1 — Introduction

Einführung in die Funktionale Programmierung

Prof. Frank Puppe, Felix Herrmann, Alexander Gehrke

Sommersemester 2020

Lehrstuhl für Informatik VI, Uni Würzburg

What is Functional Programming?

What is Functional Programming?

- Functional Programming = using only functions without side effects
- Side effects include for example:
  - modifying a variable
  - modifying a data structure in place
  - input/output (files, user input, console output, ...)

Also, while it is not strictly functional programming, we will annoy the hell out of you by going on about how static typing is great, that you should make invalid states unrepresentable and a lot of other, opinionated points.

Why would we want that anyway?

You find two functions in the wild.

```
def Func1(num: Int): String =
  if isPrime(num) then
    "Primzahl"
  else if num == 0 || num == 1 then
    "Fast nichts"
  else
    num.toString
```

```
def Func2(num: Int): String =
  if num == 0 || num == 1 then
    "Fast nichts"
  else if isPrime(num) then
    "Primzahl"
  else
    num.toString
```

Are those two functions equivalent?

Why would we want that anyway?

Those are only equivalent if we can be sure to have functions without side effects. Otherwise, this could happen:

```
def isPrime(n: Int): Boolean =
  if simplePrimeTest(n) then
    primeNumberCounter += 1
    true
  else
    normalNumberCounter += 1
    false
```

Now inverting the conditions will skew the counters for prime and normal numbers.

Why would we want that anyway?

This example might seem stupid because it had to fit on two slides, but it is not too far off from what you will see in typical code.

If we say we don't allow side effects and therefore don't allow assignment to variables, this function could not have been written this way.

I've seen code where a `checkIsInCache(page)` also deletes the given page or where `validateUserInput` also saves the input somewhere.

Can we formalize the lack of side effects?

Referential transparency (RT) and purity

An expression e is referentially transparent if, for all programs p, all occurrences of e in p can be replaced by the result of evaluating e without affecting the meaning of p. A function f is pure if the expression $f(x)$ is referentially transparent for all referentially transparent x.

The Substitution Model

- RT invariant: everything a function does is represented by its return value
- Allows substituting equals for equals in a program

$\Rightarrow$ Equational reasoning about programs

The Substitution Model — Examples

```
scala> val x = "Hello, World"
val x: String = Hello, World

scala> val r1 = x.reverse
val r1: String = dlroW ,olleH

scala> val r2 = x.reverse
val r2: String = dlroW ,olleH   // r1 and r2 are the same
```

Can replace all occurrences of x by the value x refers to:

```
scala> val r1 = "Hello, World".reverse
val r1: String = dlroW ,olleH

scala> val r2 = "Hello, World".reverse
val r2: String = dlroW ,olleH   // r1 and r2 are still the same
```

The Substitution Model — Examples

```
scala> val x = new StringBuilder("Hello")
val x: StringBuilder = Hello

scala> val y = x.append(", World")
val y: StringBuilder = Hello, World

scala> val r1 = y.toString
val r1: String = Hello, World

scala> val r2 = y.toString
val r2: String = Hello, World
```

What happens, when we replace all occurrences of `y` by its value?

The Substitution Model — Examples

y is not referentially transparent:

```
scala> val x = new StringBuilder("Hello")
val x: StringBuilder = Hello

scala> val r1 = x.append(", World").toString
val r1: String = Hello, World

scala> val r2 = x.append(", World").toString
val r2: String = Hello, World, World
```

The Substitution Model — Examples

Back to our prime example.

```
if isPrime(2) then
  println("There were " + primeNumberCounter + " primes")
//prints 2
```

```
if true then
  println("There were " + primeNumberCounter + " primes")
//prints 1
```

```
println("There were " + primeNumberCounter + " primes") //prints 1
```

Those three programs should be equivalent because we are only replacing expression by their values. They are not. `isPrime` is therefore not a pure(RT) function.

So, what do we do now?

Okay, we have a problem. What were examples of things we are not allowed to do?

1. modify variable
2. modify a data structure in place
3. input/output (files, user input, console output, ...)

Is it even possible to write useful code, which does neither of those things?

So, what do we do now?

Okay, we have a problem. What were examples of things we are not allowed to do?

1. modify variable
2. modify a data structure in place
3. input/output (files, user input, console output, ...)

Is it even possible to write useful code, which does neither of those things?

Yes, although point 3 is subtle and we can't tackle that one just yet. But let's start with 1 and 2.

So, what do we do now? — Assignment — Factorial

Let's say we have the following old java code:

```java
public static int factorial(int n) {
  int product = n;
  int nextNumber = n - 1;
  while (nextNumber > 1) {
    product = product * nextNumber;
    nextNumber = nextNumber - 1;
  }
  return product;
}
```

Okay, we have multiple problems here. We are obviously assigning to variables. Also we use a while loop. While loops are useless. Why?

So, what do we do now? — Assignment — Factorial

Let's say we have the following old java code:

```java
public static int factorial(int n) {
  int product = n;
  int nextNumber = n - 1;
  while (nextNumber > 1) {
    product = product * nextNumber;
    nextNumber = nextNumber - 1;
  }
  return product;
}
```

Okay, we have multiple problems here. We are obviously assigning to variables. Also we use a while loop. While loops are useless. Why?

Because changing the condition of the while loop would require us to perform some side effect somewhere. And a while loop whose condition doesn't change either never starts or never ends.

So, what do we do now? — Assignment — Factorial

- Usually loops use incrementation variables or iterators, i.e. mutable state
- How do we write pure loops?

So, what do we do now? — Assignment — Factorial

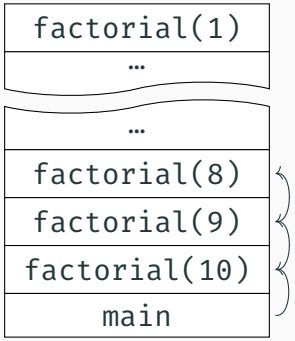
- Usually loops use incrementation variables or iterators, i.e. mutable state
- How do we write pure loops? By using recursion
- Let's implement factorial (product of all natural numbers up to a given one):

```
def factorial(n: Int): Int =
  if n <= 1 then 1
  else n * factorial(n - 1)
```

This is better. But there is one problem with rewriting while loops with recursion. Sometimes we might recurse so deep that we overflow the stack.

Aside: Recursion and the stack

The call stack holds all function invocations of our programm. If one function *a* calls another function *b*, a new element (called stack frame) for *b* is added to the stack, holding its parameters, local variables, return address etc. until the function returns.



- Calling `factorial(n)` will create *n* stack frames
- Size of the stack is limited
- If the stack becomes too large, we get a stack overflow => crash.

How do we get around that?

So, what do we do now? — Assignment — Factorial

- Compiler can optimize certain recursions to same bytecode as while loops
- Requirement: recursive call is in *tail position*, i.e. the recursive call is the last thing the function does before returning

```
//@annotation.tailrec => error
def factorial(n: Int): Int =
  if n <= 1 then 1
  else n * factorial(n - 1)
```

- Multiplication has to happen after recursive call returns

```
def factorial(n: Int): Int =
  @annotation.tailrec
  def go(n: Int, acc: Int): Int =
    if n <= 1 then acc
    else go(n-1, n*acc)

  go(n, 1)
```

- Directly returns result of recursive call

This transformation is always possible and even mechanical, but not always nice or easy.

So, what do we do now? — Assignment — Sum

Okay, factorial might be easy. But how about

```java
public static int sum(Collection<Integer> items) {
  int sum = 0;

  for (Integer item : items) {
    sum += item;
  }

  return sum;
}
```

This surely looks as if a variable is necessary.

So, what do we do now? — Assignment — Sum

But if we imagine a list can be asked for the first element(`head`), for all elements except the first (`tail`) and whether is is empty or not, the following is a purely functional solution:

```
def sum(l: List[Int]): Int =
  def go(l: List[Int], accu: Int): Int =
    if l.isEmpty then accu
    else go(l.tail, accu + l.head)

  go(l, 0)
```

We used the same trick as before: define a function `go`, which takes an accumulator to allow for tail call elimination.

So, what do we do now? — Changing data structures

Okay, fine. In the last example we got a list. But someone had to build that list. How would we do that if we can't assign to variables or change data structures in place?

So, what do we do now? — Changing data structures

Okay, fine. In the last example we got a list. But someone had to build that list. How would we do that if we can't assign to variables or change data structures in place?

Use immutable data structures. This means, on every modification, build a completely new data structure instead of changing the old one.

This might sound as if we need to accept an incredible performance hit for doing it this way. Luckily, most of the time, this isn't true.

We will talk more about lists in the next lectures, but let's use lists as a short example.

So, what do we do now? — Changing data structures

If we represent a list as a head element and a list of tail elements, the following operations are really fast:

remove the head of `l`	just return the tail of `l`
add new element `e` to `l`	just create a new list with `e` in its head and `l` as its tail
reverse `l`	recursively take the head of `l` and prepend it to a new list

None of those operations is more costly than modifying the list in place. The first two operations are so fast because they can use the old data structure because we can guarantee that the data never changes.

Immutable data structures, which can reuse the old data on various operations, are called *persistent*.

So, what do we do now?

It is possible to rewrite every piece of code to work without the use of side effects.

It might not always be easy to do or obvious how it can be done in principle, but it is always feasible.

It the next lectures, we will provide you with various strategies and patterns to make writing side effect free code easy.

Motivation Examples

On the next few slides there will be a couple of examples of programs/functions, which are either buggy or behave in an extremely unintuitive way.

We will will identify the problems with those examples but we won't give a detailed explanation of how to fix them. This will come in later lectures.

Motivation Examples — Bad Parents

On the next slide we try to write a small class, which records who has which father.

We have a small value class named `Person`, which stores a name and an age.

We also have a class which stores the father to each person and can ring in the new year, turning everyone one year older.

Motivation Examples — Bad Parents

```
public static class Person {
  String name;
  int    age;

  public Person(String name, int age) { this.name = name; this.age = age; }

  @Override public int hashCode() { return name.hashCode() + age; }
  @Override public boolean equals(Object other) {
    return other instanceof Person &&
      ((Person)other).name.equals(name) &&
      ((Person)other).age == age;
  }
}
public static class Parents {
  Map<Person, Person> father = new HashMap<>();

  public void addPerson(Person p) { father.put(p, null); }
  public void setFather(Person p, Person pa) { father.put(p, pa); }
  public Person getFather(Person p) { return father.get(p); }
  public void newYear() { for (Person p: father.keySet()) { p.age++; } }
}
```

Motivation Examples — Bad Parents

The problem here is that changing the age of a person also changes its hash value. This means that after ringing in the new year, some folks will lose their father.

The solution, of course, is to eschew mutable state and model the problem in a different way.

Motivation Examples — Bad API

In the following example, we have a small API class. It can validate whether a user name is valid (i.e. a user with that name exists). The case of the user name should not matter.

It should also provide a method to check whether an API token is valid.

Motivation Examples — Bad API

```
public static class Api {
  private List<String> users  = new ArrayList<>(); // users
  private List<String> tokens = new ArrayList<>(); // base64 crypto api tokens

  public boolean isValidUser(String userName) {
    return Util.stringExists(users, userName);
  }

  public boolean isValidApiToken(String token) {
    return Util.stringExists(tokens, token);
  }
}
public static class Util {
  public static boolean stringExists(List<String> strings, String s) {
    for (String ss : strings) {
      if (s.toLowerCase().equals(ss.toLowerCase()))
        return true;
    }
    return false;
  }
}
```

Motivation Examples — Bad API

The problem is that the `stringExists` method checks whether two strings are equal regardless of case. Base64 crypto tokens have to be checked with case in mind though, because turning a Base64 string lowercase changes the value the string represents.

Motivation Examples — Bad API

The problem is that the `stringExists` method checks whether two strings are equal regardless of case. Base64 crypto tokens have to be checked with case in mind though, because turning a Base64 string lowercase changes the value the string represents.

It might be argued that giving a misleading name to a function is always a problem, regardless of the programming paradigm. But it is possible to come to a point where the signature `boolean stringExists(List<String> l, String s)` is already incredibly suspect. This would prompt an experienced programmer to look deeper.

The solutions here are parametricity and higher order functions, both of which will be covered in later lectures.

Motivation Examples — Bad Playlist

We want to model a playlist class which holds a collection of songs and the information which song is currently selected.

The playlist might be empty, but if it is not empty, a song must be selected so that clicks on the 'next song' button are well defined.

We use -1 to mark that there is no song selected.

Let's try to find the bug in the following code.

Motivation Examples — Bad Playlist

```java
class BadPlaylist {
  public final List<String> songs = new ArrayList<String>();
  public int currentSong = -1;

  public void next() {
    if (currentSong != -1) {
      currentSong = (currentSong + 1) % songs.size();
    }
  }

  public void clear() {
    songs.clear();
    currentSong = -1;
  }

  public void setSongs(List<String> other) {
    songs.clear();
    songs.addAll(other);
    currentSong = 0;
  }
}
```

Motivation Examples — Bad Playlist

The bug was that `setSong` with an empty list as its argument would set `currentSong` to 0 instead of -1.

This would indicate that a song is selected even though there is no song.

The solution is to model the states of a our program in a way that makes it impossible to get into invalid states of this kind.

Real World Examples

If this all sounds like ivory-tower-can-t-be-used-in-the-real-world-stuff to you, there are reasons for that:

- it really is used a lot in academia because functional programming provides a sound basis for doing programming language research
- a lot of the concepts used in functional programming are directly borrowed from mathematics

Nevertheless, functional programming is gaining traction and here are a few examples where it is used today:

Real World Examples — JaneStreet

Jane Street is a trading firm handling about 13 billion dollars each day. Here is what they say about functional programming (OCaml):

> *Jane Street's technology group is small by design, which means we need to maximize the productivity of each person we hire. We believe functional programming helps us do that. But it's not about productivity alone: programming in a rich and expressive language like OCaml is just more fun.*
>
> *At Jane Street, functional programming isn't a tool we reserve for some special set of problems. From systems automation to trading systems, from monitoring tools to research code, we write everything that we can in OCaml.*

Real World Examples — Apollo Agriculture

Apollo Agriculture is a firm using machine learning to help farmers in Kenya.

AA use functional programming to combine data from various sources (satellite data, climate models, soil data, ...) to calculate how well a farmer could perform in certain places in Kenya.

They then do a risk assessment and provide credit to promising farmers as well as advice on how to increase their farm yields.

Real World Examples — Facebook

Facebook is a ...you already know what Facebook is.

Facebook uses functional programming (Haskell) for example to detect spam with a rule engine called sigma and a remote data access library called Haxl.

Both are very performant and integrate well with various other programming languages like C++.

Facebook uses FP for other tools as well.

Real World Examples — Twitter

Twitter is a ...

Twitter switched from Ruby on Rails to Scala some time ago.

They like the dual nature of OOP and FP:

> *Scala is a lot of fun to work in; yes, you can write staid, Java-like code when you start. Later, you can write Scala code that almost looks like Haskell. It can be very idiomatic, very functional — there's a lot of flexibility there.*

Real World Examples — Pandoc

Pandoc is a conversion library/program written in Haskell. It can convert from

commonmark, creole, docbook, docx, dokuwiki, epub, fb2, gfm, haddock, html, ipynb, jats, json, latex, markdown, markdown_mmd, markdown_phpextra, markdown_strict, mediawiki, man, muse, native, odt, opml, org, rst, t2t, textile, tikiwiki, twiki, vimwiki

to

asciidoc, beamer, commonmark, context, docbook, docbook5, docx, dokuwiki, epub, epub2, fb2, gfm, haddock, html, html4, icml, ipynb, jats, json, latex, man, markdown, markdown_mmd, markdown_phpextra, markdown_strict, mediawiki, ms, muse, native, odt, opml, opendocument, org, plain, pptx, rst, rtf, texinfo, textile, slideous, slidy, dzslides, revealjs, s5, tei, xwiki, zimwiki

and is used in various open source projects and companies.

Real World Examples — All the other languages

Also, nearly every other programming language starts to incorporate more and more functional themes.

- lambdas/anonymous functions were introduced to C++ and Java
- Optional was introduced to Java
- higher order functions like `map`, `flatMap` and `filter` were introduces to Java
- persistent data structures are alive and well in many languages like Scala and Clojure
- pure functions are a good idea in nearly every language (C, C++, Python, Perl, PHP, ...)
- Rust, in addition to having the concept of ownership, has functional concepts like type classes and immutable data structures