© 2018 Sudarshan S. Chawathe.

Follow the **guidelines** of the previous homework for packaging, submission, and allowable use of resources. Use the newsgroup for questions, clarifications, and discussion in general. The question marked with ⋆ is optional for COS 451 but required for COS 550.

The **goal** of this assignment is to solidify our understanding of regular expressions and related automata using the evolving language *Lexaard* from previous assignments.

1. (50 pts.) Extend Lexaard to parse and print regular expressions (*regex*es). In particular, extend the `define` and `print` statements to work with regexes. You must fully parse the regex, not just store it in a form that allows printing.

   In order to simplify parsing, we use a prefix-based syntactic representation of regular expressions instead of the infix-based representation used in the textbook. For example, the regular expression `((a|bb|cc*)a)*` is represented as follows:

   ```
   (r* (r. (r| a ( r. b b) (r. c (r* c))) a))
   ```

   The general syntax is summarized below.

syntax	interpretation
`c` (for any char c)	regex accepting the single-character string c
`r.`	regex accepting the empty string, i.e., the regex ϵ
`r/`	regex accepting nothing, i.e., the regex $\emptyset$
`(r|` *r1 r2 ...*`)`	`r1` $\cup$ `r2` $\cup \ldots$ (regex union)
`(r.` *r1 r2 ...*`)`	`r1` $\circ$ `r2` $\cup \ldots$ (regex concatenation)
`(r*` *r1*`)`	`r1`$*$ (regex star)

   As before, whitespace is insignificant except when it separates tokens that would otherwise run together. So the earlier example may also be written as follows, making it easier for humans to read:

   ```
   (r* (r. (r| a
               ( r. b b)
           (r. c
               (r* c)))
           a))
   ```

2. (50 pts.) Extend Lexaard with a function `regex2fsa` that converts a regex to an FSA using the algorithm in the proof of Lemma 1.55 in the textbook. This function is accessed in Lexaard in a manner analogous to the functions described in the previous assignment.

3. (50 pts.) Extend Lexaard to parse and print GNFAs as described in the proof of Lemma 1.60 in the textbook. In particular, extend the `define` and `print` commands to work with GNFAs. As in Question 1, you must fully parse the GNFAs.

The syntactic representation of GNFAs in Lexaard is similar to the representation used for NFAs earlier, as illustrated by the following representation of the GNFA from Figure 1.61 of the textbook, with named `q1` to `q4` in left-to-right order.

```
gnfa
gnfa1_61 GNFA on p 70 of textbook
a b
     q1  q2         q3              q4
 q1  ..  r/         (r. a (r* b))   b
 q2  ..  (r. a b)   (r* (r. a a))   (r* b)
 q3  ..  (r* a)     ..              (r| (r. a b) (r. b a))
 q4  ..  ..         ..              ..
```

The representation of GNFAs differs from the earlier representation of NFAs in three ways: First, since a GNFA has a single accept state, we do not use the `*` marker to denote accepting states. Instead, we follow the convention that the last state (see below) is the accepting state. As before, the first state is the start state.

Second, the format of the transition table, which follows the alphabet row (row 3) as before, is different, corresponding to the difference between NFA transition functions, which have type $Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ (Definition 1.37) and GNFA transition functions, which have type $(Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \to \mathcal{R}$ (Definition 1.64). The transition table is represented as an adjacency matrix of the labeled digraph that forms the GNFA's pictorial representation. In more detail, the first row of the transition table (row 4), called its header row, is a listing of the states in lexicographic order by name, with the following two exceptions: The start state is always listed first and the accept state is always listed last.

Third, following the header row is one row for each state in the GNFA, representing the outgoing transitions from that state. These per-state rows are listed in an order matching the order of states in the header row. The first column (header column) of a state's row holds the name of the state followed by a single entry for each state of the GNFA. Excluding the header row and header column of the transition table, the entry in the jth column of the ith row is either `..`, denoting the absence of an edge from state i to state j, or a regex in the syntax noted above, denoting an edge with that regex as label.

4. (50 pts.) Extend Lexaard with a function `fsa2regex` that converts an FSA to a regex using the algorithm in the proof of Lemma 1.60 in the textbook. It is accessed in Lexaard in the usual manner. If the FSA provided as an argument to this function is not a DFA, the necessary conversion should be performed automatically.

5. $\star$ (25 pts.) Extend Lexaard with a boolean function `regexEqiv` that returns true iff the two regular expressions given as its arguments are equivalent.