编译 ghc boot libraries

Felis sap...

函数式编程、编程语言、编程 话题的优秀回答者

已关注

之前提过我在开一个基于 GHC 前端的 Haskell 编译器坑，目前的做法就是通过 GHC API，调用 ghc --make 进行编译，同时用 Hooks 接口捞到中间表示，接到自定义的 codegen 上。
不过，单独拿到 home modules 的中间表示显然并不能办到什么，里面有许多 identifier 是全局 symbol，也就是说，来自 ghc-prim、base 等 boot libraries 的定义。要做一个新编译器，需要将依赖的所有 symbol 对应的中间表示都拿到，然而这些 boot libraries 是在 GHC 自身编译时就完成编译和安装的，GHC 安装好之后只剩一些 interface file 和链接好的库，没有中间表示可以用……

一周前我搭好获取中间表示的框架以来一直在苦恼这个问题。选择支大致有这么几个：

- 完全不支持标准库。有的编译器确实是这么做的，比如将 Haskell 合成到 VHDL/Verilog 的 CλaSH，这货提供了一个自定义的 Prelude，所有用户代码只能基于这个 Prelude 来开发。考虑到 CλaSH 自身对支持的 Haskell 特性有极严苛的限制（比如禁止尺寸未知的递归数据类型），大多数 Haskell 标准库里的东西的确派不上用场。这个选择无疑是最省事的。
- 假装支持标准库。通过 GHC API 编译的时候，仍然如常载入 boot libraries，并且可以 import 其中的 modules。然后在链接时，对应 boot libraries 中定义的全局 symbol，包括各种 data constructor、函数等等，使用手动实现的版本。如果编译的代码比较简单，只用到 Prelude 里面常见的一些东西（比如 list、monad、IO 等等），需要手动写的东西也不多——遇到没有手写过实现的 symbol 报错就行了。
- 实现自定义的 boot 程序，像 haste 和 ghcjs 就是这么做的。在安装我们的编译器时，boot 程序会获取 ghc boot libraries 的源代码（很有可能是魔改版），然后将其编译并获取中间表示，最后将编译后的版本注册到专门的 package database 中，之后在编译用到标准库的 module 时就有完整的标准库中间表示可以拿来链接，或者做一些全程序优化的 pass。

自己的 boot libraries 了。方法出乎意料地简单：参考 boot.sh 脚本，首先需要从 ghc repo 中复制出头文件和 boot libraries 的代码，然后新建一个空白的 package database，按照 rts -> ghc-prim -> integer-gmp -> base 的顺序，对每个 boot libraries 先编译 Setup.hs 脚本，然后用于 configure && build && install，在 configure 阶段指定我们自己的 package database 即可，刚刚在自己的开发环境里试验成功。

与普通库相比，boot libraries 的编译多少还是有一些 magic 在里面的：

- 有一些头文件是 ghc 自身编译时生成的（比如 platform constants），ghc repo 里的 includes 不够用，需要先跑一趟 ghc 编译然后复制出来用
- 关于 rts：rts 代表 Haskell 的 runtime，里面没有半点 Haskell 代码，全部都是 C/Cmm 代码，用到了 posix/win32 api，也用到了 libffi。rts 的编译流程实在太 magic 了，我怎么都搞不定，考虑到 boot 的目的是获取 Haskell 代码编译后的中间表示，所以最后索性直接把 ghc 安装目录里的 rts 复制过来，把 package config 里的路径改一下然后 ghc-pkg register 就直接用作之后 booting 的依赖库了（虽然我们自己不会去用）
- 关于 ghc-prim：ghc-prim 的编译流程也比较 magic，有2个隐藏的模块：GHC.Prim 和 GHC.PrimopWrappers 没有对应的 Haskell 源代码，而是通过 ghc repo 中的 genprimop 程序从一个描述有哪些 primop 可用的文本文件中生成源代码，具体可以参考 ghc-prim 的 Setup.hs 脚本。veggies 的作者手动重新实现了类似 genprimop 的程序，导出 LLVM 支持的 primop，我直接把 GHC 编译之后剩余的 GHC/Prim.hs 和 GHC/PrimopWrappers.hs 复制到 ghc-prim 的目录里用。
- base 一个很重要的编译选项是选择 Integer 的实现。ghc 提供了 integer-gmp 和 integer-simple 两个选择，integer-simple 中 Integer 通过一系列 unboxed int 的链表表示，而 integer-gmp 则基于 libgmp 提供 Integer 实现（内存管理则由 Haskell 侧提供）。经测试，单独实现自定义 boot 时，只有 integer-gmp 可以使用，integer-simple 会有奇怪的编译错误。

下一步的工作就是把试验场里一堆 bash 脚本翻译回 Haskell，把 boot 程序搞出来。使用 ghc frontend plugin 机制，我们可以免于实现自己的 ghc wrapper 的麻烦，不用处理 ghc 的命令行参数解析也可以用之前实现的 api wrapper 走自定义的编译流程。当然，还有一些附加工作：

- 实现魔改的 boot libraries。最关键的问题就是 C 依赖，目前初步打算把所有 C 依赖干掉，换成手写的实现
- 实现自定义的 object file 格式和 linker。考虑到可能做全程序优化的 pass，所以不能直接用 WebAssembly 的 binary code 格式，而要序列化所有的中间表示。这个相对比较好做。

距离第一个 hello world 能跑起来已经胜利在望了——虽然可以预计工程量非常可观。接下来有一大堆技术挑战：

- runtime 的实现。完全基于 WebAssembly 来做的话，几个比较坑的点：没有任意跳转（可以用 binaryen 的 relooper 解决）；不能显式操作栈帧，没有尾递归（可以自己维护栈，把所有函数调用处理成跳转；等 WebAssembly 日后支持）；没有函数指针（用函数入口的 basic

WebAssembly 来用 JavaScript 对象搞 STG heap object 的话——不就退回 haste/ghcjs 的老路了么……

- Concurrency/Parallelism 的实现。最简单的是移植原来的单线程 runtime，不过既然有 WebWorker 可以搞多线程，有 SharedArrayBuffer 和 Atomics 可以做有限度的线程间同步，等单线程 runtime 做出来了，可以考虑试试做多线程版本。
- posix API 的 polyfill。理想的情况下，针对不同的 target （Browser/Node.js），能自动屏蔽不可用的接口，给出编译期错误，比如用到 readFile 的话，在 Browser backend 上会报错。而 haste/ghcjs 原来的处理方法是把 dummy 实现塞进去，然后运行时报错，这是不好的。Emscripten 倒是提供了虚拟的文件系统和 stdin/stdout，而 Browsix 更狠，连虚拟的 socket 和 process 都有，不过我不大确定用 Emscripten 和 Browsix 提高 native 兼容性是否值得……
- Template Haskell 的实现。估计很多细节得跟 Luite Stegeman 请教一番……
- 支持 npm/webpack 之类 JavaScript 侧的包管理工具。这个对用户体验是非常重要的，不过 haste 和 ghcjs 压根就没考虑（

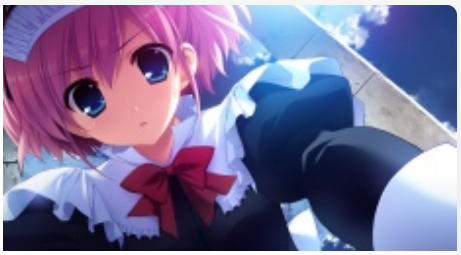
chit-chat 时间：

- 为什么要开始做编译器，而不是继续研究 EDSL 的各种奇技淫巧？因为 Any sufficiently complicated EDSL contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Haskell. （这话是我说的）
- 为什么不从头设计一个通用编程语言？ref
- 为什么不 fork 一下 haste 或者 ghcjs，而要另起炉灶？因为项目的目的中，除了“让 Haskell 在浏览器中跑起来、跑快点”这个悲愿以外，还有提供一个灵活的框架，让后来的 Haskell compiler writer 都能尽量少跟 GHC API 打交道、将注意力放在优化算法本身，不论该 compiler 的 target 是何平台。而我自己试验多种 codegen 方案也会方便不少。

发布于 2017-04-25

Haskell GHC（编程套件） 函数式编程

推荐阅读



GHC API 系列笔记（1）：入门篇

elis... 发表于不动点高校...

BuckleScript和其它编译器的比较

前几天国外的一个开发人员写的一篇文章， 在众多只关注特性而很少关注编译器质量的文章中 实属一股清流 特意把它转过来 可以看得出来BuckleScript不只领先 而且领先的不是一个身位 原链接：...

张宏波 发表于Buckl...

Dotty —— Scala的下一代编译器

Glavo

最快

昨天 Buck 我把 经在 昨天 mes

张宏

5 条评论

切换为时间排序

写下你的评论...

Felis sapiens (作者) 回复 祖与占 1 年前

stack safety for free 主要针对 free monad 实现的问题，在这里用不上。。不过前一篇还是挺不错

1 查看对话

以上为精选评论

祖与占 1 年前

> 不过既然有 WebWorker 可以搞多线程 最近发现的一篇 is.muni.cz/th/374321/fi... Concurrency support for PureScript > 不能显式操作栈帧，没有尾递归（可以自己维护栈，把所有函数调用处理成跳转；等 WebAssembly 日后支持） 用PS Stack Safety for Free 的技术可不可以? > 还有提供一个灵活的框架，让后来的 Haskell compiler writer 都能尽量少跟 GHC API 打交道 赞美司机

赞

祖与占 1 年前

orz 评论格式没了

赞

学习了！

赞

罗宸 1 年前

这不是Paul说lisp的那段么……

赞