



类型推导和合一算法的简单介绍

2011-06-04 黄毅

Hindley Milner类型系统最过瘾的就是 [类型推导](#) 功能。写程序的时候可以完全忽略变量和函数的类型，由编译器自动推导类型并做类型检查。可以说是静态语言和动态语言的完美结合。本文简单介绍类型推导的过程，以及其中的关键算法：[合一](#)。

先问一个问题，`flip id` 的类型是什么？我们知道 `id` 是原封不动地返回传给它的参数，类型是 `a -> a`；`flip` 是用来交换一个函数前两个参数的位置，类型是 `(a -> b -> c) -> b -> a -> c`，然而用 `id` 去调用 `flip` 返回的是个什么类型？心算起来还是有难度，幸好我们可以求助于 `ghci`：

```
Prelude> :t id
id :: a -> a

Prelude> :t flip
flip :: (a -> b -> c) -> b -> a -> c

Prelude> :t flip id
flip id :: b -> (b -> c) -> c
```

Haskell是如何算出这个结果的？这就是类型推导算法在起作用了。简单地说，类型推导过程分两部分进行：1. 根据类型系统规则产生一个方程组；2. 解方程组。




比如上面这个问题，我们先假定 `flip id` 的类型是 `x`，然后根据函数调用的规则以及 `id` 的类型，我们发现 `flip` 的类型应该是 `(a -> a) -> x`，同时我们已知 `flip` 的类型是：`(a -> b -> c) -> b -> a -> c`，这两个写法必须是等价的。根据这一点，我们就能得出一个方程组，通过合一算法解方程组，我们就可以得到 `x` 的值。

首先为了防止命名冲突，先进行必要的重命名，同时对 `flip` 的类型进行一点等价转换，现在两个类型便成为如下形式：

```
(d -> d      ) -> x
(a -> (b -> c)) -> (b -> a -> c)
```

根据其中的对应关系，我们可以得出这些一样等式：

```
d = a
```

[View document source](#)  . Generated on: 2019-06-29. Generated by [Docutils](#)  from [reStructuredText](#)  source.

Website content copyright © by 黄毅. All rights reserved.

```
d = b -> c
x = b -> a -> c
```

然后再通过一些等价替换就不难解出 x 的值了： $b \rightarrow (b \rightarrow c) \rightarrow c$ 。

我用Haskell写了一个 [合一算法的简单实现](#)，供大家学习参考，运行效果如下：

```
*Main> test (f [d, d, x]) (f [a, f [b, c], f [b, a, c]])
f(d,d,x)  <==>  f(a,f(b,c),f(b,a,c))
d -> f(b,c)
x -> f(b,f(b,c),c)
a -> f(b,c)
```

2条评论 精确编程  Disqus 隐私政策

 登录 ▾

 推荐 5  推文  分享

评分最高 ▾



加入讨论...

通过以下方式登录

或注册一个 DISQUS 帐号 

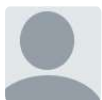
姓名



Xavier Wang (starwing) • 2年前

找到原因了，第27行 (occurs函数里) 应该是or，不应该是and

^ | ▾ • 回复 • 分享 ▾



Xavier Wang (starwing) • 2年前

我试了一下这个程序，发现一个反例：test (f[a,a]) (f[f[a,f[b,c]],f[y,y]]), 结果是：

y -> f(b,c)

a -> f(b,c)

a -> f(f(b,c),f(b,c))

有两个a，而且无法消去。是不是compose的时候得判断a是否已经存在且不能unify？不然会导出矛盾。

^ | ▾ • 回复 • 分享 ▾

转载请注明出处，收藏或分享这篇文章到：

