CS 536

Final Exam

Sunday, May 11, 1997

5:05 — 7:05

1221 CSST

Instructions

Answer any *five* others. (If you answer more, 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!).

1. CSX has the production

   ```
   name → id [ exp ]
   ```

   which generates a subscripted variable.

   Explain what your CSX code generator does to translate a subscripted `name`, so that its value is computed into a register.

   How would you change your code generator so that it also does *subscript checking*. That is, whenever a subscripted variable is evaluated, its subscript is checked (at run-time) to verify that only elements within the array are accessed. (Illegal subscripts should force a jump to the label `bad.subscript`, which prints an error message and terminates execution).

2. Assume a reduced grammar, G, contains three productions with X as the left-hand side. One of these productions is left-recursive, the second is right-recursive (is of the form $X \rightarrow \ldots X$), and the third has a right-hand side containing only terminals. Show that G must be ambiguous.

3. Programs that use matrices (two dimensional arrays) sometimes require that they be *sparse*. For sparse matrices it is the case that almost all elements are equal to a particular *default value* (usually 0).

   An n by n sparse array can be stored in far less than n^2 space by only storing non-default values.

   Assume we define a new CSX data type `sparsematrix<n>` that creates an n by n sparse matrix of integers, with a default value of 0. The value of n must be an integer literal. Thus we might have the declaration

   ```
   sparsematrix<1000> m;
   ```

   Suggest an efficient memory layout for a `sparsematrix`. Evaluate your design in terms of the space required, the time required to fetch or store a default value and the time to fetch or store a non-default value.

4. Assume we have a context-free grammar G that has the property that all productions have right-hand sides containing either two or more symbols **or** a single terminal symbol. That is, no λ-productions are allowed, and no productions that generate a single non-terminal are allowed.

   Assume we have a program generated using G that contains n terminal symbols. Show that the program was generated using no more than $2n$ derivation steps.

5. Show the MIPS code that would be generated for the CSX statement

   ```
   a = b + c;
   ```

   assuming `a` and `c` are local variables and `b` is a global variable. What code would be generated if `a`, `b` and `c` are all by-value parameters?

   It is clear that register-resident variables generate smaller (and faster) code than memory-resident variables. Assume we have one unused register, `$r`. If we decide to allocate one local variable to `$r` (treating the variable like a by-value parameter), what criteria should be use in deciding which local variable to allocate to `$r`?

6. Assume we have the CSX procedure

   ```
   proc p(int a, int b&)
      b = a + 1;
   end;
   ```

   and the call

   ```
   p(x,y);
   ```

   where `x` and `y` are integer global variables. Assume we have too few registers to use them to hold parameters. Instead we pass and access parameters using the run-time stack. Show the code you would generate for `p` and the call to `p` assuming this modified parameter passing mechanism.

7. Assume we add a `switch` statement to CSX. The `switch` statement has the same structure and semantics as the `switch` statement found in C and C++ except that a `default` case label is not allowed.

   The `switch` statement is parsed into an AST rooted by a `switchNode`. A `switchNode` has two subtree pointers: `condition` (of type `exprNode*`) and `cases` (of type `caseNode*`). `caseNode` is an AST node that represents a case within the switch, labeled by one or more integer case labels. `caseNode` has three subtree pointers: `label` (of type `intList*`), `stmts` (of type `stmts-Node*`) and `more` (or type `caseNode*`). Finally, `intListNode` is an AST node that represents a list of one of more integers. `intListNode` has an `int` field `intval` and a pointer `more` (of type `intListNode*)` that points to the rest of the list.

   Explain how you would type check a `switchNode`.