Technical University of Munich
Chair for Logic and Verification
Prof. Tobias Nipkow, Ph.D.
J. Rädle, L. Stevens, K. Kappelmann, MC Sr Eberl

WS 2020/21
06.11.2020
Deadline: 17.11.2020, 23:59

# Functional Programming and Verification

**Sheet 1**

**IMPORTANT:** You may only attend the tutorial you are matched with on TUMonline with a running Haskell environment as specified on the course website http://www21.in.tum.de/teaching/fpv/WS20/installation.html. You will also need a microphone and, optionally but very strongly recommended, a camera for the tutorial. Please read the notes on
http://www21.in.tum.de/teaching/fpv/WS20/exercises.html.

## Tutorial Exercises

**Exercise T1.1** Hello My Pair-Programming Friend :)

As a gentle kick-off, we will split into breakout rooms and check whether we can connect to our peers and compile and run the template repository provided on the installation website http://www21.in.tum.de/teaching/fpv/WS20/assets/haskell_test.zip. Talk about how lovely Haskell is and show your peers cool things you already learnt in the installation tutorial (e.g. executing functions with GHCi, Hoogle search, linter hints, etc.).

Once you made sure that you can connect to your peers, your tutor will briefly explain you the overall structure of a Haskell project. While doing so, she will also show you how to adapt the test project to set up your first tutorial project folder.

**Exercise T1.2** Hello Haskell

a) Define a function

```
offByOne :: Integer -> Integer -> Bool
```

that returns `True` if and only if one of its parameters is the successor of the other parameter.

b) Define a function

```
threeAscending :: Integer -> Integer -> Integer -> Bool
```

that returns `True` if and only if the sequence of parameters is strictly monotonically increasing.

c) Define a function

```
fourEqual :: Integer -> Integer -> Integer -> Integer -> Bool
```

that returns `True` if and only if all parameters are equal.

**Exercise T1.3** For Recursion See Recursion

a) Define a recursive function `fac :: Integer -> Integer` such that `fac n` $= n!$.

b) Define a function `sumEleven :: Integer -> Integer` such that `sumEleven n` $= \sum_{i=n}^{n+10} i$.
   *Hint:* use an auxiliary function.

**Exercise T1.4** Maximum Fun

Let `g` be the following function

```
g :: Integer -> Integer
g n = if n < 10 then n*n else n.
```

a) Define a recursive function

   ```
   argMaxG :: Integer -> Integer
   ```

   such that `argMaxG n` maximises `g` in the domain $\{0, \ldots, n\}$. Do not make any assumptions about `g`, that is write your function in a way such that it still works when the definition of `g` is changed. Return 0 for negative inputs.

b) Examine the definition of `g` above to determine when `argMaxG n` $\neq$ `n`. Use your observations to write a function `argMaxG' :: Integer -> Integer` that does not use `g` and satisfies the property `argMaxG' n = argMaxG n`.

c) Write a function `prop_argMaxGEquiv :: Integer -> Bool` that tests the equivalence for a given number.

d) Add QuickCheck version 2.* to your project dependencies and run your test in ghci using `quickCheck prop_argMaxGEquiv`. You will need to `import Test.QuickCheck` in your GHCi session before you can use the function `quickCheck`.

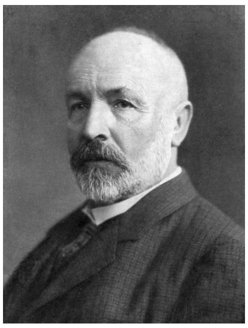
## Homework

**Important:** Read the submission guidelines on our website http://www21.in.tum.de/teaching/fpv/WS20/exercises.html.

This homework is all about numbers. You need to collect 6 out of 8 points (P) to collect a coin and become an aspiring number wizard.

**Exercise H1.1** Cantor's Creativity [a: 1P, b: 1P, c: 1P, d: 1P]

As a matter of course, Haskell knows about pairs; however, we sadly haven't learnt about them in class so far. But fear not! As shown by the great Georg Cantor, we can just encode pairs in a clever way. We do not directly follow the great Cantor's approach though but define a different encoding



Georg Cantor

function proposed by the MC Sr. The following functions only need to work for natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$.

a) Define the encoding function

```
myPair :: Integer -> Integer -> Integer
```

such that

$$\texttt{myPair x y} = 2^y(2x+1) - 1.$$

b) Define the inverse function

```
mySnd :: Integer -> Integer
```

such that

$$\texttt{mySnd (myPair x y)} = y.$$

*Hint:* Divide by 2 until the remainder is 0.

c) Define the inverse function

```
myFst :: Integer -> Integer
```

such that

$$\texttt{myFst (myPair x y)} = x.$$

d) Write a QuickCheck test with parameters $p, x, y \in \mathbb{N}$ that checks whether $p$ encodes the pair $(x, y)$.

*Hint:* You can restrict a test's domain with the `==>` operator, e.g. `x > 0 ==> x^3 >= 0`.

**Exercise H1.2** Esperantigu la entjerojn! [4P]

In this exercise, you will attempt to curry favour with the MC Senior by implementing, in his favourite programming language (Haskell), an algorithm to print numbers in his favourite spoken language (Esperanto). Concretely: write a function `numberToEo :: Integer -> String` that takes a non-negative integer below one million and returns its equivalent in words in Esperanto.

The Esperanto numeral system is quite simple: the basic building blocks are:[1]

| | | | | | |
|---|---|---|---|---|---|
| 0 | nul | 5 | kvin | 10 | dek |
| 1 | unu | 6 | ses | 100 | cent |
| 2 | du | 7 | sep | 1,000 | mil |
| 3 | tri | 8 | ok | | |
| 4 | kvar | 9 | nau | | |

[1] The correct spelling is actually 'naŭ', not 'nau', but bizarre encoding problems on Windows machines and other related problems is a can of worms the MC Sr did not feel like opening in the first week already.

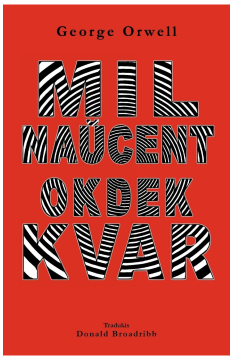



Figure 1: The Esperanto edition of George Orwell's famous book *Nineteen Eighty-Four*

*Tri* ringoj por la elfoj sub la hela ĉiel'
*Sep* por la gnomoj en salonoj el ŝton'
*Naŭ* por la homoj sub la morto-sigel'
*Unu* por la nigra reĝo sur la nigra tron'

Figure 2: The beginning of the ring poem from *Lord of the Rings* in Esperanto

Multiples of 10 between 20 and 90 and multiples of 100 between 200 and 900 are simply made by putting the corresponding digit in front of 'dek' (resp. 'cent'):

| | | | | | |
|---|---|---|---|---|---|
| 20 | dudek | 30 | tridek | 200 | ducent |

and so on. Multiples of 1000 between 2000 and one million are made by putting the corresponding number in front of 'mil', e.g.:

| | |
|---|---|
| 2,000 | du mil |
| 10,000 | dek mil |
| 11,000 | dek unu mil |
| 111,000 | cent dek unu mil |
| 234,000 | ducent tridek kvar mil |

Other numbers are composed similarly to most other languages you probably know: by combining multiples of thousands, multiple of hundreds, multiples of tens, and digits:

| | |
|---|---|
| 13 | dek tri |
| 33 | tridek tri |
| 42 | kvardek du |
| 1,984 | mil naucent okdek kvar |
| 234,567 | ducent tridek kvar mil kvincent sesdek sep |
| 937,191 | naucent tridek sep mil cent naudek unu |

**Hint:** The operator `(++) :: String -> String -> String`` allows you to concatenate two strings, e.g. `"kiel" ++ "ekzemplo" == "kiel ekzemplo"`. The `(^)` operator lets you take powers of integers.

This is it for the homework exercise. However, if you want to take part in the competition, do continue reading.

This exercise was posed by the Master of Competition Senior (MC Sr). It will be marked as part of your homework but also counts towards the competition.[2] However, for the competition, the MC Sr wants to make things a bit more exciting by letting you handle larger numbers as

[2] http://www21.in.tum.de/teaching/fpv/WS20/wettbewerb.html

well. These work much like in English or German:

| | |
|---|---|
| $10^6$ | unu miliono |
| $2 \cdot 10^6$ | du milionoj |
| $1 \cdot 10^9$ | unu miliardo |
| $2 \cdot 10^9$ | du miliardoj |
| $12,345,678$ | dek du milionoj tricent kvardek kvin mil sescent sepdek ok |

As you can see, the -j at the end indicates a plural ("two million**s**"). Starting from $10^{12}$, it becomes very systematic again:

| | | | |
|---|---|---|---|
| $10^6$ | miliono(j) | $10^9$ | miliardo(j) |
| $10^{12}$ | duiliono(j) | $10^{15}$ | duiliardo(j) |
| $10^{18}$ | triiliono(j) | $10^{21}$ | triiliardo(j) |
| ... | ... | ... | ... |
| $10^{6n}$ | $n$-iliono(j) | $10^{6n+3}$ | $n$-iliardo(j) |
| ... | ... | ... | ... |
| $10^{60}$ | dekiliono(j) | $10^{63}$ | dekiliardo(j) |

Your solution must be able to handle numbers between 0 and $10^{66} - 1$. Note that the online system will only test your program with inputs less than $10^6$, so do take care to test your program for bigger inputs yourself if you do not want to get eliminated shamefully for submitting an incorrect program.

The solution with the smallest number of tokens wins the competition. For more information about counting tokens and what library functions you are allowed to use, see the *Wettbewerb* website.

The complete solution (including self-written auxiliary functions, but excluding auxiliary functions already present in the template) must be submitted inside the comments `{-WETT-}` and `{-TTEW-}`, for example

```
{-WETT-}
helper :: Integer -> Integer
helper = ...

numberToEo :: Integer -> String
numberToEo  = ... helper ...
{-TTEW-}
```

> There is no swifter route to the corruption of thought than through the corruption of language.
>
> — George Orwell