Extracting Properties from Test Cases by Refactoring

Huiqing Li, Simon Thompson
School of Computing
University of Kent, UK
{*H.Li,S.J.Thompson*}*@kent.ac.uk*

Thomas Arts
Chalmers University / Quviq AB
Göteborg, Sweden
thomas.arts@chalmers.se

***Abstract*—A QuickCheck property is a logical statement of aspects of the behaviour of a system. We report on how similar test cases in a test suite written in Erlang can be identified and then refactored into properties, giving a generalisation of the specification implicit in the test suite. Properties give more concise, easier to maintain test suites and better test coverage. A preliminary evaluation of the techniques in industry demonstrates feasibility as well as potential benefits.**

***Keywords*-property-based testing; test cases; refactoring;**

We show how the clone detection functionality and refactorings implemented in the Wrangler refactoring tool for Erlang can be used to identify similar test cases in a test suite, and then to refactor them into QuickCheck properties. Although we describe the process for Erlang / Wrangler / Quviq QuickCheck, the same process is applicable to any programming language, or test language like TTCN-3.

The benefit of this transformation is that the property *generalises* the set of test cases, resulting in higher-level test code that is easier to understand and maintain. Properties can be tested on a wider set of values than the original tests, thus improving test coverage.

In the paper we first present the background to the work, and follow this by describing how properties are extracted, and show how we evaluated the techniques in practice. Finally we draw some conclusions and outline future work.

I. Background

Erlang [1] is a strict, dynamically typed functional programming language with support for concurrency, distribution and fault-tolerance. Wrangler [2] is an interactive tool for refactoring Erlang programs, which not only supports a collection of standard refactorings, but also has facilities to detect and eliminate code clones and to improve the module structure of projects.

There are a number of test frameworks for Erlang, including Common Test [3], EUnit [4], and Quviq QuickCheck [5]. The first two are based on writing test suites, which consist of collections of unit tests. In general, the size of a test suite grows when test cases are added. Test cases tend to have a high rate of code duplication, which is typically the result of a sequence of *copy*, *paste* and *modify* actions (cf. [6]). The clone detection and refactoring capabilities of Wrangler can be used to remove a number of testing 'bad smells' and also to introduce common testing patterns, e.g. Shared Fixture.

QuickCheck is a property-based, random testing tool. QuickCheck testing involves the specification of properties, which are typically universally qualified; it runs automatically generated test cases and examines whether the software satisfies the specified property at these values. Properties are more general and more compact than unit tests. For example, a property of the list reverse function `rev` is that reversing a list of integers twice gives the original list:

```
prop_rev() ->
  ?FORALL(Xs, list(int()), rev(rev(Xs))==Xs).
```

In `prop_rev` the functions `int` and `list` are QuickCheck generators: `int` generates random integers, and `list` generates a list of elements generated by its argument. The macro `?FORALL` binds `Xs` to a generated value. The example property is said to hold if it holds for all the values of `Xs` generated by `list(int())`.

Wrangler's 'similar' code detection is based on the notion of anti-unification [7] to detect code clones in Erlang programs; it also has a mechanism for automatic clone elimination under the user's control. The anti-unifier of two terms denotes their least-general common abstraction, and we say that two expression sequences, A and B, are similar if there exists a 'non-trivial' least-general common abstraction, C, and two substitutions which take C to A and B respectively. By 'non-trivial' we mean that the size of the least-general common abstraction should satisfy some threshold relative to the clone instances. To eliminate a clone, we define a function whose body is the anti-unifier: each instance is given by transforming the substitution into the actual parameter list. More details are reported in [8].

II. From Test Cases to Properties

With Wrangler, the approach to combining test cases into QuickCheck properties follows these steps:

- Apply Wrangler's similar code detection functionality to the test suite. The clone detector reports clone classes, each a set of code fragments in which any two of them are similar to each other. For each clone class, the clone detector also reports the least-general common abstraction of its clone instances, in the format of a new functions.
- Identify a clone class which consists of complete test cases, copy and paste the least-general common abstrac-

tion function of that clone class into the test suite module, then *rename* the function, and apply the *fold expression against function* refactoring to this function. This refactoring replaces instances of a function body by a call to that function: in this case the newly introduced function.

- Apply Wrangler's *test case to property* refactoring to the least-general common abstraction function. This refactoring searches for all the application instances of this function, and collects the actual parameters. The actual parameters collected are then analysed for dependency between parameters, and transformed into QuickCheck data generators; the function itself is wrapped up, if necessary, as an assertion. A QuickCheck property is then generated by combining the data generator and the property using the QuickCheck macro `?FORALL`.

III. A Case Study

We evaluated our techniques, among others, with a test suite from Ericsson. This test suite is written in the Erlang Common Test framework. It has 2228 lines of code, containing 4 groups of test cases, and 30 test cases in total. Applying Wrangler's clone detection to this module, it reports that 15 out of the 30 test cases are clones. Apart from those clones that consists of complete test cases, there are also clones that only cover part of a test case. As a concrete example, the clone report says that there are 3 test cases, named `create_2`, `create_3` and `create_4` respectively, are clones to each other, and their least-general common abstraction generated by Wrangler, after variable and function renaming, is:

```
create_234(Media, StreamType, Codec) ->
     ... some code elided here ...
  SidLc = {mux_id_1, Media},
  CreateData =
    #brchMuxLcAccess{sid = SidLc,
                     stream_type = StreamType,
                     local_data = LocalData,
                     codec = Codec,
                     event_module = iptermCb},
    ... some code elided here ...
  ?RESULT("DONE",[]).
```

where the instances of the parameters are emboldened.

The function `create_234_gen` is the test data generator extracted from the calls to function `create_234` after applying the *folding* refactoring to it:

```
create_234_gen() -> oneof([
  {audio_id_1,?BRCH_AUDIO,
   {?AMR,{?R_122,?BRCH_DISABLED,?BRCH_DISABLED,
          ?BRCH_BIT},33,44,40}},
  {audio_id_1,?BRCH_AUDIO,
   {?G723_1,{?R_53,?BRCH_DISABLED},33,44,40}},
  {video_id_1,?BRCH_VIDEO,
   {?H264,?BRCH_NO_OPTION,33,44,40}}]).
```

In this example `oneof` is the QuickCheck generator which randomly chooses an element from a list of generators, which can be constants as here. Wrangler is also able to generate data generators that take into account the dependency between the parameters; this aspect of the work is not covered in this paper due to space constraints.

In this case the property generated is:

```
create_234_prop() ->
  ?FORALL({Media, StreamType, Codec},
    create_234_gen(),
    create_234(Media, StreamType, Codec)).
```

Through this transformation, we are able to combine 3 test cases into one property, and the user can now extend the generator to test more cases easily.

As a final step we are able in some cases to generalise the generator, so that a finite choice is generalised to an infinite one, as we might generalise `oneof` applied to a finite set of integer lists to the generator `list(int())`.

IV. Conclusions and Future work

We implemented this for Erlang and related systems, but the approach we have outlined is generic, and can equally well be applied to other languages, such as C and TTCN-3.

We aim to carry out further case studies to evaluate and to study the practical applicability of the approach. The current implementation reports cloned test cases, but the user needs to apply refactorings to generate QuickCheck properties and then examine them. We intend to extend Wrangler so that it can generate the QuickCheck properties and generalisations automatically after a clone detection. The user then only needs to examine the report, and accept or reject the suggestions.

This research is supported by EU FP7 project ProTest, grant number 215868, http://www.protest-project.eu/.

References

[1] F. Cesarini and S. Thompson, *Erlang Programming*. O'Reilly Media, Inc., 2009.

[2] H. Li, S. Thompson *et al.*, "Refactoring with Wrangler, updated," in *ACM SIGPLAN Erlang Workshop 2008*, 2008, available from www.cs.kent.ac.uk/projects/wrangler/.

[3] http://www.erlang.org/doc/apps/common_test/index.html.

[4] Mickal Rémond, Richard Carlsson, "EUnit," http://svn.process-one.net/contribs/trunk/eunit.

[5] T. Arts *et al.*, "Testing Telecoms Software with Quviq QuickCheck," in *ACM SIGPLAN Erlang Workshop 2006*, 2006.

[6] H. Li *et al.*, "Improving your Test Code with Wrangler," School of Computing, Univ. of Kent, Tech. Rep. 4-09, 2009.

[7] G. D. Plotkin, "A Note on Inductive Generalization," *Machine Intelligence*, vol. 5, 1970.

[8] H. Li and S. Thompson, "Similar Code Detection and Elimination for Erlang Programs," in *PADL 2010*, 2010.