编译原理 SICP 修改

SICP中环境模型可以等价为龙书中(第七章)讲的的运行时刻环境么?

修改

SICP看到第三章，求值的环境模型。感觉很多概念和程序运行时的堆栈，活动记录(栈帧)有相似的地方。SICP中的环境模型和编译原理中的栈，活动记录，访问...显示全部

关注问题 写回答 邀请回答 添加评论 分享 邀请回答 举报 ...

查看全部 3 个回答

Felis sapiens
函数式编程、编程语言、编程 话题的优秀回答者

头顶青天红美铃、草莓大福、Thomson、圆角骑士魔理沙、彭飞等 50 人赞同了该回答

不是一个层次的东西。

龙书7.2 Stack Allocation of Space描述的Stack是一种用来实现函数调用的机制，每次函数调用会压一个栈帧，栈帧内包含了函数的本地变量。C/C++编译器都是这么干的。问题在于：

1. 这个模型不管尾递归优化。
2. 这个模型不管闭包。

第一个问题，尾递归优化的编译怎么办？scheme里是用continuation来做这件事，按下不表。

第二个问题，闭包是怎么回事？C没有闭包，C也没有nested function，你在函数定义里除了参数和本地变量之外，用到的其他变量都在global environment里，编译期就能把它们内存位置确定然后送进函数体里去。

对于有高阶函数的语言而言，当然就需要引入SICP3.2 The Environment Model of Evaluation里提到的闭包了，其实就是对每个lambda表达式求值时，要把当前环境捆进来，要不然以后这个函数拿出来apply的时候，遇到自由变量就傻逼了。。

解释器这么写是为了让你明白lexical scoping的工作原理如此。至于真正编译的时候是否直接用龙书描述的call stack？不直接用的话，程序怎么变换，闭包怎么优化？那就是其他的事了，欲知详情可以看Compiling with Continuations一书。
闭包完全可以用面向对象来模拟，某种程度上它们是一回事。比如你用C++11对你自定义的类的对象sort，需要一个比较符，C++11以前你需要传函数指针，或者一个实现了operator ()的函数对象。其中函数对象只要实现了operator ()可以比较两个对象返回bool就行，假设它是Comp类的，那么Comp类还可以有其他的成员，你把这其他的成员当作函数式语言的闭包里的free variable就行了。。然后执行这个类的构造函数实际上跟对lambda表达式求值，生成闭包，把环境加进来差不多就是一回事。如果这样的函数类里有其他的函数类成员，那就是nested function。。C++11里就可以直接写lambda表达式了，而编译器的处理方式就是生成了匿名类来做这事。

发布于 2014-12-29

▲ 赞同 50 ▼ 2 条评论 分享 收藏 感谢 ...

更多回答

RednaxelaFX
编程、编译原理、编程语言 等 7 个话题的优秀回答者

68 人赞同了该回答

关于作者

Felis sapiens

函数式编程、编程语言、编程 话题的优秀回答者

电影旅行敲代码、Antokha Yuuki、暮无井见铃也关注了她

回答	文章	关注者
624	40	14,871

已关注 发私信

被收藏 11 次

编程语言与编译原理 酿酿酿酿酿泉 创建	5,338 人关注
programming-compiler Jade Shan 创建	3 人关注
关于编程 李城 创建	2 人关注
工作--技术积累 弈尘 创建	2 人关注

@邵成的答案说了问题的一方面——概念的差异，我觉得讲得不错，但是应该把另一方面——概念的相似点——也补充上。我下面的答案基本上是邵成的答案的改述。

SICP 3.2说的environment of evaluation是一个抽象概念，讲编程语言原理的书上说的activation record（活动记录）也是抽象概念，这俩可以说是同一个东西。EE和AR的重点都是(名字 -> 值)的binding。

当一个语言的函数调用能满足LIFO顺序时，它的activation record就可以按照LIFO顺序组织起来。以这种方式组织起来的AR就形成了调用栈（call stack），此时AR就可以叫做栈帧（stack frame）。
由于这种情况在主流语言中很常见，硬件为了加速调用栈的实现，就设计了专门的硬件支持，也就是常见的栈指针寄存器（stack pointer register，例如x86上的ESP/RSP，ARM上的R13），以及能够间接操纵栈指针的push/pop系指令。使用这样的硬件支持的栈通常叫做native stack；由于C语言的实现广泛使用native stack，现代操作系统又常由C实现，这种栈也常被称为“the C

展开阅读全文

冯东
编程、Linux 话题的优秀回答者

3 人赞同了该回答

SICP Environment Model 和语言中的 closure 并不是对等的。特别是 SICP Chapter 3 的 model 是简化的。理解 SICP Environment Model 和常用的 closure 的差别，就能解释很多东西。因为「龙书」更强调为实现实用的编译器，而 SICP Chapter 3 只是一个理想模型。

首先是这个 Chapter 3 model 并不管理资源的回收。它的资源回收是通过 Scheme GC 来完成的。如果使用没有 GC 的语言来举例子，就无法忽略资源回收这方面。加入资源回收之后，Evniorment Model 就会大幅度向传统的 stack 靠拢。所以 Chapter 3 看似理想，是因为倚靠了 GC 来掩盖了 stack 这个大多数程序员都熟悉的概念。

第二，这个 model 完全没有考虑性能，以至于它非常强大，涵盖了一切 create function 的方式。包括 eval。而现实中，closure 出于性能和资源回收的考虑，只支持 in-source-code nested function creation。我认为这就是造成著名的 eval 语义不严格问题的原因。实用语言的 eval （比如 Lua 中的 load string ）通常不会支持 Environment Model。这样 Environment Model 中大量的 runtime overhead 可以提前到 compile time 来完成。这也是 SICP Chapter 5 的主要内

展开阅读全文

查看全部 3 个回答

编译原理与程序设计原理 1 人关
张梓铖 创建

相关问题

SICP 是不是被高估了? 36 个回答

如何评价 Why MIT stopped teaching SICP? 7 个回答

如何看待Berkeley开设的CS61A:SICP in Python课程? 8 个回答

SICP换零钱迭代方法实现，是如何写的? 18 个回答

如何评价学军中学和SICP? 22 个回答

相关推荐

淼懂物理学：理解世界的极简指南
共 31 节课 ▶ 试听

新环境下，人文社科如何致用
★★★★½ 127 人参与

七周七并发模型
Paul Butcher
113 人读过 阅读