Theory of Programming Languages

Phd. Qualifying Exam (July 1, 2016)

Answer two questions out of four. Only top two scores will be considered.

Problem 1. Static scoping and dynamic scoping:

This question concerns programs written in a simple language L, a variant of Pascal(or C). A program in L consists of a main program containing a body, some variable declarations and a collection of procedure definitions, each of which can introduce local variables. Blocks can introduce local variables as you know. Nested procedures are not permitted in L, there are no procedure valued parameters, and the languages is not strongly typed. Recursion among the procedures is permitted.

Let P be a program in language L. Define $M_{stat}[P]$ to be the result (for example, the final values of variables) of execution of P using static scope rules (also known as lexical scoping,) and $M_{dyn}[P]$ to be the result of execution of P using dynamic scope rules.

(a) Explain static scope and dynamic scope.

(b) Give a program P for which $M_{stat}[P] \neq M_{dyn}[P]$, i.e. which gives different results under different scope rules.

A preprocessor G for L translates each program P in L to a (possibly different) program $G(P)$ in L. For our purposes, a preprocessor can make at most a constant number of passes over the input program.

(c) Describe a preprocessor G that for all programs P in L satisfies

$M_{dyn}[G(P)] = M_{stat}[P]$.

Give an example with the program in (b).

State your argument why your procedure will work.

(d) Is it necessary to modify the preprocessor G if nested procedures **are permitted** in L? Why or why not?

Problem 2

Different programming languages provide different degrees of block structure, in which the scope of an identifier can be limited to a portion of the program text. Blocks are here defined always to be properly nested, i.e. either one block is completely contained within another or the two blocks are disjoint.

For example, Pascal allows one procedure definition to be nested within another; it also allows one **record** to be nested within another. C allows nesting of a record(**struct**), but not of a procedure. Fortran provides disjoint blocks (e.g. **common** and **subroutine**), but no nesting.

Consider a language with block structure, and a compiler for the language. For each of the five following assertions state to what extent you agree or disagree with it, and why. Where appropriate, be careful to consider both lifetime (temporal duration) and scope (spatial extent of visibility).

a. Block structure makes it easier to implement a variable that continues to live, keeping its value, between invocations of the procedure in which it is declared. (An example application is the seed for a pseudo-random number generator. Example constructs are **own** variables in Algol which introduced the concept and **static** variables in PL/I and C.)

b. Block structure makes a program harder to understand, because it permits identifiers to be reused for different purposes.

c. Block structure permits modularization, hence faster recompilation and easier error identification.

d. Consider a **goto** to a label in an enclosing procedure block. The code compiled for the **goto** is not appreciably more complex than the code compiled for a **goto** to a label in the same block.

e. Block structure reduces the main storage space requirement at both compilation time and execution time, permitting the use to run with a smaller machine.

Problem 3

Consider the commands from Simple Imperative Language defined by the following abstract syntax

e ::= $n \mid x \mid e + e$

b ::= $e \leq e$

C ::= x **:=** $e \mid c_0$**;** $c_1 \mid$ **if** b **then** c_0 **else** $c_1 \mid$ **while** b $c \mid$ **skip**

where "+" and "$\leq$" are addition and less-than-equal, respectively. The semantics is as usual as other imperative languages such as C. Note that "x" is a variable and "**:=**" is assignment operator.

(1) Define denotational semantics for the expression e and commands C precisely. Don't forget to define domains of semantic functions and values.

(2) Let the state σ = { *x* -> 3, *y* -> 5}. Derive the meaning for

$$[\![\textbf{if } x \leq 0 \textbf{ then } x := x + 1 \textbf{ else } y := x]\!]\ \sigma$$

(3) Explain the meaning of "compositional" semantic definitions using an example with the language above.

(4) Let the state σ = { *x* -> 3, *y* -> 5}. Derive the meaning for

$$[\![\text{while } x \leq 5 \quad x := x + 1]\!]\ \sigma$$

Problem 4

Consider a language that is familiar with you such as C or Java. Do not consider dynamic languages such as Javascript or old LISP. We are going to discuss the "role of type system" of a static-typed language such as C.

(1) Describe your understanding about "What is type system?" and "What is the role of type system?"

(2) What can "a type system" guarantee for programmers? For example, when the type system of your compiler (say C compiler) gives no warning message during compile time, what will be guaranteed during the runtime of compiled binary executables?

(3) State your understanding for the term " soundness and completeness of a type system." With your favorate language (say C), state whether the type system of your language is sound. If not, give an example.