ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Proposal for a Master's thesis

Signature Inference for QuickSpec

Supervisors: Dr. Dmitriy Traytel
Professor: Prof. David Basin
Issue date:

Prerequisites

- Very good skills in functional programming (e.g., from the FMFP course)

Introduction

Formal specifications are a powerful tool for understanding programs. For example, if we know that `filter p (xs ++ ys) = filter p xs ++ filter p ys`, we know that filter acts pointwise on its input: to understand what `filter` does, we need only understand what it does to singleton lists. When a library comes with a specification, the user need not guess how the library behaves: they can work it out for themselves.

Automated testing tools like QuickCheck [3] are designed to efficiently test whether a Haskell program satisfies a given specification. But often it is tedious and time consuming to come up with a specification in the first place.

QuickSpec [4, 6] is a tool that automatically discovers equational properties of Haskell programs by testing. The discovered properties are often good candidates to be used as a formal specification. QuickSpec inputs a maximum term size and a *signature*—a set of Haskell functions—from which it constructs equalities between terms consisting only of functions belonging to the signature and tests those equalities using QuickCheck. The equalities that pass the QuickCheck tests constitute QuickSpec's output.

QuickSpec's scalability is tightly connected to the term size and the size of the signature. It can cope well with small terms (typically made from < 10 symbols of variables) and small signatures (typically with < 10 functions). The authors of QuickSpec [6] observe that:

> The skill in using QuickSpec lies in knowing which functions to include in the signature, including auxiliary functions, and sometimes which to leave out.

Objectives

The goal of this thesis is to further improve the degree of automation of QuickSpec, yielding a new tool EasySpec.

We envision a setting in which the programmer is interested in discovering properties of one particular function `f`. Then `f` should be the *only* input to EasySpec. The tool should automatically select one or several signatures, which then can be given as input to QuickSpec. Additionally, QuickSpec must be instrumented to only generate properties that include `f`.

Selecting the signatures in a meaningful way is the main challenge of this project. We call this problem the *signature inference*. Our approach to signature inference will borrow ideas from relevance filtering used in the context of combining interactive and automatic theorem provers [2,5]. There, given a conjecture, a large database of theorems is filtered for theorems that share some similarity with the conjecture and thus could be relevant for its proof. Here, we need to filter a large database of functions for functions that share some similarity with the input `f`. To do so, a crucial task is to define the notion of *similarity*, which will include features like type classes, types, or other functions occurring in the definition of `f`. This task is open ended; i.e., there will not exist a singe best similarity measure. Instead we will explore various plausible choices, and evaluate and compare them empirically.

Tasks

This project can be subdivided into the following tasks:

1. Make yourself familiar with QuickSpec [1,4,6]. as well as relevance filtering and related techniques [2,5].
2. Design metrics to compare the output of QuickSpec for different signatures (e.g., taking the number of discovered specifications or the "quality" of discovered specification, etc. into account).
3. Design (and implement) various notions of similarity for function taking inspiration from relevance filtering techniques.
4. Implement EasySpec, the simplified interface for QuickSpec, relying on a signature inference algorithm.
5. Evaluate the different signature inference algorithm with respect to the previously defined metrics.
6. **(optional)** Optimize QuickSpec's support for non-equational properties `P :: a -> Bool`, which today can be (inefficiently) encoded as `P x == True`.
7. **(optional)** Extend QuickSpec's support for conditional properties. Automatically select conditions based on conditions in existing specifications.
8. Write the final report and prepare the presentation.

Deliverables

The following deliverables are due at the end of the project:

Final report The final report should consist of an introduction; a theoretical background section; one or more sections describing, the implementation; a detailed empirical evaluation; and a conclusion. The report may be written in English or German.

Haskell code Complete development that runs with the latest GHC release.

Presentation At the end of the project, a presentation of 30 minutes must be given during an InfSec group seminar. It should give an overview and discuss the most important highlights of the work.

References

[1] Quickspec: equational laws for free! `https://github.com/nick8325/quickspec`.

[2] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning*, 57(3):219–244, 2016.

[3] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In Martin Odersky and Philip Wadler, editors, *ICFP 2000*, pages 268–279. ACM, 2000.

[4] Koen Claessen, Nicholas Smallbone, and John Hughes. Quickspec: Guessing formal specifications using testing. In Gordon Fraser and Angelo Gargantini, editors, *TAP 2010*, volume 6143 of *LNCS*, pages 6–21. Springer, 2010.

[5] Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

[6] Nicholas Smallbone, Moa Johansson, Koen Claessen, and Maximilian Algehed. Quick specifications for the busy programmer. Under submission, `http://www.cse.chalmers.se/~nicsma/papers/quickspec2.pdf`.