The HERMIT in the Tree

Neil Sculthorpe

(joint work with Andrew Farmer, Andy Gill and Ed Komp)

Functional Programming Group
Information and Telecommunication Technology Center
University of Kansas

Swansea, Wales
12th March 2014

Motivation

- There is often a trade-off between the clarity and efficiency of a program.

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
    - less time-consuming and error prone than pen-and-paper reasoning
    - no need to modify the source file

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
  - less time-consuming and error prone than pen-and-paper reasoning
  - no need to modify the source file
- Our work: transformations in the purely functional language Haskell

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
  - less time-consuming and error prone than pen-and-paper reasoning
  - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
  - e.g. HaRe, HERA, PATH, Ultra
  - but they all operate on Haskell source code (or some variant)

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
    - less time-consuming and error prone than pen-and-paper reasoning
    - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
    - e.g. HaRe, HERA, PATH, Ultra
    - but they all operate on Haskell source code (or some variant)
- Haskell source code?

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
  - less time-consuming and error prone than pen-and-paper reasoning
  - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
  - e.g. HaRe, HERA, PATH, Ultra
  - but they all operate on Haskell source code (or some variant)
- Haskell source code? Haskell 98?

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
    - less time-consuming and error prone than pen-and-paper reasoning
    - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
    - e.g. HaRe, HERA, PATH, Ultra
    - but they all operate on Haskell source code (or some variant)
- Haskell source code? Haskell 98? Haskell 2010?

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
  - less time-consuming and error prone than pen-and-paper reasoning
  - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
  - e.g. HaRe, HERA, PATH, Ultra
  - but they all operate on Haskell source code (or some variant)
- Haskell source code? Haskell 98? Haskell 2010? Glasgow Haskell?

Motivation

- There is often a trade-off between the clarity and efficiency of a program.
- Useful to transform a clear program (specification) into an efficient program (implementation).
- We want to mechanise these transformations:
    - less time-consuming and error prone than pen-and-paper reasoning
    - no need to modify the source file
- Our work: transformations in the purely functional language Haskell
- Several existing transformation systems for Haskell programs
    - e.g. HaRe, HERA, PATH, Ultra
    - but they all operate on Haskell source code (or some variant)
- Haskell source code? Haskell 98? Haskell 2010? Glasgow Haskell?
- Alternative: GHC Core, the Glasgow Haskell Compiler's intermediate language

GHC Core

System F (polymorphic lambda calculus), extended with let-bindings, constructors and first-class proofs of type equality (coercions).

GHC Core

System F (polymorphic lambda calculus), extended with let-bindings, constructors and first-class proofs of type equality (coercions).

```
type Prog = [Bind]

data Bind = NonRec Var Expr
          | Rec [(Var, Expr)]

data Expr = Var Var
          | Lit Literal
          | App Expr Expr
          | Lam Var Expr
          | Let Bind Expr
          | Case Expr [Alt]
          | Cast Expr Coercion
          | Type Type
          | Coercion Coercion

type Alt  = (Constructor, [Var], Expr)
```

What is HERMIT?

What is HERMIT?

- Haskell Equational Reasoning
  Model-to-Implementation Tunnel

What is HERMIT?

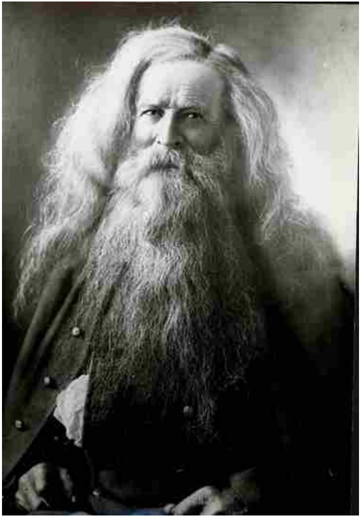
- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.

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.

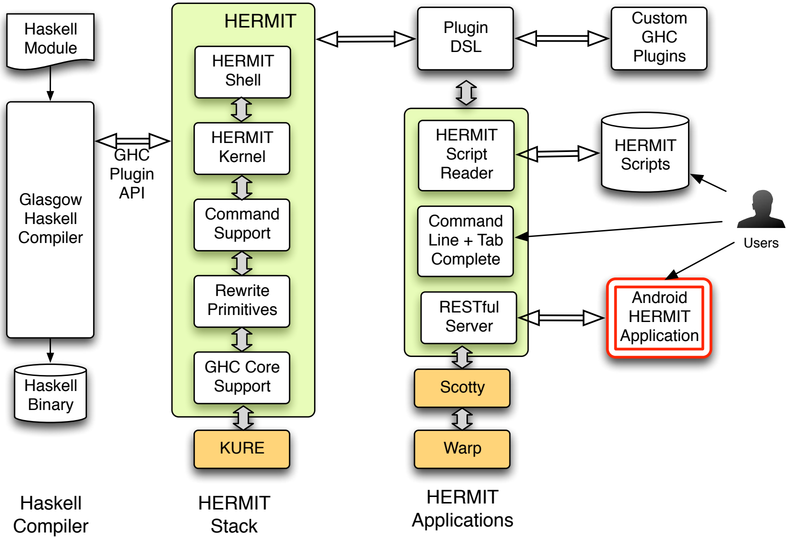
What is HERMIT?

- 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: 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 7.6 or 7.8.

1. `cabal update`
2. `cabal install hermit`
3. `hermit Main.hs +MyModule1 +MyModule2`

The `hermit` command invokes GHC on `Main.hs`, and runs HERMIT on the specified modules.

Demo: Transforming Fibonacci

$$\textbf{data}\ \mathit{Nat} = \mathsf{Zero} \mid \mathsf{Succ}\ \mathit{Nat}$$

$$\begin{aligned}
&\mathit{fib} :: \mathit{Nat} \to \mathit{Nat}\\
&\mathit{fib}\ \mathsf{Zero} &&= \mathsf{Zero}\\
&\mathit{fib}\ (\mathsf{Succ}\ \mathsf{Zero}) &&= \mathsf{Succ}\ \mathsf{Zero}\\
&\mathit{fib}\ (\mathsf{Succ}\ (\mathsf{Succ}\ \mathsf{n})) &&= \mathit{fib}\ (\mathsf{Succ}\ \mathsf{n}) + \mathit{fib}\ \mathsf{n}
\end{aligned}$$

Demo: Transforming Fibonacci

```
data Nat = Zero | Succ Nat

fib :: Nat → Nat
fib Zero                = Zero
fib (Succ Zero)         = Succ Zero
fib (Succ (Succ n)) = fib (Succ n) + fib n


fib :: Nat → Nat
fib n = let work :: Nat → (Nat, Nat)
            work Zero       = (Zero, Succ Zero)
            work (Succ m) = let (x, y) = work m
                            in (y, x + y)
        in
            fst (work n)
```

HERMIT Commands

- Core-specific rewrites, e.g.
  - beta-reduce
  - eta-expand '*x*
  - inline
- Strategy combinators (from KURE), e.g.
  - any-td *r*
  - repeat *r*
  - innermost *r*
- Navigation, e.g.
  - binding-of '*foo*, occurrence-of '*x*
  - lam-body, app-arg, case-alt 2
- Version control, e.g.
  - log
  - back, step
  - save “myscript.hec”
- Presentation, e.g.
  - set-pp-type Show
  - set-pp ghc

Adding Transformations to HERMIT

Two main ways:

- Writing a HERMIT-extension Plugin
  - using KURE on the Core AST
  - full power of Haskell
  - easy to make mistakes
- Using GHC Rules
  - lightweight (can be included in the source code of the object program)
  - type checked by GHC
  - limited by the expressiveness of RULES

GHC RULES

- GHC language feature allowing custom optimisations

GHC RULES

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

```
{-# RULES "map/map" ∀ f g xs . map f (map g xs) = map (f ∘ g) xs #-}
```

GHC RULES

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

```
{-# RULES "map/map" ∀ f g xs . map f (map g xs) = map (f ∘ g) xs #-}
```

- HERMIT adds any RULES to its available transformations

GHC RULES

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

$$\{\text{-}\#\ \text{RULES}\ \texttt{"map/map"}\ \ \forall\, f\ g\ xs\,.\ \mathit{map}\ f\ (\mathit{map}\ g\ xs) = \mathit{map}\ (f \circ g)\ xs\ \#\text{-}\}$$

- 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.

$$\{\text{-}\#\ \text{RULES}\ \texttt{"map/map"}\ \ \forall\, f\ g\ xs\ .\ \mathit{map}\ f\ (\mathit{map}\ g\ xs) = \mathit{map}\ (f \circ g)\ xs\ \#\text{-}\}$$

- 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

- HERMIT is a tool for interactively transforming GHC Core programs
- Currently very experimental
- Ongoing work: support for equational reasoning
- Publications describing HERMIT:
    - *The HERMIT in the Machine* [FGKS12] — HERMIT implementation
    - *The HERMIT in the Tree* [SFG13] — mechanising known transformations

  Publications using HERMIT to prototype new optimisations:
    - *The HERMIT in the Stream* [FHG14] — stream fusion
    - *Optimizing SYB is Easy!* [AFM14] — data-type–generic programming

References

Michael D. Adams, Andrew Farmer, and José Pedro Magalhães.
Optimizing SYB is easy!
In *Workshop on Partial Evaluation and Program Manipulation*, pages 71–82. ACM, 2014.

Andrew Farmer, Andy Gill, Ed Komp, and Neil Sculthorpe.
The HERMIT in the machine: A plugin for the interactive transformation of GHC core language programs.
In *Haskell Symposium*, pages 1–12. ACM, 2012.

Andrew Farmer, Christian Höner zu Siederdissen, and Andy Gill.
The HERMIT in the stream: Fusing Stream Fusion's concatMap.
In *Workshop on Partial Evaluation and Program Manipulation*, pages 97–108. ACM, 2014.

Neil Sculthorpe, Andrew Farmer, and Andy Gill.
The HERMIT in the tree: Mechanizing program transformations in the GHC core language.
In *Implementation and Application of Functional Languages 2012*, pages 86–103. Springer, 2013.