 완전한가? ...

입력을 통해 만들어지는 분기가 없는 G 언어도 $\top$을 만들어내는 것이 가능하다. G 언어에 속하는 다음

프로그램을 생각하자.

```
1  x := 0;
2  while 1
3    ifp x
4      x := -x
5      x := -x + 1
```

이 프로그램을 실행하면 x의 값은 $0, 1, -1, 2, -2, 3, -3, \cdots$ 로 변해 간다. 따라서 2번 줄에서 x가 가질 수 있는 값은 $\mathbb{Z}$ 전체이다. 어떤 요약해석이던 $\mathbb{Z}$를 요약하면 그 결과는 $\top$이 나올 수밖에 없다.

top이 잘 나온다는걸 설명해야겠다 denotational이 맞는가?

제 6 장 문제 축소

제 7 장 분석기 고정

제 8 장 전탐색

제 1 절 구현

중간보고서 작성 이후 구현 예정이다

제 9 장 결과

구현 이후 작성 예정이다

제 10 장 결론

구현 이후 작성 예정이다

Abstract

Given a programming language, its operational semantics, and a static analyzer, we synthesize an input program that causes the static analyzer to yield a meaningless conclusion. A meaningless conclusion refers to a state where there is no information about the possible values of a specific variable at a specific location in the program. In this study, we fix the G language, which has a small set of rules, and a corresponding static analyzer, to attempt an attack on the analyzer. We prove why such an attack is always possible and present a concrete example of the attack using a full-search-based method. Subsequently, we attempt the attack within a meaningful execution time using program synthesis techniques. The results of this attack can help identify incomplete aspects of the static analyzer, thereby aiding in its refinement. By creating programs that have the same operational semantics but lead to different conclusions by the static analyzer, they can be used as an obfuscation technique.

Keywords: Static Analysis, Program Synthesis, Abstract Interpretation, Obfuscation