什么是语义学

很多人问我如何在掌握基本的程序语言技能之后进入“语义学”的学习。现在我就简单介绍一下什么是“语义”，然后推荐一本入门的书。这里我说的“语义”主要是针对程序语言，不过自然语言里的语义，其实本质上也是一样的。

一个程序的“语义”通常是由另一个程序决定的，这另一个程序叫做“解释器”(interpreter)。程序只是一个数据结构，通常表示为语法树(abstract syntax tree)或者指令序列。这个数据结构本身其实没有意义，是解释器让它产生了意义。对同一个程序可以有不同的解释，就像上面这幅图，对画面元素的不同解释，可以看到不同的内容（少女或者老妇）。

解释器接受一个“程序”(program)，输出一个“值”(value)。用图形的方法表示，解释器看起来就像一个箭头：程序 ===> 值。这个所谓的“值”可以具有非常广泛的含义。它可能是一个整数，一个字符串，也有可能是更加奇妙的东西。

其实解释器不止存在于计算机中，它是一个很广泛的概念。其中好些你可能还没有意识到。写 Python 程序，需要 Python 解释器，它的输入是 Python 代码，输出是一个 Python 里面的数据，比如 42 或者“foo”。CPU 其实也是一个解释器，它的输入是以二进制表示的机器指令，输出是一些电信号。人脑也是一个解释器，它的输入是图像或者声音，输出是神经元之间产生的“概念”。如果你了解类型推导系统 (type inference)，就会发现类型推导的过程也是一个解释器，它的输入是一个程序，输出是一个“类型”。类型也是一种值，不过它是一种抽象的值。比如，42 对应的类型是 int，我们说 42 被抽象为 int。

所以“语义学”，基本上就是研究各种解释器。解释器的原理其实很简单，但是结构非常精巧微妙，如果你从复杂的语言入手，恐怕永远也学不会。最好的起步方式是写一个基本的 lambda calculus 的解释器。lambda calculus 只有三种元素，却可以表达所有程序语言的复杂结构。

专门讲语义的书很少，现在推荐一本我觉得深入浅出的：《Programming Languages and Lambda Calculi》。只需要看完前半部分（Part I 和 II，100来页）就可以了。这书好在什么地方呢？它是从非常简单的布尔表达式（而不是 lambda calculus）开始讲解什么是递归定义，什么是解释，什么是 Church-Rosser，什么是上下文 (evaluation context)。在让你理解了这种简单语言的语义，有了足够的信心之后，才告诉你更多的东西。比如 lambda calculus 和 CEK，SECD 等抽象机 (abstract machine)。理解了这些概念之后，你就会发现所有的程序语言都可以比较容易的理解了。