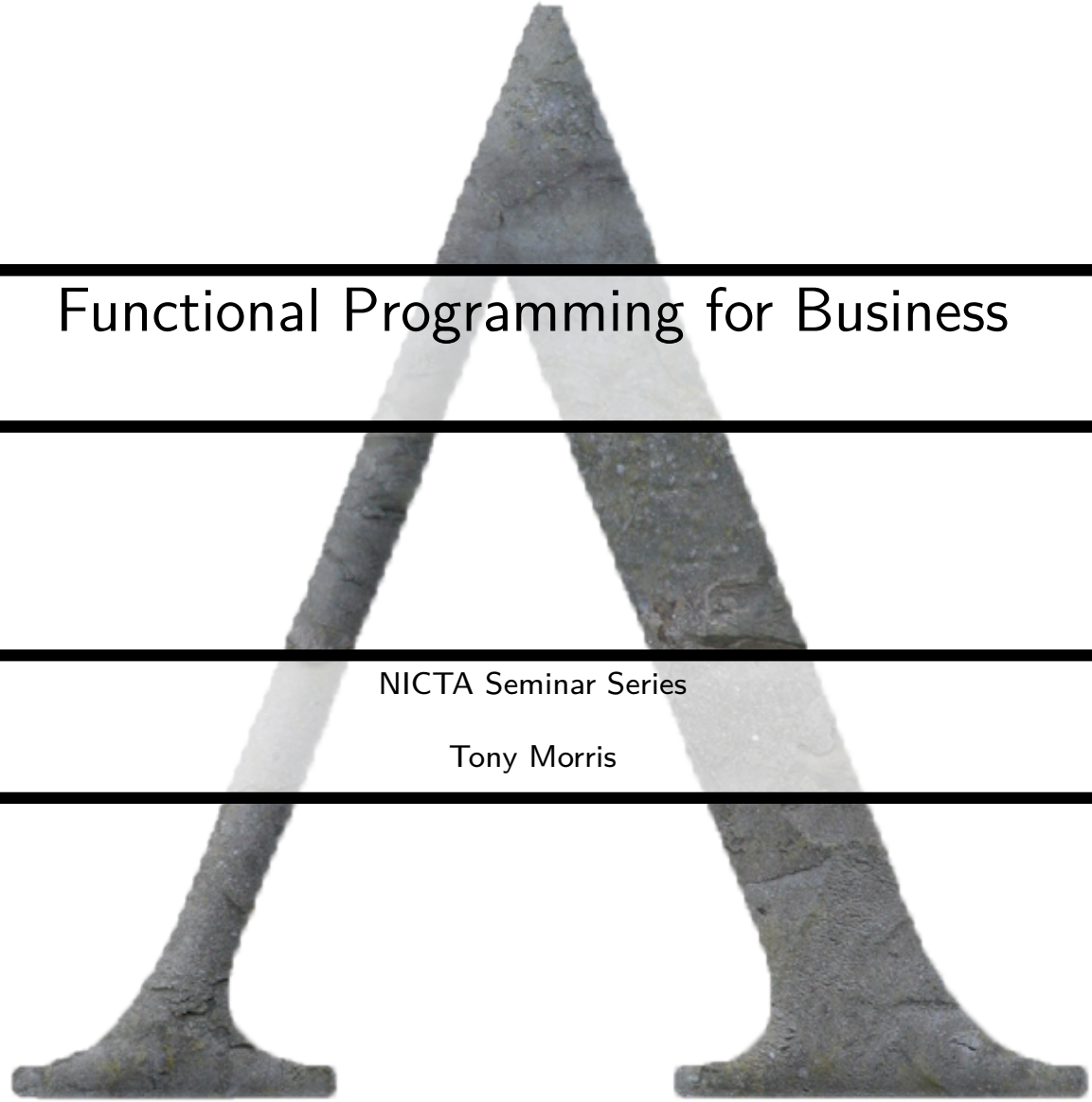
Functional Programming for Business



NICTA Seminar Series

Tony Morris

A bit about me

Senior Software Engineer, NICTA

- using functional programming for about 10 years in industry.
- have done my time in the trenches e.g. IBM
- former university lecturer; prefer sharing ideas in a more progressive context.
- I have nothing to sell you. If you choose to do it wrong, I won't "convince you".
- I'll blow you out of the market instead.
- my motivation is simple; to create more people with whom I can work.

A bit about me

Senior Software Engineer, NICTA

- using functional programming for about 10 years in industry.
- have done my time in the trenches e.g. IBM
- former university lecturer; prefer sharing ideas in a more progressive context.
- I have nothing to sell you. If you choose to do it wrong, I won't "convince you".
- I'll blow you out of the market instead.
- my motivation is simple; to create more people with whom I can work.

NICTA functional programming

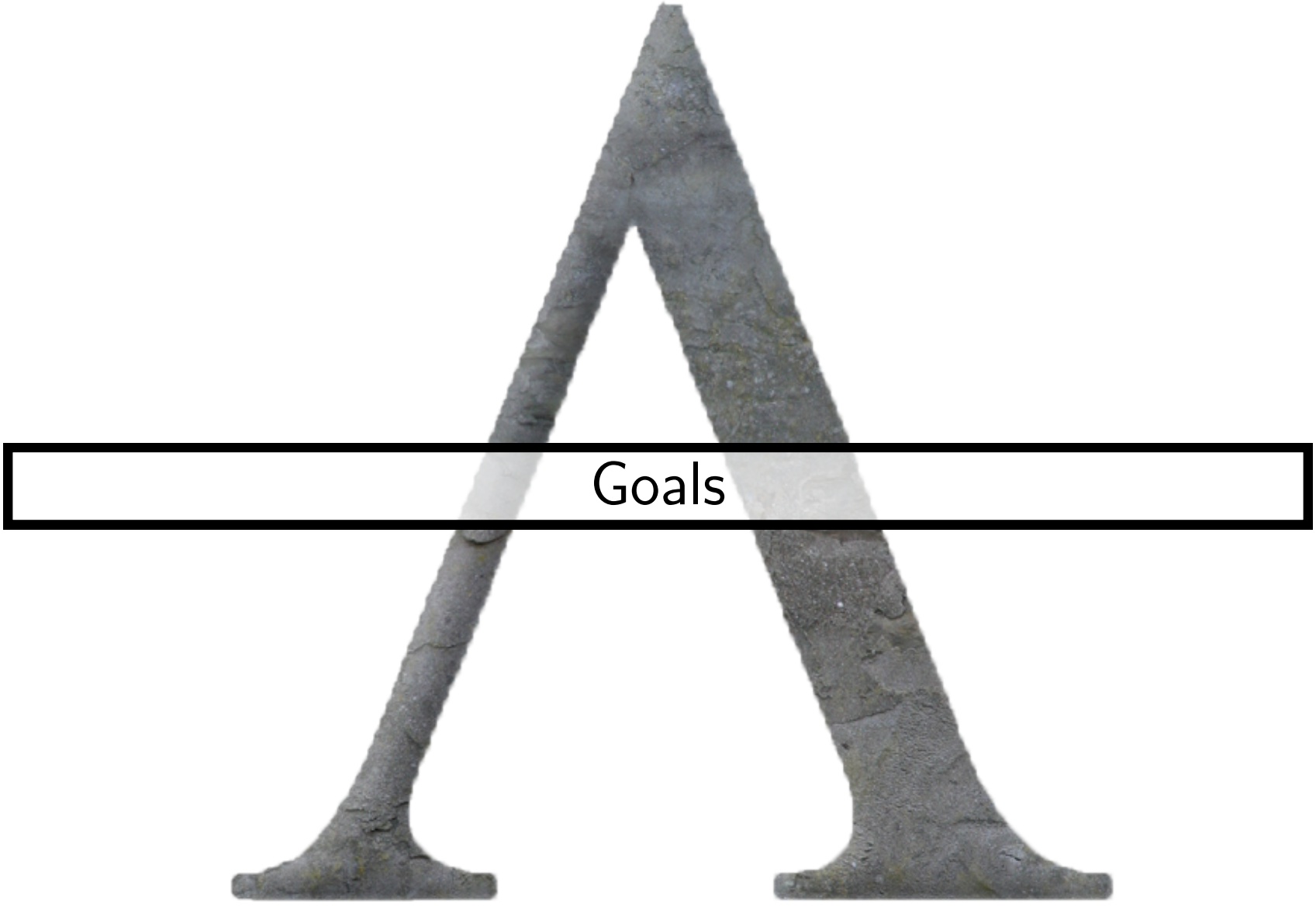
seL4

goanna

NICTA

John Backus

In a 1977 Turing Award lecture, John Backus put forward the question, Can We Be Liberated From the von Neumann Machine?[Bac78]


Goals

Goals
Goals for this talk

- discuss a concrete definition for Functional Programming (FP)
- introduce the principles of FP
- discuss a practical nomenclature to describe concepts related to FP
- become equipped with the tools to identify the hocus-pocus around FP
- have a bit of fun :)

Goals
Goals for this talk

- discuss a concrete definition for Functional Programming (FP)
- introduce the principles of FP
- discuss a practical nomenclature to describe concepts related to FP
- become equipped with the tools to identify the hocus-pocus around FP
- have a bit of fun :)

Goals
Goals for this talk

- discuss a concrete definition for Functional Programming (FP)
- introduce the principles of FP
- discuss a practical nomenclature to describe concepts related to FP
- become equipped with the tools to identify the hocus-pocus around FP
- have a bit of fun :)

Goals
Goals for this talk

- discuss a concrete definition for Functional Programming (FP)
- introduce the principles of FP
- discuss a practical nomenclature to describe concepts related to FP
- become equipped with the tools to identify the hocus-pocus around FP
- have a bit of fun :)

Goals
Goals for this talk

- discuss a concrete definition for Functional Programming (FP)
- introduce the principles of FP
- discuss a practical nomenclature to describe concepts related to FP
- become equipped with the tools to identify the hocus-pocus around FP
- have a bit of fun :)

What does FP mean?

- functional programming is a simple and principled thesis.
- from this thesis, many practical advantages follow.
- the practical consequences **do not define** functional programming.
- **all** programs achieve the principle of FP to some extent.

What does FP mean?

- functional programming is a simple and principled thesis.
- from this thesis, many practical advantages follow.
- the practical consequences **do not define** functional programming.
- **all** programs achieve the principle of FP to some extent.

What does FP mean?

- functional programming is a simple and principled thesis.
- from this thesis, many practical advantages follow.
- the practical consequences **do not define** functional programming.
- **all** programs achieve the principle of FP to some extent.

What does FP mean?
Referential Transparency

Placing `expression` under test for referential transparency

```
result = expression(args)
...
arbitrary1(result)
...
arbitrary2(result)
```

Refactor the program —has the program changed?

```
...
arbitrary1(expression(args))
...
arbitrary2(expression(args))
```

What does FP mean?
Am I functional programming?

To what extent does my program exhibit referential transparency?

- to what extent can I replace *expressions* with their *values*?
- to what extent am I functional programming? [Wad92]

What does FP mean?
Am I functional programming?

To what extent does my program exhibit referential transparency?

- to what extent can I replace *expressions* with their *values*?
- to what extent am I functional programming? [Wad92]

What does FP mean?
Tools

FAQ #1

- is (or is not) this programming language a "functional programming language?" [Sab98]
- I still do not know what one of these is.
- However...

What does FP mean?
Tools

FAQ #1

- is (or is not) this programming language a "functional programming language?" [Sab98]
- I still do not know what one of these is.
- However...

What does FP mean?
Tools

FAQ #1

- is (or is not) this programming language a "functional programming language?" [Sab98]
- I still do not know what one of these is.
- However...

What does FP mean?
Tools

We might instead ask

to what extent do my tools (including programming languages) provide support for me to exploit the principle and practical consequences that arise from functional programming?

Principles of functional programming

Frege's principle of compositionality [Jan01]

- a program is the composition of its constituent programs.
- modifying a program is the act of modifying the necessary part.
- the concept of a *program part* is *well-formed and measurable* [Hug89].

Principles of functional programming

Frege's principle of compositionality [Jan01]

- a program is the composition of its constituent programs.
- modifying a program is the act of modifying the necessary part.
- the concept of a *program part* is *well-formed and measurable* [Hug89].

Principles of functional programming

Frege's principle of compositionality [Jan01]

- a program is the composition of its constituent programs.
- modifying a program is the act of modifying the necessary part.
- the concept of a *program part* is *well-formed and measurable* [Hug89].

Principles of functional programming

Achieving program composition

- snake-oil sellers will point you at how to achieve program composition.
- or more likely, how to manage having failed to achieved it.
  - object-oriented hoo-haa
  - agile and lean and "oh look over there!"
- functional programming is necessary to the goal of composition.

Principles of functional programming

Achieving program composition

- snake-oil sellers will point you at how to achieve program composition.
- or more likely, how to manage having failed to achieved it.
  - object-oriented hoo-haa
  - agile and lean and "oh look over there!"
- functional programming is necessary to the goal of composition.

Principles of functional programming

Achieving program composition

- snake-oil sellers will point you at how to achieve program composition.
- or more likely, how to manage having failed to achieved it.
  - object-oriented hoo-haa
  - agile and lean and "oh look over there!"
- functional programming is necessary to the goal of composition.

Example

```
if(player.score > 12)
   player.setSwizzle(1000);
else
   player.setSwizzle(11);
```

refactor program

```
player.setSwizzle(player.score > 12 ? 1000 : 11);
```

Principles of functional programming

- functional programming is the extent to which this program property holds.
- *pure* functional programming is when this program property always holds.
  - including I/O programs
  - database programs
  - multi-threaded programs
  - web applications

Principles of functional programming

- functional programming is the extent to which this program property holds.
- *pure* functional programming is when this program property always holds.
  - including I/O programs
  - database programs
  - multi-threaded programs
  - web applications

Consequences of functional programming

Reasoning

- since our program expressions are *referentially transparent*, we may reason about each program part independently of all the others.
- this idea is called *equational reasoning*.
- equational reasoning gives to the ability to comprehend our programs; small or large.

Consequences of functional programming

Reasoning

- since our program expressions are *referentially transparent*, we may reason about each program part independently of all the others.
- this idea is called *equational reasoning*.
- equational reasoning gives to the ability to comprehend our programs; small or large.

Consequences of functional programming

Reasoning

- since our program expressions are *referentially transparent*, we may reason about each program part independently of all the others.
- this idea is called *equational reasoning*.
- equational reasoning gives to the ability to comprehend our programs; small or large.

Consequences of functional programming

Requirements change

- our program solution, at any level, is the composition of smaller, discrete programs . . .
- . . . only if *referential transparency is preserved*.
- if a requirement changes, we need only change those **independent parts** which correlate to that change.

Consequences of functional programming

Requirements change

- our program solution, at any level, is the composition of smaller, discrete programs . . .
- . . . only if *referential transparency is preserved*.
- if a requirement changes, we need only change those **independent parts** which correlate to that change.

Consequences of functional programming

Requirements change

- our program solution, at any level, is the composition of smaller, discrete programs . . .
- . . . only if *referential transparency is preserved*.
- if a requirement changes, we need only change those **independent parts** which correlate to that change.

Consequences of functional programming

Programs are sub-programs and can be reused

- since programs are (sometimes provably) delineated from others, the opportunity to reuse arises.
- functional programming gives rise to exploration of *principled abstraction*.

Consequences of functional programming

Programs are sub-programs and can be reused

- since programs are (sometimes provably) delineated from others, the opportunity to reuse arises.
- functional programming gives rise to exploration of *principled abstraction*.

Consequences of functional programming

Testing in isolation

- since functions do not perform *side-effects*, they can be tested in isolation.
- we can perform testing using *universal quantification*

```
> ((x ++ y) ++ z == x ++ (y ++ z))
OK, passed 100 tests.
```

Consequences of functional programming

Testing in isolation

- since functions do not perform *side-effects*, they can be tested in isolation.
- we can perform testing using *universal quantification*

```
> ((x ++ y) ++ z == x ++ (y ++ z))
OK, passed 100 tests.
```

Consequences of functional programming

Performance

- if programs are made of functions, they may be rearranged arbitrarily without altering the program outcome.
- a compiler may rearrange a program structure (but not outcome) to give optimal performance.
- existing runtime compilers do this to a small extent e.g. Java VM, .NET CLR

Consequences of functional programming

Performance

- if programs are made of functions, they may be rearranged arbitrarily without altering the program outcome.
- a compiler may rearrange a program structure (but not outcome) to give optimal performance.
- existing runtime compilers do this to a small extent e.g. Java VM, .NET CLR

Consequences of functional programming

Performance

- if programs are made of functions, they may be rearranged arbitrarily without altering the program outcome.
- a compiler may rearrange a program structure (but not outcome) to give optimal performance.
- existing runtime compilers do this to a small extent e.g. Java VM, .NET CLR

Consequences of functional programming

Proof by parametricity [Wad89]

- functions give rise to proof techniques, such as *parametricity*.
- parametricity is about deriving *theorems* from *polymorphic types*.

Consequences of functional programming

Proof by parametricity [Wad89]

- functions give rise to proof techniques, such as *parametricity*.
- parametricity is about deriving *theorems* from *polymorphic types*.

Consequences of functional programming
Parametricity

For example

Given a function with a type `(List a -> List a)`, a reader can immediately derive

Theorem

Every element in the result list appears in the input list.

Consequences of functional programming
Parametricity

Documentation

- functions give rise to parametricity.
- parametricity gives rise to proofy-carrying theorems.
- proof is a reliable and efficient method of program code comprehension.

Consequences of functional programming
Parametricity

Documentation

- functions give rise to parametricity.
- parametricity gives rise to proofy-carrying theorems.
- proof is a reliable and efficient method of program code comprehension.

Documentation

- functions give rise to parametricity.
- parametricity gives rise to proofy-carrying theorems.
- proof is a reliable and efficient method of program code comprehension.

Tying the knot

Why Functional Programming?

These reasons, and many more, are why all programming benefits by being *functional programming*

and not dysfunctional programming.

Questions?

References

- John Backus, *Can programming be liberated from the von neumann style?: a functional style and its algebra of programs*, Communications of the ACM **21** (1978), no. 8, 613–641.
- John Hughes, *Why functional programming matters*, The computer journal **32** (1989), no. 2, 98–107.
- Theo MV Janssen, *Frege, contextuality and compositionality*, Journal of Logic, Language and Information **10** (2001), no. 1, 115–136.
- Amr Sabry, *What is a purely functional language?*, Journal of Functional Programming **8** (1998), no. 01, 1–22.
- Philip Wadler, *Theorems for free!*, Proceedings of the fourth international conference on Functional programming languages and computer architecture, ACM, 1989, pp. 347–359.
- _____, *The essence of functional programming*, Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles