Solution of Homework 2

Dragon:
4.2.1, 4.2.4, 4.2.5, 4.2.7
4.3.1
4.4.3

Tiger:
3.4, 3.5, 3.6, 3.7

Dragon Book

4.2.1 Considering the context-free grammar:

$$S \rightarrow S\,S + \mid S\,S * \mid a$$

and the string aa+a*

(1) Give a leftmost derivation of the string

(2) Give a rightmost derivation of the string

(3) Give a parse tree for the string

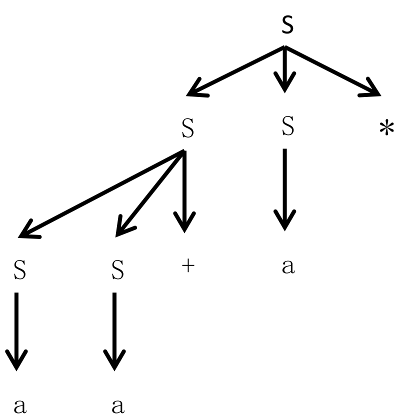
(4) Is the grammar ambiguous?

(5) Describe the language generated by the grammar

答案:

(1) $S \Rightarrow_{lm} SS* \Rightarrow_{lm} SS+S* \Rightarrow_{lm} aS+S* \Rightarrow_{lm} aa+S* \Rightarrow_{lm} aa+a*$

(2) $S \Rightarrow_{rm} SS* \Rightarrow_{rm} Sa* \Rightarrow_{rm} SS+a* \Rightarrow_{rm} Sa+a* \Rightarrow_{rm} aa+a*$



(3)

(4) 没有二义性
(5) 只含+和*，操作数均为 a 的算术表达式的后序遍历

4.2.4 Show that [] and {} extension do not add power to grammars.

答案: S->[a] 等价于 S->$\varepsilon|a$

S->{a} 等价于 S->S' S' -> $\varepsilon|aS'$

4.2.5 Use notations from 4.2.4 to simplify the following grammar:

$$\begin{array}{lcl} stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\ & | & \textbf{if } stmt \textbf{ then } stmt \\ & | & \textbf{begin } stmtList \textbf{ end} \\ stmtList & \rightarrow & stmt \textbf{ ; } stmtList \mid stmt \end{array}$$

答案:

stmt -> **if** *expr* **then** *stmt* [**else** *stmt*]

| **begin** *stmtList* **end**

stmtList -> *stmt*{**;***stmt*}

4.2.7 A grammar symbol X (terminal or non-terminal) is *useless* if there is no derivation of the form $S \Rightarrow^* wXy \Rightarrow^* wxy$. That is, X can never appear in the derivation of any sentence.

(1) Give an algorithm to eliminate all productions contains useless symbols

(2) Apply your algorithm to grammar:

S-> 0 | A

A-> AB

B->1

答案：

(1) 注意：首先要考虑产生式是否终止，其次考虑是否在产生式中出现

(a)求出非终结符是否终止

对于每个非终结符 X

  若 X 存在产生式不含非终结符，则 X 终止

  否则，对于 X 的每个产生式中的非终结符，

    若含有 X，则 X 不终止，

    否则，若每个非终结符均终止，则 X 终止，否则不终止

(b)去掉 useless

令文法 G 为空

从起始的非终结符 X 开始

  对于 X 的每个产生式 P，

    若 P 不含非终结符，则将 X->P 加入文法 G

    否则，若 P 中每个非终结符均终止，

      则将 X->P 加入文法 S'，并对每个非终结符 X'递归调用这个过程

返回文法 G

算法并不唯一，这只是一个很罗嗦的实现

(2) 对于这里给出的算法

  首先 S 和 B 终止，A 不终止

  然后从 S->0 开始，S->0 加入文法 G

    S->A,A 不终止，至此递归调用结束

  所以最终返回结果 S->0

4.3.1 The following grammar for regular expressions over symbols a and b only.

$$
\begin{array}{lll}
rexpr & \rightarrow & rexpr + rterm \mid rterm \\
rterm & \rightarrow & rterm\ rfactor \mid rfactor \\
rfactor & \rightarrow & rfactor * \mid rprimary \\
rprimary & \rightarrow & \mathbf{a} \mid \mathbf{b}
\end{array}
$$

(1) Left factor this grammar

(2) Does left factoring make the grammar suitable for top-down parsing?

(3) Eliminate left recursion from the original grammar

(4) Is the resulting grammar suitable for top-down parsing?

答案：

(1) rexpr -> rexpr + rterm | rterm
    rterm -> rterm rfactor | refactor
    rfactor -> rfactor * | rprimary
    rprimary-> a| b
(2) 不可以，因为存在左递归
(3) rexpr -> rterm rexpr'

    rexpr'-> + rterm rexpr' | ε

    rterm-> rfactor rterm'

    rterm' -> rfactor rterm' |ε

    rfactor-> rprimay rfactor'

    rfactor'-> *rfactor' |ε

    rprimary-> a | b
(4) 可以

4.4.3 计算练习 4.2.1 中文法的 FIRST 和 FOLLOW 集合

$$S \rightarrow S\ S + |\ S\ S * |\ a$$

答案：FIRST(S)=a
FOLLOW(S)= $a+*

Tiger Book

3.4 Write a grammar that accepts the same language as Grammar 3.1, but that us suitable for LL(1) parsing. That is, eliminate the ambiguity, eliminate the left recursion, and (if necessary) left-factor.

Grammar 3.1:

S -> S; S

S -> id := E

S -> print (L)

E -> id

E -> num

E -> E + E

E -> (S , E)

L -> E

L -> L , E

Answer:

S -> id := E ; S

S -> print (L) ; S

E -> (S , E)

E -> id E’

E -> num E’

E’ -> ε

E’ -> + E

L -> E L’

L’ -> ε

L’ -> , E L’

3.5 Find nullable, FIRST, and FOLLOW sets for the grammar; then construct the LL(1) parsing table.

S’ -> S $

S ->

S -> X S

B -> \ begin { WORD }

E -> \ end { WORD }

X -> B S E

X -> { S }

X -> WORD

X -> begin

X -> end

X -> \ WORD

Answer:

First:

S’	ε \ { WORD begin end
S	ε \ { WORD begin end
B	\
E	\
X	\ { WORD begin end
\	\
{	{
}	}
WORD	WORD
begin	Begin
end	end

Follow

S’	$

S	$ \ }
B	\ { WORD begin end
E	\ { WORD begin end } $
X	\ { WORD begin end } $

LL(1) Tans

	\	begin	end	WORD	{	}	$
S’	S’->S	S’->S	S’->S	S’->S	S’->S		S’->S
S	S-> ε S->XS	S->XS	S->XS	S->XS	S->XS	S-> ε	S-> ε
B	B->\begin{WORD}						
E	E->\end{WORD}						
X	X->BSE X->\WORD	X->begin	X->end	X->WORD	X->{S}		

3.6

a. Calculate nullable, FIRST, and FOLLOW for this grammar:

S -> u B D z

B -> B v

B -> w

D -> E F

E -> y

E ->

F -> x

F->

b. Constructor the LL(1) parsing table.

c. Give evidence that this grammar is not LL(1).

d. Modify the grammar as little as possible to make an LL(1) grammar that accepts the same language.

Answer:

a)

First

S	u
B	w
D	y x ε
E	y ε
F	x ε
u	u
v	v
w	w
x	x
y	y
z	z

Follow

S	$
B	y x z v
D	z
E	x z
F	z

b) LL(1) Trans

	u	v	w	x	y	z	$
S	S->uBDz						
B			B->Bv B->w				
D				D->EF	D->EF	D->EF	
E				E-> ε	E->y	E-> ε	
F				F->x		F-> ε	

c) b 中构建的 LL(1)转换表中含有包含多个语法规则的表项，所以该语法不是 LL(1)语法。

d) 将 B -> B v

B -> w

改为

B -> w B'

B' -> ε

B' -> v B'

3.7

a. Left-factor this grammar.

S -> G $

G -> P

G -> P G

P -> id : R

R ->

R -> id R

b. Show that the resulting grammar is LL(2). ...

c. Show how the tok variable and advance function should be altered for recursive-descent parsing with two-symbol lookahead.

d. Use the grammar class hierarchy (Fig 3.29) to show that the left-factored grammar is LR(2).

e. Prove that no string has two parse trees according to this (left-factored) grammar.

Answer:

a)

S -> G $

G -> P G'

G' -> ε

G' -> G

P -> id : R

R -> ε

R -> id R

b)

尝试为该文法构造 LL(1)转换表，如下

	:	id	$
S		S -> G	
G		G -> P G’	
G’		G’ -> G	G’ -> ε
P		P -> id : R	
R		R -> ε R -> id R	R -> ε

该表中[R, id]项包含两个语法规则。可以通过向后再看一个词法记号，决定使用哪个语法规则：如果是 id，那么使用规则 R->id R;如果是其它词法记号，那么使用规则 R->ε。因此，该文法是 LL(2)文法。

c)
```
int tok[2];
  tok[0] = getToken();
  tok[1] = getToken();
  void advance() {
    tok[0] = tok[1];
    tok[1] = getToken();
  }
```

d) 由图 3.29 可以看出 LL(k)文法必然是 LR(k)文法。因为 left-factored 后的文法是 LL(2)文法，所以也是 LR(2)文法。

e）因为 LL(2)文法是非二义文法，所以对于任意的可以表示的字符串，不会产生多个解析树。