Essence of Functional programming
Part 2

Clara B. Behrmann    Johan Brinch    Frej Soya

April 26, 2009
Monads Invariants
- The 3 Invariants
- Using the invariants
- Monads and lists

Continuations
- The CPS interpreter
- Call with current continuation
- Monads can express CPS
- CPS can express Monads

What’s the gain...
- I don’t care about interpreters!
- Examples
Monad Laws

The monad laws consists of 3 invariants, that any monad must adhere to:

\[
\begin{align*}
\text{return } a & \quad >>= k & = k \ a \quad (1: \text{Left identity}) \\
k & \quad >>= \text{return} & = m \quad (2: \text{Right identity}) \\
(3: \text{Associativity}) & & \\
\quad k >>= (\lambda a \rightarrow (k \ a) >>= (\lambda b \rightarrow h \ b)) & = (k >>= (\lambda a \rightarrow k \ a)) >>= (\lambda b \rightarrow h \ b)
\end{align*}
\]
The Error monads

Show how the error monad follows the invariants (or pick another monad)
Monad syntax in haskell

1-3 slides of examples comparing bind/unit with do-syntax (transliteration)
How does them monad laws help me?

1-3. How monads help the naive programmer. (Don’t break normal expectations)
For list monads

For the list monad together with the 3 invariants. The following invariants for Map and join apply.

- MapM
- JoinM

- There are 7 extra invariants.

Using the last one (8th) together with the 7 above, we can deduce the original 3 monad invariants.
Monads can generalise list comprehensions

Not that interesting?, Summary of another paper
Basicly any list comprehension in haskell can be translated
to a monad.
Thus list comprehensions are syntactic sugar for some monads.
Monad of continuations

The continuation monad “K”
Expressing continuations using monads.
TODO: Write the K monad in bind and return form.
killed this slide

Creating a CPS interpreter using monads. But this time wadlers also “simplifies”

- By removing each of occurrence of >>=
- adding “bits” to front to capture the continuation
- adding “bits to the end pass the continuation

Each operation passes on the actual computation to the next function as a 'continuation'
What does continuation mean?

Example with add Instead of evaluating the actual “add“. The “add” computations is passed on to the next expression the interpreter meets. This is done for all Terms, not just add.
Call with current continuation

This is too wordy on purpose... sorry about that ;).

- Callcc takes the continuation and creates a new Function
- The function is saved in the environment
- Interprets the remaining expression with new environment
- Var extracts the continuation from the environment
- The Function is called with the “4” as argument
The CPS interpreter can act as a Monad interpreter

By selecting the right type for \texttt{Answer}, the CPS interpreter can act as the original monad based interpreter.

TODO: Write the definitions of \texttt{PromoteK}, \texttt{ShowK}
Preserves modularity

Wadler shows it’s just as simple with the CPS style monad interpreter as using the monad interpreter. Examples with Error, State and Output monad. (Not that simple but anyhow).
Comparing CPS and Monads

Wadler compares CPS and Monads

- CPS always provides escape facility.
- For monads it’s a choice
- We can fix this with expressing CPS as monads.
Applying this from a usage viewpoint

If i’m not a language researcher, when is this cool?

▶ Example A
▶ Example B
▶ Example C
Example A

A
Example B
Example C

C

Essence of Functional programming
Clara B. Behrmann, Johan Brinch, Frej Soya

Outline

Monads Invariants
The 3 Invariants
Using the invariants
Monads and lists

Continuations
The CPS interpreter
Call with current continuation
Monads can express CPS
CPS can express Monads

What’s the gain...
I don’t care about interpreters!
Examples