CS/ECE 374 A ✧ Fall 2019
Homework 1

Due Tuesday, September 10, 2019 at 8pm

Starting with this homework, groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

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

   (a) All strings except `010`.

   (b) All strings that contain the substring `010`.

   (c) All strings that contain the subsequence `010`.

   (d) All strings that do not contain the substring `010`.

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

   (The technical terms "substring" and "subsequence" are defined in the lecture notes.)

2. Let L be the set of all strings in $\{0, 1\}^*$ that contain *at least two* occurrences of the substring `010`.

   (a) Give a regular expression for L, and briefly argue why your expression is correct.

   (b) Describe a DFA over the alphabet $\Sigma = \{0, 1\}$ that accepts the language L.

   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. (See the standard DFA rubric for more details.)

   Argue that your DFA is correct by explaining what each state in your DFA *means*. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

   [Hint: The shortest string in L has length 5.]

3. Let L denote the set of all strings $w \in \{0, 1\}^*$ that satisfy *at most two* of the following conditions:

   - The substring 01 appears in w an odd number of times.
   - $\#(1, w)$ is divisible by 3.
   - The binary value of w is *not* a multiple of 7.

   For example: The string 00100101 satisfies all three conditions, so 00100011 is ***not*** in L, and the empty string ε satisfies only the second condition, so $\varepsilon \in L$. (01 appears in ε zero times, and the binary value of ε is 0, because what else could it be?)

   Formally describe a DFA with input alphabet $\Sigma = \{0, 1\}$ that accepts the language L, by explicitly describing the states Q, the start state s, the accepting states A, and the transition function δ. Do not attempt to *draw* your DFA; the smallest DFA for this language has 84 states, which is *way* too many for a drawing to be understandable.

   Argue that your machine is correct by explaining what each state in your DFA *means*. Formal descriptions without English explanations will receive no credit, even if they are correct. (See the standard DFA rubric for more details.)

   This is an exercise in clear communication. We are not only asking you to design a *correct* DFA. We are also asking you to clearly, precisely, and convincingly explain your DFA to another human being who understands DFAs but has *not* thought about this particular problem. Excessive formality and excessive brevity will hurt you just as much as imprecision and handwaving.

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 comment:

   `/*/`   `//◇//◇↲◇↲`   `/*◇/*◇*/◇*/`

   (Questions about C comments start on the next page.)

[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.

(a) Describe a regular expression for the set of all C comments.

Solution:

$$//(/ + \star + \mathtt{A} + \diamond)^*\hookleftarrow \quad + \quad /\star\left(/ + \mathtt{A} + \diamond + \hookleftarrow + \star\star^*(\mathtt{A} + \diamond + \hookleftarrow)\right)^* \star^*\star/$$

The first subexpression matches all line comments, and the second subexpression matches all block comments. Within a block comment, we can freely use any symbol other than $\star$, but any run of $\star$s must be followed by a character in $(\mathtt{A} + \diamond + \hookleftarrow)$ or by the closing slash of the comment. ■

Rubric: Standard regular expression rubric. This is not the only correct solution.

(b) Describe a regular expression for the set of all strings composed entirely of blanks ($\diamond$), newlines ($\hookleftarrow$), and C comments.

Solution:

$$\left(\diamond + \hookleftarrow \;+\; //(/ + \star + \mathtt{A} + \diamond)^*\hookleftarrow + /\star(/ + \mathtt{A} + \diamond + \hookleftarrow + \star\star^*(\mathtt{A} + \diamond + \hookleftarrow))^*\star\star^*/\right)^*$$

This regular expression has the form $(\langle\text{whitespace}\rangle + \langle\text{comment}\rangle)^*$, where $\langle\text{whitespace}\rangle$ is the regular expression $\diamond + \hookleftarrow$ and $\langle\text{comment}\rangle$ is the regular expression from part (a). ■

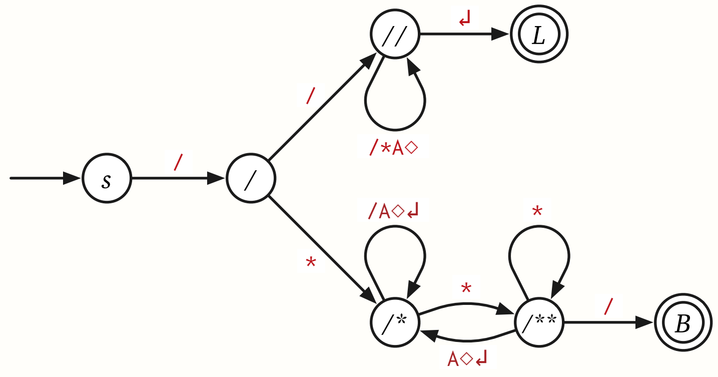
Rubric: Standard regular expression rubric. This is not the only correct solution.

Standard regular expression rubric. For problems worth 10 points:

- 2 points for a syntactically correct regular expression.
- **Homework only:** 4 points for a *brief* English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - **Deadly Sin ("Declare your variables."): No credit for the problem if the English explanation is missing, *even if the regular expression is correct*.**
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a *transcription*; don't just translate the regular-expression *notation* into English.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - −1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language.
  - −2 for incorrectly including/excluding more than one but a finite number of strings.
  - −4 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are more complex than necessary may be penalized. Regular expressions that are *significantly* too complex may get no credit at all. On the other hand, minimal regular expressions are *not* required for full credit.

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

Solution: 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:

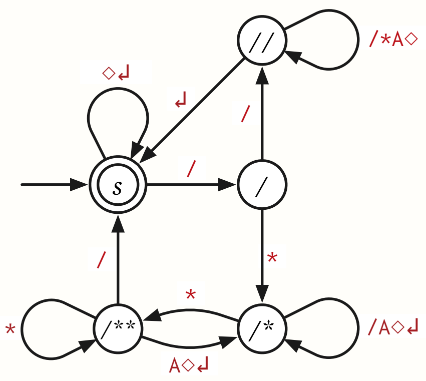
- *s* — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- *L* — We have just read a complete line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read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

■

Rubric: Standard DFA design rubric. This is not the only correct solution, or even the simplest correct solution. (We don't need two distinct accepting states.)

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

Solution: 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: Standard DFA design rubric. This is not the only correct solution, but it is the simplest correct solution.

Standard DFA design rubric. 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.
  - **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 junk/trash/reject state, say so explicitly. **Draw neatly!** If we can't read your solution, we can't give you credit for it.
  - **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - **Product constructions:** You must give a complete description of each the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA.
- **Homework only:** 4 points for *briefly* explaining the purpose of each state *in English*. This is how you argue that your DFA is correct.
  - **Deadly Sin ("Declare your variables."): No credit for the problem if the English description is missing, *even if the DFA is correct*.**
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
- 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 accepting 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.
- DFAs that are more complex than necessary may be penalized. DFAs that are *significantly* more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are *not* required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA.

*5. Recall that the reversal w^R of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

The reversal L^R of any *language* L is the set of reversals of all strings in L:

$$L^R := \left\{ w^R \;\middle|\; w \in L \right\}.$$

Prove that the reversal of every regular language is regular.

Solution: Let r be an arbitrary regular expression. We want to derive a regular expression r' such that $L(r') = L(r)^R$.

Assume for any proper subexpression s of s that there is a regular expression s' such that $L(s') = L(s)^R$.

There are five cases to consider (mirroring the definition of regular expressions).

(a) If $r = \varnothing$, then we set $r' = \varnothing$, so that

$$\begin{aligned} L(r)^R &= L(\varnothing)^R && \text{because } r = \varnothing \\ &= \varnothing^R && \text{because } L(\varnothing) = \varnothing \\ &= \varnothing && \text{because } \varnothing^R = \varnothing \\ &= L(\varnothing) && \text{because } L(\varnothing) = \varnothing \\ &= L(r') && \text{because } r = \varnothing \end{aligned}$$

(b) If $r = w$ for some string $w \in \Sigma^*$, then we set $r' := w^R$, so that

$$\begin{aligned} L(r)^R &= L(w)^R && \text{because } r = w \\ &= \{w\}^R && \text{because } L(\langle\text{string}\rangle) = \{\langle\text{string}\rangle\} \\ &= \{w^R\} && \text{by definition of } L^R \\ &= L(w^R) && \text{because } L(\langle\text{string}\rangle) = \{\langle\text{string}\rangle\} \\ &= L(r') && \text{because } r = w^R \end{aligned}$$

(c) Suppose $r = s^*$ for some regular expression s. The inductive hypothesis implies a regular expressions s' such that $L(s') = L(s)^R$. Let $\boldsymbol{r' = (s')^*}$; then we have

$$\begin{aligned} L(r)^R &= L(s^*)^R && \text{because } r = s^* \\ &= (L(s)^*)^R && \text{by definition of } {}^* \\ &= (L(s)^R)^* && \text{because } (L^R)^* = (L^*)^R \\ &= (L(s'))^* && \text{by definition of } s' \\ &= L((s')^*) && \text{by definition of } {}^* \\ &= L(r') && \text{by definition of } r' \end{aligned}$$

(d) Suppose $r = s + t$ for some regular expressions s and t. The inductive hypothesis implies regular expressions s' and t' such that $L(s') = L(s)^R$ and $L(t') = L(t)^R$.

Set $\boldsymbol{r' := s' + t'}$; then we have

$$\begin{aligned}
L(r)^R &= L(s+t)^R && \text{because } r = s+t\\
&= (L(s) \cup L(t))^R && \text{by definition of } +\\
&= \{w^R \mid w \in (L(s) \cup L(t))\} && \text{by definition of } L^R\\
&= \{w^R \mid w \in L(s) \text{ or } w \cup L(t)\} && \text{by definition of } \cup\\
&= \{w^R \mid w \in L(s)\} \ \cup\ \{w^R \mid w \cup L(t)\} && \text{by definition of } \cup\\
&= L(s)^R \cup L(t)^R && \text{by definition of } L^R\\
&= L(s') \cup L(t') && \text{by definition of } s' \text{ and } t'\\
&= L(s' + t') && \text{by definition of } +\\
&= L(r') && \text{by definition of } r'
\end{aligned}$$

(e) Suppose $r = s \bullet t$ for some regular expressions s and t. The inductive hypothesis implies regular expressions s' and t' such that $L(s') = L(s)^R$ and $L(t') = L(t)^R$. Set $\boldsymbol{r' = t's'}$; then we have

$$\begin{aligned}
L(r)^R &= L(st)^R && \text{because } r = s+t\\
&= (L(s) \bullet L(t))^R && \text{by definition of } \bullet\\
&= \{w^R \mid w \in (L(s) \bullet L(t))\} && \text{by definition of } L^R\\
&= \{(x \bullet y)^R \mid x \in L(s) \text{ and } y \in L(t)\} && \text{by definition of } \bullet\\
&= \{y^R \bullet x^R \mid x \in L(s) \text{ and } y \in L(t)\} && \text{concatenation reversal}\\
&= \{y' \bullet x' \mid x' \in L(s)^R \text{ and } y' \in L(t)^R\} && \text{by definition of } L^R\\
&= \{y' \bullet x' \mid x' \in L(s') \text{ and } y' \in L(t')\} && \text{by definition of } s' \text{ and } t'\\
&= L(t') \bullet L(s') && \text{by definition of } \bullet\\
&= L(t' \bullet s') && \text{by definition of } \bullet\\
&= L(r') && \text{by definition of } r'
\end{aligned}$$

In all five cases, we have found a regular expression r' such that $L(r') = L(r)^R$. It follows that $L(r)^R$ is regular. ■

Rubric: Standard induction rubric!!