Formal

Cornelius Diekmann

September 23, 2022

Contents

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 ‹'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 {
  (Paragraph ''823 BGB'',
   Rechtsnorm
     (Tatbestand ''Wer vorsaetzlich 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 Eigentuemer 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.'')
  )
  }›
```

fun *neuer-paragraph :: ‹(nat, 'a, 'b) gesetz ⇒ nat prg›* **where**
‹neuer-paragraph (Gesetz G) = Paragraph ((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**

‹*hinzufuegen rn* (*Gesetz G*) =
(*if rn* ∈ (*snd* ‘ *G*) *then Gesetz G else Gesetz* (*insert* (*neuer-paragraph* (*Gesetz G*), *rn*) *G*))›

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

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

Beispiel:

lemma ‹*hinzufuegen*
(*Rechtsnorm* (*Tatbestand ''tb2''*) (*Rechtsfolge Verbot*))
(*Gesetz* {(*Paragraph 1*, (*Rechtsnorm* (*Tatbestand ''tb1''*) (*Rechtsfolge Erlaubnis*)))}) =
Gesetz
{(*Paragraph 2*, *Rechtsnorm* (*Tatbestand ''tb2''*) (*Rechtsfolge Verbot*)),
(*Paragraph 1*, *Rechtsnorm* (*Tatbestand ''tb1''*) (*Rechtsfolge Erlaubnis*))}›

3 Kant’s Kategorischer Imperativ

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

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* ‹*'person* ⇒ *'world handlung* ⇒ *bool*›

Beispiel

definition *maxime-mir-ist-alles-recht* :: ‹(*'person*, *'world*) *maxime*› **where**
‹*maxime-mir-ist-alles-recht* ≡ *Maxime* (λ- -. *True*)›

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* :: ‹*'person set*› **where** ‹*bevoelkerung* ≡ *UNIV*›
definition *wenn-jeder-so-handelt*
:: ‹*'world* ⇒ (*'person*, *'world*) *handlungF* ⇒ (*'world handlung*) *set*›
where
‹*wenn-jeder-so-handelt welt handlung* ≡
(λ*handelnde-person*. *handeln handelnde-person welt handlung*) ‘ *bevoelkerung*›
fun *was-wenn-jeder-so-handelt-aus-sicht-von*
:: ‹*'world* ⇒ (*'person*, *'world*) *handlungF* ⇒ (*'person*, *'world*) *maxime* ⇒ *'person* ⇒ *bool*›
where
‹*was-wenn-jeder-so-handelt-aus-sicht-von welt handlung* (*Maxime m*) *betroffene-person* =
(∀ *h* ∈ *wenn-jeder-so-handelt welt handlung*. *m betroffene-person h*)›

definition *teste-maxime* ::
‹*'world* ⇒ (*'person*, *'world*) *handlungF* ⇒ (*'person*, *'world*) *maxime* ⇒ *bool*› **where**
‹*teste-maxime welt handlung maxime* ≡
∀ *p* ∈ *bevoelkerung*. *was-wenn-jeder-so-handelt-aus-sicht-von welt handlung maxime p*›

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*:
‹*teste-maxime welt handlung* (*Maxime m*) =
(∀ *p1*∈*bevoelkerung*. ∀ *p2*∈*bevoelkerung*. *m p1* (*handeln p2 welt handlung*))›
lemma ‹*teste-maxime welt handlung* (*Maxime m*) =
(∀ (*p1*,*p2*)∈*bevoelkerung*×*bevoelkerung*. *m p1* (*handeln p2 welt handlung*))›

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* ≡ *case maxime of Maxime m* ⇒ *list-all* (λ(*p*, *x*). *m p* (*handeln x welt handlung*)) (*List.product bevoelk 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* ⇒ *'printable-world*) ⇒ *'world* ⇒
      (*'person*, *'world*) *handlungF* ⇒ (*'person*, *'world*) *maxime*
      ⇒ ((*'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*.
      ¬*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* = {} ⟷ *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* (λ(*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, erfuellt die Handlung jemanden 3 zu geben die Maxime.

lemma ‹*teste-maxime*
        [*Alice* ↦ (*0*::*nat*), *Bob* ↦ *0*, *Carol* ↦ *0*, *Eve* ↦ *0*]
        (*HandlungF* (λ*person welt*. *welt*(*person* ↦ *3*)))
        (*Maxime* (λ*person handlung*.
            (*the* ((*vorher handlung*) *person*)) ≤ (*the* ((*nachher handlung*) *person*))))›
lemma ‹*debug-maxime show-map*
        [*Alice* ↦ (*0*::*nat*), *Bob* ↦ *0*, *Carol* ↦ *0*, *Eve* ↦ *0*]
        (*HandlungF* (λ*person welt*. *welt*(*person* ↦ *3*)))
        (*Maxime* (λ*person handlung*.
            (*the* ((*vorher handlung*) *person*)) ≤ (*the* ((*nachher handlung*) *person*))))
  = {}›

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

lemma ‹¬ *teste-maxime*
[*Alice* ↦ (*0*::*nat*), *Bob* ↦ *4*, *Carol* ↦ *0*, *Eve* ↦ *0*]
(*HandlungF* (λ*person welt*. *welt*(*person* ↦ *3*)))
(*Maxime* (λ*person handlung*.
(*the* ((*vorher handlung*) *person*)) ≤ (*the* ((*nachher handlung*) *person*))))›
lemma ‹*debug-maxime show-map*
[*Alice* ↦ (*0*::*nat*), *Bob* ↦ *4*, *Carol* ↦ *0*, *Eve* ↦ *0*]
(*HandlungF* (λ*person welt*. *welt*(*person* ↦ *3*)))
(*Maxime* (λ*person handlung*.
(*the* ((*vorher handlung*) *person*)) ≤ (*the* ((*nachher handlung*) *person*))))
= {*VerletzteMaxime Bob Bob*
(*Handlung* [(*Alice*, *0*), (*Bob*, *4*), (*Carol*, *0*), (*Eve*, *0*)]
[(*Alice*, *0*), (*Bob*, *3*), (*Carol*, *0*), (*Eve*, *0*)])}›

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 Typsignatir auf, was yu 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* =
‹*'world handlung* ⇒ *sollensanordnung* ⇒ (*'a*, *'b*) *rechtsnorm*›

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 zuammen:
„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 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* (λ- -. *True*)) (λ*h s*. *Rechtsnorm* (*Tatbestand h*) (*Rechtsfolge ''count''*)))
10 (*HandlungF* (λ*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 ‹∃ *tb rf*.
simulateOne
(*SimConsts person maxime gesetz-ableiten*)
1 handlungF
initialwelt
(*Gesetz* {})
= *Gesetz* {(*Paragraph 1*, *Rechtsnorm* (*Tatbestand tb*) (*Rechtsfolge rf*))}›

5 Gesetze

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

5.1 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*)

5.2 Case Law Relativ

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

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

6 Beispiel: Zahlenwelt

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

datatype *zahlenwelt* = *Zahlenwelt*
person ⇒ *int option* — besitz: Besitz jeder Person.

fun *gesamtbesitz* :: *zahlenwelt* ⇒ *int* **where**
gesamtbesitz (*Zahlenwelt besitz*) = *sum-list* (*List.map-filter besitz Enum.enum*)

lemma *gesamtbesitz* (*Zahlenwelt* [*Alice* ↦ *4*, *Carol* ↦ *8*]) = *12*
lemma *gesamtbesitz* (*Zahlenwelt* [*Alice* ↦ *4*, *Carol* ↦ *4*]) = *8*

```
fun meins :: person ⇒ zahlenwelt ⇒ int where
  meins p (Zahlenwelt besitz) = the-default (besitz p) 0

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

Die folgende Handlung erschafft neuen Besitz aus dem Nichts:

```
fun erschaffen :: nat ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt where
  erschaffen i p (Zahlenwelt besitz) =
    Zahlenwelt
      (case besitz p
        of None ⇒ besitz(p ↦ int i)
         | Some b ⇒ besitz(p ↦ b + int i))
```

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

```
fun individueller-fortschritt :: person ⇒ zahlenwelt handlung ⇒ bool where
  individueller-fortschritt p (Handlung vor nach) ⟷ (meins p vor) ≤ (meins p nach)
definition maxime-zahlenfortschritt :: (person, zahlenwelt) maxime where
  maxime-zahlenfortschritt ≡ Maxime (λich. individueller-fortschritt ich)


definition initialwelt ≡ Zahlenwelt [Alice ↦ 5, Bob ↦ 10]
```

Wir nehmen an unsere handelnde Person ist *Alice*.

```
definition beispiel-case-law-absolut maxime handlung ≡
  simulateOne
    (SimConsts
      Alice
      maxime
      (printable-case-law-ableiten-absolut show-zahlenwelt))
    10 handlung initialwelt (Gesetz {})
definition beispiel-case-law-relativ maxime handlung ≡
  simulateOne
    (SimConsts
      Alice
      maxime
      (case-law-ableiten-relativ delta-zahlenwelt))
    20 handlung initialwelt (Gesetz {})
```

6.1 Alice erzeugt 5 Wohlstand für sich.

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
  {(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))}›
```

6.2 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 ⇒ zahlenwelt handlung ⇒ bool where
  individueller-strikter-fortschritt p (Handlung vor nach) ⟷ (meins p vor) < (meins p nach)
```

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

```
lemma ‹beispiel-case-law-relativ
        (Maxime (λich. 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*)])
∈ *debug-maxime show-zahlenwelt initialwelt*
(*HandlungF* (*erschaffen 5*)) (*Maxime* (λ*ich*. *individueller-strikter-fortschritt ich*))›

6.3 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* ⇒ *bool* **where**
globaler-strikter-fortschritt (*Handlung vor nach*) ⟷ (*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* (λ*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* (λ*ich*. *globaler-strikter-fortschritt*))
(*HandlungF* (*erschaffen 0*)) =
Gesetz {(*Paragraph 1*, *Rechtsnorm* (*Tatbestand* []) (*Rechtsfolge Verbot*))}›

value‹*debug-maxime show-zahlenwelt initialwelt*
(*HandlungF* (*erschaffen 0*)) (*Maxime* (λ*ich*. *globaler-strikter-fortschritt*))›

6.4 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).

Helper

definition *floor* :: *real* ⇒ *nat* **where**
floor x ≡ *nat* ⌊*x*⌋

lemma *floorD*: $a \leq b \Longrightarrow \mathit{floor}\ a \leq \mathit{floor}\ b$

lemma *floor-minusD*:
  fixes $a :: \mathit{nat}$ **and** $a' :: \mathit{real}$
  shows $a \leq b \Longrightarrow a - a' \leq b - b' \Longrightarrow a - \mathit{floor}\ a' \leq b - \mathit{floor}\ b'$

7 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* ⇒ *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 ≡ *percentage* ((*steuer einkommen*) / *einkommen*)
end

Beispiel

definition *beispiel-25prozent-steuer* :: *nat* ⇒ *nat* **where**
  beispiel-25prozent-steuer e ≡ *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 $\leq$ *9888* $\Longrightarrow$ *steuer einkommen* = *0*

begin

end

fun *zonensteuer* :: (*nat* $\times$ *percentage*) *list* $\Rightarrow$ *percentage* $\Rightarrow$ *nat* $\Rightarrow$ *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* $\leq$ *e2*
$\Longrightarrow$ *zonensteuer zs spitzensteuer e1* $\leq$ *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* $\geq$ *0*

Steuer kann nicht höher sein als das Einkommen.

lemma *zonensteuer-limit*: *zonensteuer ls spitzensteuer einkommen* $\leq$ *einkommen*

lemma *zonensteuer-leistung-lohnt-sich*: *e1* $\leq$ *e2*
$\Longrightarrow$ *e1* $-$ *zonensteuer zs spitzensteuer e1* $\leq$ *e2* $-$ *zonensteuer zs spitzensteuer e2*

definition *steuerzonen2022* :: (*nat* $\times$ *percentage*) *list* **where**
steuerzonen2022 $\equiv$ [
(*10347*, *percentage 0*),
(*4579*, *percentage 0.14*),
(*43670*, *percentage 0.2397*),
(*219229*, *percentage 0.42*)
]

fun *steuerzonenAbs* :: (*nat* $\times$ *percentage*) *list* $\Rightarrow$ (*nat* $\times$ *percentage*) *list* **where**
steuerzonenAbs [] = []
| *steuerzonenAbs* ((*zone*, *prozent*)#*zonen*) =
(*zone*,*prozent*)#(*map* (λ(*z*,*p*). (*zone*+*z*, *p*)) (*steuerzonenAbs zonen*))

definition *steuerbuckets2022* :: (*nat* $\times$ *percentage*) *list* **where**
steuerbuckets2022 $\equiv$ [
(*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
      = zonensteuer zs spitzensteuer e


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›

lemma einkommenssteuer:
```

```
  einkommenssteuer einkommen =
    floor (zonensteuer steuerzonen2022 (percentage 0.45) einkommen)


interpretation steuersystem
  where steuer = einkommenssteuer
```

8 Beispiel: Steuern

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

```
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
```

Default: kein Einkommen. Um Beispiele einfacher zu schreiben.

```
definition KE :: person ⇒ int where
  KE ≡ λp. 0

lemma ‹steuerlast Alice (Handlung (Steuerwelt (KE(Alice:=8))) (Steuerwelt (KE(Alice:=5)))) = 3›
lemma ‹steuerlast Alice (Handlung (Steuerwelt (KE(Alice:=8))) (Steuerwelt (KE(Alice:=0)))) = 8›
lemma ‹steuerlast Bob   (Handlung (Steuerwelt (KE(Alice:=8))) (Steuerwelt (KE(Alice:=5)))) = 0›
lemma ‹steuerlast Alice (Handlung (Steuerwelt (KE(Alice:=−3))) (Steuerwelt (KE(Alice:=−4)))) = 1›
lemma ‹steuerlast Alice (Handlung (Steuerwelt (KE(Alice:=1))) (Steuerwelt (KE(Alice:=−1)))) = 2›

fun mehrverdiener :: person ⇒ steuerwelt handlung ⇒ person set where
  mehrverdiener ich (Handlung vor nach) = {p. (get-einkommen vor) p ≥ (get-einkommen vor) ich}

lemma ‹mehrverdiener Alice
      (Handlung (Steuerwelt (KE(Alice:=8, Bob:=12, Eve:=7))) (Steuerwelt (KE(Alice:=5))))
     = {Alice, Bob}›


definition maxime-steuern :: (person, steuerwelt) maxime where
  maxime-steuern ≡ Maxime
     (λich handlung.
        (∀p∈mehrverdiener ich handlung.
            steuerlast ich handlung ≤ steuerlast p handlung)
      ∧ (∀p∈mehrverdiener ich handlung.
            netto ich handlung ≤ 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 ≡ SimConsts
    Alice
    maxime-steuern
    (printable-case-law-ableiten-absolut (λw. show-fun (get-einkommen w)))
definition sc' ≡ SimConsts
    Alice
    maxime-steuern
    (case-law-ableiten-relativ delta-steuerwelt)

definition initialwelt ≡ Steuerwelt (KE(Alice:=8, Bob:=3, Eve:= 5))

definition beispiel-case-law h ≡ simulateOne sc 3 h initialwelt (Gesetz {})
definition beispiel-case-law' h ≡ simulateOne sc' 20 h initialwelt (Gesetz {})
```

Keiner zahlt steuern: funktioniert

```
value ‹beispiel-case-law (HandlungF (λich welt. welt))›
lemma ‹beispiel-case-law' (HandlungF (λich welt. welt)) =
  Gesetz {(Paragraph 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Erlaubnis))}›
```

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

```
definition ich-zahle-1-steuer ich welt ≡
  Steuerwelt ((get-einkommen welt)(ich := ((get-einkommen welt) ich) − 1))
lemma ‹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 ‹beispiel-case-law' (HandlungF ich-zahle-1-steuer) =
  Gesetz
  {(Paragraph 1, Rechtsnorm (Tatbestand [Verliert Alice 1])
                            (Rechtsfolge Verbot))}›
```

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 ≡
  Steuerwelt ((λe. e − 1) ∘ (get-einkommen welt))
lemma ‹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))}›
lemma ‹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))}›
```

Jetzt kommt die Steuern.thy ins Spiel.

Bei dem geringen Einkommen zahlt keiner Steuern.

```
definition jeder-zahlt steuerberechnung ich welt ≡
  Steuerwelt ((λe. e − steuerberechnung e) ∘ nat ∘ (get-einkommen welt))

definition jeder-zahlt-einkommenssteuer ≡ jeder-zahlt einkommenssteuer
lemma ‹beispiel-case-law (HandlungF jeder-zahlt-einkommenssteuer ) =
  Gesetz
  {(Paragraph 1,
    Rechtsnorm
     (Tatbestand
       ([(Alice, 8), (Bob, 3), (Carol, 0), (Eve, 5)],
        [(Alice, 8), (Bob, 3), (Carol, 0), (Eve, 5)]))
     (Rechtsfolge Erlaubnis))}›

lemma ‹simulateOne
  sc' 1
  (HandlungF jeder-zahlt-einkommenssteuer)
  (Steuerwelt (KE(Alice:=10000, Bob:=14000, Eve:= 20000)))
  (Gesetz {})
  =
  Gesetz
  {(Paragraph 1,
```

Rechtsnorm (*Tatbestand* [*Verliert Bob 511*, *Verliert Eve 1857*])
(*Rechtsfolge Erlaubnis*))}›

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

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

lemma $a \leq x \Longrightarrow$ *int* $x -$ *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 (∀ *einkommen*. *steuersystem-impl einkommen* $\leq$ *einkommen*) $\Longrightarrow$
(∀ *einkommen*. *einkommen* $\leq$ *9888* $\longrightarrow$ *steuersystem-impl einkommen* $=$ *0*) $\Longrightarrow$
∀ *welt*. *teste-maxime welt* (*HandlungF* (*jeder-zahlt steuersystem-impl*)) *maxime-steuern*
$\Longrightarrow$ *steuersystem steuersystem-impl*