

# Formal

Cornelius Diekmann

September 24, 2022

## Contents

|          |                                                  |           |
|----------|--------------------------------------------------|-----------|
| <b>1</b> | <b>Disclaimer</b>                                | <b>1</b>  |
| <b>2</b> | <b>Gesetz</b>                                    | <b>1</b>  |
| <b>3</b> | <b>Kant's Kategorischer Imperativ</b>            | <b>3</b>  |
| 3.1      | Maxime . . . . .                                 | 3         |
| 3.1.1    | Maximen Debugging . . . . .                      | 4         |
| 3.1.2    | Beispiel . . . . .                               | 5         |
| 3.2      | Allgemeines Gesetz Ableiten . . . . .            | 6         |
| 3.3      | Implementierung Kategorischer Imperativ. . . . . | 6         |
| <b>4</b> | <b>Zahlenwelt Helper</b>                         | <b>7</b>  |
| <b>5</b> | <b>Simulation</b>                                | <b>8</b>  |
| <b>6</b> | <b>Gesetze</b>                                   | <b>9</b>  |
| 6.1      | Case Law Absolut . . . . .                       | 9         |
| 6.2      | Case Law Relativ . . . . .                       | 9         |
| <b>7</b> | <b>Beispiel: Zahlenwelt</b>                      | <b>10</b> |
| 7.1      | Handlungen . . . . .                             | 10        |
| 7.2      | Setup . . . . .                                  | 10        |
| 7.3      | Alice erzeugt 5 Wohlstand für sich. . . . .      | 10        |
| 7.4      | Kleine Änderung in der Maxime . . . . .          | 12        |
| 7.5      | Maxime für Globales Optimum . . . . .            | 12        |
| 7.6      | TODO . . . . .                                   | 13        |
| <b>8</b> | <b>Experiment: Steuergesetzgebung</b>            | <b>13</b> |
| <b>9</b> | <b>Beispiel: Steuern</b>                         | <b>16</b> |

# 1 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!

# 2 Gesetz

Definiert einen Datentyp um Gesetzestext zu modellieren.

**datatype** *'a tatbestand = Tatbestand*  $\langle 'a \rangle$

**datatype** *'a rechtsfolge = Rechtsfolge*  $\langle 'a \rangle$

**datatype** *('a, 'b) rechtsnorm = Rechtsnorm*  $\langle 'a \text{ tatbestand} \rangle \langle 'b \text{ rechtsfolge} \rangle$

**datatype** *'p prg = Paragraph*  $\langle 'p \rangle$

**datatype** *('p, 'a, 'b) gesetz = Gesetz*  $\langle ('p \text{ prg} \times ('a, 'b) \text{ rechtsnorm}) \text{ set} \rangle$

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

```
value  $\langle \text{Gesetz} \{$   
  (Paragraph "823 BGB",  
    Rechtsnorm  
      (Tatbestand "Wer vorsaeztlich 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.")  
    ),  
    (Paragraph "985 BGB",  
      Rechtsnorm  
        (Tatbestand "Der Eigentuemmer einer Sache kann von dem Besitzer")  
        (Rechtsfolge "die Herausgabe der Sache verlangen")  
      ),  
    (Paragraph "303 StGB",  
      Rechtsnorm  
        (Tatbestand "Wer rechtswidrig eine fremde Sache beschaedigt oder zerstoert,")  
        (Rechtsfolge "wird mit Freiheitsstrafe bis zu zwei Jahren oder mit Geldstrafe bestraft.")  
      )  
    } \rangle
```

```
fun neuer-paragraph ::  $\langle (nat, 'a, 'b) \text{ gesetz} \Rightarrow nat \text{ prg} \rangle$  where
   $\langle \text{neuer-paragraph (Gesetz } G) = \text{Paragraph } ((\text{max-paragraph (fst } ' G)) + 1) \rangle$ 
```

Fügt eine Rechtsnorm als neuen Paragraphen hinzu:

```
fun hinzufuegen ::  $\langle ('a, 'b) \text{ rechtsnorm} \Rightarrow (nat, 'a, 'b) \text{ gesetz} \Rightarrow (nat, 'a, 'b) \text{ gesetz} \rangle$  where
   $\langle \text{hinzufuegen rn (Gesetz } G) =$ 
     $(\text{if } rn \in (\text{snd } ' G) \text{ then Gesetz } G \text{ else Gesetz (insert (neuer-paragraph (Gesetz } G), rn) G)) \rangle$ 
```

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

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

Beispiel:

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

### 3 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\\_Imperativ](https://de.wikipedia.org/wiki/Kategorischer_Imperativ)

Meine persönliche, etwas utilitaristische, Interpretation.

#### 3.1 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*  
 $(\lambda(p, x). m \ p \ (\text{handeln } x \ \text{welt handlung})) \ (List.product \ \text{bevoelk} \ \text{bevoelk})$ .

### 3.1.1 Maximen Debugging

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

```
datatype ('person, 'world) verletzte-maxime =
  VerletzteMaxime
  <'person> — verletzt für; das Opfer
  <'person> — 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 p1 p2 (map-handlung print-world (handeln p2 welt handlung)) | p1 p2.
       $\neg m \ p1 \ (\text{handeln } p2 \ \text{welt handlung}) \}$ 
```

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*

### 3.1.2 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 ( $\lambda(person::person) \ \text{welt. welt} + 1$ ))
  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 ( $\lambda person \ \text{welt. welt}(person \mapsto 3)$ ))
  (Maxime ( $\lambda person \ \text{handlung.}$ 
    (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.

**lemma**  $\langle \neg\ teste-maxime$   
 $[Alice \mapsto (0::nat), Bob \mapsto 4, Carol \mapsto 0, Eve \mapsto 0]$   
 $(HandlungF (\lambda person\ welt. welt(person \mapsto 3)))$   
 $(Maxime (\lambda person\ handlung.$   
 $\quad (the ((vorher\ handlung)\ person)) \leq (the ((nachher\ handlung)\ person)))) \rangle$

**lemma**  $\langle debug-maxime\ show-map$   
 $[Alice \mapsto (0::nat), Bob \mapsto 4, Carol \mapsto 0, Eve \mapsto 0]$   
 $(HandlungF (\lambda person\ welt. welt(person \mapsto 3)))$   
 $(Maxime (\lambda person\ handlung.$   
 $\quad (the ((vorher\ handlung)\ person)) \leq (the ((nachher\ handlung)\ person))))$   
 $= \{ VerletzteMaxime\ Bob\ Bob$   
 $(Handlung\ [(Alice, 0), (Bob, 4), (Carol, 0), (Eve, 0)]$   
 $\quad [(Alice, 0), (Bob, 3), (Carol, 0), (Eve, 0)]) \} \rangle$

### 3.2 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.

### 3.3 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
  )⟩

```

## 4 Zahlenwelt Helper

Wir werden Beispiele betrachten, in denen wir Welten modellieren, in denen jeder Person eine Zahl zugewiesen wird: *person* ⇒ *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  $0::'a$ :

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

## 5 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**  $\langle simulateOne$   
 $(SimConsts () (Maxime (\lambda -. True)) (\lambda h s. Rechtsnorm (Tatbestand h) (Rechtsfolge "count"))$   
 $10 (HandlungF (\lambda p n. Suc n))$   
 $32$   
 $(Gesetz \{\}) =$   
*Gesetz*



```
{(Paragraph 10, Rechtsnorm (Tatbestand (Handlung 41 42)) (Rechtsfolge "count")),
 (Paragraph 9, Rechtsnorm (Tatbestand (Handlung 40 41)) (Rechtsfolge "count")),
 (Paragraph 8, Rechtsnorm (Tatbestand (Handlung 39 40)) (Rechtsfolge "count")),
 (Paragraph 7, Rechtsnorm (Tatbestand (Handlung 38 39)) (Rechtsfolge "count")),
 (Paragraph 6, Rechtsnorm (Tatbestand (Handlung 37 38)) (Rechtsfolge "count")),
 (Paragraph 5, Rechtsnorm (Tatbestand (Handlung 36 37)) (Rechtsfolge "count")),
 (Paragraph 4, Rechtsnorm (Tatbestand (Handlung 35 36)) (Rechtsfolge "count")),
 (Paragraph 3, Rechtsnorm (Tatbestand (Handlung 34 35)) (Rechtsfolge "count")),
 (Paragraph 2, Rechtsnorm (Tatbestand (Handlung 33 34)) (Rechtsfolge "count")),
 (Paragraph 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 {(Paragraph 1, Rechtsnorm (Tatbestand tb) (Rechtsfolge rf))}>
```

## 6 Gesetze

Wir implementieren Strategien um (*'world, 'a, 'b*) *allgemeines-gesetz-ableiten* zu implementieren.

### 6.1 Case Law Absolut

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

**type-synonym** *'world case-law* = (*nat, ('world × 'world), sollensanordnung*) *gesetz*

Überträgt einen Tatbestand wörtlich ins Gesetz. Nicht sehr allgemein.

```
definition case-law-ableiten-absolut
  :: ('world, ('world × 'world), sollensanordnung) allgemeines-gesetz-ableiten
where
  case-law-ableiten-absolut handlung sollensanordnung =
    Rechtsnorm
      (Tatbestand (vorher handlung, nachher handlung))
      (Rechtsfolge sollensanordnung)
```

```
definition printable-case-law-ableiten-absolut
  :: ('world ⇒ 'printable-world) ⇒
    ('world, ('printable-world × 'printable-world), sollensanordnung) allgemeines-gesetz-ableiten
where
  printable-case-law-ableiten-absolut print-world h ≡
    case-law-ableiten-absolut (map-handlung print-world h)
```

## 6.2 Case Law Relativ

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

```
fun case-law-ableiten-relativ
  :: ('world handlung  $\Rightarrow$  (('person, 'etwas) aenderung) list)
     $\Rightarrow$  ('world, (('person, 'etwas) aenderung) list, sollensanordnung)
    allgemeines-gesetz-ableiten
where
  case-law-ableiten-relativ delta handlung erlaubt =
    Rechtsnorm (Tatbestand (delta handlung)) (Rechtsfolge erlaubt)
```

## 7 Beispiel: Zahlenwelt

Wir nehmen an, die Welt lässt sich durch eine Zahl darstellen, die den Besitz einer Person modelliert.

```
datatype zahlenwelt = Zahlenwelt
  person  $\Rightarrow$  int — besitz: Besitz jeder Person.
```

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

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

```
lemma gesamtbesitz (Zahlenwelt ♣[Alice := 4, Carol := 4]) = 8
```

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

```
lemma meins Carol (Zahlenwelt ♣[Alice := 8, Carol := 4]) = 4
```

### 7.1 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 := (besitz p) + int i))
```

### 7.2 Setup

```
definition initialwelt  $\equiv$  Zahlenwelt ♣[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 initialwelt (Gesetz {})

```

### 7.3 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**  $\langle$ beispiel-case-law-absolut maxime-zahlenfortschritt (HandlungF (erschaffen 5))

=

```

Gesetz
{ (Paragraph 10,
  Rechtsnorm (Tatbestand ([ (Alice, 50), (Bob, 10)], [(Alice, 55), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 9,
  Rechtsnorm (Tatbestand ([ (Alice, 45), (Bob, 10)], [(Alice, 50), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 8,
  Rechtsnorm (Tatbestand ([ (Alice, 40), (Bob, 10)], [(Alice, 45), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 7,
  Rechtsnorm (Tatbestand ([ (Alice, 35), (Bob, 10)], [(Alice, 40), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 6,
  Rechtsnorm (Tatbestand ([ (Alice, 30), (Bob, 10)], [(Alice, 35), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 5,
  Rechtsnorm (Tatbestand ([ (Alice, 25), (Bob, 10)], [(Alice, 30), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 4,
  Rechtsnorm (Tatbestand ([ (Alice, 20), (Bob, 10)], [(Alice, 25), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 3,
  Rechtsnorm (Tatbestand ([ (Alice, 15), (Bob, 10)], [(Alice, 20), (Bob, 10)]))
  (Rechtsfolge Erlaubnis)),
  (Paragraph 2,
  Rechtsnorm (Tatbestand ([ (Alice, 10), (Bob, 10)], [(Alice, 15), (Bob, 10)]))

```

```

    (Rechtsfolge Erlaubnis)),
  (Paragraph 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 <beispiel-case-law-relativ maxime-zahlenfortschritt (HandlungF (erschaffen 5)) =
  Gesetz
  {(Paragraph 1, Rechtsnorm (Tatbestand [Gewinnt Alice 5]) (Rechtsfolge Erlaubnis))}>

```

## 7.4 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 <beispiel-case-law-relativ
  (Maxime ( $\lambda$ ich. individueller-strikter-fortschritt ich))
  (HandlungF (erschaffen 5)) =
  Gesetz {(Paragraph 1, Rechtsnorm (Tatbestand [Gewinnt Alice 5]) (Rechtsfolge Verbot))}>

```

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<VerletzteMaxime Bob 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))>

```

## 7.5 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 <beispiel-case-law-relativ
  (Maxime ( $\lambda$ ich. globaler-strikter-fortschritt))
  (HandlungF (erschaffen 5)) =
  Gesetz {(Paragraph 1, Rechtsnorm (Tatbestand [Gewinnt Alice 5]) (Rechtsfolge Erlaubnis))}>

```

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

```

lemma <beispiel-case-law-relativ
  (Maxime ( $\lambda$ ich. globaler-strikter-fortschritt))
  (HandlungF (erschaffen 0)) =
  Gesetz {(Paragraph 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Verbot))}>

```

```

value<debug-maxime show-zahlenwelt initialwelt
  (HandlungF (erschaffen 0)) (Maxime ( $\lambda$ ich. globaler-strikter-fortschritt))>

```

## 7.6 TODO

Mehr ist mehr gut. Globaler Fortschritt erlaubt stehlen, solange dabei nichts vernichtet wird.  
Größer ( $>$ ) anstelle ( $>=$ ) ist hier echt spannend!

Dieser globale Fortschritt sollte eigentlich allgemeines Gesetz werden und die Maxime sollte individuelle Bereicherung sein (und die unsichtbare Hand macht den Rest. YOLO).

## 8 Experiment: Steuergesetzgebung

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.

```

locale steuer-defs =
  fixes steuer :: nat  $\Rightarrow$  nat — Einkommen -> Steuer
begin
  definition brutto :: nat  $\Rightarrow$  nat where
    brutto einkommen  $\equiv$  einkommen
  definition netto :: nat  $\Rightarrow$  nat where
    netto einkommen  $\equiv$  einkommen - (steuer einkommen)
  definition steuersatz :: nat  $\Rightarrow$  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) ≤ percentage 0.25*  
*(103::nat) > 100*

**locale** *steuersystem = steuer-defs +*  
**assumes** *wer-hat-der-gibt:*  
*einkommen-a ≥ einkommen-b ⇒ steuer einkommen-a ≥ steuer einkommen-b*

**and** *leistung-lohnt-sich:*  
*einkommen-a ≥ einkommen-b ⇒ netto einkommen-a ≥ netto einkommen-b*

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

**and** *existenzminimum:*  
*einkommen ≤ 9888 ⇒ steuer einkommen = 0*

**begin**

**end**

**fun** *zonensteuer :: (nat × percentage) list ⇒ percentage ⇒ nat ⇒ real where*  
*zonensteuer ((zone, prozent)#zonen) spitzensteuer e =*  
*((min zone e) \* prozent) + (zonensteuer zonen spitzensteuer (e - zone))*  
*| zonensteuer [] spitzensteuer e = e\*spitzensteuer*

**lemma** *zonensteuermono: e1 ≤ e2*  
*⇒ zonensteuer zs spitzensteuer e1 ≤ zonensteuer zs spitzensteuer e2*

Kein Einkommen -> keine Steuer

**lemma** *zonensteuer-zero: zonensteuer ls p 0 = 0*

Steuer ist immer positiv.

**lemma** *zonensteuer-pos: zonensteuer ls p e ≥ 0*

Steuer kann nicht höher sein als das Einkommen.

**lemma** *zonensteuer-limit: zonensteuer ls spitzensteuer einkommen ≤ einkommen*

**lemma** *zonensteuer-leistung-lohnt-sich: e1 ≤ e2*  
*⇒ e1 - zonensteuer zs spitzensteuer e1 ≤ e2 - zonensteuer zs spitzensteuer e2*

**definition** *steuerzonen2022* :: (nat × percentage) list **where**

```
steuerzonen2022 ≡ [
  (10347, percentage 0),
  (4579, percentage 0.14),
  (43670, percentage 0.2397),
  (219229, percentage 0.42)
]
```

**fun** *steuerzonenAbs* :: (nat × percentage) list ⇒ (nat × percentage) list **where**

```
steuerzonenAbs [] = []
| steuerzonenAbs ((zone, prozent)#zonen) =
  (zone,prozent)#(map (λ(z,p). (zone+z, p)) (steuerzonenAbs zonen))
```

**definition** *steuerbuckets2022* :: (nat × percentage) list **where**

```
steuerbuckets2022 ≡ [
  (10347, percentage 0),
  (14926, percentage 0.14),
  (58596, percentage 0.2397),
  (277825, percentage 0.42)
]
```

**lemma** *steuerbuckets2022*: *steuerbuckets2022* = *steuerzonenAbs steuerzonen2022*

**fun** *wfSteuerbuckets* :: (nat × percentage) list ⇒ bool **where**

```
wfSteuerbuckets [] = True
| wfSteuerbuckets [bs] = True
| wfSteuerbuckets ((b1, p1)#(b2, p2)#bs) ⇔ b1 ≤ b2 ∧ wfSteuerbuckets ((b2,p2)#bs)
```

**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
```

**lemma** *wfSteuerbucketsConsD*: *wfSteuerbuckets* (z#zs) ⇒ *wfSteuerbuckets* zs

**lemma** *wfSteuerbucketsMapD*:

```
wfSteuerbuckets (map (λ(z, y). (zone + z, y)) zs) ⇒ wfSteuerbuckets zs
```

**lemma** *mapHelp1*: *wfSteuerbuckets* zs ⇒

```
(map ((λ(s, y). (s - x, y)) ∘ (λ(z, y). (x + z, y)))) zs = zs
```

**lemma** *bucketsteuerAbs-zonensteuer*:

```
wfSteuerbuckets (steuerzonenAbs zs) ⇒
  bucketsteuerAbs (steuerzonenAbs zs) spitzensteuer e
```

$= \text{zonensteuer } \text{zs } \text{spitzensteuer } e$

**definition** *einkommenssteuer* :: nat  $\Rightarrow$  nat **where**  
*einkommenssteuer einkommen*  $\equiv$   
*floor (bucketsteuerAbs steuerbuckets2022 (percentage 0.45) einkommen)*

**value**  $\langle \text{einkommenssteuer } 10 \rangle$   
**lemma**  $\langle \text{einkommenssteuer } 10 = 0 \rangle$   
**lemma**  $\langle \text{einkommenssteuer } 10000 = 0 \rangle$   
**lemma**  $\langle \text{einkommenssteuer } 14000 = \text{floor } ((14000 - 10347) * 0.14) \rangle$   
**lemma**  $\langle \text{einkommenssteuer } 20000 =$   
 $\text{floor } ((14926 - 10347) * 0.14 + (20000 - 14926) * 0.2397) \rangle$   
**value**  $\langle \text{einkommenssteuer } 40000 \rangle$   
**value**  $\langle \text{einkommenssteuer } 60000 \rangle$

**lemma** *einkommenssteuer*:  
*einkommenssteuer einkommen* =  
*floor (zonensteuer steuerzonen2022 (percentage 0.45) einkommen)*

**interpretation** *steuersystem*  
**where** *steuer* = *einkommenssteuer*

## 9 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  $\Rightarrow$  int)* — einkommen: einkommen jeder Person (im Zweifel 0).

**fun** *steuerlast* :: *person*  $\Rightarrow$  *steuerwelt* *handlung*  $\Rightarrow$  *int* **where**  
*steuerlast p (Handlung vor nach) = ((get-einkommen vor) p) - ((get-einkommen nach) p)*

**fun** *brutto* :: *person*  $\Rightarrow$  *steuerwelt* *handlung*  $\Rightarrow$  *int* **where**  
*brutto p (Handlung vor nach) = (get-einkommen vor) p*  
**fun** *netto* :: *person*  $\Rightarrow$  *steuerwelt* *handlung*  $\Rightarrow$  *int* **where**  
*netto p (Handlung vor nach) = (get-einkommen nach) p*

Default: *DEFAULT* entspricht keinem Einkommen. Um Beispiele einfacher zu schreiben.

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



**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**  $\langle$ *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*  $\in$  *mehrverdiener* *ich* *handlung*.  
*steuerlast* *ich* *handlung*  $\leq$  *steuerlast* *p* *handlung*)  
 $\wedge$  ( $\forall$  *p*  $\in$  *mehrverdiener* *ich* *handlung*.  
*netto* *ich* *handlung*  $\leq$  *netto* *p* *handlung*)  
)

**fun** *delta-steuerwelt* :: (*steuerwelt*, *person*, *int*) *delta* **where**  
*delta-steuerwelt* (*Handlung* *vor* *nach*) =  
*Aenderung*.*delta-num-fun* (*Handlung* (*get-einkommen* *vor*) (*get-einkommen* *nach*))

**definition** *sc*  $\equiv$  *SimConsts*  
*Alice*  
*maxime-steuern*  
(*printable-case-law-ableiten-absolut* ( $\lambda$ *w*. *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**  $\langle$ *beispiel-case-law* (*HandlungF* ( $\lambda$ *ich* *welt*. *welt*)) $\rangle$

**lemma**  $\langle$ *beispiel-case-law'* (*HandlungF* ( $\lambda$ *ich* *welt*. *welt*)) =  
*Gesetz* {(*Paragraph* 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**  $\langle$ *beispiel-case-law* (*HandlungF* *ich-zahle-1-steuer*) =  
*Gesetz*

$\{(Paragraph\ 1,$   
*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*  
 $\{(Paragraph\ 1, Rechtsnorm\ (Tatbestand\ [Verliert\ Alice\ 1])$   
 $(Rechtsfolge\ Verbot))\}\rangle$*

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*  $((\lambda e. e - 1) \circ (get-einkommen\ welt))$   
**lemma**  $\langle beispiel-case-law (HandlungF\ jeder-zahle-1-steuer) =$   
*Gesetz*

$\{(Paragraph\ 3,$   
*Rechtsnorm*  
*(Tatbestand*  
 $[(Alice, 6), (Bob, 1), (Carol, - 2), (Eve, 3)],$   
 $[(Alice, 5), (Bob, 0), (Carol, - 3), (Eve, 2)])$   
*(Rechtsfolge Erlaubnis)),*  
 $(Paragraph\ 2,$   
*Rechtsnorm*  
*(Tatbestand*  
 $[(Alice, 7), (Bob, 2), (Carol, - 1), (Eve, 4)],$   
 $[(Alice, 6), (Bob, 1), (Carol, - 2), (Eve, 3)])$   
*(Rechtsfolge Erlaubnis)),*  
 $(Paragraph\ 1,$   
*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*  
 $\{(Paragraph\ 1,$   
*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*  $((\lambda e. e - steuerberechnung\ e) \circ nat \circ (get-einkommen\ welt))$

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

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

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

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

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

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

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