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*

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. 为所得的文法构建LL(1)分析表
5. 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：

1. 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。