Lecture 6

TeaPL语法设计

徐 辉
xuh@fudan.edu.cn

大纲

- 一、PEG语法
- 二、TeaPL语法设计
- 三、TeaPL语法解析

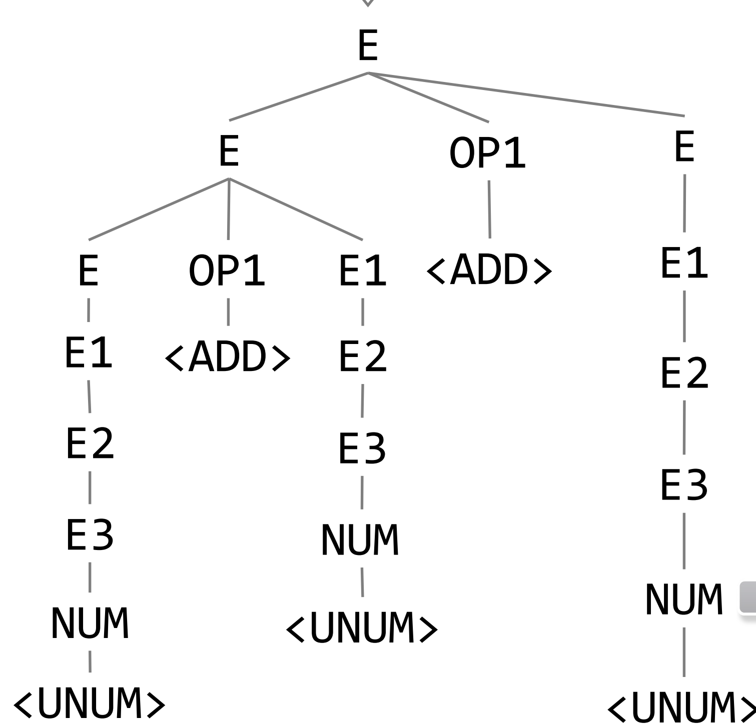
一、PEG语法

CFG的问题

- 语法复杂（避免二义性），易读性差
- 语法解析树复杂

```
[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]    | E2
[5] E2 → E3 OP3 E2
[6]    | E3
[7] E3 → NUM
[8]    | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]    | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]     | <SUB>
[13] OP2 → <MUL>
[14]     | <DIV>
[15] OP3 → <POW>
```

应用：1+2+3的语法解析树

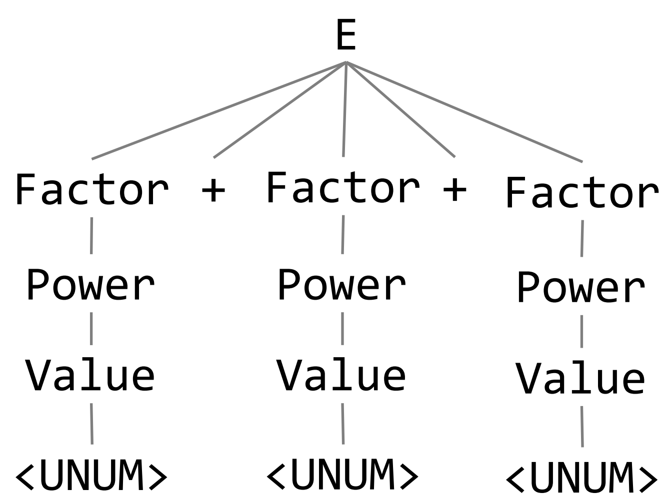


PEG: Parsing Expression Grammar

- 引入更多运算符，简化规则，如：
  - *：闭包
  - /：选择（PEG中的“/”对于匹配有顺序要求）
    - 为了方便起见，我们使用“|”，忽略匹配顺序要求
  - ''：终结符

```
[1] E → Factor (('+'|'-') Factor)*
[2] Factor → Power (('*'|'/') Power)*
[3] Power → Value ('^' Power)?
[4] Value → <UNUM> | ('-' <UNUM>) | ('(' E ')')
```

应用：1+2+3的语法解析树



二、TeaPL语法设计

使用PEG定于TeaPL：程序组成

```
program ⟼  (varDeclStmt | structDef | fnDeclStmt
             | fnDef | macro | comment)*
```

varDeclStmt	全局变量声明
structDef	数据结构定义
fnDeclStmt	函数声明
fnDef	函数定义
macro	宏
comment	注释

变量声明形式

`let a:int;`	变量声明
`let a:int = 0;`	声明时初始化
`let a;`	类型可省略
`let a = 0;`	
`let a[5]:int;`	支持数组类型
`let a[n]:int;`	
`let a[n];`	
`let a[2]:int = {0};`	数组声明时初始化
~~`let a[2]:int = {1, 2};`~~	

暂不支持：

- 二维数组：`let a[m][n];`
- 一条语句同时声明多个变量：`let i,j;`

变量声明

```
 varDeclStmt ⟼ let (varDecl | varDef) ';'

     varDecl ⟼ id (':' type)?
     varDecl ⟼ id '[' (id | num) ']' (':' type)?

      varDef ⟼ id (':' type)? '=' rightval
      varDef ⟼ id '[' (id | num) ']' (':' type)? '=' '{' num '}'

        type ⟼ primitiveType | structType | ptrType
primitiveType ⟼ int | bool | char | long | float | double
  structType ⟼ id
     ptrType ⟼ '*' type

   structDef ⟼ struct id '{' varDecl (, varDecl)* '}'
```

右值表达式

```
rightVal   ↦  arithExpr | boolExpr
arithExpr  ↦  factor (('+' | '-') factor)*
factor     ↦  power (('*' | '/') power)*
power      ↦  value ('^' power)?
value      ↦  num | id | fnCall | deref | addr | string | '(' rightVal ')'
value      ↦  id '.' id | id '[' (id | num) ']'
num        ↦  unum | ('-' unum)
deref      ↦  '*' id
addr       ↦  '&' id
string     ↦  '"' (num|id)* '"'
```

函数声明和定义

```
fn foo(a:int, b:int)->int;

fn foo(a:int, b:int)->int {
    return a + b;
}
```

函数声明

函数定义

```
fnDeclStmt  ⟼  fn fnSign ';'
    fnSign  ⟼  id '(' params ')'
    fnSign  ⟼  id '(' params ')' '->' type
    params  ⟼  (id ':' type (',' id ':' type)*) | ϵ
     fnDef  ⟼  fn fnSign codeBlock
 codeBlock  ⟼  '{' (stmt | codeBlock)* '}'
```

基本语句

```
      stmt  ↦  varDeclStmt | assignStmt | callStmt | retStmt |
               ifStmt | whileStmt | breakStmt | continueStmt
               forStmt | matchStmt

 breakStmt  ↦  break ';'
continueStmt ↦ continue ';'

assignStmt  ↦  leftVal '=' rightVal ';'
   leftVal  ↦  id | deref | id '[' (num | id) ']' | id '.' id

  callStmt  ↦  fnCall ';'
    fnCall  ↦  id '(' (rightVal (, rightVal)*) | ϵ ')'
   retStmt  ↦  ret (rightVal|ϵ) ';'

    ifStmt  ↦  if '(' boolExpr ')' codeBlock (else codeBlock)?
 whileStmt  ↦  while '(' boolExpr ')' codeBlock
```

条件表达式

```
boolExpr  ⟼ andExpr ('||' andExpr)*
 andExpr  ⟼ notExpr ('&&' notExpr)*
 notExpr  ⟼ '!'? (bitVal | '(' boolExpr ')')
  bitVal  ⟼ rightVal ('>' | '>=' | '<' | '<=' | '==' | '!=') rightVal
```

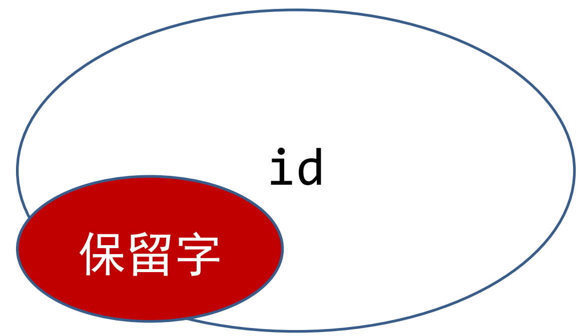
定义常量和标识符

$$\begin{aligned} \texttt{digits} &\mapsto \texttt{[0-9]}^{+} \\ \texttt{fraction} &\mapsto \texttt{.digits}|\epsilon \\ \texttt{unum} &\mapsto \texttt{digits fraction} \end{aligned}$$

$$\begin{aligned} \texttt{letter} &\mapsto \texttt{[a-zA-Z]} \\ \texttt{id} &\mapsto \texttt{letter (letter | digits)*} \end{aligned}$$

其它保留字词法定义

let	$\mapsto$	'let'
if	$\mapsto$	'if'
else	$\mapsto$	'else'
while	$\mapsto$	'while'
int	$\mapsto$	'int'
long	$\mapsto$	'long'
float	$\mapsto$	'float'
bool	$\mapsto$	'bool'
char	$\mapsto$	'char'
fn	$\mapsto$	'fn'
ret	$\mapsto$	'ret'
struct	$\mapsto$	'struct'
break	$\mapsto$	'break'
continue	$\mapsto$	'continue'



注释

- 注释内部的单词无需识别为单独的标签

```
comment  ↦  '//' (!newline)* newline
comment  ↦  '/*' (!'*/')* '*/'
newline  ↦  \r\n
```

三、语法解析

PEG解析方法

- 方法一：基于Flex和Bison，需要进行改写和适配
- 方法二：手写解析代码
- 方法三：专用PEG解析算法或工具，如Packrat parser

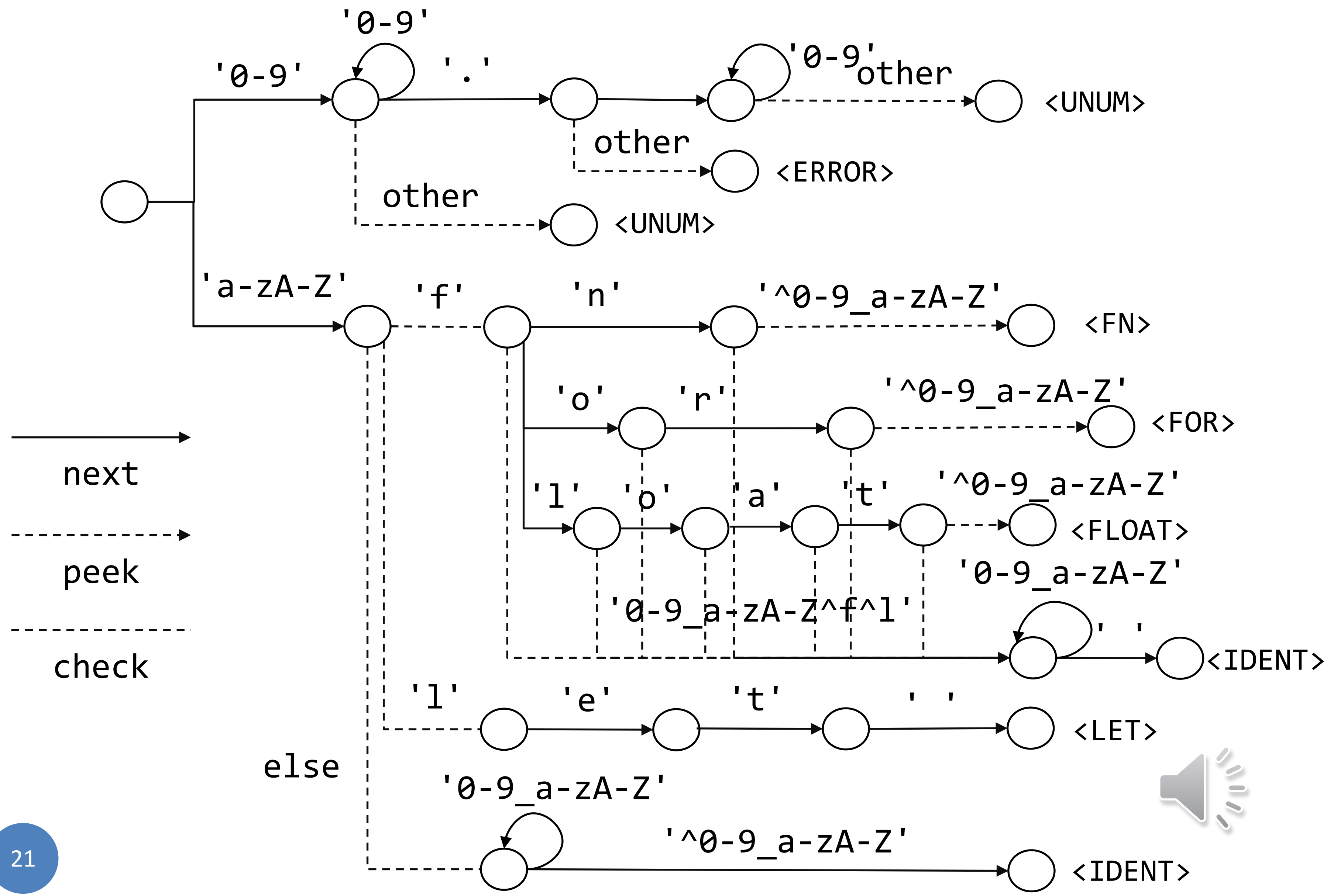
词法分析的冲突处理

- 多种匹配方案时，选择最长的匹配，如
  - '<='不应识别为'<'和'='，'ifabc'不应识别为<IF>和<IDENT>
  - Flex默认采用的规则
- 保留字优先级高于标识符，如
  - 'if'应识别为<IF>，非<IDENT>
  - Flex默认优先匹配先定义的标签

手写词法分析程序

```
cur = cstream.next();
match (cur.value) {
    '+' => tstream.add(SymToken::new(ADD,cur.pos));
    '-' => {
        if cstream.peek(1) == '>' {
            tstream.add(SymToken::new(RARROW, cur.pos));
            cur = cstream.next ();
        } else tstream.add(SymToken::new(SUB, cur.pos));
    'a'-'z' || 'A'-'Z' => {
        pos = cur.pos;
        char value[256];
        if (cur.value == 'f') {
                value[0] = 'f';
                if (cstream.peek(1) == 'n') {
                    if (!isAlphanum(cstream.peek(2))) {
                        tstream.add(SymToken::new(FN,cur.pos));
                } else if (cstream.peek(1) == 'o') { ...//for
                } else {
                    int i = 1;
                    ch = cstream.next().value;
                    while (isAlphanum(ch)) {
                        value[i] = ch;
                        ch = cstream.peek(++i).value
                    }
                    tstream.add(IdentTok::new(value,pos));
                }
    ...
```

识别数字和标识符



'0-9'
'0-9'
'.'
'0-9'
other
<UNUM>
other
<ERROR>
other
<UNUM>
'a-zA-Z'
'f'
'n'
'^0-9_a-zA-Z'
<FN>
'o'
'r'
'^0-9_a-zA-Z'
<FOR>
'l'
'o'
'a'
't'
'^0-9_a-zA-Z'
<FLOAT>
'0-9_a-zA-Z'
'0-9_a-zA-Z^f^l'
' '
<IDENT>
next
peek
check
'l'
'e'
't'
' '
<LET>
else
'0-9_a-zA-Z'
'^0-9_a-zA-Z'
<IDENT>

Token对象应记录哪些信息

```
enum Token {
    SymTok,
    IdentTok,
    UnumTok
}
```

```
struct Pos {
    line:int,
    col:int
}
```

```
struct SymTok {
    type : int, //1:ADD, 2:SUB,...
    pos : Pos
}
```

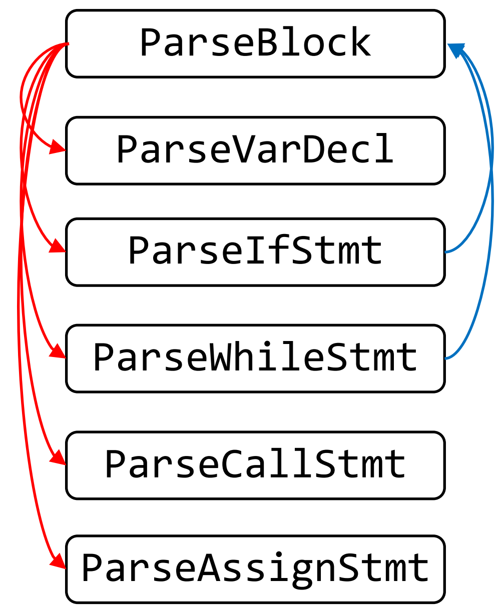
```
struct IdentTok {
    ident : *char,
    pos : Pos
}
```

```
struct UnumTok {
    unum : *char,
    pos : Pos
}
```

手写句法分析程序思路

- 递归下降LL(K)
  - 表达式解析可使用操作符优先级解析算法

```
ParseBlock(tstream) -> Node {
    cur = tstream.next();
    while (cur != tok::RBRACE) {
        match (cur.type) {
            tok::LET => ParseVarDecl(tstream);
            tok::IF => ParseIfStmt(tstream);
            tok::WHILE => ParseWhileStmt(tstream);
            tok::IDENT => {
                if(tstream.peek() == tok::LPAREN) {
                    ParseCallStmt(tstream);
                }
                else ...//
            }
        }
        cur = tstream.next();
    }
}
```



语法解析树节点类型

program ↦ (varDeclStmt | structDef | fnDeclStmt | fnDef | comment | ';')*

```
enum Node {
    varDeclStmt,
    structDef,
    fnDeclStmt,
    fnDef,
    ...
}
```

varDeclStmt ↦ let (varDecl | varDef) ';'

```
struct varDeclStmt {
    let : *SymTok,
    var : *varDeNode,
    colon : *SymTok
}
```

```
enum varDNode {
    varDecl,
    varDef
}
```