附录 A: TeaPL 语法标准

徐辉，xuh@fudan.edu.cn

A.1 TeaPL 简介

TeaPL 语言 (Teaching Programming Language) 是为编译原理课程教学设计的一门语言。该语言在使用方式上与 C 语言相似，但在一些语法上采用了 Rust 的形式，主要是便于语法解析和缺省类型实现等考虑。图 1展示了用 TeaPL 编写的阶乘函数代码。

```
fn factorial(n:int)->int {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

fn main() -> int {
    let r = factorial(n);
    return r;
}
```

图 1: TeaPL 代码样例

TeaPL 分为基础版和进阶版。基础版的类型系统和控制流功能都比较简单，适用于一个学期的编译课程开发项目。进阶版则在其基础上增加了浮点数、指针等类型和 `match-case` 等控制流功能，更接近实用的编程语言。

A.2 基本版

代码基本组成

program ↦ (varDeclStmt | fnDeclStmt | fnDef | structDef | comment | ';')* (1)

标识符和数字

id ↦ [a-zA-Z][a-zA-Z0-9]* (2)

num ↦ unum | ('-' unum) (3)

unum ↦ [0-9]+ (4)

变量声明

varDeclStmt ↦ 'let' (varDecl | varDef) ';' (5)

varDecl ↦ id (':' type)? | id '[' (id | num) ']' (':' type)? (6)

varDef ↦ id (':' type)? '=' rightVal (7)

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

类型

type ↦ primitiveType | structType (8)

primitiveType ↦ int (9)

structType ↦ id (10)

structDef ↦ 'struct' id '{' varDecl (, varDecl)* '}' (11)

右值

rightVal ↦ arithExpr (12)

arithExpr ↦ factor (('+' | '-') factor)* (13)

factor ↦ power (('*' | '/') power)* (14)

power ↦ exprUnit (15)

exprUnit ↦ num | id | fnCall | '(' rightVal ')' | id '.' id | id '[' (id | num) ']' (16)

函数声明

fnDeclStmt ↦ 'fn' fnSign ';' (17)

fnSign ↦ id '(' params? ')' '->' type? (18)

params ↦ id ':' type (',' id ':' type)* (19)

函数定义

```
       fnDef ↦ fn fnSign codeBlock                                              (20)
   codeBlock ↦ '{' stmt* '}'                                                    (21)
        stmt ↦ varDeclStmt | assignStmt | callStmt | retStmt | ifStmt           (22)
             | whileStmt | breakStmt | continueStmt
  assignStmt ↦ leftVal '=' rightVal ';'                                         (23)
     leftVal ↦ id | id '[' (num | id) ']' | id '.' id                           (24)
    callStmt ↦ fnCall ';'                                                       (25)
      fnCall ↦ id '(' ((rightVal (, rightVal)*))? ')'                           (26)
     retStmt ↦ 'ret' rightVal? ';'                                              (27)
      ifStmt ↦ 'if' '(' boolExpr ')' codeBlock (else codeBlock)?                (28)
   whileStmt ↦ 'while' '(' boolExpr ')' codeBlock                               (29)
   breakStmt ↦ 'break' ';'                                                      (30)
continueStmt ↦ 'continue' ';'                                                   (31)
```

布尔表达式

```
boolExpr ↦ andExpr ('||' andExpr)*                                              (32)
 andExpr ↦ notExpr ('&&' notExpr)*                                              (33)
 notExpr ↦ '!'? (bitVal | '(' boolExpr ')')                                     (34)
  bitVal ↦ exprUnit ('>' | '>=' | '<' | '<=' | '==' | '!=') exprUnit            (35)
```

代码注释

```
comment ↦ '//' (!newline)* newline | '/*' (!'*/')*' */'                        (36)
newline ↦ '\n'                                                                  (37)
```

A.3 进阶版

进阶版中增加的功能用红色标出。

代码基本组成

program ↦ (varDeclStmt | fnDeclStmt | fnDef | structDef | macro | comment | ';')* (38)

标识符和数字

id ↦ [a-zA-Z][a-zA-Z0-9]* (39)

num ↦ unum | ('-' unum) (40)

unum ↦ [0-9]+(.[0-9]+)? (41)

变量声明

varDeclStmt ↦ 'let' (varDecl | varDef) ';' (42)

varDecl ↦ id (':' type)? | id '[' (id | num) ']' (':' type)? (43)

varDef ↦ id (':' type)? '=' rightVal (44)

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

类型

type ↦ primitiveType | structType | ptrType (45)

primitiveType ↦ int | bool | char | long | float | double (46)

structType ↦ id (47)

structDef ↦ 'struct' id '{' varDecl (, varDecl)* '}' (48)

ptrType ↦'*' type (49)

右值

rightVal ↦ arithExpr | boolExpr (50)

arithExpr ↦ factor (('+' | '-') factor)* (51)

factor ↦ power (('*' | '/') power)* (52)

power ↦ exprUnit ('^' power)? (53)

exprUnit ↦ num | id | fnCall | '(' rightVal ')' | id '.' id (54)

↦ | id '[' (id | num) ']' | exprUnit | deref | addr | string (55)

deref ↦ '*' id (56)

addr ↦ '&' id (57)

string ↦ '"' (!'"')* '"' (58)

函数声明

fnDeclStmt ↦ 'fn' fnSign ';' (59)

fnSign ↦ id '(' params? ')' '->' type? (60)

params ↦ id ':' type (',' id ':' type)* (61)

函数定义

```
      fnDef ↦ fn fnSign codeBlock                                                    (62)
  codeBlock ↦ '{' (stmt | codeBlock)* '}'                                            (63)
       stmt ↦ varDeclStmt | assignStmt | callStmt | retStmt | ifStmt                (64)
            | whileStmt | breakStmt | continueStmt | forStmt | matchStmt
 assignStmt ↦ leftVal '=' rightVal ';'                                               (65)
    leftVal ↦ id | id '[' (num | id) ']' | id '.' id | deref                         (66)
   callStmt ↦ fnCall ';'                                                            (67)
     fnCall ↦ id '(' ((rightVal (, rightVal)*))? ')'                                (68)
    retStmt ↦ 'ret' rightVal? ';'                                                   (69)
     ifStmt ↦ 'if' '(' boolExpr ')' codeBlock (else codeBlock)?                     (70)
  whileStmt ↦ 'while' '(' boolExpr ')' codeBlock                                    (71)
  breakStmt ↦ 'break' ';'                                                           (72)
continueStmt ↦ 'continue' ';'                                                       (73)
    forStmt ↦'for' forCond codeBlock                                                (74)
    forCond ↦id 'in' ('[' (id | num) '..' (id | num) ']' | id)                      (75)
  matchStmt ↦'match' id '{' matchArm+ defaultArm? '}'                               (76)
   matchArm ↦rightVal '=>' codeBlock                                                (77)
 defaultArm ↦'_' '=>' codeBlock                                                     (78)
```

布尔表达式

```
boolExpr ↦ andExpr ('||' andExpr)*                                                  (79)
 andExpr ↦ notExpr ('&&' notExpr)*                                                  (80)
 notExpr ↦ '!'? (bitVal | '(' boolExpr ')')                                         (81)
  bitVal ↦ exprUnit ('>' | '>=' | '<' | '<=' | '==' | '!=') exprUnit                (82)
```

宏

```
macro ↦'macro!' id (id | num)                                                       (83)
```

代码注释

```
comment ↦ '//' (!newline)* newline | '/*' (!'*/')*' */'                            (84)
newline ↦ '\n'                                                                      (85)
```