Miranda – A Functional Language

By Dan Vasicek

2009/11/15

Outline of this Talk

- Some History and General Information
- Miranda Philosophy
- Introduce Syntax via Examples
- Currying &High Order Functions
- Lazy Evaluation & Infinite Lists
- Type Specification

Miranda History

- Influenced by ML, SASL
- Designed by David Turner in 1983-6
- Revised 1987 and 1989 (and currently)
- Richard Bird & Philip Wadler *An Introduction to Functional Programming using Miranda*, Prentice Hall 1988 (2nd Ed using Haskell 1998)
- Influenced Haskell, Mathematica, …

David Turner's Objective 1986

The aim of the Miranda system is to provide a modern functional language, embedded in a convenient programming environment, suitable both for teaching and as a general purpose programming tool

Miranda - Availability

- Download from http://miranda.org.uk/
- Free for personal or educational use
- Most recent version is mira-2041-i686-Cygwin.tgz ~2009 (Windows version)
- 400 KB file (but it requires Cygwin)
- Linux version mira-2042-i686-Linux.tgz
- 900 KB
- Language continues to be supported

Miranda – Why the name?

- **Miranda** is a Latin word meaning "to be wondered at"
- In Shakespeare's play, *"The Tempest"*, Miranda is the daughter of the magician, Prospero. She lives on an enchanted island, protected from all the evils of the world (which in this context may stand for side effects and other imperative features).

Miranda's Quote from the Tempest

- O, wonder!
  How many goodly creatures are there here!
  How beauteous mankind is! O brave new world,
  That has such people in't!
- The Miranda language is an introduction to the "Brave New World" of functional programming

Basic Themes of Miranda

- Miranda is purely functional - there are no side effects or imperative features of any kind
- A program, called a “script”, contains a collection of equations defining various functions and data structures
- Changing the order of equations in the script does not change the result
- No mandatory type declarations
- The language is strongly typed
- Program layout is important
- Executable “mathematics” with “minimal” programming syntax
- Uses “ordinary” mathematical notation
- Single assignment “variables”

Why consider Miranda?

- A Miranda program is typically 5 to 15 times shorter than the corresponding program in C or Java

Example Miranda Script

```
z = sq x / sq y
 sq n = n * n
 x = a + b
 y = a – b
a = 10
 b = 5
```

Notice the absence of syntactic baggage.

Notice that the order of the equations is not significant in this case. The 6 equations can be rearranged in order.

Discussion of the Script

- The first equation is

  z = sq x / sq y

  At this point the compiler/interpreter does not know what sq, x, and y are.

Discussion of the “Simple” Script

- The second line of the script is

  ```
  sq  n = n * n
  ```

  Now the “compiler” knows that “sq” is a function that takes a number as an argument and returns the square of that number as a result. And “n” is formal parameter that is not defined in the rest of the script. But no computation occurs.

Discussion of the Script

- The 3rd and 4th equations are

$x = a + b$

$y = a - b$

No computation occurs. These are just more definitions that need to stored for later use.

Discussion of the Script

- The last 2 equations define a and b
- Now the “compiler” knows enough to begin computation and returns the result:

9.0

- If we re-arranged the equations into any different order, the result would be the same.

Data Types

- Basic Data Types
  - Character - “A”, “a”, .. “Z”, “z”
  - Number - “1”, “2”, “3.14159”
  - Logical = True, False
- Data Structures
  - List - […]
  - Tuple (…)
  - User Defined (Abstract Data Types)

List Examples

- List elements must all have the same type
- week_days = ["Mon","Tue","Wed","Thur","Fri"]
- Days = week_days ++ ["Sat","Sun"]
- The “++” operator concatenates lists
- The “- -” operator does list “subtraction”
- For example:
- [1,2,3,4,5] -- [2,4] returns
- [1,3,5]
- [1..10] is shorthand for the first 10 integers
- [1..] means the infinite list of all integers

Some Example Definitions

- fac n = product [1..n]
  - Defines the factorial function
- result = sum [1,3..100]
  - Defines the sum of the odd numbers between 1 and 100. Notice that 100 is not in the sum.

Tuple Example

- Elements of a list must all be of the same type
- Elements of different types may be combined into “tuples”
- employee = ("Jones",True,False,39)
- Notice that round brackets () are used for tuples in contrast to the square brackets [] used for lists. (David Turner’s “compromise”.)

Miranda Programming Environment

- Miranda evaluates expressions and returns results interactively (interpretive)
- Syntax or type errors are signaled immediately
- Miranda interfaces with an editor (by default vi)
- There is a large library of “standard” functions
- Miranda programs may be invoked directly from the UNIX shell and may be combined via UNIX pipes

Recursion and Conditionals

gcd a b = gcd (a-b) b, **if** a>b

= gcd a (b-a), **if** a<b

= a , **if** a=b

Produces the greatest common divisor of a, b.

Quadratic Equation Solution (Local Variables)

- quadsolve a b c = error “cmplx rts", **if** delta<0
  = [-b/(2*a)], **if** delta=0
  = [-b/(2*a) + radix/(2*a),
  -b/(2*a) - radix/(2*a)], **if** delta>0
  where

delta = b*b - 4*a*c

radix = sqrt delta

Pattern Matching

- fac 0 = 1

  fac (n+1) = (n+1) * fac n

  Tail Recursion

- ack 0 n = n+1

  ack (m+1) 0 = ack m 1

  ack (m+1) (n+1) = ack m (ack (m+1) n)

  Not Primitive recursive

Fibonacci Numbers

fib 0 = 0

fib 1 = 1

fib (n+2) = fib (n+1) + fib n

Pattern Matching on Lists

```
sum [] = 0
sum (a:x) = a + sum x

product [] = 1
product (a:x) = a * product x

reverse [] = []
reverse (a:x) = reverse x ++ [a]
```

Pattern Matching to Access Elements of a Tuple

- fst (a,b) = a
- snd (a,b) = b

More Pattern Matching

- To take the first n elements from a list:

```
take 0 x = []
take (n+1) [] = []
take (n+1) (a:x) = a : take n x
```

Notice that this performs the same operation as the Mathematica function: Take[list, n]

Remove the first n Elements from a List (complement of take)

- drop 0 x = x
- drop (n+1) [] = []
- drop (n+1) (a:x) = drop n x

- Notice that this performs the same operation as the Mathematica function: Drop[list, n]

Take and Drop are Consistent

- Satisfy the Identity:
- take n x ++ drop n x = x

Functions

- can be both passed as parameters
- and returned as results
- Are left associative so that
- f x y means (f x) y (Currying)
- That is, f applied to x returns a function, which is then applied to y
- f, x, and y can all be functions

Applying Functions Repeatedly

- foldr f k [] = k
- foldr f k (a:x) = f a (foldr f k x)

- Looks a lot like the Mathematica function
- Mathematica Miranda
- Fold[f, x, list] foldr f x list
- Mathematica has extra syntax
- Fixed Point Theorem

Higher Order Function Produces Other Functions

- sum = foldr (+) 0
- product = foldr (*) 1
- reverse = foldr postfix []

  where postfix a x = x ++ [a]

Lazy Evaluation

- No sub-expression is evaluated until its value is known to be required. For Example:

cond True x y = x

cond False x y = y

- cond (x=0) 0 (1/x) will not abend
- take 5 [1..] will return a result [1, 2, 3, 4, 5]

Polymorphic strong typing

- Every expression and every sub-expression has a type
- Type can be deduced at compile time
- And any inconsistency in the type structure of a script results in a compile time error message
- Miranda scripts can include type declarations

Primitive Types

- There are three primitive types,
  - num, (1, 2, 3.14159265)
  - bool, (True, False)
  - char. (a, b, c,.. A, B, C ..)
- The type num comprises both integer and floating point numbers

Composite Types

- “John” is [char] that is a list of characters
- “John” is represented as [“J”, “o”, “h”, “n”]
- Week_days is [[char]] (a list of lists of char)

- Type specification is optional. For example when defining a function we could specify the type of the function:

```
sq :: num -> num  # This specification is optional
sq n = n * n
```

Examples of some types

- Well defined types
  - fac :: num -> num
  - ack :: num -> num -> num
  - sum :: [num] -> num
- Polymorphic Types (can have many types)
  - reverse :: [*] -> [*]
  - fst :: (*,**) -> *
  - snd :: (*,**) -> **
  - foldr :: (*->**->**) -> [*] -> **

User Defined Types

- A user can define new types with the syntax "::="
- Numerically labeled binary trees could be defined to have the type:
- tree ::= Nilt | Node num tree tree
  - Where Nilt and Node are tree constructors
  - t1 = Node 7 (Node 3 Nilt Nilt) (Node 4 Nilt Nilt)

Abstract Types
Use the keywords “abstype” & “with”

```
abstype stack *
with empty :: stack *
     isempty :: stack * -> bool
     push :: * -> stack * -> stack *
     pop :: stack * -> stack *
     top :: stack * -> *
stack * == [*]
empty = []
isempty x = (x=[])
push a x = (a:x)
pop (a:x) = x
top (a:x) = a
```

Separate compilation and linking

- **%include** "pathname"
- **%export** names
- **%free** list of overloaded functions
- **%export** list of exported functions

Some example programs available on the web

- ack.m the ackermann function
- divmodtest.m tests properties of **div** and **mod**
- fibs.m tabulates fibonacci numbers
- hanoi.m solves the problem `towers of hanoi'
- powers.m prints a table of powers
- primes.m infinite list of prime numbers
- pyths.m generates pythagorean triangles
  hamming.m prints hamming numbers
- queens.m all solutions to the eight queens problem
- queens1.m finds one solution to the eight queens problem
- quicksort.m Miranda definition of quicksort
- selflines.m curiosity - a self describing scroll of lines
- stack.m defines stack as an abstract data type
- treesort.m Miranda definition of treesort
  edigits.lit.m infinite decimal expansion of the digits of `e‘ (literate script)
- rational.m package for doing arithmetic on rationals
- refoliate.m a tree problem (literate script)
- topsort.m topological sort
- matrix.m matrix package
- set.m defines set as an abstract data type
- kate.lit.m a Miranda literate script that is also a LaTeX source file see the LaTeX output kate.pdf [59k]
  genmat.m parameterised version of matrix package j
- ust.m text formatting program
- mrev (executable) Miranda version of the UNIX `rev' command
- box (executable) program for reboxing Miranda comments
- box.m function definitions for `box'
- unify.m package for doing 1st order unification
- polish.m testbed for unify.m