Code-Checker using ATP

Niharika J, Lipisha C, Priti G
Guide: Kavita Wale

Fr. C. Rodrigues Institute of Technology, Vashi

August 9, 2015

1 Introduction

1.1 Software Verification:

Software verification is a discipline of software engineering whose goal is to assure that software fully satisfies all the expected requirements. In the context of hardware and software systems, software verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal software specification or property, using rules. Software verification can be helpful in proving the correctness of systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code. The verification of these systems is done by providing a formal proof on an abstract model of the system, the correspondence between the mathematical model and the nature of the system being otherwise known by construction. Verifying the correctness of a program involves formulating a property to be verified using a suitable logic such as first order logic or temporal logic

There are two fundamental approaches to Software verification:

1. ***Dynamic Program Analysis***, also known as Test or Experimentation - This is good for finding bugs.
2. ***Static Program Analysis***, also known as Analysis - This is useful for proving correctness of a program although it may result in false positives.

In this system we will be implementing the software using static program analysis.

- **Static Program Analysis:**

Static verification is the process of checking that software meets requirements by inspecting the code before it runs. The analysis verification method applies to verification by investigation, mathematical calculations, logical evaluation, and calculations using classical textbook methods or accepted general use computer methods. Analysis includes sampling and correlating measured data and observed test results with calculated expected values to establish conformance with requirements.

Some of the implementation techniques of formal static analysis include:

1. ***Model checking***, considers systems that have finite state or may be reduced to finite state by abstraction.
2. ***Data-flow analysis***, a lattice-based technique for gathering information about the possible set of values.
3. ***Abstract interpretation***, to model the effect that every statement has on the state of an abstract machine (i.e., it 'executes' the software based on the mathematical properties of each statement and declaration). This

abstract machine over-approximates the behaviours of the system: the abstract system is thus made simpler to analyse, at the expense of incompleteness (not every property true of the original system is true of the abstract system). If properly done, though, abstract interpretation is sound (every property true of the abstract system can be mapped to a true property of the original system). The Frama-c value analysis plugin and Polyspace heavily rely on abstract interpretation.

4. ***Hoare logic***, a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs. There is tool support for some programming languages (e.g., the SPARK programming language (a subset of Ada) and the Java Modelling Language — JML — using ESC/Java and ESC/Java2, Frama-c WP (weakest precondition) plugin for the C language extended with ACSL (ANSI/ISO C Specification Language)).

5. ***Symbolic execution***, as used to derive mathematical expressions representing the value of mutated variables at particular points in the code.

Examples of Static Program Verification: * Code conventions verification

- Bad practices (anti-pattern) detection
- Software metrics calculation
- Formal verification

1.2 Implementation of Code Checker and Deliverables:

Programmers while coding are unsure whether their code will run efficiently, given any number of test cases (input). Software verification helps these programmers to check the correctness of their code and help them improve it. In this system the user (programmer) will submit their code to be verified as an input to the system. This input source code is assumed to be syntactically correct. Hence, it will be checking for logical errors of the source code given as the input.

The Code Checker will be implemented in Python as the core language. The input i.e. the source code provided by the user will be in C language.

Once the source code is submitted as the input, the Code Checker will check whether the code is logically correct or not. If the code is logically sound (correct) the system will notify the user with a "No error" (or a successful message) or with an error message that the logic of his/her code is not correct corresponding to the test cases applied to the code for checking the correctness.