编译原理第四章第二次作业

李昊宸 2017K8009929044

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？

答:

产生回溯的原因：某个非终结符对应多个候选产生式，这些产生式的右部的第一个终结符相同，从而导致语法分析器选择了错误的产生式，在预测到最后时才发现错误，从而导致回溯。

仅用 FIRST 集合可以避免回溯吗？不可以。

举个例子，A -> α|β，并且假设β可以推导得到ε，且假设存在其他的产生式，使得 x∈FIRST(α)且∈FOLLOW(A)，这样一来ε∈FIRST(β)，从而得到 FIRST(α)∩FOLLOW(A)非空。这样的结果是当预测器读头读取到输入为 x 时，预测器并不知道是选择 A -> α还是选择 A-> β，从而导致回溯的可能。

2.考虑文法:

lexp -> *atom* | *list*

atom -> number | identifiler

list -> (*lexp–seq*)

lexp-seq -> *lexp-seq lexp* | *lexp*

a．消除左递归

b．求得该文法的 FIRST 集合和 FOLLOW 集合

c．说明所得的文法是 LL(1)文法

d．为所得的文法构建 LL(1)分析表

e．对输入串(a (b (2)) (c))给出相应的 LL(1)分析程序的动作

注：总觉得这个 identifilier 拼错了，应该是 identifier。。。

答:

a．消除左递归

lexp -> *atom* | *list*

atom -> number | identifiler

list -> (*lexp–seq*)

lexp-seq -> *lexp lexp-seq* ‘

lexp-seq’ -> *lexp* lexp-seq ‘| ε

b．

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

FIRST(atom) = { number , identifiler }

FIRST(list) = { (}

FIRST(lexp-seq) = { number , identifiler , (}

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

FOLLOW(lexp) = { $, number , identifiler , (,) }

FOLLOW (atom) = { $, number , identifiler , (,) }

FOLLOW (list) = { $, number , identifiler , (,) }

FOLLOW (lexp-seq) = {) }
FOLLOW (lexp-seq') = {) }
c．
一个文法 G 如果是 LL(1)的，当且仅当 G 的任何两个产生式 A→α|β 满足下面的条件：
1）FIRST(α)∩FIRST(β)是空集，因此 ε 不可能同时属于两个集合
2）若β可以推导得到ε，则 FIRST(α)∩FOLLOW(A)是空集，反之亦然

对于 1），ε只存在于 FIRST(lexp-seq')，故没有矛盾
对于 2），只有 *lexp-seq'* 可以推导出ε，则需要 FIRST(lexp)∩FOLLOW(*lexp-seq'*)为空集，显然该条件也满足。

故该文法是 LL（1）文法。

d．构造 LL(1)分析表

M[Non_T,T]	number	identifiler	(	)	$
lexp	*lexp* -> *atom*	*lexp* -> *atom*	*lexp* -> *list*		
atom	*atom* -> number	*atom* -> identifiler			
list			*list* -> (*lexp–seq*)		
lexp-seq	*lexp-seq* -> *lexp lexp-seq '*	*lexp-seq* -> *lexp lexp-seq '*	*lexp-seq* -> *lexp lexp-seq '*		
lexp-seq'	*lexp-seq'* -> *lexp* lexp-seq '	*lexp-seq'* -> *lexp* lexp-seq '	*lexp-seq'* -> *lexp* lexp-seq	*lexp-seq'* ->ε	

e．对输入串(a (b (2)) (c))给出相应的 LL(1)分析程序的动作
1.栈底为$,栈中只有一个符号 lexp。读头读到符号（，选择产生式 *lexp* -> *list* ，栈状态为 list $。
2.栈顶为 list，读头读到符号（，选择产生式 *list* -> (*lexp–seq*)，栈顶此时为（，与读头相同，栈顶出栈，栈状态为 lexp-seq ）$。
3.栈顶为 lexp-seq，读头读到符号 a，a 是 identifiler，选择产生式 *lexp-seq* -> *lexp lexp-seq '* ，栈状态为 lexp lexp-seq' ）$。
4.栈顶为 lexp，读头读到符号 a，a 是 identifiler，选择产生式 *lexp* -> *atom*，栈状态为 atom lexp-seq' ）$。
5.栈顶为 atom，读头读到符号 a，a 是 identifiler，选择产生式 *atom* -> identifiler，栈顶此时为 a，与读头相同，栈顶出栈，栈状态为 lexp-seq' ）$。
6.栈顶为 lexp-seq'，读头读到符号（，选择产生式 *lexp-seq'* -> *lexp* lexp-seq，栈顶此时为 lexp，选择产生式 *lexp* -> *list*，选择产生式 *list* -> (*lexp–seq*)，此时栈顶与读头相同，栈顶出栈，栈状态为 lexp-seq）lexp-sep'）$。
7.栈顶为 lexp-seq，读头为 b，b 是 identifiler，选择产生式 *lexp-seq* -> *lexp lexp-seq '* ，

栈顶此时为 lexp，选择产生式 *lexp* -> *atom*，选择产生式 *atom* -> identifiler，此时栈顶与读头相同，栈顶出栈，栈状态为 lesp-seq’ ）lexp-sep‘）$。
8. 栈顶为 lexp-seq’，读头读到符号（，选择产生式 *lexp-seq’* -> *lexp* lexp-seq，栈顶此时为 lexp，选择产生式 *lexp* -> *list*，选择产生式 *list* -> (*lexp–seq*)，此时栈顶与读头相同，栈顶出栈，栈状态为 lexp-seq）lexp-sep‘）lexp-sep‘）$。
9. 栈顶为 lexp-seq，读头为 2，2 是 number，选择产生式 *lexp-seq* -> *lexp lexp-seq ‘* ，栈顶此时为 lexp，选择产生式 *lexp* -> *atom*，选择产生式 *atom* -> number，此时栈顶与读头相同，栈顶出栈，栈状态为 lexp-seq’ ）lexp-sep‘）lexp-sep‘）$。
10.栈顶为 lexp-seq’，读头为)，选择产生式 *lexp-seq’* ->ε，栈状态为)))$，此时栈顶和读头相同，栈顶出栈，栈状态为 lexp-sep‘）lexp-sep‘）$，读头为)，选择产生式 *lexp-seq’* ->ε，栈状态为）lexp-sep‘）$，栈顶出栈，栈状态为 lexp-sep‘）$。
11.栈顶为 lexp-seq’，读头为 （，选择产生式 *lexp-seq’* -> *lexp* lexp-seq，栈顶此时为 lexp，选择产生式 *lexp* -> *list*，选择产生式 *list* -> (*lexp–seq*)，此时栈顶与读头相同，栈顶出栈，栈状态为 lexp-seq）lexp-sep‘）$。
12.栈顶为 lexp-seq，读头为 c，从是 identifiler，选择产生式 *lexp-seq* -> *lexp lexp-seq ‘* ，栈顶此时为 lexp，选择产生式 *lexp* -> *atom*，选择产生式 *atom* -> identifiler，此时栈顶与读头相同，栈顶出栈，栈状态为 lexp-seq’ ）lexp-sep‘）$。
13.栈顶为 lexp-seq’，读头为)，选择产生式 *lexp-seq’* ->ε，栈状态为）lexp-sep‘）$，栈顶与读头相同，栈顶出栈，栈状态为 lexp-sep‘）$，读头为)，选择产生式 *lexp-seq’* ->ε，栈状态为）$，栈顶与读头相同，栈顶出栈。
14.栈顶为$，读头为$，匹配成功。