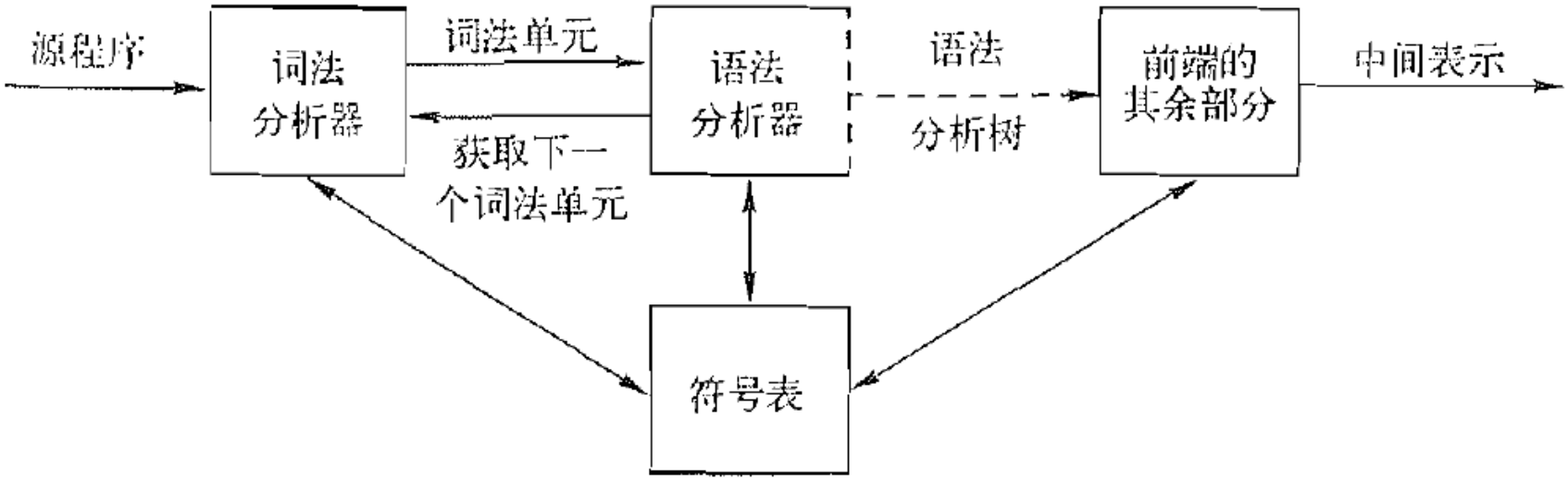
语法分析

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 24 日

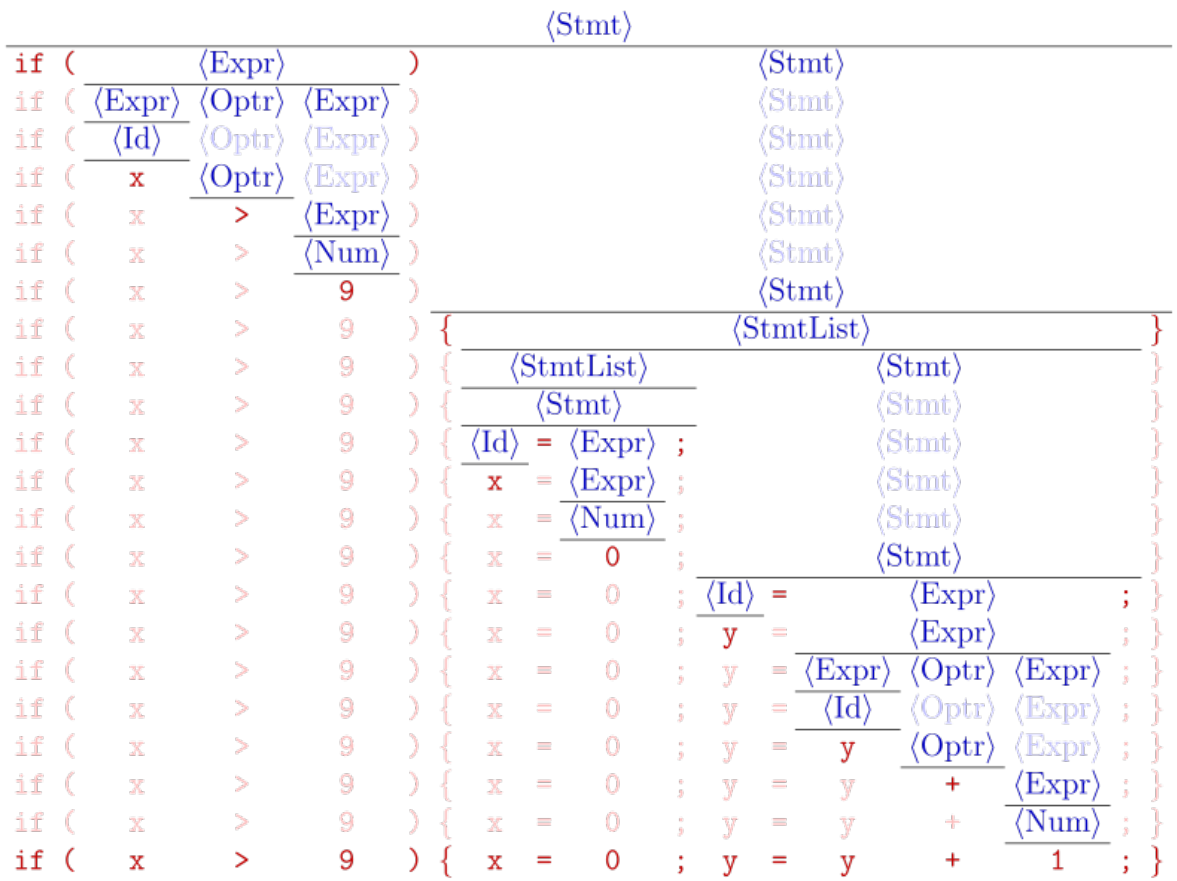
输入: 词法单元流 & **语言的语法规则**



输出: 语法分析树 (Parse Tree)

语法分析**举例**

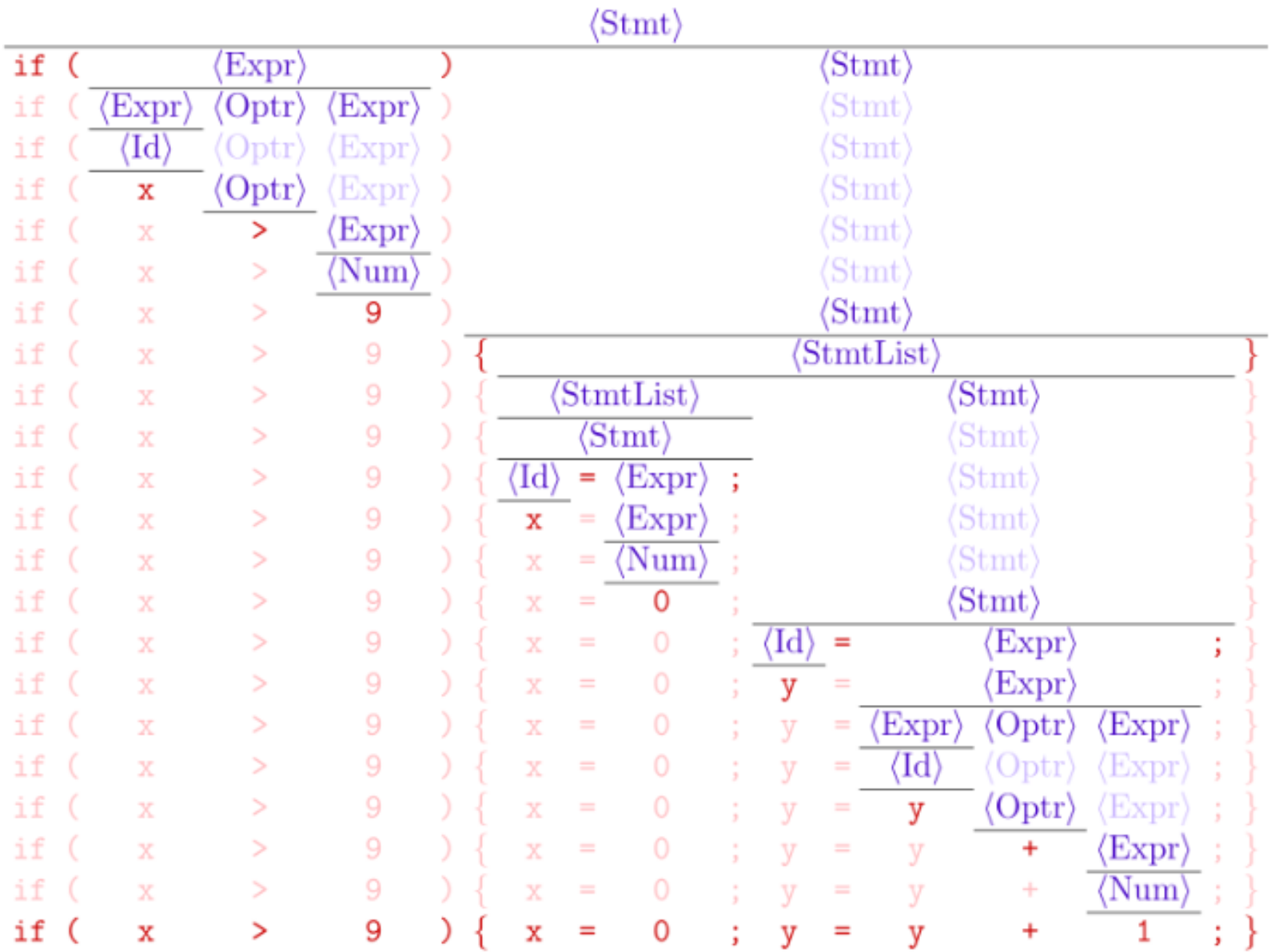
⟨Stmt⟩ → ⟨Id⟩ = ⟨Expr⟩ ;
⟨Stmt⟩ → { ⟨StmtList⟩ }
⟨Stmt⟩ → if (⟨Expr⟩) ⟨Stmt⟩
⟨StmtList⟩ → ⟨Stmt⟩
⟨StmtList⟩ → ⟨StmtList⟩ ⟨Stmt⟩
⟨Expr⟩ → ⟨Id⟩
⟨Expr⟩ → ⟨Num⟩
⟨Expr⟩ → ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
⟨Id⟩ → x
⟨Id⟩ → y
⟨Num⟩ → 0
⟨Num⟩ → 1
⟨Num⟩ → 9
⟨Optr⟩ → >
⟨Optr⟩ → +



语法分析阶段的主题之一：**上下文无关文法**

$$
\begin{aligned}
\langle \text{Stmt} \rangle &\rightarrow \langle \text{Id} \rangle \texttt{ = } \langle \text{Expr} \rangle \texttt{ ;} \\
\langle \text{Stmt} \rangle &\rightarrow \texttt{\{ } \langle \text{StmtList} \rangle \texttt{ \}} \\
\langle \text{Stmt} \rangle &\rightarrow \texttt{if (} \langle \text{Expr} \rangle \texttt{)} \ \langle \text{Stmt} \rangle \\
\langle \text{StmtList} \rangle &\rightarrow \langle \text{Stmt} \rangle \\
\langle \text{StmtList} \rangle &\rightarrow \langle \text{StmtList} \rangle \ \langle \text{Stmt} \rangle \\
\langle \text{Expr} \rangle &\rightarrow \langle \text{Id} \rangle \\
\langle \text{Expr} \rangle &\rightarrow \langle \text{Num} \rangle \\
\langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle \ \langle \text{Optr} \rangle \ \langle \text{Expr} \rangle \\
\langle \text{Id} \rangle &\rightarrow \texttt{x} \\
\langle \text{Id} \rangle &\rightarrow \texttt{y} \\
\langle \text{Num} \rangle &\rightarrow \texttt{0} \\
\langle \text{Num} \rangle &\rightarrow \texttt{1} \\
\langle \text{Num} \rangle &\rightarrow \texttt{9} \\
\langle \text{Optr} \rangle &\rightarrow \texttt{>} \\
\langle \text{Optr} \rangle &\rightarrow \texttt{+}
\end{aligned}
$$

语法分析阶段的主题之二：**构建语法分析树**



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 Torvalds

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

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

$S \rightarrow \text{print } E$

$L \rightarrow \text{end}$

$L \rightarrow ;\ S\ L$

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

顺序语句、条件语句、打印语句

$S \rightarrow$ if E then S else S

$S \rightarrow$ begin S L

$S \rightarrow$ print E

$L \rightarrow$ end

$L \rightarrow$; S L

$E \rightarrow$ num = 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

$S \rightarrow$ if E then S else S

$S \rightarrow$ begin S L

$S \rightarrow$ print E

$L \rightarrow$ end

$L \rightarrow$; S L

$E \rightarrow$ num = 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

$S \rightarrow$ if E then S else S

$S \rightarrow$ begin S L

$S \rightarrow$ print E

$L \rightarrow$ end

$L \rightarrow$; S L

$E \rightarrow$ num = 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, **预期**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

$S \rightarrow$ if E then S else S

$S \rightarrow$ begin S L

$S \rightarrow$ print E

$L \rightarrow$ end

$L \rightarrow$; S L

$E \rightarrow$ 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**

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

$$\textbf{if num} = \textbf{num then begin print num} = \textbf{num; end else num} = \textbf{num}$$

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

$$\textbf{if num} = \textbf{num then begin print num} = \textbf{num; end else num} = \textbf{num}$$

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();}

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); }
```

$$E \to E + T \mid E - T \mid T$$

$$T \to T * F \mid T/F \mid F$$

$$F \to (E) \mid \mathbf{id} \mid \mathbf{num}$$

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

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**

$$E \to E + T \mid E - T \mid T$$

$$T \to T * F \mid T/F \mid F$$

$$F \to (E) \mid \mathbf{id} \mid \mathbf{num}$$

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

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**

$$E \to E + T \mid E - T \mid T$$

$$T \to T * F \mid T/F \mid F$$

$$F \to (E) \mid \mathbf{id} \mid \mathbf{num}$$

```
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**以相同的终结符开头**

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

$$E \to E + T \mid T$$

消除左递归

$$E \to E + T \mid T$$

$$E \to TE'$$

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

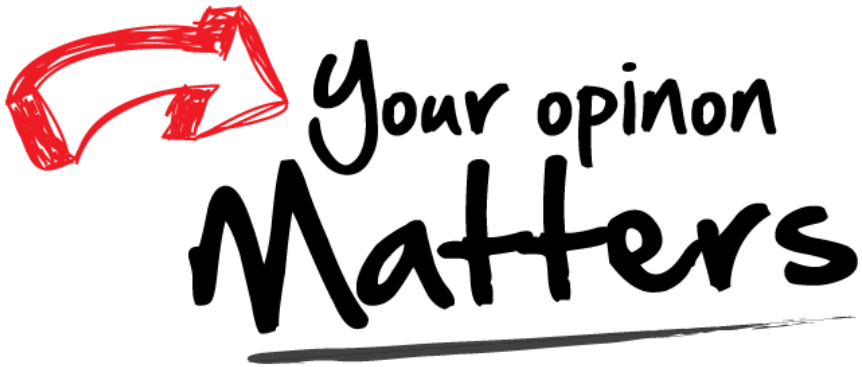
将左递归转为**右递归**

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

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

$$\begin{aligned}
E &\to TE' \\
E' &\to +\, TE' \mid \epsilon \\
T &\to FT' \\
T' &\to *\, FT' \mid \epsilon \\
F &\to (E) \mid \mathbf{id} \mid \mathbf{num}
\end{aligned}$$


Thank
You!



Office 926

hfwei@nju.edu.cn