Questions reproduced from textbook: *Programming Language Pragmatics, 4th Edition, Michael Scott, Morgan Kaufman, 2016*. Chapter 2.

Study Homework 1: regular expressions, bottom up parsing.
Answers provided.

2.1 Write regular expressions to capture the following.

(d) Floating-point constants in Ada.These match the definition of *real* in Example 2.3, except that (1) a digit is required on both sides of the decimal point, (2) an underscore is permitted between digits, and (3) an alternative numeric base may be specified by surrounding the nonexponent part of the number with pound signs, preceded by a base in decimal (e.g., 16#6.a7#e+2). In this latter case, the letters a . . f (both upper- and lowercase) are permitted as dig- its. Use of these letters in an inappropriate (e.g., decimal) number is an error, but need not be caught by the scanner.

Answer:
*Ada_int → digit ((_ | ε) digit)**
Extended_digit → digit | a | b | c | d | e | f | A | B | C | D | E | F
*Ada_extended_int → extended_digit ((_ | ε) extended_digit)**
AdaFP_num → ((Ada_int ((. Ada_int | ε))
| (Ada_int # Ada_extended_int
((. Ada_extended_int) | ε) #))
(((e | E) (+ | - | ε) Ada_int) | ε)

(f) Financial quantities in American notation. These have a leading dollar sign ($), an optional string of asterisks (*—used on checks to discourage fraud), a string of decimal digits, and an optional fractional part consisting of a dec- imal point (.) and two decimal digits. The string of digits to the left of the decimal point may consist of a single zero (0). Otherwise it must not start with a zero. If there are more than three digits to the left of the decimal point, groups of three (counting from the right) must be separated by commas (,). Example: $**2,345.67. (Feel free to use "productions" to define abbreviations, so long as the language remains regular.)

Answer:

nzdigit → 1 | 2 | 3 | 4 | 5 | 4 | 7 | 8 | 9
digit → 0 | nzdigit
group → , digit digit digit
*number → $ * * (0 | nzdigit (ε | digit | digit digit) group*) (ε | . digit digit)*

(note in last rule the * after the $ is a terminal symbol; the 2nd * is the Kleene star)

2.14 Consider the language consisting of all strings of properly balanced parentheses and brackets.

(a) Give LL(1) and SLR(1) grammar for this language.

Answer:

LL(1) grammar:

1. $P \longrightarrow S$ \$\$
2. $S \longrightarrow (S) S$
3. $S \longrightarrow [S] S$
4. $S \longrightarrow \epsilon$

SLR(1) grammar:

1. $P \longrightarrow S$ \$\$
2. $S \longrightarrow S (S)$
3. $S \longrightarrow S [S]$
4. $S \longrightarrow \epsilon$

(b) Give the corresponding LL(1) and SLR(1) parsing tables.

Answer:

LL(1) parse table:

Top-of-stack nonterminal	Current input token				
	(	)	[	]	\$\$
P	1	–	1	–	1
S	2	4	3	4	4

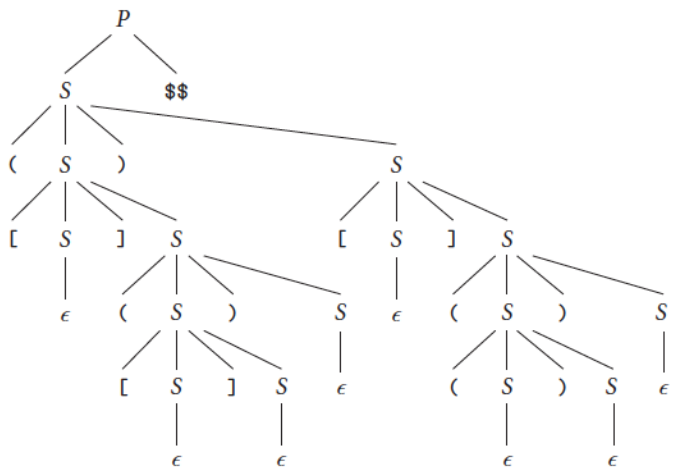
SLR(1) parse table:

Top-of-stack state	Current input symbol					
	S	(	)	[	]	\$\$
0	s1	r4	r4	r4	r4	r4
1	–	s2	–	s3	–	b1
2	s4	r4	r4	r4	r4	r4
3	s5	r4	r4	r4	r4	r4
4	–	s2	b2	s3	–	–
5	–	s2	–	s3	b3	–

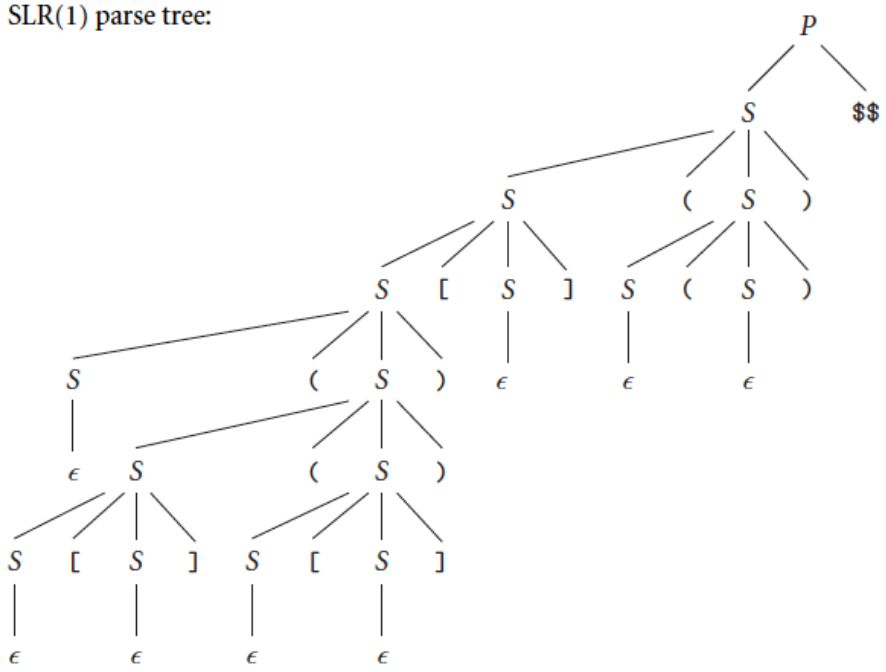
(c) For each grammar, show the parse tree for ([]([]))[](()).

Answer:

LL(1) parse tree:



SLR(1) parse tree:



(d) Give a trace of the actions of the parsers in constructing these trees.

Answer:

LL(1) trace:

Parse stack	Input stream	Comment
P	([]([]))[](()) $$	
S $$	([]([]))[](()) $$	predict $P \longrightarrow S$ $$
(S) S $$	([]([]))[](()) $$	predict $S \longrightarrow$ (S) S
S) S $$	[]([]))[](()) $$	match (
[S] S) S $$	[]([]))[](()) $$	predict $S \longrightarrow$ [S] S
S] S) S $$	]([]))[](()) $$	match [
] S) S $$	]([]))[](()) $$	predict $S \longrightarrow \epsilon$
S) S $$	([]))[](()) $$	match]
(S) S) S $$	([]))[](()) $$	predict $S \longrightarrow$ (S) S
S) S) S $$	[]))[](()) $$	match (
[S] S) S) S $$	[]))[](()) $$	predict $S \longrightarrow$ [S] S
S] S) S) S $$	]))[](()) $$	match [
] S) S) S $$	]))[](()) $$	predict $S \longrightarrow \epsilon$
S) S) S $$	))[](()) $$	match]
) S) S $$	))[](()) $$	predict $S \longrightarrow \epsilon$
S) S $$	)[](()) $$	predict $S \longrightarrow \epsilon$
) S $$	)[](()) $$	predict $S \longrightarrow \epsilon$
S $$	[](()) $$	match)
[S] S $$	[](()) $$	predict $S \longrightarrow$ [S] S
S] S $$	](()) $$	match [
] S $$	](()) $$	predict $S \longrightarrow \epsilon$
S $$	(()) $$	match]

(S) S $$	(()) $$	predict $S \longrightarrow$ (S) S
S) S $$	()) $$	match (
(S) S) S $$	()) $$	predict $S \longrightarrow$ (S) S
S) S) S $$	)) $$	match (
) S) S $$	)) $$	predict $S \longrightarrow \epsilon$
S) S $$	) $$	match)
) S $$	) $$	predict $S \longrightarrow \epsilon$
S $$	$$	match)
$$	$$	predict $S \longrightarrow \epsilon$

SLR(1) trace:

Parse stack	Input stream	Comment
0	([]([]))[](())$$	
0	*S*([]([]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1	([]([]))[](())$$	shift *S*
0S1(*2*	[]([]))[](())$$	shift (
0S1(*2*	*S*[]([]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*	[]([]))[](())$$	shift *S*
0S1(*2S4*[*3*	]([]))[](())$$	shift [
0S1(*2S4*[*3*	*S*]([]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*[*3S5*	]([]))[](())$$	shift *S*
0S1(*2*	([]))[](())$$	shift and reduce by $S \longrightarrow S$ [*S*]
0S1(*2*	*S*([]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*	([]))[](())$$	shift *S*
0S1(*2S4*(*2*	[]))[](())$$	shift (
0S1(*2S4*(*2*	*S*[]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*(*2S4*	[]))[](())$$	shift *S*
0S1(*2S4*(*2S4*[*3*	]))[](())$$	shift [
0S1(*2S4*(*2S4*[*3*	*S*]))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*(*2S4*[*3S5*	]))[](())$$	shift *S*
0S1(*2S4*(*2*	))[](())$$	shift and reduce by $S \longrightarrow S$ [*S*]
0S1(*2S4*(*2*	*S*))[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*(*2S4*	))[](())$$	shift *S*
0S1(*2*	)[](())$$	shift and reduce by $S \longrightarrow S$ (*S*)
0S1(*2*	*S*)[](())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*	)[](())$$	shift *S*
0	[](())$$	shift and reduce by $S \longrightarrow S$ (*S*)
0	*S*[](())$$	reduce by $S \longrightarrow \epsilon$
0S1	[](())$$	shift *S*
0S1[*3*	](())$$	shift [
0S1[*3*	*S*](())$$	reduce by $S \longrightarrow \epsilon$
0S1[*3S5*	](())$$	shift *S*
0	(())$$	shift and reduce by $S \longrightarrow S$ [*S*]
0	*S*(())$$	reduce by $S \longrightarrow \epsilon$
0S1	(())$$	shift *S*
0S1(*2*	())$$	shift (
0S1(*2*	*S*())$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*	())$$	shift *S*
0S1(*2S4*(*2*	))$$	shift (
0S1(*2S4*(*2*	*S*))$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*(*2S4*	))$$	shift *S*
0S1(*2*	)$$	shift and reduce by $S \longrightarrow S$ (*S*)
0S1(*2*	*S*)$$	reduce by $S \longrightarrow \epsilon$
0S1(*2S4*	)$$	shift *S*
0	$$	shift and reduce by $S \longrightarrow S$ (*S*)
0	*S*$$	shift by $S \longrightarrow \epsilon$
0S1	$$	shift *S*
0	*P*	shift and reduce by $P \longrightarrow S$ $$
[done]		

2.18 Consider the following LL(1) grammar for a simplified subset of Lisp:

$P \rightarrow E$ \$\$

$E \rightarrow$ atom | ' E | (E Es)

$Es \rightarrow E\ Es$ | ε

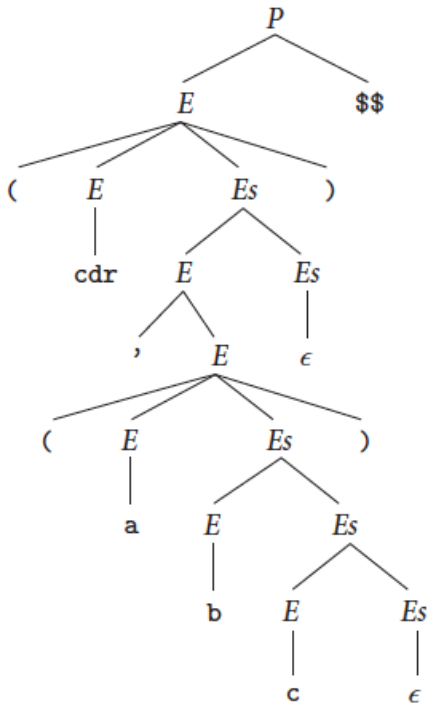
(a) What is FIRST(Es)? FOLLOW(E)? PREDICT($Es \rightarrow \varepsilon$)?

Answer:

{ atom, (, ', ϵ }
{ atom, (, ',), $$ }
{) }

(b) Give a parse tree for the string (cdr '(a b c)) $$.

Answer:



(c) Show the left-most derivation of (cdr '(a b c)) $$.

Answer:

$$
\begin{aligned}
P &\Longrightarrow \underline{E}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \underline{E}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \underline{Es}\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \underline{E}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \underline{E}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \underline{E}\ Es\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a}\ \underline{Es}\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a}\ \underline{E}\ Es\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a b}\ \underline{Es}\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a b}\ \underline{E}\ Es\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a b c}\ \underline{Es}\ \texttt{)}\ Es\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a b c}\ \texttt{)}\ \underline{Es}\ \texttt{)}\ \texttt{\$\$} \\
&\Longrightarrow \texttt{(}\ \texttt{cdr}\ \texttt{'}\ \texttt{(}\ \texttt{a b c}\ \texttt{)}\ \texttt{)}\ \texttt{\$\$}
\end{aligned}
$$

(d) Show a trace in the style of Fig 2.21 of a table-driven top-down parse of this same input.

Answer:

Parse stack	Input stream	Comment
P	`(cdr '(a b c)) $$`	
E `$$`	`(cdr '(a b c)) $$`	predict $P \longrightarrow E$ `$$`
`(` E Es `)` `$$`	`(cdr '(a b c)) $$`	predict $E \longrightarrow$ `(` E Es `)`
E Es `)` `$$`	`cdr '(a b c)) $$`	match `(`
`atom` Es `)` `$$`	`cdr '(a b c)) $$`	predict $E \longrightarrow$ `atom`
Es `)` `$$`	`'(a b c)) $$`	match `atom`
E Es `)` `$$`	`'(a b c)) $$`	predict $Es \longrightarrow E$ Es
`'` E Es `)` `$$`	`'(a b c)) $$`	predict $E \longrightarrow$ `'` E
E Es `)` `$$`	`(a b c)) $$`	match `'`
`(` E Es `)` Es `)` `$$`	`(a b c)) $$`	predict $E \longrightarrow$ `(` E Es `)`
E Es `)` Es `)` `$$`	`a b c)) $$`	match `(`
`atom` Es `)` Es `)` `$$`	`a b c)) $$`	predict $E \longrightarrow$ `atom`
Es `)` Es `)` `$$`	`b c)) $$`	match `atom`
E Es `)` Es `)` `$$`	`b c)) $$`	predict $Es \longrightarrow E$ Es
`atom` Es `)` Es `)` `$$`	`b c)) $$`	predict $E \longrightarrow$ `atom`
Es `)` Es `)` `$$`	`c)) $$`	match `atom`
E Es `)` Es `)` `$$`	`c)) $$`	predict $Es \longrightarrow E$ Es
`atom` Es `)` Es `)` `$$`	`c)) $$`	predict $E \longrightarrow$ `atom`
Es `)` Es `)` `$$`	`)) $$`	match `atom`
`)` Es `)` `$$`	`)) $$`	predict $Es \longrightarrow \epsilon$
Es `)` `$$`	`) $$`	match `)`
`)` `$$`	`) $$`	predict $Es \longrightarrow \epsilon$
`$$`	`$$`	match `)`

If you want to look at harder questions, try these

2.9 (a) Describe in English the language defined by the regular expression a*(b a* b a*) *. Your description should be a high-level characterization—one that would still make sense if we were using a different regular expression for the same language.

Answer: The set of all strings of as and bs containing an even number of bs.

(b) Write an unambiguous context-free grammar that generates the same language.

Answer:

F → As S

S→ b As b As S

As → a As

Note that putting another As between the second b and the second S in the second production would make the grammar ambiguous.

(c) Using your grammar from part (b), give a canonical (right-most) derivation of the string

b a a b a a a b b.

Answer:

$$
\begin{aligned}
G &\Longrightarrow \underline{As} \\
&\Longrightarrow \underline{S} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ \underline{S} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ As\ \mathtt{b}\ As\ \mathtt{b}\ \underline{S} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ As\ \mathtt{b}\ \underline{As}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ \underline{As}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ \mathtt{a}\ \underline{As}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \underline{As}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ As\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \underline{As}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ \underline{As}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ \mathtt{a}\ \underline{As}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow As\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \underline{As}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow \underline{As}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b} \\
&\Longrightarrow \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}
\end{aligned}
$$

2.15 Consider the following context-free grammar.

$$G \rightarrow G\,B \mid G\,N \mid \varepsilon$$
$$B \rightarrow (\,E\,)$$
$$E \rightarrow E\,(\,E\,) \mid \varepsilon$$
$$N \rightarrow (\,L\,]$$

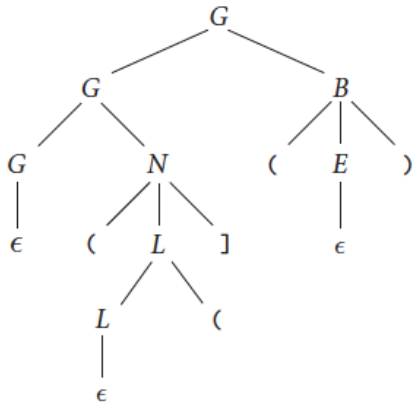
$$L \rightarrow L\,E \mid L\,(\mid \varepsilon$$

(a) Describe, in English, the language generated by this grammar. (Hint: *B* stands for "balanced"; *N* stands for "nonbalanced".) (Your description should be a high-level characterization of the language—one that is indepen- dent of the particular grammar chosen.)

Answer: Strings of balanced parentheses, with the exception that a right square bracket matches all still-open left parentheses that precede it. (Some dialects of Lisp permit these "close all" brackets.)

(b) Give a parse tree for the string (( ] ( ) .

Answer:



(c) Give a canonical (right-most) derivation of this same string.

Answer:

$$\begin{aligned}
G &\Longrightarrow G\,\underline{B} \\
&\Longrightarrow G\,(\,\underline{E}\,) \\
&\Longrightarrow \underline{G}\,(\,) \\
&\Longrightarrow G\,\underline{N}\,(\,) \\
&\Longrightarrow G\,(\,\underline{L}\,]\,(\,) \\
&\Longrightarrow G\,(\,\underline{L}\,(\,]\,(\,) \\
&\Longrightarrow \underline{G}\,(\,(\,]\,(\,) \\
&\Longrightarrow (\,(\,]\,(\,)
\end{aligned}$$

(d) What is FIRST(E) in our grammar? What is FOLLOW(E)? (Recall that FIRST and FOLLOW sets are defined for symbols in an arbitrary CFG, regardless of parsing algorithm.)

Answer: FIRST(E) = {(}; FOLLOW(E) = {(,),]}

(e) Given its use of left recursion, our grammar is clearly not LL(1). Does this language have an LL(1) grammar? Explain.

Answer: No it doesn't. Proving this is difficult, but the intuition is straightforward: In an LL(1) parser, the contents of the parse stack represents the predicted remainder of the program, and decisions are always made on the basis of the top-of-stack symbol and the next token of input. When it matches a left parenthesis, the parser must predict whether it will see a matching right parenthesis, or whether this will be "swallowed up" by a right square bracket. Since the distance to the right parenthesis or bracket is unbounded, the parser cannot make this prediction with a fixed sized grammar. Note that eliminating left recursion does not suffice to make the grammar LL(1): there is potentially unbounded