Formal

Cornelius Diekmann

September 25, 2022

Contents

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‹$3 = 2+1$›

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 $'\alpha$. 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**‹$3 = 2+1$› 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* ⇒ *nat* **where**
beispielfunktion n = *n* + *10*

Funktionsaufrufe funktionieren ohne Klammern.

lemma ‹*beispielfunktion 32* = *42*›

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* ⇒ *nat* ⇒ *nat* **where**
addieren a b = *a* + *b*

lemma ‹*addieren 32 10* = *42*›

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 ‹*addieren 10* = *beispielfunktion*›

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 ‹$\{0,2,4,6,8,10,12\} \subseteq \{n::int.\ n\ mod\ 2 = 0\}$›

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 Handlung

Beschreibt Handlungen als Änderung der Welt. Unabhängig von der handelnden Person. Wir beschreiben nur vergangene bzw. mögliche Handlungen und deren Auswirkung.

Eine Handlung ist reduziert auf deren Auswirkung. Intention oder Wollen ist nicht modelliert, da wir irgendwie die geistige Welt mit der physischen Welt verbinden müssen und wir daher nur messbare Tatsachen betrachten können.

Handlungen können Leute betreffen. Handlungen können aus Sicht Anderer wahrgenommen werden. Ich brauche nur Welt vorher und Welt nachher. So kann ich handelnde Person und beobachtende Person trennen.

datatype *'world handlung* = *Handlung* (*vorher*: ‹*'world*›) (*nachher*: ‹*'world*›)

Handlung als Funktion gewrapped. Diese abstrakte Art eine Handlung zu modelliert so ein bisschen die Absicht oder Intention.

datatype (*'person*, *'world*) *handlungF* = *HandlungF* ‹*'person* ⇒ *'world* ⇒ *'world*›

Von Außen können wir Funktionen nur extensional betrachten, d.h. Eingabe und Ausgabe anschauen. Die Absicht die sich in einer Funktion verstecken kann ist schwer zu erkennen. Dies deckt sich ganz gut damit, dass Isabelle standardmäßig Funktionen nicht printet. Eine (*'person*, *'world*) *handlungF* kann nicht geprinted werden!

fun *handeln* :: ‹*'person* $\Rightarrow$ *'world* $\Rightarrow$ (*'person*, *'world*) *handlungF* $\Rightarrow$ *'world handlung*› **where**
‹*handeln handelnde-person welt* (*HandlungF h*) = *Handlung welt* (*h handelnde-person welt*)›

Beispiel, für eine Welt die nur aus einer Zahl bestehtÖ Wenn die Zahl kleiner als 9000 ist erhöhe ich sie, ansonsten bleibt sie unverändert.

definition ‹*beispiel-handlungf* $\equiv$ *HandlungF* (λ*p n. if n* < *9000 then n+1 else n*)›

Da Funktionen nicht geprintet werden können, sieht *beispiel-handlungf* so aus: *HandlungF* -

3.1 Interpretation: Gesinnungsethik vs. Verantwortungethik

Sei eine Ethik eine Funktion, welche einem beliebigen $'\alpha$ eine Bewertung Gut = *True*, Schlecht = *False* zuordnet.

- Eine Ethik hat demnach den Typ: $'\alpha \Rightarrow$ *bool*.

Laut https://de.wikipedia.org/wiki/Gesinnungsethik ist eine Gesinnugsethik "[..] eine der moralischen Theorien, die Handlungen nach der Handlungsabsicht [...] bewertet, und zwar ungeachtet der nach erfolgter Handlung eingetretenen Handlungsfolgen."

- Demnach ist eine Gesinnungsethik: (*'person*, *'world*) *handlungF* $\Rightarrow$ *bool*.

Nach https://de.wikipedia.org/wiki/Verantwortungsethik steht die Verantwortungsethik dazu im strikten Gegensatz, da die Verantwortungsethik "in der Bewertung des Handelns die Verantwortbarkeit der *tatsächlichen Ergebnisse* betont."

- Demnach ist eine Verantwortungsethik: *'world handlung* $\Rightarrow$ *bool*.

Da *handeln* eine Handlungsabsicht (*'person*, *'world*) *handlungF* in eine konkrete Änderung der Welt *'world handlung* überführt, können wie die beiden Ethiktypen miteinander in Verbindungs setzen. Wir sagen, eine Gesinnungsethik und eine Verantwortungsethik sind konsistent, genau dann wenn für jede Handlungsabsicht, die Gesinnungsethik die Handlungsabsicht genau so bewertet, wie die Verantwortungsethik die Handlungsabsicht bewerten würde, wenn die die Handlungsabsicht in jeder möglichen Welt und als jede mögliche handelnde Person tatsächlich ausführt wird und die Folgen betrachtet werden:

definition *gesinnungsethik-verantwortungsethik-konsistent*
:: ((*'person*, *'world*) *handlungF* $\Rightarrow$ *bool*) $\Rightarrow$ (*'world handlung* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
gesinnungsethik-verantwortungsethik-konsistent gesinnungsethik verantwortungsethik $\equiv$
$\forall$ *handlungsabsicht*.
gesinnungsethik handlungsabsicht $\longleftrightarrow$
($\forall$ *person welt*. *verantwortungsethik* (*handeln person welt handlungsabsicht*))

Ich habe kein Beispiel für eine Gesinnungsethik und eine Verantwortungsethik, die tatsächlich konsistent sind.

4 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 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.'')
  ),
  (§ ''985 BGB'',
   Rechtsnorm
     (Tatbestand ''Der Eigentuemer 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
  ‹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))
```

$(Gesetz\ \{(\S\ 1,\ (Rechtsnorm\ (Tatbestand\ ''tb1'')\ (Rechtsfolge\ Erlaubnis)))\}) =$
$Gesetz$
$\{(\S\ 2,\ Rechtsnorm\ (Tatbestand\ ''tb2'')\ (Rechtsfolge\ Verbot)),$
$(\S\ 1,\ Rechtsnorm\ (Tatbestand\ ''tb1'')\ (Rechtsfolge\ Erlaubnis))\}$›

5 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.

6 Beispiel Person

Eine Beispielbevölkerung.

datatype *person* = *Alice* | *Bob* | *Carol* | *Eve*

Unsere Bevölkerung ist sehr endlich:

lemma *UNIV-person*: ‹*UNIV* = {*Alice*, *Bob*, *Carol*, *Eve*}›

Wir werden unterscheiden:

- *'person*: generischer Typ, erlaub es jedes Modell einer Person und Bevölkerung zu haben.
- *person*: Unser minimaler Beispielstyp, bestehend aus *Alice*, *Bob*, ...

7 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 handlungsabsicht* ≡
   (λ*handelnde-person*. *handeln handelnde-person welt handlungsabsicht*) ‘ *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 handlungsabsicht* (*Maxime m*) *betroffene-person* =
   (∀ *h* ∈ *wenn-jeder-so-handelt welt handlungsabsicht*. *m betroffene-person h*)›

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

$\langle$*teste-maxime welt handlungsabsicht* (*Maxime m*) =
    ($\forall$ *p1*$\in$*bevoelkerung*. $\forall$ *p2*$\in$*bevoelkerung*. *m p1* (*handeln p2 welt handlungsabsicht*))$\rangle$
lemma $\langle$*teste-maxime welt handlungsabsicht* (*Maxime m*) =
    ($\forall$ (*p1*,*p2*)$\in$*bevoelkerung*$\times$*bevoelkerung*. *m p1* (*handeln p2 welt handlungsabsicht*))$\rangle$

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 handlungsabsicht maxime* $\equiv$ *case maxime of Maxime m* $\Rightarrow$ *list-all* (λ(*p*, *x*). *m p* (*handeln x welt handlungsabsicht*)) (*List.product bevoelk bevoelk*).

7.1 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
    $\langle$*'person opfer*$\rangle$ — verletzt für; das Opfer
    $\langle$*'person taeter*$\rangle$ — handelnde Person; der Täter
    $\langle$*'world handlung*$\rangle$ — 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 handlungsabsicht (*Maxime m*) =
    {*VerletzteMaxime*
      (*Opfer p1*) (*Taeter p2*)
      (*map-handlung print-world* (*handeln p2 welt handlungsabsicht*)) | *p1 p2*.
        $\neg$*m p1* (*handeln p2 welt handlungsabsicht*)}

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

lemma *debug-maxime print-world welt handlungsabsicht maxime* = {}
    $\longleftrightarrow$ *teste-maxime welt handlungsabsicht maxime*

7.2 Beispiel

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

lemma $\langle$*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* (*Opfer Bob*) (*Taeter Bob*)
  (*Handlung* [(*Alice*, *0*), (*Bob*, *4*), (*Carol*, *0*), (*Eve*, *0*)]
    [(*Alice*, *0*), (*Bob*, *3*), (*Carol*, *0*), (*Eve*, *0*)])}›

8 Kategorischer Imperativ

8.1 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 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.

8.2 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 handlungsabsicht maxime gesetz-ableiten gesetz ≡
    let soll-handeln = if teste-maxime welt handlungsabsicht maxime
                       then
                         Erlaubnis
                       else
                         Verbot in
      (
        soll-handeln,
        hinzufuegen (gesetz-ableiten (handeln ich welt handlungsabsicht) soll-handeln) gesetz
      )›
```

9 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* ⇒ *int* allgmein zu arbeiten.

Default: Standardmäßig hat jede Person *0*:

definition *DEFAULT* :: *person* ⇒ *int* **where**
  DEFAULT ≡ λ*p*. *0*

Beispiel:

lemma ‹(*DEFAULT*(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)) *Bob* = *3*›

Beispiel mit fancy Syntax:

lemma ‹🌍[*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*] *Bob* = *3*›

lemma ‹*show-fun* 🌍[*Alice* := *4*, *Carol* := *4*] = [(*Alice*, *4*), (*Bob*, *0*), (*Carol*, *4*), (*Eve*, *0*)]›
lemma ‹*show-num-fun* 🌍[*Alice* := *4*, *Carol* := *4*] = [(*Alice*, *4*), (*Carol*, *4*)]›

abbreviation *num-fun-add-syntax* (- '(- += -')) **where**
  f(*p* += *n*) ≡ (*f*(*p* := (*f p*) + *n*))

abbreviation *num-fun-minus-syntax* (- ′(- −= -′)) **where**
  $f(p \mathrel{-=} n) \equiv (f(p := (f\ p) - n))$

lemma ‹(🌍[*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*])(*Bob* += *4*) *Bob* = *7*›
lemma ‹(🌍[*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*])(*Bob* −= *4*) *Bob* = −*1*›

lemma fixes *n*:: *int* **shows** $f(p \mathrel{+=} n)(p \mathrel{-=} n) = f$

10 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
  {(§ *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 ‹∃ *tb rf*.
  simulateOne
    (*SimConsts person maxime gesetz-ableiten*)
    1 handlungsabsicht
    initialwelt
    (*Gesetz* {})
  = *Gesetz* {(§ *1*, *Rechtsnorm* (*Tatbestand tb*) (*Rechtsfolge rf*))}›

11 Gesetze

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

11.1 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)
```

11.2 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)
```

12 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*
  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*

12.1 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.

12.2 Setup

definition *initialwelt* ≡ *Zahlenwelt* 🌍[*Alice* := *5*, *Bob* := *10*]

Wir nehmen an unsere handelnde Person ist *Alice*.

definition *beispiel-case-law-absolut maxime handlungsabsicht* ≡
  simulateOne

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

12.3 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 ⇒ 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)
```

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 ‹*beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*erschaffen 5*)) =
Gesetz
{(§ *1*, *Rechtsnorm* (*Tatbestand* [*Gewinnt Alice 5*]) (*Rechtsfolge Erlaubnis*))}›

12.4 Kleine Änderung in der Maxime

In der Maxime *individueller-fortschritt* hatten wir *meins p vor* ≤ *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 {(§ *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* (*Opfer Bob*) (*Taeter Alice*)
(*Handlung* [(*Alice*, *5*), (*Bob*, *10*)] [(*Alice*, *10*), (*Bob*, *10*)])
∈ *debug-maxime show-zahlenwelt initialwelt*
(*HandlungF* (*erschaffen 5*)) (*Maxime* (*λich. individueller-strikter-fortschritt ich*))›

12.5 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 ⇒ 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 (λich. globaler-strikter-fortschritt))
        (HandlungF (erschaffen 5)) =
  Gesetz {(§ 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 (λich. globaler-strikter-fortschritt))
        (HandlungF (erschaffen 0)) =
  Gesetz {(§ 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Verbot))}›
```

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

```
lemma ‹beispiel-case-law-relativ
        maxime-zahlenfortschritt
        (HandlungF (erschaffen 0)) =
  Gesetz {(§ 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Erlaubnis))}›
```

Wir können die Maxime für globalen Fortschritt etwas lockern:

```
fun globaler-fortschritt :: zahlenwelt handlung ⇒ bool where
 globaler-fortschritt (Handlung vor nach) ⟷ (gesamtbesitz vor) ≤ (gesamtbesitz nach)
```

Untätigkeit ist nun auch hier erlaubt:

```
lemma‹beispiel-case-law-relativ
          (Maxime (λich. globaler-fortschritt))
          (HandlungF (erschaffen 0))
  =
  Gesetz {(§ 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Erlaubnis))}›
```

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

```
lemma‹beispiel-case-law-relativ
        (Maxime (λich. globaler-fortschritt))
        (HandlungF (stehlen 5 Bob))
  =
  Gesetz
  {(§ 1, Rechtsnorm (Tatbestand [Gewinnt Alice 5, Verliert Bob 5]) (Rechtsfolge Erlaubnis))}›
```

12.6 Alice stiehlt 5

Zurück zur einfachen *maxime-zahlenfortschritt*.

Stehlen ist verboten:

lemma ‹*beispiel-case-law-relativ maxime-zahlenfortschritt* (*HandlungF* (*stehlen 5 Bob*)) =
Gesetz
{(§ *1*, *Rechtsnorm* (*Tatbestand* [*Gewinnt Alice 5*, *Verliert Bob 5*]) (*Rechtsfolge Verbot*))}›

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

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

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

lemma ‹*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*)])
}›

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 ‹*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*))}›

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

12.7 Schenken

Es ist *Alice* verboten, etwas zu verschenken:

lemma‹*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.

12.8 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*
= (λ*h*. *case h of Handlung vor nach* ⇒ (*meins Alice vor*) ≤ (*meins Alice nach*))

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

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

13 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 *steuer* Funktion arbeitet auf natürlichen Zahlen. Wir nehmen an, dass einfach immer auf ganze Geldbeträge gerundet wird. Wie im deutschen System.

Die **locale** einhält 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 ≡ *percentage* ((*steuer einkommen*) / *einkommen*)
end

Beispiel. Die *steuer* Funktion sagt, man muss 25 Prozent Steuern zahlen:

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*

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

- Wer mehr Einkommen hat, muss auch mehr Steuern zahlen.
- Leistung muss sich lohnen: Wer mehr Einkommen hat muss auch nach Abzug der Steuer mehr übrig haben.
- Existenzminimum: Es gibt ein Existenzminimum, welches nicht besteuert werden darf.

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

Eigentlich hätte ich gerne noch eine weitere Anforderung. https://de.wikipedia.org/wiki/Steuerprogression sagt "Steuerprogression bedeutet das Ansteigen des Steuersatzes in Abhängigkeit vom zu versteuernden Einkommen oder Vermögen."

Formal betrachtet würde das bedeuten *einkommen-b* $\leq$ *einkommen-a* $\Longrightarrow$ (λx. *real-of-percentage* (*steuer-defs.steuersatz einkommen-b x*)) $\leq$ (λx. *real-of-percentage* (*steuer-defs.steuersatz einkommen-a x*))

Leider haben wir bereits jetzt in dem Modell eine Annahme getroffen, die es uns quasi unmöglich macht, ein Steuersystem zu implementieren, welches die Steuerprogression erfüllt. Der Grund ist, dass wir die Steuerfunktion auf ganzen Zahlen definiert haben. Aufgrund von Rundung können wir also immer Fälle haben, indem ein höheres Einkommen einen leicht geringeren Steuersatz hat als ein geringeres Einkommen. Beispielsweise bedeutet das für *beispiel-25prozent-steuer*, dass jemand mit 100 EUR Einkommen genau 25 Prozent Steuer zahlt, jemand mit 103 EUR Einkommen aber nur ca 24,3 Prozent Steuer zahlt.

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

In der Praxis sollten diese kleinen Rundungsfehler kein Problem darstellen, in diesem theoretischen Modell sorgen sie aber dafür, dass unser Steuersystem (und wir modellieren eine vereinfachte Version des deutschen Steuerystems) keine Steuerprogression erfüllt.

Die folgende Liste, basierend auf 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 × percentage) list where
  steuerbuckets2022 ≡ [
                        (10347, percentage 0),
                        (14926, percentage 0.14),
                        (58596, percentage 0.2397),
                        (277825, percentage 0.42)
                       ]
```

Für jedes Einkommen über 277825 gilt der Spitzensteuersatz von 45 Prozent. 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*)

Beispiel. Alles unter dem Existenzminimum ist steuerfrei:

lemma ‹*einkommenssteuer 10* = *0*›
lemma ‹*einkommenssteuer 10000* = *0*›

Für ein Einkommen nur knapp über dem Existenzminimum fällt sehr wenig Steuer an:

lemma ‹*einkommenssteuer 14000* = *floor* ((*14000*−*10347*)∗*0.14*)›
lemma ‹*einkommenssteuer 14000* = *511*›

Bei einem Einkommen von 20000 EUR wird ein Teil bereits mit den höheren Steuersatz der 3. Zone besteuert:

lemma ‹*einkommenssteuer 20000* = *1857*›
lemma ‹*einkommenssteuer 20000* =
floor ((*14926*−*10347*)∗*0.14* + (*20000*−*14926*)∗*0.2397*)›

Höhere Einkommen führen zu einer höheren Steuer:

lemma ‹*einkommenssteuer 40000* = *6651*›
lemma ‹*einkommenssteuer 60000* = *11698*›

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

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

14 Beispiel: Steuern

Wir nehmen eine einfach Welt an, in der jeder Person ihr Einkommen zugeordnet wird.

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 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 ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* 🌍[*Alice*:=*8*]) (*Steuerwelt* 🌍[*Alice*:=*5*])) = *3*›
lemma ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* 🌍[*Alice*:=*8*]) (*Steuerwelt* 🌍[*Alice*:=*0*])) = *8*›

lemma ‹*steuerlast Bob (Handlung (Steuerwelt* 🌏*[Alice:=8]) (Steuerwelt* 🌏*[Alice:=5])) = 0*›
lemma ‹*steuerlast Alice (Handlung (Steuerwelt* 🌏*[Alice:=−3]) (Steuerwelt* 🌏*[Alice:=−4])) = 1*›
lemma ‹*steuerlast Alice (Handlung (Steuerwelt* 🌏*[Alice:=1]) (Steuerwelt* 🌏*[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 🌏*[Alice:=8, Bob:=12, Eve:=7]) (Steuerwelt* 🌏*[Alice:=5]))*
   = {Alice, Bob}›

Folgende Maxime versucht Steuergerechtigkeit festzuschreiben:

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)
)

14.1 Setup für Beispiele

definition *initialwelt ≡ Steuerwelt* 🌏*[Alice:=8, Bob:=3, Eve:= 5]*

definition *beispiel-case-law-absolut welt steuerfun ≡*
  simulateOne
   (SimConsts
    Alice
    maxime-steuern
    (printable-case-law-ableiten-absolut (λw. show-fun (get-einkommen w))))
   3 steuerfun welt (Gesetz {})
definition *beispiel-case-law-relativ welt steuerfun ≡*
  simulateOne
   (SimConsts
    Alice
    maxime-steuern
    (case-law-ableiten-relativ delta-steuerwelt))
   1 steuerfun welt (Gesetz {})

14.2 Beispiel: Keiner Zahlt Steuern

Die Maxime ist erfüllt, da wir immer nur kleiner-gleich fordern!

lemma ‹*beispiel-case-law-relativ initialwelt (HandlungF (λich welt. welt)) =*
  Gesetz {(§ 1, Rechtsnorm (Tatbestand []) (Rechtsfolge Erlaubnis))}›

14.3 Beiepiel: Ich zahle 1 Steuer

Das funktioniert nicht:

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

Denn jeder muss Steuer zahlen! Ich finde es super spannend, dass hier faktisch ein Gleichbehandlungsgrundsatz rausfällt, ohne dass wir soewtas jemals explizit gefordert haben.

14.4 Beiepiel: Jeder zahle 1 Steuer

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-absolut initialwelt (HandlungF jeder-zahle-1-steuer) =
Gesetz
  {(§ 3,
    Rechtsnorm
     (Tatbestand
       ([(Alice, 6), (Bob, 1), (Carol, − 2), (Eve, 3)],
        [(Alice, 5), (Bob, 0), (Carol, − 3), (Eve, 2)]))
     (Rechtsfolge Erlaubnis)),
   (§ 2,
    Rechtsnorm
     (Tatbestand
       ([(Alice, 7), (Bob, 2), (Carol, − 1), (Eve, 4)],
        [(Alice, 6), (Bob, 1), (Carol, − 2), (Eve, 3)]))
     (Rechtsfolge Erlaubnis)),
   (§ 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-relativ initialwelt (HandlungF jeder-zahle-1-steuer) =
```

Gesetz
{(§ *1*,
Rechtsnorm
(*Tatbestand* [*Verliert Alice 1*, *Verliert Bob 1*, *Verliert Carol 1*, *Verliert Eve 1*])
(*Rechtsfolge Erlaubnis*))}›

14.5 Beiepiel: Vereinfachtes Deutsches Steuersystem

Jetzt kommt die Steuern.thy ins Spiel.

definition *jeder-zahlt* :: $(nat \Rightarrow nat) \Rightarrow {'a} \Rightarrow$ *steuerwelt* $\Rightarrow$ *steuerwelt* **where**
jeder-zahlt steuerberechnung ich welt $\equiv$
Steuerwelt ((λ*e*. *e* − *steuerberechnung e*) ∘ *nat* ∘ (*get-einkommen welt*))

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

Bei dem geringen Einkommen der *initialwelt* zahlt keiner Steuern.

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

Für höhere Einkommen erhalten wir plausible Werte und niemand rutscht ins negative:

lemma ‹*beispiel-case-law-relativ*
(*Steuerwelt* 🌍[*Alice*:=*10000*, *Bob*:=*14000*, *Eve*:= *20000*])
(*HandlungF jeder-zahlt-einkommenssteuer*)
=
Gesetz
{(§ *1*,
Rechtsnorm (*Tatbestand* [*Verliert Bob 511*, *Verliert Eve 1857*])
(*Rechtsfolge Erlaubnis*))}›

15 Vereinfachtes Deutsches Steuersystem vs. die Steuermaxime

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

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

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 *maxime-imp-steuersystem*:
$(\forall$ *einkommen. steuersystem-impl einkommen* $\leq$ *einkommen*$) \Longrightarrow$
$(\forall$ *einkommen. einkommen* $\leq$ *9888* $\longrightarrow$ *steuersystem-impl einkommen* $=$ *0*$) \Longrightarrow$
$\forall$ *welt. teste-maxime welt* (*HandlungF* (*jeder-zahlt steuersystem-impl*)) *maxime-steuern*
$\Longrightarrow$ *steuersystem steuersystem-impl*

Für jedes *steuersystem-impl*::*nat* $\Rightarrow$ *nat*, mit zwei weiteren Annahmen, gilt das *steuersystem* und *maxime-steuern* in der *jeder-zahlt* Implementierung äquivalent sind.

theorem
fixes *steuersystem-impl* :: *nat* $\Rightarrow$ *nat*
assumes *steuer-kleiner-einkommen*: $\forall$ *einkommen. steuersystem-impl einkommen* $\leq$ *einkommen*
and *existenzminimum*: $\forall$ *einkommen. einkommen* $\leq$ *9888* $\longrightarrow$ *steuersystem-impl einkommen* $=$ *0*
shows
$(\forall$ *welt. teste-maxime welt* (*HandlungF* (*jeder-zahlt steuersystem-impl*)) *maxime-steuern*$)$
$\longleftrightarrow$ *steuersystem steuersystem-impl*