CMPT 379 - Summer 2019 - Midterm

(1) The following context-free grammar (CFG) G describes the set of valid regular expression strings:

$$R \rightarrow R \text{ `|' } R \mid R\,R \mid R \text{ `*' } \mid \text{`(' } R \text{ `)' } \mid a \mid b$$

a. (2pts) For each of the following regular expressions write down if they are valid or invalid.

1. `a(b***)`
2. `((ab)|)*`
3. `((a|b|a*)*|a)*`
4. `(**)`

Answer:

1. `a(b***)` - valid
2. `((ab)|)*` - invalid
3. `((a|b|a*)*|a)*` - valid
4. `(**)` - invalid

b. (4pts) Consider the following grammar G':

$$\begin{aligned} R &\rightarrow R \text{ `|' } R_1 \\ R_1 &\rightarrow R_1\,R_2 \\ R_2 &\rightarrow R_3 \text{ `*'} \end{aligned}$$

This CFG G' is supposed to be the unambiguous version of the original regular expression CFG G but it is missing some CFG rules (for example, the rule(s) for R_3 are missing).

Write down all the missing rules in CFG G' to provide a grammar that is the unambiguous version of G. Do not add any new non-terminals to the G' grammar and the alphabet is the same as CFG G.

Your grammar should resolve any ambiguity by assuming that Kleene closure, **`*'** has the highest precedence priority, followed by concatenation, $RR$, followed by alternation, **`|'** . Also assume that each operation associates to the left, e.g. RRR should be treated as $(RR)R$ and $R|R|R$ should be treated as $(R|R)|R$.

Answer:

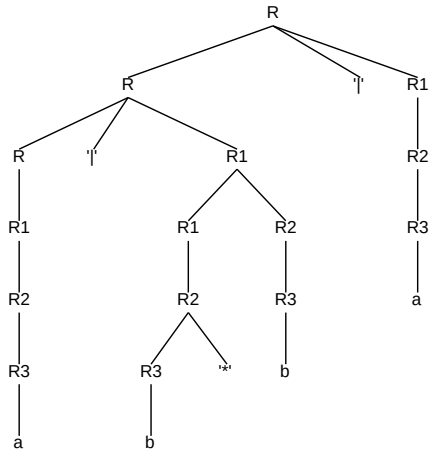
$$\begin{aligned} R &\rightarrow R \text{ `|' } R_1 \mid R_1 \\ R_1 &\rightarrow R_1\,R_2 \mid R_2 \\ R_2 &\rightarrow R_2 \text{ `*' } \mid R_3 \\ R_3 &\rightarrow \text{`(' } R \text{ `)' } \mid a \mid b \end{aligned}$$

Modifying the rule $R_2 \rightarrow R_3$ **`*'** to account for strings like `a**` is optional.

c. (4pts) Use your unambiguous grammar to parse the string `a|b*b|a` and provide the parse tree.

Answer:

```
R
R|R1
R|R1|R1
R1|R1|R1
R2|R1|R1
R3|R1|R1
a|R1|R1
a|R1 R2|R1
a|R2 R2|R1
a|R3 * R2|R1
a|b* R2|R1
a|b* R3|R1
a|b*b|R1
a|b*b|R2
a|b*b|R3
a|b*b|a
```



(2) In the following grammar G a new start symbol S' has been added:

$$S' \rightarrow S$$
$$S \rightarrow (S) \mid \epsilon$$

a. (3pts) Is G an LR(0) grammar? If yes, provide the parsing table. If no, provide all the conflicts.

Answer: No. In two item sets, for closure of $S' \rightarrow \bullet S$ and closure of $S \rightarrow (\bullet S)$ there is a shift-reduce conflict between the shift in the dotted rule $S \rightarrow \bullet(S)$ and the reduce of $S \rightarrow \epsilon\bullet$

b. (3pts) Is G an SLR(1) grammar? If yes, explain using FOLLOW sets how the shift-reduce or reduce-reduce conflicts were resolved.

Answer: Yes. FOLLOW(S) = $\{), \$\}$ which is disjoint from the set $\{($ and so we can shift on (in the dotted rule $S \rightarrow \bullet(S)$ and reduce $S \rightarrow \epsilon\bullet$ using lookahead symbols in FOLLOW(S) = $\{), \$\}$

c. (2pts) Is the grammar G SLR(1) if we *add* the rule $S \rightarrow S\ S$ to the grammar. Explain why using a brief sentence.

Answer: Grammar G is no longer SLR(1). In the new grammar, FOLLOW(S) = $\{(,), \$\}$ and so there is a shift-reduce conflict with the shift on $S \rightarrow \bullet(S)$ and reduce of the dotted rule $S \rightarrow \epsilon\bullet$. The grammar also becomes ambiguous which can be shown via two leftmost derivation of the input string (): $S \Rightarrow SS \Rightarrow \epsilon S \Rightarrow \epsilon(S) \Rightarrow \epsilon(\epsilon)$ and $S \Rightarrow SS \Rightarrow ()S \Rightarrow (S)\epsilon \Rightarrow (\epsilon)\epsilon$

d. (2pts) Using the following yacc implementation of the grammar G provide the value printed using the `printf` command for input string `((()))`. Assume the type of non-terminal S is an integer.

```
S': S          { printf("%d\n", $1); }
S : '(' S ')'  { $$ = $2+2; }
  |            { $$ = 0; }
```

Answer: 6