How to Interact with a HERMIT

Andy Gill, Andrew Farmer, Neil Sculthorpe, Adam Howell,
Robert F. Blair, Ryan Scott, Patrick G. Flor, and Michael Tabone

Functional Programming Group
Information and Telecommunication Technology Center
University of Kansas

Trends in Functional Programming
Provo, Utah
16th May 2013

What is HERMIT?

What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel

What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.

What is HERMIT?

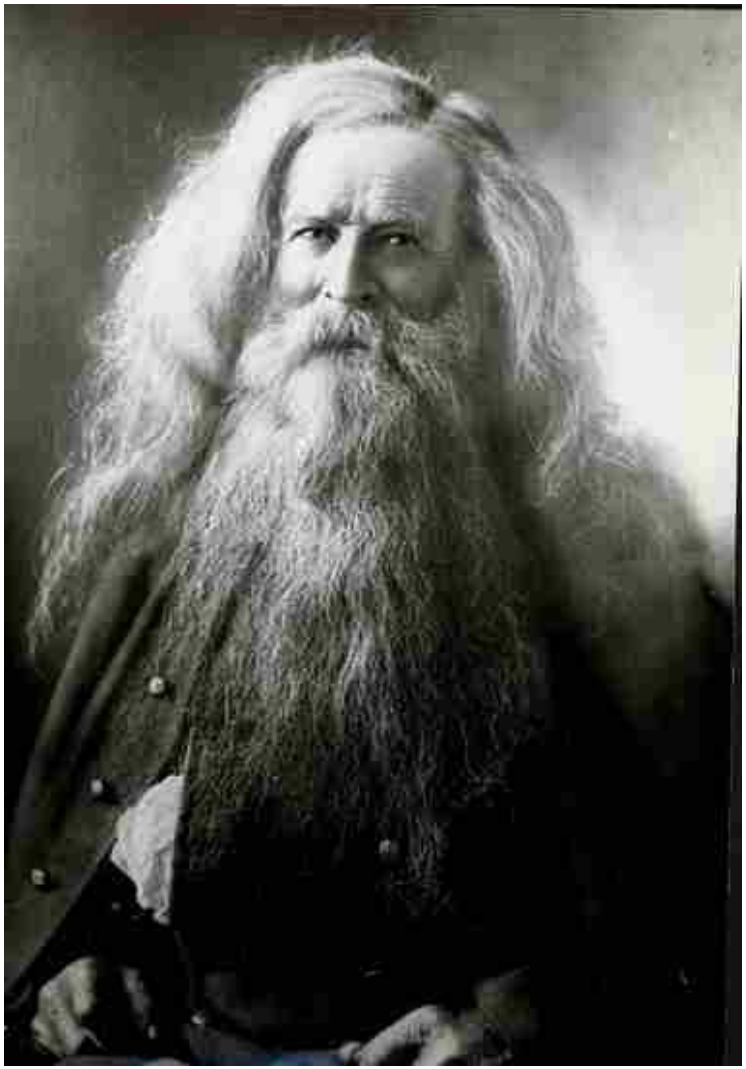
- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.
- Under development at the University of Kansas, Lawrence.

What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.
- Under development at the University of Kansas, Lawrence.
- Not to be confused with:

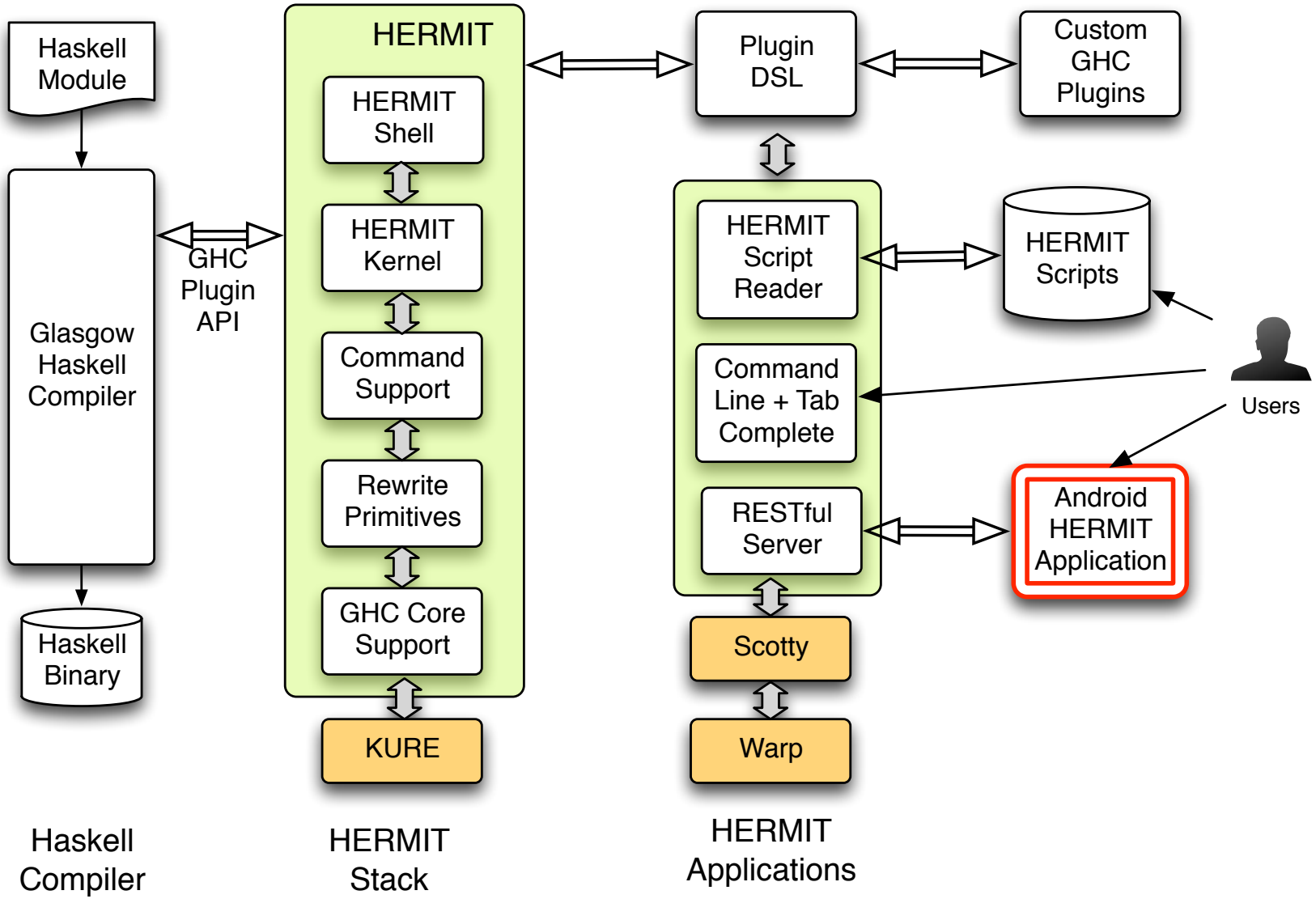
What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.
- Under development at the University of Kansas, Lawrence.
- Not to be confused with: The Kansas Hermit (1826–1909), also from Lawrence.



(image from `http://www.angelfire.com/ks/larrycarter/LC/OldGuardCameron.html`)

The HERMIT Project



Downloading and Running HERMIT

HERMIT requires GHC 7.4 or 7.6 (7.6 recommended)

1. `cabal update`
2. `cabal install hermit`
3. `hermit Main.hs`

The `hermit` command just invokes GHC with some default flags:

```
% hermit Main.hs
ghc Main.hs -fforce-recomp -O2 -dcore-lint
            -fsimple-list-literals -fplugin=HERMIT
            -fplugin-opt=HERMIT:main:Main:
```

GHC Core

```
type CoreProg = [CoreBind]
data CoreBind = NonRec Var CoreExpr
              | Rec [(Var, CoreExpr)]
data CoreExpr = Var Var
              | Lit Literal
              | App CoreExpr CoreExpr
              | Lam Var CoreExpr
              | Let CoreBind CoreExpr
              | Case CoreExpr Var Type [CoreAlt]
              | Cast CoreExpr Coercion
              | Tick CoreTickish CoreExpr
              | Type Type
              | Coercion Coercion
type CoreAlt  = (AltCon, [Var], CoreExpr)
data AltCon   = DataAlt DataCon | LitAlt Literal | DEFAULT
```

Types

```
data Type     = TyVarTy Var
              | AppTy Type Type
              | TyConApp TyCon [KindOrType]
              | FunTy Type Type
              | ForAllTy Var Type
              | LitTy TyLit
data Coercion = Refl Type
              | TyConAppCo TyCon [Coercion]
              | AppCo Coercion Coercion
              | ForAllCo TyVar Coercion
              | CoVarCo CoVar
              | AxiomInstCo CoAxiom [Coercion]
              | UnsafeCo Type Type
              | SymCo Coercion
              | TransCo Coercion Coercion
              | NthCo Int Coercion
              | InstCo Coercion Type
```

Live Demonstration

HERMIT Commands

- Core-specific rewrites, e.g.
  - beta-reduce
  - eta-expand '*x*
  - case-split '*x*
  - inline
- Strategic traversal combinators (from KURE), e.g.
  - any-td *r*
  - repeat *r*
  - innermost *r*
- Navigation, e.g.
  - up, down, left, right, top
  - consider '*foo*
  - 0, 1, 2, ...
- Version control, e.g.
  - log
  - back
  - step
  - save "myscript.hss"

GHC RULES

GHC RULES

- GHC language feature allowing custom optimisations

GHC RULES

- GHC language feature allowing custom optimisations
- e.g.

{-# RULES "map/map" $\forall$ f g xs. *map* f (*map* g xs) = *map* (f $\circ$ g) xs #-}

GHC RULES

- GHC language feature allowing custom optimisations
- e.g.

{-# RULES "map/map" $\forall$ f g xs . *map* f (*map* g xs) = *map* (f $\circ$ g) xs #-}

- HERMIT adds any RULES to its available transformations

GHC RULES

- GHC language feature allowing custom optimisations
- e.g.

{-# RULES "map/map" $\forall$ f g xs . *map* f (*map* g xs) = *map* (f $\circ$ g) xs #-}

- HERMIT adds any RULES to its available transformations
  - allows the HERMIT user to introduce new transformations

GHC RULES

- GHC language feature allowing custom optimisations
- e.g.

{-# RULES "map/map" $\forall$ f g xs. *map* f (*map* g xs) = *map* (f $\circ$ g) xs #-}

- HERMIT adds any RULES to its available transformations
  - allows the HERMIT user to introduce new transformations
  - HERMIT can be used to test/debug RULES

Summary and Publications

- HERMIT is a toolkit for interactive viewing and transformation of GHC Core programs
- Still under development
- Previous publications describing or using HERMIT:
  - The HERMIT in the Machine (Haskell '12) — describes the HERMIT implementation
  - The HERMIT in the Tree (IFL '12) — describes our experiences mechanising simple program transformations
  - Optimizing SYB is Easy! (submitted to ICFP '13) — uses HERMIT to optimise generic traversals
  - KURE (submitted to JFP) — describes the underlying strategic programming language, using examples from the HERMIT implementation