Extensionale Interpretation des Kategorischen Imperativs

Cornelius Diekmann

December 11, 2022

Abstract

Language warning: German ahead.

Extensionale Interpretation des Kategorischen Imperativs. Persönliche Interpretation basierend auf Sekundärliteratur.

Beispiel referenz: [1]

Contents

1 Disclaimer **3**
1.1 Über den Titel . . . 3

2 Schnelleinstieg Isabelle/HOL **4**
2.1 Typen . . . 4
2.2 Beweise . . . 4
2.3 Mehr Typen . . . 4
2.4 Funktionen . . . 4
2.5 Mengen . . . 5

3 Handlung **5**
3.1 Interpretation: Gesinnungsethik vs. Verantwortungethik . . . 7

4 Kant's Kategorischer Imperativ **8**

5 Beispiel Person **8**

6 Maxime **9**
6.1 Maxime in Sinne Kants? . . . 9
6.2 Die Goldene Regel . . . 10
6.3 Maximen Debugging . . . 12
6.4 Beispiel . . . 12
6.5 Maximen Kombinieren . . . 13

7 Schleier des Nichtwissens **15**
7.1 Wohlgeformte Handlungsabsicht . . . 15
7.2 Spezialfall: Maxime und Handlungsabsichten haben nette Eigenschaften . . . 16
7.3 Wohlgeformte Maxime . . . 18

8 Kategorischer Imperativ **19**
8.1 Triviale Maximen die den Kategorischen Imperativ immer Erfüllen . . . 22
8.2 Zusammenhang Goldene Regel . . . 22
8.3 Maximen die den Kategorischen Imperativ immer Erfüllen . . . 22
8.4 Ausführbarer Beispielgenerator . . . 23
8.5 Kombination vom Maximen . . . 25
8.5.1 Konjunktion . . . 25
8.5.2 Disjunktion . . . 26

9 Utilitarismus **28**
9.1 Goldene Regel und Utilitarismus im Einklang . . . 28

10 Zahlenwelt Helper **30**

11 Beispiel: Zahlenwelt **32**
11.1 Ungültige Handlung . . . 33
11.2 Nicht-Wohlgeformte Handlungen . . . 33
11.3 Wohlgeformte Handlungen . . . 34
11.4 Maxime für individuellen Fortschritt . . . 36
11.4.1 Einzelbeispiele . . . 36
11.5 Maxime für allgemeinen Fortschritt . . . 37
11.6 Maxime für strikten individuellen Fortschritt . . . 38
11.7 Maxime für globales striktes Optimum . . . 39
11.8 Maxime für globales Optimum . . . 40
11.9 Ungültige Maxime . . . 41

12 Änderungen in Welten **41**
12.1 Deltas . . . 41
12.2 Abmachungen . . . 42

13 Beispiel: Zahlenwelt2 **44**

14 Einkommensteuergesetzgebung **51**

15 Beispiel: Steuern **54**
15.1 Setup für Beispiele . . . 56
15.2 Beispiel: Keiner Zahlt Steuern . . . 56
15.3 Beispiel: Ich zahle 1 Steuer . . . 56
15.4 Beiepiel: Jeder zahle 1 Steuer . . . 56
15.5 Beispiel: Vereinfachtes Deutsches Steuersystem . . . 56

16 Vereinfachtes Deutsches Steuersystem vs. die Steuermaxime **57**

1 Disclaimer

Ich habe

- wenig 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. Dies ist ein instabiler Development Snapshot. Er enthält sinnvolles und weniger sinnvolle Experimente!

Cheers!

1.1 Über den Titel

Der Titel lautet *Extensionale Interpretation des Kategorischen Imperativs.* Dabei sind die Wörter wie folgt zu verstehen

- *Extensional* bezieht sich hier auf den Fachbegriff der Logik https://en.wikipedia.org/wiki/Extensionality, welcher besagt, dass Objekte gleich sind, wenn sie die gleichen externen Eigenschaften aufweisen. Beispielsweise sind zwei Funktionen gleich, wenn sie für alle Eingaben die gleiche Ausgabe liefern: $(f = g) = (\forall x.\ f\ x = g\ x)$. Die interne (intensionale) Implementierung der Funktionen mag unterschiedlich sein, dennoch sind sie gleich. Dies ist die natürliche Gleichheit in HOL, welche uns erlaubt unser Modell bequem zu shallow-embedden. Meine extensionale Modellierung prägt diese Theorie stark. Beispielsweise sind Handlungen extensional modelliert, d.h nur die äußerlich messbaren Ergebnisse werden betrachtet. Dies widerspricht vermutlich stark Kants Vorstellung.
- *Interpretation* besagt, dass es sic hier um meine persönliche Interpretation handelt. Diese Theorie ist keine strenge Formalisierung der Literatur, sondern enthält sehr viele persönliche Meinungen.
- *Kategorischer Imperativ* bezieht sich auf Kants Kategorischer Imperativ. Ziel dieser Theorie ist es, moralische Entscheidungen basierend auf Kants Idee zu machen.

Der Titel in einfacher Sprache: Der kategorische Imperativ, aber wohl nicht so wie Kant ihn gedacht hat, also, dass nur der innere, gute Wille zählt, sondern die gegenteilige Umsetzung, bei der wir uns auf die Ergebnisse einer Handlung fokussieren.

2 Schnelleinstieg Isabelle/HOL

2.1 Typen

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

2.2 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.

2.3 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.

2.4 Funktionen

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

fun *beispielfunktion* :: ‹$nat \Rightarrow nat$› **where**
  ‹*beispielfunktion* $n = n + 10$›

Funktionsaufrufe funktionieren ohne Klammern.

lemma ‹*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* ⇒ *nat*

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

lemma ‹*addieren 10* = *beispielfunktion*›

Zusätzlich lassen sich Funktionen im Lambda Calculus darstellen. Beispiel:

lemma ‹(λ*n*::*nat*. *n*+*10*) *3* = *13*›

lemma ‹*beispielfunktion* = (λ*n*. *n*+*10*)›

2.5 Mengen

Mengen funktionieren wie normale mathematische Mengen.

Beispiel. Die Menge der geraden Zahlen:

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

lemma ‹$\{0,2,4,6,8,10\} = \{n::nat.\ n\ mod\ 2 = 0 \wedge n \leq 10\}$›

Bei vorherigen Beispiel können wir das Prinzip der (mathematischen) *Extensionalität* sehen: Intensional sind die beiden Mengen $\{0,\ 2,\ 4,\ 6,\ 8,\ 10\}$ und $\{n.\ n\ mod\ 2 = 0 \wedge n \leq 10\}$ verschieden, da sie unterschiedlich definiert sind. Extensional betrachtet, sind die beiden Mengen jedoch gleich, da sie genau die gleichen äußeren Eigenschaften haben, d.h. da sie genau die gleichen Elemente enthalten.

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*›)

Folgende Funktion beschreibt ob eine Handlung eine No-Op ist, also eine Handlung welche die Welt nicht verändert.

definition *ist-noop* :: ‹*'world handlung* ⇒ *bool*› **where**
‹*ist-noop h* ≡ *vorher h* = *nachher h*›

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

datatype (*'person*, *'world*) *handlungsabsicht* = *Handlungsabsicht* ‹*'person* ⇒ *'world* ⇒ *'world option*›

Eine (*'person*, *'world*) *handlungsabsicht* gibt eine *'world option* zurück, anstatt einer *'world*. Handlungsabsichten sind damit partielle Funktionen, was modelliert, dass die Ausführung einer Handlungsabsicht scheitern kann. Beispielsweise könnte ein Dieb versuchen ein Opfer zu bestehlen; wenn sich allerdings kein passendes Opfer findet, dann darf die Handlung scheitern. Oder es könnte der pathologische Sonderfall eintreten, dass ein Dieb sich selbst bestehlen soll. Auch hier darf die Handlung scheitern. Von außen betrachtet ist eine solche gescheiterte Handlung nicht zu unterscheiden vom Nichtstun. Allerdings ist es für die moralische Betrachtung dennoch wichtig zu unterscheiden, ob die Handlungsabsicht ein gescheiterter Diebstahl war, oder ob die Handlungsabsicht einfach Nichtstun war. Dadurch dass Handlungsabsichten partiell sind, können wir unterscheiden ob die Handlung wie geplant ausgeführt wurde oder gescheitert ist. Moralisch sind Stehlen und Nichtstun sehr verschieden.

fun *nachher-handeln*
:: ‹*'person* ⇒ *'world* ⇒ (*'person*, *'world*) *handlungsabsicht* ⇒ *'world*›
where
‹*nachher-handeln handelnde-person welt* (*Handlungsabsicht h*) =
(*case h handelnde-person welt of Some welt'* ⇒ *welt'*
| *None* ⇒ *welt*)›

Die Funktion *nachher-handeln* besagt, dass eine gescheiterte Handlung die Welt nicht verändert. Ab diesem Punkt sind also die Handlungen "sich selbst bestehlen" und "Nichtstun" von außen ununterscheidbar, da beide die Welt nicht verändern.

definition *handeln*
:: ‹*'person* ⇒ *'world* ⇒ (*'person*, *'world*) *handlungsabsicht* ⇒ *'world handlung*›
where
‹*handeln handelnde-person welt ha* ≡ *Handlung welt* (*nachher-handeln handelnde-person welt ha*)›

Die Funktion *nachher-handeln* liefert die Welt nach der Handlung. Die Funktion *handeln* liefert eine *'world handlung*, welche die Welt vor und nach der Handlung darstellt.

Beispiel, für eine Welt die nur aus einer Zahl besteht: Wenn die Zahl kleiner als 9000 ist erhöhe ich sie, ansonsten schlägt die Handlung fehl.

definition ‹*beispiel-handlungsabsicht* ≡ *Handlungsabsicht* (λ- *n. if n* < *9000 then Some* (*n*+*1*) *else None*)›

lemma ‹*nachher-handeln "Peter"* (*42*::*nat*) *beispiel-handlungsabsicht* = *43*›
lemma ‹*handeln "Peter"* (*42*::*nat*) *beispiel-handlungsabsicht* = *Handlung 42 43*›

lemma $\langle$*nachher-handeln ''Peter'' (9000::nat) beispiel-handlungsabsicht* $=$ *9000*$\rangle$
lemma $\langle$*ist-noop (handeln ''Peter'' (9000::nat) beispiel-handlungsabsicht)*$\rangle$

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*) *handlungsabsicht* kann nicht geprinted werden!

Da Funktionen nicht geprintet werden können, sieht *beispiel-handlungsabsicht* so aus: *Handlungsabsicht -*

Um eine gescheiterte Handlung von einer Handlung welche die Welt nicht verändert zu unterscheiden, sagen wir, dass eine Handlungsabsicht ausführbar ist, wenn die ausgeführte Handlungsabsicht nicht gescheitert ist:

fun *ausfuehrbar* :: $\langle$*'person* $\Rightarrow$ *'world* $\Rightarrow$ (*'person*, *'world*) *handlungsabsicht* $\Rightarrow$ *bool*$\rangle$
where
$\langle$*ausfuehrbar p welt* (*Handlungsabsicht h*) $=$ (*h p welt* $\neq$ *None*)$\rangle$

Nicht ausführbare Handlungen resultieren in unserem Modell im Nichtstun:

lemma *nicht-ausfuehrbar-ist-noop*:
$\langle\neg$*ausfuehrbar p welt ha* $\Longrightarrow$ *ist-noop* (*handeln p welt ha*)$\rangle$

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 Gesinnungsethik "[..] 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*) *handlungsabsicht* $\Rightarrow$ *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."

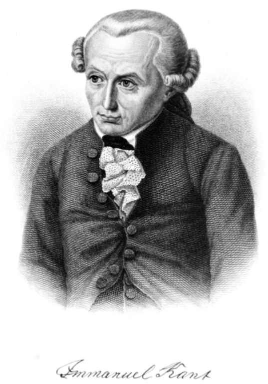
- Demnach ist eine Verantwortungsethik: *'world handlung* $\Rightarrow$ *bool*.

Da *handeln* eine Handlungsabsicht (*'person*, *'world*) *handlungsabsicht* in eine konkrete Änderung der Welt *'world handlung* überführt, können wie die beiden Ethiktypen miteinander in Verbindung 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*) *handlungsabsicht* ⇒ *bool*) ⇒ (*'world handlung* ⇒ *bool*) ⇒ *bool*› **where**
‹*gesinnungsethik-verantwortungsethik-konsistent gesinnungsethik verantwortungsethik* ≡
  ∀ *handlungsabsicht*.
    gesinnungsethik handlungsabsicht ⟷
      (∀ *person welt*. *verantwortungsethik* (*handeln person welt handlungsabsicht*))›

Ich habe aktuell kein Beispiel für eine Gesinnungsethik und eine Verantwortungsethik, die tatsächlich konsistent sind. Später (In §9.1) werden wir sehen, dass es eine Übersetzung gibt, mit der die goldene Regel und der Utilitarismus konsistent sind.

4 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

5 Beispiel Person

Wir führen eine Beispielbevölkerung für Beispiele ein. Sie besteht aus vier Personen.

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

In Isabelle/HOL steht die Konstante *UNIV* vom Typ *'a set* für die Menge aller *'a*, also das Universum über *'a*. Das Universum *UNIV* vom Typ *person set* unserer Bevölkerung ist sehr endlich:

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

Wir werden unterscheiden:

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

6 Maxime

Nach https://de.wikipedia.org/wiki/Maxime ist eine Maxime ein persönlicher Grundsatz des Wollens und Handelns. Nach Kant ist eine Maxime ein "subjektives Prinzip des Wollens".
Modell einer *Maxime*: Eine Maxime in diesem Modell beschreibt ob eine Handlung in einer gegebenen Welt gut ist.
Faktisch bruachen wir um eine Maxime zu modellieren

- *'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 *'person* aus deren Sicht die Maxime definiert ist, 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*›

Auswertung einer Maxime:

fun *okay* :: ‹(*'person*, *'world*) *maxime* ⇒ *'person* ⇒ *'world handlung* ⇒ *bool*› **where**
‹*okay* (*Maxime m*) *p h* = *m p h*›

Beispiel

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

6.1 Maxime in Sinne Kants?

Kants kategorischer Imperativ ist eine deontologische Ethik, d.h., "Es wird eben nicht bewertet, was die Handlung bewirkt, sondern wie die Absicht beschaffen ist." https://de.wikipedia.org/wiki/Kategorischer_Imperativ.
Wenn wir Kants kategorischen Imperativ bauen wollen, dürfen wir also nicht die Folgen einer Handlung betrachten, sondern nur die Absicht dahinter. Doch unsere *Maxime* betrachtet eine *'world handlung*, also eine konkrete Handlung, die nur durch ihre Folgen gegeben ist. Die Maxime betrachtet keine Handlungsabsicht (*'person*, *'world*) *handlungsabsicht*.
Kant unterscheidet unter Anderem "zwischen »apriorischen« und »empirischen« Urteilen" [1]. Wenn wir uns den Typ *'world handlung* als Beobachtung der Welt *vorher* und *nachher* anschauen, dann könnte man sagen, unser Moralbegriff der *Maxime* sei empirisch. Für Kant gilt jedoch: "Alle Moralbegriffe

[...] haben *a priori* ihren Sitz und Ursprung ausschließlich in der Vernunft" [1]. Hier widerspricht unser Modell wieder Kant, da unser Modell empirisch ist und nicht apriorisch.

Dies mag nun als Fehler in unserem Modell verstanden werden. Doch irgendwo müssen wir praktisch werden. Nur von Handlungsabsichten zu reden, ohne dass die beabsichtigten Folgen betrachtet werden ist mir einfach zu abstrakt und nicht greifbar.

Alles ist jedoch nicht verloren, denn "Alle rein mathematischen Sätze sind [...] apriorisch" [1]. Und auch Russel schlussfolgert: "Um ein ausreichendes Kriterium zu gewinnen, müßten wir Kants rein formalen Standpunkt aufgeben und die Wirkung der Handlungen in Betracht ziehen" [1].

Auch Kants kategorischer Imperativ und die Goldene Regel sind grundverschieden: https://web.archive.org/web/20220123174117/https://www.goethegymnasium-hildesheim.de/index.php/faecher/faecher/gesellschaftswissenschaften/philosophie Dennoch, betrachten wir den kategorischen Imperativ als eine Verallgemeinerung der goldenen Regel.

6.2 Die Goldene Regel

Die Goldene Regel nach https://de.wikipedia.org/wiki/Goldene_Regel sagt:

„Behandle andere so, wie du von ihnen behandelt werden willst.“

„Was du nicht willst, dass man dir tu, das füg auch keinem andern zu.“

So wie wir behandelt werden wollen ist modelliert durch eine $('person, 'world)$ *maxime*.

Die goldene Regel testet ob eine Handlung, bzw. Handlungsabsicht moralisch ist. Um eine Handlung gegen eine Maxime zu testen fragen wir uns:

- Was wenn jeder so handeln würde?
- Was wenn jeder nach dieser Maxime handeln würde?

Beispielsweise mag "stehlen" und "bestohlen werden" die gleiche Handlung sein, jedoch wird sie von Täter und Opfer grundverschieden wahrgenommen.

```
definition bevoelkerung :: ‹'person set› where ‹bevoelkerung ≡ UNIV›

definition wenn-jeder-so-handelt
  :: ‹'world ⇒ ('person, 'world) handlungsabsicht ⇒ ('world handlung) set›
 where
  ‹wenn-jeder-so-handelt welt handlungsabsicht ≡
    (λhandelnde-person. handeln handelnde-person welt handlungsabsicht) ‘ bevoelkerung›

fun was-wenn-jeder-so-handelt-aus-sicht-von
  :: ‹'world ⇒ ('person, 'world) maxime ⇒ ('person, 'world) handlungsabsicht ⇒ 'person ⇒ bool›
 where
  ‹was-wenn-jeder-so-handelt-aus-sicht-von welt m handlungsabsicht betroffene-person =
     (∀ h ∈ wenn-jeder-so-handelt welt handlungsabsicht. okay m betroffene-person h)›
```

Für eine gegebene Welt und eine gegebene Maxime nennen wir eine Handlungsabsicht genau dann moralisch, wenn die Handlung auch die eigene Maxime erfüllt, wenn die Handlung von anderen durchgeführt würde.

definition *moralisch* ::
  ‹$'world \Rightarrow ('person, 'world)$ *maxime* $\Rightarrow ('person, 'world)$ *handlungsabsicht* $\Rightarrow$ *bool*› **where**
‹*moralisch welt handlungsabsicht maxime* $\equiv$
  $\forall p \in$ *bevoelkerung. was-wenn-jeder-so-handelt-aus-sicht-von welt handlungsabsicht maxime p*›

Faktisch bedeutet diese Definition, wir bilden das Kreuzprodukt *bevoelkerung* $\times$ *bevoelkerung*, wobei jeder einmal als handelnde Person auftritt und einmal als betroffene Person.

lemma *moralisch-unfold*:
  ‹*moralisch welt* (*Maxime m*) *handlungsabsicht* $\longleftrightarrow$
    ($\forall p1 \in$*bevoelkerung.* $\forall p2 \in$*bevoelkerung. m p1* (*handeln p2 welt handlungsabsicht*))›
lemma ‹*moralisch welt* (*Maxime m*) *handlungsabsicht* $\longleftrightarrow$
    ($\forall (p1,p2) \in$*bevoelkerung*$\times$*bevoelkerung. m p1* (*handeln p2 welt handlungsabsicht*))›

lemma *moralisch-simp*:
  ‹*moralisch welt m handlungsabsicht* $\longleftrightarrow$
    ($\forall p1\ p2.$ *okay m p1* (*handeln p2 welt handlungsabsicht*))›

Wir können die goldene Regel auch umformulieren, nicht als Imperativ, sondern als Beobachtung eines Wunschzustandes: Wenn eine Handlung für eine Person okay ist, dann muss sie auch Okay sein, wenn jemand anderes diese Handlung ausführt.

Formal: *m ich* (*handeln ich welt handlungsabsicht*) $\Longrightarrow \forall p2.$ *m ich* (*handeln p2 welt handlungsabsicht*)

Genau dies können wir aus unserer Definition von *moralisch* ableiten:

lemma *goldene-regel*:
  ‹*moralisch welt m handlungsabsicht* $\Longrightarrow$
   okay m ich (*handeln ich welt handlungsabsicht*) $\Longrightarrow$
  $\forall p2.$ *okay m ich* (*handeln p2 welt handlungsabsicht*)›

Für das obige lemma brauchen wir die Annahme *m ich* (*handeln ich welt handlungsabsicht*) gar nicht.

Wenn für eine gegebene *Maxime m* eine Handlungsabsicht moralisch ist, dann ist es auch okay, wenn ich von der Handlungsabsicht betroffen bin, egal wer sie ausführt.

corollary
  ‹*moralisch welt m handlungsabsicht* $\Longrightarrow$
  $\forall p2.$ *okay m ich* (*handeln p2 welt handlungsabsicht*)›

Die umgekehrte Richtung gilt nicht, weil diese Formulierung nur die Handlungen betrachtet, die okay sind.

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

6.3 Maximen Debugging

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

record (*'person*, *'world*) *dbg-verletzte-maxime* =
  dbg-opfer :: ‹*'person*› — verletzt für; das Opfer
  dbg-taeter :: ‹*'person*› — handelnde Person; der Täter
  dbg-handlung :: ‹*'world handlung*› — Die verletzende Handlung

Alle Feldnamen bekommen das Präfix "dbg" für Debug um den Namensraum nicht zu verunreinigen.

Die folgende Funktion liefert alle Gegebenheiten welche eine Maxime verletzen:

fun *debug-maxime*
  :: ‹(*'world* ⇒ *'printable-world*) ⇒ *'world* ⇒
    (*'person*, *'world*) *maxime* ⇒ (*'person*, *'world*) *handlungsabsicht*
    ⇒ ((*'person*, *'printable-world*) *dbg-verletzte-maxime*) *set*›
where
  ‹*debug-maxime print-world welt m handlungsabsicht* =
    {⦇
      dbg-opfer = *p1*,
      dbg-taeter = *p2*,
      dbg-handlung = *map-handlung print-world* (*handeln p2 welt handlungsabsicht*)
     ⦈
     | *p1 p2*. ¬*okay 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 maxime handlungsabsicht* = {}
    ⟷ *moralisch welt maxime handlungsabsicht*›

6.4 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 ‹*moralisch*
    (*42*::*nat*)
    maxime-mir-ist-alles-recht
    (*Handlungsabsicht* (λ(*person*::*person*) *welt*. *Some* (*welt* + *1*)))›

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 ‹*moralisch*
    [*Alice* ↦ (*0*::*nat*), *Bob* ↦ *0*, *Carol* ↦ *0*, *Eve* ↦ *0*]
    (*Maxime* (λ*person handlung*.
      (*the* ((*vorher handlung*) *person*)) ≤ (*the* ((*nachher handlung*) *person*))))
    (*Handlungsabsicht* (λ*person welt*. *Some* (*welt*(*person* ↦ *3*))))›

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

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

```
lemma ‹¬ moralisch
        [Alice ↦ (0::nat), Bob ↦ 4, Carol ↦ 0, Eve ↦ 0]
        (Maxime (λperson handlung.
           (the ((vorher handlung) person)) ≤ (the ((nachher handlung) person))))
        (Handlungsabsicht (λperson welt. Some (welt(person ↦ 3))))›
lemma ‹debug-maxime show-map
        [Alice ↦ (0::nat), Bob ↦ 4, Carol ↦ 0, Eve ↦ 0]
        (Maxime (λperson handlung.
           (the ((vorher handlung) person)) ≤ (the ((nachher handlung) person))))
        (Handlungsabsicht (λperson welt. Some (welt(person ↦ 3))))
  = {⦇
      dbg-opfer = Bob,
      dbg-taeter = Bob,
      dbg-handlung = Handlung [(Alice, 0), (Bob, 4), (Carol, 0), (Eve, 0)]
                              [(Alice, 0), (Bob, 3), (Carol, 0), (Eve, 0)]
    ⦈}›
```

6.5 Maximen Kombinieren

Konjunktion (Und) zweier Maximen.

```
fun MaximeConj
  :: ‹('person, 'welt) maxime ⇒ ('person, 'welt) maxime ⇒ ('person, 'welt) maxime›
  where
‹MaximeConj (Maxime m1) (Maxime m2) = Maxime (λp h. m1 p h ∧ m2 p h)›
```

Die erwarteten Regeln auf einer Konjunktion gelten.

```
lemma okay-MaximeConj: ‹okay (MaximeConj m1 m2) p h ⟷ okay m1 p h ∧ okay m2 p h›

lemma moralisch-MaximeConj:
  ‹moralisch welt (MaximeConj m1 m2) ha ⟷ moralisch welt m1 ha ∧ moralisch welt m2 ha›

lemma moralisch-MaximeConj-False:
  ‹moralisch welt (MaximeConj m1 (Maxime (λ- -. True))) ha ⟷ moralisch welt m1 ha›

lemma moralisch-MaximeConj-True:
  ‹¬ moralisch welt (MaximeConj m1 (Maxime (λ- -. False))) ha›
```

Disjunktion (Oder) zweier Maximen.

fun *MaximeDisj*
$:: \langle ('\mathit{person},\ '\mathit{welt})\ \mathit{maxime} \Rightarrow ('\mathit{person},\ '\mathit{welt})\ \mathit{maxime} \Rightarrow ('\mathit{person},\ '\mathit{welt})\ \mathit{maxime}\rangle$
where
$\langle \mathit{MaximeDisj}\ (\mathit{Maxime}\ m1)\ (\mathit{Maxime}\ m2) = \mathit{Maxime}\ (\lambda p\ h.\ m1\ p\ h \vee m2\ p\ h)\rangle$

lemma *okay-MaximeDisj*: $\langle \mathit{okay}\ (\mathit{MaximeDisj}\ m1\ m2)\ p\ h \longleftrightarrow \mathit{okay}\ m1\ p\ h \vee \mathit{okay}\ m2\ p\ h\rangle$

Leider ist *MaximeDisj* weniger schön, weil es kein genau-dann-wenn mit der Disjunktion ($m1 \vee m2$) gibt.

lemma *moralisch-MaximeDisjI*:
$\langle \mathit{moralisch}\ \mathit{welt}\ m1\ \mathit{ha} \vee \mathit{moralisch}\ \mathit{welt}\ m2\ \mathit{ha} \Longrightarrow \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ m1\ m2)\ \mathit{ha}\rangle$

Die Rückrichtung gilt leider nicht. *MaximeDisj m1 m2* is effektiv schwächer, da sich jede Person unabhängig entscheiden darf, ob sie *m1* oder *m2* folgt. Im Gegensatz dazu sagt *moralisch welt m1 ha* ∨ *moralisch welt m2 ha*, dass für *alle* Personen entweder *m1* oder *m2* gelten muss.

lemma *moralisch-MaximeDisj1*:
$\langle \mathit{moralisch}\ \mathit{welt}\ m1\ \mathit{ha} \Longrightarrow \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ m1\ m2)\ \mathit{ha}\rangle$
lemma *moralisch-MaximeDisj2*:
$\langle \mathit{moralisch}\ \mathit{welt}\ m2\ \mathit{ha} \Longrightarrow \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ m1\ m2)\ \mathit{ha}\rangle$

lemma *moralisch-MaximeDisj-False*:
$\langle \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ m1\ (\mathit{Maxime}\ (\lambda\text{-}\ \text{-}.\ \mathit{False})))\ \mathit{ha} \longleftrightarrow \mathit{moralisch}\ \mathit{welt}\ m1\ \mathit{ha}\rangle$

lemma *moralisch-MaximeDisj-True*:
$\langle \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ m1\ (\mathit{Maxime}\ (\lambda\text{-}\ \text{-}.\ \mathit{True})))\ \mathit{ha}\rangle$

Negation.

fun *MaximeNot* $:: \langle ('\mathit{person},\ '\mathit{welt})\ \mathit{maxime} \Rightarrow ('\mathit{person},\ '\mathit{welt})\ \mathit{maxime}\rangle$
where
$\langle \mathit{MaximeNot}\ (\mathit{Maxime}\ m) = \mathit{Maxime}\ (\lambda p\ h.\ \neg\ m\ p\ h)\rangle$

lemma *okay-MaximeNot*: $\langle \mathit{okay}\ (\mathit{MaximeNot}\ m)\ p\ h \longleftrightarrow \neg\ \mathit{okay}\ m\ p\ h\rangle$

lemma *okay-DeMorgan*:
$\langle \mathit{okay}\ (\mathit{MaximeNot}\ (\mathit{MaximeConj}\ m1\ m2))\ p\ h$
$\longleftrightarrow \mathit{okay}\ (\mathit{MaximeDisj}\ (\mathit{MaximeNot}\ m1)\ (\mathit{MaximeNot}\ m2))\ p\ h\rangle$

lemma *moralisch-DeMorgan*:
$\langle \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeNot}\ (\mathit{MaximeConj}\ m1\ m2))\ \mathit{ha}$
$\longleftrightarrow \mathit{moralisch}\ \mathit{welt}\ (\mathit{MaximeDisj}\ (\mathit{MaximeNot}\ m1)\ (\mathit{MaximeNot}\ m2))\ \mathit{ha}\rangle$

7 Schleier des Nichtwissens

Rawls' Schleier des Nichtwissens https://de.wikipedia.org/wiki/Schleier_des_Nichtwissens ist ein fiktives Modell, in der Personen »über die zukünftige Gesellschaftsordnung entscheiden können, aber selbst nicht wissen, an welcher Stelle dieser Ordnung sie sich später befinden werden, also unter einem „Schleier des Nichtwissens" stehen.« Quote wikipedia

Wir bedienen uns bei der Idee dieses Modells um gültige Handlungsabsichten und Maximen zu definieren. Handlungsabsichten und Maximen sind nur gültig, wenn darin keine Personen hardgecoded werden.

Beispielsweise ist folgende Handlungsabsicht ungültig: $\lambda ich\ welt.\ \mathit{if}\ ich = Alice\ \mathit{then}\ Do\text{-}A\ welt\ \mathit{else}\ Do\text{-}B\ welt$

Handlungsabsichten und Maximen müssen immer generisch geschrieben werden, so dass die handelnden und betroffenen Personen niemals anhand ihres Namens ausgewählt werden.

unser Modell von Handlungsabsichten und Maximen stellt beispielsweise die handelnde Person als Parameter bereit. Folgendes ist also eine gültige Handlung: $\lambda ich\ welt.\ ModifiziereWelt\ welt\ ich$

Auch ist es erlaubt, Personen in einer Handlungsabsicht oder Maxime nur anhand ihrer Eigenschaften in der Welt auszuwählen. Folgendes wäre eine wohlgeformte Handlung, wenn auch eine moralisch fragwürdige: $\lambda ich\ welt.\ enteignen\ \text{`}\ \{opfer.\ besitz\ ich < besitz\ opfer\}$

Um diese Idee von wohlgeformten Handlungsabsichten und Maximen zu formalisieren bedienen wir uns der Idee des Schleiers des Nichtwissens. Wir sagen, dass Handlungsabsichten wohlgeformt sind, wenn die Handlungsabsicht gleich bleibt, wenn man sowohl die handelnde Person austauscht, als auch alle weltlichen Eigenschaften dieser Person. Anders ausgedrückt: Wohlgeformte Handlungsabsichten und Maximen sind solche, bei denen bei der Definition noch nicht feststeht, auf we sie später zutreffen.

Für jede Welt muss eine Welt-Personen Swap (wps) Funktion bereit gestellt werden, die alle Weltlichen Eigenschaften von 2 Personen vertauscht:

type-synonym $('person,\ 'world)\ wp\text{-}swap = \langle 'person \Rightarrow 'person \Rightarrow 'world \Rightarrow 'world\rangle$

Ein jeder $('person,\ 'world)\ wp\text{-}swap$ sollte mindestens folgendes erfüllen:

definition $wps\text{-}id :: \langle('person,\ 'world)\ wp\text{-}swap \Rightarrow 'world \Rightarrow bool\rangle$
where
$\langle wps\text{-}id\ wps\ welt \equiv \forall p1\ p2.\ wps\ p2\ p1\ (wps\ p1\ p2\ welt) = welt\rangle$

$$\begin{tikzcd} welt \arrow[r, "handeln\ p1"] \arrow[d, "wps\ p1\ p2"] & welt' \\ alternativ\text{-}welt \arrow[r, "handeln\ p2"] & alternativ\text{-}welt' \arrow[u, "wps\ p1\ p2"'] \end{tikzcd}$$

7.1 Wohlgeformte Handlungsabsicht

Wir sagen, eine Handlungsabsicht ist wohlgeformt, genau dann wenn sie obiges kommutatives Diagramm erfüllt, d.h. wenn folgendes equivalent ist

- handeln in einer Welt.

- zwei Personen in einer Welt zu vertauschen, in der veränderten Welt zu handeln, und die beiden Personen wieder zurück tauschen.

```
fun wohlgeformte-handlungsabsicht
  :: ‹('person, 'world) wp-swap ⇒ 'world ⇒ ('person, 'world) handlungsabsicht ⇒ bool›
where
  ‹wohlgeformte-handlungsabsicht wps welt (Handlungsabsicht h) =
   (∀p1 p2. h p1 welt = map-option (wps p2 p1) (h p2 (wps p1 p2 welt)))›
```

Folgende Folgerung erklärt die Definition vermutlich besser:

```
lemma wohlgeformte-handlungsabsicht-wpsid-imp-handeln:
  ‹wohlgeformte-handlungsabsicht wps welt ha ⟹ wps-id wps welt ⟹
    (∀p1 p2. handeln p1 welt ha =
              Handlung welt
                     (wps p2 p1 (nachher-handeln p2 (wps p1 p2 welt) ha)))›
```

Folgendes Lemma erlaubt es uns das kommutative Diagramm auch leicht anders zu zeichnen.

```
lemma wohlgeformte-handlungsabsicht-wpsid-wpssym-komm:
  assumes wpsid: ‹∀welt. wps-id wps welt›
    and wps-sym: ‹∀welt. wps p1 p2 welt = wps p2 p1 welt›
  shows ‹wohlgeformte-handlungsabsicht wps (wps p1 p2 welt) ha ⟹
    handeln p1 (wps p1 p2 welt) ha =
           map-handlung (wps p1 p2) (handeln p2 welt ha)›
```

$$\begin{tikzcd} welt \arrow[r, "handeln\ p2"] \arrow[d, "wps\ p1\ p2"] & welt' \arrow[d, "wps\ p1\ p2"] \\ alternativ\text{-}welt \arrow[r, "handeln\ p1"] & alternativ\text{-}welt' \end{tikzcd}$$

In einigen späteren Beispielen möchten wir zeigen, dass bestimmte Handlungsabsichten nicht wohlgeformt sind.

```
fun wohlgeformte-handlungsabsicht-gegenbeispiel
  :: ‹('person, 'world) wp-swap ⇒ 'world ⇒ ('person, 'world) handlungsabsicht ⇒ 'person ⇒ 'person ⇒ bool›
where
  ‹wohlgeformte-handlungsabsicht-gegenbeispiel wps welt (Handlungsabsicht h) taeter opfer ⟷
  h taeter welt ≠ map-option (wps opfer taeter) (h opfer (wps taeter opfer welt))›
```

```
lemma ‹∃p1 p2. wohlgeformte-handlungsabsicht-gegenbeispiel wps welt ha p1 p2 ⟷
        ¬wohlgeformte-handlungsabsicht wps welt ha›
```

7.2 Spezialfall: Maxime und Handlungsabsichten haben nette Eigenschaften

Dieses Kapitel darf gerne übersprungen werden, da der Spezialfall nur in bestimmten Beweisen interessant wird.

Nach der gleichen Argumentation müssten Maxime und Handlungsabsicht so generisch sein, dass sie in allen Welten zum gleichen Ergebnis kommen. Dies gilt jedoch nicht immer. Wenn dieser Sonderfall eintritt sagen wir, Maxime und Handlungsabsicht generalisieren.

```
definition maxime-und-handlungsabsicht-generalisieren
  :: ‹('person, 'world) wp-swap ⇒ 'world ⇒
      ('person, 'world) maxime ⇒ ('person, 'world) handlungsabsicht ⇒ 'person ⇒ bool›
where
  ‹maxime-und-handlungsabsicht-generalisieren wps welt m ha p =
    (∀p1 p2. (ausfuehrbar p welt ha ∧ ausfuehrbar p (wps p1 p2 welt) ha)
            ⟶ okay m p (handeln p welt ha) ⟷ okay m p (handeln p (wps p1 p2 welt) ha))›
```

Die Vorbedingungen in obiger Definition, nämlich dass die Handlungsabsicht *ausfuehrbar* ist, ist nötig, um z.B. Handlungsabsichten wie das Stehlen zu ermöglichen; jedoch gibt es beim Stehlen genau den pathologischen Grenzfall von-sich-selbst Stehlen, welcher in einer No-Op endet und das Ergebnis damit nicht moralisch falsch ist. Durch die Einschränkung auf *ausfuehrbar* Fälle lassen sich solche pathologischen Grenzfälle ausklammern.

Für eine gegebene Maxime schließt die Forderung *maxime-und-handlungsabsicht-generalisieren* leider einige Handlungen aus. Beispiel: In einer Welt besitzt *Alice* 2 und *Eve* hat 1 Schulden. Die Maxime ist, dass Individuen gerne keinen Besitz verlieren. Die Handlung sei ein globaler reset, bei dem jeden ein Besitz von 0 zugeordnet wird. Leider generalisiert diese Handlung nicht, da *Eve* die Handlung gut findet, *Alice* allerdings nicht.

```
lemma
  ‹¬ maxime-und-handlungsabsicht-generalisieren
     swap
     ((λx. 0)(Alice := (2::int), Eve := − 1))
     (Maxime (λich h. (vorher h) ich ≤ (nachher h) ich))
     (Handlungsabsicht (λich w. Some (λ-. 0)))
     Eve›
```

Die Maxime und *('person, 'world) wp-swap* können einige Eigenschaften erfüllen.

Wir kürzen das ab mit *wpsm*: Welt Person Swap Maxime.

Die Person für die Maxime ausgewertet wird und swappen der Personen in der Welt kann equivalent sein:

```
definition wpsm-kommutiert
  :: ‹('person, 'world) maxime ⇒ ('person, 'world) wp-swap ⇒ 'world ⇒ bool›
where
  ‹wpsm-kommutiert m wps welt ≡
  ∀ p1 p2 ha.
    okay m p2 (handeln p1 (wps p1 p2 welt) ha)
    ⟷
    okay m p1 (Handlung welt (wps p1 p2 (nachher-handeln p1 (wps p2 p1 welt) ha)))›
```

Wenn sowohl eine *wohlgeformte-handlungsabsicht* vorliegt, als auch *wpsm-kommutiert*, dann erhalten wir ein sehr intuitives Ergebnis, welches besagt, dass ich handelnde Person und Person für die die Maxime gelten soll vertauschen kann.

lemma *wfh-wpsm-kommutiert-simp*:
  assumes *wpsid*: ‹*wps-id wps welt*›
  shows ‹*wohlgeformte-handlungsabsicht wps welt ha* $\Longrightarrow$
  wpsm-kommutiert m wps welt $\Longrightarrow$
    okay m p2 (*handeln p1* (*wps p1 p2 welt*) *ha*)
    $\longleftrightarrow$
    okay m p1 (*handeln p2 welt ha*)›

Die Rückrichtung gilt auch, aber da wir das für alle Handlungsabsichten in der Annahme brauchen, ist das eher weniger hilfreich.

lemma *wfh-kommutiert-wpsm*:
  assumes *wpsid*: ‹*wps-id wps welt*›
  shows
  ‹$\forall$ *ha*. *wohlgeformte-handlungsabsicht wps welt ha* $\wedge$
      ($\forall$ *p1 p2*. *okay m p2* (*handeln p1* (*wps p1 p2 welt*) *ha*)
          $\longleftrightarrow$
          okay m p1 (*handeln p2 welt ha*)) $\Longrightarrow$
    wpsm-kommutiert m wps welt›

7.3 Wohlgeformte Maxime

Nach dem gleichen Konzept nach dem wir die *wohlgeformte-handlungsabsicht* definiert haben, definieren wir, was es bedeutet für eine Maxime wohlgeformt zu sein.

definition *wohlgeformte-maxime-auf*
  :: ‹*'world handlung* $\Rightarrow$ (*'person*, *'world*) *wp-swap* $\Rightarrow$ (*'person*, *'world*) *maxime* $\Rightarrow$ *bool*›
where
  ‹*wohlgeformte-maxime-auf h wps m* $\equiv$
    $\forall$ *p1 p2*. *okay m p1 h* $\longleftrightarrow$ *okay m p2* (*map-handlung* (*wps p1 p2*) *h*)›

Eigentlich sollte eine Maxime wohlgeformte sein für alle Handlungen. Jedoch definieren wir hier eine restriktive Version *wohlgeformte-maxime-auf* welche nur auf einer Handlung wohlgeformt ist. Der Grund ist leider ein Implementierungsdetail. Da wir ausführbaren Code wollen und Handlungen normalerweise nicht vollständig aufzählbar sind, werden wir auch den kategorischen Imperativ auf eine endliche Menge von Handlungsabsichten beschränken. Die eigentlich schönere (jedoch schwer zu beweisende) Forderung lautet:

definition *wohlgeformte-maxime*
  :: ‹(*'person*, *'world*) *wp-swap* $\Rightarrow$ (*'person*, *'world*) *maxime* $\Rightarrow$ *bool*›
where
  ‹*wohlgeformte-maxime wps m* $\equiv$
    $\forall$ *h*. *wohlgeformte-maxime-auf h wps m*›

Beispiel:

lemma ‹*wohlgeformte-maxime swap* (*Maxime* (λ*ich h*. (*vorher h*) *ich* $\leq$ (*nachher h*) *ich*))›

8 Kategorischer Imperativ

Wir haben mit der goldenen Regel bereits definiert, wann für eine gegebene Welt und eine gegebene Maxime, eine Handlungsabsicht moralisch ist:

- $moralisch :: 'world \Rightarrow ('person, 'world)\ maxime \Rightarrow ('person, 'world)\ handlungsabsicht \Rightarrow bool$

Effektiv testet die goldene Regel eine Handlungsabsicht.

Nach meinem Verständnis generalisiert Kant mit dem Kategorischen Imperativ diese Regel, indem die Maxime nicht mehr als gegeben angenommen wird, sondern die Maxime selbst getestet wird. Sei die Welt weiterhin gegeben, dann müsste der kategorische Imperativ folgende Typsignatur haben:

- $'world \Rightarrow ('person, 'world)\ maxime \Rightarrow bool$

Eine Implementierung muss dann über alle möglichen Handlungsabsichten allquantifizieren.

Grob gesagt: Die goldene Regel urteilt über eine Handlungsabsicht gegeben eine Maxime, der kategorische Imperativ urteilt über die Maxime an sich.

Ich behaupte, der kategorischer Imperativ lässt sich wie folgt umformulieren:

- Handle nur nach derjenigen Maxime, durch die du zugleich wollen kannst, dass sie ein allgemeines Gesetz werde.
- Handle nur nach derjenigen Maxime, durch die du zugleich wollen kannst, dass sie jeder befolgt, im Sinne der goldenen Regel.
- Handle nur nach derjenigen Maxime, durch die du zugleich wollen kannst, dass sie (Handlung und Maxime) moralisch ist.
- Wenn es jemanden gibt der nach einer Maxime handeln will, dann muss diese Handlung nach der Maxime moralisch sein.
- Für jede Handlungsabsicht muss gelten: Wenn jemand nach einer Handlungsabsicht handeln würde (getestet durch die Maxime), dann muss diese Handlung moralisch sein (getestet durch die Maxime).

Daraus ergibt sich diese Formalisierung:

Für eine bestimmte Handlungsabsicht: Wenn es eine Person gibt für die diese Handlungsabsicht moralisch ist, dann muss diese Handlungsabsicht auch für alle moralisch (im Sinne der goldenen Regel) sein.

definition *kategorischer-imperativ-auf*
  :: ‹('person, 'world) handlungsabsicht ⇒ 'world ⇒ ('person, 'world) maxime ⇒ bool›
where

‹*kategorischer-imperativ-auf ha welt m* ≡
  (∃ *ich. ausfuehrbar ich welt ha* ∧ *okay m ich* (*handeln ich welt ha*)) ⟶ *moralisch welt m ha*›

Wir beschränken uns auf die *ausfuehrbar*en Handlungsabsichten um pathologische Grenzfälle (welche keinen Rückschluss auf moralische Gesinnung lassen) auszuschließen.

Für alle möglichen (wohlgeformten) Handlungsabsichten muss dies nun gelten:

definition *kategorischer-imperativ*
  :: ‹(*'person*, *'world*) *wp-swap* ⇒ *'world* ⇒ (*'person*, *'world*) *maxime* ⇒ *bool*›
where
  ‹*kategorischer-imperativ wps welt m* ≡
    ∀ *ha. wohlgeformte-handlungsabsicht wps welt ha* ⟶
        kategorischer-imperativ-auf ha welt m›

Damit hat *kategorischer-imperativ wps*::*'world* ⇒ (*'person*, *'world*) *maxime* ⇒ *bool* die gewünschte Typsignatur.

Wir haben die interne Hilfsdefinition *kategorischer-imperativ-auf* eingeführt um den kategorischen Imperativ nur für eine Teilmenge aller Handlungen besser diskutieren zu können.

Leider fehlen mir nicht-triviale Beispiele von Maximen welche den kategorischen Imperativ uneingeschränkt auf allen Handlungsabsichten erfüllen.

Die Vorbedingung *ausfuehrbar ich welt ha* in *kategorischer-imperativ-auf* wirkt etwas holprig. Wir brauchen sie aber, um pathologische Grenzfälle auszuschließen. Beispielsweise ist von-sich-selbst stehlen eine (nicht ausführbare) No-Op. No-ops sind normalerweise nicht böse. Stehlen ist schon böse. Dieser Grenzfall in dem Stehlen zur no-op wird versteckt also den Charakter der Handlungsabsicht und muss daher ausgeschlossen werden. Da Handlungen partiell sind, ist von-sich-selbst-stehlen auch also nicht ausführbar modelliert, da Stehlen bedeutet "jemand anderen etwas wegnehmen" und im Grenzfall "von sich selbst stehlen" nicht definiert ist.

In der Definition *kategorischer-imperativ* ist *wohlgeformte-handlungsabsicht* ein technisch notwendiges Implementierungsdetail um nicht-wohlgeformte Handlungen auszuschließen.

Minimal andere Formulierung:

lemma
‹*kategorischer-imperativ wps welt m* ⟷
  (∀ *ha*.
    (∃ *p*.
        wohlgeformte-handlungsabsicht wps welt ha ∧
        ausfuehrbar p welt ha ∧
        okay m p (*handeln p welt ha*))
    ⟶ *moralisch welt m ha*)›

Der Existenzquantor lässt sich auch in einen Allquantor umschreiben:

lemma
‹*kategorischer-imperativ wps welt m* ⟷
  (∀ *ha ich*.
      wohlgeformte-handlungsabsicht wps welt ha ∧ *ausfuehrbar ich welt ha* ∧

```
    okay m ich (handeln ich welt ha) ⟶ moralisch welt m ha)›
```

Vergleich zu *moralisch*. Wenn eine Handlung moralisch ist, dann impliziert diese Handlung die Kernforderung des *kategorischer-imperativ*. Wenn die Handlungsabsicht für mich okay ist, ist sie auch für alle anderen okay.

```
lemma ‹moralisch welt m ha ⟹
     kategorischer-imperativ-auf ha welt m›
```

Die andere Richtung gilt nicht, z.B. ist die Maxime die immer False zurückgibt ein Gegenbeispiel.

```
lemma ‹m = Maxime (λ- -. False) ⟹
     kategorischer-imperativ-auf ha welt m ⟶ moralisch welt m ha
       ⟹ False›
```

Der *kategorischer-imperativ* lässt sich auch wie folgt umformulieren. Für jede Handlungsabsicht: wenn ich so handeln würde muss es auch okay sein, wenn zwei beliebige Personen so handeln, wobei einer Täter und einer Opfer ist.

```
lemma kategorischer-imperativ-simp:
  ‹kategorischer-imperativ wps welt m ⟷
    (∀ ha p1 p2 ich.
      wohlgeformte-handlungsabsicht wps welt ha ∧ ausfuehrbar ich welt ha ∧
      okay m ich (handeln ich welt ha)
      ⟶ okay m p1 (handeln p2 welt ha))›
```

Um den *kategorischer-imperativ-auf* einer Handlungsabsicht zu zeigen muss entweder die Handlungsabsicht moralisch sein, oder es darf keine Person geben, die diese Handlung auch tatsächlich unter gegebener Maxime ausführen würde:

```
lemma kategorischer-imperativ-auf2:
  ‹moralisch welt m ha ∨ ¬(∃ p. ausfuehrbar p welt ha ∧ okay m p (handeln p welt ha))
      ⟷ kategorischer-imperativ-auf ha welt m›
```

Für Beispiele wird es einfacher zu zeigen, dass eine Maxime nicht den kategorischen Imperativ erfüllt, wenn wir direkt ein Beispiel angeben.

```
definition ‹kategorischer-imperativ-gegenbeispiel wps welt m ha ich p1 p2 ≡
wohlgeformte-handlungsabsicht wps welt ha ∧
       ausfuehrbar ich welt ha ∧ okay m ich (handeln ich welt ha) ∧
      ¬ okay m p1 (handeln p2 welt ha)›
```

```
lemma ‹kategorischer-imperativ-gegenbeispiel wps welt m ha ich p1 p2 ⟹
  ¬ kategorischer-imperativ wps welt m›
```

8.1 Triviale Maximen die den Kategorischen Imperativ immer Erfüllen

Die Maxime die keine Handlung erlaubt (weil immer False) erfüllt den kategorischen Imperativ:

lemma ‹*kategorischer-imperativ wps welt* (*Maxime* (λ*ich h. False*))›

Allerdings kann mit so einer Maxime nie etwas moralisch sein.

lemma ‹¬ *moralisch welt* (*Maxime* (λ*ich h. False*)) *h*›

Die Maxime die jede Handlung erlaubt (weil immer True) erfüllt den kategorischen Imperativ:

lemma ‹*kategorischer-imperativ wps welt* (*Maxime* (λ*ich h. True*))›

Allerdings ist mit so einer Maxime alles moralisch.

lemma ‹*moralisch welt* (*Maxime* (λ*ich h. True*)) *h*›

8.2 Zusammenhang Goldene Regel

Mit der goldenen Regel konnten wir wie folgt moralische Entscheidungen treffen: ⟦*moralisch welt m handlungsabsicht*; *okay m ich* (*handeln ich welt handlungsabsicht*)⟧ ⟹ ∀ *p2*. *okay m ich* (*handeln p2 welt handlungsabsicht*)

In Worten: Wenn eine Handlungsabsicht moralisch ist (nach goldener Regel) und es okay ist für mich diese Handlung auszuführen, dann ist es auch für mich okay, wenn jeder andere diese Handlung mit mir als Opfer ausführt.

Der kategorische Imperativ hebt diese eine Abstraktionsebene höher. Wenn eine Maxime den kategorischen Imperativ erfüllt und es okay ist für mich eine Handlung nach dieser Maxime auszuführen (wie in der goldenen Regel), dann ist diese Handlungsabsicht allgemein moralisch. Die goldene Regel konnte nur folgern, dass eine Handlungsabsicht auch okay ist wenn ich das Opfer wäre, der kategorisch Imperativ schließt, dass eine Handlungsabsicht allgemein moralisch sein muss, wobei beliebige Personen (nicht nur ich) Täter und Opfer sein können.

lemma ‹*kategorischer-imperativ wps welt m* ⟹
  wohlgeformte-handlungsabsicht wps welt ha ⟹
   ausfuehrbar ich welt ha ⟹
  okay m ich (*handeln ich welt ha*) ⟹ *moralisch welt m ha*›

In Worten: Wenn eine Maxime den kategorischen Imperativ erfüllt und es für eine beliebige (wohlgeformte) Handlung auszuführen für mich okay ist diese auszuführen, dann ist diese Handlung moralisch..

8.3 Maximen die den Kategorischen Imperativ immer Erfüllen

Wenn eine Maxime jede Handlungsabsicht als moralisch bewertet, erfüllt diese Maxime den kategorischen Imperativ. Da diese Maxime jede Handlung erlaubt, ist es dennoch eine wohl ungeeignete Maxime.

lemma ‹$\forall$ *ha. moralisch welt maxime ha* $\Longrightarrow$ *kategorischer-imperativ wps welt maxime*›

Eine Maxime die das ich und die Handlung ignoriert erfüllt den kategorischen Imperativ.

lemma *blinde-maxime-katimp*:
‹*kategorischer-imperativ wps welt* (*Maxime* (λ*ich h. m*))›

Eine Maxime welche das *ich* ignoriert, also nur die Handlung global betrachtet, erfüllt den kategorischen Imperativ (mit einigen weiteren Annahmen).

theorem *globale-maxime-katimp*:
fixes *P* :: ‹*'world handlung* $\Rightarrow$ *bool*›
assumes *mhg*: ‹$\forall$ *p. maxime-und-handlungsabsicht-generalisieren wps welt* (*Maxime* (λ*ich*::*'person. P*)) *ha p*›
and *maxime-erlaubt-untaetigkeit*: ‹$\forall$ *p. ist-noop* (*handeln p welt ha*) $\longrightarrow$ *okay* (*Maxime* (λ*ich*::*'person. P*)) *p* (*handeln p welt ha*)›
and *kom*: ‹*wpsm-kommutiert* (*Maxime* (λ*ich*::*'person. P*)) *wps welt*›
and *wps-sym*:
‹$\forall$ *p1 p2 welt. wps p1 p2 welt* = *wps p2 p1 welt*›
and *wps-id*:
‹$\forall$ *p1 p2 welt. wps p1 p2* (*wps p1 p2 welt*) = *welt*›
and *wfh*: ‹*wohlgeformte-handlungsabsicht wps welt ha*›
shows ‹*kategorischer-imperativ-auf ha welt* (*Maxime* (λ*ich*::*'person. P*))›

8.4 Ausführbarer Beispielgenerator

Gegeben sei eine Welt, sowie eine Maxime, und eine Liste von Handlungsabsichten. Wir wollen nun wissen ob die Maxime und Handlungsabsichten wohlgeformt sind, und wenn ja, ob die Maxime auf diesen Handlungsabsichten den kategorischen Imperativ erfüllt, und wie die Handlungen bewertet werden.

definition *alle-moeglichen-handlungen*
:: ‹*'world* $\Rightarrow$ (*'person*::*enum*, *'world*) *handlungsabsicht list* $\Rightarrow$ *'world handlung list*›
where
‹*alle-moeglichen-handlungen welt has* $\equiv$ [*handeln p welt ha. ha* $\leftarrow$ *has*, *p* $\leftarrow$ (*Enum.enum*::*'person list*)]›

lemma *set-alle-moeglichen-handlungen*:
‹*set* (*alle-moeglichen-handlungen welt has*) = {*handeln p welt ha* | *ha p. ha*$\in$*set has*}›

record (*'person*, *'world*) *beipiel* =
bsp-welt :: ‹*'world*›
bsp-erfuellte-maxime :: ‹(*'person*, *'world*) *maxime option*›
bsp-erlaubte-handlungen :: ‹(*'person*, *'world*) *handlungsabsicht list*›
bsp-verbotene-handlungen :: ‹(*'person*, *'world*) *handlungsabsicht list*›

definition *erzeuge-beispiel*
:: ‹('person::enum, 'world) wp-swap ⇒ 'world ⇒
('person, 'world) handlungsabsicht list ⇒ ('person, 'world) maxime
⇒ ('person, 'world) beipiel option›
where
‹erzeuge-beispiel wps welt has m ≡
if (∃ h∈set (alle-moeglichen-handlungen welt has). ¬wohlgeformte-maxime-auf h wps m)
∨ (∃ ha∈set has. ¬ wohlgeformte-handlungsabsicht wps welt ha)
then None
else Some
⦇ bsp-welt = welt,
bsp-erfuellte-maxime = if ∀ ha∈set has. kategorischer-imperativ-auf ha welt m then Some m else None,
bsp-erlaubte-handlungen = [ha←has. moralisch welt m ha],
bsp-verbotene-handlungen = [ha←has. ¬ moralisch welt m ha]
⦈›

Das Ergebnis von *erzeuge-beispiel* ließt sich wie folgt.

- Wenn *bsp-erfuellte-maxime* einen *Some* term enthält ist der *kategorischer-imperativ-auf* den Handlungen erfüllt
- Die *bsp-erlaubte-handlungen* und *bsp-verbotene-handlungen* entspricht quasi dem allgemeinen Gesetz, welches besagt, welche Handlungen erlaubt oder verboten sind.

erzeuge-beispiel erzeugt nur ein Beiespiel wenn alles wohlgeformt ist.

lemma ‹erzeuge-beispiel wps welt has m = Some bsp ⟹
(∀ ha ∈ set has. wohlgeformte-handlungsabsicht wps welt ha) ∧
(∀ h ∈ set (alle-moeglichen-handlungen welt has). wohlgeformte-maxime-auf h wps m)›
lemma‹erzeuge-beispiel swap (λp::person. 0::int) [Handlungsabsicht (λp w. Some w)] (Maxime (λich w. True))
=
Some
⦇bsp-welt = (λp::person. 0::int),
bsp-erfuellte-maxime = Some (Maxime (λich w. True)),
bsp-erlaubte-handlungen = [Handlungsabsicht (λp w. Some w)],
bsp-verbotene-handlungen = []
⦈›

Der Nachteil von *erzeuge-beispiel* ist, dass der resultierende Record viele Funktionen enthält, welche eigentlich nicht geprintet werden können. Allerdings ist dies vermutlich die einzige (sinnvolle, einfache) Art eine Handlungsabsicht darzustellen.

Es wäre einfacher, nur die Handlung (also die *'world handlung*, nur die Welt vorher und nachher, ohne Absicht) aufzuschreiben. Allerdings erzeugt das ohne die Absicht (i.e. *('person, 'world) handlungsabsicht*) sehr viel Unfug, da z.B. pathologische Grenzfälle (wie z.B. sich-selsbt-bestehlen, oder die-welt-die-zufällig-im-ausgangszustand-ist-resetten) dazu, dass diese no-op Handlungen verboten sind, da die dahinterliegende Absicht schlecht ist. Wenn wir allerdings nur die Ergebnisse einer solchen Handlung (ohne die Absicht) aufschreiben kommt heraus: Nichtstun ist verboten.

Glücklicherweise hat Lars uns 4 Zeilen ML geschrieben, welche *erzeuge-beispiel* als ausführbares Beispiel benutzbar macht und dabei es auch erlaubt die Funktionen richtig zu printen, solange diese einen Namen haben.

8.5 Kombination vom Maximen

Die folgenden Lemmata über Konjunktion, Disjunktion, und Negation von Maximen werden leider etwas kompliziert. Wir führen eine Hilfsdefinition ein, welche besagt, ob es einen Fall gibt in dem die Handlungsabsicht tatsächlich ausführbar ist und die Maxime erfüllt. Dabei werden gezielt die pathologischen Grenzfälle ausgeklammert, in denen die Handlungsabsicht nicht ausführbar ist und in einer No-Op resultieren würde.

definition *ex-erfuellbare-instanz*
:: ‹(*'person*, *'world*) *maxime* ⇒ *'world* ⇒ (*'person*, *'world*) *handlungsabsicht* ⇒ *bool*›
where
‹*ex-erfuellbare-instanz m welt ha* ≡
∃*ich*. *ausfuehrbar ich welt ha* ∧ *okay m ich* (*handeln ich welt ha*)›

8.5.1 Konjunktion

lemma *MaximeConjI*:
‹*kategorischer-imperativ-auf ha welt m1* ∧ *kategorischer-imperativ-auf ha welt m2* ⟹
kategorischer-imperativ-auf ha welt (*MaximeConj m1 m2*)›

Die Rückrichtung gilt nur, wenn wir annehmen, dass es auch einen Fall gibt in dem die *MaximeConj* auch erfüllbar ist:

lemma *MaximeConjD*:
‹*ex-erfuellbare-instanz* (*MaximeConj m1 m2*) *welt ha* ⟹
kategorischer-imperativ-auf ha welt (*MaximeConj m1 m2*) ⟹
kategorischer-imperativ-auf ha welt m1 ∧ *kategorischer-imperativ-auf ha welt m2*›

Wenn wir *ex-erfuellbare-instanz* annehmen, dann verhält sich *MaximeConj* im *kategorischer-imperativ-auf* wie eine normale Konjunktion.

lemma *MaximeConj*:
‹*ex-erfuellbare-instanz* (*MaximeConj m1 m2*) *welt ha* ⟹
kategorischer-imperativ-auf ha welt (*MaximeConj m1 m2*) ⟷
kategorischer-imperativ-auf ha welt m1 ∧ *kategorischer-imperativ-auf ha welt m2*›

lemma *kategorischer-imperativ-auf-MaximeConj-comm*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeConj m1 m2*)
⟷ *kategorischer-imperativ-auf ha welt* (*MaximeConj m2 m1*)›

lemma *kategorischer-imperativ-auf-MaximeConj-True*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeConj m1* (*Maxime* (λ- -. *True*)))
⟷ *kategorischer-imperativ-auf ha welt m1*›

Achtung: Folgendes lemma ist das Gegenteil, was man von einer Konjunktion erwarten würde. Normalerweise ist $a \wedge False = False$. Bei *MaximeConj* ist dies aber *True*! Dies liegt daran, dass *Maxime* (λ- -. *False*) keine Handlung erlaubt, und damit als pathologischen Grenzfall den kategorischen Imperativ erfüllt.

lemma *kategorischer-imperativ-auf-MaximeConj-False*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeConj m1* (*Maxime* (λ- -. *False*)))›

8.5.2 Disjunktion

Für *MaximeDisj* müssen wir generell annehmen, dass einer der Fälle erfüllbar ist.

lemma *kategorischer-imperativ-auf-MaximeDisjI*:
‹(*ex-erfuellbare-instanz m1 welt ha* ∧ *kategorischer-imperativ-auf ha welt m1*) ∨
(*ex-erfuellbare-instanz m2 welt ha* ∧ *kategorischer-imperativ-auf ha welt m2*) ⟹
kategorischer-imperativ-auf ha welt (*MaximeDisj m1 m2*)›

Die Rückrichtung gilt leider nicht.

Die Annahmen sind leider sehr stark:

lemma
‹*ex-erfuellbare-instanz m welt ha* ∧ *kategorischer-imperativ-auf ha welt m*
⟹
moralisch welt m ha›

Wenn wir die Annahme stärker machen gilt auch folgendes:

lemma *kategorischer-imperativ-auf-MaximeDisjI-from-conj*:
‹*kategorischer-imperativ-auf ha welt m1* ∧ *kategorischer-imperativ-auf ha welt m2* ⟹
kategorischer-imperativ-auf ha welt (*MaximeDisj m1 m2*)›

Als Introduction rule eignet sich vermutlich folgendes besser, weil es auch erlaubt, dass eine Handlungsabsicht nicht ausführbar ist oder von keiner Maxime erfüllbar ist.

lemma *kategorischer-imperativ-auf-MaximeDisjI2*:
‹(*ex-erfuellbare-instanz m1 welt ha* ∧ *kategorischer-imperativ-auf ha welt m1*) ∨
(*ex-erfuellbare-instanz m2 welt ha* ∧ *kategorischer-imperativ-auf ha welt m2*) ∨
(¬ *ex-erfuellbare-instanz* (*MaximeDisj m1 m2*) *welt ha*)
⟹
kategorischer-imperativ-auf ha welt (*MaximeDisj m1 m2*)›

Die vorherige Introduction Rule lässt sich wie folgt erklären. Mindestens eine der *ex-erfuellbare-instanz*Fälle muss immer zutreffen:

lemma
‹*ex-erfuellbare-instanz m1 welt ha* ∨
ex-erfuellbare-instanz m2 welt ha ∨

¬ *ex-erfuellbare-instanz* (*MaximeDisj m1 m2*) *welt ha*›

Wenn wir also mental den *ex-erfuellbare-instanz* Teil ausblenden, dann liest sich obige Introduction Rule wie folgt: *kategorischer-imperativ-auf ha welt m1* ∨ *kategorischer-imperativ-auf ha welt m2* ⟹ *kategorischer-imperativ-auf ha welt* (*MaximeDisj m1 m2*). Dies ist genau die Disjunktions Introduction Rule die ich gerne hätte. Die gesamte Regel ist leider leicht komplizierter, da der entsprechende Oder-Fall immer mit dem entsprechenden *ex-erfuellbare-instanz* gepaart auftreten muss.

Eine gewöhnliche Introduction Rule (ohne die *ex-erfuellbare-instanz* Teile) gilt leider nicht.

lemma
‹*ha* = *Handlungsabsicht* (λ*p w*. *Some w*) ⟹
m1 = *Maxime* ((λ*p h*. *False*)(*Bob* := λ*h*. *True*)) ⟹
welt = (*0*::*int*) ⟹
kategorischer-imperativ-auf ha welt m1 ⟹
¬ *kategorischer-imperativ-auf ha welt* (*MaximeDisj m1 m2*)›

Zumindest gelten folgende Regeln welche einer gewöhnlichen Disjuntions Introduction ähnlich sehen (mit leicht stärkeren Annahmen):

lemma
‹(*ex-erfuellbare-instanz m1 welt ha* ∧ *kategorischer-imperativ-auf ha welt m1*)
⟹ *kategorischer-imperativ-auf ha welt* (*MaximeDisj m1 m2*)›
‹*moralisch welt m1 ha*
⟹ *kategorischer-imperativ-auf ha welt* (*MaximeDisj m1 m2*)›

lemma *moralisch-kategorischer-imperativ-auf-MaximeDisjI*:
‹*moralisch welt m1 ha* ⟹
kategorischer-imperativ-auf ha welt (*MaximeDisj m1 m2*)›

lemma *kategorischer-imperativ-auf-MaximeDisj-comm*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeDisj m1 m2*)
⟷ *kategorischer-imperativ-auf ha welt* (*MaximeDisj m2 m1*)›

Für die Grenzfälle einer Disjunktion mit *True* und *False* verhält sich *MaximeDisj* wie erwartet.

lemma *kategorischer-imperativ-auf-MaximeDisj-True*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeDisj m1* (*Maxime* (λ- -. *True*)))›
lemma *kategorischer-imperativ-auf-MaximeDisj-False*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeDisj m1* (*Maxime* (λ- -. *False*)))
⟷ *kategorischer-imperativ-auf ha welt m1*›

Die Negation verhält sich wie erwartet.

lemma *kategorischer-imperativ-auf-Maxime-DeMorgan*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeNot* (*MaximeConj m1 m2*))
⟷

kategorischer-imperativ-auf ha welt (*MaximeDisj* (*MaximeNot m1*) (*MaximeNot m2*))›

lemma *kategorischer-imperativ-auf-MaximeNot-double*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeNot* (*MaximeNot m*))
$\longleftrightarrow$ *kategorischer-imperativ-auf ha welt m*›

9 Utilitarismus

Wir betrachten hier primär einen einfachen Handlungsutilitarismus. Frei nach Jeremy Bentham. Sehr frei. Also sehr viel persönliche Auslegung.

Eine Handlung ist genau dann moralisch richtig, wenn sie den aggregierten Gesamtnutzen, d.h. die Summe des Wohlergehens aller Betroffenen, maximiert wird.

type-synonym *'world glueck-messen* = ‹*'world handlung* ⇒ *ereal*›

Wir messen Glück im Typen *ereal*, also reelle Zahlen mit ∞ und $-\infty$, so dass auch "den höchsten Preis zahlen" modelliert werden kann.

lemma ‹(λh::*ereal handlung*. *case h of Handlung vor nach* ⇒ *nach* − *vor*) (*Handlung 3 5*) = *2*›
lemma ‹(λh::*ereal handlung*. *case h of Handlung vor nach* ⇒ *nach* − *vor*) (*Handlung 3* ∞) = ∞›
lemma ‹(λh::*ereal handlung*. *case h of Handlung vor nach* ⇒ *nach* − *vor*) (*Handlung 3* ($-\infty$)) = $-\infty$›

Eine Handlung ist genau dann moralisch richtig, wenn die Gesamtbilanz einen positiven Nutzen aufweist.

definition *moralisch-richtig* :: ‹*'world glueck-messen* ⇒ *'world handlung* ⇒ *bool*› **where**
‹*moralisch-richtig glueck-messen handlung* ≡ (*glueck-messen handlung*) ≥ *0*›

9.1 Goldene Regel und Utilitarismus im Einklang

In §3.1 haben wir Gesinnungsethik und Verantwortungsethik definiert.

In diesem kleinen Intermezzo werden wir zeigen, wie sich die Gesinnungsethik der goldenen Regel in die Verantwortungsethik des Utilitarismus übersetzen lässt.

Wir modellieren die goldene Regel als Gesinnungsethik.

definition *goldene-regel-als-gesinnungsethik*
:: ‹(*'person*, *'world*) *maxime* ⇒ (*'person*, *'world*) *handlungsabsicht* ⇒ *bool*›
where
‹*goldene-regel-als-gesinnungsethik maxime handlungsabsicht* ≡
∀ *welt*. *moralisch welt maxime handlungsabsicht*›

definition *utilitarismus-als-verantwortungsethik*
:: ‹*'world glueck-messen* ⇒ *'world handlung* ⇒ *bool*›
where
‹*utilitarismus-als-verantwortungsethik glueck-messen handlung* ≡

moralisch-richtig glueck-messen handlung›

Eine Maxime ist immer aus Sicht einer bestimmten Person definiert. Wir "neutralisieren" eine Maxime indem wir diese bestimmte Person entfernen und die Maxime so allgemeingültiger machen. Alle Personen müssen gleich behandelt werden. Um die Maxime unabhängig von einer bestimmten Person zu machen, fordern wir einfach, dass die Maxime für aller Personen erfüllt sein muss.

fun *maximeNeutralisieren* :: ‹(*'person*, *'world*) *maxime* ⇒ (*'world handlung* ⇒ *bool*)› **where**
‹*maximeNeutralisieren* (*Maxime m*) = (λ*welt*. ∀ *p*::*'person*. *m p welt*)›

Nun übersetzen wir eine Maxime in die *'world glueck-messen* Funktion des Utilitarismus. Der Trick: eine verletzte Maxime wird als unendliches Leid übersetzt.

```
definition maxime-als-nutzenkalkuel
  :: ‹('person, 'world) maxime ⇒ 'world glueck-messen›
where
  ‹maxime-als-nutzenkalkuel maxime ≡
    (λwelt. case (maximeNeutralisieren maxime) welt
             of True ⇒ 1
              | False ⇒ − ∞)›
```

Für diese Übersetzung können wir beweisen, dass die Gesinnungsethik der goldenen Regel und die utilitaristische Verantwortungsethik konsistent sind:

```
theorem ‹gesinnungsethik-verantwortungsethik-konsistent
        (goldene-regel-als-gesinnungsethik maxime)
        (utilitarismus-als-verantwortungsethik (maxime-als-nutzenkalkuel maxime))›
```

Diese Konsistenz gilt nicht im allgemeinen, sondern nur wenn Glück gemessen wird mit Hilfe der *maxime-als-nutzenkalkuel* Funktion. Der Trick dabei ist nicht, dass wir einer verletzten Maxime $-\infty$ Nutzen zuordnen, sondern der Trick besteht in *maximeNeutralisieren*, welche nicht erlaubt Glück aufzuaddieren und mit Leid zu verrechnen, sondern dank des Allquantors dafür sorgt, dass auch nur das kleinste Leid dazu führt, dass sofort *False* zurückgegebn wird.

Aber auch wenn wir ordentlich aufsummieren, jedoch einer verletzten Maxime $-\infty$ Nutzen zuordnen und zusätzlich annehmen, dass die Bevölkerung endlich ist, dann funktioniert das auch:

```
fun maxime-als-summe-wohlergehen
  :: ‹('person, 'world) maxime ⇒ 'world glueck-messen›
where
  ‹maxime-als-summe-wohlergehen (Maxime m) =
    (λwelt. ∑p∈bevoelkerung. (case m p welt
                              of True ⇒ 1
                               | False ⇒ − ∞))›
theorem
  fixes maxime :: ‹('person, 'world) maxime›
  assumes ‹finite (bevoelkerung:: 'person set)›
  shows
    ‹gesinnungsethik-verantwortungsethik-konsistent
      (goldene-regel-als-gesinnungsethik maxime)
```

(*utilitarismus-als-verantwortungsethik* (*maxime-als-summe-wohlergehen maxime*))›

"Wie zu erwarten, will Kant nichts vom Utilitarismus oder sonstigen Lehren wissen, die der Moral einen außerhalb ihrer selbst liegenden Zweck zuschreiben" [1]. Die eben bewiesene Konsitenz von Gesinnungsethik und Verantwortungsethik zeigt, das unsere Grunddefinitionen bereits eine Formalisierung des Kategorischen Imperativs komplett im strengen Sinne Kants ausschließen. Dennoch finde ich unsere Interpretation bis jetzt nicht abwegig. Der große Trick besteht darin, dass wir eine (*'person*, *'world*) *handlungsabsicht* sehr einfach in eine *'world handlung* in unserem theoretischen Modell überführen können. Die widerspricht Kants Grundannahme, dass die Folgen einer Handlungsabsicht unvorhersehbar sind.

10 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* allgemein 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* −= *n*)⟧ ≡ (*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* += *n*)⟧(*p* −= *n*)⟧ = *f*›

Diskriminierungsfrei eine *'person* eindeutig anhand Ihres Besitzes auswählen:

definition *opfer-eindeutig-nach-besitz-auswaehlen*
  :: ‹*int* ⇒ (*'person* ⇒ *int*) ⇒ *'person list* ⇒ *'person option*› **where**
  ‹*opfer-eindeutig-nach-besitz-auswaehlen b besitz ps* =
    (*case filter* (λ*p*. *besitz p* = *b*) *ps*
      of [*opfer*] ⇒ *Some opfer*
      | - ⇒ *None*)›
definition *the-single-elem* :: ‹*'a set* ⇒ *'a option*› **where**
  ‹*the-single-elem s* ≡ *if card s* = *1 then Some* (*Set.the-elem s*) *else None*›

thm *is-singleton-the-elem*[*symmetric*]
lemma ‹*A* = {*the-elem A*} ⟷ *is-singleton A*›

lemma *opfer-nach-besitz-induct-step-set-simp*: ‹*besitz a* ≠ *opfer-nach-besitz* ⟹
  {*p*. (*p* = *a* ∨ *p* ∈ *set ps*) ∧ *besitz p* = *opfer-nach-besitz*} =
    {*p* ∈ *set ps*. *besitz p* = *opfer-nach-besitz*}›

lemma *opfer-eindeutig-nach-besitz-auswaehlen-the-single-elem*:
  ‹*distinct ps* ⟹
  opfer-eindeutig-nach-besitz-auswaehlen opfer-nach-besitz besitz ps =
      the-single-elem {*p* ∈ *set ps*. *besitz p* = *opfer-nach-besitz*}›

lemma *opfer-eindeutig-nach-besitz-auswaehlen-the-single-elem-enumall*:
  ‹*opfer-eindeutig-nach-besitz-auswaehlen opfer-nach-besitz besitz enum-class.enum* =
      the-single-elem {*p*. *besitz p* = *opfer-nach-besitz*}›
fun *stehlen* :: ‹*int* ⇒ *int* ⇒ *'person*::*enum* ⇒ (*'person* ⇒ *int*) ⇒ (*'person* ⇒ *int*) *option*› **where**
  ‹*stehlen beute opfer-nach-besitz dieb besitz* =
    (*case opfer-eindeutig-nach-besitz-auswaehlen opfer-nach-besitz besitz Enum.enum*
      of None ⇒ *None*
      | *Some opfer* ⇒ *if opfer* = *dieb then None else Some* ⟦⟦*besitz*(*opfer* −= *beute*)⟧(*dieb* += *beute*)⟧
)›

lemma *wohlgeformte-handlungsabsicht-stehlen*:
  ‹*wohlgeformte-handlungsabsicht swap welt* (*Handlungsabsicht* (*stehlen n p*))›

definition *aufsummieren* :: ‹(*'person*::*enum* ⇒ *int*) ⇒ *int*› **where**
  ‹*aufsummieren besitz* = *sum-list* (*map besitz Enum.enum*)›

lemma ‹*aufsummieren* (*besitz* :: *person*⇒*int*) = (∑*p*←[*Alice*,*Bob*,*Carol*,*Eve*]. *besitz p*)›

lemma ‹*aufsummieren* 🌏[*Alice* := *4*, *Carol* := *8*] = *12*›

lemma ‹*aufsummieren* 🌍[*Alice* := *4*, *Carol* := *4*] = *8*›

lemma *aufsummieren-swap*:
‹*aufsummieren* (*swap p1 p2 welt*) = *aufsummieren welt*›

lemma *list-not-empty-iff-has-element*: ‹*as* ≠ [] ⟷ (∃ *a*. *a* ∈ *set as*)›

lemma *enum-class-not-empty-list*: ‹*enum-class.enum* ≠ []›

lemma *alles-kaputt-machen-code-help*:
‹(λ-. *Min* (*range x*) − *1*) = (λ-. *min-list* (*map x enum-class.enum*) − *1*)›

swap funktioniert auch auf Mengen.

lemma ‹(*swap Alice Carol id*) ‘ {*Alice*, *Bob*} = {*Carol*, *Bob*}›

11 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* ⇒ *int* — besitz: Besitz jeder Person.›

fun *gesamtbesitz* :: ‹*zahlenwelt* ⇒ *int*› **where**
‹*gesamtbesitz* (*Zahlenwelt besitz*) = *aufsummieren besitz*›

Beispiel:

lemma ‹*gesamtbesitz* (*Zahlenwelt* 🌍[*Alice* := *4*, *Carol* := *8*]) = *12*›
lemma ‹*gesamtbesitz* (*Zahlenwelt* 🌍[*Alice* := *4*, *Carol* := *4*]) = *8*›

Mein persönlicher Besitz:

fun *meins* :: ‹*person* ⇒ *zahlenwelt* ⇒ *int*› **where**
‹*meins p* (*Zahlenwelt besitz*) = *besitz p*›

Beispiel:

lemma ‹*meins Carol* (*Zahlenwelt* 🌍[*Alice* := *8*, *Carol* := *4*]) = *4*›

Um den `SchleierNichtwissen.thy` zu implementieren:

```
fun zahlenwps :: ‹person ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt› where
  ‹zahlenwps p1 p2 (Zahlenwelt besitz) = Zahlenwelt (swap p1 p2 besitz)›
```

Beispiel:

```
lemma ‹zahlenwps Alice Carol (Zahlenwelt 🌏[Alice := 4, Bob := 6, Carol := 8])
  = (Zahlenwelt 🌏[Alice := 8, Bob := 6, Carol := 4])›
```

Alice hat Besitz, *Bob* ist reicher, *Carol* hat Schulden.

```
definition ‹initialwelt ≡ Zahlenwelt 🌏[Alice := 5, Bob := 10, Carol := −3]›
```

11.1 Ungültige Handlung

Sobald ich eine konkrete Person in einer Handlungsabsicht hardcode, ist diese nicht mehr wohlgeformt.

```
lemma ‹¬wohlgeformte-handlungsabsicht
  zahlenwps (Zahlenwelt 🌏[Alice := 5, Bob := 10, Carol := −3])
  (Handlungsabsicht (λich w. if ich = Alice then Some w else Some (Zahlenwelt (λ-. 0))))›
```

11.2 Nicht-Wohlgeformte Handlungen

```
fun stehlen-nichtwf :: ‹int ⇒ person ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹stehlen-nichtwf beute opfer dieb (Zahlenwelt besitz) =
      Some (Zahlenwelt ⟦⟦besitz(opfer −= beute)⟧(dieb += beute)⟧)›
```

Die Handlung *stehlen* diskriminiert und ist damit nicht wohlgeformt:

```
lemma ‹wohlgeformte-handlungsabsicht-gegenbeispiel zahlenwps
    (Zahlenwelt (λx. 0)) (Handlungsabsicht (stehlen-nichtwf 5 Bob))
    Alice Bob›
```

Wir versuchen, das Opfer nach Besitz auszuwählen, nicht nach Namen. Nach unserer Definition ist der Besitz ein Merkmal, nach dem man diskriminieren darf. Man darf nur nicht nach Eigenschaften der *person* diskriminieren, sondern nur nach Eigenschaften der *zahlenwelt*.

```
fun opfer-nach-besitz-auswaehlen :: ‹int ⇒ ('person ⇒ int) ⇒ 'person list ⇒ 'person option› where
  ‹opfer-nach-besitz-auswaehlen - - [] = None›
| ‹opfer-nach-besitz-auswaehlen b besitz (p#ps) =
    (if besitz p = b then Some p else opfer-nach-besitz-auswaehlen b besitz ps)›

fun stehlen-nichtwf2 :: ‹int ⇒ int ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt option› where
    ‹stehlen-nichtwf2 beute opfer-nach-besitz dieb (Zahlenwelt besitz) =
      (case opfer-nach-besitz-auswaehlen opfer-nach-besitz besitz Enum.enum
         of None ⇒ None
          | Some opfer ⇒ Some (Zahlenwelt ⟦⟦besitz(opfer −= beute)⟧(dieb += beute)⟧)
      )›
```

Leider ist diese Funktion auch diskriminierend: Wenn es mehrere potenzielle Opfer mit dem gleichen Besitz gibt, dann bestimmt die Reihenfolge in *enum-class.enum* wer bestohlen wird. Diese Reihenfolge ist wieder eine Eigenschaft von *person* und nicht *zahlenwelt*.

lemma‹*handeln Alice* (*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := −*3*])
(*Handlungsabsicht* (*stehlen-nichtwf2 5 10*))
= *Handlung* (*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := − *3*])
(*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := − *3*])›
lemma‹*handeln Bob* (*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := −*3*])
(*Handlungsabsicht* (*stehlen-nichtwf2 5 10*))
= *Handlung* (*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := − *3*])
(*Zahlenwelt* 🌏[*Alice* := *5*, *Bob* := *15*, *Carol* := − *3*])›
lemma ‹*wohlgeformte-handlungsabsicht-gegenbeispiel*
zahlenwps
(*Zahlenwelt* 🌏[*Alice* := *10*, *Bob* := *10*, *Carol* := −*3*]) (*Handlungsabsicht* (*stehlen-nichtwf2 5 10*))
Alice Bob›

fun *schenken* :: ‹*int* ⇒ *person* ⇒ *person* ⇒ *zahlenwelt* ⇒ *zahlenwelt option*› **where**
‹*schenken betrag empfaenger schenker* (*Zahlenwelt besitz*) =
Some (*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-nichtwf i* = *schenken* (−*i*)›

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

11.3 Wohlgeformte Handlungen

Die folgende Handlung erschafft neuen Besitz aus dem Nichts:

fun *erschaffen* :: ‹*nat* ⇒ *person* ⇒ *zahlenwelt* ⇒ *zahlenwelt option*› **where**
‹*erschaffen i p* (*Zahlenwelt besitz*) = *Some* (*Zahlenwelt* ⟦*besitz*(*p* += *int i*)⟧)›
lemma ‹*wohlgeformte-handlungsabsicht zahlenwps welt* (*Handlungsabsicht* (*erschaffen n*))›

Wenn wir das Opfer eindeutig auswählen, ist die Handlungsabsicht "Stehlen" wohlgeformt. Allerdings wird niemand bestohlen, wenn das Opfer nicht eindeutig ist.

fun *stehlen4* :: ‹*int* ⇒ *int* ⇒ *person* ⇒ *zahlenwelt* ⇒ *zahlenwelt option*› **where**
‹*stehlen4 beute opfer-nach-besitz dieb* (*Zahlenwelt besitz*) =
map-option Zahlenwelt (*stehlen beute opfer-nach-besitz dieb besitz*)›

Reset versetzt die Welt wieder in den Ausgangszustand. Eine sehr destruktive Handlung.

fun *reset* :: ‹*person* ⇒ *zahlenwelt* ⇒ *zahlenwelt option*› **where**
‹*reset ich* (*Zahlenwelt besitz*) = *Some* (*Zahlenwelt* (λ -. *0*))›

Der *reset* ist im moralischen Sinne vermutlich keine gute Handlung, dennoch ist es eine wohlgeformte Handlung, welche wir betrachten können:

lemma ‹*wohlgeformte-handlungsabsicht zahlenwps welt* (*Handlungsabsicht reset*)›

```
fun alles-kaputt-machen :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹alles-kaputt-machen ich (Zahlenwelt besitz) = Some (Zahlenwelt (λ -. Min (besitz ‘ UNIV) − 1))›
lemma ‹alles-kaputt-machen Alice (Zahlenwelt 🌍[Alice := 5, Bob := 10, Carol := −3])
  = Some (Zahlenwelt 🌍[Alice := −4, Bob := −4, Carol := −4, Eve := −4])›
```

Auch die unmögliche (niemals ausführbare) Handlung lässt sich modellieren.

```
fun unmoeglich :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹unmoeglich - - = None›
```

Folgende Funktion ist inspiriert durch das https://de.wikipedia.org/wiki/Collatz-Problem.

```
fun collatz:: ‹int ⇒ int› where
    ‹collatz n = (if n mod 2 = 0 then n div 2 else 3*n + 1)›
lemma ‹collatz 19 = 58›
```

Es folgt eine Handlungsabsicht, basierend auf dem Collatz-Problem. Das eigentliche Collatz-Problem ist an dieser stelle nicht relevant, da wir nur eine Iteration machen. Allerdings ist das eine spannende Handlungsabsicht, da diese sowohl den Besitz erhöhen kann, aber auch verringern kann.

```
fun collatzh:: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
    ‹collatzh ich (Zahlenwelt besitz) = Some (Zahlenwelt (besitz( ich := collatz (besitz ich))))›
```

Die Handlungsabsicht *collatzh* ist tatsächlich immer wohlgeformt.

```
lemma ‹wohlgeformte-handlungsabsicht zahlenwps welt (Handlungsabsicht collatzh)›
```

Allerdings werden wir *collatzh* nicht weiter betrachten. Das Ergebnis vorweg: Ein kategorischer Imperativ, egal welche vielversprechende Maxime, gilt nicht für die Handlungsabsicht *collatzh*. Der Grund ist, oberflächlich gesprochen, dass diese Handlungsabsicht keinen eindeutigen Charakter hat. Die Handlungsabsicht kann sowohl Besitz verringern als auch vermehren. In vielen Welten wird es Leute geben, für die *collatzh* eine positive Wirkung hat. Jedoch ist *collatzh* wohl allgemein nicht *moralisch*, da es normalerweise auch Leute gibt, für die *collatzh* eine negative Auswirkung hat. Daher kann eine Maxime *collatzh* nicht allgemein beurteilen. Jedoch ist auch diese Meta-Aussage eine spannende Aussage: Der kategorische Imperativ sagt (dadurch, dass er nicht erfüllt ist), dass die Handlungsabsicht *collatz* nicht durch eine unserer Maximen beurteilt werden sollte, bzw. sollten wir ein allgemeines Gesetz bauen wollen, so können wir weder *collatzh* uneingeschränkt in die Liste erlaubter Handlungsabsichten aufnehmen, noch können wir uneingeschränkt *collatzh* uneingeschränkt in die Liste verbotener Handlungsabsichten aufnehmen.

Die Beispielhandlungsabsichten, die wir betrachten wollen. Wir lassen *collatzh* mal aus.

```
definition ‹handlungsabsichten ≡ [
  Handlungsabsicht (erschaffen 5),
  Handlungsabsicht (stehlen4 5 10),
  Handlungsabsicht reset,
  Handlungsabsicht alles-kaputt-machen,
  Handlungsabsicht unmoeglich
]›
```

lemma *wfh-handlungsabsichten*:
‹*ha* ∈ *set handlungsabsichten* ⟹ *wohlgeformte-handlungsabsicht zahlenwps welt ha*›

11.4 Maxime für individuellen Fortschritt

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* (λ*ich*. *individueller-fortschritt ich*)›

reset erfüllt das nicht, aber das normale *stehlen*.

lemma ‹*ha* ∈ {
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen-nichtwf 5 Bob*),
Handlungsabsicht (*stehlen4 5 10*),
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich
} ⟹ *maxime-und-handlungsabsicht-generalisieren zahlenwps welt maxime-zahlenfortschritt ha p*›

Nicht alle Handlungen generalisieren, z.B. *reset* und *collatzh* nicht:

lemma
‹¬ *maxime-und-handlungsabsicht-generalisieren*
zahlenwps (*Zahlenwelt* 🌏[*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*])
maxime-zahlenfortschritt (*Handlungsabsicht reset*) *Alice*›

lemma
‹¬ *maxime-und-handlungsabsicht-generalisieren*
zahlenwps (*Zahlenwelt* 🌏[*Alice* := *2*, *Bob* := *3*])
maxime-zahlenfortschritt (*Handlungsabsicht collatzh*) *Alice*›

Die *maxime-zahlenfortschritt* erfüllt **nicht** den *kategorischer-imperativ* da *Alice* nach der Maxime z.B. *Bob* bestehlen dürfte.

lemma ‹*kategorischer-imperativ-gegenbeispiel*
zahlenwps initialwelt maxime-zahlenfortschritt
(*Handlungsabsicht* (*stehlen4 1 10*))
Alice
Bob
Alice›

11.4.1 Einzelbeispiele

In jeder Welt ist die *Handlungsabsicht* (*erschaffen n*) *moralisch*:

lemma ‹*moralisch welt maxime-zahlenfortschritt* (*Handlungsabsicht* (*erschaffen n*))›

In kein Welt ist Stehlen *moralisch*:

lemma ‹¬ *moralisch welt maxime-zahlenfortschritt* (*Handlungsabsicht* (*stehlen-nichtwf 5 Bob*))›

In unserer *initialwelt* in der *Bob* als Opfer anhand seines Besitzes als Opfer eines Diebstahls ausgewählt würde, ist stehlen dennoch nicht *moralisch*, obwohl die Handlungsabsicht wohlgeformt ist:

lemma ‹¬ *moralisch initialwelt maxime-zahlenfortschritt* (*Handlungsabsicht* (*stehlen4 5 10*))›

Da Schenken und Stehlen in dieser Welt equivalent ist, ist Schenken auch unmoralisch:

lemma ‹¬ *moralisch welt maxime-zahlenfortschritt* (*Handlungsabsicht* (*schenken 5 Bob*))›

TODO: erklaeren

lemma ‹*erzeuge-beispiel*
zahlenwps initialwelt
handlungsabsichten
(*Maxime individueller-fortschritt*) =
Some
⦇*bsp-welt* = *Zahlenwelt* 🌍[*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*],
bsp-erfuellte-maxime = *None*,
bsp-erlaubte-handlungen = [*Handlungsabsicht* (*erschaffen 5*), *Handlungsabsicht unmoeglich*],
bsp-verbotene-handlungen = [*Handlungsabsicht* (*stehlen4 5 10*), *Handlungsabsicht reset*, *Handlungsabsicht alles-kaputt-machen*]⦈›

11.5 Maxime für allgemeinen Fortschritt

Allerdings können wir die Maxime generalisieren, indem wir *individueller-fortschritt* für jeden fordern. Effektiv wird dabei das *ich* ignoriert.

definition *maxime-altruistischer-fortschritt* :: ‹(*person*, *zahlenwelt*) *maxime*› **where**
‹*maxime-altruistischer-fortschritt* ≡ *Maxime* (λ*ich h*. ∀ *pX*. *individueller-fortschritt pX h*)›

Folgendes Beispiel zeigt, dass die *maxime-altruistischer-fortschritt* den kategorischen Imperativ (für diese *initialwelt* und *handlungsabsichten*) erfüllt; zu sehen an dem *Some* Term im *bsp-erfuellte-maxime*.

Die Handlungsabsichten werden eingeordnet wie erwartet: *erschaffen* ist gut, *stehlen4*, *reset*, *alles-kaputt-machen* ist schlecht.

lemma ‹*erzeuge-beispiel*
zahlenwps initialwelt
handlungsabsichten
maxime-altruistischer-fortschritt =
Some
⦇*bsp-welt* = *Zahlenwelt* 🌍[*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*],

```
    bsp-erfuellte-maxime = Some maxime-altruistischer-fortschritt,
    bsp-erlaubte-handlungen = [Handlungsabsicht (erschaffen 5), Handlungsabsicht unmoeglich],
    bsp-verbotene-handlungen = [Handlungsabsicht (stehlen4 5 10), Handlungsabsicht reset, Handlungsabsicht
alles-kaputt-machen]⦈›
```

Das ist ein sehr schönes Beispiel.

Die Aussage, dass die *maxime-altruistischer-fortschritt* den kategorischen Imperativ für bestimmte Handlungsabsichten und Welten erfüllt generalisiert noch weiter. Für alle Welten und alle wohlgeformten Handlungsabsichten welche mit der Maxime generalisieren erfüllt die Maxime den kategorischen Imperativ.

```
theorem kapimp-maxime-altruistischer-fortschritt: ‹
  ∀ p. maxime-und-handlungsabsicht-generalisieren zahlenwps welt maxime-altruistischer-fortschritt ha p ⟹
  wohlgeformte-handlungsabsicht zahlenwps welt ha ⟹
  kategorischer-imperativ-auf ha welt maxime-altruistischer-fortschritt›
```

Allgemein scheint dies eine sehr gute Maxime zu sein (für dieses sehr beschränkte Weltenmodell).

```
corollary ‹ha ∈ set handlungsabsichten ⟹
  kategorischer-imperativ-auf ha welt maxime-altruistischer-fortschritt›
```

Dies wirft die Frage auf: "gibt es überhaupt wohlgeformte Handlungsabsichten, welche nicht mit *maxime-altruistischer-fortschritt* generalisieren?" Die Antwort liefert *collatzh*.

```
lemma
    ‹¬ maxime-und-handlungsabsicht-generalisieren
      zahlenwps (Zahlenwelt 🌏[Alice := 2, Bob := 3])
      maxime-altruistischer-fortschritt (Handlungsabsicht collatzh) Alice›
```

11.6 Maxime für strikten individuellen Fortschritt

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)›
```

TODO: erklaeren. Erfuellt nicht kategorischen imperativ und alles ist verboten

```
lemma ‹erzeuge-beispiel
  zahlenwps initialwelt
  handlungsabsichten
  (Maxime individueller-strikter-fortschritt) =
Some
  ⦇bsp-welt = Zahlenwelt 🌏[Alice := 5, Bob := 10, Carol := −3],
   bsp-erfuellte-maxime = None,
   bsp-erlaubte-handlungen = [],
```

bsp-verbotene-handlungen = *handlungsabsichten*⦈›

In keiner Welt ist die Handlung *erschaffen* nun *moralisch*:

lemma ‹¬ *moralisch welt*
(*Maxime* (λ*ich*. *individueller-strikter-fortschritt ich*)) (*Handlungsabsicht* (*erschaffen 5*))›

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
‹⦇
dbg-opfer = *Bob*, *dbg-taeter* = *Alice*,
dbg-handlung = *Handlung* [(*Alice*, *5*), (*Bob*, *10*), (*Carol*, −*3*)] [(*Alice*, *10*), (*Bob*, *10*), (*Carol*, −*3*)]
⦈
∈ *debug-maxime show-zahlenwelt initialwelt*
(*Maxime* (λ*ich*. *individueller-strikter-fortschritt ich*)) (*Handlungsabsicht* (*erschaffen 5*)) ›

11.7 Maxime für globales striktes 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 ‹*moralisch initialwelt*
(*Maxime* (λ*ich*. *globaler-strikter-fortschritt*)) (*Handlungsabsicht* (*erschaffen 5*))›

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

lemma ‹¬*moralisch initialwelt*
(*Maxime* (λ*ich*. *globaler-strikter-fortschritt*)) (*Handlungsabsicht* (*erschaffen 0*))›

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

lemma ‹*moralisch initialwelt*
maxime-zahlenfortschritt (*Handlungsabsicht* (*erschaffen 0*))›

TODO: erklaeren.

```
lemma ‹erzeuge-beispiel
  zahlenwps initialwelt
  handlungsabsichten
  (Maxime (λich. globaler-strikter-fortschritt)) =
Some
  ⦇bsp-welt = Zahlenwelt 🌍[Alice := 5, Bob := 10, Carol := −3],
   bsp-erfuellte-maxime = Some (Maxime (λich. globaler-strikter-fortschritt)),
   bsp-erlaubte-handlungen = [Handlungsabsicht (erschaffen 5)],
   bsp-verbotene-handlungen = [
    Handlungsabsicht (stehlen4 5 10),
    Handlungsabsicht reset,
    Handlungsabsicht alles-kaputt-machen,
    Handlungsabsicht unmoeglich]⦈›
```

11.8 Maxime für globales Optimum

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 ‹moralisch initialwelt
        (Maxime (λich. globaler-fortschritt)) (Handlungsabsicht (erschaffen 0))›
theorem
‹∀ p. maxime-und-handlungsabsicht-generalisieren zahlenwps welt
     (Maxime (λich. globaler-fortschritt)) ha p ⟹
 wohlgeformte-handlungsabsicht zahlenwps welt ha ⟹
  kategorischer-imperativ-auf ha welt (Maxime (λich::person. globaler-fortschritt))›
```

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

```
lemma ‹moralisch initialwelt
        (Maxime (λich. globaler-fortschritt)) (Handlungsabsicht (stehlen-nichtwf 5 Bob))›
lemma ‹moralisch initialwelt
        (Maxime (λich. globaler-fortschritt)) (Handlungsabsicht (stehlen4 5 10))›
```

TODO: erklaeren.

```
lemma ‹erzeuge-beispiel
  zahlenwps initialwelt
  handlungsabsichten
  (Maxime (λich. globaler-fortschritt)) =
Some
  ⦇bsp-welt = Zahlenwelt 🌍[Alice := 5, Bob := 10, Carol := −3],
   bsp-erfuellte-maxime = Some (Maxime (λich. globaler-fortschritt)),
   bsp-erlaubte-handlungen = [
```

Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen4 5 10*),
Handlungsabsicht unmoeglich],
bsp-verbotene-handlungen = [
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen]⦈›

11.9 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*))›
lemma ‹¬*wohlgeformte-maxime-auf*
(*handeln Alice initialwelt* (*Handlungsabsicht* (*stehlen4 5 10*))) *zahlenwps*
(*Maxime* (λ*ich. individueller-fortschritt Alice*))›
lemma ‹*wohlgeformte-maxime-auf*
(*handeln Alice initialwelt* (*Handlungsabsicht* (*stehlen4 5 10*))) *zahlenwps*
(*Maxime* (λ*ich. individueller-fortschritt ich*))›

12 Änderungen in Welten

datatype (*'person*, *'etwas*) *aenderung* = *Verliert* ‹*'person*› ‹*'etwas*› | *Gewinnt* ‹*'person*› ‹*'etwas*›

Beispiel: [*Gewinnt Alice 3*, *Verliert Bob 3*].

12.1 Deltas

Deltas, d.h. Unterschiede zwischen Welten.

type-synonym (*'world*, *'person*, *'etwas*) *delta* =
‹*'world handlung* ⇒ ((*'person*, *'etwas*) *aenderung*) *list*›

Von einer (*'person*, *'etwas*) *aenderung* betroffene.

definition *betroffen* :: ‹(*'person*, *'etwas*) *aenderung* ⇒ *'person*›
where
‹*betroffen a* ≡ *case a of Verliert p -* ⇒ *p* | *Gewinnt p -* ⇒ *p*›

definition *betroffene* :: ‹(*'person*, *'etwas*) *aenderung list* ⇒ *'person list*›
where
‹*betroffene as* ≡ *map betroffen as*›

lemma ‹*betroffene* [*Verliert Alice* (*2::int*), *Gewinnt Bob 3*, *Gewinnt Carol 2*, *Verliert Eve 1*]

```
  = [Alice, Bob, Carol, Eve]›
lemma ‹betroffene [Verliert Alice (5::nat), Gewinnt Bob 3, Verliert Eve 7]
  = [Alice, Bob, Eve]›
lemma ‹betroffene [Verliert Alice (5::nat), Gewinnt Alice 3]
  = [Alice, Alice]›

fun aenderung-ausfuehren
  :: ‹('person, 'etwas::{plus,minus}) aenderung list ⇒ ('person ⇒ 'etwas) ⇒ ('person ⇒ 'etwas)›
where
  ‹aenderung-ausfuehren [] bes = bes›
| ‹aenderung-ausfuehren (Verliert p n # deltas) bes = aenderung-ausfuehren deltas ⟦bes(p −= n)⟧›
| ‹aenderung-ausfuehren (Gewinnt p n # deltas) bes = aenderung-ausfuehren deltas ⟦bes(p += n)⟧›

lemma
‹aenderung-ausfuehren
  [Verliert Alice (2::int), Gewinnt Bob 3, Gewinnt Carol 2, Verliert Eve 1]
  (🌏[Alice:=8, Bob:=3, Eve:= 5])
  =
  (🌏[Alice:=6, Bob:=6, Carol:=2, Eve:= 4])›

lemma
‹aenderung-ausfuehren
  [Verliert Alice (2::int), Verliert Alice 6]
  (🌏[Alice:=8, Bob:=3, Eve:= 5])
  =
  (🌏[Bob:=3, Eve:= 5])›
```

12.2 Abmachungen

Eine *('person, 'etwas) aenderung list* wie z.B. *[Gewinnt Alice 3, Verliert Bob 3]* ließe sich gut verwenden, um eine Abmachung zwischen *Alice* und *Bob* zu modellieren. Allerdings ist diese Darstellung unpraktisch zu benutzen. Beispielsweise sind *[Gewinnt Alice 3, Verliert Bob 3]*, *[Verliert Bob 3, Gewinnt Alice 3]*, *[Gewinnt Alice 1, Gewinnt Alice 1, Gewinnt Alice 1, Verliert Bob 3, Verliert Carol 0]*, extensional betrachtet alle equivalent. Es ist praktischer, eine Darstellung zu wählen, in der syntaktische und semantische Äquivalenz zusammenfallen. Das bedeutet, eine Abmachung muss eindeutig dargestellt werden. Ein Kandidat dafür wäre eine Map *'person ⇀ 'etwas*, da diese eindeutig einer *'person* ein *'etwas* zuordnet. Dies funktioniert allerdings nur, wenn *'etwas* mit Plus und Minus dargestellt werden kann, um *Gewinnt* und *Verliert* darzustellen. Allerdings ist auch diese Darstellung nicht eindeutig, da z.B. *[Alice ↦ 0::'a] = Map.empty* semantisch gilt, solange *0::'a* ein neutrales Element ist. Deshalb stellen wir eine Abmachung als eine totale Funktion *'person ⇒ 'etwas* dar. *(λ-. 0::'a)(Alice := 3::'a, Bob := − (3::'a))* bedeutet *Alice* bekommt 3, *Bob* verliert 3.

```
type-synonym ('person, 'etwas) abmachung = ‹'person ⇒ 'etwas›


fun to-abmachung
  :: ‹('person, 'etwas::{ord,zero,plus,minus,uminus}) aenderung list ⇒ ('person, 'etwas) abmachung›
```

where
  ‹*to-abmachung* [] = (λ*p*. *0*)›
| ‹*to-abmachung* (*delta* # *deltas*) =
    ⟦(*to-abmachung deltas*)(*betroffen delta* += *aenderung-val delta*)⟧›

lemma ‹[*to-abmachung* [*Gewinnt Alice* (*3*::*int*)], *to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*]]
        = [(λ*p*.*0*)(*Alice* := *3*), (λ*p*.*0*)(*Alice* := *3*, *Bob* := −*3*)]›

definition *abmachung-ausfuehren*
  :: ‹('*person*, '*etwas*::{*plus*,*minus*}) *abmachung* ⇒ ('*person* ⇒ '*etwas*) ⇒ ('*person* ⇒ '*etwas*)›
where
  ‹*abmachung-ausfuehren a besitz* ≡ λ*p*. *a p* + (*besitz p*)›

Beispiel:

lemma
  ‹*abmachung-ausfuehren*
    (*to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*])
    (🌏[*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*])
  = (🌏[*Alice*:=*11*, *Bob*:=*0*, *Eve*:= *5*])›

Laut https://de.wikipedia.org/wiki/Konsens#Konsens_im_Rechtssystem lässt sich Konsens wie folg definieren: "die Übereinstimmung der Willenserklärungen beider Vertragspartner über die Punkte des Vertrages". Wir können also *to-abmachung* [*Gewinnt Alice* (*3*::'*a*), *Verliert Bob* (*3*::'*a*)] verwenden, um Konsens zu modellieren. Dabei müssen alle Betroffenen die gleiche Vorstellung der Abmachung haben. Beispielsweise lässt sich der gesamte Konsens in einer Welt darstellen als '*person* ⇒ ('*person*, '*etwas*) *abmachung list*, wobei jeder person genau die Abmachungen zugeordnet werden, deren sie zustimmt. Die Abmachungen sind in einer Liste und keiner Menge, da eine Person eventuell bereit ist, Abmachungen mehrfach auszuführen.

type-synonym ('*person*, '*etwas*) *globaler-konsens* = ‹'*person* ⇒ ('*person*, '*etwas*) *abmachung list*›

definition *abmachungs-betroffene* :: ‹('*person*::*enum*, '*etwas*::*zero*) *abmachung* ⇒ '*person list*›
where
  ‹*abmachungs-betroffene a* ≡ [*p*. *p* ← *Enum.enum*, *a p* ≠ *0*]›

lemma ‹*abmachungs-betroffene* (*to-abmachung* [*Gewinnt Bob* (*3*::*int*), *Verliert Alice 3*])
  = [*Alice*, *Bob*]›

definition *enthaelt-konsens*
  :: ‹('*person*::*enum*, '*etwas*::*zero*) *abmachung* ⇒ ('*person*, '*etwas*) *globaler-konsens* ⇒ *bool*›
where
  ‹*enthaelt-konsens abmachung konsens* ≡ ∀ *betroffene-person* ∈ *set* (*abmachungs-betroffene abmachung*).
      abmachung ∈ *set* (*konsens betroffene-person*)›

lemma *enthaelt-konsens-swap*:
  ‹*enthaelt-konsens* (*swap p1 p2 a*) (*konsensswap p1 p2 konsens*) = *enthaelt-konsens a konsens*›

Eine (ausgeführte) Abmachung einlösen, bzw. entfernen.

definition *konsens-entfernen*
:: ‹(*'person*::*enum*, *'etwas*::*zero*) *abmachung* ⇒ (*'person* ⇒ (*'person*, *'etwas*) *abmachung list*)
⇒ (*'person* ⇒ (*'person*, *'etwas*) *abmachung list*)›
where
‹*konsens-entfernen abmachung kons* =
fold (λ*p k*. *k*(*p* := *remove1 abmachung* (*k p*))) (*abmachungs-betroffene abmachung*) *kons*›

lemma
‹*konsens-entfernen*
(*to-abmachung* [*Gewinnt Alice* (*3*::*int*), *Verliert Bob 3*])
((λ-. [])(
Alice := [*to-abmachung* [*Gewinnt Alice 3*], *to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*]],
Bob := [*to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*]])
)
= (λ-. [])(
Alice := [*to-abmachung* [*Gewinnt Alice 3*]],
Bob := [])›

Alternative Definition:

lemma *konsens-entfernen-simp*:
‹*konsens-entfernen a kons*
= (λ*p*. *if p* ∈ *set* (*abmachungs-betroffene a*) *then remove1 a* (*kons p*) *else* (*kons p*))›

definition *reverse-engineer-abmachung*
:: ‹(*'person*::*enum* ⇒ *'etwas*::*linordered-ab-group-add*) *handlung* ⇒ (*'person*, *'etwas*) *abmachung*›
where
‹*reverse-engineer-abmachung h* ≡
fold (λ*p acc*. *acc*(*p* := (*nachher h p*) − (*vorher h p*))) *Enum.enum* (λ-. *0*)›

lemma *reverse-engineer-abmachung-delta-num-fun*:
‹*reverse-engineer-abmachung h* = *to-abmachung* (*delta-num-fun h*)›
lemma *reverse-engineer-abmachung*:
reverse-engineer-abmachung (*Handlung welt welt'*) = *a* ⟷ *abmachung-ausfuehren a welt* = *welt'*

13 Beispiel: Zahlenwelt2

record *zahlenwelt* =
besitz :: ‹*person* ⇒ *int*›
konsens :: ‹(*person*, *int*) *globaler-konsens*›
staatsbesitz :: ‹*int*› — Der Staat ist keine natürliche Person und damit besonders.
umwelt :: ‹*int*›

```
definition initialwelt :: ‹zahlenwelt›
  where
‹initialwelt ≡ ⦇
  besitz = 🌍[Alice := 5, Bob := 10, Carol := −3],
  konsens = (λ-. [])(
    Alice := [to-abmachung [Gewinnt Alice 3], to-abmachung [Gewinnt Alice 3, Verliert Bob 3]],
    Bob := [to-abmachung [Gewinnt Alice 3, Verliert Bob 3]]),
  staatsbesitz = 9000,
  umwelt = 600
 ⦈›
```

Mein persönlicher Besitz:

```
fun meins :: ‹person ⇒ zahlenwelt ⇒ int› where
  ‹meins p welt = (besitz welt) p›

lemma ‹meins Carol initialwelt = −3›
```

Wenn *reverse-engineer-abmachung* hier nicht genau die gleiche Abmachung berechnet wie später eingelöst, dann wird das ganze exploitable. Da eine (*'person*, *'etwas*) *abmachung* aber eine eindeutige Darstellung sein sollte, müsst das so funktionieren.

```
definition hat-konsens :: ‹zahlenwelt handlung ⇒ bool›
where
  ‹hat-konsens h ≡
    let abmachung = reverse-engineer-abmachung (map-handlung besitz h)
    in enthaelt-konsens abmachung (vorher h)›
```

Eine Handlung die keine Änderung bewirkt hat keine Betroffenen und damit immer Konsens.

```
lemma ‹hat-konsens (handeln p welt (Handlungsabsicht (λp w. Some w)))›

lemma ‹hat-konsens (handeln Alice initialwelt
        (Handlungsabsicht (λp w. Some (w⦇ besitz := ⟦⟦(besitz w)(Alice += 3)⟧(Bob −= 3)⟧ ⦈))))›
lemma ‹¬ hat-konsens (handeln Alice initialwelt
          (Handlungsabsicht (λp w. Some (w⦇ besitz := ⟦⟦(besitz w)(Alice += 4)⟧(Bob −= 4)⟧ ⦈))))›
```

```
definition abmachung-ausfuehren
  :: ‹(person, int) abmachung ⇒ zahlenwelt ⇒ zahlenwelt›
where
  ‹abmachung-ausfuehren abmachung welt ≡
    welt⦇ besitz := Aenderung.abmachung-ausfuehren abmachung (besitz welt) ⦈›

lemma ‹abmachung-ausfuehren (to-abmachung [Gewinnt Alice 3]) initialwelt
  = initialwelt⦇ besitz := ⟦(besitz initialwelt)(Alice += 3)⟧⦈›
```

Um eine (*person*, *int*) *abmachung* einzulösen wird diese erst ausgeführt und danach aus dem globalen Konsens entfernt, damit die Abmachung nicht mehrfach eingelöst werden kann.

```
definition abmachung-einloesen :: ‹(person, int) abmachung ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹abmachung-einloesen delta welt ≡
  if enthaelt-konsens delta welt
  then Some ((abmachung-ausfuehren delta welt)⦇ konsens := konsens-entfernen delta (konsens welt)⦈)
  else None›
```

```
lemma‹abmachung-einloesen (to-abmachung [Gewinnt Alice 3, Verliert Bob 3]) initialwelt
 = Some
  ⦇
    besitz = 🌏[Alice := 8, Bob := 7, Carol := −3],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Gewinnt Alice 3]],
      Bob := []),
    staatsbesitz = 9000,
    umwelt = 600
  ⦈›
```

```
lemma‹abmachung-einloesen (to-abmachung [Gewinnt Alice 3]) initialwelt
 = Some
  ⦇
    besitz = 🌏[Alice := 8, Bob := 10, Carol := −3],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Gewinnt Alice 3, Verliert Bob 3]],
      Bob := [to-abmachung [Gewinnt Alice 3, Verliert Bob 3]]),
    staatsbesitz = 9000,
    umwelt = 600
  ⦈›
```

```
lemma‹abmachung-einloesen (to-abmachung [Verliert Bob 3]) initialwelt = None›
```

Die Handlungsabsicht *abmachung-einloesen* stellt keine *wohlgeformte-handlungsabsicht* dar, da in der Abmachung Personen hardcedoded sind.

```
lemma ‹¬ wohlgeformte-handlungsabsicht zahlenwps initialwelt
        (Handlungsabsicht (λp w. abmachung-einloesen (to-abmachung [Gewinnt Alice 3]) w))›
```

Wir können aber schnell eine wohlgeformte Handlungsabsicht daraus bauen, indem wir nicht die Abmachung an sich in die Handlungsabsicht hardcoden, sondern indem wir eine bestehende Abmachung in der Welt referenzieren.

```
definition existierende-abmachung-einloesen :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹existierende-abmachung-einloesen p welt ≡
  case (konsens welt) p
  of [] ⇒ None
  |  d#- ⇒ abmachung-einloesen d welt›
```

lemma ‹*wohlgeformte-handlungsabsicht zahlenwps initialwelt*
    (*Handlungsabsicht existierende-abmachung-einloesen*)›

In jeder Welt ist damit die Handlungsabsicht wohlgeformt.

lemma ‹*wohlgeformte-handlungsabsicht zahlenwps welt*
    (*Handlungsabsicht existierende-abmachung-einloesen*)›

Es ist nur möglich eine *existierende-abmachung-einloesen*, wenn alle Betroffenen auch zustimmen. Es is beispielsweise nicht möglich, dass *Alice* eine Handlung ausführt, die *Carol* betrifft, ohne deren Zustimmung.

```
lemma ‹¬ ausfuehrbar Alice
  ⦇
    besitz = 🌍[Alice := 5, Bob := 10, Carol := −3],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Verliert Carol 3]]
      ),
    staatsbesitz = 9000,
    umwelt = 600
  ⦈
  (Handlungsabsicht existierende-abmachung-einloesen)›
```

Nur wenn *Carol* zustimmt wird die Handlung möglich.

```
lemma ‹ausfuehrbar Alice
  ⦇
    besitz = 🌍[Alice := 5, Bob := 10, Carol := −3],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Verliert Carol 3]],
      Carol := [to-abmachung [Verliert Carol 3]]
      ),
    staatsbesitz = 9000,
    umwelt = 600
  ⦈
  (Handlungsabsicht existierende-abmachung-einloesen)›
```

Da *Alice* nicht betroffen is, bleibt [*Verliert Carol* (*3*::'*a*)] bei *Alice* übrig.

```
lemma ‹nachher-handeln Alice
  ⦇
    besitz = 🌍[Alice := 5, Bob := 10, Carol := −3],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Verliert Carol 3]],
      Carol := [to-abmachung [Verliert Carol 3]]
      ),
    staatsbesitz = 9000,
```

```
    umwelt = 600
  ⦈
  (Handlungsabsicht existierende-abmachung-einloesen)
= ⦇
    besitz = 🌍[Alice := 5, Bob := 10, Carol := −6],
    konsens = (λ-. [])(
      Alice := [to-abmachung [Verliert Carol 3]],
      Carol := []
      ),
    staatsbesitz = 9000,
    umwelt = 600
  ⦈›
```

Für*existierende-abmachung-einloesen* gilt immer *hat-konsens*. Das *reverse-engineer-abmachung* macht also das Richtige.

```
lemma hat-konsens-existierende-abmachung-einloesen:
  hat-konsens (handeln p welt (Handlungsabsicht existierende-abmachung-einloesen))
```

Ressourcen können nicht aus dem Nichts erschaffen werden.

```
fun abbauen :: ‹nat ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹abbauen i p welt = Some (welt⦇ besitz := ⟦(besitz welt)(p += int i)⟧, umwelt := (umwelt welt) − int i ⦈)›

lemma ‹wohlgeformte-handlungsabsicht zahlenwps welt (Handlungsabsicht (abbauen n))›

lemma ‹wohlgeformte-handlungsabsicht zahlenwps initialwelt (Handlungsabsicht (abbauen n))›
```

```
fun reset :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹reset ich welt = Some (welt⦇ besitz := λ -. 0⦈)›

lemma ‹wohlgeformte-handlungsabsicht zahlenwps welt (Handlungsabsicht reset)›

fun alles-kaputt-machen :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹alles-kaputt-machen ich welt = Some (welt⦇ besitz := λ -. Min ((besitz welt) ‘ UNIV) − 1 ⦈)›

lemma alles-kaputt-machen-code[code]:
  ‹alles-kaputt-machen ich welt =
   Some (welt⦇ besitz := (λ-. min-list (map (besitz welt) enum-class.enum) −1)⦈)›
```

```
fun unmoeglich :: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹unmoeglich - - = None›
```

```
fun individueller-fortschritt :: ‹person ⇒ zahlenwelt handlung ⇒ bool› where
  ‹individueller-fortschritt p (Handlung vor nach) ⟷ (meins p vor) ≤ (meins p nach)›

definition maxime-altruistischer-fortschritt :: ‹(person, zahlenwelt) maxime› where
  ‹maxime-altruistischer-fortschritt ≡
      Maxime (λich h. ∀pX. individueller-fortschritt pX h)›
```

```
value[simp] ‹erzeuge-beispiel
  zahlenwps initialwelt
  [Handlungsabsicht (abbauen 5),
   Handlungsabsicht existierende-abmachung-einloesen,
   Handlungsabsicht reset,
   Handlungsabsicht alles-kaputt-machen,
   Handlungsabsicht unmoeglich]
  maxime-altruistischer-fortschritt›
```

```
definition maxime-hatte-konsens :: ‹(person, zahlenwelt) maxime› where
  ‹maxime-hatte-konsens ≡ Maxime (λich h. hat-konsens h)›
```

```
lemma ‹∀h ∈ set (alle-moeglichen-handlungen initialwelt [Handlungsabsicht
existierende-abmachung-einloesen]).
 wohlgeformte-maxime-auf
    h zahlenwps
    maxime-hatte-konsens›
```

```
lemma ‹wohlgeformte-maxime zahlenwps maxime-hatte-konsens›

lemma ‹erzeuge-beispiel
  zahlenwps initialwelt
  [Handlungsabsicht existierende-abmachung-einloesen]
  maxime-hatte-konsens
= Some
  (|bsp-welt = initialwelt,
```

bsp-erfuellte-maxime = *Some maxime-hatte-konsens*,
bsp-erlaubte-handlungen = [*Handlungsabsicht existierende-abmachung-einloesen*],
bsp-verbotene-handlungen = []⦈›

lemma ‹*erzeuge-beispiel*
zahlenwps initialwelt
[*Handlungsabsicht* (*abbauen 5*),
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich]
maxime-altruistischer-fortschritt
= *Some*
⦇*bsp-welt* = *initialwelt*,
bsp-erfuellte-maxime = *Some maxime-altruistischer-fortschritt*,
bsp-erlaubte-handlungen = [*Handlungsabsicht* (*abbauen 5*), *Handlungsabsicht unmoeglich*],
bsp-verbotene-handlungen = [*Handlungsabsicht reset*, *Handlungsabsicht alles-kaputt-machen*]⦈›

lemma ‹*erzeuge-beispiel*
zahlenwps initialwelt
[*Handlungsabsicht* (*abbauen 5*),
Handlungsabsicht existierende-abmachung-einloesen,
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich]
(*MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens*)
= *Some*
⦇*bsp-welt* = *initialwelt*,
bsp-erfuellte-maxime = *Some* (*MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens*),
bsp-erlaubte-handlungen = [*Handlungsabsicht* (*abbauen 5*), *Handlungsabsicht existierende-abmachung-einloesen*, *Handlungsabsicht unmoeglich*],
bsp-verbotene-handlungen = [*Handlungsabsicht reset*, *Handlungsabsicht alles-kaputt-machen*]⦈›

lemma *maxime-und-handlungsabsicht-generalisieren zahlenwps welt*
maxime-hatte-konsens (*Handlungsabsicht existierende-abmachung-einloesen*) *p*

lemma *mhg-katimp-maxime-hatte-konsens*:
‹∀ *p*. *maxime-und-handlungsabsicht-generalisieren zahlenwps welt maxime-hatte-konsens ha p* ⟹
wohlgeformte-handlungsabsicht zahlenwps welt ha ⟹
kategorischer-imperativ-auf ha welt maxime-hatte-konsens›

lemma *wpsm-kommutiert-altruistischer-fortschritt*:
‹*wpsm-kommutiert maxime-altruistischer-fortschritt zahlenwps welt*›

lemma *mhg-katimp-maxime-altruistischer-fortschritt*:
‹$\forall$ *p. maxime-und-handlungsabsicht-generalisieren zahlenwps welt maxime-altruistischer-fortschritt ha p* $\Longrightarrow$
wohlgeformte-handlungsabsicht zahlenwps welt ha $\Longrightarrow$
kategorischer-imperativ-auf ha welt maxime-altruistischer-fortschritt›

theorem
‹*ex-erfuellbare-instanz maxime-altruistischer-fortschritt welt ha* $\wedge$
($\forall$ *p. maxime-und-handlungsabsicht-generalisieren*
zahlenwps welt maxime-altruistischer-fortschritt ha p)
$\vee$
ex-erfuellbare-instanz maxime-hatte-konsens welt ha $\wedge$
($\forall$ *p. maxime-und-handlungsabsicht-generalisieren*
zahlenwps welt maxime-hatte-konsens ha p) $\Longrightarrow$
wohlgeformte-handlungsabsicht zahlenwps welt ha $\Longrightarrow$
kategorischer-imperativ-auf ha welt
(*MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens*)›

14 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* $\Rightarrow$ *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* $\Rightarrow$ *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. 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* ≤ *einkommen-a* ⟹ (λ*x*. *real-of-percentage* (*steuer-defs.steuersatz einkommen-b x*)) ≤ (λ*x*. *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*›

15 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 *steuerwps* :: ‹*person* ⇒ *person* ⇒ *steuerwelt* ⇒ *steuerwelt*› **where**
  ‹*steuerwps p1 p2 (Steuerwelt besitz) = Steuerwelt (swap p1 p2 besitz)*›

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}›
```

```
lemma mehrverdiener-betrachtet-nur-ausgangszustand:
  ‹mehrverdiener p (handeln p' welt h) = mehrverdiener p (Handlung welt welt)›
```

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)
         )›
```

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

```
lemma ‹wpsm-kommutiert (Maxime
      (λich handlung.
           (∀p∈mehrverdiener ich handlung.
                steuerlast ich handlung ≤ steuerlast p handlung))) steuerwps welt›
```

```
lemma wfh-steuerberechnung-jeder-zahlt-int:
  ‹ha = Handlungsabsicht (λich w. Some (Steuerwelt ((λe. e − steuerberechnung e) ∘ (get-einkommen w))))
    ⟹ wohlgeformte-handlungsabsicht steuerwps welt ha›
```

Wenn die Steuerfunktion monoton ist, dann kann ich auch einen sehr eingeschraenkten kat imp zeigen.

```
lemma ‹
  (⋀e1 e2. e1 ≤ e2 ⟹ steuerberechnung e1 ≤ steuerberechnung e2) ⟹
  ha = Handlungsabsicht (λich w. Some (Steuerwelt ((λe. e − steuerberechnung e) ∘ (get-einkommen w))))
⟹
  kategorischer-imperativ-auf ha welt
    (Maxime
```

(λich handlung.
(∀ p∈mehrverdiener ich handlung.
steuerlast ich handlung ≤ steuerlast p handlung)))›

15.1 Setup für Beispiele

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

15.2 Beispiel: Keiner Zahlt Steuern

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

lemma ‹*moralisch initialwelt maxime-steuern* (*Handlungsabsicht* (λ*ich welt*. *Some welt*))›

15.3 Beispiel: Ich zahle 1 Steuer

Das funktioniert nicht:

definition ‹*ich-zahle-1-steuer ich welt* ≡
Some (*Steuerwelt* ⟦(*get-einkommen welt*)(*ich* −= *1*)⟧)›
lemma ‹¬ *moralisch initialwelt maxime-steuern* (*Handlungsabsicht ich-zahle-1-steuer*)›

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

15.4 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* ≡
Some (*Steuerwelt* ((λ*e*. *e* − *1*) ∘ (*get-einkommen welt*)))›
lemma ‹*moralisch initialwelt maxime-steuern* (*Handlungsabsicht jeder-zahle-1-steuer*)›

15.5 Beispiel: Vereinfachtes Deutsches Steuersystem

Jetzt kommt die Steuern.thy ins Spiel.

definition *jeder-zahlt* :: ‹(*nat* ⇒ *nat*) ⇒ '*a* ⇒ *steuerwelt* ⇒ *steuerwelt*› **where**
‹*jeder-zahlt steuerberechnung ich welt* ≡
Steuerwelt ((λ*e*. *e* − *steuerberechnung e*) ∘ *nat* ∘ (*get-einkommen welt*))›

definition ‹*jeder-zahlt-einkommenssteuer p w* ≡ *Some* (*jeder-zahlt einkommenssteuer p w*)›

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

lemma ‹*moralisch initialwelt maxime-steuern* (*Handlungsabsicht jeder-zahlt-einkommenssteuer*)›

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

lemma ‹*moralisch*

(*Steuerwelt* 🌍[*Alice*:=*10000*, *Bob*:=*14000*, *Eve*:= *20000*])
maxime-steuern
(*Handlungsabsicht jeder-zahlt-einkommenssteuer*)›
lemma ‹*delta-steuerwelt*
(*handeln*
Alice (*Steuerwelt* 🌍[*Alice*:=*10000*, *Bob*:=*14000*, *Eve*:= *20000*])
(*Handlungsabsicht jeder-zahlt-einkommenssteuer*))
= [*Verliert Bob 511*, *Verliert Eve 1857*]›

16 Vereinfachtes Deutsches Steuersystem vs. die Steuermaxime

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

lemma *steuersystem-imp-maxime*:
‹*steuersystem steuersystem-impl* $\Longrightarrow$
($\forall$ *welt*. *moralisch welt*
maxime-steuern
(*Handlungsabsicht* (λp *w*. *Some* (*jeder-zahlt steuersystem-impl p w*)))))›

Mit genug zusätzlichen Annahmen gilt auch die Rückrichtung:

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*. *moralisch welt maxime-steuern* (*Handlungsabsicht* (λp *w*. *Some* (*jeder-zahlt steuersystem-impl p w*)))
$\Longrightarrow$ *steuersystem steuersystem-impl*›

Dass die eine Richtung gilt (Maxime impliziert *steuersystem*), die andere Richtung (*steuersystem* impliziert Maxime) jedoch nicht ohne weiter Annahmen, stimmt auch mit Russels Beobachtung überein: "Kants Maxime [das allgemeine Konzept, nicht meine Implementierung] scheint tatsächlich ein notwendiges, jedoch nicht *ausreichendes* Kriterium der Tugens zu geben" [1]. Insbesondere Russels Folgesatz freut mich, da er mir bestätigt, dass unsere extensionale Betrachtung von Handlungen vielversprechend ist: "Um ein ausreichendes Kriterium zu gewinnen, müßten wir Kants rein formalen Standpunkt aufgeben und die Wirkung der Handlungen in Betracht ziehen" [1].

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*. *moralisch welt maxime-steuern* (*Handlungsabsicht* (λp *w*. *Some* (*jeder-zahlt steuersystem-impl p w*))))
$\longleftrightarrow$ *steuersystem steuersystem-impl*›

References

[1] B. Russell. *Philosophie des Abendlandes — Ihr Zusammenhang mit der politischen und sozialen Entwicklung*. 2012. Aus dem Englischen von Elisabeth Fischer-Wernecke und Ruth Gillischewski, durchgesehen von Rudolf Kaspar.