

Project 4

CMSC 838Z, Spring 2004

Due Wednesday, March 31 (e-mail to me)

This assignment considers *program verification*, which is the process of proving properties about the executions of programs. We will do some written exercises using Hoare Logic, a logic about program executions, and also make use of the Simplify automatic theorem prover.

For this assignment, you will get experience using the (*aka rccjava*), a typechecker that checks for race conditions in Java programs.

Written Exercises

Please do the following exercises from your handout material:

- Exercises 4.3, number 2 (page 242).
- Exercises 4.5, number 2b (page 245). Show your program and the proof that it satisfies the given post condition using a proof tableaux, as shown in the examples prior to this exercise.
- Exercises 4.6, number 1 (page 250).

For a primer, try the practice exercises at <http://www.cis.ksu.edu/~huth/lics/tutor/chap4/questions.html>.

Proving Properties about Programs

Now for the fun part. We will write our own little program verifier. Given a program Π , you will prove that executing a program in a state that satisfies precondition Π_{pre} will result in a state that satisfies postcondition Π_{post} . This will occur in two steps:

- Starting from the postcondition Π_{post} , perform verification condition generation on Π , resulting in a verification condition $VC(\Pi)$. You will write your own verification condition generator following the algorithm we went over in class, and which is described in Necula's notes (see <http://www.cs.berkeley.edu/~necula/autded/lecture03.pdf>).

- Prove that Π_{pre} implies $VC(\Pi)$. To do this, we will simply invoke the SIMPLIFY theorem prover.

We will define programs as having the following grammar:

$$\begin{aligned}
 E & ::= n \mid x \mid (+ E E) \mid (- E E) \mid (* E E) \\
 B & ::= \text{TRUE} \mid \text{FALSE} \mid (\text{NOT } B) \mid (\text{AND } B B) \mid (\text{OR } B B) \mid (\text{EQ } E E) \mid (< E E) \\
 C & ::= (\text{SET! } x E) \mid (\text{SEQ } C C) \mid (\text{IF } B C C) \mid (\text{WHILE } P B C)
 \end{aligned}$$

All program forms are s-expressions, as in Lisp or Scheme. This will simplify the process of feeding assertions to SIMPLIFY for proof, which expects s-expressions in the form above, and will also simplify the process of writing a parser for your language. All WHILE expressions are annotated with an assertion P , which serves as the loop invariant. Our assertion language P essentially extends boolean expressions with implication and universal quantification:

$$\begin{aligned}
 P & ::= (\text{FORALL } (V) P) \mid (\text{IMPLIES } P P) \mid \text{TRUE} \mid \text{FALSE} \mid (\text{NOT } P) \\
 & \quad \mid (\text{AND } P P) \mid (\text{OR } P P) \mid (\text{EQ } E E) \mid (< E E) \\
 V & ::= x \mid Vx
 \end{aligned}$$

Your prover will read in programs which start with a precondition, then have a command, and conclude with a postcondition. As an example, the factorial program on page 249 of the handout would be written as follows:

```

(TRUE)                                ; precondition
(SEQ (SET! y 1)
 (SEQ (SET! z 0)
 (SEQ (WHILE
      (EQ Y (FACT Z))                ; loop invariant
      (NOT (EQ Z X))                 ; loop guard
      (SEQ (SET! Z (+ Z 1)) ; loop body
            (SET! Y (* Y Z)))))))
(EQ Y (FACT X))                       ; postcondition

```

The text following ; are comments. Note that the loop invariant and postcondition here is not actually expressible in our assertion language P , since we don't have a predicate FACT. We'll have to relegate ourselves to simpler assertions.

Your program will read in a program Π like the above, generate a verification condition $VC(\Pi)$ for it, and then feed the compound assertion to simplify that suggests the precondition implies the verification condition. SIMPLIFY should return either valid, or invalid. You can ignore the error context. Your program should print out the assertion you gave to simplify, and whether SIMPLIFY thought it was valid or not.

To test your solution, try out your answers to the exercises above, and see what verifications get generated and whether SIMPLIFY can prove them correct. Submit for me your working prover, and its output for these exercises.

Notes

Information on SIMPLIFY, particularly its input syntax, can be found at <http://research.compaq.com/SRC/esc/Simplify.html>. SIMPLIFY is installed in `/fs/unsupported/Simplify-1.5.4` on junkfood; the `bin` directory contains binaries for Solaris, Linux, Windows, and Mac OSX. Let me know if you have any trouble using them.

To reduce your workload, feel free to work together to write a parser and pretty printer for the language. Indeed, if you feel particularly generous, you can post your parser and pretty printer to the Wiki (or at least a URL for it). Make sure you work individually on the verification condition generation and proving parts.