GHC 源码剖析：类型检查

陈逸凡，吴昊泽

2017 年 5 月 30 日

1 概述

Haskell 是一门静态类型、强定型、多态的纯函数式语言，其设计精妙的类型系统也是使 Haskell 独具特色的重要原因。在 GHC 编译的过程中，类型检查也是编译器前端的重要一环。类型检查一方面可以在编译期就发现由于类型不匹配造成的错误，另一方面也可以确定一些多态函数的类型，以减小多态造成的开销。

2 类型论简介

2.1 类型和类型系统

类型，可以认为是满足一定性质的“值”的集合。在计算时对表达式的参数和结果的类型做约定，可以保证计算保持程序员所期望的性质。

为了能够形式化地描述代码与类型相关的性质和行为，许多语言都会定义自己的类型系统。完备的类型系统应当可以做到可计算的代码一定是类型上匹配的，而类型匹配的代码也可以执行计算。这样可以保证程序语义与程序员设计时的逻辑的一致性。而一个表达能力强大的类型系统，还应当支持更加丰富的类型定义，进而支持多态等类型特性，这样能在要求程序员写出满足类型要求的代码的同时，赋予程序员更高的编写代码的灵活性。

2.2 类型系统与类型检查的算法

学术界对于函数式语言的类型系统已经有非常深入的研究，对各种类型系统做类型推导也有非常成熟的算法。

函数式语言以 lambda 演算为理论基础。可以将 lambda 表达式视作描述将一个个子表达式所表示的函数组装成更复杂的函数的过程。对于带类型的 lambda 表达式，其每一个句子（term），都会伴随对于作为参数的各个子句以及该句子本身的类型约束，于是将作用在每一个子句上的类型约束集合起来，就能得到一个约束方程组。对于这个约束方程组用下面的合一算法求解，就能得到整个程序以及其中每个子句的类型信息。这就是类型重建方法的基本思路。[3][4]。

```
unify(ta,tb):
  ta = find(ta)
  tb = find(tb)
  if both ta,tb are terms of the form D p1..pn with identical D,n then
    unify(ta[i],tb[i]) for each corresponding ith parameter
  else
  if at least one of ta,tb is a type variable then
    union(ta,tb)
  else
    error 'types do not match'
```

对于带一阶参数多态的 lambda 演算系统也就是 Hindley-Milner 系统[1]，这一算法几乎可以在线性时间内得到程序的类型信息。但是，对于更复杂的系统，如 Girard-Reynolds 系统即 System F [2]，普通的类型重建方法在不带类型提示的时候被证明是不可判定的 [9]。这就需要对类型系统加一些限制了。

3 GHC-Haskell 中的类型系统

接下来需要介绍一下 GHC 中定义的类型系统。

3.1 Haskell 语言的特点

Haskell 2010 标准中 [1] 定义了一个略强于 HM 系统的类型系统，可以认为是“使用范围较广的语言”范围内最为强大的。

[1]一阶多态只允许类型参数的量词出现在函数类型的顶层，也就是说，不能以多态函数为返回值

[2]system F 也被称为二阶 lambda 演算，允许类型参数的量词出现在任意位置

一方面，Haskell 是强静态类型的语言，运算过程中值的类型不能改变，参数的类型必须与函数定义时约定的类型相一致，不允许隐式类型转换，这对代码提出了较强的限制。

但另一方面，Haskell 又提供了很大的灵活性，除了由编译器负责实现的预定义的标准类型之外，还允许程序员在代码中不依赖预定义内容地自定义各种类型，类型构造器允许接受类型参数，所以实现了不弱于 C++ 模板类的表达能力的自定义类型。在多态范式的实现上，Haskell 既通过类型参数实现了参数多态（parametric polymorphism），也可以通过类型类[3]实现了特定多态（ad-hoc polymorphism）。此外相当重要的一点在于，函数的类型签名不是必需的，大部分类型信息可以由编译器通过类型推导自动得出。

而 GHC 还对 Haskell 语法进行了扩展，这样就导致一个更加丰富的类型系统。

3.2 GHC 内部的类型系统

GHC 在类型系统方面的扩展 [6] 主要有

- RankNTypes 实现了更接近 system F 的更高阶的参数多态。[4]
- Generalised Algebraic Data Type (GADTs) 允许值构造器（也是一种函数）返回更丰富的类型，从一个角度实现了 System F_ω 的一部分功能。
- Type families 允许对具有一定性质的类型构造器做分类，也实现了 System F_ω 的一个非常小的子集。
- Equality constraints 可以约束两个类型变量具有相同的类型。
- Coercion 强制类型转换，用于将一些冗余的值构造过程消除。

这些扩展之后，原本的带约束的 HM 系统已经不足以表达 GHC Haskell 的类型系统了。所以 GHC 使用了一个称之为 System FC 的类型系统 [5][2]，囊括了上面这些扩展。限于篇幅，不展开对于该类型系统的证明，只给出结

[3]类型类的作用可与 Java 中的 Interface 类比

[4]需要注意的是，实际程序中往往最多需要二阶参数多态，而更高阶参数多态是不可判定的。

论，这一类型系统是“几乎”完备的，只对开启扩展后的少数代码，有可能会导致通过类型检查的代码在运行时会发生类型错误。

3.3 GHC 中类型检查的流程

前面一篇提到，GHC 中将重命名和类型检查两个流程结合在一起执行。所有的类型检查重命名函数都在 ghc/compiler/typechecker/TcRnDriver.hs 文件中。在编译器流水线中，最外层的入口为 tcRnModule 函数，这一函数接受解析树为参数后，先处理隐式导入的 Prelude 模块[5]和 import 显式导入的模块[6]，将其导出的名字和类型都添加到环境中，并重命名所有出现的导入的名字，然后再以本文件中所有的顶层绑定为入口，对语法树进行自顶向下的扫描（其间也包括之前提到的对包含中缀运算符的表达式的解析树进行重构等操作），将生成的类型约束全部添加到环境中，之后进行一次约束求解，生成所有当前模块所有函数的类型信息。在这之后会再根据模块导出信息检查是否有未用到的绑定，以及 main 函数是否已经被导出等内容。这样对一个模块文件的检查和重命名就基本完成了。

不同于一般的 HM(X) 约束求解方法，Haskell 采用了一种称为 OutsideIn(X) 的约束求解方法 [8]，基本思路就是逐模块进行类型重建，对每个模块，先对其导入的模块进行类型重建，得到其求解后的类型信息，按照引入方式添加全部或部分导出名字的类型信息到当前模块的环境中，再对当前模块生成类型约束后，再进行一次约束求解，得到当前模块的类型信息。这样模块化地进行类型重建，能够提高类型检查的效率，而且能适应 GHC 对类型系统的扩展。

3.4 GHC 对标准类型的实现

Haskell 标准类型是机器无关的一套通用标准，而 GHC 需要在本地上实现对这些标准类型的支持，就需要用与机器相关的方式来实现这些类型。

在实现上，GHC 实现了十多种原始类型（primitive types），包括 32 位整数、64 位整数、单精度浮点数、双精度浮点数、布尔值、字符等等。这些是在机器相关的 prim 库 [7] 中定义的，再将这些原始类型用类型标签来封

[5]包含所有 Haskell 程序所必需的基础定义

[6]Haskell 要求模块间导入关系不能成环

装，就得到了标准类型。但在 Haskell 本身的语法中，是无法表达这样的原始类型的，所以 GHC 使用 C 语言代码表示原始类型的值构造和运算函数，再用外部函数接口[7]引入的。

而在类型层面，也需要对这些原始类型的性质做一定的形式化。在 GHC 扩展语法中，这些原始类型被放到一个名为'#' 的 kind[8] 中，这些类型的值被称为 unboxed values （未装箱的值）；而普通的 Haskell 语法内的类型则都是'*' kind 的成员，称为 boxed value。这样能对原始类型的式子与标准类型的式子有效隔离，避免函数混用导致恶劣的后果。也有利于对程序性质进行分析验证。另一方面，开放 unboxed value 的接口也为特殊目的的程序做性能优化提供了途径，比如引入 SIMD 向量化指令等等（尽管并不推荐这么做）。

参考文献

[1] The Haskell community. Haskell 2010 language report.

[2] R. A. Eisenberg. System fc, as implemented in ghc. University of Pennsylvania Department of Computer and Information Science Technical Report, MS-CIS-15-09, 2015.

[3] R. Hindley. The principal type-scheme of an object in combinatory logic. Transactions of the American Mathematical Society, 146:29–60, 1969.

[4] Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 17(3):348 – 375, 1978.

[5] Martin Sulzmann, Manuel M. T. Chakravarty, Simon Peyton Jones, and Kevin Donnelly. System f with type equality coercions. In Proceedings of the 2007 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, TLDI '07, pages 53–66, New York, NY, USA, 2007. ACM.

[6] The GHC team. Ghc language features.

[7] FFI，使 Haskell 可以与其他语言在约定调用接口的情况下调用其他语言的代码。

[8] 一个 kind 是具有一定性质的类型的集合

[7] The GHC team. The ghc-prim package.

[8] Dimitrios Vytiniotis, Simon Peyton Jones, Tom Schrijvers, and Martin Sulzmann. Outsidein(x): Modular type inference with local assumptions. Journal of Functional Programming, 21:333–412, September 2011.

[9] J.B. Wells. Typability and type checking in system f are equivalent and undecidable. Annals of Pure and Applied Logic, 98(1):111 – 156, 1999.