TUTORIAL 1 DATE: 22nd FEBRUARY 2001

---

1. **Consider the following Lex-like specification of three distinct tokens:**

$(aba)+$	**Token 1**
$(a(b)*a)$	**Token 2**
$a(a\|b)$	**Token 3**

   (a) **Draw an appropriate syntax tree for the specification which can be directly converted to a DFA. Be careful, because the resulting DFA has to distinguish between lexemes corresponding to the three tokens.**

   (b) **Number the leaves of the syntax tree from left to right. Now construct the DFA labeling each state with sets of positions.**

   (c) **Indicate the token(s) found against each accepting state. Are there any clashes? If yes, why?**

2. **The time taken to do lexical analysis, it is generally thought, is linear in the length of the input string. The question below shows that this is not always the case.**

   Consider the following Lex-like description of two tokens:

   abc – **Token 1**
   $(abc)^*d$ – **Token 2**

   (a) **Draw the DFA corresponding to this regular expression. Indicate against each final state, the token recognised by the state.**

   (b) **Give an input string whose complete tokenization (by complete tokenization we mean extraction of all the tokens in the given string) would take time that is quadratic in its length.**

3. **Comments in C consist of any sequence of characters enclosed between `/*` and `*/`. The enclosed characters should not contain the sequesnce `*/`. Examples of valid comments are `/* this is a comment */`, `/**/` and `/*/*/`. Example of non-comments are `/*/` and `/*/*/*/`.**

   Write a regular expression for C comments. You can assume that apart from / and *, the input alphabet consists of the characters `a-z`. Explain why your solution is correct.

4. **Similarly strings are characters enclosed within `"` and `"`. A `"` enclosed in a string is represented as `""`. Write a regular expression for strings.**

5. **Show that the following grammar, intended to remove the *if-then-else* ambiguity, is still ambiguous.**

   stmt → `if expr then` *stmt* | *matched_statement*
   matched_statement → `if expr then` *matched_statement* `else` *stmt*