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
neil@ittc.ku.edu

Midwest Verification Day
Lawrence, Kansas
20th September 2012

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

What is HERMIT?

What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel

What is HERMIT?

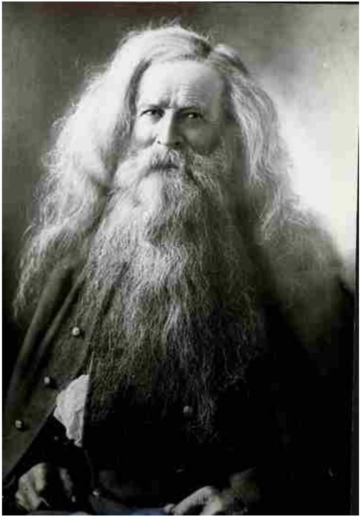
- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.

What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.
- Not to be confused with:

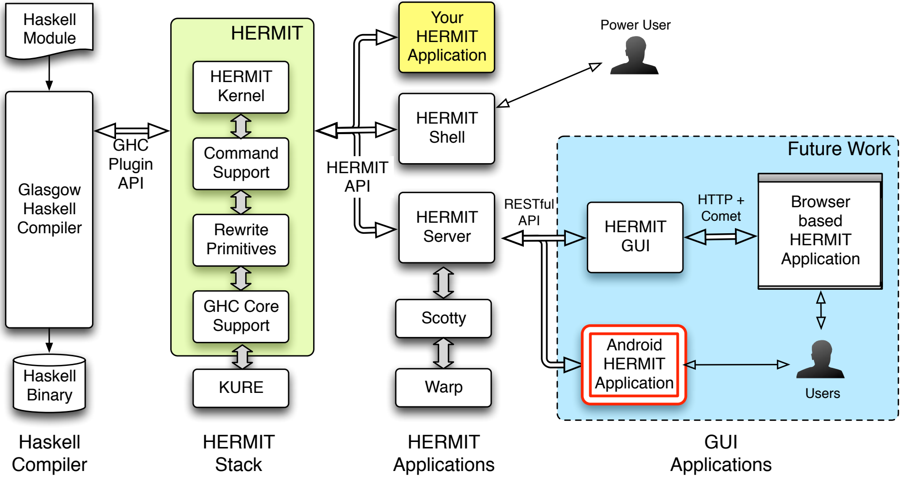
What is HERMIT?

- Haskell Equational Reasoning Model-to-Implementation Tunnel
- A scriptable toolkit for interactive transformation of GHC Core programs.
- Not to be confused with: The Kansas Hermit (1826–1909). Abolitionist, Teacher, Lawrence Founding Father, Brigadier General, Treehouse Dweller, Long-distance Walker and Critic of the Lawrence Elite.



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

The HERMIT Project



Downloading and Running HERMIT

HERMIT requires GHC 7.4 (will be 7.6 compatible very soon).

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

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

```
ghc Main.hs -fforce-recomp -O2 -dcore-lint
            -fsimple-list-literals -fplugin=HERMIT
```

Demonstration: Unrolling Fibonacci

As a first demonstration, let's transform the *fib* function by unrolling the recursive calls once.

```
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
```

Demonstration: Unrolling Fibonacci

As a first demonstration, let's transform the *fib* function by unrolling the recursive calls once.

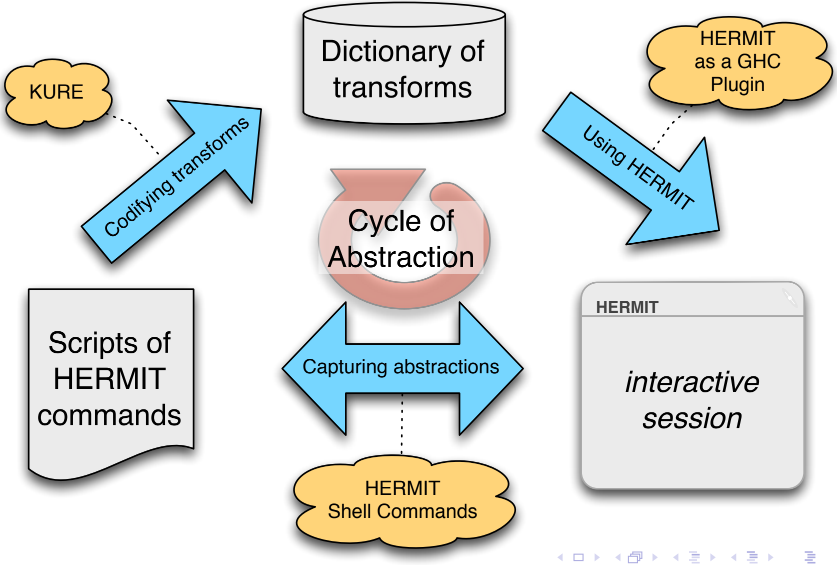
```
data Nat = Zero | Succ Nat

fib :: Nat → Nat
fib Zero              = Zero
fib (Succ Zero)       = Succ Zero
fib (Succ (Succ n)) = (case Succ n of
                           Zero            → Zero
                           Succ Zero       → Succ Zero
                           Succ (Succ m) → fib (Succ m) + fib m)
                         +
                       (case n of
                           Zero            → Zero
                           Succ Zero       → Succ Zero
                           Succ (Succ m) → fib (Succ m) + fib m)
```

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"

Developing Transformations



Tupling Transformation: Fib

```
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)
```

Summary

- HERMIT is a tool for interactive transformation of GHC Core programs
- Still early in development
- Next step: an equational reasoning framework that only allows correctness preserving transformations
- Publications:
    - *The HERMIT in the Machine* (Haskell '12) — describes the HERMIT implementation
    - *The HERMIT in the Tree* (submitted to IFL '12) – describes our experiences mechanising existing program transformations