Functional Programming

Christopher D. Clack

FUNCTIONAL PROGRAMMING

Lecture 4

MIRANDA

FUNCTIONAL PROGRAMMING MIRANDA

This lecture introduces the functional language Miranda, including a brief demonstration of the Miranda system and some basic discussion of Miranda syntax, types, tuples and conditionals

An online book is available that was originally intended as a self-study text for first-year first-term undergraduates who may not have had prior experience of programming (and should therefore be an easy read for students on this module)

This book is required (essential) reading up to and including Chapter 6. A link is provided on the Moodle page

CONTENTS

FUNCTIONAL PROGRAMMING MIRANDA

MIRANDA IN CONTEXT

Designed & implemented by David Turner (PhD student of Dana Scott, who was Church's PhD student)

Miranda uses graph reduction with sharing and normal-order reduction ("lazy evaluation"), and has a polymorphic type system [see implementation lectures for graph reduction and sharing]

Miranda's type system is simpler than Haskell's (e.g. no type classes) yet both are based on the Hindley-Milner system. Miranda's syntax is clean and uncluttered, ideal for modelling specifications and for teaching

Here we use Miranda primarily for its elegance and lack of syntactic clutter when discussing concepts and styles for **pure** functional programming

Miranda in context

David Turner (PhD student of Dana Scott, who was Church's PhD student) is the designer and implementor of three purely functional programming languages: SASL (1972), KRC (1981) and Miranda (1985)

Miranda had a strong influence on the later language Haskell. It uses graph reduction with sharing and normal-order reduction (known as "lazy evaluation"), and has a polymorphic type system

Almost everything that is in Miranda is also in Haskell, but Haskell has some concepts (such as type classes and monadic I/O) that do not appear in Miranda. Miranda's type system is simpler than Haskell's (e.g. with a single numeric type) yet both are based on the Hindley-Milner system. Miranda's syntax is clean and uncluttered, ideal for modelling specifications and for teaching: by contrast Haskell is a bigger language with implementations that give much faster runtime performance

Here we use Miranda primarily for its elegance and lack of syntactic clutter when discussing concepts and styles for **pure** functional programming

FUNCTIONAL PROGRAMMING
MIRANDA

MIRANDA INSTALLATION AND USE

Runs on Unix (Mac OS, SunOS, Linux, Cygwin …)

Is normally used from the command line (Terminal app, SSH)

Default access: remote login to knuckles.cs.ucl.ac.uk

- available to all CS students, and on request to non-CS UCL students

Alternative access (not fully supported): install Miranda on your own computer at home

See the COMP0020 Moodle page

Miranda Installation and Use

Miranda runs on Unix (including Mac OS, SunOS, Linux and Cygwin) and is always used from the command line (e.g. on MacOS via the Terminal app, or via SSH running in the Terminal app)

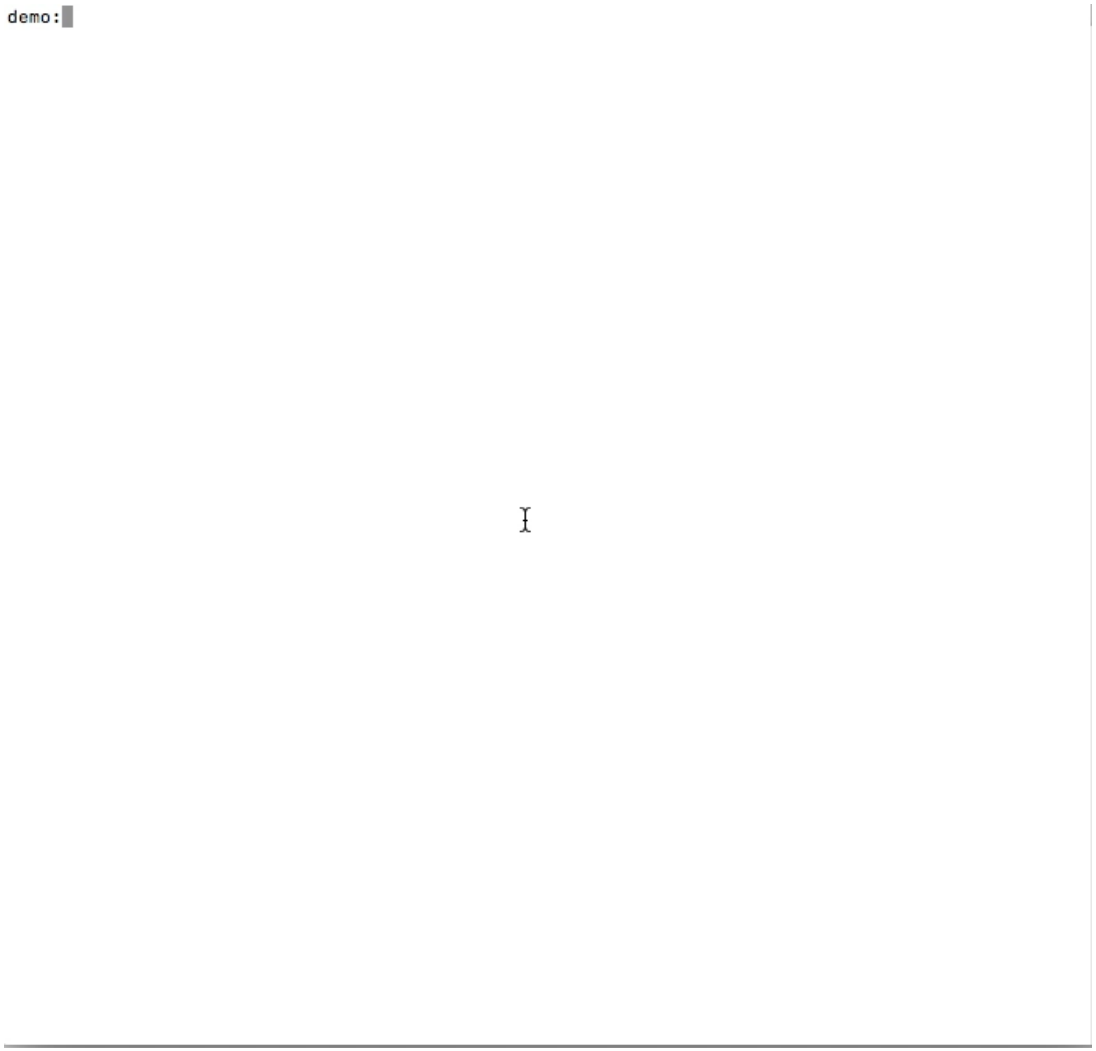
The **default way** to use Miranda is via remote login to the UCL-CS Linux server (knuckles.cs.ucl.ac.uk). A login account on knuckles is provided automatically to all CS students, and on request to UCL students from other departments

An **alternative** (not fully supported) way is to install Miranda on your own computer at home.

See the COMP0020 Moodle page for more information (under "Resources")

FUNCTIONAL PROGRAMMING MIRANDA

MIRANDA DEMONSTRATION



Miranda demonstration

Here is a “first steps” demonstration of Miranda running in a Terminal window on MacOS High Sierra. Typing “mira” launches the Miranda system, showing an introductory message and confirming that the default file for your program is called “script.m”. Finally the prompt “Miranda” is displayed, because Miranda is an **interpretive** environment

At the Miranda prompt you can either type a command (starting with a forward-slash) or an expression to be evaluated

The command to quit and return to the Unix shell prompt is either “/quit” or “/q”. The next most important command is “/h” to get simple help

Miranda will evaluate arithmetic expressions such as (3+4)*(6+7). More helpfully, your expression can also use names whose values have been defined and saved in the file “script.m” (the file “script.x” contains compiled code)

“/e” will launch your default text editor (here we use “vim” but it can be changed) and here are some example bindings of names to expressions (x = 3+4, y = 6+7, and main = x*y). Save and quit and Miranda automatically compiles and type-checks your program. The name **main** has no special meaning in Miranda, so ask Miranda to evaluate the name main

We can also define functions e.g.

```
inc x = x + 1
main = inc  45
```

FUNCTIONAL PROGRAMMING MIRANDA

COMMENTS AND NAMES

```
|| Example Miranda code

|| Here is a simple definition for some text:

message = “hello world”

|| Here is a function that adds one to a number:

inc x = x + 1        || You can place another comment here
```

Comments and names

It is very important to use comments in your Miranda programs (as in any programming language)

In Miranda a comment starts with the two vertical bar characters “||” and continues to the end of the line

This means that comments can either occupy an entire line, or can be placed to the right of a line of code

The simple definitions seen so far bind a NAME to an EXPRESSION. When defining a **function** we give the function name followed by one or more argument names followed by “=“ and then the function body (which is an expression)

Each binding must be **unique** within a specified scope (i.e. an area of code where the binding is effective):

- The scope of an argument name is the function body (only)
- Nested scope (see later) permits different bindings for the same name (with the innermost binding taking precedence over outer bindings) but re-using names is not a good idea

Names MUST start with a LOWERCASE alphabetic character: thereafter the name may contain uppercase and lowercase characters, numbers and underscores

FUNCTIONAL PROGRAMMING MIRANDA

TYPES, TYPE CHECKING AND TYPE INFERENCE

- Numbers: 42 is of type **num**
- Characters ’A’ is of type **char**
- Text: ”hello" is of type **[char] (pronounced “list of char”)**
- Truth values: True is of type **bool**
- Functions: a function of one argument will be of type **argtype -> resulttype** (e.g. num -> num)

Type checking occurs at compile time, to check that operators/functions are applied to data of the correct type

At the Miranda prompt, typing a name followed by two colons gives the type of that name. In a program you can specify the type of a name by writing the name followed by two colons and then followed by the type

If types are not given by the programmer, Miranda will attempt to **infer** types from the way that names are used

Types

Types help to improve the organisation of data and programs, and help to detect errors – the type system is a debugger

All data has a type, for example:

- Numbers: 42 is of type **num**
- Characters: ’A’ (in single quotes) is of type **char**
- Text: ”hello" (in double quotes) is of type **[char] (pronounced “list of char”)**
- Truth values: True and False (starting with capital T and F) are both of type **bool**
- Functions: a function of one argument will be of type **argtype -> resulttype** (where -> is pronounced “arrow”) – e.g. num -> num [see later for more complex types]

Type checking occurs at compile time, before the program is run, to check that operators (e.g. +) and functions are applied to data of the correct type

At the Miranda prompt you can request the type for a name by typing the name followed by two colons

In a program you can specify the type of a name by writing the name followed by two colons and then followed by the type

If types are not given by the programmer, Miranda will attempt to **infer** types from analysis of how operators and functions are applied to names and values

FUNCTIONAL PROGRAMMING MIRANDA

TUPLES AND CONDITIONALS

A **tuple** is a data structure that can hold many expressions or values, and each may have a different type.

A tuple of a num and a bool has type (num, bool) and the ordering of items is important.

- (”Ted", 23, "2 May Rd", False) :: ([char], num, [char], bool)
- (34, True) :: (num, bool)

  Note that (34, True) DOES NOT EQUAL (True, 34) – they are different values and different types
- ("increment", inc) :: ([char], num->num)

Conditional guards in function definitions are checked in top-down order: the first True triggers its result. E.g.

f x = “good morning”, if (x < 12)
= “good afternoon”, if (x < 17)
= “goodbye”, otherwise

Tuples

A simple Miranda data structure is the tuple

A tuple can hold many expressions or values, each of which can be a different type. A 2-tuple holds 2 expressions and is a different type to a 3-tuple (which holds 3 expressions). A tuple of a num and a bool has type (num, bool)

The ordering of items in a tuple is important. Here are some example tuples:

- (”Ted", 23, "2 May Rd", False) has type ([char], num, [char], bool)
- (34, True) has type (num, bool)

  Note that (34, True) DOES NOT EQUAL (True, 34) – they are different values and different types
- ("increment", inc) has type ([char], num->num)

Conditionals

In Miranda conditional evaluation is specified using “guards” within function definitions. For example:

f x = “good morning”, if (x < 12)
= “good afternoon”, if (x < 17)
= “goodbye”, otherwise

Conditional guards are checked in order, top down. The first to evaluate to True triggers the associated result and no further checking is done

FUNCTIONAL PROGRAMMING
MIRANDA

In summary, this lecture has introduced the functional language Miranda, including a brief demonstration of the Miranda system and some basic discussion of Miranda syntax, types, tuples and conditionals

SUMMARY

- Miranda in context
- Miranda installation and use
- Miranda demonstration
- Comments and names
- Types, type checking and type inference
- Tuples and Conditionals

FUNCTIONAL PROGRAMMING
MIRANDA