

# INFORMATIK I

Tutorium 6 — 22. November 2024

True or False, that is the question

Name Last  
Universität Münster



L<sup>A</sup>T<sub>E</sub>X-Vorlage von  
Florian Sihler

# Übungsblatt 5

1

## Aufgabe 1

Beweisen Sie die De-Morgan'schen Gesetze mit Wahrheitstafeln.

Lösung 1:  $\overline{a \cap b} = \bar{a} \cup \bar{b}$

$a$	$b$	$a \cap b$	$\overline{(a \cap b)}$	$\bar{a}$	$\bar{b}$	$\bar{a} \cup \bar{b}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Spalten 4 und 7 stimmen überein. Daher sind die Ausdrücke gleich.

## Aufgabe 1

Beweisen Sie die De-Morgan'schen Gesetze mit Wahrheitstafeln.

Lösung 1:  $\overline{a \sqcup b} = \bar{a} \sqcap \bar{b}$

$a$	$b$	$a \sqcup b$	$\overline{(a \sqcup b)}$	$\bar{a}$	$\bar{b}$	$\bar{a} \sqcap \bar{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Spalten 4 und 7 stimmen überein. Daher sind die Ausdrücke gleich.

## Aufgabe 2: a)

Schreiben Sie eine Funktion für NAND.

## Lösung 2: a)

```
nand :: Bool -> Bool -> Bool  
nand a b = not (a && b)
```

## Aufgabe 2: b)

Schreiben Sie eine Funktion für XOR.

## Lösung 2: b)

```
xor :: Bool -> Bool -> Bool  
xor a b = (a && not b) || (not a && b)
```

## Aufgabe 3

Programmieren Sie zweistellige Boole'sche Funktion `quiet_atmosphere`, die zurückgeben soll, ob es ruhig genug ist, um der Vorlesung zu folgen.

### Lösung 3: mit UND und NICHT

```
quiet_atmosphere :: Bool -> Bool -> Bool
quiet_atmosphere quiet_neighbors construction_noise = ←
    quiet_neighbors && not construction_noise
```

## Aufgabe 3

Programmieren Sie zweistellige Boole'sche Funktion `quiet_atmosphere`, die zurückgeben soll, ob es ruhig genug ist, um der Vorlesung zu folgen.

### Lösung 3: mit ODER und NICHT

```
quiet_atmosphere :: Bool -> Bool -> Bool
quiet_atmosphere quiet_neighbors construction_noise =  $\neg$ 
    not (not quiet_neighbors || construction_noise)
```



## Aufgabe 4: a)

Programmieren Sie Algorithmus aus Aufgabe 2 von Übungsblatt 3 als rekursive Funktion. Welche Basisfälle lassen sich ableiten?

## Lösung 4: a)

```
euclid :: Integer -> Integer -> Integer
euclid m n
  | m < 0 || n < 0    = error "Eingabe_negativ!"
  | m == 0            = n                                -- Basisfall 1
  | n == 0            = m                                -- Basisfall 2
  | m > n              = euclid (m - n) n                -- rekursive Aufrufe
  | otherwise         = euclid m (n - m)
```

## Aufgabe 4: b)

Wieso terminiert Ihr Algorithmus?

## Lösung 4: b)

Im Fehlerfall oder bei Erreichen eines Basisfalls endet der Algorithmus.

In den rekursiven Aufrufen wird einer der Parameter um eine Zahl ungleich 0 reduziert. Dies kann nur endlich oft geschehen, bevor ein Basisfall mit Terminierung erreicht wird.

Warum ist es schon wieder 3 Uhr? xD

**Name Last**

Münster, 24. Januar 2025

[name.last@uni-muenster.de](mailto:name.last@uni-muenster.de)