COMS W4115 Fall 2020: Homework Assignment 1 Solutions

Programming Languages and Translators

Deadline: Monday, October 19, 2020 by 11:59 PM

Overview

This homework assignment will test your knowledge of regular expressions and finite automata, which you have learned in class. Please submit your assignment via Gradescope by the deadline. This is an *individual* assignment, and you must adhere to the academic integrity policies of the course.

Total Points: 50
Number of Problems: 5

Problems

1. **(6 Points)** Create regular expressions for the following languages over the alphabet $\Sigma = \{0, 1\}$:

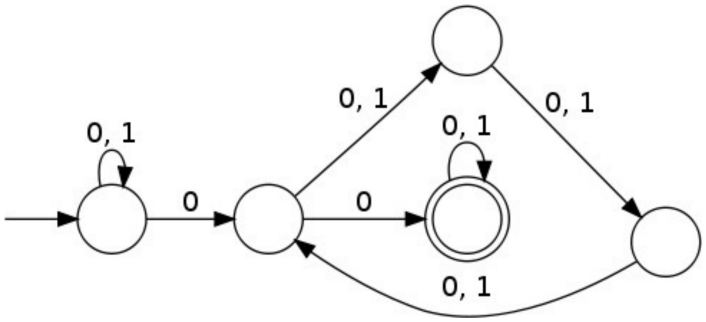
   (a) **(3 Points)** All inputs with an odd number of "0"s

   Two possible regular expressions are 1*01*(01*01*)* and (1 | 01*0)*01*. Note that | is the OR operator here.

   (b) **(3 Points)** All inputs ending with a "1" and without the sequence of digits "10"

   The regular expression is 0^*1^+. If we think of the set of inputs that don't contain "10," then all inputs must contain all of the "0"s before the "1"s if both happen to exist. Given this and the fact that all valid inputs must end with a "1" (think $^+$ instead of *), this is the correct regular expression.

   Assume that empty inputs may be valid.



2. **(4 Points)** Consider the above nondeterministic finite automaton (NFA) over the alphabet $\Sigma = \{0, 1\}$. Create a regular expression for the language represented by this NFA.

   The correct regular expression for the language represented by this NFA is $(0 \mid 1)^*0((0 \mid 1)(0 \mid 1)(0 \mid 1))^*0(0 \mid 1)^*$. Note that $\mid$ is the OR operator here.

3. **(10 Points)** The following list provides a lexical specification of rules (regular expressions) used in this problem, in order of priority:

   - $[a-z][a-z\ 0-9]^*$
   - $for \mid while \mid if \mid else$[1]
   - $[0-9]^+$
   - $[0-9]^*[a-z]$
   - *dictionary*

   Please specify which of the above rules would be used to tokenize each of the following tokens. Here are some important notes:

   - **It is possible that more than one rule is needed.** As a hint, you can *partition* the input. Partitioning allows for splitting the input into lexemes and applying the rules of a lexical specification to individual lexemes. Suppose you had the input *abc*; this input can be partitioned into three characters, each of which satisfies the first rule (and would result in a valid set of rules). However, note further that the entire input itself satisfies the first rule. Because of maximal munch, we choose the latter solution over the former. In some cases, though, it is necessary to partition the input.
   - **It is also possible that no rules apply.** If it is not possible to partition the input such that each lexeme is matched by at least one of the rules, then the input is not valid for the language.

   In all cases, please provide a one- or two-sentence explanation as to why rules were chosen or no rules were chosen. Assume that we follow the *rule of thumb* and *maximal munch* policies.

   (a) **(2 Points)** *coms*4115

   Rule 1 can be used, as the entire input is covered by the rule (applying the maximal munch and rule of thumb policies).

   (b) **(2 Points)** *while*

   Rule 1 can be used again, as the entire input is covered by the rule (applying the maximal munch and rule of thumb policies). The key aspect of this problem is to recognize that the first rule would be chosen over the second by rule of thumb (even though *while* is a keyword); the fact that the first rule covers all of the cases of the second rule indicates an inadequacy of this lexical specification.

   (c) **(2 Points)** 123*abc*

   Rules 4 and 1 apply, in that order. We can partition the input into 123*a* and *bc*. The first lexeme is matched by rule 4, and the second lexeme is then matched by rule 1.

   Note that we could have chosen to partition the input as 123 and *abc*; however, this is incorrect because of maximal munch. We choose as much of the input as possible to be covered by at least one of the rules. In this case, 123*a* is the longest substring of the input that can be covered by a rule, rule 4.

[1] The "|" represents an OR operation.

(d) **(2 Points)** *dictionary*

For the same reasoning as in part (b), rule 1 is chosen. The first rule covers fifth rule, which will never be chosen.

(e) **(2 Points)** The empty string

None of the rules apply, since every rule requires at least one input character. The first rule requires a letter, the second rule requires a selection from a fixed set of keywords, the third rule requires at least one digit, the fourth rule requires that the token end in a letter, and the fifth rule requires selecting a particular identifier.

Here are some examples:

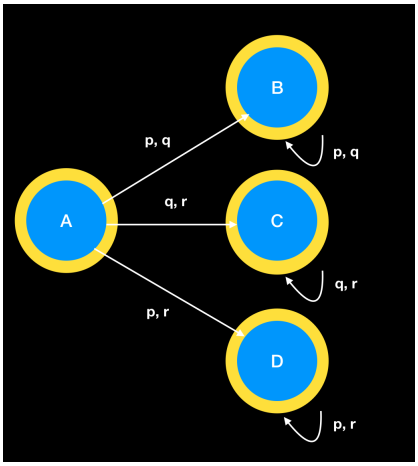
- *a*123: the first rule is a match by the *rule of thumb* policy, and only one rule is needed because the entire token is matched by the rule (maximal munch).
- _*abc*: none of the rules apply, since the underscore is not matched by any rule.

4. **(20 Points)** For a given alphabet $\Sigma = \{p, q, r\}$, assume there is a regular language L representing all strings that use at most two of the three letters in this alphabet. Examples of valid strings include *rprpp*, *p*, *qqq*, *pqqqp*, and the empty string. An example of an invalid string is *pqrpr*, since it contains all three letters.

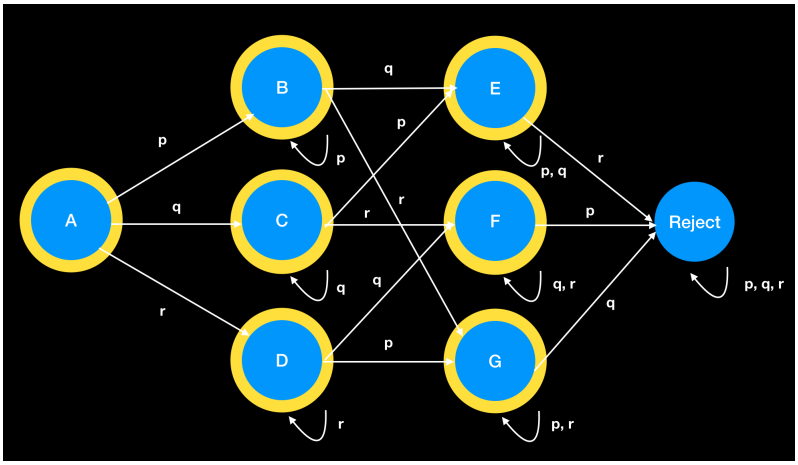
   (a) **(5 Points)** Create a regular expression for L.

   A regular expression for L is $(p \mid q)^* \mid (q \mid r)^* \mid (p \mid r)^*$. Note that $\mid$ is the OR operator here. You can have strings with at most two of the three letters, so there are three possible combinations of two-letter pairings. Since the empty string is valid, you use * on the groupings instead of $^+$.

   (b) **(5 Points)** Draw an NFA for L. You are not required to include ϵ transitions, but you can include them if they help you.



   (c) **(10 Points)** Draw a DFA for L. You do not need to show the transition table, but including the transition table in your solution may help us assign partial credit.



5. **(10 Points)** For a given alphabet $\Sigma = \{0, 1\}$, assume there is a regular language L representing the set of binary numbers whose decimal value is divisible by 3. Examples of valid numbers are 0 (0), 11 (3), and 1111 (15), while 0111 (7) would be considered invalid. Numbers are unsigned (assume only non-negative numbers) and can have any number of leading "0"s. Further, note that inputs are processed from left to right, so a number like 0011101 reads 0 first, followed by 0 again, then 1, and so on. Please draw a DFA for L.

   The following is a DFA for L: