CS/ECE 374 A ✧ Fall 2023

Homework 2

Due Wednesday, September 6, 2023 at 9pm Central Time

(One day later than usual because of Labor Day)

- Starting with this homework, please start your solution to each *lettered subproblem* ((a), (b), (c), etc.) on a new page. Yes, even if the previous subproblem is only one line long. Please also remember to tell Gradescope which page(s) are relevant for which subproblems.

---

1. For each of the following languages over the alphabet $\{0, 1\}^*$, describe an equivalent regular expression, and briefly explain why your regular expression is correct. There are infinitely many correct answers for each language.

   (a) All strings in 1^*01^* whose length is a multiple of 3.

   (b) All strings that begin with the prefix 001, end with the suffix 100, and contain an odd number of 1s.

   (c) All strings that contain both 0011 and 1100 as substrings.

   (d) All strings that contain the substring 01 an odd number of times.

   (e) $\{0^a 1^b 0^c \mid a \geq 0 \text{ and } b \geq 0 \text{ and } c \geq 0 \text{ and } a \equiv b + c \pmod 2\}$.

2. For each of the following languages over the alphabet $\Sigma = \{0, 1\}$, describe a DFA that accepts the language, and briefly describe the purpose of each state. You can describe your DFA using a drawing, or using formal mathematical notation, or using a product construction; see the standard DFA rubric.

   (a) All strings in 1^*01^* whose length is a multiple of 3.

   (b) All strings that represent a multiple of 5 in base 3. For example, this language contains the string 10100, because $10100_3 = 90_{10}$ is a multiple of 5. (Yes, base 3 allows the digits 0, 1, and 2, but your input string will never contain a 2.)

   (c) All strings containing the substring 01010010. (The required substring is $p_6 = v_6$ from Homework 1.)

   (d) All strings whose ninth-to-last symbol is 0, or equivalently, the set

   $$\left\{x0z \mid x \in \Sigma^* \text{ and } z \in \Sigma^8\right\}.$$

   (e) All strings w such that $(\#(0, w) \bmod 3) + (\#(1, w) \bmod 7) = (|w| \bmod 4)$.

   [Hint: Don't try to draw the last two.]

3. ***Practice only. Do not submit solutions.***

This question asks about strings over the set of *pairs* of bits, which we will write vertically. Let Σ_2 denote the set of all bit-pairs:

$$\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

We can interpret any string w of bit-pairs as a $2 \times |w|$ matrix of bits; each row of this matrix is the binary representation of some non-negative integer, possibly with leading 0s. Let $hi(w)$ and $lo(w)$ respectively denote the *numerical values* of the top and bottom row of this matrix. For example, $hi(\varepsilon) = lo(\varepsilon) = 0$, and if

$$w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0011 \\ 0101 \end{bmatrix}$$

then $hi(w) = 3$ and $lo(w) = 5$.

(a) Describe a DFA that accepts the language $L_{+1} = \{w \in \Sigma_2^* \mid hi(w) = lo(w) + 1\}$.
For example, $w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1100 \\ 1011 \end{bmatrix} \in L_{+1}$, because $hi(w) = 12$ and $lo(w) = 11$.

(b) Describe a regular expression for L_{+1}.

(c) Describe a DFA that accepts the language $L_{\times 3} = \{w \in \Sigma_2^* \mid hi(w) = 3 \cdot lo(w)\}$.
For example, $w = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1001 \\ 0011 \end{bmatrix} \in L_3$, because $hi(w) = 9$ and $lo(w) = 3$.

(d) Describe a regular expression for $L_{\times 3}$.

*(e) Describe a DFA that accepts the language $L_{\times 3/2} = \{w \in \Sigma_2^* \mid 2 \cdot hi(w) = 3 \cdot lo(w)\}$.
For example, $w = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1001 \\ 0110 \end{bmatrix} \in L_{\times 3/2}$, because $hi(w) = 9$ and $lo(w) = 6$.

(Don't bother with the regular expression for this one.)

Solved problem

4. ***C comments*** are the set of strings over alphabet $\Sigma = \{\texttt{*}, \texttt{/}, \texttt{A}, \diamond, \hookleftarrow\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here $\hookleftarrow$ represents the newline character, $\diamond$ 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 `//` $\cdots$ $\hookleftarrow$
   - Block comments: Strings of the form `/*` $\cdots$ `*/`

   Following the C99 standard, we explicitly disallow ***nesting*** comments of the same type. A line comment starts with `//` and ends at the first $\hookleftarrow$ 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:

   `/***/` $\quad$ `//◇//◇↲` $\quad$ `/*///◇*◇↲**/` $\quad$ `/*◇//◇↲◇*/`

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

   `/*/` $\quad$ `//◇//◇↲◇↲` $\quad$ `/*◇/*◇*/◇*/`

   (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 (`"` $\cdots$ `"`) or a (multi-)character literal (`'` $\cdots$ `'`).
- 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 (`"` $\cdots$ `'` $\cdots$ `"`) 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:

$$\texttt{//}(\texttt{/} + \texttt{*} + \texttt{A} + \diamond)^*\hookleftarrow \quad + \quad \texttt{/*}\Big(\texttt{/} + \texttt{A} + \diamond + \hookleftarrow + \texttt{**}^*(\texttt{A} + \diamond + \hookleftarrow)\Big)^* \ \texttt{*}^*\texttt{*/}$$

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 $\texttt{*}$, but any run of $\texttt{*}$s must be followed by a character in $(\texttt{A} + \diamond + \hookleftarrow)$ 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:

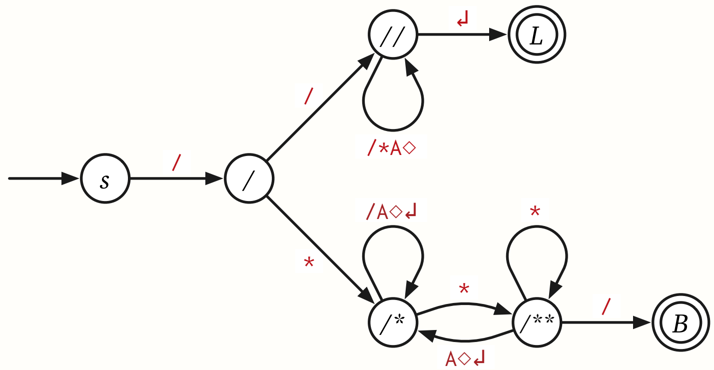
$$\Big(\diamond + \hookleftarrow \ + \ \texttt{//}(\texttt{/} + \texttt{*} + \texttt{A} + \diamond)^*\hookleftarrow + \texttt{/*}(\texttt{/} + \texttt{A} + \diamond + \hookleftarrow + \texttt{**}^*(\texttt{A} + \diamond + \hookleftarrow))^*\,\texttt{**}^*\texttt{/}\Big)^*$$

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.

(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:

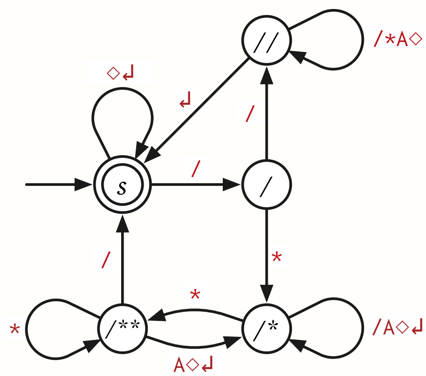
- *s* — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- *L* — We have just read a complete line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read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

*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 every regular expression s smaller than r 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 $\boldsymbol{r' = t' \bullet s'}$; 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!!