

CS3342 – Assignment 1
due Feb. 10, 2022
2-day no-penalty extension until: Feb. 12, 11:55pm
(SRA's cannot be used to extend further)

1. (20pt) We have discussed a simple calculator (textbook p.54, slide 2a.16), where the C-style comments are described using this regular expressions (`\n` is *newline*):

$$\begin{aligned} \text{comment} \quad \longrightarrow \quad & /* (non-* | * non-/*)^* */ \\ & | // (non-\n)^*\n \end{aligned}$$

Actually, the regular expression above for C comments is wrong! (Always think for yourself; there can be errors and incorrect information anywhere!) You are required to:

- (a) (10pt) Prove there is an error; that means the regular expression either accepts a string which is not a comment, or fails to accept a valid comment. You need to give an example of such a string. Explain why your example is correct.
 - (b) (10pt) Give a valid regular expression. This should be a nice, intuitive looking regular expression, not something horrible produced by jflap. (You can use jflap if you want, but the result must look readable.) You are allowed to use *non- x* (or $\neq x$, or $[\sim x]$) to denote any character different from x ; x can also be a set of characters. Explain why your regular expression is correct.
2. (20pt) Keywords fit the definition of identifiers and scanners identify them as such, and then look them up a table of keywords, because otherwise the deterministic finite automaton is unnecessarily larger. This question addresses the size of this DFA.

Assume that the identifiers and keywords are defined as follows:

$$\begin{aligned} \text{identifier} \quad \longrightarrow \quad & (_ | \text{letter})(_ | \text{letter} | \text{digit})^* \\ \text{letter} \quad \longrightarrow \quad & \text{a} | \text{b} | \dots | \text{z} \\ \text{digit} \quad \longrightarrow \quad & 0 | 1 | \dots | 9 \end{aligned}$$

- (a) (10pt) Draw a deterministic finite automaton that recognizes the following keywords directly:

$$\text{keyword} \quad \longrightarrow \quad \text{this} | \text{throw} | \text{throws} | \text{try}$$

Each keyword will be identified in a separate accepting state, which is different from any state recognizing identifiers. Label each accepting state accordingly. You are allowed to use *non- x* (or $\neq x$, or $[\sim x]$) to denote any character different from x ; x can also be a set of characters.

- (b) (10pt) Assuming the definition for *identifier* stays the same as the one above and that all keywords belong to letter^+ , what is the maximum number of states this DFA can have for a language with 40 keywords? Explain your answer. (*Hint*: the maximum number of states depends on the lengths of the keywords.)
3. (20pt) The driver for a table-driven scanner in Fig. 2.11 (textbook p.66, slide 2a.28) is built to handle the case when $t_1 <_p w <_p t_2$, for some tokens t_1, t_2 and a non-token string w , where the relation $<_p$ indicates a proper prefix. Consider the table-driven scanner in Fig. 2.12 (textbook p.67, slide 2a.29).
- (a) (10pt) Identify such a situation $t_1 <_p w <_p t_2$ in the DFA associated with this scanner. Indicate what strings t_1, t_2 , and w are, as well as the token types for t_1, t_2 .

- (b) (10pt) Give an example of an error that causes the above scanner to unread more than two characters in line 9 from bottom: `unread remembered_chars`. Indicate the string causing the error and the unread characters.
4. (40pt) Consider the following unambiguous grammar, G , for the dangling else problem:

1. $program \rightarrow stmt \$\$$
2. $stmt \rightarrow balanced_stmt$
3. $stmt \rightarrow unbalanced_stmt$
4. $balanced_stmt \rightarrow \text{if } cond \text{ then } balanced_stmt \text{ else } balanced_stmt$
5. $balanced_stmt \rightarrow other_stmt$
6. $unbalanced_stmt \rightarrow \text{if } cond \text{ then } stmt$
7. $unbalanced_stmt \rightarrow \text{if } cond \text{ then } balanced_stmt \text{ else } unbalanced_stmt$
8. $cond \rightarrow c_i, i \geq 1$
9. $other_stmt \rightarrow s_i, i \geq 1$

Nonterminals: $\{program, stmt, balanced_stmt, unbalanced_stmt, other_stmt, cond\}$

Terminals: $\{\text{if}, \text{then}, \text{else}, c_i, s_i, \$\$ \}$

Starting nonterminal: $program$

- (a) (5pt) Show the parse tree of G for the input:

`if c1 then if c2 then s1 else if c3 then s2 $$.`

- (b) (10pt) Compute all sets $FIRST(X)$, $FOLLOW(X)$, for all nonterminals X , as shown in the example in Figure 2.23 (textbook p.87, slide 2b.21). The algorithm in Fig. 2.24 (textbook p.88, slides 2b.19-20) for computing these is adding symbols in four steps, three of which are of interest for our question (the 1..3 labels are added here for future reference):

1. add $FIRST(Y_i)$ to $FIRST(X)$
2. add $string_FIRST(\beta)$ to $FOLLOW(B)$
3. add $FOLLOW(A)$ to $FOLLOW(B)$

Each of these steps uses a production to add symbols to a set. For each symbol added to any of the sets you computed, indicate the step and the production used: $(step, prod)$, $1 \leq step \leq 3$, $1 \leq prod \leq 9$. The meaning is that production $prod$ was used to add the symbol at step $step$. If the same symbol is added multiple times, give the $(step, prod)$ for the first time it is added.

- (c) (5pt) Compute all sets $PREDICT(p)$, for all productions p , as shown in the example in Figure 2.23 (textbook p.87, slide 2b.21).
- (d) (5pt) Prove that G is not LL(1).
- (e) (10pt) Show how you can employ on G the techniques we used for attempting to make a grammar LL(1).
- (f) (5pt) Is the new grammar LL(1)? Prove your answer.

READ ME! Submit your answers as a *single pdf file* in OWL. Solutions should be typed but readable (by others!) hand-written solutions are acceptable. Source code, if required, is submitted as separate files.

JFLAP: You are allowed to use JFLAP to help you solve the assignment. Make sure you understand what it does; JFLAP will not be available during in-person exams!

L^AT_EX: For those interested, the best program for scientific writing is L^AT_EX. It is far superior to all the other programs, it is free, and you can start using it in minutes; here is an introduction: <https://tobi.oetiker.ch/lshort/lshort.pdf>