CS/ECE 374 Sec A ✧ Spring 2021

Homework 2

Due Wednesday, February 1, 2023 at 10am

---

0 Several problems on creating regular expressions and DFAs can be found in Jeff's notes, Sipser's book, Aho-Motwani-Ullman book, and several others. They cover a wide range from easy to hard. We have placed several optional problem on PrarieLearn that you can try out.

1. For each of the following languages assume that the alphabet is $\{0, 1\}$. Give a regular expression that describes that language, and briefly argue why your expression is correct.

   (a) All strings that that end in `10` *and* contain `111` as a substring.

   (b) All strings that do not contain the *subsequence* `010`.

   (c) All string in which the number of `1`s is not divisible by 3.

   (d) A finite language $L = \{w_1, w_2, \ldots, w_k\}$ where each w_i is a binary string.

   (e) The complement $\bar{L}$ of a finite language $L = \{w_1, w_2, \ldots, w_k\}$. You may want to consider the parameter $h = \max_{1 \le i \le k} |w_i|$, the length of the longest string in L.

2. DFA design. For the first three parts describe a DFA by drawing it explicitly and briefly explain the *meaning* of each state.

   (a) All strings in $\{0, 1\}^*$ that end in `11`.

   (b) All strings in $\{0, 1\}^*$ that start with `01` and whose length is divisible by 4.

   (c) All strings in $\{0, 1\}^*$ that do *not* contain `100110` as a substring.

   (d) Consider a fixed string $s = a_1 a_2 \ldots a_k$ of length k. Let $L_s = \{w \in \Sigma^* \mid w \text{ contains string } s \text{ as a substring}\}$. Describe in formal tuple notation a DFA M that accepts L_s. Do not try to optimize number of states. Briefly describe the meaning of the states and your construction. How many states does your DFA have as a function of k?

   (e) Let L_1, L_2, L_3, L_4 be regular languages over Σ accepted by DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$, $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$, $M_3 = (Q_3, \Sigma, \delta_3, s_3, A_3)$, and $M_4 = (Q_4, \Sigma, \delta_4, s_4, A_4)$ respectively. Describe a DFA $M = (Q, \Sigma, \delta, s, A)$ in terms of M_1, M_2, M_3, M_4 that accepts $L = \{w \mid w \text{ is contained in exactly one of } \{L_1, L_2, L_3, L_4\}\}$. Formally specify the components Q, δ, s, and A for M in terms of the components of M_1, M_2, M_3, M_4. You do not need to prove correctness of the construction.

3. **Not for submission.** Consider the strings over the alphabet {0, 1, 2} as representing ternary numbers (i.e., numbers in base 3). Let L be the language of strings that represent ternary numbers divisible by 5. For example, 120 would be in the language since $120_3 = 1 \cdot 3^2 + 2 \cdot 3 = 15$, while 200 would not.

   Describe a DFA over the alphabet $\Sigma = \{0, 1, 2\}$ that accepts the language L. Argue that your machine accepts every string in L and nothing else, by explaining what each state in your DFA *means*.

   You may either draw the DFA or describe it formally, but the states Q, the start state s, the accepting states A, and the transition function δ must be clearly specified.

4. **Not for submission.** Give a string w we use the notation w^R to denote its reverse. We can extend this notion to languages as follows. Given a language $L \subseteq \Sigma^*$ we define the reverse of L, denoted by L^R as follows:

$$L^R = \{w^R \mid w \in L\}.$$

   In other words L^R is the language consisting of the reverses of the strings in L. In this problem your goal is to develop an algorithm that given a regular expression r, converts it into a regular expression r' such that $L(r')$ is $(L(r))^R$. As a byproduct this also shows that L^R is regular whenever R is regular. Fix Σ to be {0, 1} for simplicity.

   - Consider each of the base cases of the regular expressions. For each r corresponding to the base cases define r' appropriately such that $L(r') = (L(r))^R$.
   - Suppose $r = r_1 + r_2$. Assuming that you have found, recursively, regular expressions r'_1 and r'_2 for the reverses of r_1, r_2 respectively, describe a regular expression r' for the reverse of $L(r)$.
   - Suppose $r = r_1 r_2$. Do the same as in the preceding part.
   - Suppose $r = (r_1)^*$. Find a regular expression r' for the reverse of $L(r)$.
   - Assuming $r = 0^* + (01 + 100)^*(11^* + \epsilon + 0)$ what would your algorithm output for the reverse of $L(r)$?

Solved problem

4. ***C comments*** are the set of strings over alphabet $\Sigma = \{$`∗`, `/`, `A`, `⋄`, `↲`$\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here `↲` represents the newline character, `⋄` represents any other whitespace character (like the space and tab characters), and `A` represents any non-whitespace character other than `∗` or `/`.[1] There are two types of C comments:

    - Line comments: Strings of the form `//···↲`.
    - Block comments: Strings of the form `/∗···∗/`.

    Following the C99 standard, we explicitly disallow ***nesting*** comments of the same type. A line comment starts with `//` and ends at the first `↲` after the opening `//`. A block comment starts with `/∗` and ends at the the first `∗/` completely after the opening `/∗`; in particular, every block comment has at least two `∗`s. For example, each of the following strings is a valid C comment:

    - `/∗∗∗/`
    - `//⋄//⋄↲`
    - `/∗///⋄∗⋄↲∗∗/`
    - `/∗⋄//⋄↲⋄∗/`

    On the other hand, *none* of the following strings is a valid C comments:

    - `/∗/`
    - `//⋄//⋄↲⋄↲`
    - `/∗⋄/∗⋄∗/⋄∗/`

    (a) Describe a DFA that accepts the set of all C comments.

    (b) Describe a DFA that accepts the set of all strings composed entirely of blanks (`⋄`), newlines (`↲`), and C comments.

    You must explain *in English* how your DFAs work. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

---

[1]The actual C commenting syntax is considerably more complex than described here, because of character and string literals.

- The opening `/∗` or `//` of a comment must not be inside a string literal (`“···“`) or a (multi-)character literal (`'···'`).
- The opening double-quote of a string literal must not be inside a character literal (`'"'`) or a comment.
- The closing double-quote of a string literal must not be escaped (`\"`)
- The opening single-quote of a character literal must not be inside a string literal (`"···'···"`) or a comment.
- The closing single-quote of a character literal must not be escaped (`\'`)
- A backslash escapes the next symbol if and only if it is not itself escaped (`\\`) or inside a comment.

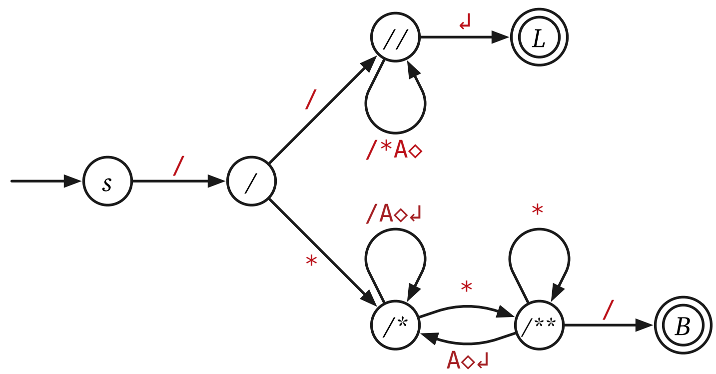
For example, the string `"/∗\\\"∗/"/∗"/∗\"/∗"∗/` is a valid string literal (representing the 5-character string `/∗\"\∗/`, which is itself a valid block comment!) followed immediately by a valid block comment. ***For this homework question, just pretend that the characters `'`, `"`, and `\` don't exist.***

Commenting in C++ is even more complicated, thanks to the addition of *raw* string literals. Don't ask.

Some C and C++ compilers do support nested block comments, in violation of the language specification. A few other languages, like OCaml, explicitly allow nesting block comments.

Solution:

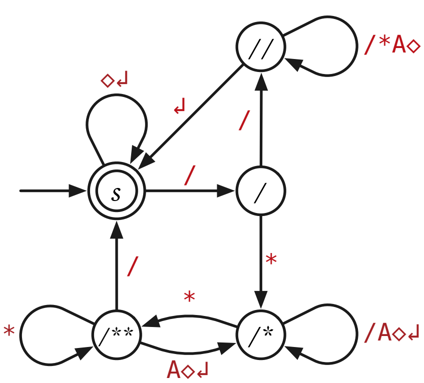
(a) The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- s — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- L — We have read a complete line comment.
- /* — We are reading a block comment, and we did not just read a $*$ after the opening /$*$.
- /** — We are reading a block comment, and we just read a $*$ after the opening /$*$.
- B — We have read a complete block comment.

(b) By merging the accepting states of the previous DFA with the start state and adding white-space transitions at the start state, we obtain the following six-state DFA. Again, all missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- s — We are between comments.
- / — We just read the initial / of a comment.
- // — We are reading a line comment.
- /* — We are reading a block comment, and we did not just read a $*$ after the opening /$*$.
- /** — We are reading a block comment, and we just read a $*$ after the opening /$*$.

■

Rubric: 10 points = 5 for each part, using the standard DFA design rubric (scaled)

Rubric (DFA design): For problems worth 10 points:

- 2 points for an unambiguous description of a DFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
  - **For drawings:** Use an arrow from nowhere to indicate s, and doubled circles to indicate accepting states A. If $A = \varnothing$, say so explicitly. If your drawing omits a reject state, say so explicitly. **Draw neatly!** If we can't read your solution, we can't give you credit for it,.
  - **For text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - **For product constructions:** You must give a complete description of the states and transition functions of the DFAs you are combining (as either drawings or text), together with the accepting states of the product DFA.
- **Homework only:** 4 points for *briefly* and correctly explaining the purpose of each state *in English*. This is how you justify that your DFA is correct.
  - For product constructions, explaining the states in the factor DFAs is enough.
  - **Deadly Sin:** ("Declare your variables.") No credit for the problem if the English description is missing, *even if the DFA is correct*.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - -1 for a single mistake: a single misdirected transition, a single missing or extra accept state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
  - -2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - -4 for incorrectly accepting/rejecting an infinite number of strings.
- DFA drawings with too many states may be penalized. DFA drawings with *significantly* too many states may get no credit at all.
- Half credit for describing an NFA when the problem asks for a DFA.