Introduction to Formal Methods **Fall 1397**

Assignment 2: Automatic Program Verifier

Hossein Hojjat & Fatemeh Ghassemi *Mehr 29*

1 Introduction

The goal of this assignment is to write a simple verification engine to verify programs in a simple imperative language. For implementing the engine in this project you are free to use any programming language that you are comfortable with. The input to your engine is textual and the output is printed on the screen (standard output).

2 Language

The small language of this assignment has the following expressions and statements. The type of identifiers in this language is (unbounded) integers. There is no need to type check a given program prior to verification, we assume that if the syntax of a program matches the given BNF then the program is valid. Due to simplicity of the language, no prior specific compiler construction knowledge is required for this assignment, you can use any simple syntax analysis technique to recognize the program structure. To make your language more interesting, you can add more constructs to the language (such as `assume` statement or modular expressions).

Program	::=	Stmt ; <EOF>
Stmt	::=	`skip`
		Id `:=` IExp
		Stmt `;` Stmt
		`if` (BExp) Stmt `else` Stmt `endif`
		`while` (BExp) Stmt `endwhile`
		`assert` (BExp)
BExp	::=	IExp <= IExp
		IExp == IExp
		`not` BExp
		BExp `and` BExp
		BExp `or` BExp
		(BExp)
IExp	::=	*Int*
		Id
		IExp `+` IExp
		IExp `-` IExp
		(IExp)

- *Id* represents a sequence of letters, digits and underscores, starting with a letter and which is not a keyword. Identifiers are case-sensitive.
- *Int* the domain of (unbounded) integer numbers, with usual operations on them.
- <EOF> represents the special end-of-file character.

3 Interface

Your program will be called from the command-line using the following format (`verif` is the name of your tool):

```
verif [options] <sourcefile>.imp
```

Try to avoid using any other format since it will make grading the of your project difficult (and you may lose points if you do not follow this format). We describe the possible options to your tool in each task.

4 Task 1

Generate the control-flow-graph for the input program, as discussed in Lecture 9. For better testing the generated CFG, plot it using the *dot* program. *dot* is a very simple program for drawing directed graphs (`http://www.graphviz.org/`). If your program is called with the `-c` option, it will generate a `png` file that shows the CFG of the input `imp` program. Test your program with different benchmarks, it is important to ensure about the correctness of CFG generation before trying to verify programs.

5 Task 2

Generate the set of Horn clauses representing the input program. If your verification engine is called with the `-h` option, it will print out the generated Horn clauses on the screen using the SMT-LIB format. Try a set of examples and inspect your Horn clauses to see if they are correct.

6 Task 3

When the user calls your program with the `-v` argument, it will verify the input file by checking the satisfiability of the generated Horn clauses. For satisfiability you need to use an off-the-shelf Horn solver (e.g. Z3, Eldarica, Spacer or Hoice). Your program will either print "correct" for a program with no bug. Or, it will print "incorrect" for a program that has a bug. Note that if the program has no `assert` at all, we assume that there is nothing to be checked for it and the verifier returns correct.

Bonus Print out the values under which the program will hit the error.

Note: You can either install the theorem prover on your machine and your code executes the theorem prover as an external application (i.e. z3 can be called from the command line), or you can add the required libraries to call the theorem prover from its libraries.

7 Deadline and Deliverables

The deadline of this project is Aban 11th at 11:59pm. Make a zip from your files to make uploading the files easier. Since you are free in choosing your programming language, it is important that you make it easy to build your project. Please include a Readme file in your submission to note how the user can build and run your program.

Late submissions will get penalty for each day that the submission is late.