语法分析

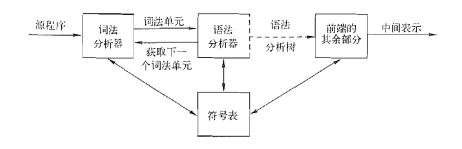
魏恒峰

hfwei@nju.edu.cn

2020年11月24日



输入: 词法单元流 & 语言的语法规则



输出: 语法分析树 (Parse Tree)

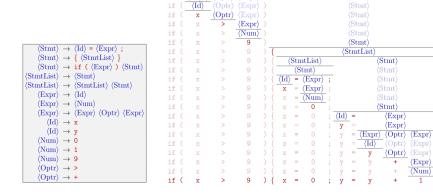
语法分析举例

(Optr) (Expr)

(Expr)

(Stmt)

(Stmt)



(Expr)

if (

(Expr)

(Num)

语法分析阶段的主题之一: 上下文无关文法

```
\langle \text{Stmt} \rangle \rightarrow \langle \text{Id} \rangle = \langle \text{Expr} \rangle;
            \langle Stmt \rangle \rightarrow \{ \langle StmtList \rangle \}
           \langle Stmt \rangle \rightarrow if (\langle Expr \rangle) \langle Stmt \rangle
\langle StmtList \rangle \rightarrow \langle Stmt \rangle
\langle StmtList \rangle \rightarrow \langle StmtList \rangle \langle Stmt \rangle
           \langle \text{Expr} \rangle \rightarrow \langle \text{Id} \rangle
           \langle \text{Expr} \rangle \rightarrow \langle \text{Num} \rangle
           \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle
                    \langle \mathrm{Id} \rangle \to \mathbf{x}
                    \langle \mathrm{Id} \rangle \to \mathbf{v}
            \langle \text{Num} \rangle \rightarrow 0
            \langle \text{Num} \rangle \rightarrow 1
            \langle \text{Num} \rangle \rightarrow 9
            \langle \text{Optr} \rangle \rightarrow >
            \langle \text{Optr} \rangle \rightarrow +
```

4/19

语法分析阶段的主题之二: 构建语法分析树

	$\langle \mathrm{Stmt} \rangle$														
if	((Expr))						(St	$\mathrm{mt}\rangle$				
if	(Kent (Expr)	(Optr)	(Expr))						(St	$\mathrm{mt}\rangle$				
if	$\langle Id \rangle$	(Optr)	(Expr)							(St	$\mathrm{mt}\rangle$				
if	(x	(Optr)	(Expr)							St	$\mathrm{mt} \rangle$				
if	(x	>	(Expr)							(St	$\mathrm{mt}\rangle$				
if	(x	>	(Num))	$\langle \mathrm{Stmt} \rangle$										
if	(x	>	9)	$\langle \mathrm{Stmt} \rangle$										
if	(x	>	9) {	(StmtList)									}	
if	(x	>	9		((StmtList)					($Stmt\rangle$		}	
if	(x	>	9			(Stmt)						Stmt			
if	(x	>	9		$\langle \mathrm{Id} \rangle$	=	(Expr)	;				Stmt			
if	(x	>	9		х	=	$\langle Expr \rangle$					Stmt			
if	(x	>	9		(x	=	(Num)					Stmt			
if	(x	>	9		x	=	0				($\operatorname{Stmt} \rangle$			
if	(x	>	9		(x			; ($\operatorname{Id} \rangle$	=		(Expr)		; }	
if	(x	>	9		(x			;	у	=		$\langle \text{Expr} \rangle$; }	
if	(x	>	9		(x	=			У	=	(Expr		(Expr)	; }	
if	(x	>	9		(x	=			у	=	$\langle \mathrm{Id} \rangle$	$\langle \text{Optr} \rangle$	$\langle \text{Expr} \rangle$; }	
if	(x	>	9		(x	=			у	=	у	$\langle \text{Optr} \rangle$	$\langle \text{Expr} \rangle$; }	
if	(x	>	9		(x	=			у	=	У	+	$\langle \text{Expr} \rangle$; }	
if	(x	>	9		(x	=				=		+	(Num)	; }	
if	(x	>	9) {	(x	=	0	;	у	=	у	+	1	; }	
											4 🗆 🕨	4 ♣ 4	∌ → ∢ ∌	→ 1	

语法分析阶段的主题之三: 错误恢复



报错、恢复、继续分析

"TALK IS CHEAP SHOW ME THE CODE"

- Linus Tarvalds

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$

 $S \rightarrow \text{begin } S L$
 $S \rightarrow \text{print } E$

$$L \rightarrow \text{end}$$

 $L \rightarrow ; S L$

$$E \rightarrow \text{num} = \text{num}$$

顺序语句、条件语句、打印语句

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$

$$S \rightarrow \text{begin } S L$$

$$S \rightarrow \text{print } E$$

$$L \rightarrow \text{end}$$

$$L \rightarrow \text{; } S L$$

$$E \rightarrow \text{num} = \text{num}$$

每个产生式都以一个终结符开头,且这些终结符各不相同

因此,仅根据输入中的当前词法单元,就可以确定应该使用哪条产生式

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$

$$S \rightarrow \text{begin } S L$$

$$S \rightarrow \text{print } E$$

$$L \rightarrow \text{end}$$

$$L \rightarrow \text{; } S L$$

$$E \rightarrow \text{num} = \text{num}$$

enum token {IF, THEN, ELSE, BEGIN, END, PRINT, SEMI, NUM, EQ};
extern enum token getToken (void);

getToken: 语法分析器按需向词法分析器索要下一个词法单元

```
L \rightarrow end
L \rightarrow ; S L
        S \rightarrow \text{if } E \text{ then } S \text{ else } S
        S \rightarrow \text{begin } S L
        S \rightarrow \text{print } E
                                                        E \rightarrow |\text{num}| = |\text{num}|
enum token tok:
void advance() {tok=getToken();}
void eat(enum token t) {if (tok==t) advance(); else error();}
           eat(t): 根据当前的产生式, <mark>预期</mark>的词法单元应该是 t
             匹配 t,继续试图匹配下一个词法单元;否则,报错
```

```
L \rightarrow \text{end}
L \rightarrow ; S L
        S \rightarrow \text{if } E \text{ then } S \text{ else } S
        S \rightarrow \text{begin } S L
        S \rightarrow \text{print } E
                                                  E \rightarrow |num| = num
void S(void) {switch(tok) {
           case IF: eat(IF); E(); eat(THEN); S();
                                               eat(ELSE); S(); break;
           case BEGIN: eat(BEGIN); S(); L(); break;
           case PRINT: eat(PRINT); E(); break;
           default: | error();
void L(void) {switch(tok) {
           case END: eat(END); break;
case SEMI: eat(SEMI); S(); L(); break;
           default: | error();
void E(void) { eat(NUM); eat(EQ); eat(NUM); }
```

为每个非终结符写一个递归函数

对于每个产生式, 写一个 case 分支语句

 $\mbox{if } 1=2 \mbox{ then begin print } 2=1; \mbox{ end else } 1=1 \\ \mbox{if num} = \mbox{num then begin print num} = \mbox{num}; \mbox{ end else num} = \mbox{num}$

if 1 = 2 then begin print 2 = 1; end else 1 = 1if num = num then begin print num = num; end else num = num



板书演示这个语法分析器的工作过程

```
enum token {IF, THEN, ELSE, BEGIN, END, PRINT, SEMI, NUM, EQ};
extern enum token getToken(void);
enum token tok:
void advance() {tok=getToken();}
void eat(enum token t) {if (tok==t) advance(); else error();}
eat(ELSE); S(); break;
        case BEGIN: eat(BEGIN); S(); L(); break;
        case PRINT: eat(PRINT); E(); break;
        default: error();
    L(void) {switch(tok) {
    case END: eat(END); break;
    case SEMI: eat(SEMI); S(); L(); break;
       default: |error();
void E(void) { eat(NUM); eat(EQ); eat(NUM); }
```

```
E 
ightarrow E + T \mid E - T \mid T T 
ightarrow T * F \mid T/F \mid F F 
ightarrow (E) \mid \mathbf{id} \mid \mathbf{num}
```

15/19

E 在**不消耗任何词法单元**的情况下, 直接递归调用 E, 造成**死循环**

$$E
ightarrow E + T \mid E - T \mid T$$
 $T
ightarrow T * F \mid T/F \mid F$ $F
ightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$

E 在**不消耗任何词法单元**的情况下, 直接递归调用 E, 造成死循环

$$E
ightarrow E + T \mid E - T \mid T$$
 $T
ightarrow T * F \mid T/F \mid F$ $F
ightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$

```
void E(void) {switch (tok) {
          case ?: E(); eat(PLUS); T(); break;
          case ?: E(); eat(MINUS); T(); break;
          case ?: T(); break;
          default: error();
})
```

更重要的是, E 与 T 的产生式所能生成的句子可能<mark>以相同的终结符开头</mark> 因此, 无法仅根据输入中当前的词法单元确定要使用的生成式

消除左递归

$$E \rightarrow E + T \mid T$$

消除左递归

$$E \rightarrow E + T \mid T$$

$$E \to TE'$$

 $E' \to + TE' \mid \epsilon$

将左递归转为右递归

$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

$$E o TE'$$
 $E' o + TE' \mid \epsilon$
 $T o FT'$
 $T' o * FT' \mid \epsilon$
 $F o (E) \mid \mathbf{id} \mid \mathbf{num}$

Thank You!



Office 926 hfwei@nju.edu.cn