AOSCC 2025 | 欢迎

重铸 PL 荣光

Haskell 工具链生态与未来展望

comonad

上海交通大学 Linux 用户组 (SJTUG)

我是谁?

- 上海交通大学 Linux 用户组 (SJTUG) 成员
  - 参与了并不存在的吃吃喝喝
  - 主讲了真实存在的技术分享 但讲得很烂
    - 试图复兴 SJTUG 技术分享传统
- 其他身份:
  - 编程语言理论 (PL) 初心者 写证明的
  - NixOS 重度用户 打包的

内容提要

- Haskell 语言设计、工具链
- GHC 实现简介h
- GHC 架构适配与发行版支持现状
- Haskell 生态的主要痛点
- 社区近期工作与未来展望

什么是 Haskell？

Haskell is a safe, purely functional programming language with a fast, concurrent runtime. —— Haskell Weekly

前身：Miranda，早期惰性求值函数式语言

- 「强大的」类型系统：H-M 类型推导 + System FC 类型演算（伏笔）
  - 基于或由 Haskell/GHC 启发的类型论前沿探索，例如：
    - Idris2：依值类型通用编程
    - Agda：定理证明器
    - Liquid Haskell：精化类型扩展
    - ...

什么是 Haskell？

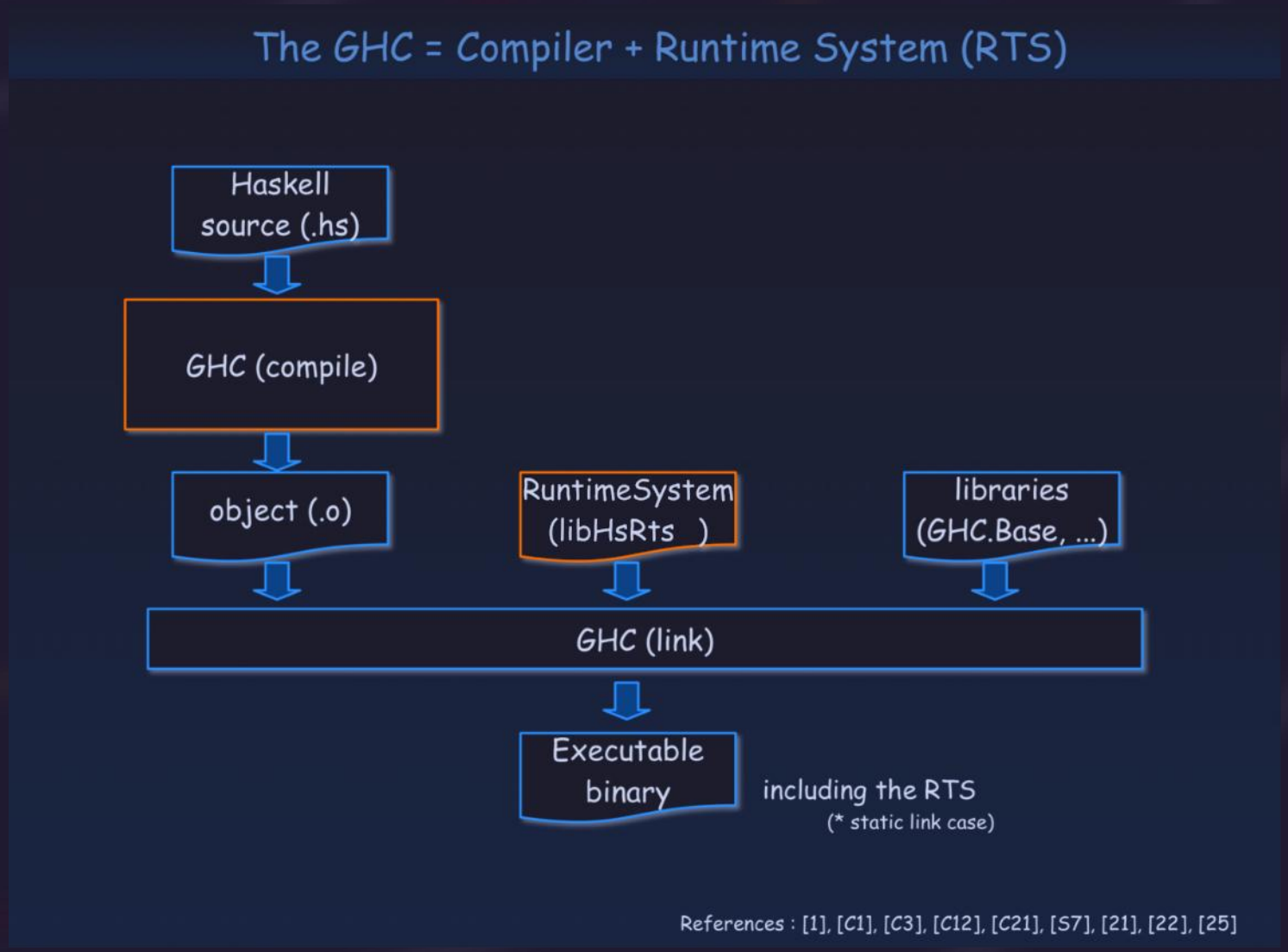
- 「先进的」语言特性/扩展：
  - 命令式风格抽象：Monad with do-notation
  - 内建并发支持：
    - STM (软件事务内存)：无锁并发抽象
    - MVar/TVar：线程间通信、并发原语等
- 「稳固的」系统基础软件生态：
  - Pandoc、Shellcheck、Dhall、Darcs、...

Haskell 编译工具链

Glasgow Haskell Compiler (GHC) 是目前 Haskell 事实标准的编译器实现。

- GHC = GHC Compiler + GHC Runtime System
  - Runtime System (RTS)：管理与操作系统直接交互的部分
    - 内存分配、垃圾回收、线程管理、FFI、...

Haskell 编译工具链



Haskell 编译工具链

- Hackage：Haskell 生态最主要的软件源
  - 总量高达 18000+

    nix eval --impure --expr "with import <nixpkgs> {}; builtins.length (builtins.attrNames haskellPackages)"
  - 但其中大部分缺乏维护已无法构建
    - 粗略统计 Hackage 在 GHC 9.8 可用的软件包数量约 8000
- Stackage：源自 Hackage 的稳定 Haskell 包集
  - 总量 3000+

Haskell 编译工具链

- Cabal: 由 Haskell Community 维护的 Haskell 包管理器/构建系统
  - 支持声明依赖版本约束，通过 Hackage index-state 锁定依赖版本 (cabal.project, cabal.project.freeze)
  - 支持多软件包元数据/依赖描述 (<package>.cabal)

Haskell 编译工具链

- Stack: 由 CommercialHaskell 维护的 Haskell 包管理器/构建系统
  - 支持声明依赖版本约束，通过 Stackage 快照锁定 GHC 及依赖版本 (stack.yaml)
  - 支持多版本依赖描述 (stack-*.yaml, stack-*.yaml.lock)
  - 支持软件包元数据描述 (package.yaml)

Haskell 编译工具链

- Nix: 由 Nix Community 维护的通用包管理器/构建系统
  - 全量同步 Hackage
  - 依赖 Nix/Nixpkgs 控制版本
  - 兼容 Stack、Cabal 构建系统 (cabal2nix、haskell-flake、...)

GHC

- Glasgow Haskell Compiler (GHC) 是目前唯一活跃维护的，事实标准的 Haskell 编译器实现

  ~~Haskell a.k.a GHC Language~~

GHC 实现简介

编译阶段: Haskell Language -> Core -> STG -> Cmm -> Backend

- Core
  - 基于 **System F with Type Equality Coercions** (a.k.a. **System FC**)
    以支持 GADTs、Type families 等语言特性
    - 为支持高阶类型，实际的类型推导不总是保持完备性（伏笔回收）
  - 由 Haskell 代码进行去糖化、重命名等操作得到

GHC 实现简介

编译阶段: Haskell Language -> Core -> STG -> Cmm -> Backend

- STG (Spineless Tagless G-machine)
  - 惰性求值的抽象状态机模型
  - 描述堆分配、垃圾收集的抽象模型

GHC 实现简介

编译阶段: Haskell Language -> Core -> STG -> Cmm -> Backend

- Cmm (C minus minus)
  - 平台无关、函数式风格的中间表示
  - 提供类汇编的基本指令
  - 注意: 与 C / C++ 无关

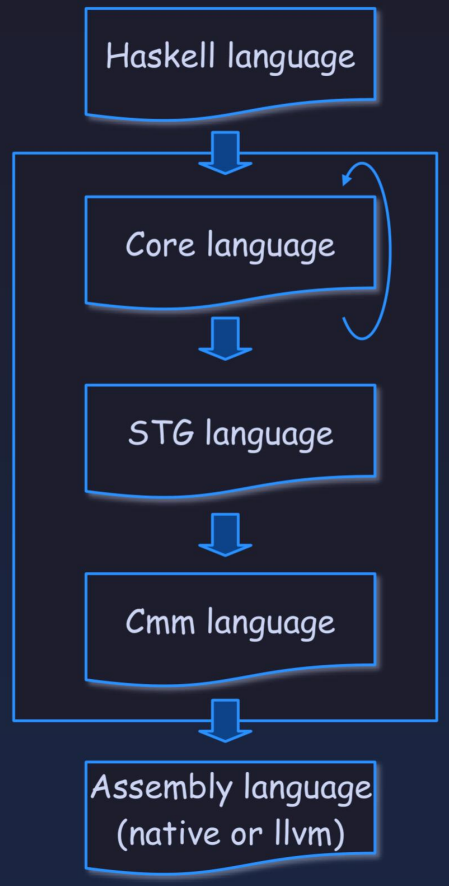
GHC 实现简介

编译阶段: Haskell Language -> Core -> STG -> Cmm -> Backend

- Backend
  - 原生平台：NCG、LLVM、C
    - 原生代码生成器 (Native Code Generator, NCG) 直接由 Haskell 编写，无需 C 编译器即可生成目标平台代码
    - 目前 NCG 后端性能/可维护性均为最优
  - Web 平台：JavaScript、WebAssembly (WASM)
    - 源于 GHCJS，提供 JavaScript 互操作性

GHC 实现简介

GHC transitions between five representations



GHC compile steps

each intermediate code can be dumped by :

```
$ ghc -ddump-parsed
$ ghc -ddump-rn
```

```
$ ghc -ddump-ds
$ ghc -ddump-simpl
$ ghc -ddump-prep
```

```
$ ghc -ddump-stg
```

```
$ ghc -ddump-cmm
$ ghc -ddump-opt-cmm
```

```
$ ghc -ddump-llvm
$ ghc -ddump-asm
```

References : [1], [C3], [C4], [9], [C5], [C6], [C7], [C8], [S7], [S8], [21], [22], [25]

GHC 构建流程与架构支持

GHC 构建系统演进：Makefile-based (弃用) -> Hadrian -> ghc-toolchain (开发中)

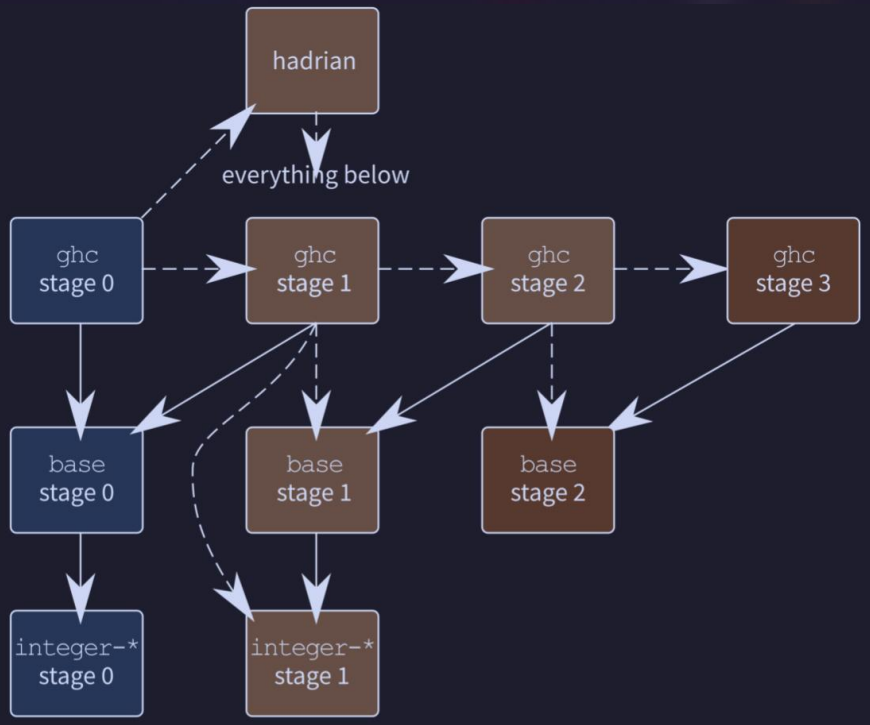
- Hadrian：基于 Shake 开发的现代构建系统
  - Shake 支持自动检测依赖，增量构建等现代特性
  - Shake 完全由 Haskell 编写，可提高 Haskell 自举比例

Hadrian *基本实现了* GNU Makefile 的构建功能（伏笔）

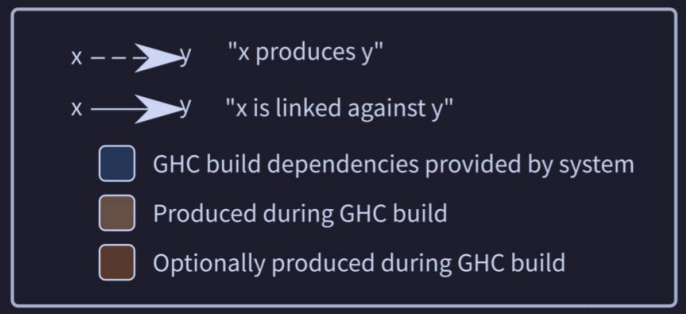
GHC 构建流程与架构支持

GHC 构建系统演进：Makefile-based (弃用) -> Hadrian -> ghc-toolchain (开发中)

- 构建前准备：boot + configure
  - boot (Python script)：检测依赖项、仓库子模块状态
  - configure (GNU autoconf)：根据平台、依赖状态等生成构建配置
- 构建：stageBoot -> stage0 -> stage1 -> stage2
  - 非常复杂，看图罢



Hadrian path	/usr/...	_build/stage0	_build/stage1	_build/stage2
ghc runs on	build	build	target	target
produces code for	build	target	target	target



GHC 构建流程与架构支持

GHC 构建系统演进：Makefile-based (弃用) -> Hadrian -> ghc-toolchain (开发中)

- ghc-toolchain：下一代 Haskell 工具链管理工具
  - 专注于工具链管理和配置，尤其是交叉编译等复杂场景

    `ghc-toolchain --host=<triple> --target=<triple>`
  - GHC 构建流程中将作为 Hadrian 构建系统的补充支持
    - `./configure --enable-ghc-toolchain` 生成目标平台配置 (config.target)

GHC 构建流程与架构支持

- GHC 架构支持（部分统计）：
  - T1:
    - x86_64-windows
    - x86_64-linux, aarch64-linux
    - aarch64-darwin
  - T2:
    - x86_64-freebsd, x86_64-openbsd
    - powerpc-linux, powerpc64-linux, powerpc64le-linux
    - s390-linux

GHC 构建流程与架构支持

- GHC 架构支持（部分统计）：
  - T3:
    - aarch64-windows
    - **riscv64-linux**
    - **loongarch64-linux**

- 充分体现了架构之间的阶级落差！

GHC 构建流程与架构支持

- RV64
  - LLVM 后端：GHC 9.2 起
    - 然而两年后才默认开启 GHCi 构建支持
  - NCG 后端：GHC 9.12 起
    - OpenSUSE Tumblewood 维护了 GHC 9.10 的移植补丁
    - Dwarf 调试支持已完成，等待测试 (GHC MR#13760)
    - RVV 1.0 支持适配中 (GHC issues#25331)

GHC 构建流程与架构支持

- LA64
  - LLVM 后端：GHC 9.6 起
  - NCG 后端：GHC 9.12 起
    - atomicRMW, Cmpxchg, Xchg 支持等待合入 (GHC MR#14528)
    - SIMD 支持适配中

发行版支持

发行版	GHC 版本选择	库支持规模	维护策略	RV64/LA64 支持
Debian	9.0, 9.6	3000+	手动维护 patchset，检查可用性	RV64 支持，LA64 支持 (Debian Haskell Team)
Fedora	9.6, 9.8	800+	.spec 手动打包，检查可用性	RV64 支持，LA64 支持 (Fedora Haskell SIG)
Arch Linux	9.4	1000+	PKGBUILD 手动打包，检查可用性	RV64 社区支持 (archriscv)，LA64 暂无 (loongarch-lcpu)
NixOS	9.6, 9.8	~8000/18000+	cabal2nix + Hydra CI 自动构建 Hackage	RV64 无，LA64 无

- 我们的语言正在蒸蒸日上……吗？

理想很美好，但是...

- 包管理/构建系统
  - “Cabal 地狱”
    - 构建可用性差，无成熟的依赖求解工具
      - cabal v2-freeze 支持锁定 Hackage index-state，但不支持锁定 GHC 版本
    - 命令文档混乱，例如：
      - Q：`cabal v1-build/build/v2-build/new-build` 有什么区别?
        A：cabal-install >= 3.0 起，234 均指代新版 Nix-style 沙盒构建；
        1 指代旧版构建，不推荐继续使用
      - Q：文档所示命令 `cabal build --with-prog=PATH` 支持指定哪些工具?
        A：参考 `cabal v1-build --help | tail -n 13 | head -n 4`，新版文档移除了详细描述

理想很美好，但是...

- 包管理/构建系统
  - Stack
    - + Stackage LTS/Nightly 快照能够保证可构建性
    - - 快照限制在特定 GHC 版本以及可用依赖范围下使用
  - Nix
    - ++ Nix flakes 有更强的锁定机制 (flake.lock)，且 Nixpkgs 全量同步 Hackage，既保证较大程度的可构建性，又提供了更宽松的依赖选择空间
    - + cabal2nix, haskell-flake 等 Nix 项目能够集成到现有 Haskell 工具链中
    - -- Nix 生态严重缺乏文档指引，学习成本较高

理想很美好，但是...

- 编译器实现
  - GHC 垄断编译工具链
    - GHC 组件、扩展与 Haskell 工具链强耦合
      - 例如 Hadrian <-> Cabal，Template Haskell (TH) <-> GHCi runtime
    - 目前除了 GHC 可用的 Haskell 实现几乎仅有 MicroHs
      - MicroHs 实现了 Haskell2010 的较大子集 + 部分 GHC 扩展，但仍与 GHC 有明显差距
      - Hugs, nhc98, uhc, jhc 等编译器已停止维护多年，并且仅能支持到 Haskell2010

理想很美好，但是...

- 编译器实现
  - GHC 标准推动缓慢
    - Haskell2010 - GHC2021 之间长期没有更新成文标准
    - GHC2024 标准仍未稳定
      - GADTs，Template Haskell 等扩展在不同版本下行为可能不一致
    - Haskell 项目普遍依赖 Haskell2010 甚至 GHC2021 之外的 GHC 扩展，其他编译器适配压力巨大

理想很美好，但是...

- 编译器实现
  - Hadrian 仅在满足 **build = host /= target** 约束的平台三元组进行交叉构建（伏笔回收）
    - GHC 9.4+ 弃用 Makefile 构建方式后，Hadrian 交叉构建 stage2 产物仍为 stage1 的交叉编译器，无法生成目标平台的原生编译器
    - 这也是目前 RISC-V 和 LoongArch 在各发行版上通过交叉构建自举的主要障碍之一
    - Tracking issue：https://github.com/loongson-community/nixpkgs/issues/1

理想很美好，但是...

- 语言设计
  - 字符串抽象泄露
    - Haskell 内置多种字符串实现：String、Data.ByteString、Data.Text

```haskell
type String = [Char]
data ByteString = BS {-# UNPACK #-} !(ForeignPtr Word8) -- payload
                     {-# UNPACK #-} !Int                -- length
                     -- ^ @since 0.11.0.0
data Text = Text
  {-# UNPACK #-} !A.Array -- ^ bytearray encoded as UTF-8
  {-# UNPACK #-} !Int     -- ^ offset in bytes (not in Char!), pointing to a start of UTF-8 sequence
  {-# UNPACK #-} !Int     -- ^ length in bytes (not in Char!), pointing to an end of UTF-8 sequence
  deriving (Typeable)
```

    - 标准库仍然广泛使用性能低下的 String，而非更高性能的 ByteString 或 Text
    - 至少，打开 OverloadedString 扩展可以减少字符串类型转换的痛苦……

理想很美好，但是...

- 语言设计
  - Record 类型默认行为不友好
    - 字段名全局唯一
      - DuplicateRecordFields：解决字段名冲突问题
    - 选择器污染全局命名空间
      - NoFieldSelectors：避免生成顶层选择器函数
    - 字段更新不兼容多态
      - OverloadedRecordDo：允许多态函数更新 Record 字段
    - 此外 GHC 也提供了若干语法糖扩展：
      - OverloadedRecordUpdate：允许在 Record 构造中使用 monad do-notation
      - OverloadedRecordDot：支持通过 `foo.bar` 访问 Record 字段

理想很美好，但是...

- 语言设计
  - Haskell 标准库改进鲜有进展
    - Haskell Discourse 有若干关于标准库性能、使用体验等问题的长帖
    - 2017 年 Haskell Foundation 发布 foundation 库，旨在改进 Haskell 标准库
      - 然而 2023 年发布最后一版后就归档了.....

社区近期工作进展与未来展望

- GHC Cmm 文档重构 (GSoC 2025)
- GHC 9.14 起发布长期支持 (LTS) 版本
  - 提供至少两年非功能性支持
- Haskell Foundations 筹款迁移基础设施
  - ARM CI、备份服务等
- Nixpkgs 树外移植龙架构/RISC-V GHC 工作推进中
- ……

结语

- Make Haskell Great Again!
  - 不只是“学术语言”
- 如何开始贡献 Haskell/GHC 项目?
  - 使用 Haskell 写 Nix 工具，斩获大量 stars
  - 跟进 GHC 发行版架构适配，打包 Haskell 应用，成为知名社区工作者
  - 给 GHC 提交 Merge Request (MR) 并合入核心代码，成为编译器大师
  - 基于 Haskell 生态，研究类型系统~~并投稿 PL 会议，成为学术之星~~
- Avoid $ success at all costs —— Haskell’s motto

提问环节?