

# Proj2 Lab 7 Experiment

## 预备阅读：

---

### 1. 初始化环境的方式(env)：

- 环境变量主要保存于 `initial-objects` 中，可以通过 `map` 与 `car` `cdr` 进行访问；
- 在 `initial-objects` 的基础之上，通过 `extend-environment` 在空表上建立一个新的环境：`initial-env`
- 该环境此时只包含一些简单的常量( `true` `false` `nil` 等等)，需要在其上进行拓展：

1. 定义 `<object>` 类：在 *TOOL* 中，`<object>` 类是没有 `superclass` 和 `slot` 的原始类( `*primitive-class*` )，同时 `<object>` 被定义在 `initial-env` 这一最早的外围环境上。

```
(define-variable! '<object>' *primitive-class* initial-env)
```

2. 添加 `scheme-object-classes`： *TOOL* 中把 `scheme` 的原始类定义为 `<object>` 的子类，通过 `tool-eval` 解析类定义进行拓展，同样 `scheme` 类也定义在 `initial-env` 上
3. 添加 `initial-procedures`： *TOOL* 把 `scheme` 的一些基本操作转换为 `generic-function` 的形式，实现方法与上述类似。主要问题在于对 `gf` 添加 `method` 的过程：（在 `proj2.scm` 中有测试过程）

1. 添加 `method` 时首先确定 `generic-function`，这里采取的策略是，由于之前已经将 `gf` 加入到 `initial-env` 中，因此这里可以直接用 `tool-eval` 查找符号表以返回对应的 `gf`。

```
gf的结构 (以+为例) :  
(generic-function +  
  (((class <number> ((class <object> () ())) ())) ()))  
  
(class <number> ((class <object> () ())) ()))  
  . #<procedure:++>))
```

2. 之后从 `initial-procedure` 中提取参数列表：

1. `(cadr entry)` 返回的是类名形成的 `list` ,  
如 `(<number> <number>)`
2. 以类名为参数查找符号表中对应的类信息：

```
class的结构  
(class <number>  
  ((class <object> () ())) ;superclass  
  ()) ; slots
```

3. 通过 `install-method-in-generic-function` 将method  
插入到gf中

## 2. TOOL 解析 `method-definition` 的策略：

1. 首先在env ( 根据 `driver-loop` 的代码, 这个 `env` 应该  
为 `the-global-environment` 即全局的环境变量 ) 中寻找这个  
definition中使用到的generic-function

1. 假如没找到则报错 ( 这是这个lab需要修改的地方 )
2. 假如找到了则进入下一步

2. 然后调用 `install-method-in-generic-function` :

1. gf已经找到, 可以直接使用
2. 找到 `paramlist` 中定义每个类名对应的类 ( 依然是在env  
中 )
3. 通过lambda表达式定义过程

## 实验方案：

---

1. 在TOOL 解析 `method-definition` 时，会首先调用 `tool-eval` 去解析参数表达式中的 `generic-function` 的名称。如果该 `generic-function` 尚未定义，则会在 `lookup-variable-value` 时报错（即符号表中不存在对应的名称）
2. `lookup-variable-value` 在判断对应变量名称是否在符号表中时，依赖的是对应的 `binding` 是否存在。可根据此性质建立谓词：

```
;;1. a predicate to make sure if the variable is in the env
(define (is-variable-in-env? var env)
  (let ((b (binding-in-env var env)))
    (found-binding? b)))
```

3. 之后利用该谓词先进行判断，是否存在需要的gf，如果不存在，则利用 `tool-eval` 建立新的gf：

```
(if (is-variable-in-env? gf-name env)
    'found-generic-function
    (begin
      (tool-eval '(define-generic-function ,gf-name) env)
      (display 'new-generic-function-built)))
```

4. 环境的选择：
  - 在本次实现中使用的是全局变量，即 `driver-loop` 传递的 `the-global-environment`
  - 优点：之后method添加时使用的都是全局环境，每次寻找对应的gf时也是首先搜索全局环境，这样可以最快的定位。
  - 缺点：暂无，因为在TOOL的实现中，并未出现多环境的情况，即，所有求值和应用过程依赖的只有一个全局环境。