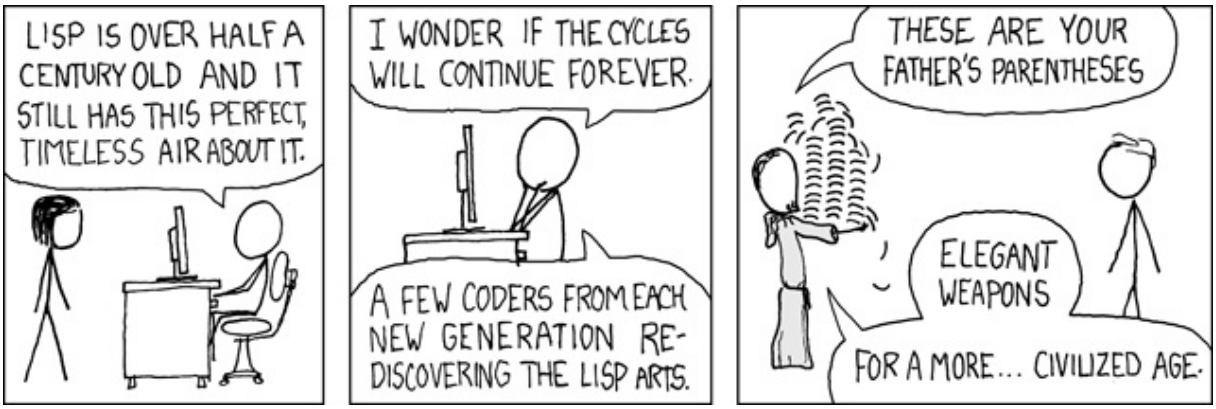
谈语法



使用和研究过这么多程序语言之后，我觉得几乎不包含多余功能的语言，只有一个：Scheme。所以我觉得它是学习程序设计最好的入手点和进阶工具。当然 Scheme 也有少数的问题，而且缺少一些我想要的功能，但这些都瑕不掩瑜。在用了很多其它的语言之后，我觉得 Scheme 真的是非常优美的语言。

要想指出 Scheme 所有的优点，并且跟其它语言比较，恐怕要写一本书才讲的清楚。所以在这篇文章里，我只提其中一个最简单，却又几乎被所有人忽视的方面：语法。

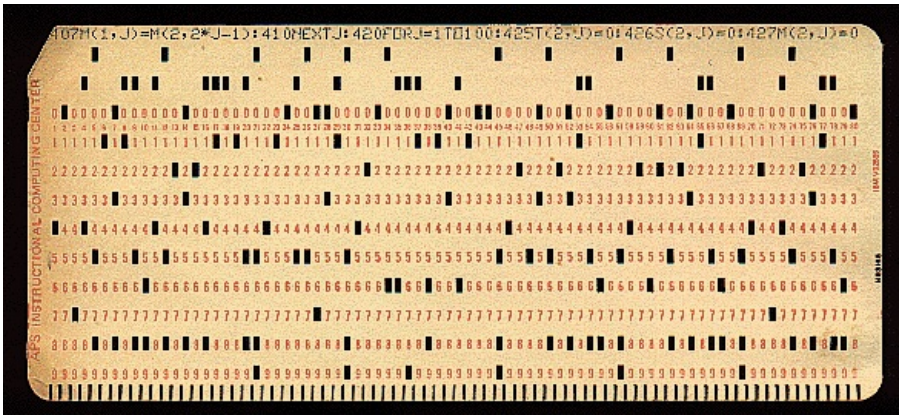
其它的 Lisp “方言”也有跟 Scheme 类似的语法（都是基于“S表达式”），所以在这篇（仅限这篇）文章里我所指出的“Scheme 的优点”，其实也可以作用于其它的 Lisp 方言。从现在开始，“Scheme”和“Lisp”这两个词基本上含义相同。

我觉得 Scheme （Lisp） 的基于“S表达式”（S-expression）的语法，是世界上最完美的设计。其实我希望它能更简单一点，但是在现存的语言中，我没有找到第二种能与它比美。也许在读过这篇文章之后，你会发现这种语法设计的合理性，已经接近理论允许的最大值。

为什么我喜欢这样一个“全是括号，前缀表达式”的语言呢？这是出于对语言结构本质的考虑。其实，我觉得语法是完全不应该存在的东西。即使存在，也应该非常的简单。因为语法其实只是对语言的本质结构，“抽象语法树”（abstract syntax tree，AST），的一种编码。一个良好的编码，应该极度简单，不引起歧义，而且应该容易解码。在程序语言里，这个“解码”的过程叫做“语法分析”（parse）。

为什么我们却又需要语法呢？因为受到现有工具（操作系统，文本编辑器）的限制，到目前为止，几乎所有语言的程序都是用字符串的形式存放在文件里的。为了让字符串能够表示“树”这种结构，人们才给程序语言设计了“语法”这种东西。但是人们喜欢耍小聪明，在有了基本的语法之后，他们开始在这上面大做文章，使得简单的问题变得复杂。

Lisp （Scheme 的前身）是世界上第二老的程序语言。最老的是 Fortran。Fortran 的程序，最早的时候都是用打孔机打在卡片上的，所以它其实是几乎没有语法可言的。



显然，这样写程序很痛苦。但是它却比现代的很多语言有一个优点：它没有歧义，没有复杂的 parse 过程。

在 Lisp 诞生的时候，它的设计者们一下子没能想出一种好的语法，所以他们决定干脆先用括号把这语法树的结构全都括起来，一个不漏。等想到更好的语法再换。

自己想一下，如果要表达一颗“树”，最简单的编码方式是什么？就是用括号把每个节点的“数据”和“子节点”都括起来放在一起。Lisp 的设计者们就是这样想的。他们把这种完全用括号括起来的表达式，叫做“S表达式”（S 代表

“symbolic”）。这貌似很“粗糙”的设计，甚至根本谈不上“设计”。奇怪的是，在用过一段时间之后，他们发现自己已经爱上了这个东西，再也不想设计更加复杂的语法。于是S表达式就沿用至今。

在使用过 Scheme，Haskell，ML，和常见的 Java，C，C++，Python，Perl，…… 之后，我也惊讶的发现，Scheme 的语法，不但是最简单，而且是最好看的一个。这不是我情人眼里出西施，而是有一定理论依据的。

首先，把所有的结构都用括号括起来，轻松地避免了别的语言里面可能发生的“歧义”。程序员不再需要记忆任何“运算符优先级”。

其次，把“操作符”全都放在表达式的最前面，使得基本算术操作和函数调用，在语法上发生完美的统一，而且使得程序员可以使用几乎任何符号作为函数名。

在其他的语言里，函数调用看起来像这个样子：`f(1)`，而算术操作看起来是这样：`1+2`。在 Lisp 里面，函数调用看起来是这样`(f 1)`，而算术操作看起来也是这样`(+ 1 2)`。你发现有什么共同点吗？那就是 `f` 和 `+` 在位置上的对应。实际上，加法在本质也是一个函数。这样做的好处，不但是突出了加法的这一本质，而且它让人可以用跟定义函数一模一样的方式，来定义“运算符”！这比起 C++ 的“运算符重载”强大很多，却又极其简单。

关于“前缀表达式”与“中缀表达式”，我有一个很独到的见解：我觉得“中缀表达式”其实是一种过时的，来源于传统数学的历史遗留产物。几百年以来，人们都在用 `x+y` 这样的符号来表示加法。之所以这样写，而不是 `(+ x y)`，是因为在没有计算机以前，数学公式都得写在纸上，写 `x+y` 显然比 `(+ x y)` 方便简洁。但是，中缀表达式却是容易出现歧义的。如果你有多个操作符，比如 `1+2*3`。那么它表示的是 `(+ 1 (* 2 3))` 呢，还是 `(* (+ 1 2) 3)`？所以才出现了“运算符优先级”这种东西。看见没有，S表达式已经在这里显示出它没有歧义的优点。你不需要知道 `+` 和 `*` 的优先级，就能明白 `(+ 1 (* 2 3))` 和 `(* (+ 1 2) 3)` 的区别。第一个先乘后加，而第二个先加后乘。

对于四则运算，这些优先级还算简单。可是一旦有了更多的操作，就容易出现混淆。这就是为什么数学（以及逻辑学）的书籍难以看懂。 实际上，那些看似复杂的公式，符号，不过是在表示一些程序里的“数据结构”，“对象”以及“函数”。大部分读数学书的时间，其实是浪费在琢磨这些公式：它们到底要表达的什么样一个“数据结构”或者“操作”！这个“琢磨”的过程，其实就是程序语言里所谓的“语法分析”（parse）。

这种问题在微积分里面就更加明显。微积分难学，很大部分原因，就是因为微积分的那些传统的运算符，其实不是很好的设计。如果你想了解更好的设计，可以参考一下 Mathematica 的公式设计。试试在 Mathematica 里面输入“单行”的微积分运算（而不使用它传统的“2D语法”）。

其实 Lisp 已经可以轻松地表示这种公式，比如对 `x^2` 进行微分，可以表示成

```
(D '(^ x 2) 'x)
```

看到了吗？微分不过是一个用于处理符号的函数 `D`，输入一个表达式和另一个符号，输出一个新的表达式。

同样的公式，传统的数学符号是这个样子：

$$\frac{d}{dx}(x^2)$$

这是什么玩意啊？`d` 除以 `dx`，然后乘以 `x` 的平方？

在 Lisp 里，你其实可以比较轻松地实现符号微分的计算。SICP里貌似有一节就是教你写个符号微分程序。做微积分这种无聊的事情，就是应该交给电脑去做。总之，这从一方面显示了，Lisp 的语法其实超越了传统的数学。

其实我一直都在想，如果把数学看成是一种程序语言，它也许就是世界上语法最糟糕的语言。数学里的“变量”，几乎总是没有明确定义的作用域（scope）。也就是说他们只有“全局变量”。上一段话的 x，跟下一段话的 x，经常指的不是同一个东西。所以训练有素的数学家，总是避免使用同一个符号来表示两种不同的东西。很快他们就发现所有的拉丁字母都用光了，于是乎开始用希腊字母。大写的，小写的，粗体的，斜体的，花体的，…… 而其实，他们只不过是想实现 C++ 里的 “namespace”。

可惜的是，很多程序语言的设计者没能摆脱数学的思想束缚，对数学和逻辑有盲目崇拜的倾向。所以他们继续在新的语言里使用中缀表达法。Haskell，ML，Coq，Agda，这些“超高级”的语言设计，其实都中了这个圈套。在 Coq 和 Agda 里面，你不但可以使用中缀表达式，还可以定义所谓的 “mixfix” 表达式。这样其实是把简单的问题复杂化。想让自己看起来像“数学”，很神秘的样子，其实是学会了数学的糟粕，自讨苦吃。

另外，由于 Lisp 的表达能力和灵活性比其他语言要大很多，所以类似 C 或者 Pascal 那样的语法其实不能满足 Lisp 的需要。在 Lisp 里，你可以写 (+ 10 (if test 1 2)) 这样的代码，然而如果你使用 C 那样的无括号语法，就会发现没法很有效的嵌入里面的那个条件语句而不出现歧义。这就是为什么 C 必须使用 test? 1 : 2 这样的语法来表示 Lisp 的 if 能表示的东西。然而即使如此，你仍然会经常被迫加上一对括号，结果让程序非常难看，最后的效果其实还不如用 Lisp 的语法。在 C 这样的语言里，由于结构上有很多限制，所以才觉得那样的语法还可以。可是一旦加入 Lisp 的那些表达能力强的结构，就发现越来越难看。JavaScript（node.js）就是对此最好的一个证据。

最后，从美学的角度上讲，S表达式是很美观的设计。所有的符号都用括号括起来，这形成一种“流线型”的轮廓。而且由于可以自由的换行排版，你可以轻松地对齐相关的部分。在 Haskell 里，你经常会发现一些很蹩脚，很难看的地方。这是因为中缀表达式的“操作符”，经常不能对在一起。比如，如果你有像这样一个 case 表达式：

```
case x
  Short _ -> 1
  VeryLooooooooooooooooooooooooog _ -> 2
```

为了美观，很多 Haskell 程序员喜欢把那两个箭头对齐。结果就成了这样：

```
case x
  Short _                          -> 1
  VeryLooooooooooooooooooooooooog _ -> 2
```

作为一个菜鸟级摄影师，你不觉得第一行中间太“空”了一点吗？

再来看看S表达式如何表达这东西：

```
(case x
  (-> (Short _) 1)
  (-> (VeryLooooooooooooooooooooooooog _) 2))
```

发现“操作符总在最前”的好处了吗？不但容易看清楚，而且容易对齐，而且没有多余的间隙。

其实我们还可以更进一步。因为箭头的两边全都用括号括起来了，所以其实我们并不需要那两个箭头就能区分“左”和“右”。所以我们可以把它简化为：

```
(case x
  ((Short _) 1)
  ((VeryLooooooooooooooooooooooooog _) 2))
```

最后我们发现，这个表达式“进化”成了 Lisp 的 case 表达式。

Lisp 的很多其它的设计，比如“垃圾回收”，后来被很多现代语言（比如 Java）所借鉴。可是人们遗漏了一个很重要的东西：Lisp 的语法，其实才是世界上最好的语法。