CS/ECE 374 A (Spring 2022)
Past HW2 Problems with Solutions

Problem Old.2.1: *C comments* are the set of strings over alphabet $\Sigma = \{*, /, A, \square, \ll\texttt{Enter}\gg\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here $\ll\texttt{Enter}\gg$ 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 $// \cdots \ll\texttt{Enter}\gg$.
- 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 $\ll\texttt{Enter}\gg$ 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:

- $/ * * * /$
- $//\square//\square \ll\texttt{Enter}\gg$
- $/ * ///\square * \square \ll\texttt{Enter}\gg * * /$
- $/ * \square//\square \ll\texttt{Enter}\gg \square * /$

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

- $/*/$
- $//\square//\square \ll\texttt{Enter}\gg \square \ll\texttt{Enter}\gg$
- $/ * \square/ * \square * /\square * /$

(a) Describe a DFA that accepts the set of all C comments.

(b) Describe a DFA that accepts the set of all strings composed entirely of blanks ($\square$), newlines ($\ll\texttt{Enter}\gg$), and C comments.

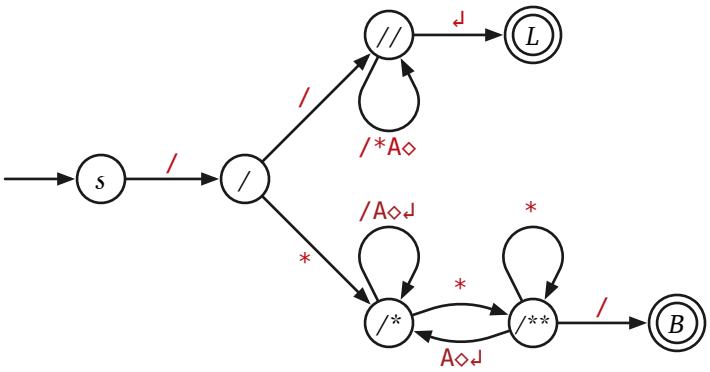
You must explain *in English* how your DFAs work. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

[1] 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 ($\backslash"$)
- 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 ($\backslash'$)
- A backslash escapes the next symbol if and only if it is not itself escaped ($\backslash\backslash$) or inside a comment.

Solution:

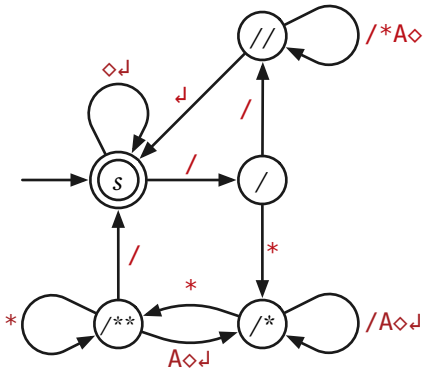
(a) The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



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.

(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.



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 /*.

Note: Generally, a DFA can be described in several ways:

- **Drawings:** Use an arrow from nowhere to indicate s, and doubled circles to indicate accepting states A. If $A = \varnothing$, 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,.
- **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
- **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.

In homeworks, we should correctly explain 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 enough.)
- DFA drawings with too many states may be penalized.
- Describing an NFA when the problem asks for a DFA will be severely penalized.

Problem Old.2.2. Give a regular expression that describes the following language over the alphabet $\{0, 1\}$, and briefly argue why your expression is correct.

- 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.

Solution: Let's start with some strings that are in the language: 11^*0. In fact, the star closure of this language is also in the language: $(11^*0)^*$. We can of course suffix this with a run of ones: $(11^*0)^*1^*$. This indeed captures all the strings in the language that starts with 1, since one can break the string after each appearance of 0, which give the above expression.

So, we only have to handle, in addition, the strings in the language that starts with 0. Such a strings can be generated only by $0(11^*0)^*1^*$. As such, the overall regular expression is

$$(\varepsilon + 0)(11^*0)^*1^*.$$

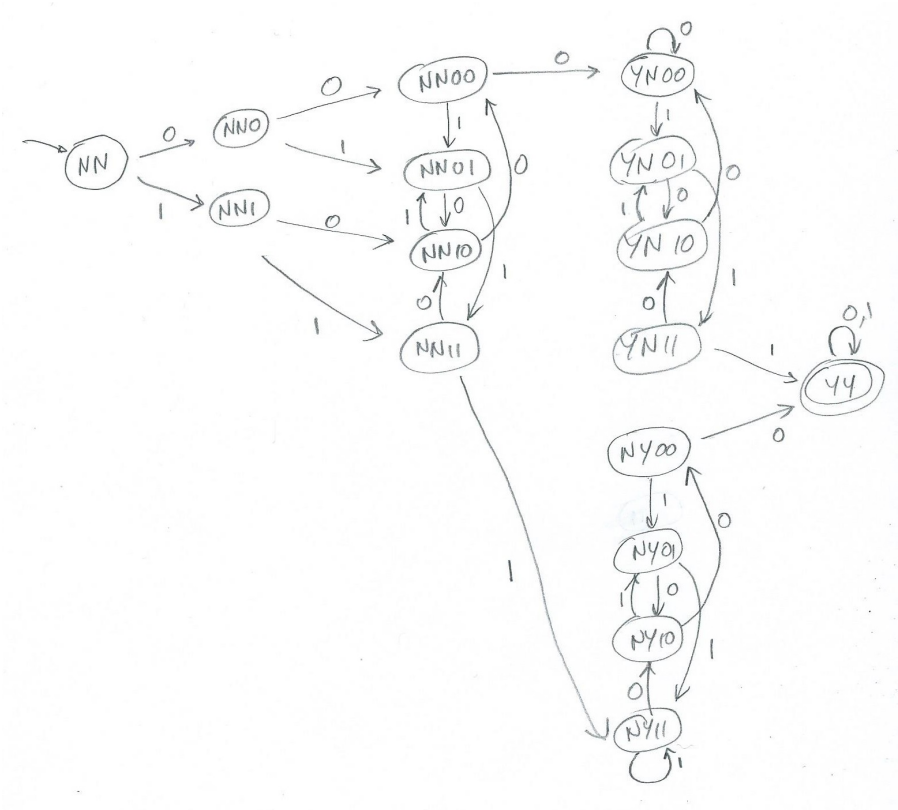
Let us prove that this is correct. Given a string w in the language, break it into strings $s_1, \ldots, s_k$, where we cut the string w after each appearance of 0 (i.e., $w = s_1 s_2 \cdots s_k$). If $s_1 = 0$, then we choose 0 in the left side, otherwise, we choose ε in the above regular expression. Similarly, if s_k contains no 0s, then we assign it to the suffix 1^*. All other strings are assigned to the generating expression 11^*0. To see the correctness, observe that every s_i is a non-empty run of 1s followed by a zero.

Note: More regular expression examples can be found in lab 1b.

Problem Old.2.3: Describe a DFA that over the alphabet $\Sigma = \{0, 1\}$ that accepts the following language:

- All strings that contain the substrings 000 and 111.

Solution: The idea is to keep track of the last two characters we have seen, and whether we have seen 000 or 111 yet. A drawing of the DFA is given below:



- The state NNx means that we have not seen 000 nor 111, and the last one or two characters we have seen are x.
- The state YNx means that we have seen 000 but have not seen 111, and the last two characters we have seen are x.

- The state NYx means that we have not seen 000 but have seen 111, and the last two characters we have seen are x.
- The state YY means that we have seen both 000 and 111.

[Note: an alternative solution can be obtained by a product construction for two DFAs with 4 states each (for the language of all strings containing 000, and the language of all strings containing 111), yielding a total of 16 states.]

Note: More DFA examples can be found in lab 2a and lab 2b.

Drawing editors: You may draw your DFAs by hand (if the drawing is neat and clear), or you may use a general-purpose drawing editor (e.g., ipe), or some drawing editor specifically for directed graphs or automata (e.g., "dot" from graphviz).

Problem Old.2.4: Describe a DFA that accepts each of the following language:

> 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.

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.

Solution: Define DFA $M = (Q, \Sigma, s, \delta, A)$ where $\Sigma = \{0,1,2\}$, and

$$\begin{aligned} Q &= \{(i,j,k) : \ 0 \le i < 11, \ 0 \le j < 13, \ 0 \le j < 17\} \\ s &= (0,0,0) \\ A &= \{(i,j,k) \in Q : i = 0 \text{ or } j = 0 \text{ or } k = 0\}. \end{aligned}$$

(The number of states is $11 \cdot 13 \cdot 17 = 2431$.)

The transition function $\delta : Q \times \Sigma \to Q$ is defined as follows: for all $(i,j,k) \in Q$,

$$\begin{aligned} \delta((i,j,k),0) &= ((i+1) \bmod 11, \ j, \ k) \\ \delta((i,j,k),1) &= (i, \ (j+1) \bmod 13, \ k) \\ \delta((i,j,k),2) &= (i, \ j, \ (k+1) \bmod 17). \end{aligned}$$

Explanation: The state (i,j,k) means that the number of 0's seen so far is $i \pmod{11}$, and the number of 1's is $j \pmod{13}$ and the number of 2's is $k \pmod{17}$.