QuickChecking C code in Haskell

Sebastian Lagerman
seblag@student.chalmers.se

October 2016

1 Introduction

Testing of your code is important, but people don't want to spend an equal amount of time to write test for the same amount of code. One way to solve this problem is to generate tests.

The goal of this proposal is to build a tool or application that one can use to QuickCheck C code against a Haskell specification. Then use this to test open-source libraries in C, for example for data structures, image manipulation, compression, encryption, etc. This tool will be a good way to streamline the testing to ensure software quality. Knowledge of this will hopefully be useful in code testing to reduce the amount of time spent on writing tests for code. For this Master thesis it will be tested on open source C code.

2 Context

Prior work has already been done towards QuickChecking libraries in Erlang [1]. 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 [**?**]. This leaves us with a very interesting options, to extend Haskells QuickChecking to be able to test C libraries.

3 Goals and Challenges

The goal of the project is to create a tool for generating tests for open source C libraries. The tool should be easy and feasible to use in regular C development.

Another way to solve this problem would be to create and use a property-based library in C, however this has already been done by Scott Vokes [**?**]. To avoid reinventing the wheel and to improve Chalmers own QuickCheck we propose to use Haskell instead. This would make the goal to provide a Haskell module for QuickChecking properties for the following C libraries:

- limits

- cmath
- string
- time
- uchar
- wchar
- wctype

These were chosen because they are commonly used libraries in C development [**?**]. It will be good to use some standard libraries to start with to make sure that the problems that are found in the beginning actually occur inside the testing module that will be created. More libraries may be added if time permits such as a library that contains bugs.

Some possible libraries that could be created with known bugs in them could be:

- data structures
- compression
- encryption

4 Approach

I will create a structure for associating C libraries with their corresponding properties. Then, I will implement the tool which will analyse a library and use QuickCheck to test the properties.

Before starting the implementation of the tool, I will need to dig deeper into the inner workings of QuickCheck and a C parsing tool for example inline-c. An advantage of using inline-c would be that the Constructor Arbitrary is already defined for the standard types in C. Since QuickCheck is written in Haskell the properties will be written in the same language. However a possibility would be to make the properties written in C as well for ease of usage. An example of how properties could look would be:

```
import qualified Language.C.Inline as C

c.include "<math.h>"

prop_c_math :: Int -> Property
prop_c_math i = [C.exp| abs(neg_i)|] == i
    where neg_i = -i
```

In order to show the correctness of the tool's output, I will construct a test suite containing unit tests for each library listed above. The output will be using the same output look as QuickChecking would normally look. For example:

Successful:

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

Bug found:

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

As extra work (given there is enough time) I also want to extend this module and make it modular and not to focus on testing C libraries. If so then similar libraries in other languages could be tested.

References

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

[2] Andrew Thompson. Bullet-proofing helium's embedded c code with erlang quickcheck.