

Solved problem

1 *C comments* are the set of strings over alphabet $\Sigma = \{*, /, A, \square, \langle\langle\text{Enter}\rangle\rangle\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here $\langle\langle\text{Enter}\rangle\rangle$ represents the newline character, \square 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 $// \dots \langle\langle\text{Enter}\rangle\rangle$.
- Block comments: Strings of the form $/* \dots */$.

Following the C99 standard, we explicitly disallow *nesting* comments of the same type. A line comment starts with $//$ and ends at the first $\langle\langle\text{Enter}\rangle\rangle$ after the opening $//$. A block comment starts with $/*$ and ends at 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:

- $/* */$
- $// \square / \square \langle\langle\text{Enter}\rangle\rangle$
- $/* // \square * \square \langle\langle\text{Enter}\rangle\rangle * * /$
- $/* \square / \square \langle\langle\text{Enter}\rangle\rangle \square * /$

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

- $/* /$
- $// \square / \square \langle\langle\text{Enter}\rangle\rangle \square \langle\langle\text{Enter}\rangle\rangle$
- $/* \square / * \square * / \square * /$

1.A. Describe a DFA that accepts the set of all C comments.

1.B. Describe a DFA that accepts the set of all strings composed entirely of blanks (\square), newlines ($\langle\langle\text{Enter}\rangle\rangle$), and C comments.

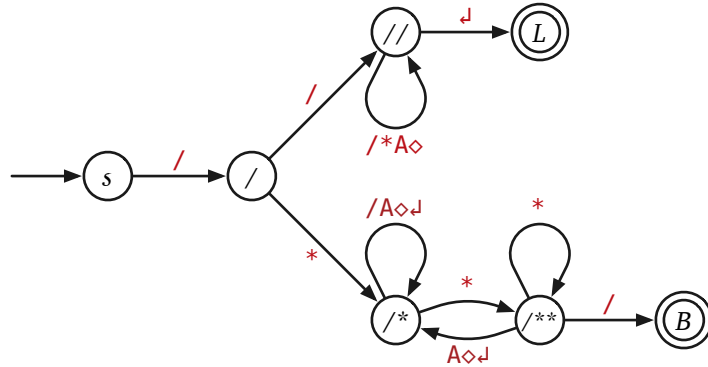
You must explain *in English* how your DFAs work. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

¹The 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 ($" \dots "$) or a (multi-)character literal ($' \dots '$).
- The opening double-quote of a string literal must not be inside a character literal ($' \dots '$) or a comment.
- The closing double-quote of a string literal must not be escaped ($\backslash "$)
- The opening single-quote of a character literal must not be inside a string literal ($" \dots ' \dots "$) or a comment.
- The closing single-quote of a character literal must not be escaped ($\backslash '$)

Solution:

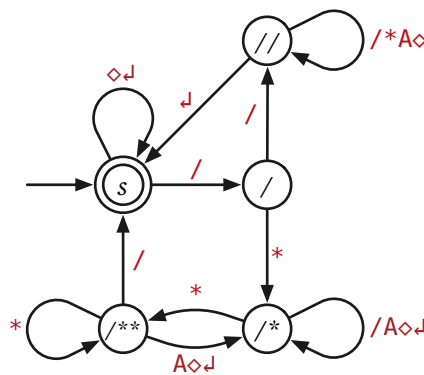
- 1.A. The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



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.

- 1.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.



- 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.

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 $A = \emptyset$, 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.
 - 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.

3 Questions

2 (100 PTS.) Regularize this [Spring, 2019].

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.

- 2.A. (20 PTS.) All strings that contain the subsequence 101.
- 2.B. (20 PTS.) All strings that do not contain the subsequence 111.
- 2.C. (20 PTS.) All strings that start in 11 and contain 110 as a substring.
- 2.D. (20 PTS.) All strings that do not contain the substring 100.
- 2.E. (20 PTS.) All strings in which every nonempty maximal substring of consecutive 0s is of length 1. For instance 1001 is not in the language while 10111 is.

3 (100 PTS.) Then, shalt thou find two runs of three [Spring, 2019].

Let L be the set of all strings in $\{0, 1\}^*$ that contain the substrings 000 and 111.

- 3.A. (60 PTS.) Describe a DFA that 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.
- 3.B. (40 PTS.) Give a regular expression for L , and briefly argue why the expression is correct.

4 (100 PTS.) Construct This [Spring, 2019]

Let L_1 and L_2 be regular languages over Σ accepted by DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$, respectively.

- 4.A. (30 PTS.)
Describe a DFA $M = (Q, \Sigma, \delta, s, A)$ in terms of M_1 and M_2 that accepts $L = L_1 \cup \overline{L_2} \cup \{\epsilon\}$. Formally specify the components Q, δ, s , and A for M in terms of the components of M_1 and M_2 .
- 4.B. (30 PTS.)
Let $H_1 \subseteq Q_1$ be the set of states q such that there exists a string $w \in \Sigma^*$ where $\delta_1^*(q, w) \in A_1$. Consider the DFA $M' = (Q_1, \Sigma, \delta_1, s_1, H_1)$. What is the language $L(M')$? Formally prove your answer!
- 4.C. (40 PTS.) Suppose that for every $q \in A_2$ and $a \in \Sigma$, we have $\delta_2(q, a) = q$. Prove that $\epsilon \in L_2$ if and only if $L_2 = \Sigma^*$.

5 (100 PTS.) Spring 2020 Q2.1

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.

- 5.A. All strings that contain 10110110 or 1101 as a substring.
- 5.B. All strings that begin with 110 and do **not** end with 0110.
- 5.C. All strings x such that the number of 0's in x is divisible by 3 and x contains 1101 as a substring.
- 5.D. All strings x such that between any two 1's in x , the number of 0's is divisible by 3. (For example, 0100010000001100 is in the language, but 010001000000101 is not.)

6 (100 PTS.) Spring 2020 Q2.2

Describe a DFA that accepts each of the following languages over the alphabet $\{0, 1\}$. Describe briefly what each state in your DFA *means*.

- 6.A. All strings that contain 101100 as a substring.
- 6.B. All strings x such that the number of 0's in x is divisible by 3 and x does **not** end in 110. [Hint: use the product construction.]
- 6.C. All strings x such that between any two 1's in x , the number of 0's is divisible by 3. (For example, 0100010000001100 is in the language, but 010001000000101 is not.)

7 (100 PTS.) Spring 2020 Q2.3

Describe a DFA that accepts each of the following languages. Describe briefly what each state in your DFA *means*. Do not attempt to draw your DFA (the number of states could be huge!). Instead, give a formal description of the states Q , the start state s , the accepting states A , and the transition function δ . Describe briefly what each state in your DFA *means*.

- 7.A. All strings in $\{0, 1, 2\}^*$ such that the number of 0's is divisible by 11, or the number of 1's is divisible by 13, or the number of 2's is divisible by 17.
- 7.B. The language L from Problem 1.2, i.e., of all strings in $\{0, 1\}^*$ that contain a balanced substring with length at least 6. (Recall that a string is *balanced* if it has the same number of 0's and 1's.) [Hint: you may use the result from Problem 1.2.]