Scheme Interpreter(PPCA 2014)

张曦虎

2014.8.11

目录：

1. 简介：

2. 当前 API 可支持的特性(v.1.0.0)：

a) Syntax

b) Data type

c) Procedure

3. 对于 API 改进的想法(v.2.0.0)：

4. 感受

5. 致谢

1.简介：

本工程是按照 Scheme 的 R5RS 标准（部分功能与 R5RS 标准有所差别），使用 C++语言实现的一个 Scheme 解释器。目前只添加了部分功能。利用当前的 API 接口完善解释器后可以实现大部分 Scheme 的功能。工程已在 Visual Studio 2010 C++环境下成功编译。由于使用了 C++11 的部分特性，需要使用支持 C++11 的编译器进行编译生成。

编译环境：

请保证至少 128MB 的堆空间和栈空间。

在 Visual Studio 中设置堆保留空间和栈保留空间为 33554432

或加入： /HEAP:"33554432" /STACK:"33554432"

在 VS2010 下请使用 Release 模式，如使用其它方式编译，请尽可能的打开优化。

代码内部调试部分：

Debugger\debugger.h： 启用#define __DEBUG_MODE_ON_

可以在大部分函数中启用类型或边界检查。

Parser\parser.cpp： 启用#define __PARSER_DEBUGGER_

可在屏幕上输出解释器的执行步骤。

运行：

请确保 library.scm 与 Scheme_Interpreter 在同一个目录下。

文件的输出存放在 SchemeInterpreter.log 内部。

Usage: Scheme_Interpreter <filename>.

2.当前 API 已支持（可支持）特性：

注：support 指程序中已可以支持，

can be supported 表示该特性在当前 API 环境下可以作为补丁加入程序。

Unsupported 表示该特性需要改变底层 API 才能得到支持。

与 R5RS 有差别的功能已用 violation 标出。

a）R5RS Syntax

- R5RS Section 4.1.1 Variable references:
  - <variable>   <syntax support>

- R5RS Section 4.1.2 Literal expressions:
  - (quote <datum>)   <syntax support>
  - '<datum>   <syntax support>
  - <constant>   <syntax support>

- R5RS Section 4.1.3 Procedure calls:
  - (<operator> <operands>…)   <syntax support>

    R5RS claims that the evaluation of operator and operands are in an unspecified consistent order. In this implementation, the evaluation order is always from left to right.(due to the side effect.)

- R5RS Section 4.1.4 Procedures
  - (lambda <formals> <body>)   <syntax support>

- R5RS Section 4.1.5 If-predicates
  - (if <test> <consequent> <alternate>)   <syntax support>
  - (if <test><consequent>)   <syntax support>

- R5RS Section 4.1.6 Assignment
  - (set! <variable> <expression>)

    R5RS claims that the result of the set! expression is unspecified. In this implementation, the return value is an object of void type.

    Buildin-procedures can be assigned, but buildin-syntax cannot be overlapped.

- R5RS Section 4.2.1 Conditionals
  - (cond <clause>…)   <syntax support>

    R5RS requires "cond" to have at least one clause. In this implementation, no clause “cond” is allowed and it will return an object of void type. If no clause’s test evaluate to true value, “cond” will return an object of void type.

R5RS claims that “cond” is a library syntax. In this implementation, “cond” is a syntax due to “define-syntax” is not implemented.

- (case <clause>…) <unsupported>
- (and <test>…) <procedure **violation** can be fixed>

  In R5RS, “and” is a library syntax, but in this implementation, “and” is a primitive procedure and it will evaluate all the test without interruption.(Due to “lazy evaluation” is not implemented, as well as “promise”)
- (or <test>…) <procedure **violation** can be fixed>

  In R5RS, “or” is a library syntax, but in this implementation, “or” is a primitive procedure and it will evaluate all the test without interruption.(Due to “lazy evaluation” is not implemented, as well as “promise”)

• R5RS Section 4.2.2 Binding Constructs

- (let <bindings> <body>) <syntax support>

  In R5RS, “let” is a library syntax, but in this implementation, “let” is a syntax due to “define-syntax” is unsupported.

  R5RS claims that the evaluations of <init>s is in unspecified order. In this implementation, the order is always from left to right.
- (let* <bindings> <body>) <syntax support>

  In R5RS, “let*” is a library syntax, but in this implementation, “let*” is a syntax due to “define-syntax” is unsupported.
- (letrec <bindings> <body>) <syntax **violation**>

  In R5RS, “letrec” is a library syntax, but in this implementation, “let*” is a syntax due to “define-syntax” is unsupported.

  R5RS claims that each binding of a <variable> has the entire “letrec” expression as its region. In this implementation, “letrec” acts like “let” but it will change the lambda expression’s scope to the “letrec” region.(without any checking)

• R5RS Section 4.2.3 Sequencing

- (begin <expression>…) <syntax support>

  R5RS requires “begin” followed by at least one expression. In this implementation, no expression following “begin” is allowed and it will return an object of void type instead.

• R5RS Section 4.2.4 Iteration

- (do ((<variable> <init> <step>)…)

  (<test> <expression> …) <command> …)

  <library syntax unsupported>

- R5RS Section 4.2.5 Delayed evaluation
  - (delay <expression> …)   <library syntax   unsupported>

- R5RS Section 4.2.6 Quasiquotation
  - (quasiquote <qq template>)   <syntax   can be supported>
  - `<qq template>   <syntax   can be supported>

- R5RS Section 4.3 Macros

  Unsupported.

- R5RS Section 5.2 Definition
  - (define <variable> <expression>)   <syntax   support>
  - (define (<variable> <formals>) <body>)   <syntax   support>
  - (define(<variable> . <formals>) <body>)   <syntax   support>

    R5RS requires definitions to appear at the top level or at the beginning of a <body>. This restriction is removed in this implementation. R5RS did not specify the return value, which is an object of void in this implementation.

b）Data Types

- Number
  - Complex   <partially support>

    Some inexact complex number cannot be recognized. Such as ".23+.23i",due to some inexact real number cannot be recognized.
  - Real   <partially support>

    Some inexact real number cannot be recognized, such as ".23".
  - Rational   <support>
  - Integer   <support>
- Booleans   <support>
- Pairs and lists   <support>
- Symbols   <support>
- Characters   <support>
- Strings   <support>
- Promises   <unsupported>
- Procedures   <support>
- Void(Unspecified Value)   <support>
- Environment   <partially support>

c）Procedures

- R5RS Section 6.1 Equivalence predicates
  - (eqv? obj_1 obj_2) <procedure support>
  - (eq? obj_1 obj_2) <procedure support>

    In this implementation, “eq?” function is the same as “eqv?”
  - (equal? obj_1 obj_2) <library procedure support>
- R5RS Section 6.2.5 Numerical operation
  - (number? obj) <procedure support>
  - (complex? obj) <procedure support>

    In this implementation, “complex?” is the same as “number?”.
  - (real? obj) <procedure support>
  - (integer? obj) <procedure support>
  - (exact? obj) <procedure support>
  - (inexact? obj) <procedure support>
  - (= z_1 z_2 z_3…) <procedure support>
  - (< x_1 x_2 x_3…) <procedure support>
  - (> x_1 x_2 x_3…) <procedure support>
  - (<= x_1 x_2 x_3…) <procedure support>
  - (>= x_1 x_2 x_3…) <procedure support>
  - (zero? x) <library procedure support>
  - (positive? x) <library procedure support>
  - (negative? x) <library procedure support>
  - (odd? x) <library procedure support>
  - (even? x) <library procedure support>
  - (max x_1 x_2…) <library procedure can be supported>
  - (min x_1 x_2…) <library procedure can be supported>
  - (+ z_1…) <procedure support>
  - (* z_1…) <procedure support>
  - (- z_1…) <procedure support>
  - (/ z_1…) <procedure support>
  - (abs x) <library procedure support>
  - (quotient n_1 n_2) <procedure support>
  - (remainder n_1 n_2) <procedure support>
  - (modulo n_1 n_2) <procedure support>
  - (gcd n_1…) <library procedure can be supported>
  - (lcm n_1…) <library procedure can be supported>
  - (numerator q) <procedure can be supported>
  - (denominator q) <procedure can be supported>

-	(floor x)	<procedure	can be supported>
-	(ceiling x)	<procedure	can be supported>
-	(truncate x)	<procedure	can be supported>
-	(round x)	<procedure	can be supported>
-	(exp z)	<procedure	can be supported>
-	(log z)	<procedure	can be supported>
-	(sin z)	<procedure	can be supported>
-	(cos z)	<procedure	can be supported>
-	(tan z)	<procedure	can be supported>
-	(asin z)	<procedure	can be supported>
-	(acos z)	<procedure	can be supported>
-	(atan z)	<procedure	can be supported>
-	(atan y x)	<procedure	can be supported>
-	(sqrt z)	<procedure	support>
-	(expt z_1 z_2)	<procedure	can be supported>
-	(make-rectangular x_1 x_2)	<procedure	can be supported>
-	(make-polar x_3 x_4)	<procedure	can be supported>
-	(real-part z)	<procedure	can be supported>
-	(imag-part z)	<procedure	can be supported>
-	(magnitude z)	<procedure	can be supported>
-	(angle z)	<procedure	can be supported>
-	(exact->inexact z)	<procedure	can be supported>
-	(inexact->exact)	<procedure	can be supported>

• R5RS Section 6.2.6 Numercial input and output

-	(number->string z)	<procedure	can be supported>
-	(number->string z radix)	<procedure	can be supported>
-	(string->number string)	<procedure	can be supported>
-	(string->number string radix)	<procedure	can be supported>

• R5RS Section 6.3.1 Booleans

-	(not obj)	<library procedure	support>
-	(boolean? obj)	<library procedure	support>

• R5RS Section 6.3.2 Pairs and lists

-	(pair? obj)	<procedure	support>
-	(cons obj_1 obj_2)	<procedure	support>
-	(car pair)	<procedure	support>
-	(cdr pair)	<procedure	support>

- (set-car! pair)	<procedure support>
- (set-cdr! pair)	<procedure support>
- (caar pair)	<library procedure support>
- (cadr pair)	<library procedure support>
...	
- (cdddar pair)	<library procedure support>
- (cddddr pair)	<library procedure support>
- (null? obj)	<library procedure support>
- (list? obj)	<library procedure support>
- (list obj…)	<library procedure support>
- (length list)	<library procedure support>
- (append list…)	<library procedure support>
- (reverse list)	<library procedure support>
- (list-tail list k)	<library procedure support>
- (list-ref list k)	<library procedure support>
- (memq obj list)	<library procedure support>
- (memv obj list)	<library procedure support>
- (member obj list)	<library procedure support>
- (assq obj alist)	<library procedure support>
- (assv obj alist)	<library procedure support>
- (assoc obj alist)	<library procedure support>

• R5RS Section 6.3.3 Symbols

- (symbol? obj)	<procedure support>
- (symbol->string symbol)	<procedure can be supported>
- (string->symbol string)	<procedure can be supported>

• R5RS Section 6.3.4 Characters

- (char? obj)	<procedure can be supported>
- (char=? char$_1$ char$_2$)	<procedure can be supported>
- (char<? char$_1$ char$_2$)	<procedure can be supported>
- (char>? char$_1$ char$_2$)	<procedure can be supported>
- (char<=? char$_1$ char$_2$)	<procedure can be supported>
- (char>=? char$_1$ char$_2$)	<procedure can be supported>
- (char-ci=? char$_1$ char$_2$)	<library procedure can be supported>
- (char-ci<? char$_1$ char$_2$)	<library procedure can be supported>
- (char-ci>? char$_1$ char$_2$)	<library procedure can be supported>
- (char-ci<=? char$_1$ char$_2$)	<library procedure can be supported>
- (char-ci>=? char$_1$ char$_2$)	<library procedure can be supported>

- (char-alphabetic? char)	<library procedure	can be supported>
- (char-numeric? char)	<library procedure	can be supported>
- (char-whitespace? char)	<library procedure	can be supported>
- (char-upper-case? letter)	<library procedure	can be supported>
- (char-lower-case? letter)	<library procedure	can be supported>
- (char->integer char)	<procedure	can be supported>
- (integer->char n)	<procedure	can be supported>
- (char-upcase char)	<library procedure	can be supported>
- (char-downcase char)	<library procedure	can be supported>

• R5RS Section 6.3.5 Strings

- (string? obj)	<procedure	can be supported>
- (make-string k)	<procedure	can be supported>
- (make-string k char)	<procedure	can be supported>
- (string char…)	<library procedure	can be supported>
- (string-length string)	<procedure	can be supported>
- (string-ref string k)	<procedure	can be supported>
- (string-set! string k char)	<procedure	can be supported>
- (string=? string$_1$ string$_2$)	<library procedure	can be supported>
- (string-ci=? string$_1$ string$_2$)	<library procedure	can be supported>
- (string<? string$_1$ string$_2$)	<library procedure	can be supported>
- (string>? string$_1$ string$_2$)	<library procedure	can be supported>
- (string<=? string$_1$ string$_2$)	<library procedure	can be supported>
- (string>=? string$_1$ string$_2$)	<library procedure	can be supported>
- (string-ci<? string$_1$ string$_2$)	<library procedure	can be supported>
- (string-ci>? string$_1$ string$_2$)	<library procedure	can be supported>
- (string-ci<=? string$_1$ string$_2$)	<library procedure	can be supported>
- (string-ci>=? string$_1$ string$_2$)	<library procedure	can be supported>
- (substring string start end)	<library procedure	can be supported>
- (string-append string…)	<library procedure	can be supported>
- (string->list string)	<library procedure	can be supported>
- (list->string list)	<library procedure	can be supported>
- (string-copy string)	<library procedure	can be supported>
- (string-fill! string char)	<library procedure	can be supported>

• R5RS Section 6.3.6 Vectors	<unsupported>

• R5RS Section 6.4 Control features

- (procedure? obj)	<procedure	can be supported>

- (apply proc arg$_1$ …)	<procedure	support>
- (map proc list$_1$…)	<library procedure	support>
- (for-each proc list$_1$…)	<library procedure	can be supported>
- (force promise)	<library procedure	unsupported>
- (call-with-current-continuation proc)	<procedure	unsupported>
- (values obj…)	<procedure	unsupported>
- (call-with-values producer consumer)	<procedure	unsupported>
- (dynamic-wind before thunk after)	<procedure	unsupported>
• R5RS Section 6.5 Eval	<can be supported>	
• R5RS Section 6.6.1 Ports	<unsupported>	
• R5RS Section 6.6.2 Input	<unsupported>	
• R5RS Section 6.6.3 Output		
- (display obj)	<library procedure	support>
- (newline)	<library procedure	support>
- (write obj)	<library procedure	unsupported>
- (write obj port)	<library procedure	unsupported>
- (display obj port)	<library procedure	unsupported>
- (newline port)	<library procedure	unsupported>
- (write-char char)	<library procedure	unsupported>
- (write-char char port)	<library procedure	unsupported>
- (load filename)	<optional procedure	unsupported>
- (transcript-on filename)	<optional procedure	unsupported>
- (transcript-off)	<optional procedure	unsupported>

3.对于 API 改进的想法(v.2.0.0):

关于 tail recursion:

由于底层 API 的 Eval 实现方式为递归调用函数。一直在申请栈空间，所以没有办法保证能够很好地控制递归。如果要实现 tail recursion,需要将 eval 中的调用部分改写为手动栈实现，便于调试和实现新的功能。

关于底层实现改进:

再次阅读了下 R5RS 和 MIT-Scheme Reference 后，对于 syntax rule 有一些粗略的想法。

考虑底层实现的 Eval 和 Apply:

```
shared_ptr<Object> Eval(shared_ptr<Object> expr, shared_ptr<Environment> envi)
{
    #ifdef __PARSER_DEBUGGER_
    std::cout << "Eval processing...    :";
    buildin::_Display(expr);
    std::cout << std::endl;
    #endif

    if (_SelfEvaluating(expr)) return expr;
    if (_IsVariable(expr)) return LookUpVariableValue(static_pointer_cast<Symbol>(expr), envi);
    if (_IsQuoted(expr)) return TextOfQuotation(expr);
    if (_IsAssignment(expr)) return EvalAssignment(expr, envi);
    if (_IsDefinition(expr)) return EvalDefinition(expr, envi);
    if (_IsIfPredicate(expr)) return EvalIf(expr, envi);
    if (_IsLambda(expr)) return MakeProcedure(LambdaParameters(expr), LambdaBody(expr), envi);
    if (_IsBegin(expr)) return EvalSequence(BeginActions(expr), envi);
    if (_IsLet(expr)) return EvalLet(LetParameters(expr), LetBody(expr), envi);
    if (_IsLetStar(expr)) return EvalLetStar(LetParameters(expr), LetBody(expr), envi);
    if (_IsLetRec(expr)) return EvalLetRec(LetParameters(expr), LetBody(expr), envi);
    if (_IsCondition(expr)) return EvalCond(CondSequence(expr), envi);
    if (_IsApplication(expr))
    {
        //buildin::_Display(ListOfValue(Operands(expr), envi));
        return Apply(Eval(Operator(expr), envi),
                                    ListOfValue(Operands(expr), envi));
    }

    throw Debugger::DebugMessage("Error: Unknown expression type -- EVAL\n");
}
```

```
shared_ptr<Object> Apply(shared_ptr<Object> procedure, shared_ptr<Object> arguments)
{
    #ifdef __PARSER_DEBUGGER_
    std::cout << "Applying Procedure && arguments...  ";
    buildin::_Display(arguments);
    std::cout << std::endl;
    #endif

    if (procedure -> getType() == PRIMITIVE_PROCEDURE)
    {
        return ApplyPrimitiveProcedure(procedure, arguments);
    }
    else
    if (procedure -> getType() == COMPOUND_PROCEDURE)
    {
        /*return EvalSequence(ProcedureBody(procedure)
                            ,static_pointer_cast<Environment>
                            (static_pointer_cast<Procedure>(procedure)
                            -> ExtendEnvironment(arguments)));*/

        shared_ptr<Environment> procedureEnv;
        procedureEnv = static_pointer_cast<Procedure>(procedure) -> ExtendEnvironment(arguments);

        shared_ptr<Object> exv;
        exv = EvalSequence(ProcedureBody(procedure)
                            ,procedureEnv);

        if (!ContainProcedure(exv)) procedureEnv -> EmptyEnv();

        return exv;
    }

    throw Debugger::DebugMessage("Unknown procedure type -- APPLY\n");
}
```

这是以《计算机程序的构造和解释》上的模版为基础写出来的 Eval 基础程序，但是我认为可以继续简化。

需要保留的是 SelfEvaluating 和 Variable。SelfEvaluating 的结果是自身，所以不需要进一步 Eval，存储于变量中的结果是已经经过 Eval 处理的结果，所以不需要 Eval。

而从“_IsQuoted”至“_IsCondition”之间的所有语句整合到 Apply 当中。然后作特殊的 SyntaxApply 即可。

Apply 则改写为：如果查到的所谓的“operator”为 syntax，则执行对应的 syntax-rule（保留所有 operands 引用，“引用”也意味着这个值可以被修改。对于该引用暂不进行 eval，当需要 Eval 的时候再进行 Eval，当需要作赋值的时候再作赋值），如果 operator 为 procedure 则对其 operands 执行 eval 后再作传参（应用序）。最后将 syntax 归入当前 environment。

高度整合后的 Scheme 期望下只有三个部分：List、Syntax、Procedure。

4. 感受：

写 Scheme 解释器之前感觉解释器这个工程难度非常高。

而且最开始对 Scheme 的语法理解也仅仅是达到了语法树的部分。

对于 Scheme 的 car 和 cdr 所产生的 list 一开始也不是很了解，而在编写过程中才发现左儿子右兄弟这个性质就隐含在 car 和 cdr 中。Environment 这个看上去有点复杂的东西本质上居然也是 list。（理论上可以用 list 实现，但速度太慢了。）

将函数看成对象，任何复合函数都可以化为一元函数的形式。造就了使用 list 结构就可以进行 eval 的杰作。

程序即数据，数据即程序。完成这个解释器的编写也算是加深了对冯诺依曼体系的理解吧。（程序和数据以二进制代码形式不加区别地存放在存储器中，存放位置由地址决定。）

（接下来是瞎扯，有些东西都是凭自己的理解写的，不一定对，还望指出。不胜感激。）

首先要说的是 lambda 这个东西：

有个东西叫 currying，中文译名“柯里化”而不是“咖喱化”。有个东西叫泛函分析，函数是对象，函数映射到函数。（或者干脆说有个东西叫公理化集合论得了）

当函数成了对象，当函数可以只由带一个参数的函数复合而成，什么都解决了。

一个猜想（毕竟没仔细读 R5RS，所以只是自己的理解，不是 R5RS 所作要求）：

函数表达式的标志是 **lambda**，而在它手下定义的东西是基于应用序的 **procedure**.

符号保留字的标志是 **syntax-rule**，定义的东西是基于正则序（类似正则序，更准确的说是带入一种引用）的 **syntax**.

突然觉得这两个保留字都可以用一套体系来执行了。

甚至我们可以支持一种新的通用 definition，它可以指定 operands 中哪些部分作严格求值，哪些部分作非严格求值（带引用）。从而囊括 procedure 和 syntax 两个部分的方法。在此基础上可以实现 Macro。

5．致谢：

感谢赵卓越学长在Scheme的特性和解释器的实现以及与解释器相关的各个知识上的耐心指导。

感谢周耀达同学在 Scheme 与 C++11 特性的讨论中给予的点拨。

参考文献：

[1] R. Kelsey, W. Clinger, J. Rees (eds.), Revised5 Report on the Algorithmic Language Scheme, Higher-Order and Symbolic Computation, Vol. 11, No. 1, August, 1998.

[2]Harold Abelson, Gerald Jay Sussman, Julie Sussman, Structure and Interpretation of Computer Programs, Second Edition.

[3]MIT Scheme Reference: http://www.cse.iitb.ac.in/~as/mit-scheme/scheme.html