Tutorium Funktionale Programmierung 2019

Part 4 -Enumerations, Type-Declaration, Type-Variables and Type-Class

VO - Part 3 (until slide 25)

Benedikt Dornauer, 27.10.2019

Type Bool

True && True || not True

(&&) :: Bool -> Bool -> Bool →

&&		Output
True	True	True
True	False	False
False	True	False
False	False	False

||		Output
True	True	True
True	False	True
False	True	True
False	False	False

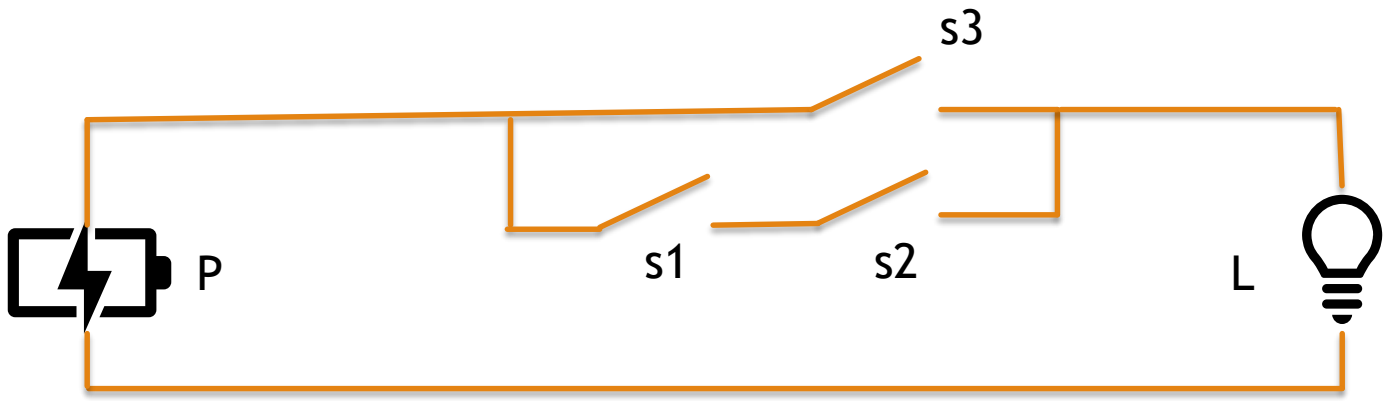
← **(||) :: Bool -> Bool -> Bool**

not :: Bool -> Bool →

not	Output
True	False
False	True

Exercise 4.1.: Bool

The electric circuit diagram is given in the following image.



s1, s2, s3 … switches
ON/OFF – True/False

L … light
P … power

s1	s2	s3	L
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

1. Complete the *table* to get an overview about the circuit diagram.
2. Find a function ***lightState :: Bool -> Bool -> Bool -> Bool*** that returns *True* if the light is ON and *False* if the light is Off.

Enumerations

"a **data type** where none of the
constructor functions have any arguments"

```
data Wert = One | Two | Three | Four deriving Show
```

- Give an example with a type we already know?
- What are the value constructors?
- What does *deriving Show* mean?

Ref. : https://en.wikibooks.org/wiki/Haskell/More_on_datatypes

Exercise 4.2: Enumerations

You want to create a new enumeration called **Grade**. This enumeration consists of constructors *Excellent*, *Average*, *Poor* and *NotPossible*. A function called *receiveGrade* exits. This function returns the grade according to the points in an exam.

The exam with n points is…

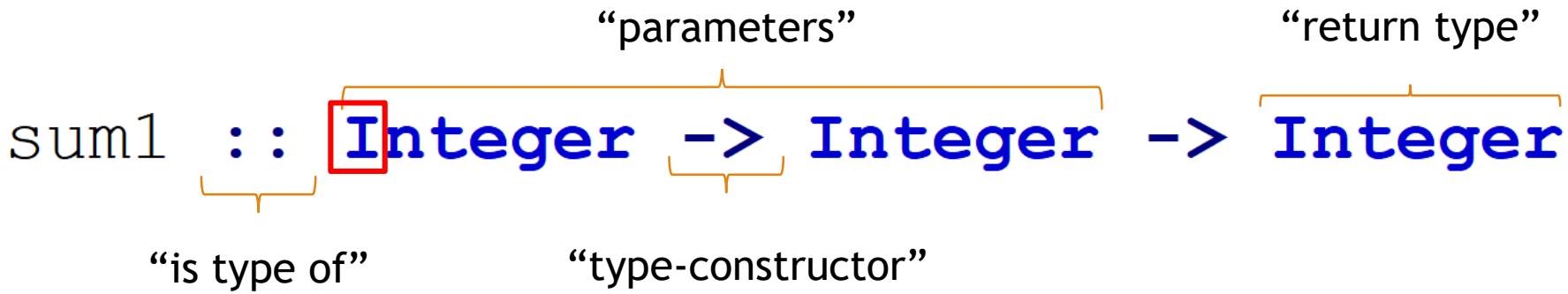
… Excellent if 100 >= n > 80

… Average if 80 >= n > 50

… Poor if 50 >= n >=0

… NotPossible if n < 0 || n > 100

Type-Declaration




```
sum1 a b = a + b
```

```
*Main> sum1 4.1 3

<interactive>:8:6: error:
    * No instance for (Fractional Integer)
        arising from the literal `4.1'
    * In the first argument of `sum1', namely `4.1'
      In the expression: sum1 4.1 3
      In an equation for `it': it = sum1 4.1 3
*Main> sum1 4 3
7
```

Type-Variables

```
myTakeFirst :: a -> b -> c -> a
myTakeFirst r s t = r
```

- *a*, *b*, *c* are the type variables
- for polymorphism
- begin with lowercase letter
- Informal: *a*, *b*, *c* can be of any type
- Informal: like a “placeholder”

Type-Class

```
sum2 :: (Num a) => a -> a -> a
```

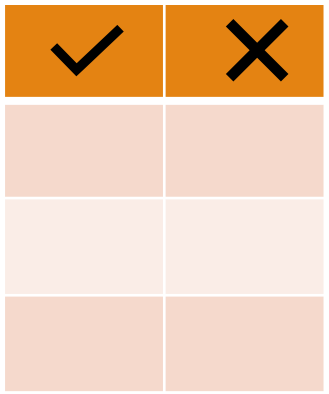
"class constraint"

```
sum2 a b = a + b
```

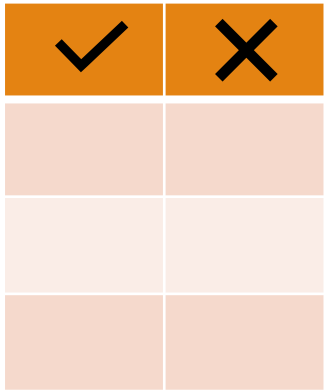
```
*Main> sum2 4.0 3
7.0
```

Exercise 4.3: Type-Declaration, Type-Variables and Type-Class

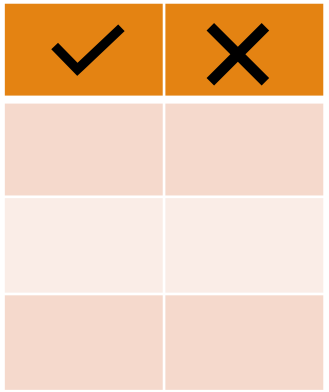
- **f :: a -> b -> b -> a**
  - f 2 True 5 ~> 2 ?
  - f 3 False True ~> 3 ?
  - f 5 5 4 ~> 3?



- **r :: a -> b -> Integer**
  - r 3 3 ~> 4 ?
  - r 3 3 ~> 4.0 ?
  - r 3 True ~> 4 ?



- **g :: (Eq a) => a -> a -> Bool**
  - g Excellent 3 ~> True ?
  - g Excellent Poor ~> True ?
  - g 3 3.0 ~> True ?



Questions? Need help? Feedback? etc.

- benedikt.dornauer@student.uibk.ac.at