CS 536 Final Exam

Monday, May 15, 1995

2:45 PM – 4:45 PM CDT

1131 Humanities Bldg

Instructions

Answer *any five* questions. (If you answer six or seven, only the first five will count.) Each question is worth 20 points. Please try to make your answers neat and coherent. Remember, if we can't read it, it's wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

Question #1

Let X be a non-terminal. Let Last(X) be the set of terminal symbols that occur as the last (rightmost) symbols in a derivation from X. Formally, Last(X) = $\{a \in V_t \mid X \Rightarrow^* \cdots a\}$. Assume we have a context-free grammar that contains no λ-productions. Explain how to compute Last(X).

How must your solution be changed if λ-productions are allowed?

Question #2

Assume we have an abstract syntax tree, T, for a CSX program that contains no procedures or functions. Outline how you would write a recursive function accessCnt(T) that traverses T and estimates how many times each global variable is accessed. A variable is accessed each time it is read from or written into. You may assume that in conditionals the then and else parts are each chosen 50% of the time. Moreover, loops, on average, iterate 10 times.

How would your design for accessCnt change if non-recursive procedures and functions, without reference parameters, are allowed?

Question #3

When a run-time error occurs, some programs print an error message (with a line number), followed by a *call trace* which lists the sequence of pending subroutine calls at the time the error occurred. For example, we might have:

Zero divide error in line 12 in procedure "WhosOnFirst"
  Called from line 34 in procedure "WhatsOnSecond"
  Called from line 56 in procedure "IDontKnowsOnThird"
  Called from line 78 in procedure "Main"

Explain what you would have to do to include a call trace (with line numbers) in the code your CSX compiler generates.

Question #4

We will say that a context-free grammar is *indirectly left-recursive* if the grammar contains two productions of the form $A \to B\ \alpha$ and $B \to A\ \beta$.

(a) (5 points)
Show that an indirectly left-recursive grammar can never be LL(1).

(b) (15 points)
Give a grammar transformation that converts indirectly left-recursive grammars into equivalent grammars that are neither indirectly left-recursive nor left-recursive.

Question #5

Assume we add a *conditional expression* of the form Ex_1 ? Ex_2 : Ex_3 to CSX. Ex_1 is an expression that returns a `Bool`. If Ex_1 is true, expression Ex_2 is evaluated, and its value is the value of the conditional expression. If Ex_1 is false, expression Ex_3 is evaluated, and its value is the value of the conditional expression.

Outline the changes that would be needed in a CSX type-checker and code generator to handle conditional expressions. Illustrate your answer using the following example:

```
A = I<>0 ? J/I : Maxint;
```

Question #6

In CSX parameters may be passed by value or reference. Some programming languages include another parameter-passing mechanism *value-result*. When an identifier is passed as a value-result parameter, a copy of its value is passed to the callee at the start of a call (just like a value parameter). During the call, the parameter is treated like a local variable. At the end of the call, the value of the local variable is copied back to the identifier of the caller. (Only identifiers may be passed by value-result.) Thus in the following, `I` has a value of 20 after the call:

```
Main test;
      Int I=10;

      proc P(Int valres A);  // valres means
                             //    value-result parameter
            A = A*2;
      end;

      P(I); // I == 20 after the call
end
```

Explain how you would implement value-result parameters in your CSX compiler. You should explain how identifiers would be passed and how value-result parameters would be accessed within a procedure or function. Show the code you'd generate for the above test program.

Question #7

Programs that use matrices (two dimensional arrays) often require that they be *symmetric*. For symmetric matrices it is always the case that A[b][c] == A[c][b]. An n by n symmetric array can be stored in less than n^2 space (by not storing redundant elements).

Assume we define a new CSX data type `symmatrix<n>` that creates an n by n symmetric matrix of integers. The value of n must be an integer literal. Thus we might have the declaration

```
symmatrix<100> m;
```

Suggest an efficient memory layout for a `symmatrix`. What code would be generated to access `A[i][j]`, assuming `A` is a `symmatrix`?