HW 1
50 points
Posted Tuesday, September 2
Due Thursday, September 18 at midnight

Note: Lecture 2 covers material for 1-4 (except for 3(e)). Lectures 3 and 4 cover 5 and 6. Submit solutions in file `HW1Solutions.pdf`. Submission size limit in Submitty is set to 2.5MB.

Problem 1. Describe the languages denoted by the following regular expressions using the highest level characterization possible.

(a) (0.5pts) `a(a|b)*a`
(b) (0.5pts) `((ϵ|a)b*)*`
(c) (0.5pts) `(a|b)*a(a|b)(a|b)`
(d) (0.5pts) `a*ba*ba*`
(e) (3pts) `(aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*((ab|ba)(aa|bb)*a|b)`

Problem 2. The following grammar generates binary strings that have *even* values. S is the start symbol.

$S \to A$
$A \to B$ `0` $\mid A$ `0` $\mid$ `0`
$B \to A$ `1` $\mid B$ `1` $\mid$ `1`

Your task is to construct a similar grammar that generates strings with values divisible by 3.

(a) (3pts) Fill in the blanks to complete the grammar:
$S \to A$
$A \to A$ `0` $\mid \ldots$
$B \to ...$
...
(b) (1pts) Using your grammar, show a derivation that generates 21 in binary notation.
(c) (4pts) Prove that all binary strings generated by your grammar have values divisible by 3. (*Note: Use induction on the number of nodes in the parse tree.*)
(d) (4pts) Prove that your grammar generates all binary strings with values divisible by 3.

Problem 3. Consider the grammar

(1) $S \to$ `a` S `b` S
(2) $S \to$ `b` S `a` S
(3) $S \to \epsilon$

(a) (1pts) Show that the grammar is ambiguous by constructing two different leftmost derivations for a string. Use the shortest string possible.
(b) (1pts) Construct the corresponding rightmost derivations for the string from (a).
(c) (2pts) Construct the corresponding parse trees.
(d) (3pts) One can see that the grammar generates strings with equal number of `a`'s and `b`'s. Does it generate *all* such strings? (*Note: You do not need to write a full formal proof. If*

you answer YES, outline an inductive argument. If you answer NO, show a string with equal number of `a`*'s and* `b`*'s that cannot be generated by the grammar.*)

(e) (5pts) Finally, write a recursive descent parser with backtracking in Python. Include function `dfparse(s:str) -> list[int]` that takes a string and returns the list of productions the parser applied (if the parse succeeded). An unusual requirement is that your backtracking parser tries the S productions in reverse order: it first tries (3), then (2), then (1). Include the function in file `backtrack.py` and turn in Submitty. You may assume that we'll test with string inputs only. A few sample runs are included:

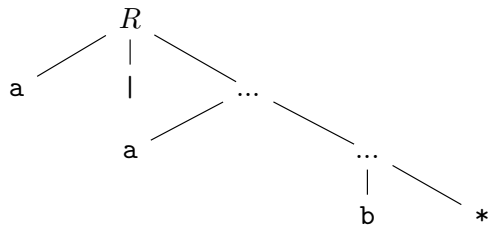
```
python3 -i backtrack.py
>>> dfparse('')
[3]

python3 -i backtrack.py
>>> dfparse('aba')
[]

python3 -i backtrack.py
>>> dfparse('abab')
[1, 3, 1, 3, 3]
```

Problem 4. The following is an ambiguous grammar that generates regular expressions over symbols `a` and `b`:

$$R \rightarrow R \texttt{|} R \;\mid\; R\,R \;\mid\; R^{*} \;\mid\; \texttt{(}\,R\,\texttt{)} \;\mid\; \texttt{a} \;\mid\; \texttt{b}$$

(a) (2pts) How many parse trees are there for regular expression `a|ab`*? *Note: you do not need to draw the trees, just write the answer.*

Disambiguate the grammar so that `a|ab`* gives rise to a parse tree with this shape:



(b) (3pts) Disambiguate into a *left recursive* grammar.

(c) (3pts) Disambiguate into a *right recursive* grammar.

Problem 5. Consider the following LL(1) grammar that generates S-expressions, an essential structure in Lisp-like languages. S and Ss are the nonterminals, `num` (number), `sym` (symbol), `(`

and `)` are the terminals, i.e., the tokens.

$$
\begin{array}{lcl}
\mathit{Start} & \rightarrow & S\$\$ \\
S & \rightarrow & \texttt{num} \\
S & \rightarrow & \texttt{sym} \\
S & \rightarrow & \texttt{(}\ Ss\ \texttt{)} \\
Ss & \rightarrow & S\ Ss \\
Ss & \rightarrow & \epsilon
\end{array}
$$

(a) (2.5pts) What is FOLLOW(Ss)? FOLLOW(S)? PREDICT($Ss \rightarrow \epsilon$)?
(b) (2.5pts) Draw a parse tree for the string `( 5 ( a ) ) $$`.
(c) (3pts) Consider a recursive descent parser running on the same input. At the point where token `a` is matched, which recursive descent routines will be active (i.e., what routines will have a frame on the run-time stack)?

Problem 6. For each grammar below, determine if it is LL(1) or not and if it is ambiguous or not. You do not need to justify your answer, just write YES or NO. As an example of the format for answers we are looking for: (x) LL(1): YES, Ambiguous: YES.

(a) (1pts) $A \rightarrow 0\ A\ 1 \mid 0\ 1$
(b) (1pts) $A \rightarrow +\ A\ A \mid *\ A\ A \mid \texttt{a}$
(c) (1pts) $A \rightarrow A\ (\ A\ )\ A \mid \epsilon$
(d) (1pts) $A \rightarrow B\ \texttt{a} \mid \texttt{b}\ B\ \texttt{c} \mid \texttt{d c} \mid \texttt{b d c} \qquad B \rightarrow \texttt{d}$
(e) (1pts) $S \rightarrow C\texttt{a}C\texttt{b} \mid D\texttt{b}D\texttt{a} \qquad C \rightarrow \epsilon \qquad D \rightarrow \epsilon$