Functional and Concurrent Programming

Simon Thompson
s.j.thompson@kent.ac.uk

CO545 Lecture 1
CO545: functional and concurrent programming
CO545: functional and concurrent programming

Lectures
22 lectures: introduction, functional programming, concurrent programming, going further
CO545: functional and concurrent programming

**Lectures**
- 22 lectures: introduction, functional programming, concurrent programming, going further

**Classes**
- 11 two-hour terminal sessions: from this week
CO545: functional and concurrent programming

Lectures
22 lectures: introduction, functional programming, concurrent programming, going further

Classes
11 two-hour terminal sessions: from this week

Resources
Moodle for slides, lecture recordings, programs, class and seminar resources
CO545: functional and concurrent programming

**Lectures**
22 lectures: introduction, functional programming, concurrent programming, going further

**Classes**
11 two-hour terminal sessions: from this week

**Resources**
Moodle for slides, lecture recordings, programs, class and seminar resources

**Lecturers**
Simon Thompson
s.j.thompson@kent.ac.uk
Dominic Orchard
d.a.orchard@kent.ac.uk
What will I learn?
What will I learn?

**Functional ideas**
- Values, names, evaluation,
- structured types, lists,
- higher-order functions,
- recursion, PBT.
What will I learn?

**Functional ideas**
- Values, names, evaluation,
- structured types, lists,
- higher-order functions,
- recursion, PBT.

**Concurrent ideas**
- Processes and messages,
- process ids and spawn,
- asynchrony and mailboxes,
- fail-safe and exits, …
What will I learn?

**Functional ideas**
Values, names, evaluation, structured types, lists, higher-order functions, recursion, PBT.

**Concurrent ideas**
Processes and messages, process ids and spawn, asynchrony and mailboxes, fail-safe and exits, …

**Put it into practice**
Using these ideas in practice in the Erlang programming language.
What will I learn?

<table>
<thead>
<tr>
<th>Functional ideas</th>
<th>Concurrent ideas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values, names, evaluation, structured types, lists, higher-order functions, recursion, PBT.</td>
<td>Processes and messages, process ids and spawn, asynchrony and mailboxes, fail-safe and exits, …</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Put it into practice</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using these ideas in practice in the Erlang programming language.</td>
<td>Haskell for “lazy”, typed programming. Other concurrency models; scaling up Erlang systems.</td>
</tr>
</tbody>
</table>
How will I learn?
Lectures ... interaction, Q&A, ...
Terminal sessions ... supervisor
Lecturers, anon Q&A, drop-in, ...
How will I learn?
How will I be assessed?

Four assessments: two functional, two concurrent.
How will I be assessed?

Four assessments: two functional, two concurrent.

Four assessments: a “driving test”, an in-class test, two take-homes.
Programming means making this …
Intel Xeon Phi 'Knight's Corner': 1+ teraFLOPS with double precision; 62 x86 processor cores and a 512 bit GDDR5 memory controller
... do this
A list of programming languages fetched from Wikipedia in CSV format – https://github.com/jamhall/programming-languages-csv
Hardware

Low-level but …

… fiendishly complicated.
The Turing machine

http://www.math.uri.edu/~kulenm/mth381pr/comput/fig.jpg
The von Neumann model
The von Neumann model

Computation is …

… performed in a sequence of steps

… through changing values stored in memory
Scaling up the von Neumann model
Scaling up the von Neumann model
Levels of abstraction

Instructions to store and fetch values from memory …

… and to perform arithmetical operations on these values and registers.

Linear sequence of code, with (un)conditional jumps.
Examples

- Object-oriented
- Object-based
- Structured
- Procedural
- Assembler
- Machine code

010010...
Levels of abstraction

Instructions to store and fetch values from memory …

… and to perform arithmetical operations on these values and registers.

Linear sequence of code, with (un)conditional jumps.

Plus symbolic names, library routines.
Examples

- object-oriented
- object-based
- structured
- procedural
- assembler
- machine code
- SPARC, 86x, ...
- 010010...
Levels of abstraction

Symbolic variables, simple control structures …

… reuse of sections of code, using global variables.
Examples

- object-oriented
- object-based
- structured
- procedural
- assembler
- machine code

- BASIC
- SPARC, 86x, ...
- 010010...
Levels of abstraction

Symbolic variables, simple control structures …

… reuse of sections of code, w/ parameters, return values, scopes.

Defined composite data types.
Examples

- Pascal, Modula 2
- BASIC
- SPARC, 86x, ...
- 010010...

Object-oriented

Object-based

Structured

Procedural

Assembler

Machine code
Levels of abstraction

Group together variables with the procedures and functions that operate on them.
Examples

JavaScript, Haxe
Pascal, Modula 2
BASIC
SPARC, 86x, …
010010…
Levels of abstraction

Group together variables with the procedures and functions that operate on them.

Inheritance allows objects to extend / modify other objects.
Examples

- Java, C#, C++
- JavaScript, Haxe
- Pascal, Modula 2
- BASIC
- SPARC, 86x, …
- 010010…

- object-oriented
- object-based
- structured
- procedural
- assembler
- machine code
The von Neumann model
Calculation

\[ 4x + 3y \]

when \( x = 2 \) and \( y = 6 \)

\[ = 4(2) + 3(6) \]

\[ = 8 + 18 \]

\[ = 26 \]

http://www.wikihow.com/Evaluate-an-Algebraic-Expression
Calculation

Computation = calculation.

Programming = defining your own functions

\[ f(x, y) = 4x + 3y \]

? \( f(2, 6) \)
  -> 4*2 + 3*6
  -> 8 + 18
  -> 26
Calculation

No variables / storage locations.

Just values of expressions …

… and names for values.

\[
f(x, y) = 4x + 3y
\]

? f 2 6
   --> 4*2 + 3*6
   --> 8 + 18
   --> 26
Abstraction hiding the VNM

No variables / storage locations.

Just values of expressions …

… and names for values.
Abstraction hiding the VNM

No variables / storage locations.

Just values of expressions …
… and names for values.
Abstraction hiding the VNM

No variables / storage locations.

Just values of expressions …
… and names for values.

Calculation
Immutable data
Named values and subexpressions

Von Neumann
Storage allocation, modification, data structure mutation
The variety of functional languages

**Haskell**
Leading “lazy” language: calculate on demand.
Strongly typed.
Side effects only via “monads”

**LISP**
Earliest language with funs.
Symbolic computing: eval.
Weak types.
AI applications

**OCaml**
Strongly typed but “strict”.
Side effects but controlled.
Systems programming e.g Xen
Basis of F# (in .NET fwd)

**Erlang**
Weakly typed, strict.
Concurrency at heart.
Side effects controlled.
Use? telecoms, WhatsApp, …
The variety of functional languages

**Haskell**
Leading “lazy” language: calculate on demand.
Strongly typed.
Side effects only via “monads”

**LISP**
Earliest language with funs.
Symbolic computing: eval.
Weak types.
AI applications

**OCaml**
Strongly typed but “strict”.
Side effects but controlled.
Systems programming e.g Xen
Basis of F# (in .NET fwd)

**Erlang**
Weakly typed, strict.
Concurrency at heart.
Side effects controlled.
Use? telecoms, WhatsApp, …
Interpreter

source program

program results
Compiler

source program

target program
Compiler

source program \rightarrow \text{front end} \rightarrow \text{IR} \rightarrow \text{back end} \rightarrow \text{target program}
Compiler

Uncover the structure in the text

Lexical analysis: find the words.
Parsing: find the structure.
Analysis: find aspects of the meaning.
Generate an intermediate representation.
Uncover the structure in the text

Lexical analysis: find the words.
Parsing: find the structure.
Analysis: find aspects of the meaning.
Generate an intermediate representation.

Compiler

IR: e.g. SSA format
Optimisation = Analysis + Transformation
Optimiser will combine many small optimisations e.g …
… data flow: float out calculations from loops.
Uncover the structure in the text
Lexical analysis: find the words.
Parsing: find the structure.
Analysis: find aspects of the meaning.
Generates an intermediate representation.

IR: e.g. SSA format
Optimisation = Analysis + Transformation
Optimiser will combine many small optimisations e.g ...
... data flow: float out calculations from loops.

Generate instructions for the target machine.
Allocating registers to symbolic variables.
Scheduling instructions.
Concurrency
Concurrency
Concurrency

Threads = multiple modifiers + one memory
Concurrency

Threads = multiple modifiers + one memory

Processes = multiple modifiers + their own memory
Fight over scarce resources

Message-passing concurrency

Each process has its own memory …

… garbage collected separately, too.

Communication is via message passing …

… no shared memory at all.

Concurrency is a design artefact …

… but maps onto hardware parallelism.
You know Java ... now here's Erlang

The basis of Erlang is functional ...
You know Java ... now here's Erlang

The basis of Erlang is functional ...

... which means that it works like a calculator ...
You know Java ... now here's Erlang

The basis of Erlang is functional ... 

... which means that it works like a calculator ... 

... but with more complex values than numbers.
Erlang is also concurrent

Erlang is also concurrent …
Erlang is also concurrent

Erlang is also concurrent …

… allowing different things to go on at the same time …
Erlang is also concurrent ...

... allowing different things to go on at the same time ...

... just like in the real world, and most computer systems.
Why functional?

Functional is more high-level: functions become data.
Why functional?

Functional is more high-level: functions become data.

With no side-effects (“immutable data”) algorithms clearer.
Why functional?

Functional is more high-level: functions become data.

With no side-effects ("immutable data") algorithms clearer.

New architectures (many/multicore) and patterns e.g. map/reduce.
Why concurrency?

Modularity: different “processes” in different processes.
Why concurrency?

Modularity: different “processes” in different processes.

Robustness: “let it fail” and others sort it out.
Why concurrency?

Modularity: different “processes” in different processes.

Robustness: “let it fail” and others sort it out.

Concurrency (independence) vs parallelism (happen at same time).
Pragmatic

At its core, Erlang is functional …
Pragmatic

At its core, Erlang is functional …

… and it does allow some side-effects (e.g. communication) …
At its core, Erlang is functional …

… and it does allow some side-effects (e.g. communication) …

… but not others (no Java-style variables).
Pragmatic

Erlang has types …
Pragmatic

Erlang has types …

… has a more free-wheeling approach to types than Java …
Pragmatic

Erlang has types …

… has a more free-wheeling approach to types than Java …

… but can use tools to check type correctness.
Useful

Erlang was designed by Ericsson for building telecoms systems …
Useful

Erlang was designed by Ericsson for building telecoms systems …

… that must be concurrent, high-availability, robust, fault-tolerant …
Useful

Erlang was designed by Ericsson for building telecoms systems …
… that must be concurrent, high-availability, robust, fault-tolerant …
Used today by WhatsApp!, VISA, PaddyPower, AlertLogic, æternity, …
Useful

Even if what you’re going to do is program in Java …
Useful

Even if what you’re going to do is program in Java …

… it’s really useful to have another perspective on how to program.
Don’t forget …

Lecture on Tuesday, 10:00 here (in Grimond Lecture Theatre 1).

Terminal sessions start this week.