CS520 Theory of Programming Languages

(Software Area)

Phd. Qualifying Exam (January 13, 2017)

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

Problem 1. Language run-time structure:

Consider a language that has:

- dynamic scoping
- third class procedures and functions (i.e., procedures and functions that can be declared and called, but cannot be passed as arguments or assigned as variable values);
- dynamic types such as pointers and extensible arrays;
- no recursion.

Describe a runtime structure that will maintain correct execution for this language. Make it as simple as possible. Include a description of how memory is laid out by the compiler, and distinguish this from how memory is manipulated at run-time. Explain how correct values are found at run-time for the variable references in the code. Use diagrams to make your answer as clear as possible.

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 expressions defined by the following abstract syntax

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

where "+" and "×" are addition and multiplication, respectively. Note that "x" is a variable.

(1) Define **denotational semantics** or **natural semantics** for e precisely. Don't forget to define domains of your functions and values.

(2) We will implement this language E with <S,E,C>-machine. The <S,E,C>-machine is an abstract machine. S is a stack of values (ordered sequence). E is an environment (function : Variable → Value). C is a command sequence defined as following:

C	→	add. C	// add top two values and push the result back
	|	mul. C	// multiply top two values and push the result back
	|	push(x). C	// push the value of the variable x
	|	push(n). C	// push the constant n
	|	ε	empty command

Define the **transition semantics** for command C.

(3) Define the compilation rules from e programs to C command sequences.

Problem 4

Consider a language with expressions defined by the following abstract syntax

$e ::= n \mid x \mid e + e \mid e \times e \mid \text{let } x = e \text{ in } e$

where "+" and "×" are addition and multiplication, respectively. Note that "x" is a variable.

Design your type system for e precisely. What is the goal of your type system? That is, what can your type system guarantee for programmers?