CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020

Solved problem

- **1** *C* comments are the set of strings over alphabet $\Sigma = \{*, /, A, \Box, \ll \texttt{Enter}\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here $\ll \texttt{Enter}$ represents the newline character, \Box 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 //···· «Enter».
 - 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 \ll Enter \gg 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:

- /***/
- //□//□ ≪Enter≫
- / * ///□*□≪Enter≫ * * /
- / * □//□ ≪Enter≫ □ * /

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

- /*/
- //□//□ ≪Enter≫ □ ≪Enter≫
- /*□/*□*/□*/
- **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 (□), newlines (≪Enter≫), and C comments.

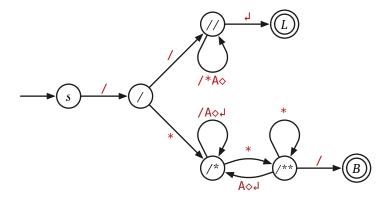
You must explain *in English* how your DFAs work. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

 $^1{\rm The}$ 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 (\backslash')

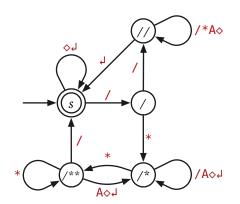
Solution:

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.

Commenting in C++ is even more complicated, thanks to the addition of *raw* string literals. Don't ask.

For example, the string $"/* \setminus "/*"/*"/*"/*$ is a valid string literal (representing the 5-character string $/* \setminus "/*/$, 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.

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 /*.

<u>*Rubric:*</u> 10 points = 5 for each part, using the standard DFA design rubric (scaled)

<u>*Rubric:*</u>[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.