合肥工业大学

系统软件综合设计报告
编译原理分册

设计题目 LR1 分析、Scheme 解释器

学生姓名 余梓俊

学　　号 2018211991

专业班级 计算机 18-3 班

指导教师 李宏芒 唐益明

完成日期 2021.7.15

目录

一、设计目的及设计要求

1. 设计目的

①目的

《编译原理》是计算机专业的一门重要课程，其中包含大量软件设思想。大家通过课程设计，实现一些重要的算法或个完整编译序模型能够进一步加深理解和掌握所学知识，对提高自己的软件设计水平具有十分重要意义[1]。

②要求

按照《编译原理课程设计指导书（含参考选题）》（2016 版）的有关要求完成算法设计、代码编写与调试以及课设报告的撰写。

2. 设计要求

题目：32. LR 分析器总控程序的实现

设计内容及要求：设计内容及要求：对 P.101 中的文法，按图 5.5LR 分析表构造 LR 分析器。 要求程序按 P.102 例 5.7 那样，对于输入串 i*i+i，输出 LR 分析器的工作过程。

实现结果：构造了 LR(0)分析总控程序和 LR(1)分析总控程序，均可以从产生式计算得到项目集和分析表，并输出输入串的分析过程。

题目：自拟. Python 实现 Scheme 语言子集的解释器

实现结果：实现了一个 Python 运行的 Scheme 语言的 REPL，支持最基本 Scheme 语句，例如局部绑定、高阶函数，提供了对 list 的操作，支持运行文件中的 Scheme 代码，可以简单地报告程序中的错误，不支持任何高级功能例如 macro、call/cc。

二、开发环境描述

OS: Ubuntu 20.04 focal(on the Windows Subsystem for Linux)

IDE: Visual Studio Code

Python Runtime: Python 3.8.10

JavaScript Runtime: V8 9.1.269.36

三、设计内容、主要算法描述

1. LR(1)相关原理与算法

First 构造：

对每一文法符号 X∈VT∪VN 构造 FIRST(X)，连续使用下面的规则，直至每个集合 FIRST 不再增大为止：

1) 若 X∈VT，则 FIRST(X)＝{X}。

2) 若 X∈VN，且有产生式 X→a…，则把 a 加入到 FIRST(X)中；若 X→ε也是一条产生式，则把ε也加到 FIRST(X)中。

例程. FIRST 集不动点算法

```
Foreach (nonterminal N)
  FIRST(N) = {}

While (some set is still changing)
  Foreach (production p: N->β1 ... βn)
    Foreach (βi from β1 upto βn)
      If βi == a
        FIRST(N) ∪= {a}
        Break
      If βi == M
        FIRST(N) ∪= FIRST(M)
          If M is not in NULLABLE
            Break
```

其中用到的，计算可以推出空串的集合的算法，亦即 NULLALBE 集合算法，伪代码如下：

例程. NULLALBE 集不动点算法

```
NULLABLE = {}
While (NULLABLE is still changing)
  Foreach (production p: X->β)
  If β == epsilon
    NULLABLE ∪= {X}
  If β == Y1 ... Yn
    If Y1 ∈ NULLABLE and ... and Yn ∈ NULLABLE
      NULLABLE ∪= {X}
```

Follow 构造：

对于文法 G 的每个非终结符 A 构造 FOLLOW(A)的办法是，连续使用下面的规则，直至每个 FOLLOW 不再增大为止：

1）对于文法的开始符号 S，置＃于 FOLLOW(S)中；

2）若 $A \rightarrow \alpha B \beta$是一个产生式，则把 FIRST(β)\{ε}加至 FOLLOW(B)中；

3）若 $A \rightarrow \alpha B$ 是一个产生式，或 $A \rightarrow \alpha B \beta$是一个产生式而$\beta \rightarrow \varepsilon$ （即 $\varepsilon \in FIRST(\beta)$），则把 FOLLOW(A)加至 FOLLOW(B)中。

例程. FOLLOW 集不动点算法

```
Foreach (nonterminal N)
  FOLLOW(N) = {}

While (some set is still changing)
  Foreach (production p: N->β1 ... βn)
    Temp = FOLLOW(N)
    Foreach (βi from βn downto β1)
      If βi == a
        Temp = {a}
      If βi == M
        FOLLOW(N) ∪= Temp
          If M is not in NULLABLE
            Temp = FIRST(M)
          Else temp ∪= FIRAT(M)
```

提取所有有效识别活前缀的式子：

形式上我们说一个LR(1)项目[A→α・β, a]对于活前缀γ是有效的，如果存在规范推导$S \Rightarrow \delta A\omega \Rightarrow \delta\alpha\beta\omega$其中，

1) γ＝δα；

2) a是ω的 第一个符号，或者a为#而ω为ε。

[A→α・Bβ, a]对活前缀γ＝δα是有效的，则对于每个形如B→ξ的产生式， 对任何b∈FIRST(βa)，[B→・ξ, b]对γ也是有效的。

项目集I 的闭包CLOSURE(I)构造：

1) I的任何项目都属于CLOSURE(I)。
2) 若项目[A→α・Bβ, a]属于CLOSURE(I)，B→ξ 是一个产生式，那么，对于FIRST(βa) 中的每个终结符b，如果[B→・ξ, b]原来不在CLOSURE(I)中，则把它加进去。
3) 重复执行步骤2，直至CLOSURE(I)不再增大为止。

GO构造：

令I是一个项目集，X是一个文法符号，函数GO(I，X)定义为：GO(I，X)＝CLOSURE(J)其中J＝{任何形如[A→αX・β, a]的项目| [A→α・Xβ, a]∈I}

例程. GO()函数

```
Goto(C, X)
  Temp = {}
  Foreach (C’s item i: A-α・Xβ)
    Temp ∪= {A→αX・β, a}
    Return closure(Temp)
```

例程. LR(0) closure()函数

```
Closure(C)
  While (C is still changing)
    Foreach (C’s item i: A-α・Xβ)
      C ∪= {X->...}
```

分析表构造：

令每个 Ik 的下标 k 为分析表的状态，令含有[S′→・S, #]的 Ik 的 k 为分析器的初态。

1） 若项目[A→α・aβ, b]属于 Ik 且 GO(Ik, a)＝Ij， a 为终结符，则置 ACTION[k, a]为 “sj”。
2） 若项目[A→α・，a]属于 Ik，则置 ACTION[k, a]为 “rj”；其中假定 A→α为文法 G′的第 j 个产生式。
3） 若项目[S′→S・, #]属于 Ik，则置 ACTION[k, #]为 “acc”。
4） 若 GO(Ik，A)＝Ij，则置 GOTO[k, A]=j。
5） 分析表中凡不能用规则 1 至 4 填入信息的空白栏均填上“出错标志”。

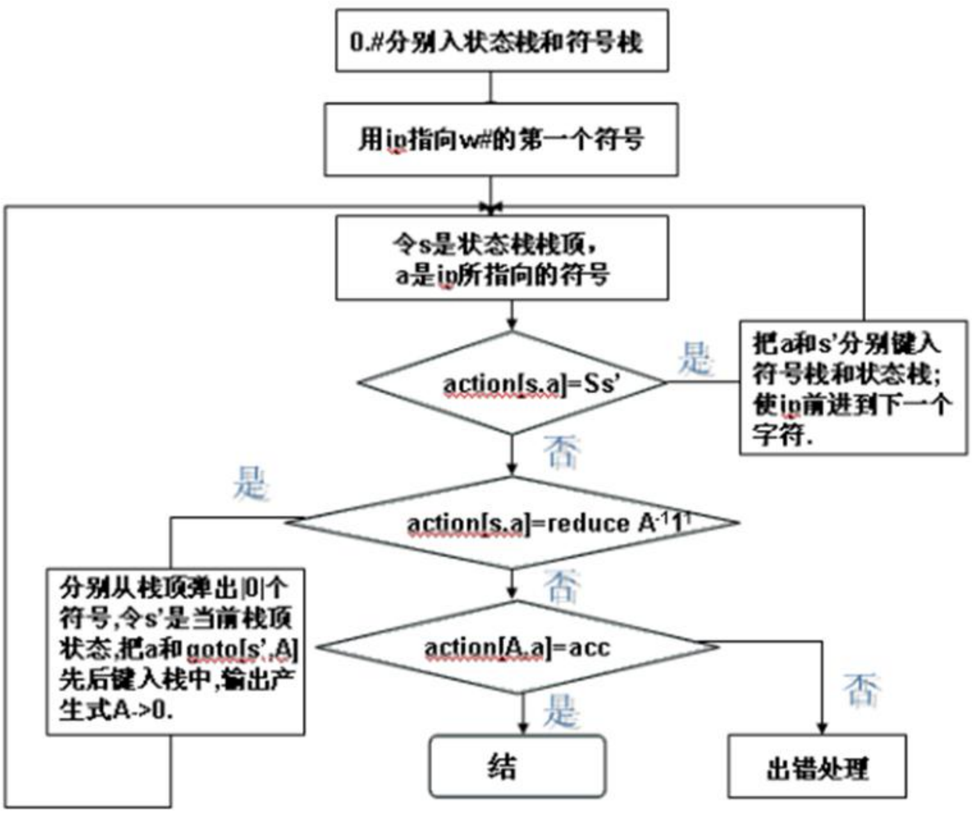
例程. LR 分析表构造算法

```
C0 = closure(S’->S$)
SET = {C0}
Q = enQueue(C0)
While (Q is not empty)
  C = deQueue(Q)
    Foreach (x ∈ (N∪T))
      D = goto(C, x)
    If (x ∈ N)
      ACTION[C, x] = D
    Else GOTO[C, x] = D
    If (D not ∈ SET)
      SET ∪= {D}
      enQeue(D)
```

分析过程：

例程. LR 分析过程

```
Stack = []
Push($)
Push(1)
While (true)
  Token = nextToken()
  State s = stack[top]
  If ACTION[s, t] == 'si'
    Push(t)
    Push(i)
  Elif ACTION[s, t] == 'ri'
    Pop(right hand of production j)
    State s = stack[top]
    Push(X)
    Push(GOTO[s, X])
  Else error
```



2. Scheme 解释器相关原理与算法

这一部分的解释器的实现主要参考《Structure and Interpretation of Computer Programs》一书 4.1 节中的元循环求值器。书中用 Lisp 语言实现了一个 Lisp 语言的解释器，并述：用与被求值的语言同样的语言写出的求值器被称为元循环。

在这里，我采用 Python 语言改写了这个求值器的原始实现。

由于 Lisp 家族的语言的语法采用了 S-expression，代码本身就与语法树一一对应，因此实现起来更加方便。例如：

- 变量：x
- 函数：(lambda (x) e)
- 绑定：(let ([x e1]) e2)
- 调用：(e1 e2)
- 算术：(• e2 e2)

（其中，• 是一个算术操作符，可以选择 +, -, *, / 其中之一）

更加具体一些：我们输入表达式 '(+ 1 2) ，它就输出值，整数 3。需要注意的是，表达式是一个数据结构，而不是一个字符串。我们用 S-expression 来存储表达式。比如表达式 '(+ 1 2) 其实是一个链表（list），它里面的内容是三个符号（symbol）：+, 1 和 2，而不是字符串"(+ 1 2)"。

要理解 Scheme 语言与解释器的实现，首先要理解 Scheme 代码的解释规则（evaluation rule），即环境模型（Environment Model）：

1. To evaluate a combination (a compound expression other than a special form), evaluate the subexpressions and then apply the value of the operator subexpression to the values of the operand subexpressions.
2. To apply a compound procedure to a set of arguments, evaluate the body of the procedure in a new environment. To construct this environment, extend the environment part of the procedure object by a frame in which the formal parameters of the procedure are bound to the arguments to which the procedure is applied.

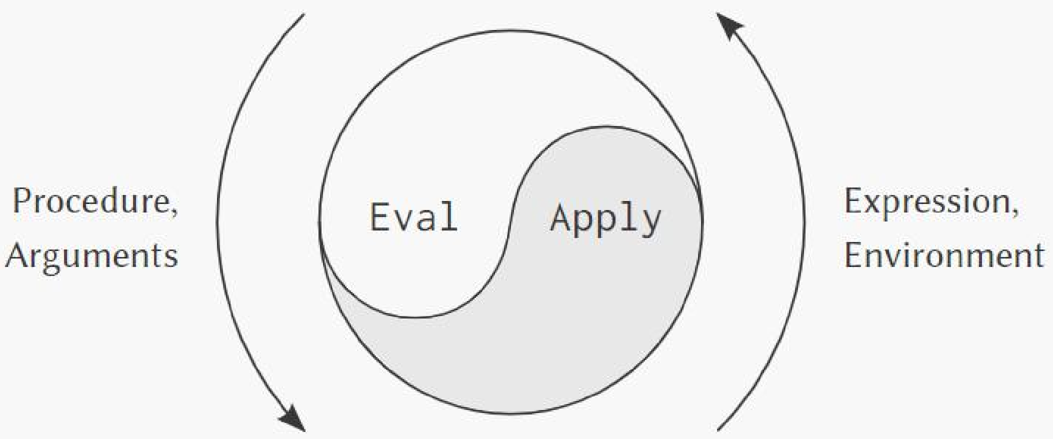


Figure 4.1: The eval-apply cycle exposes the essence of a computer language.

例程. eval

```
def meval(exp, env):
    # primitives
    if is_self_evaluating(exp):
        return exp
    if is_variable(exp):
        return lookup_variable_value(exp, env)
    # special forms
    if is_definition(exp):
        return eval_definition(exp, env)
    if is_lambda(exp):
        return make_procedure(lambda_parameters(exp), lambda_body(ex
p), env)
    if is_if(exp):
        return eval_if(exp, env)
    ...
    ...
    ...
    # combinations
    if is_application(exp):
        return mapply(meval(operator(exp), env),
                      [meval(e, env) for e in pair_to_list(operands(
exp))])
    raise Exception("Unknown expression type")
```

例程. apply

```
def mapply(procedure, arguments: list):
    if is_primitive_procedure(procedure):
        return primitive_proc_underlying_proc(procedure)(*arguments)
    elif is_compound_procedure(procedure):
        new_env = extend_environment(
            pair_to_list(procedure_parameters(procedure)),
            arguments,
            procedure_environment(procedure))
        return [meval(e, new_env) for e in procedure_body(procedure)
][-1]
```

eval 的工作即为对表达式求值。当表达式是过程调用（application）时，eval 分别对 operator 和 operands 求值，前者的结果是创建或查找到一个 procedure，后者的结果将作为这个 procedure 的 arguments；eval 接下来调用 apply，而 apply 的工作即为 Environment Model 中画新的环境框的动作（创建新环境，绑定 arguments 到 procedure 的 parameter 上），画完框之后工作又交回给 eval（在新环境中调用 eval 求值 procedure 的 body）。

在本实现中，表达式可以是如下几种类别：

例程. 表达式种类

```
exp -> int
     | float
     | str
     | bool
     | the_empty_list
     | Symbol
     | Pair (list of <exp>)
```

接下来我们看一下 if 表达式的求值过程，其他 special forms 的求值原理与此相同。if 表达式的语法如下：(if <e1> <e2> <e3>)。

例程. eval_if

```python
def eval_if(exp, env):
    if is_true(meval(if_predicate(exp), env)):
        return meval(if_consequent(exp), env)
    else:
        return meval(if_alternative(exp), env)

def is_if(exp):
    return exp.car == Symbol('if')

def if_predicate(exp):
    return exp.cdr.car

def if_consequent(exp):
    return exp.cdr.cdr.car

def if_alternative(exp):
    return exp.cdr.cdr.cdr.car
```

可以看到，对 if 表达式求值，就是对 e1 求值，若结果为真，则求值 e2 并将结果作为整个 if 表达式的返回值，反之求值 e3。

为了实现 S-expression，我定义了 Symbol 和 Pair 类。

例程. Symbol 类

```python
class Symbol:
    def __init__(self, name: str) -> None:
        self.name = name

    def __repr__(self) -> str:
        return self.name

    def __eq__(self, o: object) -> bool:
        return isinstance(o, Symbol) and self.name == o.name

    def __hash__(self) -> int:
        return self.name.__hash__()
```

在 Symbol 类中，通过定义__eq__函数和__hash__函数，使得 Symbol 类实例可以作为字典的键。在本实现中，环境模型的变量绑定，正是一个字典。

例程. Pair 类

```
class Pair:
    def __init__(self, car, cdr) -> None:
        self.car = car
        self.cdr = cdr

    def __iter__(self):
        pair = self
        while not isinstance(pair, TheEmptyList):
            yield pair.car
            pair = pair.cdr

    def __repr__(self) -> str:
        return '(' + ' '.join([str(e) for e in self]) + ')'
```

在 Pair 类中，通过定义__iter__函数，使得 Pair 类实例可迭代，从而可以快速地转换为 Python 的 list。

例程. Scheme 的 pair 与 Python 的 list 的转换

```
def pair_to_list(p: Pair) -> list:
    if is_null(p):
        return []
    return list(p)


def list_to_pair(lst: list) -> Pair:
    if not len(lst):
        return the_empty_list
    return Pair(lst[0], list_to_pair(lst[1:]))
```

前面已经提到，S-expression 和语法树一一对应，因此我们的实现中，scheme_read 函数既是一个词法分析器（Scanner），又是一个语法分析器。

例程. 词法、语法分析器的部分代码

```python
def scheme_read(f: BufferedStream):
    f.remove_whitespace()
    c = f.getc()
    if is_number(c, f.peek()):
        f.ungetc(c)
        return read_number(f)
    if c == '#':
        ...
    if is_initial(c):
        f.ungetc(c)
        return read_symbol(f)
    if c == '\"':
        return read_string(f)
    if c == '(':
        return read_pair(f)
    raise Exception("Unknown syntax")

def read_pair(f: BufferedStream) -> Pair or TheEmptyList:
    f.remove_whitespace()
    c = f.getc()
    if c == ')':
        return the_empty_list
    f.ungetc(c)
    car = scheme_read(f)
    f.remove_whitespace()
    cdr = read_pair(f)
    return Pair(car, cdr)
```

可以看到，这里实际上也存在着类似“eval-apply 元循环”的相互递归。scheme_read 读到左括号时调用 read_pair，read_pair 又调用 scheme_read。

最后，在这个 Python 程序进入时，我们需要开启一个驱动循环，不停地求值当前表达式，并将结果打印出来。

例程. driver_loop

```
def driver_loop():
    f = BufferedStream(sys.stdin)
    the_global_environment = setup_environment()
    while True:
        try:
            print('\n]=> ', end='', flush=True)
            result = meval(scheme_read(f), the_global_environment)
            print(';==>', result)
        except Exception as e:
            print("Error:", e)
        except KeyboardInterrupt:
            exit()
```

四、设计的输入和输出形式

1. 输入形式

（1） LR1 文法分析器的输入为若干条产生式，例如

例程. LR1 输入形式

```
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i

可以通过如下形式表示 epsilon
E->TG
G->+TG
G->
```

以及一个待分析的输入串。

（2） Scheme 解释器的输入为任意程序段。

2. 输出形式

（1）LR1 文法分析器的输出 ACTION 表、GOTO 表以及分析过程表。

（2）Scheme 解释器的输出为表达式的求值结果。

五、程序运行的结果

1. LR1 分析器运行结果

LR0 分析结果（成功）：

状态栈	符号栈	剩余串	动作
0	#	accd#	移进，状态1入栈
0 1	#a	ccd#	移进，状态4入栈
0 1 4	#ac	cd#	移进，状态4入栈
0 1 4 4	#acc	d#	移进，状态5入栈
0 1 4 4 5	#accd	#	规约，状态10入栈
0 1 4 4 10	#accA	#	规约，状态10入栈
0 1 4 10	#acA	#	规约，状态6入栈
0 1 6	#aA	#	规约，状态3入栈
0 3	#E	#	分析成功

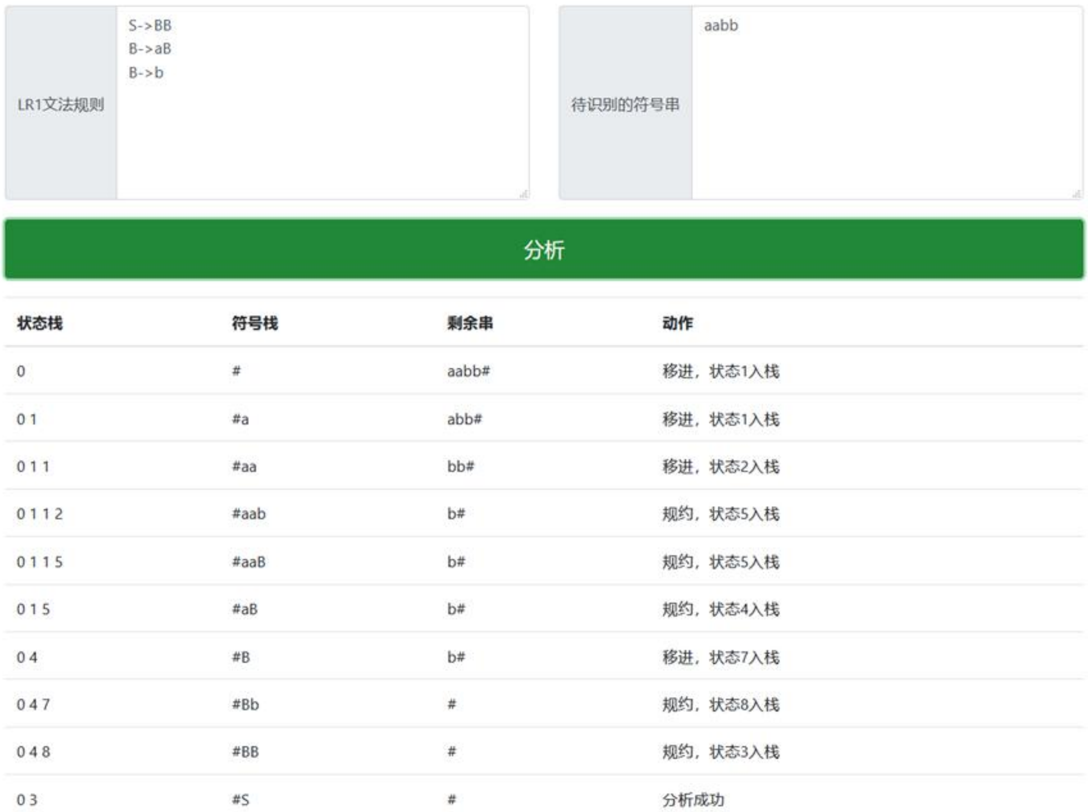
LR0 分析结果（失败）：

状态栈	符号栈	剩余串	动作
0	#	acdbcd#	移进，状态1入栈
0 1	#a	cdbcd#	移进，状态4入栈
0 1 4	#ac	dbcd#	移进，状态5入栈
0 1 4 5	#acd	bcd#	规约，状态10入栈
0 1 4 10	#acA	bcd#	规约，状态6入栈
0 1 6	#aA	bcd#	规约，状态3入栈
0 3	#E	bcd#	分析失败

LR1 分析结果（失败）：

状态栈	符号栈	剩余串	动作
0	#	abbb#	移进，状态1入栈
0 1	#a	bbb#	移进，状态2入栈
0 1 2	#ab	bb#	规约，状态5入栈
0 1 5	#aB	bb#	规约，状态4入栈
0 4	#B	bb#	移进，状态7入栈
0 4 7	#Bb	b#	分析失败

LR1 分析结果（成功）：



状态栈	符号栈	剩余串	动作
0	#	aabb#	移进，状态1入栈
0 1	#a	abb#	移进，状态1入栈
0 1 1	#aa	bb#	移进，状态2入栈
0 1 1 2	#aab	b#	规约，状态5入栈
0 1 1 5	#aaB	b#	规约，状态5入栈
0 1 5	#aB	b#	规约，状态4入栈
0 4	#B	b#	移进，状态7入栈
0 4 7	#Bb	#	规约，状态8入栈
0 4 8	#BB	#	规约，状态3入栈
0 3	#S	#	分析成功

ACTION 表：

0 4 7	#Bb	#	规约，状态8入栈
0 4 8	#BB	#	规约，状态3入栈
0 3	#S	#	分析成功

ACTION	#	a	b
0		s1	s2
1		s1	s2
2		r3	r3
3	accept		
4		s6	s7
5		r2	r2
6		s6	s7
7	r3		
8	r1		
9	r2		

GOTO 表：

9	r2

GOTO	S	B
0	3	4
1		5
2		
3		
4		8
5		
6		9
7		
8		
9		

出错处理：

输入格式错误
×
LR1文法规则
E->E+T
E-
提示：
1. 可推出 epsilon 写法为：N->
2. 区分大小写
3. 一条规则一行
4. 不需要额外添加第 0 条规则（N'->N）
Close
状态栈
符号栈
剩余串
动作

2. Scheme 解释器运行结果

简单表达式：

```
~/code/python/pyscheme master !2 ?1 ❯ python repl.py

]=> 1
;==> 1

]=> #t
;==> True

]=> (and 1 #f)
;==> False

]=> (+ 1 2 3 4)
;==> 10

]=> (cons 1 (cons 2 (cons 3 ())))
;==> (1 2 3)

]=> #\a
;==> a

]=> #\space
;==>
```

读取文件中的代码：

sample.scm

```scheme
(define (fib n)
  (if (<= n 1)
      1
      (+ (fib (- n 1))
         (fib (- n 2)))))


(define (sqrt x)
  (define (try guess old-guess)
    (if (good-enough? guess old-guess)
        guess
        (try (improve guess) guess)))

  (define (good-enough? guess old-guess)
    (= guess old-guess))

  (define (improve guess)
    (/ (+ guess (/ x guess)) 2))

  (try 1.0 x))

```

```
~/code/python/pyscheme master ?1 ❯ python repl.py

]=> (load "sample.scm")
;==> ok

]=> (fib 2)
;==> 2

]=> (fib 3)
;==> 3

]=> (fib 4)
;==> 5

]=> (sqrt 9)
;==> 3.0

]=> (sqrt 10)
;==> 3.162277660168379
```

高阶函数与 list 操作：

```
~/code/python/pyscheme master !1 ?1 python repl.py

]=> (define (map proc l)
  (if (null? l)
      nil
      (cons (proc (car l))
            (map proc (cdr l)))))
;==> ok

]=>
(define (one-to-n n)
  (define (iter i)
    (if (= i n)
        (cons i ())
        (cons i (iter (+ i 1)))))
  (iter 1))
;==> ok

]=> (let ((one-to-5 (one-to-n 5)))
  (map (lambda (x) (* x x))
       l))
Error: Unbound variable: l

]=> (let ((one-to-5 (one-to-n 5)))
  (map (lambda (x) (* x x))
       one-to-5))
;==> (1 4 9 16 25)

]=> 
```

上图中也可以看到错误处理：Error: Unbound variable: l。

其他错误处理：

```
~/code/python/pyscheme master !2 ?1 ❯ python repl.py

]=> a
Error: Unbound variable: a

]=> 1/3
Error: Rational not implemented

]=> (define (f x) x)
;==> ok

]=> (f 1 2 3)
Error: Too many arguments

]=> #abcd
Error: Boolean value must be #t or #f, not #a

]=> Error: Unbound variable: bcd

]=> #\abcd
Error: Invalid character ab...
```

六、总结体会

在这次课程设计中，我温习了一年之前的编译原理课程中教授的编译器前端的知识，回顾了编译器的整体结构，词法分析、自顶向下与自底向上的文法分析等等。

课程设计中的 LR1 文法器部分是对去年的实验代码的重构和完善，在编码过程中我也认识到了过去的自己的编码中存在的问题，尤其是整个程序的结构。

Scheme 解释器部分是对前段时间自学的计算机经典教材《Structure and Interpretation of Computer Programs》的一次简单的回顾。书中的第四章用 Scheme 实现了几个 Scheme 语言的变体的求值器，例如惰性求值、非确定性计算等等。此次课程设计，通过用 Python 重写书中最基本版本的求值器，再一次加深了我对编程语言与解释器的理解，以及对解释与编译之间的区别的认识。

七、源程序清单

1. GitHub 链接

https://github.com/wine99/hfut-cs-assignments/tree/master/%E7%BC%96%E8%AF%91%E5%8E%9F%E7%90%86/lr1

https://github.com/wine99/pyscheme

2. LR1 分析器源码

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>LR(1)文法分析</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/css/bootstrap.min.css"
    integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
  <style type="text/css">
    .container {
      margin-top: 100px;
    }

    .row {
      margin-bottom: 20px;
    }

    .form-control {
      min-height: 200px;
    }

    .btn-row {
      justify-content: center;
    }

    .btn {
      width: 100%;
      height: 60px;
      font-size: 1.3rem;
    }

    .col-middle {
      padding-left: 0;
      padding-right: 0;
    }

    .col-left {
      padding-right: 0;
    }
```

```
    .col-left {
      padding-left: 0;
    }
  </style>
</head>

<body>
  <div id="app">

    <div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-labe
lledby="exampleModalLabel"
      aria-hidden="true">
      <div class="modal-dialog" role="document">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">{{errorMsg}}</h5>
            <button type="button" class="close" data-dismiss="modal" aria-label="Cl
ose">
              <span aria-hidden="true">&times;</span>
            </button>
          </div>
          <div class="modal-body">
            提示：<br>1. 可推出 epsilon 写法为：N-> <br>2. 区分大小写 <br>3. 一条规则一
行 <br>4. 不需要额外添加第 0 条规则（N'->N）
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-dismiss="modal">Cl
ose</button>
          </div>
        </div>
      </div>
    </div>

    <div class="container">

      <div class="row">
        <div class="col">
          <div class="input-group">
            <div class="input-group-prepend">
              <span class="input-group-text">LR1 文法规则</span>
            </div>
            <textarea class="form-control" aria-label="LR1 文法规则
" v-model="rules">{{rules}}</textarea>
```

```
          </div>
        </div>

        <div class="col">
          <div class="input-group">
            <div class="input-group-prepend">
              <span class="input-group-text">待识别的符号串</span>
            </div>
            <textarea class="form-control" aria-label="待识别的符号串
" v-model="inputString">{{inputString}}</textarea>
          </div>
        </div>
      </div>

      <div class="row btn-row">
        <div class="col">
          <button type="button" class="btn" :class="btnState" @click="start">分析
</button>
        </div>
      </div>

      <div class="row">
        <div class="col">
          <table class="table">
            <thead>
              <tr>
                <th scope="col">状态栈</th>
                <th scope="col">符号栈</th>
                <th scope="col">剩余串</th>
                <th align="center" scope="col">动作</th>
              </tr>
            </thead>
            <tbody>
              <tr v-for="step in steps">
                <td>{{step.statusStack}}</td>
                <td>{{step.tokenStack}}</td>
                <td>{{step.leftString}}</td>
                <td align="center">{{step.action}}</td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
```

```
    <div class="row" v-if="itemSetIds.length">
      <div class="col">
        <table class="table">
          <thead>
            <tr>
              <th scope="col" :class="actionTableWidth">ACTION</th>
              <th scope="col" :class="actionTableWidth" v-for="terminal in termin
als">{{terminal}}</th>
            </tr>
          </thead>
          <tbody>
            <tr v-for="itemSetId in itemSetIds">
              <th scope="row" :class="actionTableWidth">{{itemSetId}}</th>
              <td :class="actionTableWidth" v-for="terminal in terminals">
                {{actionTable[itemSetId][terminal]}}
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>

    <div class="row" v-if="itemSetIds.length">
      <div class="col">
        <table class="table">
          <thead>
            <tr>
              <th scope="col" :class="gotoTableWidth">GOTO</th>
              <th scope="col" :class="gotoTableWidth" v-for="nonTerminal in nonTe
rminals">{{nonTerminal}}</th>
            </tr>
          </thead>
          <tbody>
            <tr v-for="itemSetId in itemSetIds">
              <th scope="row" :class="gotoTableWidth">{{itemSetId}}</th>
              <td :class="gotoTableWidth" v-for="nonTerminal in nonTerminals">
                {{gotoTable[itemSetId][nonTerminal]}}
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
```

```
    </div>
  </div>
</body>

<script>

  function getLeftHand(rule) {
    return rule.substring(0, rule.indexOf('->'));
  }

  function getRightHand(rule) {
    return rule.substring(rule.indexOf('->') + 2);
  }

  function checkRules(rules) {
    return rules.findIndex(rule => !rule.includes('->')) !== -1 ?
      false : true;
  }

  function getSetsTotalSize(sets) {
    return Object.values(sets)
      .map(set => set.size)
      .reduce((prev, curr) => prev + curr);
  }

  function appendSet(setA, setB) {
    setB.forEach(item => { setA.add(item) });
  }

  function initToken(rules) {
    let nonTerminals = new Set();
    let terminals = new Set();
    terminals.add('#');
    for (const rule of rules) {
      nonTerminals.add(getLeftHand(rule));
    }
    for (const rule of rules) {
      getRightHand(rule).split('').forEach(token => {
        if (!nonTerminals.has(token))
          terminals.add(token);
      })
    }

    return [nonTerminals, terminals]
```

```
}

function getNullables(rules) {
  let nullables = new Set();
  let size = 0;
  let newSize = 0;
  do {
    size = nullables.size;
    for (const rule of rules) {
      const nonTerminal = getLeftHand(rule);
      const rightHand = getRightHand(rule)
      if (rightHand === '') nullables.add(nonTerminal);
      else {
        let nullable = true;
        for (const token of rightHand.split('')) {
          if (!nullables.has(token)) {
            nullable = false;
            break;
          }
        }
        if (nullable) nullables.add(nonTerminal);
      }
    }
    newSize = nullables.size;
  } while (newSize !== size)

  return nullables;
}

function getFirst(rules, nonTerminals, terminals, nullables) {
  const first = {};
  for (const nonTerminal of nonTerminals) {
    first[nonTerminal] = new Set();
  }
  let size = 0;
  let newSize = 0;
  do {
    size = getSetsTotalSize(first);
    for (const rule of rules) {
      const nonTerminal = getLeftHand(rule);
      for (const token of getRightHand(rule).split('')) {
        if (terminals.has(token)) {
          first[nonTerminal].add(token);
          break;
```

```
        }
        else {
          appendSet(first[nonTerminal], first[token])
          if (!nullables.has(token)) break;
        }
      }
    }
    newSize = getSetsTotalSize(first);
  } while (newSize !== size)

  return first;
}

function getFollow(rules, nonTerminals, terminals, nullables, first) {
  const follow = {};
  for (const nonTerminal of nonTerminals) {
    follow[nonTerminal] = new Set();
  }
  follow[rules[0][0]].add('#');
  let size = 0;
  let newSize = 0;
  do {
    size = getSetsTotalSize(follow);
    for (const rule of rules) {
      const nonTerminal = getLeftHand(rule);
      let temp = new Set(follow[nonTerminal]);
      for (const token of getRightHand(rule).split('').reverse()) {
        if (terminals.has(token)) {
          temp = new Set([token]);
        }
        else {
          appendSet(follow[token], temp);
          if (nullables.has(token)) appendSet(temp, first[token]);
          else temp = new Set(first[token]);
        }
      }
    }
    newSize = getSetsTotalSize(follow);
  } while (newSize !== size)

  return follow;
}

function addDotToRule(rule) {
```

```
  const startIndex = rule.indexOf('->') + 2;
  return rule.substring(0, startIndex) +
    '`' + rule.substring(startIndex);
}

function getTokenAfterDot(item) {
  return item[item.indexOf('`') + 1] || '';
}
function getTokenAfterAfterDot(item) {
  return item[item.indexOf('`') + 2] || '';
}

function moveDotToNext(item) {
  const index = item.indexOf('`');
  if (index === item.length - 1) return item;
  return `${item.substring(0, index)}${item.charAt(index + 1)}` +
    `\`${item.substring(index + 2)}`;
}

function startLR1(rules, inputString) {
  if (inputString[inputString.length - 1] !== '#')
    inputString = inputString + '#';
  if (!checkRules(rules)) {

    return ['format', [], []];
  }

  const originStart = rules[0][0];
  rules.unshift(`${originStart}'->${originStart}`);

  const [nonTerminals, terminals] = initToken(rules);
  const nullables = getNullables(rules);
  const leftString = inputString.split('');
  const statusStack = ['0'];
  const tokenStack = ['#'];
  const steps = [];

  const first = getFirst(rules, nonTerminals, terminals, nullables);
  const follow = getFollow(rules, nonTerminals, terminals, nullables, first);

  const [actionTable, gotoTable, itemSets, hasConflicts] = getTable();
  if (hasConflicts)
    return ['notLR1', steps, []];
```

```
    while (true) {
      steps.push({
        statusStack: statusStack.join(' '),
        tokenStack: tokenStack.join(''),
        leftString: leftString.join(''),
      });

      const token = leftString[0];
      const status = statusStack[statusStack.length - 1];
      if (actionTable[status][token] === undefined) {
        steps[steps.length - 1].action = '分析失败';
        return ['fail', steps, [nonTerminals, terminals, actionTable, gotoTable, it
emSets.size]];
      }
      else if (actionTable[status][token][0] === 's') {
        leftString.shift();
        tokenStack.push(token);
        const pushingStatus = actionTable[status][token].substring(1);
        statusStack.push(pushingStatus);
        steps[steps.length - 1].action = `移进，状态${pushingStatus}入栈`;
      }
      else if (actionTable[status][token][0] === 'r') {
        const reduceRuleIndex =
          Number(actionTable[status][token].substring(1));
        const reduction = getLeftHand(rules[reduceRuleIndex]);
        const popLength = getRightHand(rules[reduceRuleIndex]).length;
        for (let i = 0; i < popLength; ++i) {
          statusStack.pop();
          tokenStack.pop();
        }
        tokenStack.push(reduction);
        const currStatus = statusStack[statusStack.length - 1];
        const pushingStatus = gotoTable[currStatus][reduction];
        if (pushingStatus === undefined) {
          steps[steps.length - 1].action = '分析失败';
          return ['fail', steps, [nonTerminals, terminals, actionTable, gotoTable,
itemSets.length]];
        }
        statusStack.push(pushingStatus);
        steps[steps.length - 1].action = `规约，状态${pushingStatus}入栈`;
      }
      else /* if (actionTable[status][token].toLowerCase() === 'accept') */ {
        steps[steps.length - 1].action = '分析成功';
```

```
      return [undefined, steps, [nonTerminals, terminals, actionTable, gotoTable,
itemSets.length]];
    }
  }


  function getTable() {
    let hasConflicts = false;
    const originNonTerminals = new Set(nonTerminals);
    originNonTerminals.delete(`${originStart}'`);

    const actionTable = {};
    const gotoTable = {};
    const itemSets = [];
    const itemSetQueue = [];

    const firstItemSet = {
      id: '0',
      items: closure(new Set([{
        string: addDotToRule(rules[0]),
        followed: new Set(['#']),
      }])),
    };
    itemSets.push(firstItemSet);
    itemSetQueue.push(firstItemSet);

    while (itemSetQueue.length) {
      const currItemSet = itemSetQueue.shift();
      if (actionTable[currItemSet.id] === undefined)
        actionTable[currItemSet.id] = {};
      if (gotoTable[currItemSet.id] === undefined)
        gotoTable[currItemSet.id] = {};

      let pointingItemSet = null;
      for (const terminal of terminals) {
        pointingItemSet = go(currItemSet, terminal);
        /* shift */
        if (pointingItemSet)
          actionTable[currItemSet.id][terminal] =
            `s${pointingItemSet.id}`;
      }

      for (const nonTerminal of originNonTerminals) {
        pointingItemSet = go(currItemSet, nonTerminal);
        /* goto */
```

```
    if (pointingItemSet)
      gotoTable[currItemSet.id][nonTerminal] =
        pointingItemSet.id;
  }
  for (item of currItemSet.items) {
    if (item.string[item.string.length - 1] === '`') {
      const reductionRule = rules.findIndex(rule =>
        rule === item.string.substring(0, item.string.length - 1));
      /* reduce */
      if (reductionRule !== 0) {
        for (const followed of item.followed) {
          if (actionTable[currItemSet.id][followed])
            hasConflicts = true;
          else actionTable[currItemSet.id][followed] =
            `r${reductionRule}`;
        }
      }
      /* accept */
      else actionTable[currItemSet.id]['#'] = 'accept';
    }
  }
}

return [actionTable, gotoTable, itemSets, hasConflicts];


function go(itemSet, token) {
  let tempSet = new Set();
  for (const item of itemSet.items) {
    if (getTokenAfterDot(item.string) === token)
      tempSet.add({
        string: moveDotToNext(item.string),
        followed: item.followed,
      });
  }

  if (!tempSet.size) return null;
  tempSet = closure(tempSet);
  const existedSetId = findSameSet(tempSet);
  if (existedSetId === -1) {
    const newItemSet = {
      id: String(itemSets.length),
      items: tempSet,
    };
    itemSets.push(newItemSet);
```

```
    itemSetQueue.push(newItemSet);

    return newItemSet;
  }
  else return itemSets.find(itemSet => itemSet.id === existedSetId);
}

function closure(items) {
  let size = 0;
  let newSize = 0;
  do {
    size = items.size;
    const _items = new Set(items);
    for (const item of items) {
      const tokenAfterDot = getTokenAfterDot(item.string);
      const tokenAfterAfterDot = getTokenAfterAfterDot(item.string);
      if (nonTerminals.has(tokenAfterDot)) {
        for (const rule of rules) {
          if (rule[0] === tokenAfterDot)
            addNewItem(
              addDotToRule(rule),
              getFollowed(tokenAfterAfterDot, item.followed),
              _items
            );
        }
      }
    }
    items = _items;
    newSize = items.size;
  } while (size !== newSize)

  return items;
}

function getFollowed(tokenAfterAfterDot, originFollowed) {
  if (tokenAfterAfterDot === '') return new Set(originFollowed);
  if (terminals.has(tokenAfterAfterDot))
    return new Set([tokenAfterAfterDot]);
  const temp = new Set(first[tokenAfterAfterDot]);
  if (nullables.has(tokenAfterAfterDot)) appendSet(temp, originFollowed);
  return temp;
}

function addNewItem(string, followed, items) {
```

```
        for (const item of items) {
          if (item.string === string) {
            appendSet(item.followed, followed);
            return;
          }
        }
        items.add({ string, followed });
      }

      function findSameSet(items) {
        const _items = mergeStringAndFollowed(items);
        for (const itemSet of itemSets) {
          const _difference = mergeStringAndFollowed(itemSet.items);
          for (const elem of _items) {
            if (_difference.has(elem)) {
              _difference.delete(elem)
            } else {
              _difference.add(elem)
            }
          }
          if (_difference.size === 0) return itemSet.id;
        }
        return -1;

        function mergeStringAndFollowed(items) {
          const temp = new Set();
          items.forEach(item => temp.add(
            `${item.string}${Array.from(item.followed).sort().join('')}`
          ));
          return temp;
        }
      }
    }
  }
</script>
<script src="https://cdn.jsdelivr.net/npm/jquery@3.4.1/dist/jquery.slim.min.js"
  integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+
n" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
  integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooA
o" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/js/bootstrap.min.js"
  integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB
6" crossorigin="anonymous"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      rules:
`S->BB
B->aB
B->b`,
      inputString: 'aabb',
      isAccepted: null,
      steps: [],
      nonTerminals: null,
      terminals: null,
      actionTable: null,
      gotoTable: null,
      itemSetIds: [],
      errorMsg: '',
    },

    computed: {
      btnState() {
        return this.isAccepted ? 'btn-success' :
          (this.isAccepted === null ? 'btn-primary' : 'btn-danger');
      },
      actionTableWidth() {
        return `col-${12 / (this.terminals.size + 1)}`;
      },
      gotoTableWidth() {
        return `col-${12 / (this.nonTerminals.size + 1)}`;
      },
    },

    methods: {
      start() {
        const rules = this.rules.split('\n').map(rule => rule.replace(/\s+/g, ''));
        const inputString = this.inputString.trim();
        while (rules[rules.length - 1] === '') rules.pop();
        if (!rules.length || !inputString) return;

        const [errorMsg, steps, info] = startLR1(rules, inputString);
        this.steps = steps;

        if (errorMsg === undefined || errorMsg === 'fail') {
```

```
          [nonTerminals, this.terminals, this.actionTable, this.gotoTable, itemSets
Size,
            ...rest] = info;
          for (const nonTerminal of nonTerminals) {
            if (nonTerminal[nonTerminal.length - 1] === "'") {
              nonTerminals.delete(nonTerminal);
              break;
            }
          }
          const itemSetIds = [];
          for (let i = 0; i < itemSetsSize; ++i) {
            itemSetIds.push(String(i));
          }
          this.nonTerminals = nonTerminals;
          this.itemSetIds = itemSetIds;
          if (errorMsg === undefined) this.isAccepted = true;
          else this.isAccepted = false;
        }
        else if (errorMsg === 'format' || errorMsg === 'notLR1') {
          this.isAccepted = null;
          this.nonTerminals = null;
          this.terminals = null;
          this.actionTable = null;
          this.gotoTable = null;
          this.itemSetIds = [];
          this.errorMsg = errorMsg === 'format' ? '输入格式错误' : '非 LR(1)文法'
          $('#exampleModal').modal('show');
        }
      }
    }
  });
</script>

</html>
```

3. Scheme 解释器源码

repl.py

```python
import sys
from buffered_stream import BufferedStream
from scheme_read import scheme_read
from eval_apply import meval
from scheme_env import setup_environment


def driver_loop():
    f = BufferedStream(sys.stdin)
    the_global_environment = setup_environment()
    while True:
        try:
            print('\n]=> ', end='', flush=True)
            result = meval(scheme_read(f), the_global_environment)
            print(';==>', result)
        except Exception as e:
            print("Error:", e)
        except KeyboardInterrupt:
            exit()


if __name__ == '__main__':
    driver_loop()
```

buffered_stream.py

```python
import sys
class BufferedStream:
    def __init__(self, stream=sys.stdin) -> None:
        self._stream = stream
        self._buffered = []

    def getc(self) -> str:
        if self._buffered:
            return self._buffered.pop(0)
        return self._stream.read(1)

    def ungetc(self, c) -> None:
        self._buffered.insert(0, c)

    def peek(self) -> str:
        if self._buffered:
            return self._buffered[0]
        c = self._stream.read(1)
        self._buffered.append(c)
        return c

    def remove_whitespace(self) -> None:
        c = self.getc()
        while c:
            if c == ' ':
                c = self.getc()
            elif c == '\n':
                c = self.getc()
            elif c == ';':
                c = self.getc()
                while c and c != '\n':
                    c = self.getc()
            else:
                self.ungetc(c)
                Break
```

scheme_read.py

```python
from scheme_types import Symbol, Pair, TheEmptyList, the_empty_list
from buffered_stream import BufferedStream


def scheme_read(f: BufferedStream):
    f.remove_whitespace()
    c = f.getc()
    if is_number(c, f.peek()):
        f.ungetc(c)
        return read_number(f)
    if c == '#':
        c = f.getc()
        if c == 't':
            return True
        if c == 'f':
            return False
        if c == '\\':
            return read_character(f)
        raise Exception(f"Boolean value must be #t or #f, not #{c}")
    if is_initial(c):
        f.ungetc(c)
        return read_symbol(f)
    if c == '\"':
        return read_string(f)
    if c == '(':
        return read_pair(f)
    raise Exception("Unknown syntax")


def read_character(f: BufferedStream) -> str:
    c = f.getc()
    next_c = f.peek()
    if c == 's' and next_c == 'p':
        read_expected_string(f, "pace", c)
        return ' '
    if c == 'n' and next_c == 'e':
        read_expected_string(f, "ewline", c)
        return '\n'
    if not (next_c == ' ' or next_c == '\n' or next_c == ';'):
        raise Exception(f"Invalid character {c}{next_c}...")
    return c
```

```python
def read_expected_string(f: BufferedStream,
                         expected_string: str,
                         initial_c: str) -> None:
    """
    Consume expected characters from the input buffer.
    @param initial_c This parameter is only for print error message.
    """
    c = f.getc()
    string_read = [c]
    for i in range(len(expected_string)):
        if c == expected_string[i]:
            c = f.getc()
            string_read.append(c)
        else:
            raise Exception(f"Invalid character {initial_c}{string_read}...")


def read_string(f: BufferedStream) -> str:
    buf = []
    c = f.getc()
    while c != '\"':
        buf.append(c)
        c = f.getc()
    return ''.join(buf)


def read_symbol(f: BufferedStream) -> Symbol:
    buf = []
    c = f.getc()
    while not is_delimiter(c):
        buf.append(c)
        c = f.getc()
    f.ungetc(c)
    return Symbol(''.join(buf))


def read_pair(f: BufferedStream) -> Pair or TheEmptyList:
    f.remove_whitespace()
    c = f.getc()
    if c == ')':
        return the_empty_list
    f.ungetc(c)
    car = scheme_read(f)
    f.remove_whitespace()
    cdr = read_pair(f)
```

```python
    return Pair(car, cdr)


def read_number(f: BufferedStream) -> int or float:
    buf = []
    c = f.getc()
    while not is_delimiter(c):
        buf.append(c)
        c = f.getc()
    f.ungetc(c)
    buf = ''.join(buf)
    if '.' in buf:
        return float(buf)
    elif '/' in buf:
        raise Exception("Rational not implemented")
    else:
        return int(buf)


def is_delimiter(c: str) -> bool:
    return c in (' ', '(', ')', '\"', ';', '\n') or c is None


def is_initial(c: str) -> bool:
    return c.isalpha() or c in '+-*/<>=?!&'


def is_number(c: str, next_c: str) -> bool:
    return c.isdigit() or \
        (c == '.' and next_c.isdigit()) or \
        (c == '-' and (next_c == '.' or next_c.isdigit()))
```

eval_apply.py

```python
'''
exp -> int
     | float
     | str
     | bool
     | the_empty_list
     | Symbol
     | Pair (list of <exp>)
'''

from scheme_types import Symbol, Pair, the_empty_list
from scheme_types import is_null, is_true
from scheme_types import list_to_pair, pair_to_list
from scheme_types import PrimitiveProcedure, CompoundProcedure
from scheme_env import lookup_variable_value
from scheme_env import set_variable_value, define_variable
from scheme_env import extend_environment

from buffered_stream import BufferedStream
from scheme_read import scheme_read


def meval(exp, env):
    # primitives
    if is_self_evaluating(exp):
        return exp
    if is_variable(exp):
        return lookup_variable_value(exp, env)
    # special forms
    if is_quoted(exp):
        return text_of_quotation(exp)
    if is_assignment(exp):
        return eval_assignment(exp, env)
    if is_definition(exp):
        return eval_definition(exp, env)
    if is_lambda(exp):
        return make_procedure(lambda_parameters(exp), lambda_body(exp), env)
    if is_if(exp):
        return eval_if(exp, env)
    if is_begin(exp):
        return [meval(e, env) for e in begin_actions(exp)][-1]
    if is_cond(exp):
```

```python
        return meval(cond_to_if(exp), env)
    if is_let(exp):
        return meval(let_to_combination(exp), env)
    if is_load(exp):
        return eval_load(exp, env)
    # combinations
    if is_application(exp):
        return mapply(meval(operator(exp), env),
                      [meval(e, env) for e in pair_to_list(operands(exp))])
    raise Exception("Unknown expression type")


def mapply(procedure, arguments: list):
    if is_primitive_procedure(procedure):
        return primitive_proc_underlying_proc(procedure)(*arguments)
    elif is_compound_procedure(procedure):
        new_env = extend_environment(
            pair_to_list(procedure_parameters(procedure)),
            arguments,
            procedure_environment(procedure))
        return [meval(e, new_env) for e in procedure_body(procedure)][-1]


def eval_if(exp, env):
    if is_true(meval(if_predicate(exp), env)):
        return meval(if_consequent(exp), env)
    else:
        return meval(if_alternative(exp), env)


def eval_assignment(exp, env):
    set_variable_value(assignment_variable(exp),
                       meval(assignment_value(exp), env),
                       env)
    return 'ok'


def eval_definition(exp, env):
    define_variable(definition_variable(exp),
                    meval(definition_value(exp), env),
                    env)
    return 'ok'


def is_self_evaluating(exp):
    return isinstance(exp, int) or \
```

```python
           isinstance(exp, float) or \
           isinstance(exp, str) or \
           isinstance(exp, bool) or \
           is_null(exp)


def is_variable(exp):
    return isinstance(exp, Symbol)


def is_quoted(exp):
    return exp.car == Symbol('quote')

def text_of_quotation(exp):
    return exp.cdr.car


def is_assignment(exp):
    return exp.car == Symbol('set!')

def assignment_variable(exp):
    return exp.cdr.car

def assignment_value(exp):
    return exp.cdr.cdr.car


'''
(define <var> <val>)
OR
(define (<var> <param_1> ... <param_n>)
  <body>)
->
(define <var>
  (lambda (<param_1> ... <param_n>)
    <body>))
'''

def is_definition(exp):
    return exp.car == Symbol('define')

def definition_variable(exp):
    if isinstance(exp.cdr.car, Symbol):
        return exp.cdr.car
    else:
```

```python
        return exp.cdr.car.car

def definition_value(exp):
    if isinstance(exp.cdr.car, Symbol):
        return exp.cdr.cdr.car
    else:
        return make_lambda(exp.cdr.car.cdr, exp.cdr.cdr)


def is_lambda(exp):
    return exp.car == Symbol('lambda')

def lambda_parameters(exp):
    return exp.cdr.car

def lambda_body(exp):
    return exp.cdr.cdr

def make_lambda(params, body):
    if is_null(body):
        raise Exception("Procedure has no body")
    return Pair(Symbol('lambda'), Pair(params, body))


def is_if(exp):
    return exp.car == Symbol('if')

def if_predicate(exp):
    return exp.cdr.car

def if_consequent(exp):
    return exp.cdr.cdr.car

def if_alternative(exp):
    if not is_null(exp.cdr.cdr.cdr):
        return exp.cdr.cdr.cdr.car
    else:
        return False

def make_if(predicate, consequent, alternative):
    return Pair(Symbol('if'),
                Pair(predicate,
                     Pair(consequent,
                          Pair(alternative, the_empty_list))))
```

```python
def is_begin(exp):
    return exp.car == Symbol('begin')

def begin_actions(exp):
    return exp.cdr

def is_last_exp(seq):
    return is_null(seq.cdr)

def first_exp(seq):
    return seq.car

def rest_exps(seq):
    return seq.cdr

def sequence_to_exp(seq):
    if is_null(seq):
        return seq
    elif is_last_exp(seq):
        return first_exp(seq)
    else:
        return Pair(Symbol('begin'), seq)


def is_cond(exp):
    return exp.car == Symbol('cond')

def cond_to_if(exp):
    return expand_clauses(cond_clauses(exp))

def expand_clauses(clauses):
    if is_null(clauses):
        # No else clause
        return False
    first = clauses.car
    rest = clauses.cdr
    if is_cond_else_clause(first):
        if not is_null(rest):
            raise Exception("Else clause is not the last")
        return sequence_to_exp(cond_actions(first))
    return make_if(cond_predicate(first),
                   sequence_to_exp(cond_actions(first)),
                   expand_clauses(rest))

def cond_clauses(exp):
```

```python
    return exp.cdr

def cond_predicate(clause):
    return clause.car

def cond_actions(clause):
    return clause.cdr

def is_cond_else_clause(clause):
    return cond_predicate(clause) == Symbol('else')


def is_let(exp):
    return exp.car == Symbol('let')

def let_to_combination(exp):
    return Pair(make_lambda(let_vars(exp), let_body(exp)),
                let_vals(exp))

def let_bindings(exp):
    if is_null(exp.cdr.car):
        raise Exception("Let has no bindings")
    return exp.cdr.car

def let_body(exp):
    if is_null(exp.cdr.cdr):
        raise Exception("Let has no body")
    return exp.cdr.cdr

def let_vars(exp):
    return list_to_pair([binding.car for binding in let_bindings(exp)])

def let_vals(exp):
    return list_to_pair([binding.cdr.car for binding in let_bindings(exp)])


def is_application(exp):
    return isinstance(exp, Pair)

def operator(exp):
    return exp.car

def operands(exp):
    return exp.cdr
```

```python
def is_load(exp):
    return exp.car == Symbol('load')

def eval_load(exp, env):
    result = 'ok'
    with open(load_filename(exp), 'r') as file:
        f = BufferedStream(file)
        while f.peek():
            result = meval(scheme_read(f), env)
            f.remove_whitespace()
    return result

def load_filename(exp):
    return exp.cdr.car


def is_primitive_procedure(proc):
    return isinstance(proc, PrimitiveProcedure)

def primitive_proc_underlying_proc(proc: PrimitiveProcedure):
    return proc.underlying_primitive_proc


def make_procedure(params, body, env):
    return CompoundProcedure(params, body, env)

def is_compound_procedure(proc):
    return isinstance(proc, CompoundProcedure)

def procedure_parameters(proc: CompoundProcedure):
    return proc.parameters

def procedure_body(proc: CompoundProcedure):
    if is_null(proc.body):
        raise Exception("Procedure has no body")
    return proc.body

def procedure_environment(proc: CompoundProcedure):
    return proc.environment
```

scheme_types.py

```python
class Symbol:
    def __init__(self, name: str) -> None:
        self.name = name

    def __repr__(self) -> str:
        return self.name

    def __eq__(self, o: object) -> bool:
        return isinstance(o, Symbol) and self.name == o.name

    def __hash__(self) -> int:
        return self.name.__hash__()


class TheEmptyList:
    def __repr__(self) -> str:
        return '()'


the_empty_list = TheEmptyList()


def is_null(exp) -> bool:
    return isinstance(exp, TheEmptyList)


class Pair:
    def __init__(self, car, cdr) -> None:
        self.car = car
        self.cdr = cdr

    def __iter__(self):
        pair = self
        while not isinstance(pair, TheEmptyList):
            yield pair.car
            pair = pair.cdr

    def __repr__(self) -> str:
        return '(' + ' '.join([str(e) for e in self]) + ')'


class PrimitiveProcedure():
    def __init__(self, fn) -> None:
```

```python
        self.underlying_primitive_proc = fn

    def __repr__(self) -> str:
        return "Primitive procedure"


class CompoundProcedure():
    def __init__(self, params, body, env) -> None:
        self.parameters = params
        self.body = body
        self.environment = env

    def __repr__(self) -> str:
        return "Compound procedure"


"""
Only False is false
"""
def is_true(val) -> bool:
    if isinstance(val, bool):
        return val
    if val == 0:
        return True
    if val == "":
        return True
    if is_null(val):
        return True
    return bool(val)


def pair_to_list(p: Pair) -> list:
    if is_null(p):
        return []
    return list(p)


def list_to_pair(lst: list) -> Pair:
    if not len(lst):
        return the_empty_list
    return Pair(lst[0], list_to_pair(lst[1:]))
```

scheme_env.py

```python
import math
from scheme_types import Symbol, Pair, is_null, the_empty_list
from scheme_types import PrimitiveProcedure
from primitive_procedures import scheme_not, scheme_and, scheme_or


class Environment:
    def __init__(self, bindings: dict, base) -> None:
        self.bindings = bindings
        self.base = base


the_empty_environment = None


def extend_environment(params: list, args: list, base: Environment):
    """
    @param params: list of Symbols
    """
    if len(params) < len(args):
        raise Exception("Too many arguments")
    elif len(params) > len(args):
        raise Exception("Too few arguments")
    return Environment(dict(zip(params, args)), base)


def lookup_variable_value(var: Symbol, env: Environment):
    frame = env
    while frame != the_empty_environment:
        if var in frame.bindings:
            return frame.bindings[var]
        frame = frame.base
    raise Exception(f"Unbound variable: {var}")


def set_variable_value(var: Symbol, val, env: Environment):
    frame = env
    while frame != the_empty_environment:
        if var in frame.bindings:
            frame.bindings[var] = val
            return 'ok'
        frame = env.base
    raise Exception(f"Unbound variable: {var}")
```

```python
def define_variable(var: Symbol, val, env: Environment):
    env.bindings[var] = val
    return 'ok'


def setup_environment():
    global_environment = extend_environment([], [], the_empty_environment)
    global_bindings = {
        Symbol('true'): True,
        Symbol('false'): False,
        Symbol('nil'): the_empty_list,
        Symbol('null?'): PrimitiveProcedure(is_null),
        Symbol('car'): PrimitiveProcedure(lambda pair: pair.car),
        Symbol('cdr'): PrimitiveProcedure(lambda pair: pair.cdr),
        Symbol('cons'): PrimitiveProcedure(lambda a, b: Pair(a, b)),
        Symbol('+'): PrimitiveProcedure(lambda *ops: sum(ops)),
        Symbol('-'): PrimitiveProcedure(lambda a, b: a - b),
        Symbol('*'): PrimitiveProcedure(lambda *ops: math.prod(ops)),
        Symbol('/'): PrimitiveProcedure(lambda a, b: a / b),
        Symbol('<'): PrimitiveProcedure(lambda a, b: a < b),
        Symbol('>'): PrimitiveProcedure(lambda a, b: a > b),
        Symbol('<='): PrimitiveProcedure(lambda a, b: a <= b),
        Symbol('>='): PrimitiveProcedure(lambda a, b: a >= b),
        Symbol('='): PrimitiveProcedure(lambda a, b: a == b),
        Symbol('not'): PrimitiveProcedure(scheme_not),
        Symbol('and'): PrimitiveProcedure(scheme_and),
        Symbol('or'): PrimitiveProcedure(scheme_or),
        Symbol('exit'): PrimitiveProcedure(lambda: exit())
    }
    global_environment.bindings = global_bindings
    return global_environment
```