Liskell

Lisp Syntax with Haskell Semantics

ILC07

Clemens FRUHWIRTH
clemens@endorphin.org

Static Typing

- Static typing means doing type checking at compile time
- Catch errors earlier
- Programs that compile work (yes, I'm serious)
- No runtime type checking
- Requires no type tags for objects

Pure Programming

- Side Effects depend on the order of evaluation
- Pure programming means side effect free programming
- When we omit side effects the order of evaluation becomes unimportant
- Removing order and sequence allows parallelization (vectorization)

Type inference

- Type annotations are boring and cost programmer time
- Type inference frees you from type annotations
- Type inference is guaranteed to work by the language design in Haskell

Why not Haskell?

- Syntax uncomfortable for editing
- Inflexible Syntax
- Meta-Programming not integrated well due to syntax

Why Liskell?

- Haskell inside
  - Functional, Pure, Lazy, Strong and Static Typing, Type Inference, Type Classes
- Lisp outside
  - Symbolic expression syntax, Meta-Programming

Haskell vs. Liskell

```
module Main where

fact1 n =
  if n == 0
     then 1
     else n * (fact1 (n-1))

fact2 n =
  let rec = fact2 (n-1)
  in case n == 0 of
       True -> 1
       False -> n * rec
```

```
(defmodule Main _ ())

(define (fact1 n)
  (if (== n 0)
      1
      (* n (fact1 (- n 1)))))

(define (fact2 n)
  (let ((rec (fact2 (- n 1))))
    (case (== n 0)
      (True 1)
      (False (* n rec)))))
```

Live Coding

Meta-programming in Liskell

- The compilation environment contains a series of Parse Tree Transformers
- The compilation environment is mutable by the source code compiled
- Parse Tree Transformers are an abstraction of Macros
- backquoting and defmacro are built on top of Parse Tree Transformers

cond expansion

- We can implement cond by expanding it to nested if clauses.

```
(cond ((> a b) LT)
      ((> b a) GT)
      (True    EQ))
```

→

```
(if (> a b)
    LT
    (if (> b a)
        GT
        (if True
            EQ
            (error ".."))))
```

cond implemented as macro

```
(defmacro (cond pts)
  (case pts
    ((Nil) `(error "Broken cond"))
    ((Cons (Cons guard
                 (Cons action Nil))
           rest)
     `(if ,guard
          ,action
          (cond ,@rest)))))

(add-dspr (expression cond))
```

Performance of Lis/Haskell

Program & Logs	Execution Time (Faster)	Memory Use (Smaller)
binary-trees	-1	3.9
chameneos	7.7	32
fannkuch	-2.5	2.7
fasta	-2.1	98
k-nucleotide	1.4	-1.9
mandelbrot	-1	4.8
n-body	-1.8	3.8
nsieve	1.5	4.6
nsieve-bits	1.2	3.5
partial-sums	-1.2	4.2
pidigits	2.2	9
recursive	1.2	6
reverse-complemen	-2.5	-1.2
spectral-norm	-2.4	17
startup	6.9	
sum-file	1.9	3.9

http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=ghc&lang2=sbcl

Conclusions

- Types are a cheap but effective safety net
- Functional pure programming is worth pursuing (vectorization)
- Liskell dresses Haskell nicely
- Liskell is available
- Liskell extends meta-programming into the pure functional programming realm