CS164: Written Assignment 1
(On Lexers and Lexer Generators)

Assigned: Tuesday, Sep 16, 2004
Due: Tuesday, Sep 23, 2004, at the beginning of class.

Grading and Submission

Your answers must be brief and easy to understand. Your grade (credit/no credit) will depend partly on how easy it is for us to understand and verify your answer. Submit your written assignments either in the classroom (before the lecture) or in 283 Soda. *No late homeworks are accepted.*

1 Generating a Lexer

This problem asks you to generate a lexer from a high-level specification of the lexer. Essentially, you are to do by hand what the lexer generator you'll create in PA2 will do automatically.

Question: (**a**) Draw the NFA for the following `lex` specification (the NFA we are referring to is the R from slide 9 in Lecture 3; you can read about `lex` in either textbook). (You need to draw the state machine graphically; don't write the Java code produced by the lexer generator.) To simplify the problem, assume that the action can do nothing more than return a token (e.g., no assignments are allowed). Your accepting states thus should store only the value of the token.

```
(aba)+    { return TOKEN_1; }
(a(b*)a)  { return TOKEN_2; }
(a|b)     { return TOKEN_3; }
```

(**b**) Write down the pseudo-code of the algorithm for using the state machine to scan the input string with this NFA. This algorithm should be a high-level version of the code you are to write in `LookaheadLexer.nextToken()` in PA2. Your lexer must follow the maximal-munch rule. To show that it does that, you need to show what to do when the NFA (i) reaches an accepting state; (ii) is able to read the next input character but it is not in an accepting state; and (iii) gets stuck. Indicate what the algorithm does with the input (does it read the next character; does it return some look-ahead characters to the input?).

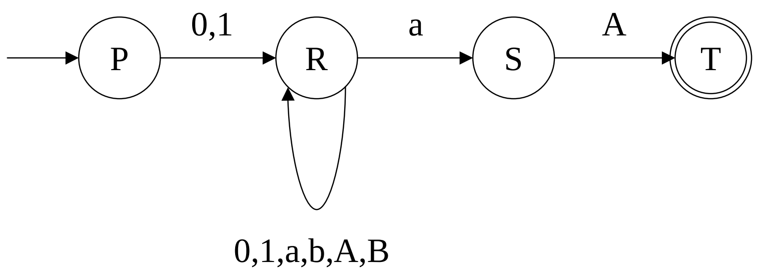
(**c**) Show each step of the lexer on the string abaabbaba. For each step, show what input character was consumed, what is the remaining content of the input, what states the NFA transited to, and also what NFA action was performed (e.g., were any characters returned to the input in this step)?

(**d**) To implement the maximal-munch rule, the lexer must perform look-ahead (i.e., read some characters that may need to be returned back to the input stream). Write a short lexer specification (with just two simple regular expressions) on which the necessary look-ahead is unbounded (i.e., the lexer may need to return to the input as many characters as the length of the input (minus some small constant)).

2 Speeding Up the Lexer (Removing Non-Determinism)

While the simple lexer generator that you will implement in PA2 will produce an NFA, the industrial-quality lexer generators like `lex` may convert their NFA to a DFA, in order to speedup lexing. You are not asked to implement NFA-to-DFA conversion in PA2, but you will get a practice with the conversion procedure in this homework.

Question: Convert the following NFA to a DFA.

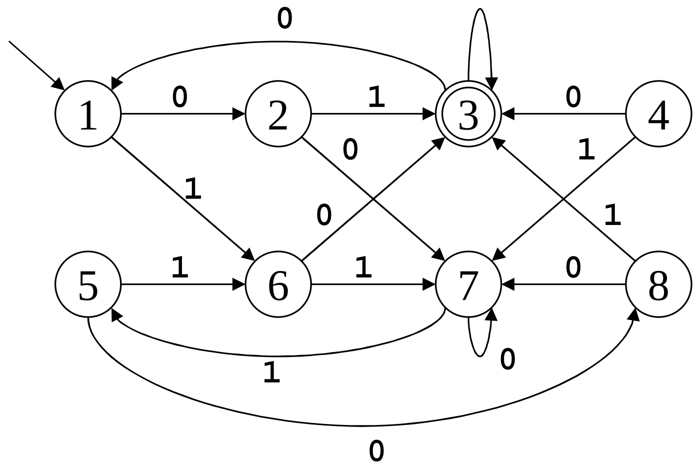


3 Shrinking the Lexer (DFA Minimization)

After an NFA is converted to a DFA, the DFA may not be as small as possible. This problem illustrates what a lexer generator may do to ensure that the resulting DFA-based lexer occupies as little memory as possible.

Question: First, study and understand the DFA minimization algorithm. This algorithm was sketched in the lecture. More details are in Section 3.6.2 in [FLC], or in Algorithm 3.6 on page 142 in [ASU].

Now, the DFA below contains many more states than are necessary to recognize the desired language. Your task is to use the minimization algorithm to minimize this DFA.


1
2
3
4
5
6
7
8
0
1