Programming Languages

Qualifying Exam
Fall 2014

April 30, 2014

NOTE: this exam is open book and open notes. You are not allowed to use calculator or any sort of electronic devices.

Question 1 [40 Points]

Consider the following syntax for an imperative language:

```
<program>     ::= <block>
<block>       ::= { <declarations> <lstatement> }
<declaration> ::= epsilon
              |   <procedure> <declaration>
<procedure>   ::= proc <Id> <block>
<lstatement>  ::= <number>: <statement>
              |   <lstatement> ; <lstatement>
<statement>   ::= <Id> = <expression>
              |   <Id> : <statement>
              |   goto <Id>
              |   call <Id>
              |   <block>
              |   if <expression> goto <Id>
<expression>  ::= <number>
              |   <Id>
              |   <expression> + <expression>
```

The language is a simple imperative language that allows nested definitions of procedures and `goto` statements. Note that each statement in a block has a label attached (used as target of `gotos`). You can assume that the programs you are dealing with have statements in each block labeled by distinct contiguous numbers.

Provide the denotational semantics of this language; you should make the following assumptions and you should meet the following requirements:

- the language can use only the single data type Nat
- the semantics should realize static scoping
- non-local `gotos` (i.e., jumps to labels not present in the current block) result in erroneous executions

Note: If it helps, you can also assume that the labels in each block are $1, 2, \ldots$.

Question 2 [20 Points]

Answer the following questions concerning the language described in Question 1:

(5 Points) Since the language does not include variable declarations, why do we need to worry about static vs. dynamic scoping?

(15 Points) Let us consider a variant of the language where `goto` statements are allowed to jump outside of the current procedure. Describe the implications of this possibility and what kind of changes are needed at the implementation level to support it.

Question 3 [40 Points]

Answer the following questions:

(15 Points) Extend the Hoare's set of axioms to introduce an axiom for the `repeat-until` loop (with the same meaning as in Pascal).

(25 Points) Write a simple annotated program (i.e., a program that includes preconditions, postconditions, and loop invariants) that computes $\lfloor \sqrt[3]{x} \rfloor$ for an input x; the program should use only basic arithmetic operations (addition, subtractions, product) and only `repeat-until` as a looping

construct. Your program should have the following precondition (p) and postcondition (q):

$$\begin{array}{ll} p: & x \geq 1 \\ q: & out = \lfloor \sqrt[3]{x} \rfloor \end{array}$$

Note: you should translate the $\lfloor \cdot \rfloor$ in a mathematical definition (e.g., that uses arithmetic operations and comparisons like $=, \leq, \geq, \ldots$).

Sketch the use of the Hoare axiom introduced in the first question to prove the correctness of the loop part.