CS/ECE 374 A ✧ Spring 2018

Homework 1

Due Tuesday, January 30, 2018 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 expression is correct.

   (a) All strings except `001`.

   (b) All strings that end with the suffix `001001`.

   (c) All strings that contain the substring `001`.

   (d) All strings that contain the subsequence `001`.

   (e) All strings that do not contain the substring `001`.

   (f) All strings that do not contain the subsequence `001`.

2. Let L denote the set of all strings in $\{0, 1\}^*$ that contain all four strings `00`, `01`, `10`, and `11` as substrings. For example, the strings `110011` and `01001011101001` are in L, but the strings `00111` and `1010101` are not.

   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 accept states A, and the transition function δ. Do not attempt to *draw* your DFA; the smallest DFA for this language has 20 states, which is too many for a drawing to be understandable.

   Argue that your machine accepts every string in L and nothing else, 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.

3. Let L be the set of all strings in $\{0,1\}^*$ that contain *exactly one* occurrence of the substring 010.

   (a) Give a regular expression for L, and briefly argue why your expression is correct. *[Hint: You may find the shorthand notation $A^+ = AA^*$ useful.]*

   (b) Describe a DFA over the alphabet $\Sigma = \{0,1\}$ 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. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

Solved problem

4. ***C comments*** are the set of strings over alphabet $\Sigma = \{\star, /, \mathsf{A}, \diamond, \text{↲}\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here ↲ 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 $\star$ or /.[1] There are two types of C comments:

   - Line comments: Strings of the form $//\cdots$↲
   - Block comments: Strings of the form $/\star\cdots\star/$

   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 $/\star$ and ends at the the first $\star/$ completely after the opening $/\star$; in particular, every block comment has at least two $\star$s. For example, each of the following strings is a valid C comment:

   $$/\star\star\star/ \qquad //\diamond//\diamond\text{↲} \qquad /\star///\diamond\star\diamond\text{↲}\star\star/ \qquad /\star\diamond//\diamond\text{↲}\diamond\star/$$

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

   $$/\star/ \qquad //\diamond//\diamond\text{↲}\diamond\text{↲} \qquad /\star\diamond/\star\diamond\star/\diamond\star/$$

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

   Solution:

   $$//(/ + \star + \mathsf{A} + \diamond)^*\text{↲} \quad + \quad /\star\left(/ + \mathsf{A} + \diamond + \text{↲} + \star\star^*(\mathsf{A} + \diamond + \text{↲})\right)^* \star^*\star/$$

   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 $(\mathsf{A} + \diamond + \text{↲})$ or by the closing slash of the comment. ■

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

- The opening $/\star$ 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.

Rubric: Standard regular expression rubric

(b) Describe a regular expression for the set of all strings composed entirely of blanks (⋄), newlines (↲), and C comments.

Solution:

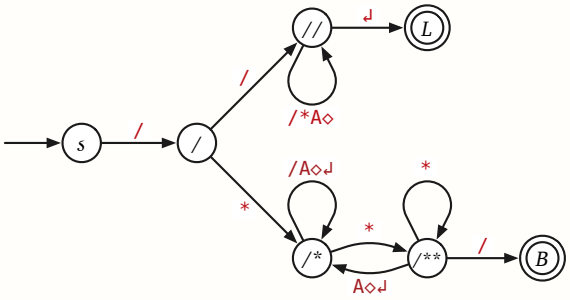
$$\big(\diamond + \text{↲} \;+\; //(/ + * + A + \diamond)^* \text{↲} + /*(/ + A + \diamond + \text{↲} + **^*(A + \diamond + \text{↲}))^* **^* / \big)^*$$

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

Rubric: Standard regular expression rubric

(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:

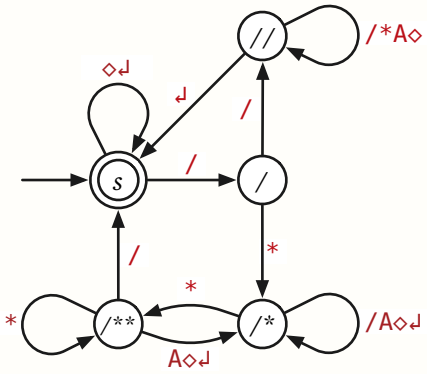
- s — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- L — We have just read a complete line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read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

■

Rubric: Standard DFA design rubric

(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

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 *transciption*; 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 longer than necessary may be penalized. Regular expressions that are *significantly* longer than necessary may get no credit at all.

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.
  - **For drawings:** Use an arrow from nowhere to indicate s, and doubled circles to indicate accepting states A. If $A = \emptyset$, 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.
  - **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 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 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.