

Análise do Sistema de Tipos de Haskell

Rafael Castro, Karina G. Roggia, Cristiano D. Vasconcellos

rafaelcgs10@gmail.com

Departamento de Ciência da Computação
Centro de Ciências e Tecnológicas
Universidade do Estado de Santa Catarina

09/10/2017

Sumário

- Introdução
- Análise Lógica do Sistema de Tipos de Haskell
- Análise Categórica do Sistema de Tipos de Haskell
- Conclusões

Introdução

- Proposições como tipos (Isomorfismo de Curry-Howard) estabelece uma relação entre provas em sistemas de tipos e provas em lógica construtivista.
- A relação é estendida para Teoria das Categorias por meio das Categorias Cartesianas Fechadas, o que resulta em Curry-Howard-Lambek.
- Frequentemente Haskell é apontado como linguagem de programação cujo sistema de tipos é baseado em lógica e Teoria das Categorias.

Objetivo deste trabalho é apresentar uma análise matematicamente coerente sobre tais alegações.

Análise Lógica do Sistema de Tipos de Haskell

- Em Haskell existe o termo `undefined :: a`, que serve para representar uma computação com erro ou um loop infinito. Funciona como um termo “coringa”, pois seu tipo unifica para qualquer outro. Qualquer tipo é habitado por ele.
- A lógica isomorfa ao sistema de tipos de Haskell tem todas as proposições como teoremas.
- Haskell não tem o tipo \perp , pois não há tipo não habitado.
- Portanto, o sistema de tipos de Haskell é trivial.

Análise Categórica do Sistema de Tipos de Haskell

Segundo a *Wiki* do Haskell:

Os objetos de **Hask** são tipos de Haskell e os morfismos dos objetos A a B são funções de Haskell do tipo $A \rightarrow B$. O morfismo identidade do objeto A é $id :: A \rightarrow A$, e a composição dos morfismos f e g é $f . g = x \rightarrow f(g x)$. (Tradução do autor)

Análise Categórica do Sistema de Tipos de Haskell

Segundo a *Wiki* do Haskell:

Os objetos de **Hask** são tipos de Haskell e os morfismos dos objetos A a B são funções de Haskell do tipo $A \rightarrow B$. O morfismo identidade do objeto A é $id :: A \rightarrow A$, e a composição dos morfismos f e g é $f . g = x \rightarrow f(g x)$. (Tradução do autor)

```
undef1 :: a -> b
undef1 = undefined
```

```
undef2 :: a -> b
undef2 = \_ -> undefined
seq undef1 1 = ⊥
seq undef2 1 = 1
```

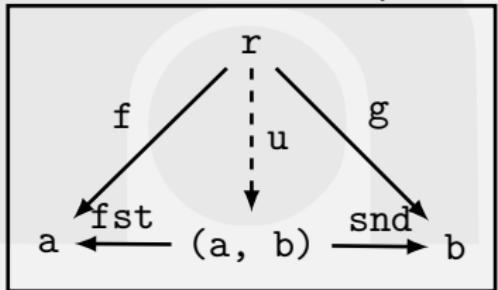
$undef1 . id = undef2 \Rightarrow undef1 . id \neq undef1.$

Categoria PHask

- a) $\text{Obj}_{\text{PHask}}$ é o conjunto de todos os tipos de Haskell.
- b) $\text{Mor}_{\text{PHask}}$ é o conjunto de todas as funções em Haskell (tipáveis) que são totais. Além disso, um morfismo $f : A \rightarrow B$ representa a classe de equivalência de funções do tipo A ao tipo B que tem o mesmo mapeamento.
- c) ∂_0 e ∂_1 são as funções que levam, respectivamente, uma função ao seu tipo de origem e destino.
- d) A composição é dada pela função $(.) :: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$ e a prova da associatividade é a mesma utilizada em **Set**.
- e) O morfismo identidade é gerado pelas funções identidade $\text{id} :: a \rightarrow a$, que pelo polimorfismo garante que todo objeto (tipo) tem um morfismo (função) identidade.
Observa-se que **PHask**, diferente de **Hask**, não acontece $\text{undef1} . \text{id} \neq \text{undef1}$.

A categoria PHask é uma categoria cartesiana fechada

- ① O objeto terminal é o tipo () .
- ② Utiliza-se o dado tupla padrão: o produto é dado pelos morfismos das funções $\text{fst} :: (\text{a}, \text{b}) \rightarrow \text{a}$ e $\text{snd} :: (\text{a}, \text{b}) \rightarrow \text{b}$, e pelo tipo:
 $\text{data } (\text{a}, \text{b}) = (,) \{ \text{fst} :: \text{a}, \text{snd} :: \text{b} \}$. Assim, para qualquer tipo r e para quaisquer funções $f :: r \rightarrow \text{a}$ e $g :: r \rightarrow \text{b}$ deve existir uma única função $u :: r \rightarrow (\text{a}, \text{b})$ capaz de fazer o diagrama abaixo comutar.



A melhor função u que garante a comutatividade desse diagrama somente pode ser:

$$\begin{aligned} u &:: r \rightarrow (\text{a}, \text{b}) \\ u \ r &= (f \ r, g \ r). \end{aligned}$$



- ③ O objeto exponencial em Haskell é formado pelo objeto $b \rightarrow c$ e pelo morfismo eval, onde b e c são tipos quaisquer e eval é gerado pela função

$$\text{eval} :: (b \rightarrow c, c) \rightarrow c$$

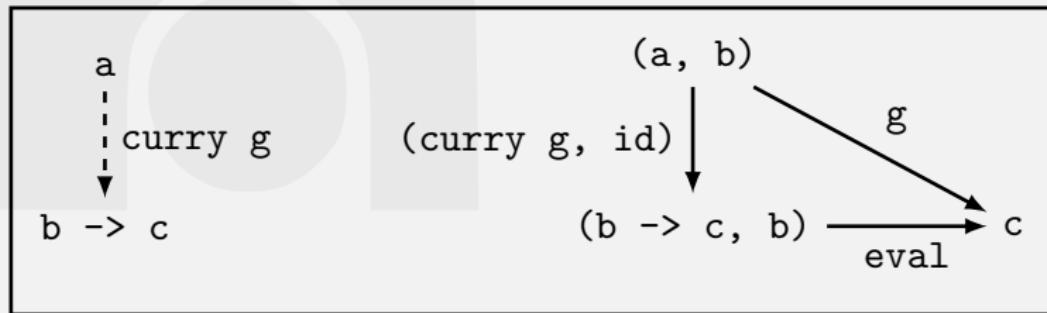
$$\text{eval } f = f \ x,$$

pois para todo tipo a e função $g :: (a, b) \rightarrow c$, existe um único morfismo curry g , onde

$$\text{curry} :: ((a, b) \rightarrow c) \rightarrow a \rightarrow b \rightarrow c$$

$$\text{curry } f = \lambda x \rightarrow \lambda y \rightarrow f(x, y),$$

que faz o diagrama abaixo comutar.



Análise Categórica de Classes de Tipos

- As classes de tipos do Haskell são utilizadas para definir algumas estruturas matemáticas, como relações (por exemplo: equivalência e ordem), semigrupos, monoides, funtores e mònadas.
- A classe **Functor** indica, por seu nome, representar as estruturas categóricas dos funtores, portanto deve ser um mapeamento entre categorias, o qual preserva origem e destino dos morfismos, identidade dos objetos e a composição
- Um construtor de tipos é uma função que recebe tipos como argumentos e retorna algum tipo, por exemplo o construtor **Either** recebe dois tipos **a**, **b** e retorna o tipo **Either a b**.
- ```
class Functor (f :: * -> *) where
 fmap :: (a -> b) -> f a -> f b.
```

# Análise Categórica de Classes de Tipos

- A assinatura de `fmap` garante a preservação da origem e do destino, porém a identidade e a composição não são asseguradas. Seria necessário que

`fmap id = id`

`fmap (f . g) = fmap f . fmap g`

## Conclusões

- Curry-Howard-Lambek é utilizado no sistema de tipos de Haskell, não somente para definir algumas classes de tipos, mas também para garantir o bom comportamento do sistema.
- Foi identificado alguns problemas em Haskell, em particular a ausência de uma semântica operacional que seria fundamental para definir uma categoria dos tipos e funções.
- Como alternativa foi construído a categoria **PHask** a partir de um subconjunto de Haskell com apenas funções totais.