発表概要

JVM上の動的言語のための抽象解釈

馬谷 誠二[1,a)]

2018年11月1日発表

言語仕様が厳密には規定されていないプログラミング言語上でプログラムを書く場合，実行時の振舞いを正確に把握するには，リファレンス実装を使って実際に実行せざるを得ない．同様に，言語仕様が厳密に規定されていない JVM 上の言語で書かれたプログラムの性質を静的に解析しようとすれば，通常，リファレンス実装（ほとんどの場合，バイトコードへのコンパイラ）の生成したバイトコード命令列を解析する以外に選択肢がない．JVM バイトコードを対象とする静的解析器は数多く存在する．それらの解析器は，Java や Scala のような静的型付き言語からコンパイルされたバイトコード命令列に対しては上手く機能する．一方，動的な言語からコンパイルされたバイトコード命令列に適用すると，ほとんどの解析器は有用な情報を取得することができない．そのような精度の低下の主な原因は，動的言語の動的な振舞いを実現するために用いられる実行時機構の複雑さにある．本発表では，既存のバイトコードレベルの抽象解釈フレームワークを用いた動的言語向け抽象解釈器の構築方法を提案する．我々の抽象解釈器は，抽象解釈と「具象解釈」を混ぜながら実行することにより，バイトコードレベルの解析技術に依存しながらも，動的言語の振舞いをある程度正確に把握することが可能である．提案する方法を使って Clojure 言語のための抽象解釈器の構築を実際に行った．さらに，標準ライブラリ中のいくつかの関数定義を用いて，提案する構築方法の有効性を確認した．

Presentation Abstract

Abstract Interpretation for JVM-hosted Dynamic Languages

Seiji Umatani[1,a)]

Presented: November 1, 2018

When writing a program in a programming language whose semantics is not specified precisely, we have to actually execute our program using the reference implementation in order to reason about its behavior. Similarly, if we want to analyze the characteristics of a program written in a JVM-hosted language whose semantics is not precise, we usually have no choice other than analyzing bytecode instructions generated by the reference implementation (i.e., a bytecode compiler). There are many available static analyzers targeting JVM bytecode. They work well for bytecode instructions compiled from a statically-typed language such as Java and Scala. On the other hand, when applied to the ones compiled from a dynamic language, most analyzers fail to obtain useful information. Such low accuracy is mainly caused by the complexity of runtime mechanisms used for realizing dynamic behavior of the dynamic language. In this presentation, we propose a method for building an abstract interpreter that exploits an existing bytecode-level abstract interpretation framework. While relying on the bytecode-level analysis, our abstract interpreter can understand the exact behavior of programs written in a JVM-hosted dynamic language to some extent. We actually implemented an abstract interpreter for Clojure using the proposed method and confirmed its effectiveness by applying the abstract interpreter to several functions defined in the standard library.

This is the abstract of an unrefereed presentation, and it should not preclude subsequent publication.

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University, Kyoto 606–8501, Japan
[a)] umatani@kuis.kyoto-u.ac.jp