

# Formal

Cornelius Diekmann

September 24, 2022

## Contents

<b>1</b>	<b>Schnelleinstieg Isabelle/HOL</b>	<b>1</b>
1.1	Typen . . . . .	1
1.2	Beweise . . . . .	1
1.3	Mehr Typen . . . . .	1
1.4	Funktionen . . . . .	2
1.5	Mengen . . . . .	2
<b>2</b>	<b>Disclaimer</b>	<b>2</b>
<b>3</b>	<b>Gesetz</b>	<b>3</b>
<b>4</b>	<b>Kant's Kategorischer Imperativ</b>	<b>4</b>
<b>5</b>	<b>Maxime</b>	<b>4</b>
5.0.1	Maximen Debugging . . . . .	6
5.0.2	Beispiel . . . . .	6
<b>6</b>	<b>Kategorischer Imperativ</b>	<b>7</b>
6.1	Allgemeines Gesetz Ableiten . . . . .	7
6.2	Implementierung Kategorischer Imperativ. . . . .	8
<b>7</b>	<b>Zahlenwelt Helper</b>	<b>9</b>
<b>8</b>	<b>Simulation</b>	<b>10</b>
<b>9</b>	<b>Gesetze</b>	<b>11</b>
9.1	Case Law Absolut . . . . .	11
9.2	Case Law Relativ . . . . .	11
<b>10</b>	<b>Beispiel: Zahlenwelt</b>	<b>11</b>
10.1	Handlungen . . . . .	12
10.2	Setup . . . . .	12
10.3	Alice erzeugt 5 Wohlstand für sich. . . . .	13

10.4 Kleine Änderung in der Maxime . . . . .	14
10.5 Maxime für Globales Optimum . . . . .	14
10.6 Alice stiehlt 5 . . . . .	15
10.7 Schenken . . . . .	16
10.8 Ungültige Maxime . . . . .	17
<b>11 Einkommensteuergesetzgebung</b>	<b>17</b>
<b>12 Beispiel: Steuern</b>	<b>19</b>

# 1 Schnelleinstieg Isabelle/HOL

## 1.1 Typen

Typen werden per `::` annotiert. Beispielsweise sagt `3::nat`, dass 3 eine natürliche Zahl (*nat*) ist.

## 1.2 Beweise

Die besondere Fähigkeit im Beweisassistent Isabelle/HOL liegt darin, maschinengeprüfte Beweise zu machen.

Beispiel:

**lemma** $\langle 3 = 2 + 1 \rangle$

In der PDFversion wird der eigentliche Beweis ausgelassen. Aber keine Sorge, der Computer hat den Beweis überprüft. Würde der Beweis nicht gelten, würde das PDF garnicht compilieren.

Ich wurde schon für meine furchtbaren Beweise zitiert. Ist also ganz gut, dass wir nur Ergebnisse im PDF sehen und der eigentliche Beweis ausgelassen ist. Am besten kann man Beweise sowieso im Isabelle Editor anschauen und nicht im PDF.

## 1.3 Mehr Typen

Jeder Typ der mit einem einfachen Anführungszeichen anfängt ist ein polymorpher Typ. Beispiel: `'a` oder `'α`. So ein Typ ist praktisch ein generischer Typ, welcher durch jeden anderen Typen instanziiert werden kann.

Beispielsweise steht `'nat` für einen beliebigen Typen, während `nat` der konkrete Typ der natürlichen Zahlen ist.

Wenn wir nun `3::'a` schreiben handelt es sich nur um das generische Numeral 3. Das ist so generisch, dass z.B. noch nicht einmal die Plusoperation darauf definiert ist. Im Gegensatz dazu ist `3::nat` die natürliche Zahl 3, mit allen wohlbekannten Rechenoperationen. Im Beweis obigen **lemma** $\langle 3 = 2 + 1 \rangle$  hat Isabelle die Typen automatisch inferiert.

## 1.4 Funktionen

Beispiel: Eine Funktionen welche eine natürliche Zahl nimmt und eine natürliche Zahl zurück gibt ( $nat \Rightarrow nat$ ):

```
fun beispielfunktion :: nat  $\Rightarrow$  nat where  
  beispielfunktion n = n + 10
```

Funktionsaufrufe funktionieren ohne Klammern.

```
lemma  $\langle$ beispielfunktion 32 = 42 $\rangle$ 
```

Funktionen sind gecurried. Hier ist eine Funktion welche 2 natürliche Zahlen nimmt und eine natürliche Zahl zurück gibt ( $nat \Rightarrow nat \Rightarrow nat$ ):

```
fun addieren :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat where  
  addieren a b = a + b
```

```
lemma  $\langle$ addieren 32 10 = 42 $\rangle$ 
```

Currying bedeutet auch, wenn wir *addieren* nur mit einem Argument aufrufen (welches eine natürliche Zahl *nat* sein muss), dass wir eine Funktion zurückbekommen, die noch das zweite Argument erwartet, bevor sie das Ergebnis zurückgeben kann.

Beispiel: *addieren 10 :: nat  $\Rightarrow$  nat*

Zufälligerweise ist *addieren 10* equivalent zu *beispielfunktion*:

```
lemma  $\langle$ addieren 10 = beispielfunktion $\rangle$ 
```

Zusätzlich lassen sich Funktionen im Lambda Calculus darstellen. Beispiel:

```
lemma  $(\lambda n::nat. n+10)$  3 = 13
```

```
lemma beispielfunktion =  $(\lambda n. n+10)$ 
```

## 1.5 Mengen

Mengen funktionieren wie normale mathematische Mengen.

Beispiel. Die Menge der geraden Zahlen:

```
lemma  $\langle \{0,2,4,6,8,10,12\} \subseteq \{n::int. n \bmod 2 = 0\} \rangle$ 
```

## 2 Disclaimer

Ich habe

- kein Ahnung von Philosophie.

- keine Ahnung von Recht und Jura.
- und schon gar keine Ahnung von Strafrecht oder Steuerrecht.

Und in dieser Session werden ich all das zusammenwerfen.

Cheers!

### 3 Gesetz

Definiert einen Datentyp um Gesetzestext zu modellieren.

```
datatype 'a tatbestand = Tatbestand <'a>
```

```
datatype 'a rechtsfolge = Rechtsfolge <'a>
```

```
datatype ('a, 'b) rechtsnorm = Rechtsnorm <'a tatbestand> <'b rechtsfolge>
```

```
datatype 'p prg = Paragraph <'p> (§)
```

```
datatype ('p, 'a, 'b) gesetz = Gesetz <('p prg × ('a, 'b) rechtsnorm) set>
```

Beispiel, von <https://de.wikipedia.org/wiki/Rechtsfolge>:

```
value <Gesetz {
  (§ "823 BGB",
    Rechtsnorm
      (Tatbestand "Wer vorsaeztzlich oder fahrlaessig das Leben, den Koerper, die Gesundheit, (...),
        das Eigentum oder (...) eines anderen widerrechtlich verletzt,"))
      (Rechtsfolge "ist dem anderen zum Ersatz des daraus entstehenden Schadens verpflichtet.")
  ),
  (§ "985 BGB",
    Rechtsnorm
      (Tatbestand "Der Eigentuemmer einer Sache kann von dem Besitzer")
      (Rechtsfolge "die Herausgabe der Sache verlangen")
  ),
  (§ "303 StGB",
    Rechtsnorm
      (Tatbestand "Wer rechtswidrig eine fremde Sache beschaedigt oder zerstoert,"))
      (Rechtsfolge "wird mit Freiheitsstrafe bis zu zwei Jahren oder mit Geldstrafe bestraft.")
  )
}>
```

```
fun neuer-paragraph :: <(nat, 'a, 'b) gesetz ⇒ nat prg> where
  <neuer-paragraph (Gesetz G) = § ((max-paragraph (fst ' G)) + 1)>
```

Fügt eine Rechtsnorm als neuen Paragraphen hinzu:

```
fun hinzufuegen :: <('a,'b) rechtsnorm ⇒ (nat,'a,'b) gesetz ⇒ (nat,'a,'b) gesetz> where
```

$\langle \text{hinzufuegen } rn \text{ (Gesetz } G) =$   
 $(\text{if } rn \in (snd \text{ ' } G) \text{ then Gesetz } G \text{ else Gesetz (insert (neuer-paragraph (Gesetz } G), rn) G))) \rangle$

Moelliert ob eine Handlung ausgeführt werden muss, darf, kann, nicht muss:

**datatype** *sollensanordnung* = *Gebot* | *Verbot* | *Erlaubnis* | *Freistellung*

Beispiel:

**lemma**  $\langle \text{hinzufuegen}$   
 $(\text{Rechtsnorm (Tatbestand "tb2") (Rechtsfolge Verbot)})$   
 $(\text{Gesetz } \{(\S 1, (\text{Rechtsnorm (Tatbestand "tb1") (Rechtsfolge Erlaubnis)}))\}) =$   
 $\text{Gesetz}$   
 $\{(\S 2, \text{Rechtsnorm (Tatbestand "tb2") (Rechtsfolge Verbot)}),$   
 $(\S 1, \text{Rechtsnorm (Tatbestand "tb1") (Rechtsfolge Erlaubnis)})\} \rangle$

## 4 Kant's Kategorischer Imperativ



*Immanuel Kant*

„Handle nur nach derjenigen *Maxime*, durch die du zugleich wollen kannst, dass sie ein allgemeines Gesetz werde.“

[https://de.wikipedia.org/wiki/Kategorischer\\_Impervativ](https://de.wikipedia.org/wiki/Kategorischer_Impervativ)

Meine persönliche, etwas utilitaristische, Interpretation.

## 5 Maxime

Modell einer *Maxime*: Eine *Maxime* in diesem Modell beschreibt ob eine Handlung in einer gegebenen Welt gut ist.

Faktisch ist eine *Maxime*

- '*person*': die handelnde Person, i.e., *ich*.
- '*world handlung*': die zu betrachtende Handlung.

- *bool*: Das Ergebnis der Betrachtung. *True* = Gut; *False* = Schlecht.

Wir brauchen sowohl die *'world handlung* als auch die handelnde *'person*, da es einen großen Unterschied machen kann ob ich selber handel, ob ich Betroffener einer fremden Handlung bin, oder nur Außenstehender.

**datatype** (*'person*, *'world*) *maxime* = *Maxime*  $\langle 'person \Rightarrow 'world\ handlung \Rightarrow bool \rangle$

Beispiel

**definition** *maxime-mir-ist-alles-recht* ::  $\langle ('person, 'world)\ maxime \rangle$  **where**  
 $\langle maxime-mir-ist-alles-recht \equiv Maxime\ (\lambda\ -. True) \rangle$

Um eine Handlung gegen eine Maxime zu testen fragen wir uns:

- Was wenn jeder so handeln würde?
- Was wenn jeder diese Maxime hätte? Bsp: stehlen und bestohlen werden.

**definition** *bevoelkerung* ::  $\langle 'person\ set \rangle$  **where**  $\langle bevoelkerung \equiv UNIV \rangle$

**definition** *wenn-jeder-so-handelt*

::  $\langle 'world \Rightarrow ('person, 'world)\ handlungF \Rightarrow ('world\ handlung)\ set \rangle$

**where**

$\langle wenn-jeder-so-handelt\ welt\ handlung \equiv$   
 $(\lambda handelnde-person.\ handeln\ handelnde-person\ welt\ handlung)\ 'bevoelkerung \rangle$

**fun** *was-wenn-jeder-so-handelt-aus-sicht-von*

::  $\langle 'world \Rightarrow ('person, 'world)\ handlungF \Rightarrow ('person, 'world)\ maxime \Rightarrow 'person \Rightarrow bool \rangle$

**where**

$\langle was-wenn-jeder-so-handelt-aus-sicht-von\ welt\ handlung\ (Maxime\ m)\ betroffene-person =$   
 $(\forall h \in wenn-jeder-so-handelt\ welt\ handlung.\ m\ betroffene-person\ h) \rangle$

**definition** *teste-maxime* ::

$\langle 'world \Rightarrow ('person, 'world)\ handlungF \Rightarrow ('person, 'world)\ maxime \Rightarrow bool \rangle$  **where**

$\langle teste-maxime\ welt\ handlung\ maxime \equiv$

$\forall p \in bevoelkerung.\ was-wenn-jeder-so-handelt-aus-sicht-von\ welt\ handlung\ maxime\ p \rangle$

Faktisch bedeutet diese Definition, wir bilden das Kreuzprodukt Bevölkerung x Bevölkerung, wobei jeder einmal als handelnde Person auftritt und einmal als betroffene Person.

**lemma** *teste-maxime-unfold*:

$\langle teste-maxime\ welt\ handlung\ (Maxime\ m) =$   
 $(\forall p1 \in bevoelkerung.\ \forall p2 \in bevoelkerung.\ m\ p1\ (handeln\ p2\ welt\ handlung)) \rangle$

**lemma**  $\langle teste-maxime\ welt\ handlung\ (Maxime\ m) =$

$(\forall (p1,p2) \in bevoelkerung \times bevoelkerung.\ m\ p1\ (handeln\ p2\ welt\ handlung)) \rangle$

Hier schlägt das Programmiererherz höher: Wenn *'person* aufzählbar ist haben wir ausführbaren Code: *teste-maxime* = *teste-maxime-exhaust* *enum-class.enum* wobei *teste-maxime-exhaust* implementiert ist als *teste-maxime-exhaust* *bevoelk* *welt* *handlung* *maxime*  $\equiv case\ maxime\ of\ Maxime\ m \Rightarrow list-all\ (\lambda(p, x).\ m\ p\ (handeln\ x\ welt\ handlung))\ (List.product\ bevoelk\ bevoelk)$ .

### 5.0.1 Maximen Debugging

Der folgende Datentyp modelliert ein Beispiel in welcher Konstellation eine gegebene Maxime verletzt ist:

```
datatype 'person opfer = Opfer 'person
datatype 'person taeter = Taeter 'person
datatype ('person, 'world) verletzte-maxime =
  VerletzteMaxime
  <'person opfer> — verletzt für; das Opfer
  <'person taeter> — handelnde Person; der Täter
  <'world handlung> — Die verletzende Handlung
```

Die folgende Funktion liefert alle Gegebenheiten welche eine Maxime verletzen:

```
fun debug-maxime
  :: ('world  $\Rightarrow$  'printable-world)  $\Rightarrow$  'world  $\Rightarrow$ 
    ('person, 'world) handlungF  $\Rightarrow$  ('person, 'world) maxime
     $\Rightarrow$  (('person, 'printable-world) verletzte-maxime) set
where
  debug-maxime print-world welt handlung (Maxime m) =
    { VerletzteMaxime
      (Opfer p1) (Taeter p2)
      (map-handlung print-world (handeln p2 welt handlung)) | p1 p2.
         $\neg m$  p1 (handeln p2 welt handlung)}
```

Es gibt genau dann keine Beispiele für Verletzungen, wenn die Maxime erfüllt ist:

```
lemma debug-maxime print-world welt handlung maxime = {}  $\longleftrightarrow$  teste-maxime welt handlung maxime
```

### 5.0.2 Beispiel

Beispiel: Die Welt sei nur eine Zahl und die zu betrachtende Handlung sei, dass wir diese Zahl erhöhen. Die Mir-ist-alles-Recht Maxime ist hier erfüllt:

```
lemma <teste-maxime
  (42::nat)
  (HandlungF ( $\lambda$ (person::person) welt. welt + 1))
  maxime-mir-ist-alles-recht>
```

Beispiel: Die Welt ist modelliert als eine Abbildung von Person auf Besitz. Die Maxime sagt, dass Leute immer mehr oder gleich viel wollen, aber nie etwas verlieren wollen. In einer Welt in der keiner etwas hat, erfüllt die Handlung jemanden 3 zu geben die Maxime.

```
lemma <teste-maxime
  [Alice  $\mapsto$  (0::nat), Bob  $\mapsto$  0, Carol  $\mapsto$  0, Eve  $\mapsto$  0]
  (HandlungF ( $\lambda$ person welt. welt(person  $\mapsto$  3)))
  (Maxime ( $\lambda$ person handlung.
    (the ((vorher handlung) person))  $\leq$  (the ((nachher handlung) person))))>
lemma <debug-maxime show-map
  [Alice  $\mapsto$  (0::nat), Bob  $\mapsto$  0, Carol  $\mapsto$  0, Eve  $\mapsto$  0]
```

$$\begin{aligned}
& (HandlungF (\lambda person\ welt.\ welt(person \mapsto 3))) \\
& (Maxime (\lambda person\ handlung.\ \\
& \quad (the ((vorher\ handlung)\ person)) \leq (the ((nachher\ handlung)\ person)))) \\
& = \{\} \rangle
\end{aligned}$$

Wenn nun *Bob* allerdings bereits 4 hat, würde die obige Handlung ein Verlust für ihn bedeuten und die *Maxime* ist nicht erfüllt.

$$\begin{aligned}
\textbf{lemma } \langle \neg \text{ teste-maxime} \\
& [Alice \mapsto (0::nat), Bob \mapsto 4, Carol \mapsto 0, Eve \mapsto 0] \\
& (HandlungF (\lambda person\ welt.\ welt(person \mapsto 3))) \\
& (Maxime (\lambda person\ handlung.\ \\
& \quad (the ((vorher\ handlung)\ person)) \leq (the ((nachher\ handlung)\ person)))) \rangle \\
\textbf{lemma } \langle \text{ debug-maxime show-map} \\
& [Alice \mapsto (0::nat), Bob \mapsto 4, Carol \mapsto 0, Eve \mapsto 0] \\
& (HandlungF (\lambda person\ welt.\ welt(person \mapsto 3))) \\
& (Maxime (\lambda person\ handlung.\ \\
& \quad (the ((vorher\ handlung)\ person)) \leq (the ((nachher\ handlung)\ person)))) \\
& = \{ VerletzteMaxime (Opfer Bob) (Taeter Bob) \\
& \quad (Handlung [(Alice, 0), (Bob, 4), (Carol, 0), (Eve, 0)] \\
& \quad [(Alice, 0), (Bob, 3), (Carol, 0), (Eve, 0)]) \} \rangle
\end{aligned}$$

## 6 Kategorischer Imperativ

### 6.1 Allgemeines Gesetz Ableiten

Wir wollen implementieren:

„Handle nur nach derjenigen *Maxime*, durch die du zugleich wollen kannst, dass sie ein **allgemeines Gesetz** werde.“

Für eine gebene Welt haben wir schon eine Handlung nach einer *Maxime* untersucht: *teste-maxime*

Das Ergebnis sagt uns ob diese Handlung gut oder schlecht ist. Basierend darauf müssen wir nun ein allgemeines Gesetz ableiten.

Ich habe keine Ahnung wie das genau funktionieren soll, deswegen schreibe ich einfach nur in einer Typsignatur auf, was zu tun ist:

Gegeben:

- *'world handlung*: Die Handlung
- *sollensanordnung*: Das Ergebnis der moralischen Bewertung, ob die Handlung gut/schlecht.

Gesucht:



- (*'a*, *'b*) *rechtsnorm*: ein allgemeines Gesetz

**type-synonym** (*'world*, *'a*, *'b*) *allgemeines-gesetz-ableiten* =  
 $\langle 'world\ handlung \Rightarrow sollensanordnung \Rightarrow ('a, 'b)\ rechtsnorm \rangle$

Soviel vorweg: Nur aus einer von außen betrachteten Handlung und einer Entscheidung ob diese Handlung ausgeführt werden soll wird es schwer ein allgemeines Gesetz abzuleiten.

## 6.2 Implementierung Kategorischer Imperativ.

Und nun werfen wir alles zusammen:

„Handle nur nach derjenigen *Maxime*, durch die du zugleich wollen kannst, dass sie ein allgemeines Gesetz werde.“

Eingabe:

- *'person*: handelnde Person
- *'world*: Die Welt in ihrem aktuellen Zustand
- (*'person*, *'world*) *handlungF*: Eine mögliche Handlung, über die wir entscheiden wollen ob wir sie ausführen sollten.
- (*'person*, *'world*) *maxime*: Persönliche Ethik.
- (*'world*, *'a*, *'b*) *allgemeines-gesetz-ableiten*: wenn man keinen Plan hat wie man sowas implementiert, einfach als Eingabe annehmen.
- (*nat*, *'a*, *'b*) *gesetz*: Initiales allgemeines Gesetz (normalerweise am Anfang leer).

Ausgabe: *sollensanordnung*: Sollen wir die Handlung ausführen? (*nat*, *'a*, *'b*) *gesetz*: Soll das allgemeine Gesetz entsprechend angepasst werden?

**definition** *kategorischer-imperativ* ::

```

<'person =>
'world =>
('person, 'world) handlungF =>
('person, 'world) maxime =>
('world, 'a, 'b) allgemeines-gesetz-ableiten =>
(nat, 'a, 'b) gesetz
=> (sollensanordnung × (nat, 'a, 'b) gesetz)>

```

**where**

```

<kategorischer-imperativ ich welt handlung maxime gesetz-ableiten gesetz ≡
let soll-handeln = if teste-maxime welt handlung maxime
then
    Erlaubnis
else
    Verbot in
(

```

*soll-handeln,*  
*hinzufuegen (gesetz-ableiten (handeln ich welt handlung) soll-handeln) gesetz*  
 ) $\rangle$

## 7 Zahlenwelt Helper

Wir werden Beispiele betrachten, in denen wir Welten modellieren, in denen jeder Person eine Zahl zugewiesen wird:  $person \Rightarrow int$ . Diese Zahl kann zum Beispiel der Besitz oder Wohlstand einer Person sein, oder das Einkommen. Wobei Gesamtbesitz und Einkommen über einen kurzen Zeitraum recht unterschiedliche Sachen modellieren.

Hier sind einige Hilfsfunktionen um mit  $person \Rightarrow int$  allgemein zu arbeiten.

Default: Standardmäßig hat jede Person 0:

**definition** *DEFAULT* ::  $person \Rightarrow int$  **where**  
*DEFAULT*  $\equiv \lambda p. 0$

Beispiel:

**lemma**  $\langle (DEFAULT(Alice:=8, Bob:=3, Eve:= 5)) Bob = 3 \rangle$

Beispiel mit fancy Syntax:

**lemma**  $\langle \clubsuit[Alice:=8, Bob:=3, Eve:= 5] Bob = 3 \rangle$

**lemma**  $\langle show\_fun \clubsuit[Alice := 4, Carol := 4] = [(Alice, 4), (Bob, 0), (Carol, 4), (Eve, 0)] \rangle$

**lemma**  $\langle show\_num\_fun \clubsuit[Alice := 4, Carol := 4] = [(Alice, 4), (Carol, 4)] \rangle$

**abbreviation** *num-fun-add-syntax* (- '(- += -')) **where**

$f(p += n) \equiv (f(p := (f p) + n))$

**abbreviation** *num-fun-minus-syntax* (- '(- -= -')) **where**

$f(p -= n) \equiv (f(p := (f p) - n))$

**lemma**  $\langle (\clubsuit[Alice:=8, Bob:=3, Eve:= 5])(Bob += 4) Bob = 7 \rangle$

**lemma**  $\langle (\clubsuit[Alice:=8, Bob:=3, Eve:= 5])(Bob -= 4) Bob = -1 \rangle$

**lemma** **fixes** *n*:: *int* **shows**  $f(p += n)(p -= n) = f$

## 8 Simulation

Gegeben eine handelnde Person und eine Maxime, wir wollen simulieren was für ein allgemeines Gesetz abgeleitet werden könnte.

**datatype** ('person, 'world, 'a, 'b) *simulation-constants* = *SimConsts*  
     'person — handelnde Person  
     ('person, 'world) *maxime*  
     ('world, 'a, 'b) *allgemeines-gesetz-ableiten*

...

... Die Funktion *simulateOne* nimmt eine Konfiguration ('person, 'world, 'a, 'b) *simulation-constants*, eine Anzahl an Iterationen die durchgeführt werden sollen, eine Handlung, eine Initialwelt, ein Initialgesetz, und gibt das daraus resultierende Gesetz nach so vielen Iterationen zurück.

Beispiel: Wir nehmen die mir-ist-alles-egal Maxime. Wir leiten ein allgemeines Gesetz ab indem wir einfach nur die Handlung wörtlich ins Gesetz übernehmen. Wir machen  $10::'a$  Iterationen. Die Welt ist nur eine Zahl und die initiale Welt sei  $32::'a$ . Die Handlung ist es diese Zahl um Eins zu erhöhen, Das Ergebnis der Simulation ist dann, dass wir einfach von  $32::'a$  bis  $42::'a$  zählen.

**lemma** <*simulateOne*  
     (*SimConsts* ()) (*Maxime* ( $\lambda$ - . *True*)) ( $\lambda$ h s. *Rechtsnorm* (*Tatbestand* h) (*Rechtsfolge* "count"))  
     10 (*HandlungF* ( $\lambda$ p n. *Suc* n))  
     32  
     (*Gesetz* {}) =  
     *Gesetz*  
     { (§ 10, *Rechtsnorm* (*Tatbestand* (*Handlung* 41 42)) (*Rechtsfolge* "count")),  
     ( § 9, *Rechtsnorm* (*Tatbestand* (*Handlung* 40 41)) (*Rechtsfolge* "count")),  
     ( § 8, *Rechtsnorm* (*Tatbestand* (*Handlung* 39 40)) (*Rechtsfolge* "count")),  
     ( § 7, *Rechtsnorm* (*Tatbestand* (*Handlung* 38 39)) (*Rechtsfolge* "count")),  
     ( § 6, *Rechtsnorm* (*Tatbestand* (*Handlung* 37 38)) (*Rechtsfolge* "count")),  
     ( § 5, *Rechtsnorm* (*Tatbestand* (*Handlung* 36 37)) (*Rechtsfolge* "count")),  
     ( § 4, *Rechtsnorm* (*Tatbestand* (*Handlung* 35 36)) (*Rechtsfolge* "count")),  
     ( § 3, *Rechtsnorm* (*Tatbestand* (*Handlung* 34 35)) (*Rechtsfolge* "count")),  
     ( § 2, *Rechtsnorm* (*Tatbestand* (*Handlung* 33 34)) (*Rechtsfolge* "count")),  
     ( § 1, *Rechtsnorm* (*Tatbestand* (*Handlung* 32 33)) (*Rechtsfolge* "count")) } >

Eine Iteration der Simulation liefert genau einen Paragraphen im Gesetz:

**lemma** < $\exists$  tb rf.  
     *simulateOne*  
     (*SimConsts* person maxime gesetz-ableiten)  
     1 handlungF  
     initialwelt  
     (*Gesetz* {})  
     = *Gesetz* { (§ 1, *Rechtsnorm* (*Tatbestand* tb) (*Rechtsfolge* rf)) } >

## 9 Gesetze

Wir implementieren Strategien um  $(\text{'world}, \text{'a}, \text{'b})$  *allgemeines-gesetz-ableiten* zu implementieren.

### 9.1 Case Law Absolut

Gesetz beschreibt: wenn (vorher, nachher) dann Erlaubt/Verboten, wobei vorher/nachher die Welt beschreiben. Paragraphen sind einfache natürliche Zahlen.

**type-synonym**  $\text{'world case-law} = (\text{nat}, (\text{'world} \times \text{'world}), \text{sollensanordnung}) \text{ gesetz}$

Überträgt einen Tatbestand wörtlich ins Gesetz. Nicht sehr allgemein.

**definition** *case-law-ableiten-absolut*

$:: (\text{'world}, (\text{'world} \times \text{'world}), \text{sollensanordnung}) \text{ allgemeines-gesetz-ableiten}$

**where**

$\text{case-law-ableiten-absolut handlung sollensanordnung} =$   
 $\text{Rechtsnorm}$   
 $(\text{Tatbestand} (\text{vorher handlung}, \text{nachher handlung}))$   
 $(\text{Rechtsfolge sollensanordnung})$

**definition** *printable-case-law-ableiten-absolut*

$:: (\text{'world} \Rightarrow \text{'printable-world}) \Rightarrow$   
 $(\text{'world}, (\text{'printable-world} \times \text{'printable-world}), \text{sollensanordnung}) \text{ allgemeines-gesetz-ableiten}$

**where**

$\text{printable-case-law-ableiten-absolut print-world h} \equiv$   
 $\text{case-law-ableiten-absolut} (\text{map-handlung print-world h})$

### 9.2 Case Law Relativ

Case Law etwas besser, wir zeigen nur die Änderungen der Welt.

**fun** *case-law-ableiten-relativ*

$:: (\text{'world handlung} \Rightarrow ((\text{'person}, \text{'etwas}) \text{ aenderung}) \text{ list})$   
 $\Rightarrow (\text{'world}, ((\text{'person}, \text{'etwas}) \text{ aenderung}) \text{ list}, \text{sollensanordnung})$   
 $\text{allgemeines-gesetz-ableiten}$

**where**

$\text{case-law-ableiten-relativ delta handlung erlaubt} =$   
 $\text{Rechtsnorm} (\text{Tatbestand} (\text{delta handlung})) (\text{Rechtsfolge erlaubt})$

## 10 Beispiel: Zahlenwelt

Wir nehmen an, die Welt lässt sich durch eine Zahl darstellen, die den Besitz einer Person modelliert. Der Besitz ist als ganze Zahl *int* modelliert und kann auch beliebig negativ werden.

**datatype** *zahlenwelt* = *Zahlenwelt*

$\text{person} \Rightarrow \text{int} \text{ — besitz: Besitz jeder Person.}$

```
fun gesamtbesitz :: zahlenwelt  $\Rightarrow$  int where
  gesamtbesitz (Zahlenwelt besitz) = sum-list (map besitz Enum.enum)
```

```
lemma gesamtbesitz (Zahlenwelt  $\clubsuit$ [Alice := 4, Carol := 8]) = 12
```

```
lemma gesamtbesitz (Zahlenwelt  $\clubsuit$ [Alice := 4, Carol := 4]) = 8
```

```
fun meins :: person  $\Rightarrow$  zahlenwelt  $\Rightarrow$  int where
  meins p (Zahlenwelt besitz) = besitz p
```

```
lemma meins Carol (Zahlenwelt  $\clubsuit$ [Alice := 8, Carol := 4]) = 4
```

## 10.1 Handlungen

Die folgende Handlung erschafft neuen Besitz aus dem Nichts:

```
fun erschaffen :: nat  $\Rightarrow$  person  $\Rightarrow$  zahlenwelt  $\Rightarrow$  zahlenwelt where
  erschaffen i p (Zahlenwelt besitz) = Zahlenwelt (besitz(p += int i))
```

```
fun stehlen :: int  $\Rightarrow$  person  $\Rightarrow$  person  $\Rightarrow$  zahlenwelt  $\Rightarrow$  zahlenwelt where
  stehlen beute opfer dieb (Zahlenwelt besitz) =
    Zahlenwelt (besitz(opfer -= beute)(dieb += beute))
```

```
fun schenken :: int  $\Rightarrow$  person  $\Rightarrow$  person  $\Rightarrow$  zahlenwelt  $\Rightarrow$  zahlenwelt where
  schenken betrag empfaenger schenker (Zahlenwelt besitz) =
    Zahlenwelt (besitz(schenker -= betrag)(empfaenger += betrag))
```

Da wir ganze Zahlen verwenden und der Besitz auch beliebig negativ werden kann, ist Stehlen äquivalent dazu einen negativen Betrag zu verschenken:

```
lemma stehlen-ist-schenken: stehlen i = schenken ( $-i$ )
```

Das Modell ist nicht ganz perfekt, .... Aber passt schon um damit zu spielen.

## 10.2 Setup

```
definition initialwelt  $\equiv$  Zahlenwelt  $\clubsuit$ [Alice := 5, Bob := 10]
```

Wir nehmen an unsere handelnde Person ist *Alice*.

```
definition beispiel-case-law-absolut maxime handlung  $\equiv$ 
  simulateOne
    (SimConsts
      Alice
      maxime
      (printable-case-law-ableiten-absolut show-zahlenwelt))
  10 handlung initialwelt (Gesetz {})
```

```
definition beispiel-case-law-relativ maxime handlung  $\equiv$ 
  simulateOne
    (SimConsts
```

```

Alice
maxime
(case-law-ableiten-relativ delta-zahlenwelt))
20 handlung initiahwelt (Gesetz {})

```

### 10.3 Alice erzeugt 5 Wohlstand für sich.

Wir definieren eine Maxime die besagt, dass sich der Besitz einer Person nicht verringern darf:

```

fun individueller-fortschritt :: person  $\Rightarrow$  zahlenwelt handlung  $\Rightarrow$  bool where
  individueller-fortschritt p (Handlung vor nach)  $\longleftrightarrow$  (meins p vor)  $\leq$  (meins p nach)
definition maxime-zahlenfortschritt :: (person, zahlenwelt) maxime where
  maxime-zahlenfortschritt  $\equiv$  Maxime ( $\lambda$ ich. individueller-fortschritt ich)

```

Alice kann beliebig oft 5 Wohlstand für sich selbst erschaffen. Das entstehende Gesetz ist nicht sehr gut, da es einfach jedes Mal einen Snapshot der Welt aufschreibt und nicht sehr generisch ist.

```

lemma <beispiel-case-law-absolut maxime-zahlenfortschritt (HandlungF (erschaffen 5))
=
Gesetz
{ (§ 10,
  Rechtsnorm (Tatbestand ([ (Alice, 50), (Bob, 10)], [(Alice, 55), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 9,
  Rechtsnorm (Tatbestand ([ (Alice, 45), (Bob, 10)], [(Alice, 50), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 8,
  Rechtsnorm (Tatbestand ([ (Alice, 40), (Bob, 10)], [(Alice, 45), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 7,
  Rechtsnorm (Tatbestand ([ (Alice, 35), (Bob, 10)], [(Alice, 40), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 6,
  Rechtsnorm (Tatbestand ([ (Alice, 30), (Bob, 10)], [(Alice, 35), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 5,
  Rechtsnorm (Tatbestand ([ (Alice, 25), (Bob, 10)], [(Alice, 30), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 4,
  Rechtsnorm (Tatbestand ([ (Alice, 20), (Bob, 10)], [(Alice, 25), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 3,
  Rechtsnorm (Tatbestand ([ (Alice, 15), (Bob, 10)], [(Alice, 20), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 2,
  Rechtsnorm (Tatbestand ([ (Alice, 10), (Bob, 10)], [(Alice, 15), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (§ 1,
  Rechtsnorm (Tatbestand ([ (Alice, 5), (Bob, 10)], [(Alice, 10), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)) }

```

›

Die gleiche Handlung, wir schreiben aber nur die Änderung der Welt ins Gesetz:

**lemma**  $\langle$  *beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*erschaffen* 5)) =  
*Gesetz*  
 $\{(\S\ 1, \text{Rechtsnorm } (\text{Tatbestand } [\text{Gewinnt Alice } 5]) (\text{Rechtsfolge Erlaubnis}))\}$   $\rangle$

## 10.4 Kleine Änderung in der Maxime

In der Maxime *individueller-fortschritt* hatten wir *meins p vor*  $\leq$  *meins p nach*. Was wenn wir nun echten Fortschritt fordern: *meins p vor*  $<$  *meins p nach*.

**fun** *individueller-strikter-fortschritt* :: *person*  $\Rightarrow$  *zahlenwelt handlung*  $\Rightarrow$  *bool* **where**  
*individueller-strikter-fortschritt* *p* (*Handlung vor nach*)  $\longleftrightarrow$  (*meins p vor*)  $<$  (*meins p nach*)

Nun ist es *Alice* verboten Wohlstand für sich selbst zu erzeugen.

**lemma**  $\langle$  *beispiel-case-law-relativ*  
*(Maxime ( $\lambda$ ich. individueller-strikter-fortschritt ich))*  
*(HandlungF* (*erschaffen* 5)) =  
*Gesetz*  $\{(\S\ 1, \text{Rechtsnorm } (\text{Tatbestand } [\text{Gewinnt Alice } 5]) (\text{Rechtsfolge Verbot}))\}$   $\rangle$

Der Grund ist, dass der Rest der Bevölkerung keine *strikte* Erhöhung des eigenen Wohlstands erlebt. Effektiv führt diese Maxime zu einem Gesetz, welches es einem Individuum nicht erlaubt mehr Besitz zu erschaffen, obwohl niemand dadurch einen Nachteil hat. Diese Maxime kann meiner Meinung nach nicht gewollt sein.

Beispielsweise ist *Bob* das Opfer wenn *Alice* sich 5 Wohlstand erschafft, aber *Bob's* Wohlstand sich nicht erhöht:

**lemma**  $\langle$  *VerletzteMaxime* (*Opfer Bob*) (*Taeter Alice*)  
*(Handlung* [(*Alice*, 5), (*Bob*, 10)] [(*Alice*, 10), (*Bob*, 10)])  
 $\in$  *debug-maxime show-zahlenwelt initialwelt*  
*(HandlungF* (*erschaffen* 5)) (*Maxime* ( $\lambda$ ich. individueller-strikter-fortschritt ich))  $\rangle$

## 10.5 Maxime für Globales Optimum

Wir bauen nun eine Maxime, die das Individuum vernachlässigt und nur nach dem globalen Optimum strebt:

**fun** *globaler-strikter-fortschritt* :: *zahlenwelt handlung*  $\Rightarrow$  *bool* **where**  
*globaler-strikter-fortschritt* (*Handlung vor nach*)  $\longleftrightarrow$  (*gesamtbesitz vor*)  $<$  (*gesamtbesitz nach*)

Die Maxime ignoriert das *ich* komplett.

Nun ist es *Alice* wieder erlaubt, Wohlstand für sich selbst zu erzeugen, da sich dadurch auch der Gesamtwohlstand erhöht:

**lemma**  $\langle$  *beispiel-case-law-relativ*

$$\begin{aligned} & (Maxime (\lambda ich. \text{ globaler-strikter-fortschritt})) \\ & (HandlungF (\text{erschaffen } 5)) = \\ & Gesetz \{(\$ 1, Rechtsnorm (\text{Tatbestand } [\text{Gewinnt Alice } 5]) (\text{Rechtsfolge Erlaubnis}))\} \} \end{aligned}$$

Allerdings ist auch diese Maxime auch sehr grausam, da sie Untätigkeit verbietet:

**lemma**  $\langle \text{beispiel-case-law-relativ}$   

$$\begin{aligned} & (Maxime (\lambda ich. \text{ globaler-strikter-fortschritt})) \\ & (HandlungF (\text{erschaffen } 0)) = \\ & Gesetz \{(\$ 1, Rechtsnorm (\text{Tatbestand } []) (\text{Rechtsfolge Verbot}))\} \} \end{aligned}$$

Unsere initiale einfache *maxime-zahlenfortschritt* würde Untätigkeit hier erlauben:

**lemma**  $\langle \text{beispiel-case-law-relativ}$   

$$\begin{aligned} & \text{maxime-zahlenfortschritt} \\ & (HandlungF (\text{erschaffen } 0)) = \\ & Gesetz \{(\$ 1, Rechtsnorm (\text{Tatbestand } []) (\text{Rechtsfolge Erlaubnis}))\} \} \end{aligned}$$

Wir können die Maxime für globalen Fortschritt etwas lockern:

**fun** *globaler-fortschritt* :: *zahlenwelt handlung*  $\Rightarrow$  *bool* **where**  
*globaler-fortschritt* (*Handlung vor nach*)  $\longleftrightarrow$  (*gesamtbesitz vor*)  $\leq$  (*gesamtbesitz nach*)

Untätigkeit ist nun auch hier erlaubt:

**lemma**  $\langle \text{beispiel-case-law-relativ}$   

$$\begin{aligned} & (Maxime (\lambda ich. \text{ globaler-fortschritt})) \\ & (HandlungF (\text{erschaffen } 0)) \\ & = \\ & Gesetz \{(\$ 1, Rechtsnorm (\text{Tatbestand } []) (\text{Rechtsfolge Erlaubnis}))\} \} \end{aligned}$$

Allerdings ist auch Stehlen erlaubt, da global gesehen, kein Besitz vernichtet wird:

**lemma**  $\langle \text{beispiel-case-law-relativ}$   

$$\begin{aligned} & (Maxime (\lambda ich. \text{ globaler-fortschritt})) \\ & (HandlungF (\text{stehlen } 5 \text{ Bob})) \\ & = \\ & Gesetz \\ & \{(\$ 1, Rechtsnorm (\text{Tatbestand } [\text{Gewinnt Alice } 5, \text{ Verliert Bob } 5]) (\text{Rechtsfolge Erlaubnis}))\} \} \end{aligned}$$

## 10.6 Alice stiehlt 5

Zurück zur einfachen *maxime-zahlenfortschritt*.

Stehlen ist verboten:

**lemma**  $\langle \text{beispiel-case-law-relativ}$  *maxime-zahlenfortschritt* (*HandlungF* (*stehlen* 5 *Bob*)) =  
*Gesetz*  

$$\{(\$ 1, Rechtsnorm (\text{Tatbestand } [\text{Gewinnt Alice } 5, \text{ Verliert Bob } 5]) (\text{Rechtsfolge Verbot}))\} \}$$



Auch wenn *Alice* von sich selbst stehlen möchte ist dies verboten, obwohl hier keiner etwas verliert:

**lemma**  $\langle$  *beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*stehlen 5 Alice*)) =  
*Gesetz* { (§ 1, *Rechtsnorm* (*Tatbestand []*) (*Rechtsfolge Verbot*)) }  $\rangle$

Der Grund ist, dass *Alice* die abstrakte Handlung "Alice wird bestohlen" gar nicht gut fände, wenn sie jemand anderes ausführt:

**lemma**  $\langle$  *debug-maxime show-zahlenwelt initialwelt*  
(*HandlungF* (*stehlen 5 Alice*)) *maxime-zahlenfortschritt* =  
{ *VerletzteMaxime* (*Opfer Alice*) (*Taeter Bob*)  
(*Handlung* [(*Alice*, 5), (*Bob*, 10)] [(*Bob*, 15)]),  
*VerletzteMaxime* (*Opfer Alice*) (*Taeter Carol*)  
(*Handlung* [(*Alice*, 5), (*Bob*, 10)] [(*Bob*, 10), (*Carol*, 5)]),  
*VerletzteMaxime* (*Opfer Alice*) (*Taeter Eve*)  
(*Handlung* [(*Alice*, 5), (*Bob*, 10)] [(*Bob*, 10), (*Eve*, 5)])  
}  $\rangle$

Leider ist das hier abgeleitete Gesetz sehr fragwürdig: *Rechtsnorm* (*Tatbestand []*) (*Rechtsfolge Verbot*)

Es besagt, dass Nichtstun verboten ist.

Indem wir die beiden Handlungen Nichtstun und Selbstbestehlen betrachten, können wir sogar ein widersprüchliches Gesetz ableiten:

**lemma**  $\langle$  *simulateOne*  
(*SimConsts*  
*Alice*  
*maxime-zahlenfortschritt*  
(*case-law-ableiten-relativ delta-zahlenwelt*)  
20 (*HandlungF* (*stehlen 5 Alice*)) *initialwelt*  
(*beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*erschaffen 0*)))  
=  
*Gesetz*  
{ (§ 2, *Rechtsnorm* (*Tatbestand []*) (*Rechtsfolge Verbot*)),  
(§ 1, *Rechtsnorm* (*Tatbestand []*) (*Rechtsfolge Erlaubnis*)) }  $\rangle$

Meine persönliche Conclusion: Wir müssen irgendwie die Absicht mit ins Gesetz schreiben.

## 10.7 Schenken

Es ist *Alice* verboten, etwas zu verschenken:

**lemma**  $\langle$  *beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*schenken 5 Bob*))  
=  
*Gesetz*  
{ (§ 1,

*Rechtsnorm (Tatbestand [Verliert Alice 5, Gewinnt Bob 5]) (Rechtsfolge Verbot))*›

Der Grund ist, dass *Alice* dabei etwas verliert und die *maxime-zahlenfortschritt* dies nicht Erlaubt. Es fehlt eine Möglichkeit zu modellieren, dass *Alice* damit einverstanden ist, etwas abzugeben. Doch wir haben bereits in *stehlen i = schenken (− i)* gesehen, dass *stehlen* und *schenken* nicht unterscheidbar sind.

## 10.8 Ungültige Maxime

Es ist verboten, in einer Maxime eine spezielle Person hardzucoden. Da dies gegen die Gleichbehandlung aller Menschen verstoßen würde.

Beispielsweise könnten wir *individueller-fortschritt* nicht mehr parametrisiert verwenden, sondern einfach *Alice* reinschreiben:

**lemma** *individueller-fortschritt Alice*  
 $= (\lambda h. \text{case } h \text{ of Handlung vor nach} \Rightarrow (\text{meins Alice vor}) \leq (\text{meins Alice nach}))$

Dies würde es erlauben, dass *Alice* Leute bestehlen darf:

**lemma** *beispiel-case-law-relativ*  
 $(\text{Maxime } (\lambda ich. \text{individueller-fortschritt Alice}))$   
 $(\text{HandlungF } (\text{stehlen } 5 \text{ Bob}))$   
 $=$   
*Gesetz*  
 $\{(\S \ 1, \text{Rechtsnorm } (\text{Tatbestand } [\text{Gewinnt Alice } 5, \text{ Verliert Bob } 5]) (\text{Rechtsfolge Erlaubnis}))\}$ ›

## 11 Einkommensteuergesetzgebung

Basierend auf einer stark vereinfachten Version des deutschen Steuerrechts. Wenn ich Wikipedia richtig verstanden habe, habe ich sogar aus Versehen einen Teil des österreichischen Steuersystem gebaut mit deutschen Konstanten.

Folgende **locale** nimmt an, dass wir eine Funktion *steuer::nat ⇒ nat* haben, welche basierend auf dem Einkommen die zu zahlende Steuer berechnet.

Die **locale** enthält einige Definition, gegeben die *steuer* Funktion.

Eine konkrete *steuer* Funktion wird noch nicht gegeben.

**locale** *steuer-defs* =  
**fixes** *steuer* :: *nat* ⇒ *nat* — Einkommen -> Steuer  
**begin**  
**definition** *brutto* :: *nat* ⇒ *nat* **where**  
*brutto einkommen* ≡ *einkommen*  
**definition** *netto* :: *nat* ⇒ *nat* **where**  
*netto einkommen* ≡ *einkommen* − (*steuer einkommen*)  
**definition** *steuersatz* :: *nat* ⇒ *percentage* **where**

```

    steuersatz einkommen  $\equiv$  percentage ((steuer einkommen) / einkommen)
end

```

Beispiel

```

definition beispiel-25prozent-steuer :: nat  $\Rightarrow$  nat where
    beispiel-25prozent-steuer e  $\equiv$  nat [real e * (percentage 0.25)]

```

```

lemma beispiel-25prozent-steuer 100 = 25
    steuer-defs.brutto 100 = 100
    steuer-defs.netto beispiel-25prozent-steuer 100 = 75
    steuer-defs.steuersatz beispiel-25prozent-steuer 100 = percentage 0.25

```

```

lemma steuer-defs.steuersatz beispiel-25prozent-steuer 103 = percentage (25 / 103)
    percentage (25 / 103)  $\leq$  percentage 0.25
    (103::nat) > 100

```

Folgende **locale** erweitert die *steuer-defs* **locale** und stellt einige Anforderungen die eine gültige *steuer* Funktion erfüllen muss.

```

locale steuersystem = steuer-defs +
assumes wer-hat-der-gibt:
    einkommen-a  $\geq$  einkommen-b  $\implies$  steuer einkommen-a  $\geq$  steuer einkommen-b

```

```

and leistung-lohnt-sich:
    einkommen-a  $\geq$  einkommen-b  $\implies$  netto einkommen-a  $\geq$  netto einkommen-b

```

— Ein Existenzminimum wird nicht versteuert. Zahl Deutschland 2022, vermutlich sogar die falsche Zahl.

```

and existenzminimum:
    einkommen  $\leq$  9888  $\implies$  steuer einkommen = 0

```

**begin**

**end**

Die folgende Liste, basierend auf [https://de.wikipedia.org/wiki/Einkommensteuer\\_\(Deutschland\)#Tarif\\_2022](https://de.wikipedia.org/wiki/Einkommensteuer_(Deutschland)#Tarif_2022), sagt in welchem Bereich welcher Prozentsatz an Steuern zu zahlen ist. Beispielsweise sind die ersten 10347 steuerfrei.

```

definition steuerbuckets2022 :: (nat  $\times$  percentage) list where
    steuerbuckets2022  $\equiv$  [
        (10347, percentage 0),
        (14926, percentage 0.14),
        (58596, percentage 0.2397),
        (277825, percentage 0.42)
    ]

```

Wir ignorieren die Progressionsfaktoren in Zone 2 und 3.

Folgende Funktion berechnet die zu Zahlende Steuer, basierend auf einer Steuerbucketliste.

```
fun bucketsteuerAbs :: (nat × percentage) list ⇒ percentage ⇒ nat ⇒ real where
  bucketsteuerAbs ((bis, prozent)#mehr) spitzensteuer e =
    ((min bis e) * prozent)
    + (bucketsteuerAbs (map (λ(s,p). (s-bis,p)) mehr) spitzensteuer (e - bis))
| bucketsteuerAbs [] spitzensteuer e = e*spitzensteuer
```

Die Einkommenssteuerberechnung, mit Spitzensteuersatz 45 Prozent und finalem Abrunden.

```
definition einkommenssteuer :: nat ⇒ nat where
  einkommenssteuer einkommen ≡
    floor (bucketsteuerAbs steuerbuckets2022 (percentage 0.45) einkommen)

value <einkommenssteuer 10>
lemma <einkommenssteuer 10 = 0>
lemma <einkommenssteuer 10000 = 0>
lemma <einkommenssteuer 14000 = floor ((14000-10347)*0.14)>
lemma <einkommenssteuer 20000 =
  floor ((14926-10347)*0.14 + (20000-14926)*0.2397)>
value <einkommenssteuer 40000>
value <einkommenssteuer 60000>
```

Die *einkommenssteuer* Funktion erfüllt die Anforderungen an *steuersystem*.

```
interpretation steuersystem
  where steuer = einkommenssteuer
```

## 12 Beispiel: Steuern

Wenn die Welt sich durch eine Zahl darstellen lässt, ...

Achtung: Im Unterschied zum BeispielZahlenwelt.thy modellieren wir hier nicht den Gesamtbesitz, sondern das Jahreseinkommen. Besitz wird ignoriert.

```
datatype steuerwelt = Steuerwelt
  (get-einkommen: person ⇒ int) — einkommen: einkommen jeder Person (im Zweifel 0).
```

```
fun steuerlast :: person ⇒ steuerwelt handlung ⇒ int where
  steuerlast p (Handlung vor nach) = ((get-einkommen vor) p) - ((get-einkommen nach) p)
```

```
fun brutto :: person ⇒ steuerwelt handlung ⇒ int where
  brutto p (Handlung vor nach) = (get-einkommen vor) p
fun netto :: person ⇒ steuerwelt handlung ⇒ int where
  netto p (Handlung vor nach) = (get-einkommen nach) p
```

**lemma**  $\langle \text{steuerlast Alice (Handlung (Steuerwelt } \clubsuit[Alice:=8]) (Steuerwelt \clubsuit[Alice:=5])) = 3 \rangle$   
**lemma**  $\langle \text{steuerlast Alice (Handlung (Steuerwelt } \clubsuit[Alice:=8]) (Steuerwelt \clubsuit[Alice:=0])) = 8 \rangle$   
**lemma**  $\langle \text{steuerlast Bob (Handlung (Steuerwelt } \clubsuit[Alice:=8]) (Steuerwelt \clubsuit[Alice:=5])) = 0 \rangle$   
**lemma**  $\langle \text{steuerlast Alice (Handlung (Steuerwelt } \clubsuit[Alice:=-3]) (Steuerwelt \clubsuit[Alice:=-4])) = 1 \rangle$   
**lemma**  $\langle \text{steuerlast Alice (Handlung (Steuerwelt } \clubsuit[Alice:=1]) (Steuerwelt \clubsuit[Alice:=-1])) = 2 \rangle$

**fun** *mehrverdiener* :: *person*  $\Rightarrow$  *steuerwelt* *handlung*  $\Rightarrow$  *person* *set* **where**  
*mehrverdiener* *ich* (*Handlung* *vor* *nach*) = {*p*. (*get-einkommen* *vor*) *p*  $\geq$  (*get-einkommen* *vor*) *ich*}

**lemma**  $\langle \text{mehrverdiener Alice}$   
     (*Handlung* (*Steuerwelt*  $\clubsuit[Alice:=8, Bob:=12, Eve:=7]$ ) (*Steuerwelt*  $\clubsuit[Alice:=5]$ ))  
   = {*Alice, Bob*} $\rangle$

**definition** *maxime-steuern* :: (*person*, *steuerwelt*) *maxime* **where**  
*maxime-steuern*  $\equiv$  *Maxime*  
   ( $\lambda$ *ich* *handlung*.  
     ( $\forall p \in \text{mehrverdiener } ich \text{ handlung.}$   
       *steuerlast* *ich* *handlung*  $\leq$  *steuerlast* *p* *handlung*)  
      $\wedge$  ( $\forall p \in \text{mehrverdiener } ich \text{ handlung.}$   
       *netto* *ich* *handlung*  $\leq$  *netto* *p* *handlung*)  
   )

**definition** *sc*  $\equiv$  *SimConsts*  
   *Alice*  
   *maxime-steuern*  
   (*printable-case-law-ableiten-absolut* ( $\lambda w. \text{show-fun (get-einkommen } w)$ ))

**definition** *sc'*  $\equiv$  *SimConsts*  
   *Alice*  
   *maxime-steuern*  
   (*case-law-ableiten-relativ* *delta-steuerwelt*)

**definition** *initialwelt*  $\equiv$  *Steuerwelt*  $\clubsuit[Alice:=8, Bob:=3, Eve:= 5]$

**definition** *beispiel-case-law* *h*  $\equiv$  *simulateOne* *sc* 3 *h* *initialwelt* (*Gesetz* {})

**definition** *beispiel-case-law'* *h*  $\equiv$  *simulateOne* *sc'* 20 *h* *initialwelt* (*Gesetz* {})

Keiner zahlt steuern: funktioniert

**value**  $\langle \text{beispiel-case-law (HandlungF } (\lambda ich \text{ welt. welt})) \rangle$   
**lemma**  $\langle \text{beispiel-case-law' (HandlungF } (\lambda ich \text{ welt. welt})) =$   
   *Gesetz* {( $\S$  1, *Rechtsnorm* (*Tatbestand* [])) (*Rechtsfolge* *Erlaubnis*)} $\rangle$

Ich zahle 1 Steuer: funnktioniert nicht, .... komisch, sollte aber? Achjaaaaaa, jeder muss ja Steuer zahlen, ....

**definition** *ich-zahle-1-steuer* *ich* *welt*  $\equiv$   
   *Steuerwelt* ((*get-einkommen* *welt*)(*ich* := ((*get-einkommen* *welt*) *ich*) - 1))

**lemma**  $\langle \text{beispiel-case-law (HandlungF } ich\text{-zahle-1-steuer) =}$

*Gesetz*  
 $\{(\S\ 1,$   
*Rechtsnorm*  
 $($  *Tatbestand*  
 $(([$  *(Alice, 8), (Bob, 3), (Carol, 0), (Eve, 5)],*  
 $[$  *(Alice, 7), (Bob, 3), (Carol, 0), (Eve, 5)])*  
 $($  *Rechtsfolge Verbot* $))\}$   
**lemma**  $\langle$  *beispiel-case-law'*  $($  *HandlungF ich-zahle-1-steuer*  $) =$   
*Gesetz*  
 $\{(\S\ 1,$  *Rechtsnorm*  $($  *Tatbestand*  $[$  *Verliert Alice 1]*  
 $($  *Rechtsfolge Verbot* $))\}\rangle$

Jeder muss steuern zahlen: funktioniert, ist aber doof, denn am Ende sind alle im Minus.  
Das *ich* wird garnicht verwendet, da jeder Steuern zahlt.

**definition** *jeder-zahle-1-steuer ich welt*  $\equiv$   
*Steuerwelt*  $((\lambda e. e - 1) \circ (\text{get-einkommen welt}))$   
**lemma**  $\langle$  *beispiel-case-law*  $($  *HandlungF jeder-zahle-1-steuer*  $) =$   
*Gesetz*

$\{(\S\ 3,$   
*Rechtsnorm*  
 $($  *Tatbestand*  
 $(([$  *(Alice, 6), (Bob, 1), (Carol, - 2), (Eve, 3)],*  
 $[$  *(Alice, 5), (Bob, 0), (Carol, - 3), (Eve, 2)])*  
 $($  *Rechtsfolge Erlaubnis* $)),$   
 $(\S\ 2,$   
*Rechtsnorm*  
 $($  *Tatbestand*  
 $(([$  *(Alice, 7), (Bob, 2), (Carol, - 1), (Eve, 4)],*  
 $[$  *(Alice, 6), (Bob, 1), (Carol, - 2), (Eve, 3)])*  
 $($  *Rechtsfolge Erlaubnis* $)),$   
 $(\S\ 1,$   
*Rechtsnorm*  
 $($  *Tatbestand*  
 $(([$  *(Alice, 8), (Bob, 3), (Carol, 0), (Eve, 5)],*  
 $[$  *(Alice, 7), (Bob, 2), (Carol, - 1), (Eve, 4)])*  
 $($  *Rechtsfolge Erlaubnis* $))\}\rangle$   
**lemma**  $\langle$  *beispiel-case-law'*  $($  *HandlungF jeder-zahle-1-steuer*  $) =$   
*Gesetz*  
 $\{(\S\ 1,$   
*Rechtsnorm*  
 $($  *Tatbestand*  $[$  *Verliert Alice 1, Verliert Bob 1, Verliert Carol 1, Verliert Eve 1]*  
 $($  *Rechtsfolge Erlaubnis* $))\}\rangle$

Jetzt kommt die Steuern.thy ins Spiel.

Bei dem geringen Einkommen zahlt keiner Steuern.

**definition** *jeder-zahlt steuerberechnung ich welt*  $\equiv$   
*Steuerwelt*  $((\lambda e. e - \text{steuerberechnung } e) \circ \text{nat} \circ (\text{get-einkommen welt}))$

**definition** *jeder-zahlt-einkommenssteuer*  $\equiv$  *jeder-zahlt einkommenssteuer*

**lemma**  $\langle$  *beispiel-case-law* (*HandlungF* *jeder-zahlt-einkommenssteuer*) =  
*Gesetz*  
 $\{(\S$  1,  
*Rechtsnorm*  
*(Tatbestand*  
 $[($ (*Alice*, 8), (*Bob*, 3), (*Carol*, 0), (*Eve*, 5)],  
 $[($ (*Alice*, 8), (*Bob*, 3), (*Carol*, 0), (*Eve*, 5))])  
*(Rechtsfolge Erlaubnis)) $\rangle$*

**lemma**  $\langle$  *simulateOne*

*sc' 1*  
(*HandlungF* *jeder-zahlt-einkommenssteuer*)  
(*Steuerwelt*  $\bullet$ [*Alice*:=10000, *Bob*:=14000, *Eve*:= 20000])  
(*Gesetz* {})  
=  
*Gesetz*  
 $\{(\S$  1,  
*Rechtsnorm* (*Tatbestand* [*Verliert Bob* 511, *Verliert Eve* 1857])  
*(Rechtsfolge Erlaubnis)) $\rangle$*

Die Anforderungen fuer ein *steuersystem* und die *maxime-steuern* sind vereinbar.

**lemma** *steuersystem steuersystem-impl*  $\implies$   
 $(\forall$  *welt. teste-maxime welt* (*HandlungF* (*jeder-zahlt steuersystem-impl*)) *maxime-steuern*)

**lemma**  $a \leq x \implies \text{int } x - \text{int } (x - a) = a$

Danke ihr nats. Macht also keinen Sinn das als Annahme in die Maxime zu packen....

**lemma** *steuern-kleiner-einkommen-nat*:  
*steuerlast ich* (*Handlung welt* (*jeder-zahlt steuersystem-impl ich welt*))  
 $\leq$  *brutto ich* (*Handlung welt* (*jeder-zahlt steuersystem-impl ich welt*))

**lemma**  $(\forall$  *einkommen. steuersystem-impl einkommen*  $\leq$  *einkommen*)  $\implies$   
 $(\forall$  *einkommen. einkommen*  $\leq$  9888  $\implies$  *steuersystem-impl einkommen* = 0)  $\implies$   
 $\forall$  *welt. teste-maxime welt* (*HandlungF* (*jeder-zahlt steuersystem-impl*)) *maxime-steuern*  
 $\implies$  *steuersystem steuersystem-impl*