QuickChecking C code in Haskell

Sebastian Lagerman
seblag@student.chalmers.se

October 2016

1 Introduction

The programming language C has existed since the early 70s [1]. As such there exists a lot of C libraries. For each of those libraries the programmer whom wrote it had to perform some sort of a test. These test are run to establish confidence in the correctness of the software. By just inserting some sort of input and then expecting a certain kind of output is a simple test. Two common ways of testing your C code is either unit testing or property based testing. However the problem with writing unit tests is that it takes a lot of efforts. John Hughes wrote [5] that writing unit tests for n different features will probably result in a test suite of unit tests with three to four test cases per feature. Though this will not find all the bugs, because some only occur when a pair of features interact. Writing a test suite that covers all of this will result in quadratic amount of work and still this will not find all the bugs, because some only occur when three features interact. John Hughes summarized it as: you can never be "done" with testing so don't write tests, generate them. With generated test you get the same amount of certainty for less efforts. We can generate states in property based testing, but to perform a test then we need a property and a validation method. Given a property that holds then we can generate random inputs for that property, also know as property based testing. Currently it is possible to perform property based testing directly in C, however according to Paul Hudak and Mark P. Jones research paper then writing code in Haskell is more expressive and require less lines of code compared to Haskell [4]. Since Haskell is more expressive then C then there will be communication problems between the two languages which can be eased into by creation of boilerplate code for creation of the features. Assuming that a C library that you would like to test is a limited amount of lines of code then it has a limited amount of features. A problem that occurs when you are trying to perform property based testing in Haskell on C code is that Haskell doesn't have stateful testing. This will be one of the bigger problems we will handle during

2 Context

Prior work has already been done towards QuickChecking other languages in Erlang [2]. One of QuviQ services is to property based test code through their Erlang QuickCheck. However, this particular tool utilises Erlang instead of Haskell and isn't an open-source project.

Similarly there exists property based testing libraries in C [6]. This leaves us with a very interesting options, to extend Haskells module QuickCheck to be able to test C libraries. More importantly to generate the boiler plate code.

3 Goals and Challenges

The goal of the project is to create a tool for generating boiler plate code for tests for C libraries. The tool should be easy to use for a developer with knowledge in both C and Haskell development.

The module can be tested on the following open source libraries to begin with to later move on to harder libraries with structs.

- cmath
- string
- time

These were chosen because they are commonly used libraries in C development [1] which also only uses primitive types in C. More libraries may be added if time permits.

Some possible libraries that could be created with structs in them for testing the generation of types, could be:

- compression
- encryption

The module will be tested and verified with QuickCheck though the properties will be found during development.

4 Approach

We will create a structure for generating boiler plate code from C libraris to try and streamline the process of testing libraries. The properties will the user of this module have to write. Once at least one property have been written then the library will be able to be tested. Though if the library contains a struct more work from the user will be needed. That work will be to construct the arbitrary instance of these structs.

Before starting the implementation of the tool, we will need to dig deeper into the inner workings of QuickCheck and the C parsing module inline-c. An advantage of using inline-c would be that the instance Integral is already defined for the standard types in C. Which will make writing the Arbitrary instance for these will be simple. Since QuickCheck is written in Haskell then the properties will be written in the same language. An example of how the generated file could look like based on the library math with only the abs function in it, would be:

```
import Foreign.C.Types
import Test.QuickCheck
import qualified Language.C.Inline as C

C.include "<math.h>"

instance Arbitrary CInt where
  arbitrary = arbitrarySizedIntegral
  shrink    = shrinkIntegral

c_abs :: CInt -> CInt
c_abs i = [C.pure| int{ abs( $(int i) ) } |] :: CInt

-- Write your properties here
-- begin each property function with "prop_"
prop_negateValue :: CInt -> Bool
prop_negateValue a = c_abs a == c_abs (-a)

return []
runTests = $quickCheckAll

main :: IO ()
main = runTests >>= print
```

In order to show the correctness of the tool, we will be using QuickCheck. The output of the generated files will be using the same output printout as QuickCheck. For example:

Successful:

```
Main> quickCheckingC prop_negateValue
OK: passed 100 tests.
```

Bug found:

```
Main> quickCheckingC prop_negateValue
Falsifiable, after 1 tests:
[2]
[-2]
```

5 Time Plan

5.1 Week 13-15

These weeks will be spent researching QuickCheck and inline-c. We will read relevant papers (such as [3] and [5]) and analyze them to create a module for stateful testing in Haskells QuickCheck. This is mainly preparations to learn the tools we need to use.

5.2 Week 15-18

With the information we gathered from the previous weeks then we will create a stateful testing module.

Preparations for the first seminar will also be made (watching the required videos and completing the exercises).

5.3 Week 18-28

After which we will create another module for generating boiler plate code for stateful testing of C code. As well as work on the halftime report.

5.4 Week 28-31

During these weeks, we'll work on finishing the project report as well as the opposition. The main content has been added previous weeks, so this will mainly include formatting, reference additions and other minor things.

5.5 Noteworthy dates:

13 Mars 2017 - Start of project

19 April 2017 - Seminar 1

5 May 2017 - Seminar 2, Presentation

References

[1] C Standard library. `https://en.wikipedia.org/wiki/C_standard_library`, November 2016. Online; accessed 11-November-2016.

[2] Thomas Arts, John Hughes, Ulf Norell, and Hans Svensson. Testing autosar software with quickcheck. Technical report, Quviq and Chalmers University of Technology.

[3] Koen Claessen and John Hughes. Testing monadic code with quickcheck. Technical report, Department of Computer Science, Chalmers University of Technology, 2002.

[4] Paul Hudak and Mark P. Jones. Haskell vs. ada vs. c++ vs. awk vs. ... an experiment in software prototyping productivity. Technical report, Yale University, Department of Computer Science, July 1994. This work was supported by the Advanced Research Project Agency and the Office of Naval Research under ARPA Order 8888, Contract N00014-92-C-0153.

[5] John Hughes. Experiences with quickcheck: Testing the hard stuff and staying sane. Technical report, Chalmers University of Technology and Quviq AB, 2016.

[6] Scott Vokes. theft. `https://github.com/silentbicycle/theft`, October 2016. [Online; accessed 29-December-2016].