EDSL相关杂记（1）

Felis sap...
函数式编程、编程语言、编程 话题的优秀回答者

本期专栏热热身，讲述EDSL的几个背景问题。

EDSL好处都有啥，谁说对了就给他

既然是EDSL，那么我们一般直接用宿主语言的表达式来构成EDSL的AST了。这样做的一些好处：

- 不用写解析器；可以将宿主语言的函数当成宏使用；特定类型的EDSL可以借用宿主语言的一些语法糖，比如monadic EDSL可以用Haskell的do-notation写得好看。
- 对于有简单类型系统的EDSL，可以在宿主语言里声明带类型AST，然后借助宿主语言的类型推导，省去实现类型检查，不用额外标类型也保证类型安全。
- 实现带绑定的EDSL（譬如函数参数等，而且需要实现静态作用域），可以用HOAS实现AST中的绑定，将宿主语言的substitution机制偷来实现EDSL的substitution。如此可以静态排除scope错误，substitution操作保证是安全（capture-free）的。

语法方面的另一个好消息是，我们可以先实现一门EDSL，然后借助Template Haskell和QuasiQuotes扩展，为其实现一个concrete syntax及其parser——这个parser在Haskell编译期运行，生成Haskell EDSL的AST，如果有语法错误是编译期报告的。

类型方面，我们的EDSL也许是单类型/多类型的，可能是动态类型，或者带更高级的一些类型特性（比如Session Type之类）。得益于Haskell强悍的类型系统，许多带复杂类型的EDSL都可以编码。类型安全性和便捷性存在一定冲突，之后会有例子讲到。

Reification

任务——在宿主语言里，将宿主语言的函数“序列化”成first-order的语法树！这是function reification，另外还有monadic reification（可以视作前者的特殊情况），对应EDSL是用一个monad表示的情况（命令式语言常见）

别忘记我们的Haskell是一门木有运行时反射的语言，使用GHC魔法在运行期将一个函数值像剥洋葱一样剥开看定义然后数着参数一个个生成字符串参数名什么的，木有这种事！那么reification能做到什么程度呢，HOAS真的对代码生成不友好？默念九字真言，“类型是人类的好朋友”，具体做法将在讲higher-order/monadic EDSL的章节揭晓，而且简单到像作弊！

Expression Problem

说说所谓Expression Problem：怎样让EDSL可扩展（extensible）、可组合（composable）。可扩展意味着同一个EDSL可以方便地增加新的操作，或者说后端——能够解释执行、序列化，或者通过不同的pass进行一定优化处理后，生成某个目标平台的代码。可组合意味着可以将EDSL拆解成一系列不同的更小语言并方便地组合操作——比如将函数抽象与调用做成一个子集，算术运算做成一个子集，I/O操作做成另一个子集，对EDSL的某个操作单独在每个子集上实现，然后将其一组合就能实现对整个EDSL的该项操作。

举个例子，对Scheme熟悉的同学也许听说过Nanopass Framework的名字，这是Kent Dybvig等人提倡的实现教学/商业级编译器的架构，将编译过程组织为数十个pass，每个pass对一种AST略经修改变成新的AST。在Scheme这样的动态语言里实现Nanopass尚可，而在ML家族语言里，如果每次AST形状略有变化都要声明新的数据类型然后写一堆boilerplate，或者图麻烦搞个大数据类型然后用文档注明什么pass下什么形状的AST不可能出现，那Nanopass可就没搞头了撒。

下一期专栏会讲两种Expression Problem的解决方法，各有好坏。在之后具体讲述各类EDSL的实现方法时，我有时会忽略可组合性的要求，退回到用原始的ADT编码EDSL，以方便我和读者把注意力放在更适当的地方（

AST Typing Problem

可以看看Edward Yang的一篇博文。叫做AST Annotation Problem更合适，可以看作Expression Problem的延伸：怎样为AST节点方便地标注上额外信息，不影响不使用该信息的函数？那篇博文以及LtU讨论贴中总结了许多种不同的方法，其中一种（fixpoint functor）与下期专栏讲的Data types à la carte相关。之后的专栏不会涉及AST Typing Problem，我们关注的对象是类型安全的EDSL，并无在AST节点上标注parsing位置或类型信息的必要。

顺便说个我觉得更符合这名字的问题吧：静态类型的EDSL，所有表达式都在Haskell中写出的话，自然可以推导/标注（已知）类型，但如果需要运行期通过parsing等过程构建AST，如何处理未知的类型，且不伤害类型安全性？答案将在讲type-level EDSL的章节揭晓。

Observable Sharing

高效的代码，怎样将递归的AST转换为非递归的顶点带编号的图，common sub-expression指向同一顶点？之后讲first-order EDSL的例子时我会用两类不同方式解决这个问题：hash-consing，以及指针比较。

Deep/Shallow Embedding

Deep/Shallow Embedding的概念非常浅显，不过揭晓之前，先提另一对相关的概念吧：Syntax/Semantics。

Syntax特指Abstract Syntax，也就是EDSL“长什么样”，或者说怎样构造出EDSL的程序。一般来说会有两类构造，一类属于primitive，是最简单的EDSL表达式（比如常量等）；一类属于combinator，将其他EDSL程序组合起来生成新的程序。假如EDSL有AST，用ADT建模，那么primitive和combinator从该数据类型的构造器一看就知；但是EDSL设计者出于各种考虑常常会选择隐藏EDSL的定义，只export一个抽象的数据类型，以及一系列“smart constructor”（类型签名看起来像data constructor但也许不是）用于组合EDSL。

Semantics指语义，也就是EDSL“是啥意思”或者“能办啥事”。为EDSL实现了一个eval函数的话，也就实现了这个EDSL的指称语义（Denotational Semantics）。同一个EDSL，有可能实现不同的eval函数，从而赋予其多重语义。举例：

```haskell
-- Give me a Parser a, I give you a parsing function
runParser :: Parser a -> String -> Maybe (a, String)

-- Give me a State a, I give you a state-passing function
runState :: State s a -> s -> (a, s)

-- Give me an arithmetic expression, I give you the result
evalExpr :: Expr -> Double

-- Give me an arithmetic expression, I show you what it is
showExpr :: Expr -> String
```

如果EDSL的定义方式即嵌入了其语义，比如

```haskell
newtype State s a = State { runState :: s -> (a, s) }
```

这样的EDSL，采用的即是Shallow Embedding。Shallow Embedding的EDSL只允许一种解释方式。而如果EDSL的定义方式仅停留在语法层面，不指定任何语义，则称为Deep Embedding，比如：

```
Put :: s -> State s ()
```

（以上两种State怎样组合/执行暂时略去，有兴趣的同学可自行探究）

Deep/Shallow Embedding各自优缺点都很明显。Shallow Embedding只允许单一语义，不过在很多场合（例如Parsing组合子）下单一语义已经够用，采用Shallow可以简化实现；Deep Embedding将语法/语义分离开，使得EDSL的多种解释方式以及优化变得可能。

两种不同Embedding的关系并非水火不容，而是一体两面。Folding domain-specific languages: deep and shallow embeddings指出，Shallow实际上是Deep的数据类型上的fold（catamorphism）；Combining deep and shallow embedding of domain-specific languages提出了在Deep框架里整合Shallow扩展的方法。而下一期将要介绍的两种Expression Problem解决方式，都可以说是Shallow与Deep的某种结合。

编辑于 2016-10-09

Haskell 函数式编程 编程语言理论

文章被以下专栏收录

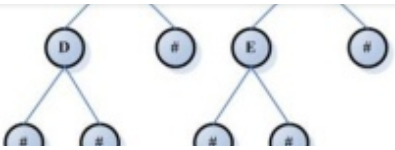
推荐阅读

:DSL相关杂记：预告

elis... 发表于雾雨魔法店

仙境里的Haskell（之四）——Functor类型类

大魔头-诺... 发表于魔鬼中的天...



仙境里的Haskell（之三）

大魔头-诺... 发表于魔鬼中的天...

仙境

大魔

2 条评论

切换为时间排序

写下你的评论...

包遵信 1 年前

题图是啥?

赞

岳鹏 回复 包遵信 1 年前

灰色的果实 入巢莳菜

赞 查看对话