1 Chapter 4

Pg. 282

1. Consider the following context-free grammar

$$\begin{array}{lcl} \text{A} & \rightarrow & \text{B} \mid \text{Cx} \mid \epsilon \\ \text{B} & \rightarrow & \text{C} \mid \text{yA} \\ \text{C} & \rightarrow & \text{w} \mid \text{z} \mid \text{B} \end{array}$$

• Compute FIRST(A), FIRST(B), FIRST(C)

FIRST(A) = FIRST(B) ∪ FIRST(Cx) ∪ FIRST(ϵ)
FIRST(A) = $\{w, z, y, \epsilon\}$

FIRST(B) = FIRST(C) ∪ FIRST(yA)
FIRST(B) = $\{w, z, y\}$

FIRST(C) = FIRST(w) ∪ FIRST(z) ∪ FIRST(B) FIRST(C) = $\{w, z\}$ ∪ FIRST(B)
FIRST(C) = $\{w, z\}$ ∪ FIRST(C) ∪ FIRST(yA)
FIRST(C) = $\{w, z, y\}$

• Compute FOLLOW(A), FOLLOW(B), FOLLOW(C)

FOLLOW(A) = {\$} **Rule 1**
FOLLOW(A) = FOLLOW(A) ∪ FOLLOW(B) = $\{\$, x\}$ **Rule 3**

FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(A) **Rule 3**
FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(C) = $\{\$, x\}$ **Rule 3**

FOLLOW(C) = FOLLOW(C) ∪ FIRST(x) **Rule 2**
FOLLOW(C) = FOLLOW(C) ∪ FOLLOW(B) = $\{\$, x\}$ **Rule 3**

• Compute PREDICT Sets

PREDICT(A → B) = FIRST(B) = $\{w, z, y\}$
PREDICT(A → Cx) = FIRST(Cx) = $\{w, z, y\}$
PREDICT(A → ϵ) = FIRST(ϵ)\{ϵ} ∪ FOLLOW(A) = $\{\$, x\}$
PREDICT(B → C) = FIRST(C) = $\{w, z, y\}$
PREDICT(B → yA) = FIRST(yA) = $\{y\}$
PREDICT(C → w) = FIRST(w) = $\{w\}$
PREDICT(C → z) = FIRST(z) = $\{z\}$
PREDICT(C → B) = FIRST(B) = $\{w, z, y\}$

2. Recall the dangling else problem exhibited by this grammar:

$$\begin{array}{lcl} IfStat & \rightarrow & \textbf{if } Expr \textbf{ then } Statement \\ & | & \textbf{if } Expr \textbf{ then } Statement \textbf{ else } Statement \end{array}$$

Explain why adding a fi terminal like this:

$$\begin{array}{lcl} IfStat & \rightarrow & \textbf{if } Expr \textbf{ then } Statement \textbf{ fi} \\ & | & \textbf{if } Expr \textbf{ then } Statement \textbf{ else } Statement \textbf{ fi} \end{array}$$

resolves the problem of the "dangling else" problem
Pg. 155, 224, 232

Adding the token **fi** causes the token to be a member of FOLLOW(*Statement*). Since the dangling else problem causes a shift/reduce conflict, then the problem can be resolved using **LR(1)** items.

3. Consider the following context-free grammar G

$$\begin{array}{lcl} \text{A} & \rightarrow & \text{Aa} \mid \text{Ac} \mid \text{b} \mid \text{A B B} \\ \text{B} & \rightarrow & \text{aB} \mid \text{a c B} \mid \text{d} \end{array}$$

$\alpha_0 = \text{a}$
$\alpha_1 = \text{c}$
$\alpha_2 = \text{B B}$

$\beta_0 = \text{b}$

$$\begin{array}{lcl} \text{A} & \rightarrow & \text{bA'} \\ \text{A'} & \rightarrow & \text{aA'} \mid \text{cA'} \mid \text{B B} \mid \epsilon \\ \text{B} & \rightarrow & \text{aB'} \mid \text{d} \\ \text{B'} & \rightarrow & \text{cB} \mid \text{B} \end{array}$$

4. Consider one production for a context-free grammar, G:

$\text{A} \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$

What **must** be satisfied with respect to the above production for G to be parsable using a predictive parser?

For a predictive parser, the G must be at least **LL(1)**; and although the pairwise disjointness test is necessary, it is not sufficient, then:

G is $LL(1)$ if and only if $\forall i(1 \leq i \leq m) : \forall j, k : 1 \leq j, k \leq n_i \wedge (j \neq k)$,

$\text{PREDICT}(\text{A} \rightarrow \alpha_{i,j}) \cap \text{PREDICT}(\text{A} \rightarrow \alpha_{i,k}) = \emptyset$

5. Consider the following grammar:

E → P (E) | v T
P → f | ϵ
B → + E | ϵ

Compute the FIRST & FOLLOW sets and fill in the M parsing table to be used with an LL(1) table driven parser. Also, show, using your parsing table, how a table driven parser would parse the string $f(v)$

FIRST(E) = FIRST(P(E)) ∪ FIRST(vT) = $\{f, v, (\}$
FIRST(P) = FIRST(f) ∪ FIRST(ϵ) = $\{f\epsilon\}$
FIRST(T) = FIRST(tE) ∪ FIRST(ϵ) = $\{t, \epsilon\}$

FOLLOW(E) = {$} **Rule 1**
FOLLOW(E) = FOLLOW(E) ∪ FIRST())\{ϵ} = {$,)} **Rule 2**
FOLLOW(E) = FOLLOW(E) ∪ FOLLOW(T) = {$,)} **Rule 3**

FOLLOW(P) = FOLLOW(P) ∪ FIRST((E))\{ϵ} = {(} **Rule 2**

FOLLOW(T) = FOLLOW(T) ∪ FOLLOW(E) = {$,)} **Rule 3**

PREDICT(E → P(E)) = FIRST(P(E)) = $\{f, (\}$
PREDICT(E → vT) = FIRST(vT) = $\{v\}$
PREDICT(P → f) = FIRST(f) = $\{f\}$
PREDICT(P → ϵ) = FIRST(ϵ)\{ϵ} ∪ FOLLOW(T) = {$,)}
PREDICT(T → +E) = FIRST(+E) = {+}
PREDICT(T → ϵ) = FIRST(ϵ)\{ϵ} ∪ FOLLOW(T) = {$,)}

	(	)	v	f	+	$
E	E → P(E)		E → vT	E → P(E)		
P	E → ϵ			P → f		
T		T → ϵ			T → +E	→ ϵ

6. Consider the following context-free grammar

A $\rightarrow$ id = E | id = A
E $\rightarrow$ T + E | T
T $\rightarrow$ id | num | f(E)

• Give a left-most derivation of the string $a = b = c + f(a + 3)$

$$
\begin{aligned}
\text{A} &\Longrightarrow \text{id} = \text{A} \\
&\Longrightarrow \text{a} = \text{A} \\
&\Longrightarrow \text{a} = \text{id} = \text{E} \\
&\Longrightarrow \text{a} = \text{b} = \text{E} \\
&\Longrightarrow \text{a} = \text{b} = \text{T} + \text{E} \\
&\Longrightarrow \text{a} = \text{b} = \text{id} + \text{E} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{E} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(E)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(T} + \text{E)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(id} + \text{E)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(a} + \text{E)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(a} + \text{T)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(a} + \text{num)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(a} + \text{3)}
\end{aligned}
$$

• Give a right-most derivation of the string $a = b = c + f(a + 3)$

$$
\begin{aligned}
\text{A} &\Longrightarrow \text{id} = \text{A} \\
&\Longrightarrow \text{id} = \text{id} = \text{E} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{E} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(E)} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(T} + \text{E)} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(T} + \text{num)} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(T} + \text{3)} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(id} + \text{3)} \\
&\Longrightarrow \text{id} = \text{id} = \text{T} + \text{f(a} + \text{3)} \\
&\Longrightarrow \text{id} = \text{id} = \text{id} + \text{f(a} + \text{3)} \\
&\Longrightarrow \text{id} = \text{id} = \text{c} + \text{f(a} + \text{3)} \\
&\Longrightarrow \text{id} = \text{b} = \text{c} + \text{f(a} + \text{3)} \\
&\Longrightarrow \text{a} = \text{b} = \text{c} + \text{f(a} + \text{3)}
\end{aligned}
$$

7. Consider a general grammar G with m nonterminals:

$G_i \rightarrow \alpha_{i1} \mid \alpha_{i2} \mid ... \mid \alpha_{i_{n_i}}, i = 0, ..., m$

Where

$$\forall\ i : 0 \leq i \leq m : \bigcap_{j=1}^{n_i} \text{FIRST}(\alpha_{i_j}) = \emptyset$$

Explain why the above equation is not a (correct) necessary condition for the grammar to be LL(1).

Note: The above equation is not the correct way to test for disjointness. However, it is an often made mistake - this exercise should prevent you from making the same mistake

The above equation is not correct since it intersects all First sets rather than checking them in pairs.

8. Let M[A, a] be the contents of a LL(1) table-driven parsing table indexed by a non-terminal A and a terminal a (The lookahead).

When parsing we can be in one of the following 3 situations (X is the symbol on top of the stack.
Pg. 146

a) X = a = \$.

Case 1: The top of the stack is also a \$. Accept the input.

Case 2: The top of the stack is a terminal, not \$; Signal Error unless stack consists of nullable nonterminals.

b) X = $a \neq$ \$.

Case 1: Update lookahead to next token on input & remove top of stack.

Case 2: Signal error - Token does not match top of stack.

c) X is a non-terminal.

Case 1: M[X, a] is a valid production, pop X off the stack and push the right hand side (in reverse order)

Case 2: M[X, a] is an error, signal error.

9. For a production A $\rightarrow X_1X_2...X_n$ how many different LR(0) items can you create? Show what they look like.

$n + 1$ LR(0) items can be created, e.g.

A $\rightarrow \bullet X_1X_2...X_n$
A $\rightarrow X_1 \bullet X_2...X_n$
A $\rightarrow X_1X_2... \bullet X_n$
A $\rightarrow X_1X_2...X_n\bullet$

10. Let G be an *ambiguous* context-free grammar. We know that the length of S $\Longrightarrow {}^{*}_{RM}\omega$ is not always the same length as S $\Longrightarrow {}^{*}_{LM}\omega$

Consider the string abba. Create a context-free grammar that proves this point, and show the 2 different derivations of different length.

A $\rightarrow$ B | aB
B $\rightarrow$ aA | bbB | Ba | ϵ

Left-most Derivation
A $\Longrightarrow$ aB
A $\Longrightarrow$ abbB
A $\Longrightarrow$ abbBa
A $\Longrightarrow$ abba

Right-most Derivation
A $\Longrightarrow$ B
A $\Longrightarrow$ Ba
A $\Longrightarrow$ aAa
A $\Longrightarrow$ aBa
A $\Longrightarrow$ abbBa
A $\Longrightarrow$ abba

11. If a context-free grammar is not LL(1), can it then be LR(1) without changing anything? Explain and/or give an example.

If a context-free grammar is not LL(1) then it fails the sufficient test:

$\forall i(q \leq i \leq m)$: $\forall\ j, k \leq j, k \leq n_i \wedge (j \neq k)$,
PREDICT(A $\rightarrow \alpha_{i,j}$) $\cap$ PREDICT(A $\rightarrow \alpha_{i,k}$) $\neq \emptyset$.

Then it follows that the grammar in question contains common subexpressions. Since an LR(1) parser performs a right-most derivation in reverse, then the next state is determined by the current top of the stack and the look-ahead and therefore the common subexpressions do not make the grammar ambiguous. Hence, no change is needed.

12. Assume a context-free grammar has the following production:

$B \rightarrow \beta$

and that:

$S \implies {}^{*}\alpha B\delta \implies \alpha\beta\delta$

Also assume that:

$\beta \implies {}^{*}\epsilon$, and $\epsilon \notin$ FIRST(δ)

Explain why M[B, b] = { B $\rightarrow \beta$}. $\forall$ b $\in$ FIRST(β), as well as $\forall$ b $\in$ FOLLOW(B).

1. M[B, b] = { B $\rightarrow \beta$ } $\forall$b $\in$ FIRST(β) because the lookahead matches each of the first terminals of β.

2. M[B, b] = { B $\rightarrow \beta$ } $\forall$b $\in$ FOLLOW(B) because, since, B is nullable, then the lookahead must match the next set of terminals; i.e. FIRST(δ)

14. Recall that shift/reduce parsers have two different actions: either they can shift a token from input on to the stack (*shift* s), or they can reduce the handle on the stack by a production (*reduce* A $\rightarrow \alpha$)

- Explain why an LR parser that works like this always mimcs a reverse right most derivation. (Hint: What is the relationship between the location of the handle on the stack and the configuration of the parser after the reduction has been performed).

An LR parser that has a shift and reduce action always mimics a reverse right most derivation since it pushes all matching input (in addition to the state) before it performs a series of reductions until it arrives at the initial sentential form.

- What is the difference between a *reverse right most* and a *left most* derivation?

A reverse right most derivation derives the output starting from the latest sentential form to the most earliest (bottom-up). A left most derivation derives the output starting from the earliest sentential form to the latest.

15. Let G be a context-free grammar. Explain why
closure({[A $\rightarrow \alpha$a$\beta\bullet$}) = {[A $\rightarrow \alpha$a$\beta\bullet$}].

The longest input has been matched and the only action the automaton can take is a reduction.

16. Using an LR(0) item set, explain what causes a shift/reduce conflict.

A shift/reduce conflict occurs when there is one shift item where the next symbol is a terminal and one reduce item in an action table such that:

$[A \rightarrow \alpha\bullet \text{ a } \beta]$
$[B \rightarrow \alpha\bullet]$

And a $\in$ FOLLOW(B)

17. Give the necessary conditions for a reduce/reduce conflict to be present between these two items: [A $\rightarrow \alpha\bullet$] and [B $\rightarrow \beta\bullet$]
Pg. 218, 219

A reduce/reduce conflict occurs when there are two reduce items in an action table and their follow sets are not disjoint; i.e.

FOLLOW(A) $\cap$ FOLLOW(B) $\neq \emptyset$

18. Consider the following context-free grammar

$$\begin{array}{lcl} A & \rightarrow & BCx \mid y \\ B & \rightarrow & yA \mid \epsilon \\ C & \rightarrow & Ay \mid \epsilon \end{array}$$

• Show a left-most derivation of the string yyyyx.

Left-most Derivation

$$\begin{array}{lcl} A & \Longrightarrow & BCx \\ & \Longrightarrow & yACx \\ & \Longrightarrow & yACx \\ & \Longrightarrow & yyCx \\ & \Longrightarrow & yyAyx \\ & \Longrightarrow & yyyyx \end{array}$$

• Show a right-most derivation of the string yyyyx.

Right-most Derivation

$$\begin{array}{lcl} A & \Longrightarrow & BCx \\ & \Longrightarrow & BAyx \\ & \Longrightarrow & Byyx \\ & \Longrightarrow & yAyyx \\ & \Longrightarrow & yyyyx \end{array}$$

• Is this grammar ambiguous? Prove that it is or is not.

The grammar is ambiguous since it has two different parse trees for the left & right most derivations.

19. Consider the following context-free grammar:

S'	$\rightarrow$	S
S	$\rightarrow$	A A
S	$\rightarrow$	b c
A	$\rightarrow$	b a A
A	$\rightarrow$	c

Construct the SLR(1) automaton using LR(0) item sets and create the corresponding action and goto tables.

20. For each of the LR(0) item sets in the states of the SLR(1) automaton from the previous question, compute the kernel.
21. In which way does an LR(1) item differ from an LR(0) item?

LR(1) items contain a reduction lookahead; LR(0) items do not.
For LR(o) items, reductions are done on the FOLLOW sets, for LR(1) on the reduction lookahead.

The reduction lookeahead is also a member of FOLLOW(A).

22. When fusing states in the creation of the LALR(1) graph based on the LR(1) graph, what condition must be satisfied?
Pg. 232, 235

Given two states I_i, I_j and $core(I_i) = I_j$ I_i and I_j can be merged unless the merger creates conflicts.

23. For a context-free grammar G, if the corresponding LALR(1) and SLR(1) graphs are of equal size, argue that G is SLR(1); that is, if there are no conflicts in the LALR(1) graph, then the SLR(1) will also be conflict-free.

For any two states I_i, I_j recall the condition to merge two states: I_i and I_j can merge if the merger does not cause a conflict.

For some grammar G, suppose that the corresponding SLR(1) & LALR(1) are of equal size and there are no conflicts in the LALR(1) graph.

Since the LALR(1) is a consolidated, equivalent graph, then the graphs must be the same and hence, grammar G is SLR(1).

Therefore, grammar G is SLR(1).

24. Given an LR(1) graph for a context-free grammar G, describe an algorithm for transforming it into the corresponding SLR(1) graph for the same grammar.

2 Chapter 5

Pg. 321

1. Consider the following context-free grammar:

$$
\begin{array}{lcl}
\text{E} & \rightarrow & \text{T E'} \\
\text{E'} & \rightarrow & \text{+ T E'} \mid \epsilon \\
\text{T} & \rightarrow & \text{F T'} \\
\text{T'} & \rightarrow & \text{* F T'} \mid \epsilon \\
\text{F} & \rightarrow & \text{(E)} \mid \text{id}
\end{array}
$$

Give a left-most derivation of the string $id + id * id$.

Left-most Derivation
E $\Longrightarrow$ T E'
E $\Longrightarrow$ F T' E'
E $\Longrightarrow$ id T' E'
E $\Longrightarrow$ id E'
E $\Longrightarrow$ id + T E'
E $\Longrightarrow$ id + F T' E'
E $\Longrightarrow$ id + id T' E'
E $\Longrightarrow$ id + id * F T'
E $\Longrightarrow$ id + id * id T'
E $\Longrightarrow$ id + id * id

Give a right-most derivation of the string $id + id * id$.

Right-most Derivation
E $\Longrightarrow$ T E'
E $\Longrightarrow$ T + T E'
E $\Longrightarrow$ T + T
E $\Longrightarrow$ T + F T'
E $\Longrightarrow$ T + F * F T'
E $\Longrightarrow$ T + F * F
E $\Longrightarrow$ T + F * id
E $\Longrightarrow$ T + id * id
E $\Longrightarrow$ F T' + id * id
E $\Longrightarrow$ F + id * id
E $\Longrightarrow$ id + id * id

Are the parse trees for the left-most and the right-most derivations the same? If yes, draw the tree, if no draw the two different trees.

Explain why both derivations will always be of the same length if they produce the same syntax tree.

2. Give a good reason to have all parse tree nodes in a compiler inherit from the same super class?

A Good reason to have all parse tree nodes in a compiler inherit from the same super class is to be able to support LR(1) & LALR(1) parsing since the its' the right-most derivation in reverse, the parse tree nodes would have to be able to be casted to some sort of parent class as the parser pops things off the stack.

3. Consider a new relational operator *in* which has the following syntax:

RelationalExpr ::= *Expression* **in** { *Expression*[, *Expression*]* }

An example could be **a + 7 in { c, 4, b - 2 }** if the value of **a + 7** is equal to (==) the value of **c**, or equal to 4, or equal to the value of $b - 2$.