JUnitMe2.0 : JUnit tests generation with Alloy and CodeModel

Auteurs :
Salla DIAGNE
Anis TELLO

16th December 2015

Table of contents

Introduction

"Every program is guilty until proven innocent"

During the development phase, developers spend lots of time to write tests corresponding to code they have written in the program.

A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage.[1]

But what if developers didn't have to spend hours writing tests to have a good code coverage? What if there was a magic stick that can generate a bunch of unit tests?

Since Java is one of the most used programming languages in the world, we have decided to find a solution that would generate automatically Java unit tests. That would save time for developers, and would give the time and the energy to focus on working on business layer.

This project is an extended version of an application developed by Valentin Lefils and Quentin Marrecau [2] [3] . The first version of the application has treated the same problems we are facing, but only on small example of Java programs.

In this project, we present our tool: JUnitMe2.0. Our tool can generate automatically Java unit tests for any Java open source application. From a description of specification, our tool generate all instances that covere the data specifications, then generate the unit tests corresponding to these instances.

The rest of this report is organized as follows : Section 1.1 provides motivation and the goals of this work. Section 1.2 describes an overview of our tool: JUnitMe2.0. Section 1.3 describes the algorithms. Section 1.4 explains the implementation and the architecture, section 1.5 provides an example of a use case and expected results. Section 2.1 evaluates our tool from a complexity point of view. Section 2.2 evaluates our tool from a performance point of view. Section 2.3 evaluates the ease of use of our tool. Finally, Section 3. concludes this report.

Chapter 1

Technical work

1.1 Goal

The goal of this tool is to generate a large amount of Java unit tests automatically so that we can achieve a good coverage of code with no time or effort spent. So developers can focus on developing the business layer without worrying about testing the code they have written. Our tool can be used to verify that no bugs will occur when objects of the program interact with each other. In other words, our tool can verify that there is no actual error exists for an application.

1.2 Overview

1.3 Algorithm

1.3.1 Alloy to Java

1.3.2 Java to Alloy

1.4 Implementation & architecture

1.4.1 Technologies

In order to realise our project we have used four different techonogies: Spoon, Alloy, Alloy Analyzer and CodeModel.

Spoon

In order to analyze and transform source code, we needed an efficient and powerful library. We have chosen Spoon, a high-quality open-source library created and maintained by INRIA (French Institute for Research in Computer Science and Automation (French: Institut national de recherche en informatique et en automatique)). Providing a complete and fine-grained JAVA metamodel, Spoon enables us to perform effortless treatments in differents parts of code. These treatments are performed by processors, which are able to browse, modify, or even add any program element (class, method, field, statements, expressions...).[4][5] Spoon can also be used on validation purpose, to ensure that your programs respect some programming conventions or guidelines, or for program transformation, by using a pure-JAVA template engine.[6]

Alloy

Alloy is a language for describing structures and a tool for exploring them. It has been used in a wide range of applications from finding holes in security mechanisms to designing telephone switching networks. An Alloy model is a collection of constraints that describes (implicitly) a set of structures, for example: all the possible security configurations of a web application, or all the possible topologies of a switching network.[7]

Alloy Analyzer

Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model by generating sample structures, and to check properties of the model by generating counterexamples.[7]

CodeModel

CodeModel is a Java library for code generators; it provides a way to generate Java programs in a way much nicer than PrintStream.println(). With CodeModel, we can build the java source code by first building AST, then writing it out as text files that is Java source files.[8]

1.4.2 Implementation

The main parts of our project is generating Alloy source code from a given Java program, then generating Alloy instances. Thereafter modelizing these instances in Java to, finally, convert these modelized instances to Java unit tests.

Generating Alloy source code

Base model The base model is a meta-model, It is the base of the generated java to Alloy program. It is composed of three parts: Types, methods, methods constraints and arguments of these methods.

Types

```
abstract sig Type{
}

sig Object{
    type : Type
}
```

Methods

```
abstract sig CallWithNext extends Call{
    nextMethod : Call
}

sig MethodCall extends CallWithNext{
    receiver : Object,
    method : Method,
    params : seq Object
}

one sig Begin extends CallWithNext{
}

abstract sig Method {
    paramTypes : seq Type,
    receiverType : Type
}
```

Methods constraints

```
------ Method Constraints-------


-- There must be one and only one method.next= end
fact{
    one m : MethodCall • m.nextMethod in End
}

-- Method call cant link to itself
fact{
    all mc : MethodCall • mc.nextMethod≠mc
}

-- The receiver has the right type
fact{
    all mc : MethodCall •  mc.receiver.type=mc.method.receiverType
}
```

```
-- A call can't be done twice
fact{
all mc: MethodCall • no mc2 : MethodCall •  (mc2 in mc.^nextMethod) ∧ (mc2.nextMethod=mc)
}

-- All methodCall Has been called
fact{
    all mc : MethodCall • one c : CallWithNext • c.nextMethod=mc
}

-- Object calling a method has the right type
fact{
    all mc:MethodCall • mc.receiver.type=mc.method.receiverType
}
```

Methods arguments

```
------- Param Constraints ------


-- Types verification (method declaration/method calling)
fact{
    all mc : MethodCall •validParam[mc.method,mc]
}

pred validParam[method : Method, call : MethodCall]{
        call.params.type=method.paramTypes
        #call.params=#method.paramTypes
     all  pt : method.paramTypes.elems • all p : call.params.elems • call.params.idxOf[p]=
         method.paramTypes.idxOf[pt] implies pt=p.type
}



-------- Primitive Types ------------

one sig Gen_Double extends Type{}
one sig Gen_Integer extends Type{}
one sig Gen_Float extends Type{}
one sig Gen_Boolean extends Type{}
one sig Gen_Byte extends Type{}
one sig Gen_Character extends Type{}
one sig Gen_Long extends Type{}
one sig Gen_Short extends Type{}
```

Code generation To generate Alloy instances for an existing Java program we used Spoon. Spoon uses AST (Abstract syntax tree)[9] to browse the structure of a specified program. With informations collected in the AST, we are able to generate the code of Alloy program corresponding to the initial Java program. The generated code is then grafted to the base

model which allows us to ******.

Generating Java unit tests

After having Alloy source code for a given Java program,*****

1.5 Utilisation

Chapter 2

Evaluation

2.1 Complexity

2.2 Performance

2.3 Ease of use

Conclusion

using **Spoon** Java library to analyze and trasform source code, **Alloy** a language and tool for relational models, **Alloy Analyzer** a solver that takes the constraints of a model and finds structures that satisfy them and **CodeModel** a Java library for code generators.

Bibliography

[1] Wikipedia. Code coverage. https://en.wikipedia.org/wiki/Code_coverage.

[2] Valentin Lefils and Quentin Marrecau. Génération de tests junit avec alloy. http://static.monperrus.net/iagl/2014/rendu3-alloy-test-data-generation/01_Junit_Generation_Marrecau_Lefils/Rapport.pdf, 2015.

[3] Valentin Lefils and Quentin Marrecau. Junitme (github). https://github.com/user4me/JunitMe.

[4] INRIA. Spoon - source code analysis and transformation for java.

[5] Martin Monperrus, Renaud Pawlak, Nicolas Petitprez, Carlos Noguera, and Lionel Seinturier. Spoon: A library for implementing analyses and transformations of java source code.

[6] Spoon - javasource.

[7] MIT. Alloy: Official website.

[8] SUN. Codemodel.

[9] Wikipedia. Abstract syntax tree. https://en.wikipedia.org/wiki/Abstract_syntax_tree.