CS 536

Midterm Exam

Tuesday, November 4, 1997

1:00 — 2:20 PM

1221 CSST

Instructions

Answer question #1 and any three others. (If you answer more, only the first four will count.) Point values are as indicated. 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. (1 point)
   What university uses the team nickname "Wisconsin Badgers"?

2. (33 points)
   Let Equal be the set of strings that contain an equal number of a's and b's (in any order). That is, Equal = {ab, ba, aabb, abba, abab, baab, baba, bbaa, ...}.

   Is Equal a regular set? If it is, give a regular expression or finite automaton that defines it. If Equal isn't regular, explain carefully why.

3. (33 points)
   Let s be a string. Define insert#(s) to be a function that inserts a # into each possible position in s. If s is n characters long, insert# returns a set of n+1 strings (since there are n+1 places a # may be inserted in a string of length n).

   For example, insert#(abc) = {#abc, a#bc, ab#c, abc#}. insert# applied to a set of strings is the union of insert# applied to members of the set. Hence insert#(ab, de) = {#ab, a#b, ab#, #de, d#e, de#}.

   Let R be any regular set (my choice). Show that insert#(R) is a regular set.

4. (a) (13 points)
   Is the set of strings that contain no repeated character regular? Why?

   (b) (20 points)
   Write a regular expression for comments that begin with << and end with >>. The body of the comment may contain any character sequence except >> (so that >> always marks the end of the comment).

5. (a) (20 points)
Most programming languages allow users to define new procedures and functions. A few even allow users to define new operators.
Assume we extend CSX to allow users to define new operators composed of one or two characters chosen from `'#'`, `'@'`, `'%'` and `'^'`. An operator is defined just as a member function is except that the "name" of the function is an operator:

```
int %(int a, int b) { // body of % operator's definition
}
```

User-defined operators are used just as predefined operators are:

```
a = b % c;
```

Ignore parsing and type checking issues involved in implementing user-defined operators. How would your CSX scanner have to be changed to allow the declaration and use of such operators?

(b) (13 points)
It is clear that user-defined operators would be a lot more useful if we could use the characters that appear in standard predefined operators: `'+'`, `'-'`, `'*'`, `'/'`, `'='`, `'>'`, `'<'`, `'!'`, `'&'` and `'|'`.
Assume that users *may not* redefine existing operators like + or `!=`, but that they may create new operators like `**` or `<>`.
How must your solution to (a) be changed to support this extension?

6. (33 points)

It is often the case that what looks like an undeclared identifier is actually a valid identifier that has been "misspelled." If we could determine that an apparently undeclared identifier was "very close" to a single valid (properly declared) identifier, we might "repair" the misspelling to the valid identifier (with a suitable warning, of course).

Assume we try to repair only identifiers of three or more characters, and that an identifier i_1 is considered a possible misspelling of a valid identifier i_2 if i_1 can be converted into i_2 by adding, deleting or changing a single character. Thus given the valid identifier cat, we would recognize car (one character changed), scat (one character added) and at (one character deleted) as possible misspellings.
The `Registration` class you programmed knows about all valid identifiers (which are registered before they are used). Assume we add a member function

```
String validId(String badId);
```

`validId` returns the text of a valid, already registered identifier if exactly one such identifier exists that differs from `badId` by exactly one character insertion, deletion or replacement. If more than one possible repair of `badId` exists, or no repair of `badId` exists, `validId` returns the null string.

Explain how `validId` might be *efficiently* implemented. (An algorithm that looks at *all* identifiers and tries to generate *all* possible repairs is certainly not efficient. A string turkey could never be repaired to duck, and should never even be considered.)