作业 2（主要部分）:Boa 解释器

1.0 版

截止时间:2022 年 9 月 23 日星期五 22:00

这项任务的目的是获得使用 monad 构建简单解释器的实践经验。在开始工作之前,请通读整个作业文本。

注意该作业还包括一系列简单的热身练习,如 Absalon 中所述。对于那些,您只需要提交您的工作代码,而不是单独的设计/实施文档、评估或测试证据。如果您想就热身练习的解决方案交流任何其他内容,请将您的评论作为注释放在源代码中。

1 博阿语

Boa 语言是 Python 3 的一个很小的子集,其意图是有效的 Boa 程序通常应该给出与在 Python 中运行时相同的结果。在本次作业中,我们将只使用 Boa 的抽象语法;下周,我们将考虑具体的语法和解析。

Boa 使用一种结构化值,由以下代数数据类型给出:

```
数据值=
    无值
  |真值|假值
  | IntVal 整数
  | StringVal 字符串
  | ListVal [值]
```

也就是说,Boa 值是三个特殊原子之一,即 None、True 或 False、整数、字符串或（可能为空的）值列表。

Boa 表达式具有以下形式之一:

```
数据Exp =
    常量值
  |变量名称
  |操作操作 Exp Exp
  |不是经验
  |呼叫 FName [Exp]
  |列表[Exp]
  |购买Exp [CClause]

类型VName =字符串类型
FName =字符串
```

```
数据操作=加|减号|时代|分区|国防部|方程|减|更大|在

数据Cclause =
      CCFor VName Exp
    | CCF Exp
```

表达式的预期语义应该不足为奇,并带有以下注释:

- Const v 的计算结果为值 v。
- Var x 计算为当前绑定到变量x 的值。如果 x 没有当前绑定,
  它的评估表明一个错误。
- Oper o e1 e2将运算符 o 应用于e1和e2的值(必须按该顺序计算)。算术运算符(Plus,...,Mod)的含义是显而易见的,两个参数都必须是整数(否则会发出错误信号)。尝试除以 0 或取模也表示错误。

  比较运算符(Eq、Less、Greater)返回(Boa 原子)True 或 False,这取决于参数之间的对应关系是否成立。 Eq 可用于比较任意值的结构相等性,而其他两个只能比较整数。最后,运算符 In 要求其第二个参数值是一个列表,并检查第一个参数值是否出现(由 Eq-test 确定)

  那个清单。
- Note e 返回其参数值的逻辑否定。为此,原子 None 和 False、整数 0、空字符串和空列表被认为代表虚假(因此它们的否定返回 True),而所有其他值被认为代表真实(因此返回 False否定时)。
- Call f [e1,...,en] (其中 n ⩾ 0)计算参数元组(e1, ..., en) (从左到右),然后根据结果值调用内置函数 f .目前,Boa 中只有两个这样的函数:

  – range 可以在 1 到 3 个整数参数上调用。最通用的形式,带有参数(n1, n2, n3),生成从n1到(但不包括!) n2 的整数列表,步进n3。例如,取n1 = 3、 n2 = 10(或 11)和n3 = 2 将生成列表值 [3, 5, 7, 9]。步长n3可以是正数或负数,但不能为 0。如果n1 ⩾ n2 (当n3 > 0 时)或n1 ⩽ n2 (当n3 < 0 时),结果列表将为空。

    如果只提供了两个参数,则它们用于n1和n2,其中n3取为 1;并且单个参数用作n2 的值,其中n1 = 0 和n3 = 1。 - print 可以在任何数量的任何类型的参数上调用。它将所有参数打印在一行上,由单个空格分隔。简单值(原子和整数)以自然方式打印。字符串直接打印,没有任何外部引号。1列表打印在“[”和“]”之间,元素分隔

1请注意,在 Python 中,当字符串不是打印的直接参数,而是出现在列表或其他正在打印的数据结构中的更深处时,它会被格式化为可以轻松解析回的样式。具体来说,这些字符串被包围通过单引号 (“ ”),字符串中的任何引号、反斜杠或换行符将分别呈现为两个字符序列“\ ”、“\\”和“\n”。您可以(但不是必须)在 Boa 中实现这种改进的行为。

由“， ”（逗号和空格）。调用 print 的结果总是只是特殊的
原子无。

例如,调用 print 参数计算为：

[IntVal 42, StringVal foo , ListVal [TrueVal, ListVal []], IntVal (-1)]

应该导致以下输出行（没有前导或尾随空格）：

42 富 [真,[]] -1

尝试调用上述两个函数之外的任何其他函数都会发出错误信号。

- List [e1,...,en] （其中 n ⩾ 0)简单地将表达式e1, ..., en （从左到右)计算为值,并将它们打包为单个列表值。

- Compr e0 [cc1,...,ccn] (其中 n ⩾ 0) 是一个列表推导式,本质上类似于 Haskell 的 [e0 | q1, ..., qn]。这里,Boa for-clause CCFor xe 对应于 Haskell 生成器 x <- e。 （如果 e 不计算为列表值,则会发出错误信号。） x 的绑定可能会影响 x 的任何先前绑定,并且在所有以下子句以及主体表达式e0 中都是可见的。

  Boa if 子句 CCIf e 对应于 Haskell 中的布尔守卫,其中 e 的值被解释为真值,其方式与上述非表达式相同,因此,例如,CCIf (List []) 将是被认为是失败的守卫,而 CCIf (ConstVal (IntVal 7)) 会成功。

  请注意,在 Boa 的具体语法中,我们稍后将要求 n ⩾ 1,并且 cc1是 CCFor 子句（不是 CCIf）。然而,像 Haskell 一样,Boa 抽象语法的语义没有施加这样的限制。

  另外,请记住,所有 Boa 表达式（包括但不限于e0)可能会打印输出作为评估的副作用。

最后,Boa 程序由一系列语句组成：

```
类型程序= [Stmt]

数据标准=
    SDef VName Exp
  | SExp Exp
```

定义语句 SDef xe 将 e 计算为一个值,并将 x 绑定到该值以用于序列中的其余语句。表达式语句 SExp e 只计算 e 并丢弃结果值。 （但任何打印和/或评估 e 产生的错误仍然有效。）

每当发出错误信号时,评估或执行会立即停止并显示错误消息,并且所有当前的变量绑定都将被丢弃。但是,仍然会保留在错误发生之前成功生成的任何输出行。

一个更大的 Boa 程序示例,包括具体和抽象语法,以及
预期输出,可在附录 A 中找到。

2 优秀的翻译

可以在模块 BoaAST 中找到上述定义（带有数据类型声明的通常派生子句）。解释器本身存在于模块 BoaInterp 中,并引入了以下类型：

```
输入Env = [(VName, Value)]

数据运行错误= EBadVar VName | EBadFun FName | EBadArg 字符串推导(Eq, Show)
```

我们选择将环境表示为简单的关联列表（即[(x1,v1),...,(xn,vn)]形式的列表,其中列表中变量 x 的第一个绑定（如果有)被视为当前的。

我们区分了三种运行时错误： EBadVar x 表示试图访问未绑定变量 x;相应地,EBadFun f 表示我们试图调用未定义的函数 f。所有其他可能的错误都与将无效参数或参数列表传递给 Boa 构造或函数有关,并表示为 EBadArg s,其中 s 是一些信息丰富的、人类可读的错误消息。解释的 Boa 程序中的所有错误都应该通过 RunError 报告,而不是通过调用 Haskell 错误函数。 （解释器本身的实际缺陷,例如未处理的 Boa 功能,或以某种方式达到“不可能”的情况,可能（如果相关)仍然使用错误。）

Boa 计算的主要类型构造函数如下：

```
newtype Comp a = Comp {runComp :: Env -> (Either RunError a, [String])}
```

也就是说,计算对环境具有（只读)访问权限,并返回运行时错误或预期类型的结果,以及（在任何一种情况下)计算产生的可能为空的输出行列表。

使 Comp 成为 Monad 的实例（以及 Functor 和 Applicative,但您可以使用通常的样板代码）。在报告中,简要说明您的 return 和 >>= 函数是如何工作的。然后,为 monad 定义以下关联操作： ***

```
abort :: RunError -> Comp一看::
VName -> Comp Value withBinding ::
VName -> Value -> Comp a -> Comp a output :: String -> Comp ()
```

在这里,abort re用于发出运行时错误re 的信号。查看 x 返回变量 x 的当前绑定（如果 x 未绑定,则表示 EBadVar x 错误）。相反,除了任何其他当前绑定之外,使用绑定 xvm 的操作运行计算 m,其中 x 绑定到 v。 （任何先前对 x 的绑定在运行 m 时暂时无法访问。)最后,输出 s 将行 s 附加到输出列表。 s 不应包含尾随换行符（除非该行来自打印本身包含嵌入换行符的字符串值）。

您的其余代码（除非下面明确指出)不应依赖于 Comp a 类型的确切定义;也就是说,它既不应该直接使用值构造函数 Comp,也不应该直接使用投影runComp,而总是通过上述函数之一。

接下来,定义以下辅助函数：

```
truthy :: Value -> Bool操作:: Op
-> Value -> Value -> Either String Value apply :: FName -> [Value] -> Comp
Value
```

如前所述,真值 v 只是确定值 v 代表真还是假。 operation o v1 v2将运算符 o 应用于参数v1和v2,返回结果值,如果一个或两个参数不适合

手术。类似地,apply f [v1,...,vn]将内置函数 f 应用于（已评估的)参数元组v1, ..., vn,如果 f 不是有效的函数名（EBadFun）,可能会发出错误信号),或者参数对于函数 (EBadArg) 无效。

最后,定义主要的解释器函数,

```
eval :: Exp -> Comp Value exec ::
Program -> Comp () execute ::
Program -> ([String], Maybe RunError)
```

eval e 是在当前环境中计算表达式 e 并返回其值的计算。同样,exec p 是执行程序（或程序片段)p 产生的计算,没有名义上的返回值,但 p 中的任何副作用仍在计算中发生。最后,execute p 显式返回输出行列表,以及在初始环境中执行 p 产生的错误消息（如果相关）,其中不包含变量绑定。 （为了（仅)实现执行,您可以使用 monad 类型的 runComp 投影。)在报告中,至少解释一下您是如何实现 Boa 的列表***推导式的。

提示:在实现推导式的 eval 时,首先覆盖列表中只有一个子句的情况,然后尝试逐步将您的代码推广到其他子句列表。

驱动程序

为了您的方便,我们为您的解释器提供了一个简单的独立包装器:stack build 将创建可执行的 boa,然后您可以将其作为 stack run boa -- -i program.ast 运行,其中 program.ast 是抽象语法程序树。 （目录examples/中提供了一些示例）。特别是,这将以 Boa 用户可以看到的形式向您显示解释程序的打印输出。

[本文档的其余部分与作业 1 的内容基本相同,但在此重复以方便参考。任何重大变化都以红色突出显示。]

3 上交什么

3.1 代码

表单为了促进人工和自动反馈,请务必严格遵循本节中的代码打包说明。我们为热身和主要部分中的所有请求功能提供骨架/存根文件。这些存根文件打包在分发的 code.zip 中。它包含一个目录 code/,有几个子目录组织为 Stack 项目。您应该按照指示编辑提供的存根文件,并保持其他所有内容不变。

至关重要的是,您不要更改任何导出函数的提供类型,因为这会使您的代码与我们的测试框架不兼容。此外,不要删除您未实现的任何功能的绑定;只需将它们保留为未定义即可。

提交作业时,将您的代码再次打包为单个 code.zip （不是.rar、.tar.gz 或类似文件）,其结构与原始文件完全相同。重建 code.zip 时,请注意仅包含构成您实际提交的文件:您的源代码和支持文件（构建配置、测试等）,但不要包含过时/实验版本、备份、编辑器自动保存文件、修订版-控制元数据、 .stack-work 目录和

类似。如果您的最终 code.zip 比分发的版本大很多,那么您可能包含了一些您不应该包含的内容。

对于热身部分,只需将您的函数定义放在 code/part1/src/Warmup.hs 中,如果有指示。

对于主要部分,您的代码必须放在文件code/part2/src/BoaInterp.hs 中。它应该只导出请求的功能。任何测试或示例都应放在 code/part2/tests/ 下的单独模块中。为了获得灵感,我们在 code/part2/tests/Test.hs 中提供了一个非常简约(而且远非完整)的测试套件。如果您正在使用 Stack(为什么不使用?),您可以通过目录 code/part2/ 中的堆栈测试来构建和运行套件。

此分配的定义(例如,类型 Exp)可在文件 .../src/BoaAST.hs 中找到。
您应该只导入这个模块,而不是直接将其内容复制到 BoaInterp。当然,不要修改 BoaAST 中的任何内容。

内容与往常一样,您的代码应该有适当的注释。特别是,尝试为您定义的任何辅助“帮助”函数提供简短的非正式规范,无论是本地的还是全局的。另一方面,避免用英文改写代码在 Haskell 中已经清楚表达的内容的琐碎注释。尝试使用一致的缩进样式,并避免超过 80 个字符的行,因为这些通常包含在打印列表中(或者如果有人使用比您更窄的编辑器窗口),使它们难以阅读。

您可以(但不应该为此作业)仅从核心 GHC 库导入附加功能:您的解决方案代码应使用提供的 package.yaml 使用从目录 code/partn/ 发布的堆栈构建进行编译。对于您的测试,您可以使用 Stack LTS 发行版的课程强制版本中的其他相关包。

我们强烈建议使用 Tasty 来组织您的测试。
在您的测试套件中,请记住还包括任何相关的负面测试用例,即验证您的代码是否正确检测并报告错误情况的测试。此外,如果您的代码中已知某些功能缺失或错误,则相应的测试用例仍应与给定输入的正确预期输出进行比较(即测试应该失败),而不是与您的代码当前返回的任何不正确结果进行比较.

理想情况下,您的代码在使用 ghc(i) -W 编译时应该不会发出警告;否则,请添加注释,解释为什么任何此类警告在每个特定情况下都是无害或不相关的。
如果代码中的某些问题根本无法编译整个文件,请务必在提交之前注释掉有问题的部分,否则所有自动化测试都会失败。

3.2 报告

除了代码之外,您还必须提交一份简短的(通常为 1-2 页)报告,包括以下两点,仅用于主要(非热身)部分:

- 记录您所做的任何(重要的)设计和实施选择。这包括但不一定限于回答作业文本中明确提出的任何问题(在空白处标有*** ,以加强强调)。专注于高层次的方面和想法,并解释为什么你做了一些不明显的事情,而不仅仅是你做了什么。在报告中逐个功能地进行详细的代码演练是非常不合适的;关于函数如何工作的技术注释属于代码中的注释。

- 对您提交的代码的质量以及它满足作业要求的程度(尽您的理解和知识)进行诚实、合理的评估。务必清楚地解释任何已知或可疑的缺陷。

记录评估的基础是非常重要的（例如,一厢情愿、零散的例子、系统测试、正确性证明？）。包括您在源提交中编写的任何自动测试,明确如何运行它们,并在报告中总结结果。如果您的代码的某些方面或属性无法以自动化方式轻松测试,请解释原因。

我们强烈建议（并且可能会在以后强制要求)将您的评估分为以下小节/段落：

完整性是否至少在原则上实现了所有要求的（以及可选的)功能,即使不一定完全工作?如果没有,您对如何实现缺少的部分有任何具体的想法吗？

正确性所有实现的功能是否都能正常工作,或者是否存在已知的错误或其他限制?在后一种情况下,您对如何潜在地解决这些问题有任何想法吗？

效率您的代码的运行时时间和空间使用（正如您期望它由 Haskell 执行的那样;您不需要实际对其进行基准测试)是否合理地匹配,至少渐近地,从正确的实现中自然会假设什么?如果没有,您是否有关于如何显着提高代码性能的想法？

鲁棒性在相关的地方（可能无处可去）,您的代码（包括所有导出的函数,而不仅仅是执行)在超出规范使用时表现如何,即,当给定的输入可能是 Haskell 类型正确但仍然非法/由于一些更复杂的原因无效?根据定义,对于这种情况,没有规定的“正确”行为,但最好还是“合理”:以信息性错误停止可能是明智的;可能不会因 Haskell 模式匹配错误或类似错误而崩溃。（注意,对于解释的 Boa 程序引起的运行时错误（例如,被零除）,解释器的正确行为是指定的,即返回相关的 RunError。）

可维护性通用代码片段是否通过参数化辅助定义合理共享,或者是否存在大量复制粘贴片段形式的代码重复,并进行了微小的更改？（请注意,这个问题可能也适用于您的测试套件！)您的代码是否尊重单子抽象,仅通过相关操作函数（中止、查看等)表达所需的功能,而不是直接依赖 Comp 的实现?否则代码是否处于您认为良好的状态（正确布局、注释等）？

其他您认为值得一提的其他任何事情,无论是正面的还是负面的。

前两点应该通过参考你的正式测试的结果来证实。对于其他人,您还应该通过相关示例或其他证据来证明您的评估是正确的。

您提交的报告应该是一个名为 report.pdf 的 PDF 文件,与（不是在里面！)code.zip 一起上传。该报告应包括您的代码和测试列表（但不包括已经提供的辅助文件)作为附录。

3.3 时间表

在今年的 AP 课程中,我们特别关注学生在课程中的工作量。帮助我们获得比正式课程提供的更详细、更完整、更及时的画面

在评估结束时,我们要求您填写每周的简短时间表,详细说明您在作业的各个部分和方面以及其他与课程相关的活动上花费了多少时间。

时间表模板位于主代码目录中的单独文件 timesheet.txt 中,并且是机器处理的,因此正确填写它很重要。

特别是,对于每个时间类别,您应该以自然格式报告您花费的时间(小时和/或分钟),例如“2 小时”、“15 米”、“120 米”、“1 小时 30 米”等。记住包含单位特别重要,因为没有默认值。对于花费不到一个小时左右的活动,尽量做到精确;不要只是将它们四舍五入到最接近的小时。

如果您不知道或不记得您在特定类别中花费了多少时间,请给出您的最佳估计。如果您对一个或多个类别没有有意义的数字,或者不想说,只需将时间写为单个“x”。如果您以 2 人小组的形式提交作业,请报告每个类别的平均时间使用情况(即,您个人贡献的总和除以 2)。最后,如果你以前学过 AP(或者知道其他地方的材料),那么了解你本周在课程上花费的实际时间对我们来说比你估计需要多少时间更有用。一切都是第一次。

您可以包括注释(以“#”开头并一直运行到行尾)解释或详细说明您的数字,特别是如果您认为它们可能是非典型的。但是,时间表主要用于自动处理,因此任何文本评论都可能无法系统地注册。如果您有任何重要的观点或意见想要引起我们的注意,您应该直接向教学人员提出。

为帮助确保对类别的解释一致,请使用以下指南
计算各种课程相关活动的线路:

分配(不是实际的类别。留作数字2以供识别)。

安装GHC/Stack 在您的平台上安装和运行所花费的时间。不要包括不必要的设置调整(例如让语法突出显示在您最喜欢的编辑器中工作,更不用说尝试正确配置任何花哨的 IDE 功能;本课程不需要这些。)这个类别可能为零作业 2,假设您为作业 1 进行了必要的安装和设置。

本周花在推荐阅读材料上的时间,即使与作业没有直接关系,以及您专门为在作业中取得进展而进行的任何补充阅读(可能来自建议来源以外)。

讲座花在听讲座上的时间。如果您出于某种原因跳过(部分)讲座,这可能少于标称的每周 4 小时。

探索在非强制性活动中花费在 Haskell 编程上的时间,无论是基于建议的练习还是您自己的实验。这还包括作业的可选部分的工作,无论您最终是否将其交给反馈。

热身作业热身部分所花费的时间。

开发时间花在编写和调试分配的主要部分的代码上
ment,包括您在开发期间进行的任何集成或临时测试。

测试时间,超出上述开发时间,专门用于以自动化测试套件的形式记录您的测试。请注意,如果您在代码之前或与代码一起编写了相关的测试用例,这可能（接近)为零。

报告编写设计/实施决策和评估的时间。

其他上述类别未正确涵盖的任何其他与课程相关的活动。如果您在这里报告了重要的时间,那么在评论中包含一个简短的解释性说明会很有帮助。

如果上述方向中的任何内容不清楚（并且会严重影响您报告的数字）,请在论坛上寻求澄清。请注意,类别和/或其描述可能会根据需要为以后的分配进行调整。

您的时间表编号（或明显缺少编号)不会影响您的作业成绩,也不会被助教评论。但我们希望您能尽可能准确和完整地回答,以帮助我们正确了解哪些活动对 AP 工作量有显着影响,以及哪些具体的纠正措施可以为以后的分配和/或未来运行课程。

你不必用秒表或类似的东西给自己计时,但一定要尝试在名义上专门用于特定活动的时间段中考虑任何重大的暂停或中断（例如,午休时间）。您可能会发现一个专用的时间跟踪工具（例如 clockify.me 或任何数量的免费应用程序)通常有用且信息丰富,不仅适用于 AP。

3.4 概述

详细的上传说明,特别是关于团体提交的物流,可以在 Absalon 提交页面上找到。

我们还希望提供一个自动化系统,就您计划提交的代码向您提供初步反馈,包括形式、正确性、样式等问题。强烈建议您利用这个机会来验证您的提交,并且 - 如有必要 -修复或以其他方式解决（例如,通过记录已知缺陷)它的任何合法问题

揭露。

但是请注意,通过我们的自动化测试并不能替代进行和记录您自己的测试。您的评估必须能够独立进行,而无需依赖我们工具的输出。

AA 样例 Boa 程序

这是一个更大的 Boa 程序示例,采用具体的类似 Python 的语法以提高可读性（在讲义中以文件 code/part2/examples/misc.boa 的形式提供）：

```
squares = [x*x for x in range(10)];打印（[123， [正方形，
打印（321）]]）； print(  奇数平方：  , [x for x in
squares if x % 2 == 1]); n = 5;复合材料= [j for i in range(2, n) for j in range(i*2, n*n, i)];
print(  打印下面的所有素数  , n*n); [print(x) for x in range(2,n*n) if x not incomposite]
```

这是相应的抽象语法,作为 Program 类型的 Haskell 值（文件 .../misc.ast）：

```
[SDef “正方形”
        (Compr (操作时间(Var  x  ) (Var  x  ))
                [CCFor  x  (调用  range  [Const (IntVal 10)])]),
 SExp （调用“打印” [List [Const （IntVal 123），
                                列表[Var  squares  ，
                                        调用“打印” [Const (IntVal 321)]]]]),
 SExp （调用“打印” [Const （StringVal “奇数平方：”），
                        Compr (Var  x  ) [CCFor  x  (Var  squares  ),
                                        CCIf (Oper Eq (Oper Mod (Var  x  )
                                                                (常数 (IntVal 2)))
                                                        (常量(IntVal 1)))]]),
 SDef  n  (Const (IntVal 5)),
 SDef “复合”
        (Compr (Var  j  ) [CCFor  i  (Call  range  [Const (IntVal 2),Var  n  ]),
                        CCFor  j  (调用  range  [操作时间(Var  i  )
                                                        (常数 (IntVal 2)),
                                                操作时间(Var  n  ) (Var  n  ),
                                                其中“我” ]]),
 SExp (调用  print  [Const (StringVal  Printing all primes below  ),
                        操作时间(Var  n  ) (Var  n  )]),
 SExp (Compr (Call  print  [Var  x  ])
                [CCFor  x  (调用  range  [Const (IntVal 2),
                                        操作时间(Var  n  ) (Var  n  )]),
                 CCIf (Not (Oper In (Var  x  ) (Var  composites  )))])]
```

运行时,该程序应打印以下行,所有行都没有前导或尾随空格（文件 .../misc.out）：

```
321
[123, [[0, 1, 4, 9, 16, 25, 36, 49, 64, 81], 无]]
奇数方块:[1, 9, 25, 49, 81]
打印所有低于 25 的素数 2

3
5
7
11
13
17
19
23
```