第四章第二次作业

1. 使得文法的预测分析产生回溯的原因是什么？仅使用 FIRST 集合可以避免回溯吗？为什么？

Answer

引起回溯的原因是，当文法中关于某个非终结符的产生式有多个候选，且根据当前的输入符号无法确定选用唯一的产生式，就会引起回溯。

仅使用 FIRST 集合不可以避免回溯。因为引起回溯的原因除了①相同左部的产生式的右部的 FIRST 集合交际不为空之外，还包括②文法中含有左递归、和③相同非终结符产生式的右部都存在 ε，且该非终结符的 FOLLOW 集合中含有其他产生式右部 FIRST 集合中的元素。因此仅使用 FIRST 集合不可以避免回溯。

2. 考虑文法：

```
lexp→atom | list
atom→number | identifier
list→(lexp-seq)
lexp-seq→lexp-seq lexp | lexp
```

1) 消除左递归
2) 求得该文法的 FIRST 集合和 FOLLOW 集合
3) 说明所得的文法是 LL(1)文法
4) 为所得的文法构造 LL(1)分析表
5) 对输入串(a (b (2)) (c))给出相应得 LL(1)分析程序的动作

Answer

1) 对于最后一句产生式 lexp-seq→lexp-seq lexp | lexp，消除这个产生式的直接左递归：

   lexp-seq→ lexp A　　A → lexp A | ε

   故消除左递归后的文法如下：

   lexp→atom | list

   atom→number | identifier

   list→(lexp-seq)

   lexp-seq→ lexp A

   A → lexp A | ε

2) FIRST(A) = { number, identifier , (, ε }

   FIRST(lexp-seq) = { number, identifier , (, ε }

   FIRST(list) = { (}

FIRST(atom) = { number, identifier }

FIRST(lexp) = { number, identifier, (}

FOLLOW(lexp) = FIRST(A)∪FOLLOW(A)∪{ $ } = { number, identifier , (,) , $ }

FOLLOW(atom) = FOLLOW(lexp) = { number, identifier , (,) , $ }

FOLLOW(list) = FOLLOW(lexp) = { number, identifier , (,) , $ }

FOLLOW(lexp-seq) = {) }

FOLLOW(A) = FOLLOW(lexp-seq) = {) }

3) 1)中得出的文法不存在左公因子或左递归；

对于产生式 lexp→atom | list，FIRST(atom)∩FIRST(list)=Φ，ε 不属于 FIRST(atom)，ε 不属于 FIRST(list);

对于产生式 lexp-seq’ → number lexp-seq’ | identifier lexp-seq’ | (lexp-seq) lexp-seq’ | ε，

FIRST((lexp-seq) lexp-seq’)∩FIRST(number lexp-seq’)∩FIRST(identifier lexp-seq’)∩FIRST(ε) = Φ，

FIRST(lexp-seq’)∩FOLLOW(lexp-seq’) = Φ，因此该文法为 LL(1)文法。

非终结符	输入符号				
	number	**identifier**	**(**	**)**	**$**
lexp	lexp→atom	lexp→atom	lexp→list		
atom	atom→number	atom→identifier			
list			list→(lexp-seq)		
lexp-seq	lexp-seq→ lexp A	lexp-seq→lexp A	lexp-seq→ lexp A		
A	A → lexp A	A → lexp A	A → lexp A	A → ε	

4)

已匹配	栈	输入	动作
	lexp$	(a (b (2)) (c))$	
	list $	(a (b (2)) (c))$	输出 lexp→list
	(lexp-seq)$	(a (b (2)) (c))$	输出 list→(lexp-seq)
(	lexp-seq)$	a (b (2)) (c))$	匹配 (
(	lexp A) $	a (b (2)) (c))$	输出 lexp-seq→lexp A
(	atom A)$	a (b (2)) (c))$	输出 lexp→atom
(	identifier A)$	a (b (2)) (c))$	输出 atom→identifier
(a	A)$	(b (2)) (c))$	匹配 a
(a	lexp A)$	(b (2)) (c))$	输出 A → lexp A
(a	list A)$	(b (2)) (c))$	输出 lexp→list

(a	(lexp-seq)A)$	(b (2)) (c))$	输出 list→(lexp-seq)
(a(	lexp-seq)A)$	b (2)) (c))$	匹配 (
(a(	lexp A)A)$	b (2)) (c))$	输出 lexp-seq→lexp A
(a(	atom A)A)$	b (2)) (c))$	输出 lexp→atom
(a(	identifier A)A)$	b (2)) (c))$	输出 atom→identifier
(a(b	A)A)$	(2)) (c))$	匹配 b
(a(b	lexp A)A)$	(2)) (c))$	输出 A → lexp A
(a(b	list A)A)$	(2)) (c))$	输出 lexp→list
(a(b	(lexp-seq) A)A)$	(2)) (c))$	输出 list→(lexp-seq)
(a(b(	lexp-seq) A)A)$	2)) (c))$	匹配 (
(a(b(	lexp A) A)A)$	2)) (c))$	输出 lexp-seq→ lexp A
(a(b(	atom A) A)A)$	2)) (c))$	输出 lexp→atom
(a(b(	number A) A)A)$	2)) (c))$	输出 atom→number
(a(b(2	A) A)A)$	)) (c))$	匹配 2
(a(b(2	) A)A)$	)) (c))$	输出 A → ε
(a(b(2)	A)A)$	) (c))$	匹配)
(a(b(2)	)A)$	) (c))$	输出 A → ε
(a(b(2))	A)$	(c))$	匹配)
(a(b(2))	lexp A)$	(c))$	输出 A → lexp A
(a(b(2))	list A)$	(c))$	输出 lexp→list
(a(b(2))	(lexp-seq)A)$	(c))$	输出 list→(lexp-seq)
(a(b(2))(	lexp-seq)A)$	c))$	匹配 (
(a(b(2))(	lexp A)A)$	c))$	输出 lexp-seq→lexp A
(a(b(2))(	atom A)A)$	c))$	输出 lexp→atom
(a(b(2))(	identifier A)A)$	c))$	输出 atom→identifier
(a(b(2))(c	A)A)$	))$	匹配 c
(a(b(2))(c	)A)$	))$	输出 A → ε
(a(b(2))(c)	A)$	)$	匹配)
(a(b(2))(c)	)$	)$	输出 A → ε
(a(b(2))(c))	$	$	匹配)