Automated Test Generation for Transformations using Symbolic Execution*

[Extended Abstract]

Ahmad Salim Al-Sibahi[1]

IT University of Copenhagen, Copenhagen, Denmark
asal@itu.dk

1 Introduction

Transformations appear in most modern IT systems today, from system integration and adaptation, model translations in model-driven development to code optimization, refactoring and code generation. For example, consider the *Rename-Field* [3] refactoring presented in Figure 1, which aims to not only rename a field but also correctly update all references to it.

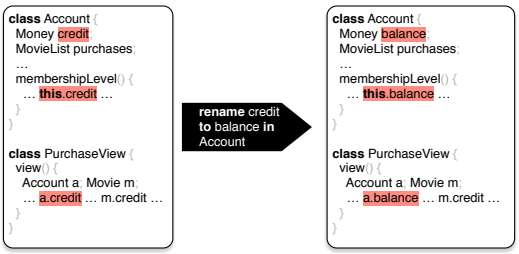


Figure 1: The Rename-Field refactoring: rename the definition of credit to balance and update all references accordingly.

Testing such transformations is hard and few solutions exist for generating good tests automatically [4]. Existing test generators [9, 7] do not support high-level transformations, and existing solutions rely on black-box generators [1] disregarding the code under test.

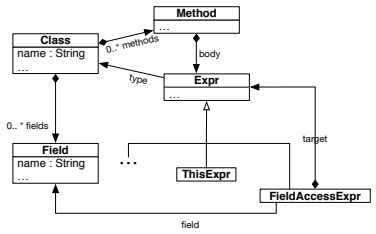
2 Method

Our primary goal is to develop a technique that enables effectively generating tests for transformations. In order to do so we:

- Develop a small formal transformation language, in the style of `IMP`, called `TRON` which supports high-level constructs such as deep matching on structures, first-class set operations and ownership links. An simplified version of the *Rename-Field* refactoring in `TRON` is presented in Figure 2; the deep matching `foreach`-loop on line 6 gets all object references of type FieldAccessExpr in the input class.
- Significantly extend existing symbolic execution techniques [2] to handle transformations written in `TRON`, which is used to build a white-box test generation tool.

*Supported by *The Danish Council for Independent Research* (grant no. 0602-02327B) under the Sapere Aude scheme, project VARIETE.

- Build a white-box test generator which is based on the develop symbolic execution technique.



(a) Abstract syntax for simple object-oriented programs

```
1 input:  class: Class, old_field: Field, new_field: Field
2 precondition: old_field ∈ class.fields ∧ new_field ∉ class.fields
3
4 // the refactoring program
5 class.fields := (class.fields \ old_field) ∪ new_field
6 foreach faexpr ∈ class match* FieldAccessExpr do
7   if faexpr.field = old_field ∧
8       faexpr.target.type = class then
9         faexpr.field := new_field
10  else skip
```

(b) A Refactoring in TRON

Figure 2: A simplified version of the rename-field *refactoring example in TRON*

3 Evaluation Results

We implemented the white-box test generator in Scala, and ran it on a series of TRON programs consisting of toy transformation and four simplified refactorings: *Rename-Field*, *Rename-Method*, *Replace-Delegation-with-Inheritance* and *Extract-Superclass*. On the toy transformation, our white-box test generation tool achieved 100% branch coverage, beating the black-box test generation tool that we used as baseline (having coverage 20%-66.$\overline{6}$%, except for one where it did achieve full coverage). Our white-box test generation tool did well on the refactorings as well, beating the baseline black-box test generation tool each time and achieving full coverage in two out of the four refactorings. Performance-wise the white-box test generation tool was between 2.2×–31.2× slower than the black-box test generation tool, which we believe is reasonable due to the more work required but still leaves some room for optimisations in future work.

4 Beyond TRON

Like IMP, TRON is primarily used as a vehicle for the development of formal techniques and is not meant to be used to write realistic programs. In the future, we hope to use our experiences and developed techniques to work with a more realistic high-level transformation language or framework with similar features, for example ATL [5], Uniplate [6] or Kiama [8].

References

[1] Brett Daniel, Danny Dig, Kely Garcia, and Darko Marinov. Automated testing of refactoring engines. In *FSE 2007, Dubrovnik, Croatia, September 3-7, 2007*, pages 185–194, 2007.

[2] Xianghua Deng, Jooyong Lee, and Robby. Efficient and formal generalized symbolic execution. *Autom. Softw. Eng.*, 19(3):233–301, 2012.

[3] Martin Fowler. *Refactoring - Improving the Design of Existing Code.* Addison Wesley object technology series. Addison-Wesley, 1999.

[4] Milos Gligoric, Farnaz Behrang, Yilong Li, Jeffrey Overbey, Munawar Hafiz, and Darko Marinov. Systematic testing of refactoring engines on real software projects. In *ECOOP 2013, France, July 1-5, 2013*, pages 629–653, 2013.

[5] Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In *Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica, October 2-7, 2005*, pages 128–138, 2005.

[6] Neil Mitchell and Colin Runciman. Uniform boilerplate and list processing. In *Haskell 2007, Freiburg, Germany, September 30, 2007*, pages 49–60, 2007.

[7] Adrián Riesco. Test-case generation for maude functional modules. In *WADT 2010, Etelsen, Germany, July 1-4, 2010*, pages 287–301, 2010.

[8] AnthonyM. Sloane. Lightweight language processing in kiama. In JoÃčoM. Fernandes, Ralf LÃďmmel, Joost Visser, and JoÃčo Saraiva, editors, *Generative and Transformational Techniques in Software Engineering III*, volume 6491 of *Lecture Notes in Computer Science*, pages 408–425. Springer Berlin Heidelberg, 2011.

[9] Willem Visser, Corina S. Pasareanu, and Sarfraz Khurshid. Test input generation with java pathfinder. In *ISSTA 2004, Boston, Massachusetts, USA, July 11-14, 2004*, pages 97–107, 2004.