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)`

(a) All strings of a's and b's that start and end with an a. Ex: $\{aa, aba, aaaa, abba, ...\}$
(b) All possible strings of a's and b's including the empty string. Ex: $\{\epsilon, a, b, ab, ba, aa, bb, aaa, ...\}$
(c) All strings of a's and b's whose third to last character is a 'a'. Ex: $\{aaaa, aaab, aaba, aabb, baaa, baab...\}$
(d) All strings of a's and b's that contain exactly two b's. The two b's can be right next to each other or separated by any number of a's. Ex: $\{bb, abb, bab, bba, abab, ababa, ...\}$
(e) All strings of a's and b's that contains an even number of a's and an odd number of b's. Ex: $\{b, aab, abb, aba, aaabb, ababb, ...\}$

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

$S \to A$
$A \to B\ \mathtt{0} \mid A\ \mathtt{0} \mid \mathtt{0}$
$B \to A\ \mathtt{1} \mid B\ \mathtt{1} \mid \mathtt{1}$

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:

$S \to A$
$A \to A\ \mathtt{0} \mid B\ \mathtt{1} \mid \mathtt{0}$
$B \to C\ \mathtt{0} \mid A\ \mathtt{1} \mid \mathtt{1}$
$C \to B\ \mathtt{0} \mid C\ \mathtt{1}$

(b) (1pts) Using your grammar, show a derivation that generates 21 in binary notation.

21 in binary is 10101. One possible derivation is:

$S \Rightarrow A$
$\Rightarrow B1$ ($A \to B1$)
$\Rightarrow C01$ ($B \to C0$)
$\Rightarrow C101$ ($C \to C1$)
$\Rightarrow B0101$
$\Rightarrow 10101$

(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.*)

Induction Hypothesis: Assume that for any terminal string w,

1. If w is derived from A, then val(w) ($\equiv 0 \mod 3$)
2. If w is derived from B, then val(w) ($\equiv 1 \mod 3$)
3. If w is derived from C, then val(w) ($\equiv 2 \mod 3$)

Where val(w) is the integer value of the binary string w.

Base Case: Length of w ($|w|$)= 1. The only derivable strings of length 1 from our grammer is

1. $w = 0$ from A val(w) $= 0 \equiv 0 \mod 3$, holds true with our hypothesis.
2. $w = 1$ from B. val(w) $= 1 \equiv 1 \mod 3$, holds true with our hypothesis.
3. No string of length 1 can be derived from C. which still holds true with our hypothesis.

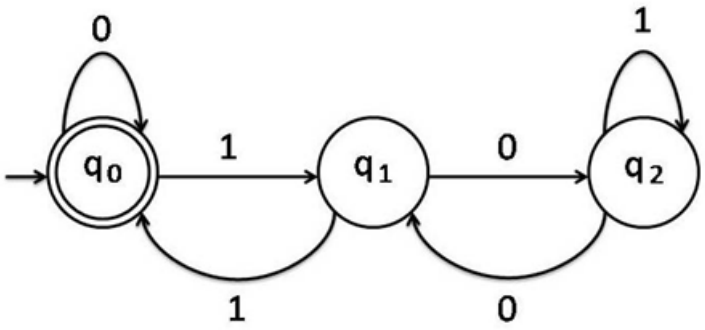
Inductive Step: Assumiming that the three statements of our induction hypothesis holds for all terminal strings of length $\leq$ k. Let w be the terminal string of length k + 1 derived from A, B or C. We consider each production used to get k + 1 length of w and check if each holds true to the induction hypothesis.

1. For A $\rightarrow A0$. Suppose A derives string x of length k + 1. Appending 0 gives string $x0$ whose value is val(x0) = 2*val(x). By our inductive hypothesis, val(x) $\equiv$ 0. Therefore, val(x0) $\equiv 2 * 0 \equiv 0 \mod 3$. Holds true.
2. For A $\rightarrow B1$. Suppose B derives string y of length k + 1. Appending 1 gives string $y1$ whose value is val(y1) = 2*val(y) + 1. By our inductive hypothesis, val(y) $\equiv$ 1. Therefore, val(y1) $\equiv 2 * 1 + 1 \equiv 3 \equiv 0 \mod 3$. Holds true.
3. For A $\rightarrow$ 0. This is our base case which we have already shown to hold true.
4. For B $\rightarrow C0$. Suppose C derives string z of length k + 1. Appending 0 gives string $z0$ whose value is val(z0) = 2*val(z). By our inductive hypothesis, val(z) $\equiv$ 2. Therefore, val(z0) $\equiv 2 * 2 \equiv 4 \equiv 1 \mod 3$. Holds true.
5. For B $\rightarrow A1$. Suppose A derives string x of length k + 1. Appending 1 gives string $x1$ whose value is val(x1) = 2*val(x) + 1. By our inductive hypothesis, val(x) $\equiv$ 0. Therefore, val(x1) $\equiv 2 * 0 + 1 \equiv 1 \mod 3$. Holds true.
6. For B $\rightarrow$ 1. This is our base case which we have already shown to hold true.
7. For C $\rightarrow B0$. Suppose B derives string y of length k + 1. Appending 0 gives string $y0$ whose value is val(y0) = 2*val(y). By our inductive hypothesis, val(y) $\equiv$ 1. Therefore, val(y0) $\equiv 2 * 1 \equiv 2 \mod 3$. Holds true.
8. For C $\rightarrow C1$. Suppose C derives string z of length k + 1. Appending 1 gives string $z1$ whose value is val(z1) = 2*val(z) + 1. By our inductive hypothesis, val(z) $\equiv$ 2. Therefore, val(z1) $\equiv 2 * 2 + 1 \equiv 5 \equiv 2 \mod 3$. Holds true.

And thus, by induction, we have show that our induction hypothesis holds true for all terminal strings derived from each of our productions from our grammer for lengths k + 1.

(d) (4pts) Prove that your grammar generates all binary strings with values divisible by 3.

Let w be all binary strings with $val(w)$ divisible by 3. If we look at the DFA for remainder mod 3 while reading w from left to right. Each spot over is mirrored by a production in the grammer, giving us a derivation of w from S, our start point. Hence, the grammer generates all binary strings with values divisible by 3.



DFA mod 3 = 0 (divisible by 3)

Problem 3. Consider the grammar

(1) $S \to \texttt{a}\ S\ \texttt{b}\ S$
(2) $S \to \texttt{b}\ S\ \texttt{a}\ S$
(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.

1st Derivation:

$$\begin{aligned} S &\Rightarrow aSbS \\ &\Rightarrow abSaSbS \\ &\Rightarrow ab\epsilon aSbS \\ &\Rightarrow abaSbS \\ &\Rightarrow aba\epsilon bS \\ &\Rightarrow ababS \\ &\Rightarrow abab\epsilon \\ &\Rightarrow abab \end{aligned}$$

2nd Derivation:

$$\begin{aligned} S &\Rightarrow aSbS \\ &\Rightarrow a\epsilon bS \\ &\Rightarrow abS \\ &\Rightarrow abaSbS \\ &\Rightarrow aba\epsilon bS \\ &\Rightarrow ababS \\ &\Rightarrow abab\epsilon \\ &\Rightarrow abab \end{aligned}$$

Constructed the string "abab" in in two different leftmost derivations, showing that the grammer is ambiguous.

(b) (1pts) Construct the corresponding rightmost derivations for the string from (a).

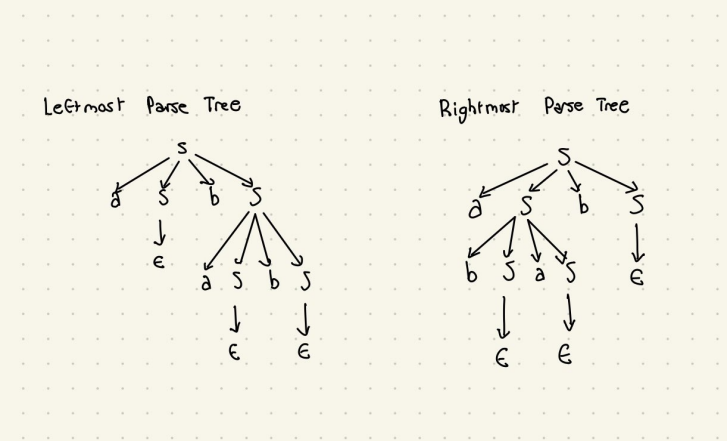
1st Derivation:

$$\begin{aligned} S &\Rightarrow aSbS \\ &\Rightarrow aSb\epsilon \\ &\Rightarrow aSb \\ &\Rightarrow abSaSb \\ &\Rightarrow abSa\epsilon b \\ &\Rightarrow abSab \\ &\Rightarrow ab\epsilon ab \\ &\Rightarrow abab \end{aligned}$$

2nd Derivation:

$S \Rightarrow aSbS$
$\Rightarrow aSbaSbS$
$\Rightarrow aSbaSb\epsilon$
$\Rightarrow aSbaSb$
$\Rightarrow aSba\epsilon b$
$\Rightarrow aSbab$
$\Rightarrow a\epsilon bab$
$\Rightarrow abab$

(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.*)

Yes, the grammar does generate all strings with an equal number of a's and b's.

We can prove this by induction.

It'll look something like this:

Induction Hypothesis: Assume that every string with exactly k a's and k b's for some k $\geq$ 0 can be generated by the grammer.

Base Case: For n = 0, the empty string ϵ can be generated by rule 3. 0 a's and 0 b's. Holds true to our hypothesis.

Inductive Step: Let w be any nonempty string with n = k + 1 a's and b's, check each production of our grammer to make sure our hypothesis holds true for 2(k + 1) length string generated by our production holds true to our induction hypothesis.

1. For production 1, $S \rightarrow aSbS$. Any string w generated by this production must start with an 'a' and have a corresponding 'b' that matches it. All other productions of S will follow in the same manner, with S only either able to transform into bSaS or ϵ. By our inductive hypothesis, all strings generated by S will have an equal number of a's

and b's. Hence, any string w generated by this production will have an equal number of a's and b's.

2. For production 2, $S \rightarrow bSaS$. Any string w generated by this production must start with an 'b' and have a corresponding 'a' that matches it. Same case here.

3. For production 3, $S \rightarrow \epsilon$. This is our base case which we have already shown to hold true.

(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]
```

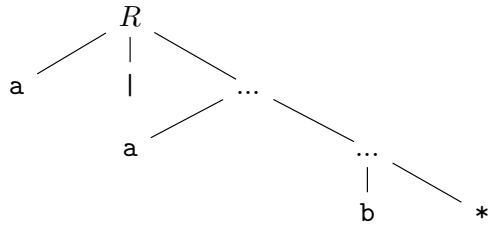
Problem 4. The following is an ambiguous grammar that generates regular expressions over symbols `a` and `b`:

$$R \to R|R \mid R\ R \mid R^* \mid (\ R\ ) \mid \texttt{a} \mid \texttt{b}$$

(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.*

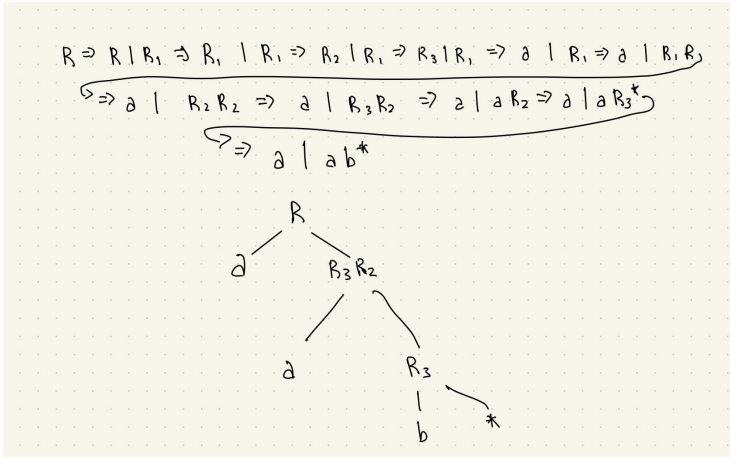
There are **5** parse trees for the regular expression `a|ab`*.

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



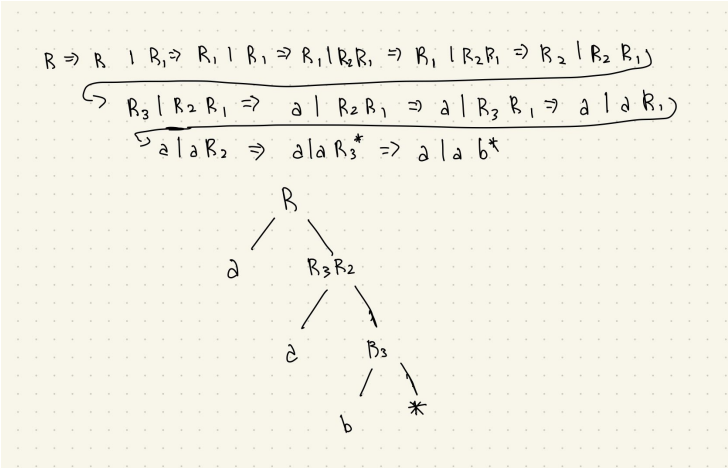
(b) (3pts) Disambiguate into a *left recursive* grammar.

$R \to R \mid R_1$
$R_1 \to R_1 R_2 \mid R_2$ aroun $R_2 \to R_3^* \mid R_3$
$R_3 \to (R) \mid a \mid b$



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

$R \to R_1 \mid R \mid R_1$
$R_1 \to R_2 R_1 \mid R_2$
$R_2 \to R_3 \mid R_3^*$
$R_3 \to a \mid b \mid (R)$



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.

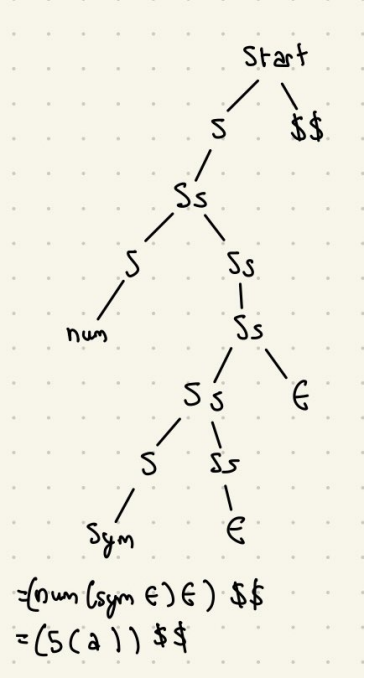
$$
\begin{array}{lcl}
Start & \rightarrow & S \texttt{ \$\$} \\
S & \rightarrow & \texttt{num} \\
S & \rightarrow & \texttt{sym} \\
S & \rightarrow & \texttt{(}\ Ss\ \texttt{)} \\
Ss & \rightarrow & S\ Ss \\
Ss & \rightarrow & \epsilon
\end{array}
$$

(a) (2.5pts) What is FOLLOW(Ss)? FOLLOW(S)? PREDICT($Ss \rightarrow \epsilon$)?

FOLLOW(Ss) = {)}
FOLLOW(S) = $\{num, sym, (,), \$\$\}$
PREDICT($Ss \rightarrow \epsilon$) = {)}

(b) (2.5pts) Draw a parse tree for the string `( 5 ( a ) ) $$`.

$num = 5, sym = a$



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

(a) LL(1): NO, Ambiguous: NO
(b) LL(1): YES, Ambiguous: NO
(c) LL(1): NO, Ambiguous: YES
(d) LL(1): NO, Ambiguous: YES
(e) LL(1): YES, Ambiguous: NO