Computer Science Department
Stanford University
Comprehensive Examination in Compilers
Autumn 1994

October 20, 1994

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 2 pages.

3. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

CS Comprehensive Exam: Compilers (30 points)

1. (7 points)

   Comments in the C programming language are introduced by the characters `/*` and terminated by the characters `*/`. They do not nest and they do not occur within string or character literals. For each of the following regular expressions, indicate whether or not the expression describes precisely the set of all valid C comments. For those that do not, provide a counterexample. Assume for convenience that the character set is {`*`,`/`,`a`}.

   (a) `/*(/|a|*)*/` with the inner group starred: $/*(/|a|*)^**/$

   (b) $/*(a|/)^**(*|a(a|/)^**)^*/$

   (c) $/*(a|/)^**(*|a(a|/)^**)^**/$

2. (8 points)

   The following grammar generates all regular expressions over the alphabet $\{a, b\}$:

   $$R \rightarrow R + R \mid RR \mid R^* \mid (R) \mid a \mid b$$

   where '+' denotes 'or'. Unary $*$ has highest precedence, followed by concatenation; disjunction has the lowest precedence. All operators are left associative.

   (a) Show that the grammar is ambiguous.

   (b) Construct an equivalent unambiguous grammar that enforces the correct operator precedence and associativity.

   (c) Eliminate any left-recursive productions and productions with common prefixes from your grammar in (b).

3. (4 points) What is meant by ***structural equivalence*** and ***name equivalence*** of types? Is one more restrictive than the other? Is one easier for a compiler to check than the other?

4. (11 points)

   Suppose you are asked to extend the C programming language to allow overloading of user-defined functions, based on the number of formal parameters and their types.

   Answer the following questions as briefly, *yet clearly*, as you can. We are more interested in abstract operations (*e.g.*, "store x in a table, along with its size") than in implementation details (*e.g.*, "for performance, implement the table with a doubly-linked list of AVL trees").

   (a) How can a compiler keep track of the parameters and their types? Describe what should be recorded when a function declaration is processed.

(b) How can a compiler determine which function to invoke in a procedure call? Describe what and how information can be retrieved when resolving a procedure call.

(c) Describe type errors that can occur as a result of this language extension, and how a compiler can detect them.