Contents

History of Functional Programming

- Combinatory logic λ-logic (1920s, 1930s), Foundations for *mathematics*, not computing
- WW2 changed everything
- LISP (late 50s, early 60s), Artifical Intelligence but very functional
- APL (early 60s), symbol-based, functions/operators as building blocks
- ML, SASL, NPL (1970s), type-inference, pattern-matching
- FP - John Backus Turing Award Speech (1977), inventor of Fortran and much parsing technology argues for functional programming
- Haskell starts (1987)

ML

- Robin Milner and co
- Developing early theorem provers
- Provers based on a logic called the Logic of Computable Functions
- Needed a very well-defined programming language to implment them
- Enter Meta Language (ML)
- Still the basis for most modern theorem provers
- Evolved into SML and OCaml

A Running Example

Expressions

We are going to write functions that manipulate expressions in a variety of ways

```
data Expr = Val Float
          | Add Expr Expr
          | Mul Expr Expr
          | Sub Expr Expr
          | Dvd Expr Expr
      deriving Show
```

So the expression (10+5)*90 is written

```
Mul (Add (Val 10) (Val 5)) (Val 90)
```

An Evaluator

We can write a function to calculate the result of the expressions

```
eval :: Expr -> Float
eval (Val x) = x
eval (Add x y) = eval x + eval y
eval (Mul x y) = eval x * eval y
eval (Sub x y) = eval x - eval y
eval (Dvd x y) = eval x / eval y
```

A Simplifier

We can write a function to simplify an expression

```
simp :: Expr -> Expr
simp (Val x) = (Val x)
simp (Add e1 e2) = let (Val x) = simp e1
                       (Val y) = simp e2
                   in Val (x+y)
...
```

Matching if simp e1 == Val x, x = value of simp e1

```
simp(Add e1 e2) = Val(eval(simp e1) + eval(simp e2))
```