

## Discussion Section: Regular Expressions and Grammars

---

Handout written by David Underhill.

### Problem 1 A lex of Fun

- a. **Precedence.** What factors affect a rule's matching precedence in lex?
- b. **!matching.** Write a lexer which ECHOs all input lines which do not contain the string EVIL.
- c. **States.** Extend the lexer so that when it scans the string `lenient` it will transition to a new state in which it ECHOs all lines (even EVIL ones). Typing `strict` should cause it to transition back to the default behavior. Use states to construct this functionality.

### Problem 2 Hard Hat Area: Regular Scientists at Work

Construct a regular expression over the alphabet  $\{c, s\}$  such that the first and last letters are identical.

### Problem 3 Hello, Aloha, Halo, Bonjour, ...

- a. Take your regular expression from Problem 2 and construct a  $\lambda$ -NDFSA using the rules from Thompson's construction. Show your work step-by-step and indicate which rule you apply at each step in order to get to next one.
- b. Now that you have an  $\lambda$ -NDFSA, convert it to an equivalent DFA. You should first remove any  $\lambda$ -cycles (if any), then remove any  $\lambda$ -transitions, and finally remove any remaining non-determinism. Show each of these stages (e.g. no  $\lambda$ -cycles, no  $\lambda$ -transitions, and the final product).
- c. Minimize the number of states in your DFA (or prove that it already has the minimal number of states). Show your work.
- d. Minimize the number of states in the DFA shown in Figure 1.
- e. Construct an equivalent context-free grammar which could be parsed with an LL(1) parser.

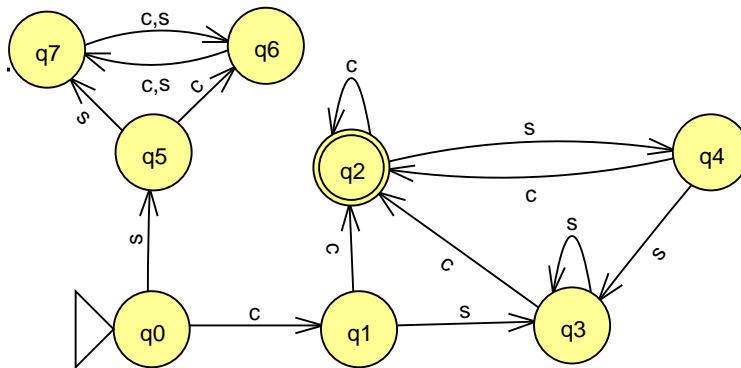


Figure 1: A DFA which is less concise than it could be.

#### Problem 4 Dive, Dive, Dive!

- Show the parse tree(s) for a left-most derivation of the input  $ccsc$ .
- Generate a top-down predictive parser's parse table for the LL(1) grammar you just created.
- Trace through a parse for the input  $ccsc$  with your predictive parser's table.