编译原理

第四章
语法分析

哈尔滨工业大学 陈鄞

第5讲（语法分析_2）要点

- 如何实现自顶向下的分析?
  - 递归下降分析法
- 什么样的文法能够实现确定的自顶向下分析?
  - LL(1)文法
- 如何实现确定的自顶向下分析?
  - 递归的方式：　基于预测分析表对递归下降分析法进行扩展
  - 非递归的方式：基于预测分析表构造自动机

1、递归的预测分析

➢ 是对递归下降分析框架的扩展

```
A( Token )
{
    if Token∈SELECT(A →α1)
        code1;
    if Token∈SELECT(A →α2)
        code2;
        …
    if Token∈SELECT(A →αn)
        coden;
}
```

$A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_n$

code*1*　code*2*　code*n*

code*i*: 设$\alpha_i = X_1 X_2 \ldots X_k$

for (j = 1 to k)

- if $X_j \in V_T$
  - if $X_j == Token$ then GetNext (Token)
  - else ($X_j \neq Token$) Error( )
- else ($X_j \in V_N$) X_j( )

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
    begin
      if TOKEN≠'program' then ERROR;
      GETNEXT(TOKEN);
      DECLIST(TOKEN);
      if TOKEN≠':' then ERROR;
      GETNEXT(TOKEN);
      TYPE(TOKEN)
      if TOKEN≠';' then ERROR;
      GETNEXT(TOKEN);
      STLIST(TOKEN);
      if TOKEN≠'end' then ERROR;
      GETNEXT(TOKEN);
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLIST(TOKEN);
    begin
        if TOKEN≠'id'  then ERROR;
        GETNEXT(TOKEN);

        DECLISTN(TOKEN);
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLISTN(TOKEN);
    begin
        if TOKEN =',' then
          begin
            GETNEXT(TOKEN);
            if TOKEN≠'id'  then ERROR;
            GETNEXT(TOKEN);
            DECLISTN(TOKEN);
          end
        else  if TOKEN≠':'  then ERROR;
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLIST(TOKEN);
    begin
        if TOKEN≠'s'  then ERROR;
        GETNEXT(TOKEN);
        STLISTN(TOKEN);
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLISTN(TOKEN);
        begin
            if TOKEN =';' then
               begin
                 GETNEXT(TOKEN);

                 if TOKEN≠'s'  then ERROR;
                 GETNEXT(TOKEN);

                 STLISTN(TOKEN);
               end
            else  if TOKEN≠'end'  then ERROR;

        end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure TYPE(TOKEN);
    begin
        if TOKEN≠'real' and TOKEN≠'int'
            then ERROR;
        GETNEXT(TOKEN);
    end
```

例（MOOC）

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
        begin
            if TOKEN≠'program'  then ERROR;

            GETNEXT(TOKEN);
            DECLIST(TOKEN);

            if TOKEN≠':'  then ERROR;

            GETNEXT(TOKEN);
            TYPE(TOKEN);

            GETNEXT(TOKEN);
            if TOKEN≠';'  then ERROR;

            GETNEXT(TOKEN);
            STLIST(TOKEN);

            if TOKEN≠'end'  then ERROR;
        end
```

例（MOOC）

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure TYPE(TOKEN);
        begin
            if TOKEN≠'real' and TOKEN≠'int'
               then ERROR;
        end
```

例（MOOC）

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
program DESCENT;
    begin
        GETNEXT(TOKEN);
        PROGRAM(TOKEN);
        GETNEXT(TOKEN);
        if TOKEN≠'$'  then ERROR;
    end
```

2、非递归的预测分析

基于预测分析表
构造自动机

分析表

RG⇔FA

S \ Σ	a_1	a_2	…	a_n
s_1				
s_2				
…				
s_m				

DFA 表项中的状态唯一（确定的）

⇕

NFA 表项中的状态未必唯一（非确定的）

CFG⇔PDA

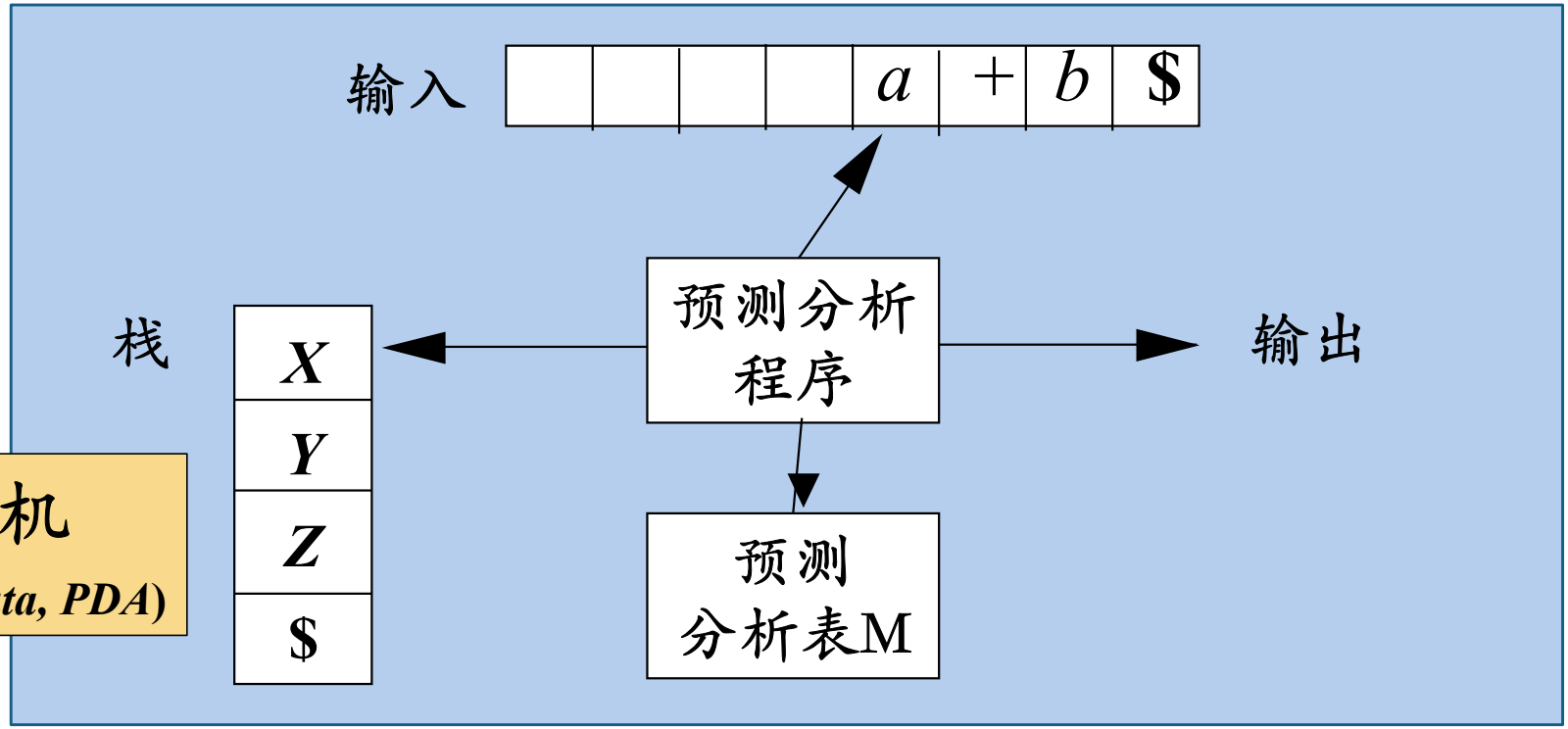
V_N \ V_T	a_1	a_2	…	a_n
A_1				
A_2				
…				
A_m				

表项中的产生式未必唯一（非确定的）

LL(1)文法 预测分析表 表项中的产生式唯一（确定的）

2、非递归的预测分析

- 非递归的预测分析不需要为每个非终结符编写递归下降过程，而是根据预测分析表构造一个自动机，也叫表驱动的预测分析



下推自动机

(*Push Down Automata, PDA*)

结束