How to Be Both
Lazy and Reasonable
Tim Chevalier
Scholarship Skills
March 17, 2008
Talk Overview

- You want to reason about your programs.
- Programming in a lazy functional language helps.
- Implementing those languages is hard.
Reasoning About Programs

\[ f \left( \text{someComplicated Function()} \right) \]

\[ f(\text{int } x) \begin{cases} 1 & \text{if today is Monday()} \\ x & \text{else} \end{cases} \]

Can we replace this with 1?
Evaluation Strategies

Strict evaluation
Evaluate function arguments to values
Pass values to functions

Lazy evaluation (call-by-name)
Represent arguments as thunks (delayed computations)
Pass thunks to functions
Roadmap

- Reasoning is good. ✓
- Shall we be lazy or strict?
- Laziness has consequences
  - Reasoning about programs: Easy
  - Reasoning about implementation: hard
  - Can we do better?
The Dilemma

Strict language | Lazy language
Close to hardware | Abstracted from hardware
Good performance | Bad performance
Difficult reasoning | Easy reasoning

Can we have the best of both worlds?
Sort of...
Defending Laziness

Correctness? Ha! Your language is too slow.

Too Slow?! I'll show you!

Don't care about correctness?! I'll show you!

The Practitioner

The Implementors ("Alice", "Bob")
The Trouble With Laziness

\[ f(x) = x^{**} \]

\[ f(4^2 + 4^2) \]

\( (4^2 + 4^2)^{*} (4^2 + 4^2) \)

Repeated computation
A Solution and a New Problem

Call-by-need

\[ f(42 + 42) \]

Sharing

Update

\[ (84 + 84) \]

"Compute 42 + 42"

"Return 84"
In Short...

Hardware is strict! Strict evaluation!
Time Passes

Programming is hard

Lazy evaluation!

Math is easy!
More Problems

Laziness is slow!

Code's still easy

Compiler is hard...

Call-by-need!
Simplicity

Must reason about this

What doesn’t need to be there?
Laziness by Transformation

- Goal:
  Exploit existing compilers for strict languages
  Code may be hard to reason about, but compiler isn’t!

- Existing Work:
  Aimed at performance

- Another possible goal:
  Simplicity
Writing Lazy Code in a Strict Language

à la Scheme:

(delay e) \iff (x () \Rightarrow e)

(force e) \iff (e ())

But it's not so simple:

- This is call-by-name
- Getting call-by-need is harder.
Compiling By Transformation

- Laziness explicit in intermediate language
- Not implicit in target language

Diagram:

- Haskell
- Compiler
  - Front-End
  - Strict Code
  - Simplifier
  - Linker
  - Link on Native Code
- RTS
- Profit!! (e.g. Reasoning)
Conclusions

• Laziness is nice
  (Makes reasoning about programs easier)

• Laziness is not so nice
  (Makes reasoning about RTS harder)

• Combine strict language compilers with program transformation
  (And get the best of both worlds)
Thanks to:

Tom Harke
Rebekah Leslie
Andrew Tolmach

Jim Hook
Andrew McCrcreight

(No one of whom endorses the format of this talk)

Questions?