CS/ECE 374 A ✧ Fall 2021
Homework 2

Due Tuesday, September 7, 2021 at 8pm

- **Submit your written solutions electronically to Gradescope as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner).
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.

---

1. Let L be the set of all strings w in $\{\mathtt{A},\mathtt{B}\}^*$ for which $\#(\mathtt{ABBA}, w) \geq 2$. Here $\#(x, w)$ denotes the number of occurences of the substring x in the string w.

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

   (b) Describe a DFA over the alphabet $\Sigma = \{\mathtt{A},\mathtt{B}\}$ 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 be heavily penalized, even if they are perfectly correct.

   [Hint: The shortest string in L has length 7.]

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

   - The number of times the substring 01 appears in w is *not* divisible by 3[1].
   - The length of w is even.
   - The binary value of w equals $2 \pmod 3$.

   For example: The string 0101 satisfies all three conditions, so 0101 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 36 states, which is *far* 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 be heavily penalized, even if they are perfectly 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 could be as problematic as imprecision and handwaving.

[1] Recall that a is divisible by b if and only if $a \equiv 0 \pmod b$.

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.
  - 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.

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.
    - Use doubled circles to indicate accepting states A.
    - If $A = \varnothing$, you must say so explicitly.
    - If your drawing omits a junk/trash/reject state, you must 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. But you must still give an explicit description of the states Q, the start state s, and the accepting states A.
  - **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.
  - In particular, each state must have a mnemonic name.
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
  - Yes, we mean it: A perfectly correct drawing of a perfectly correct DFA with no state explanation is worth at most 6 points.
- 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.

Solved problem

3. ***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 `/`.[2] 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.)

[2]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, pretend that the characters `'`, `"`, and `\` do not 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:

$$//(/ + \star + \mathsf{A} + \diamond)^*\hookleftarrow \quad + \quad /\star\,\big(/ + \mathsf{A} + \diamond + \hookleftarrow + \star\star^*(\mathsf{A} + \diamond + \hookleftarrow)\big)^*\ \star^*\star/$$

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 $(\mathsf{A} + \diamond + \hookleftarrow)$ 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:

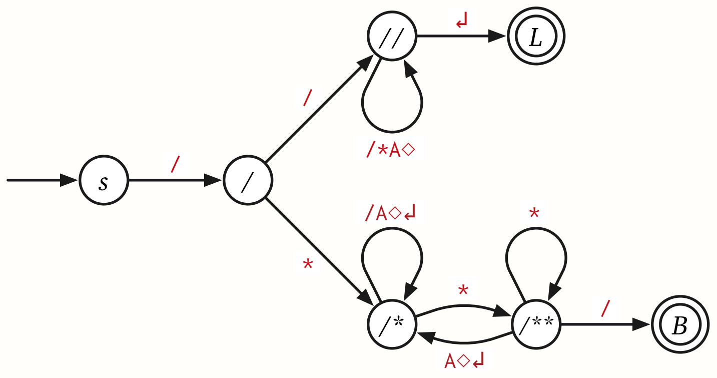
$$\big(\diamond + \hookleftarrow\ +\ //(/ + \star + \mathsf{A} + \diamond)^*\hookleftarrow + /\star(/ + \mathsf{A} + \diamond + \hookleftarrow + \star\star^*(\mathsf{A} + \diamond + \hookleftarrow))^*\ \star^*\star/\big)^*$$

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:

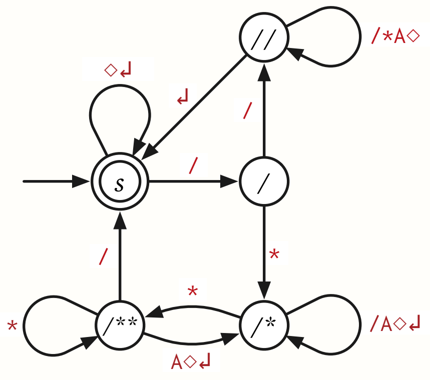
- *s* — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- *L* — We have just read a complete line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read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