Einführung in Mathematische Modellierung mit Isabelle/HOL am Beispiel Philosophie:
Extensionale Interpretation des Kategorischen Imperativs

Cornelius Diekmann

April 7, 2023

Abstract

Language: German.

In diesem Artikel modellieren wir eine persönlich angehauchte Interpretation von Kants kategorischem Imperativ. Primär ist dieses Dokument eine Einführung in die Modellierung mit Isabelle/HOL am Beispiel Philosophie. Das bevorzugte Leseformat ist die Proof Document Outline, welche alle wichtigen Definitionen, Beispiele und Ergbnisse (Lemmata und Theoreme) beinhaltet, jedoch die eigentlichen Beweise auslässt. Dieses Dokument ist im Theorembeweiser Isabelle/HOL geschrieben, d.h. dass beim Bau des Dokuments alle Beweise vom Computer überprüft werden. Wir können uns daher sicher sein, dass die Beweise stimmen und sie guten Gewissens im finalen Dokument auslassen. Dies erlaubt es uns, uns ganz auf die Feinheiten der Definitionen und Bedeutung der Behauptungen zu fokussieren.

Bei der "extensionalen Interpretation des Kategorischen Imperativs" handelt es sich um meine persönliche Auslegung von Kants kategorischem Imperativ, welche teilweise Kant widerspricht. Der Hautpunterschied ist die Extensionalität, wie später im Dokument erläutert wird. Dies erlaubt es uns, ein *shallow embedding* morlaischer Definitionen in unsere mathematische Logik zu konstruieren.

Meine primäre Inspiration ist die Sekundärliteratur von Bertrand Russell. Russel war sowohl klassischer Mathematiker, einer der Urherber der Principia Mathematica, als auch Philosoph. Ich behaupte, hätte Russel damals Zugang zu Isabelle/HOL gehabt, wären wir heute weiter und Higher-Order Logik (HOL) wäre Russels Logik der Wahl geworden.

Tiefgreifende neue philosophische Einsichten werden wir in diesem Artikel nicht entwicken. In Beispielen werden werden wir einige bereits bekannte Ergebnisse sehen: Z.B. Steheln ist schlecht, etwas Wertvolles erschaffen ist gut, Eigentumsübergang ist nur okay wenn Konsens herrscht, bei der Bemessung von Steuern sollte jeder gleich behandelt werden. Diese Einsichten sind weder neu noch überraschend. Der interessante Punkt ist jedoch, dass wir diese Einsichten aus der sehr generischen, allgemeinen und selbstreferentielle Formalisierung unseres kategorischen Imperativs herleiten werden! Während unser Artikel mit diesen Beispielen abschließen wird, steht dem allgemeinen Konzept und der Möglichkeit komplexere und umfassendere Weltenmodelle zu entwickeln um kompliziertere Fragestellungen zu diskutieren nichts im Weg.

Contents

1 Disclaimer

Ich habe

- mir philosophische Grundlagen nur im Selbststudium beigebracht. Meine Primärquellen sind
  - Die deutsche Übersetzung von Bertrand Russells 1946 erstveröffentlichtem "History of Western Philosophy" [1]. Von diesem Buch existiert auch eine Onlinefassung https://archive.org/details/PHILOSOPHIEDESABENDLANDESVONBERTRANDRUSSEL.
  - Eva Böhringers YouTube Kanal "Ethik-Abi by BOE" https://www.youtube.com/@EthikAbibyBOE.
  - Wikipedia im Allgemeinen. Innerhalb des Dokuments versuche ich Definitionen aus Wikipedia zu verwenden, da diese einfach und ohne Paywall zugänglich sind.

  - Weitere Bücher oder Internetquellen ohne herausragende Bedeutung für mich. Zu Beispiel stand mein "Kant für die Hand" Würfel ohne große Einsicht sehr lange herum, bis er schließlich dem Kind zum Opfer fiel.

- wenig Ahnung von deutschem Steuerrecht. Das Steuer-Beispiel im hinteren Abschnitt dieses Dokuments ist zwar an das deutsche Einkommensteuerrecht angelehnt (Quelle: Wikipedia), dennoch ist dieses Beispiel nur der Idee nach richtig. Faktisch ist es falsch und ich empfehle niemanden seine Steuererklärung basierend auf dieser Theorie zu machen.
- keine Ahnung von Jura und Strafrecht. Die thys enthalten noch ein fehlgeschlagenes Experiment welches versucht aus dem kategorischen Imperativ ein Gesetz (im rechtlichen Sinne) abzuleiten. Dieses Experiment ist allerdings fehlgeschlagen und ist auch nicht im kompilierten pdf Dokument enthalten.

Dieses Dokument ist ein instabiler Development Snapshot, entwickelt auf https://github.com/diekmann/kant. 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.

Wann immer möglich sind Quellen als klickbare URL direkt ins Dokument integriert.
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 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.

Wir werden die meisten Hilfsfakten als **lemma** kennzeichnen. Wichtige Fakten werden wir **theorem** nennen. Zusätzlich führen wir noch das **beispiel** Kommando ein, um Lemmata von Beispielen zu unterscheiden.

Folgende drei Aussagen sind alle equivalent und maschinengeprüft korrekt:

lemma ‹$3 = 2+1$›
theorem ‹$3 = 2+1$›
beispiel ‹$3 = 2+1$›

Freitext, so wie der aktuelle Satz, sind nicht (oder nur minimal) maschinell geprüft.

2.2 Typen

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

Einige vordefinierte Typen in Isabelle/HOL:

- Natürliche Zahlen. Typ nat. Beispielsweise $0::nat$, $1::nat$, $2::nat$, $3::nat$. Auf natürlichen Zahlen ist die Nachfolgerfunktion Suc definiert. Beispielsweise ist $Suc\ 2 = 3$.
- Ganze Zahlen. Typ int. Beispielsweise $0::int$, $1::int$, $-\ 1::int$, $-\ 42::int$.
- Listen. Typ $'\alpha\ list$. Beispielsweise $[]::nat\ list$, $[]::int\ list$, $[0,\ 1,\ 2,\ 3]::nat\ list$, $[0,\ 1,\ -\ 1,\ -\ 42]::int\ list$.
- Strings. Typ $string$. Beispielsweise $''Hello,\ World''::char\ list$.

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 Noch mehr Typen

Eigene Typen können unter Anderem mit dem Keyword **datatype** eingeführt werden. Im folgenden Beispiel führen wir einen **datatype** für Farben ein.

datatype *beispiel-farbe = Rot | Gruen | Blau*

Eine variable *x::beispiel-farbe* kann entweder den Wert *Rot*, *Gruen*, oder *Blau* haben. Dies lässt sich auch beweisen:

beispiel‹*x = Rot ∨ x = Gruen ∨ x = Blau*›

Wir können auch einen Schritt weitergehen und eine Liste von *beispiel-farbe* selbst implementieren.

datatype *beispiel-farbe-liste = FLLeer | FLKopf ‹beispiel-farbe› ‹beispiel-farbe-liste›*

Eine *beispiel-farbe-liste* ist hier rekursiv definiert:

- Entweder ist die Liste *FLLeer* und enthält keine Elemente.
- Oder es gibt bereits eine *beispiel-farbe-liste* und über den *FLKopf* Konstruktor hängen wir eine weitere *beispiel-farbe* an. Die Abkürzung *FLKopf* steht hier für Farben-Listen-Kopf.

Beispielsweise können wir immer länger werdende *beispiel-farbe-liste*n welche nur *Rot*e Elemente enthalten wie folgt konstruieren:

- *FLLeer* enthält 0 Elemente.
- *FLKopf Rot FLLeer* enthält ein *Rot* Element.
- *FLKopf Rot (FLKopf Rot FLLeer)* enthält zwei *Rot* Elemente.
- *FLKopf Rot (FLKopf Rot (FLKopf Rot FLLeer))* enthält drei *Rot* Elemente.

Das Konzept Liste kann weiter verallgemeinert werden. Wir können eine generische Liste bauen, welche nicht nur *beispiel-farbe*n aufnehmen kann, sondern eine polymorphe Liste, welche beliebige Typen speichern kann.

datatype *'α beispiel-liste = Leer | Kopf ‹'α› ‹'α beispiel-liste›*

Der Typ *'α* steht hierbei für einen Platzhalter für beliebige Typen. Beispielsweise können wir mit der generischen *'α beispiel-liste* wieder unsere *beispiel-farbe-liste* simulieren:

- *Leer*::*beispiel-farbe beispiel-liste* enthält 0 Elemente.
- *Kopf Rot Leer*::*beispiel-farbe beispiel-liste* enthält ein *Rot* Element.
- *Kopf Rot* (*Kopf Rot Leer*)::*beispiel-farbe beispiel-liste* enthält zwei *Rot* Elemente.
- *Kopf Rot* (*Kopf Rot* (*Kopf Rot Leer*))::*beispiel-farbe beispiel-liste* enthält drei *Rot* Elemente.

Die Liste kann jedoch auch andere Typen von Elementen speichern.

- *Kopf 2* (*Kopf 1* (*Kopf 0 Leer*))::*nat beispiel-liste*
- *Kopf ''Erstes Element''* (*Kopf ''Letzes Element'' Leer*)::*char list beispiel-liste*

Die Länge einer *'α beispiel-liste* lässt sich über folgende rekursive Funktion wie folgt definieren:

fun *beispiel-liste-laenge* :: ‹*'α beispiel-liste* ⇒ *nat*› **where**
‹*beispiel-liste-laenge Leer* = *0*›
| ‹*beispiel-liste-laenge* (*Kopf - ls*) = *Suc* (*beispiel-liste-laenge ls*)›

Funktionen werden oft über Pattern-Matching implementiert, d.h., dass der gegebene Datentyp zerlegt wird und eine Fallunterscheidung getroffen wird.

- Für den Basisfall *Leer* wird *0* zurückgegeben.
- Für den rekursiven Fall *Kopf* in dem wir ein Kopfelement haben welches wir ignorieren und einer Folgeliste *ls* rufen wir *beispiel-liste-laenge* rekursiv mit der Folgeliste auf und geben den Nachfolger der so berechneten Zahl zurück.

beispiel ‹*beispiel-liste-laenge Leer* = *0*›
beispiel ‹*beispiel-liste-laenge* (*Kopf Rot* (*Kopf Rot* (*Kopf Rot Leer*))) = *3*›

Zusätzlich können den einzelnen Feldern in Datentypen spezielle Namen gegeben werden. Beispielsweise:

datatype *'α beispiel-liste-mit-namen* =
LeerMN | *KopfMN* (*kopfelement*: ‹*'α*›) (*schwanzliste*: ‹*'α beispiel-liste-mit-namen*›)

Der Fall *LeerMN* bleibt unverändert. Um Verwechslung zu vermeiden haben wir den einzelnen Fällen das Suffix MN (Mit-Namen) gegeben, da die Konstruktoren *Leer* und *Kopf* bereits durch das vorherige Beispiel definiert sind. Im *KopfMN*-Fall haben nun die einzelnen Felder Namen.

beispiel‹*kopfelement* (*KopfMN Rot LeerMN*) = *Rot*›
beispiel‹*schwanzliste* (*KopfMN Rot LeerMN*) = *LeerMN*›

Die von Isabelle mitgelieferte Standardimplementierung einer Liste sieht unserem Beispiel recht ähnlich, allerdings liefert Isabelle noch zusätzlichen Syntactic Sugar um Listen komfortabler darzustellen. Die Implementierung einer Liste in der Standardbibliothek ist: **datatype** *'a list* = [] | (#) *'a* (*'a list*)

2.5 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.

beispiel ‹$beispielfunktion\ 32 = 42$›

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* :: ‹$nat \Rightarrow nat \Rightarrow nat$› **where**
‹$addieren\ a\ b = a + b$›

beispiel ‹$addieren\ 32\ 10 = 42$›

Currying bedeutet auch, wenn wir *addieren* nur mit einem Argument aufrufen (welches eine natürliche Zahl *nat* sein muss), dass wir eine Funktion zurückbekommen, die noch das zweite Argument erwartet, bevor sie das Ergebnis zurückgeben kann.
Beispiel: $addieren\ 10::nat \Rightarrow nat$
Zufälligerweise ist *addieren 10* equivalent zu *beispielfunktion*:

beispiel ‹$addieren\ 10 = beispielfunktion$›

Zusätzlich lassen sich Funktionen im Lambda Calculus darstellen. Beispiel:

beispiel ‹$(\lambda n::nat.\ n{+}10)\ 3 = 13$›

beispiel ‹$beispielfunktion = (\lambda n.\ n{+}10)$›

Im vorhergehenden Beispiel wurden zwei Funktionen (die *beispielfunktion* die wir oben definiert haben und ein lambda-Ausdruck) als equivalent bewiesen. Dafür wurde die Extensionalität verwendet.

2.6 Mengen

Mengen funktionieren wie normale mathematische Mengen.
Beispiel. Die Menge der geraden Zahlen:

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

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

Bei vorherigen Beispiel können wir das Prinzip der (mathematischen) *Extensionalität* sehen: Intensional sind die beiden Mengen $\{0, 2, 4, 6, 8, 10\}$ und $\{n.\ n\ mod\ 2 = 0 \land n \leq 10\}$ 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.

2.7 First-Order Logic

First-Order Logic, oder auch Prädikatenlogik erster Stufe, besteht aus folgenden Symbolen.

- Konjunktion. Der Term $A \wedge B$ besagt, dass sowohl A als auch B wahr sind. Dies entspricht dem logischen "Und".
- Disjunktion. Der Term $A \vee B$ besagt, dass A oder B (oder beide) wahr sind. Dies entspricht dem logischen "Oder".
- Negation. Der Term $\neg A$ besagt, dass A nicht wahr ist. Dies entspricht dem logischen "Nicht".
- Implikation. Der Term $A \longrightarrow B$ besagt, dass wenn A gilt dann gilt auch B. Dies entspricht dem logischen "Wenn-Dann". Unsere Mathematik ist hardcore klassisch und es gilt $A \longrightarrow B = (\neg A \vee B)$.
- All-Quantifier. Der Term $\forall x.\ P\ x$ besagt, dass $P\ x$ wahr ist für alle x. Dies entspricht dem logischen "Für-Alle".
- Existenz-Quantifier. Der Term $\exists x.\ P\ x$ besagt, dass es ein x gibt, für das $P\ x$ gilt. Dies entspricht dem logischen "Es-Existiert".
- Des Weiteren gibt es noch die Abkürzung $A \longleftrightarrow B$, welche bedeutet, dass A genau dann gilt wenn B gilt. Dies entspricht dem logischen "Genau-Dann-Wenn". Genau genommen ist $A \longleftrightarrow B$ eine Abkürzung für $A = B$. Oft ist $A \longleftrightarrow B$ praktischer zu schreiben, da es schwächer bindet. Genau genommen gilt ‹$(A \longleftrightarrow B) = ((A) = (B))$›. Die schwache Bindung von $A \longleftrightarrow B$ wird dann relevant, wenn A und B komplizierte Ausdrücke sind, da wir uns im Gegensatz zu $(A) = (B)$ Klammern sparen können.

2.8 Higher-Order Logic (HOL)

First-Order Logic ist in Higher-Order Logic eingebettet. Higher-Order Logic (HOL) ist die native Sprache von Isabelle/HOL. Im Vergleich zur First-Order Logic besteht HOL nur aus zwei essentiellen Symbolen:

- All-Quantifier. Der Term $\bigwedge x.\ P\ x$ besagt $\forall x.\ P\ x$. Eigentlich ist zwischen dem First-Order und dem Higher-Order All-Quantifier kein wesentlicher Unterschied. Genau genommen lässt sich der First-Order All-Quantifier via HOL einführen wie folgt: $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$.

  Da mathematisch der Ausdruck $P\ x$ besagt, dass P für beliebiges x—sprich: alle x— gilt, ist folgendes equivalent:

  - $\bigwedge x.\ P\ x$

- $\forall x.\ P\ x$
- $P\ x$

- Implikation. Der Term $A \Longrightarrow B$ besagt $A \longrightarrow B$. Eigentlich ist zwischen der First-Order und der Higher-Order Implikation kein wesentlicher Unterschied.

  Die Implikation assoziiert nach rechts. Dies bedeutet ‹$A \longrightarrow B \longrightarrow C \longleftrightarrow (A \longrightarrow (B \longrightarrow C))$›.

  Logisch gilt damit auch folgendes: ‹$A \longrightarrow B \longrightarrow C \longleftrightarrow A \wedge B \longrightarrow C$›.

  In Isabelle/HOL werden wir viele Lemmata der Form $A \Longrightarrow B \Longrightarrow C$ sehen, welche zu lesen sind als: Aus A und B folgt C. Dies ist gleichbedeutend mit $A \wedge B \Longrightarrow C$. Da die Higher-Order Implikation einer der Kernbausteine von HOL sind ist die Formulierung welche nur die Implikation verwendet sehr praktisch für Isabelle.

2.9 Über Isabelle/HOL

Isabelle/HOL ist ein interaktiver Beweisassistent und kann von https://isabelle.in.tum.de/ bezogen werden.

Isabelle stammt aus der Tradition der LCF (Logic for Computable Functions) Theorembeweiser. Die Standardlogik ist Higher-Order Logic (HOL), welche auf der klassischen Mathematik basiert.

Isabelle basiert auf einem mathematischen Microkernel. Zum aktuellen Zeitpunkt mit Isabelle2022 befindet sich das Herzstück dieses mathematischen Microkernels in der Datei `~~/src/Pure/thm.ML`, welche aus nur ca 2500 Zeilen ML Code besteht. Dies ist relativ wenig Code welchem wir vertrauen müssen. Die Korrektheit aller Ergebnisse und Beweise dieser Theory hängen nur von der Korrektheit dieses Microkernels ab.

Isabelle/HOL ist ein interaktiver Beweisassistent, was bedeutet, Isabelle hilft einem Benutzer dabei Beweise zu schreiben. Alle Beweise müssen von Isabelles Microkernel akzeptiert werden, welcher die Korrektheit garantiert. Im Gegensatz zu interaktiven Beweisassistenten gibt es auch automatische Theorembeweiser, wie z.B. Z3. Z3 besteht aus mehreren hunderttausend Zeilen C Code (https://github.com/Z3Prover/z3) und ist damit im Vergleich zu Isabelles Microkernel riesig. Dies bedeutet auch, dass einer riesige Menge an Code vertraut werden muss, um einem Beweis von Z3 zu vertrauen. Glücklicherweise arbeiten Isabelle und z.B. Z3 sehr gut zusammen: Z3 kann losgeschickt werden um einen Beweis zu suchen. Sobald Z3 meldet, dass ein Beweis gefunden wurde, akzeptiert Isabelle diesen jedoch nicht blind, sondern Isabelle akzeptiert den gefundenen Beweis nur, wenn der Beweis sich gegen Isabelles Microkernel wiedergeben lässt. Somit kombiniert Isabelle das Beste aus vielen Welten: Die starke Korrektheitsgarantie eines mathematischen Microkernels der LCF Reihe mit der Automatisierung der neuesten Generationen von automatischen Theorembeweisern.

Der Unterschied zwischen z.B. Z3 und Isabelle/HOL zeigt sich auch in einigen Beispielen: Während der Z3 Bugtracker sehr viele Soundness Issues zeigt, d.h. Fälle in denen Z3 etwas falsches beweisen hat, hat es meines Wissens nach im Isabelle/HOL Kernel in über 30 Jahren keinen logischen Fehler gegeben, welcher normale Benutzer betroffen hat. Natürlich gab es auch in Isabelle/HOL Bugs und in künstlich geschaffenen Grenzfällen konnte Isabelle überzeugt werden einen falschen Fakt zu akzeptieren, jedoch handelt es sich hier um wenige Einzelfälle welche speziell konstruiert wurden um Isabelle anzugreifen – kein einziger Bug hat unter normalen Umständen zu logischer Inkonsistenz geführt. Umgekehrt ist Z3 schnell und automatisch. Während Isabelle vom Benutzer verlangt, dass Beweise manuell gefunden und formalisiert werden müssen kann Z3 teils extrem komplizierte Probleme schnell und automatisch lösen.

Isabelle integriert nicht nur mit automatischen Beweisern wie Z3, Isabelle integriert auch mit automatischen Gegenbeispielsfindern, wie z.B. **quickcheck** oder **nitpick**. Dies bedeutet, sobald der Benutzer einen vermeintlichen Fakt in Isabelle eintippt, schickt Isabelle Prozesse los um ein Gegenbeispiel zu finden. Diese so gefundenen Gegenbeispiele sind oft unintuitiv. Allerdings sind es echte Gegenbeispiele und der Computer erweist sich als grausamer unnachgiebiger Diskussionspartner. Dies führt dazu, dass wir nicht mit halbgaren oberflächlichen Argumenten durchkommen. Oft eröffnen diese automatischen Gegenbeispiele auch eine neue Sicht auf die Dinge und helfen, die eigenen impliziten Annahmen zu erkennen und zu hinterfragen. Der Computer erweist sich hier als perfekter logischer geduldiger Diskussionspartner, denn "der wahre Philosoph ist gewillt, alle vorgefaßten Meinungen einer Prüfung zu unterziehen" [1]. Und da alle Fakten welche wir ultimativ als wahr behaupten wollen durch den mathematischen Microkernel müssen, sind logische Flüchtigkeitsfehler in unserer Argumentation ausgeschlossen. Allerdings können wir immer noch falsche Annahmen aufstellen, auf welche wir unsere Ergebnisse stützen. Jedoch müssen wir diese Annahmen explizit treffen und aufschreiben, denn sonst ließe sich nichts beweisen.

2.10 Ist das KI?

Nein!

Dieser Abschnitt ist etwas opinionated und handwavy. Wenn aktuell von Künstlicher Intelligenz (KI) gesprochen wird, wird damit oft Machine Learning gemeint. Ich kürze Machine Learning absichtlich nicht mit ML ab, da die Verwechslung mit der Programmiersprache Standard Meta Language (ML) https://de.wikipedia.org/wiki/Standard_ML zu groß ist. Isabelle ist in ML geschrieben und wenn im Umfeld von Isabelle von ML geredet wird, ist meistens nie Machine Learning gemeint. Genau genommen benutzt Isabelle die Poly/ML Implementierung, welche das tollste Logo aller Programmiersprachen hat.

Poly/ML Logo von polyml.org

Zurück zum Machine Learning. Eine der aktuell beliebtesten Implementierungen für maschinelles Lernen sind neuronale Netze. Dabei wird ein Algorithmus mit einer Unmenge an Trainingsdaten gefüttert, um ein sogenanntes neuronales Netzwerk zu trainieren. Ist so ein riesiges Netzwerk einmal trainiert ist es quasi unmöglich auf eine einzelne Zahl im trainierten Netzwerk zu zeigen und zu verstehen warum das Netz an genau dieser Stelle diese Zahl hat. Der Einsatz solcher Methoden um moralische Entscheidungen zu treffen ist meiner Meinung nach zurecht umstritten. Insbesondere da die Ausgaben eines neuronalen Netzes auf den Trainingsdaten beruhen. Wenn die Trainingsdaten bereits von versteckten Vorurteilen durchsetzt sind, werden diese natürlich auch in das Netz übernommen.

Unser Ansatz der Modellierung in einem Theorembeweiser ist grundverschieden von Entscheidungsfindungsansätzen basierend auf Machine Learning.

- *Axiomatische Grundlage.* Machine Learning basiert auf den Trainingsdaten. Wenn ein neuronales Netzwerk auf mehreren Milliarden Trainingsdaten basiert, ließe sich ketzerisch sagen, dass das logische System auf mehreren Milliarden Axiomen beruht. Diese Axiomatische Grundlage kann schnell inkonsistent werden und es ist unmöglich diese riesige Menge an Axiomen zu hinterfragen.

  Im Gegensatz dazu steht die Anzahl der Axiome in Isabelle/HOL. Es handelt sich größtenteils um die gewöhnlichen Axiome der klassischen Mathematik, über die sich klassische Mathematiker größtenteils seit über 100 Jahren einig sind (https://en.wikipedia.org/wiki/Axiom_of_choice sagt "The axiom of choice was formulated in 1904 by Ernst Zermelo"). Auch ist die Anzahl der Axiome minimal. Wenn wir die Isabelle Sources nach dem Kommando **axiomatization** durchsuchen, finden wir außerhalb von Beispielen nur eine Hand voll Axiomen. In diesem Artikel verwenden wir das Kommando **axiomatization** gar nicht. Hinzu kommen noch axiomatische Konstruktionen, wobei jede solche Konstruktion jedoch sehr genau auf ihre Korrektheit geprüft wurde. In diesem Artikel führen wir keine neuen axiomatischen Konstruktionen ein und verlassen uns komplett auf die Konstruktionen der Standardbibliothek, wie z.B. **datatype** oder **fun**.

- *Nachvollziehbarkeit der Ergebnisse.* Während es bei einem großen neuronalen Netz nahezu unmöglich ist zu verstehen warum ein bestimmter Wert an einer bestimmten Stelle steht und was dies zu bedeuten hat, lässt sich dieses Dokument immer wieder neu kompilieren und alle Beweise wiederholen. Dabei ließe sich genau nachvollziehen wie ein Fakt es in den Isabelle Microkernel geschafft hat. Zusätzlich stellt Isabelle diverse `trace` Kommandos bereit. Es lässt sich also nachvollziehen, warum etwas gilt.

- *Vertrauen in die Ergebnisse.* Wie bereits erwähnt basiert Isabelle auf einem mathematischen Microkernel, wodurch ein extrem hohes Vertrauen in alle Ergebnisse erzielt wird.

Allerdings darf Machine Learning uns natürlich beim Beweise- und Gegenbeispiel-Finden helfen; so angebunden wie automatisierte Beweiser wie Z3. Das bedeutet, Machine Learning hilft bei der Entwicklung, trifft jedoch selbst keine Entscheidungen und alle Ergebnisse werden von Isabelles Kernel überprüft.

2.11 Deep Embedding vs. Shallow Embedding

Im Bereich der mathematischen Modellierung wird oft zwischen Deep Embedding und Shallow Embedding unterschieden.

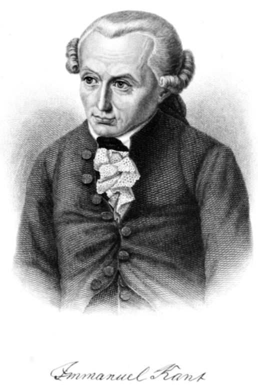
Deep Embedding bedeutet, dass unsere domänenspezifische Logik komplett in der Meta-Logik neu implementiert wird. Die Meta-Logik wird von der domänenspezifischen Logik wegabstrahiert. In Isabelle/HOL ist die Meta-Logik HOL. Die domänenspezifische Logik ist unsere Logik mit der wir moralische Schlussfolgerungen treffen wollen. Wenn wir unsere "moralische Logik" komplett in HOL deep embedden wollen, bedeutet dass, dass wir alle Konzepte unserer Logik komplett neu implementieren. So könnten wir beispielsweise definieren: **datatype** *moralisch* = *Moralisch* | *NichtMoralisch*. Ein Deep Embedding hat den Vorteil, dass wir die Semantik unserer Logik komplett selbst definieren können. Wenn sich also HOL und unsere Logik verschieden verhalten sollen, ist ein Deep Embedding unerlässlich. Der Nachteil ist allerdings, dass wir alles was wir brauchen selbst definieren müssen und die wir z.B. nicht von der enormen Menge an Lemmata der Isabelle Standardbibliothek profitieren.

Im Gegensatz dazu steht ein Shallow Embedding. Dabei wird die domänenspezifische Logik direkt in der Meta-Logik definiert. Alle Konzepte der Meta-Logik werden übernommen. Dies funktioniert nur, wenn sich die Meta-Logik so wie unsere domänenspezifische Logik verhält. Die Abstraktionsebene die unsere domänenspezifische Logik über die Meta-Logik spannt wird so dünn wie möglich gehalten und so viele Konzepte wie möglich der Meta-Logik werden Eins-zu-Eins in der domänenspezifischen Logik übernommen. Beispielsweise könnten wir uns entscheiden keinen neuen Datentyp **datatype** *moralisch* einzuführen. Wir könnten einen bestehenden HOL Datentyp weiterverwenden, beispielsweise **type-synonym** *moralisch* = *bool*. Dies funktioniert nur, wenn sich der HOL-Typ *bool* tatsächlich so verhält, wie wir es gerne für unseren domänenspezifischen "moralisch" Typen hätten. Ein riesiger Vorteil ist, dass wir bei einem Shallow Embedding die gesamte Isabelle/HOL Standardbibliothek an bestehenden Theories und Lemmata weiterverwenden können. Beispielsweise sind alle Lemmata über *bool* automatisch für uns verfügbar und bestehende Beweistaktiken funktionieren automatisch auf bestehenden HOL Typen.

Das Beispiel über **datatype** *moralisch* ist etwas übertrieben und keine neuen Datentypen einzuführen und nur z.B. **type-synonym** *moralisch* = *bool* zu verwenden ist natürlich etwas radikal. Einem Shallow Embedding ist es nicht verboten neue Datentypen einzuführen und eine eigene Semantik auf diesen Datentypen zu definieren. Dennoch gilt der allgemeine Grundsatz für ein Shallow Embedding: Die Meta-Logik wird so weit weiterverwendet wie möglich.

Wir versuchen in diesem Artikel wann immer möglich ein Shallow Embedding zu machen und so viel von HOL und Isabelles Standardbibliothek weiterzuverwenden wie möglich.

3 Kants 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

4 Handlung

In diesem Abschnitt werden wir Handlungen und Handlungsabsichten modellieren.

Wir beschreiben Handlungen als ein Änderung der Welt. Das Modell einer Handlung ist rein auf die extern beobachtbare Änderung der Welt beschränkt. Die handelnden Person ist dabei Nebensache. Wir beschreiben nur vergangene bzw. hypothetische Handlungen und deren Auswirkungen.

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

Eine Handlung ist reduziert auf die beobachtbaren Auswirkungen der Handlung. Die dahinterliegende Handlungsabsicht, bzw. Intention oder "Wollen" sind in einer *'welt handlung* nicht modelliert. Dies liegt daran, dass wir irgendwie die geistige Welt mit der physischen Welt verbinden müssen und wir daher am Anfang nur messbare Tatsachen betrachten können. Diese initiale Entscheidung, eine Handlung rein auf ihre beobachtbaren und messbaren Auswirkungen zu reduzieren, ist essentiell für diese Theorie.

Handlungen können Leute betreffen. Handlungen können aus Sicht Anderer wahrgenommen werden. Unser Modell einer Handlung enthält jedoch nur die Welt vorher und Welt nachher. So können wir handelnde Person und beobachtende Person trennen.

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

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

Folgende Definition ist eine Handlung als Funktion gewrapped. Diese abstrakte Art eine Handlung darzustellen erlaubt es nun, die Absicht oder Intention hinter einer Handlung zu modellieren.

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

Im Vergleich zu unserer *'welt handlung* sehen wir bereits am Typen, dass eine (*'person*, *'welt*) *handlungsabsicht* nicht nur eine einfache Aussage über die *'welt* trifft, sondern auch die Absicht der handelnden *'person* beinhaltet.

Die Idee ist, dass eine (*'person*, *'welt*) *handlungsabsicht* eine generische Handlungsabsicht modelliert. Beispielsweise *Handlungsabsicht* (*λich welt. brezen-kaufen welt ich*).

Eine (*'person*, *'welt*) *handlungsabsicht* gibt eine *'welt option* zurück, anstatt einer *'welt*. 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. Denn moralisch sind Stehlen und Nichtstun sehr verschieden.

Folgende Funktion modelliert die Ausführung einer Handlungsabsicht.

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

Gegeben die *handelnde-person::'person*, die *welt::'welt* in ihrem aktuellen Zustand, und eine *ha::('person, 'welt) handlungsabsicht*, so liefert *nachher-handeln handelnde-person welt ha::'welt* die potenziell veränderte Welt zurück, nachdem die Handlungsabsicht ausgeführt wurde.
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.

Dank der Hilfsdefinition *nachher-handeln* können wir nun "Handeln" allgemein definieren. Folgende Funktion überführt effektiv eine *('person, 'welt) handlungsabsicht* in eine *'welt handlung*.

```
definition handeln
  :: ‹'person ⇒ 'welt ⇒ ('person, 'welt) handlungsabsicht ⇒ 'welt 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 *'welt 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)›
```

```
beispiel ‹nachher-handeln ''Peter'' (42::nat) beispiel-handlungsabsicht = 43›
beispiel ‹handeln ''Peter'' (42::nat) beispiel-handlungsabsicht = Handlung 42 43›
beispiel ‹nachher-handeln ''Peter'' (9000::nat) beispiel-handlungsabsicht = 9000›
beispiel ‹ist-noop (handeln ''Peter'' (9000::nat) beispiel-handlungsabsicht)›
```

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, 'welt) 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* :: ‹$'person \Rightarrow 'welt \Rightarrow ('person, 'welt)$ *handlungsabsicht* $\Rightarrow$ *bool*›
where
‹*ausfuehrbar p welt* (*Handlungsabsicht h*) = (*h p welt* $\neq$ *None*)›

Nicht ausführbare Handlungen resultieren in unserem Modell im Nichtstun:

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

4.1 Interpretation: Gesinnungsethik vs. Verantwortungethik

Nur basierend auf unserem Modell einer *Handlung* und *Handlungsabsicht* können wir bereits erste Aussagen über moralische Bewertungen treffen.

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: ($'person$, $'welt$) *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: $'welt$ *handlung* $\Rightarrow$ *bool*.

Da *handeln* eine Handlungsabsicht ($'person$, $'welt$) *handlungsabsicht* in eine konkrete Änderung der Welt $'welt$ *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 ausgeführt wird und die Folgen betrachtet werden:

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

$(\forall\, person\ welt.\ verantwortungsethik\ (handeln\ person\ welt\ handlungsabsicht))$›

Ich habe 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.

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

In diesem Abschnitt werden wir das Konzept einer Maxime modellieren.

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*.
- *'welt handlung*: die zu betrachtende Handlung.
- *bool*: Das Ergebnis der Betrachtung. *True* = Gut; *False* = Schlecht.

Wir brauchen sowohl die *'welt 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*, *'welt*) *maxime* = *Maxime* ‹*'person* ⇒ *'welt handlung* ⇒ *bool*›

Auswertung einer Maxime:

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

Beispiel

definition *maxime-mir-ist-alles-recht* :: ‹(*'person*, *'welt*) *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 *'welt handlung*, also eine konkrete Handlung, die nur durch ihre Folgen gegeben ist. Die Maxime betrachtet keine Handlungsabsicht (*'person*, *'welt*) *handlungsabsicht*.

»Zweifellos hat Immanuel Kant eine Art von Gesinnungsethik vertreten« https://de.wikipedia.org/wiki/Gesinnungsethik#Gesinnungsethik_bei_Kant. Wie wir bereits im Abschnitt 4.1 gesehen haben, sollte eine Maxime demnach eine (*'person*, *'welt*) *handlungsabsicht* und keine *'welt handlung* betrachten. Dennoch haben wir uns für unsere *extensionale* Interpretation für eine (*'person*, *'welt*) *handlungsabsicht* entschieden. Und auch wenn wir das Zitat der https://de.wikipedia.org/w/index.php?title=Gesinnungsethik&oldid=218409490#Gesinnungsethik_bei_Kant weiterlesen, sehen wir, dass unser Modell zumindest nicht komplett inkonsistent ist: »Zweifellos hat Immanuel Kant eine Art von Gesinnungsethik vertreten, die allerdings nicht im Gegensatz zu einer Verantwortungsethik, sondern allenfalls zu einer bloßen „Erfolgsethik“ steht.«

Kant unterscheidet unter Anderem "zwischen »apriorischen« und »empirischen« Urteilen" [1]. Wenn wir uns den Typ *'welt 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, 'welt) 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*
  :: ‹*'welt* ⇒ (*'person*, *'welt*) *handlungsabsicht* ⇒ (*'welt handlung*) *set*›
 where
  ‹*wenn-jeder-so-handelt welt handlungsabsicht* ≡
   (λ*handelnde-person*. *handeln handelnde-person welt handlungsabsicht*) ‘ *bevoelkerung*›

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

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

lemma *moralisch-unfold*:
 ‹*moralisch welt* (*Maxime m*) *handlungsabsicht* ⟷
    (∀ *p1*∈*bevoelkerung*. ∀ *p2*∈*bevoelkerung*. *m p1* (*handeln p2 welt handlungsabsicht*))›
lemma ‹*moralisch welt* (*Maxime m*) *handlungsabsicht* ⟷
    (∀ (*p1*,*p2*)∈*bevoelkerung*×*bevoelkerung*. *m p1* (*handeln p2 welt handlungsabsicht*))›

lemma *moralisch-simp*:

$\langle \mathit{moralisch\ welt\ m\ handlungsabsicht} \longleftrightarrow$
$\quad (\forall \mathit{p1\ p2}.\ \mathit{okay\ m\ p1}\ (\mathit{handeln\ p2\ welt\ handlungsabsicht}))\rangle$

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: $\mathit{okay\ m\ ich}\ (\mathit{handeln\ ich\ welt\ handlungsabsicht}) \Longrightarrow \forall \mathit{p2}.\ \mathit{okay\ m\ ich}\ (\mathit{handeln\ p2\ welt\ handlungsabsicht})$

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

lemma *goldene-regel*:
$\langle \mathit{moralisch\ welt\ m\ handlungsabsicht} \Longrightarrow$
$\mathit{okay\ m\ ich}\ (\mathit{handeln\ ich\ welt\ handlungsabsicht}) \Longrightarrow$
$\forall \mathit{p2}.\ \mathit{okay\ m\ ich}\ (\mathit{handeln\ p2\ welt\ handlungsabsicht})\rangle$

Für das obige lemma brauchen wir die Annahme $\mathit{m\ ich}\ (\mathit{handeln\ ich\ welt\ handlungsabsicht})$ 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
$\langle \mathit{moralisch\ welt\ m\ handlungsabsicht} \Longrightarrow$
$\forall \mathit{p2}.\ \mathit{okay\ m\ ich}\ (\mathit{handeln\ p2\ welt\ handlungsabsicht})\rangle$

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

Entweder ist etwas moralisch, oder es gibt zwei Personen, für die eine Handlungsabsicht nicht *okay* ist.

lemma *moralisch-oder-nicht-okay*:
$\mathit{moralisch\ welt\ m\ ha} \oplus (\exists \mathit{p1\ p2}.\ \neg\ \mathit{okay\ m\ p2}\ (\mathit{handeln\ p1\ welt\ ha}))$

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* $(\lambda(p,\ x).\ \mathit{m\ p}\ (\mathit{handeln\ x\ welt\ handlungsabsicht}))$ (*List.product bevoelk bevoelk*).

6.3 Maximen Debugging

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

record (*'person*, *'welt*) *dbg-verletzte-maxime* =
dbg-opfer :: ⟨*'person*⟩ — verletzt für; das Opfer
dbg-taeter :: ⟨*'person*⟩ — handelnde Person; der Täter
dbg-handlung :: ⟨*'welt 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
  :: ‹('welt ⇒ 'printable-world) ⇒ 'welt ⇒
      ('person, 'welt) maxime ⇒ ('person, 'welt) 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:

```
beispiel ‹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.

```
beispiel ‹moralisch
           [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))))›
beispiel ‹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.

beispiel ‹¬ *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))))›
beispiel ‹*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*
:: ‹*('person, 'welt) maxime ⇒ ('person, 'welt) maxime ⇒ ('person, 'welt) maxime*›
where
‹*MaximeDisj (Maxime m1) (Maxime m2) = Maxime (λp h. m1 p h ∨ m2 p h)*›

lemma *okay-MaximeDisj*: ‹*okay (MaximeDisj m1 m2) p h ⟷ okay m1 p h ∨ okay m2 p h*›

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

lemma *moralisch-MaximeDisjI*:
‹*moralisch welt m1 ha* $\vee$ *moralisch welt m2 ha* $\Longrightarrow$ *moralisch welt* (*MaximeDisj m1 m2*) *ha*›

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* $\vee$ *moralisch welt m2 ha*, dass für *alle* Personen entweder *m1* oder *m2* gelten muss.

lemma *moralisch-MaximeDisj1*:
‹*moralisch welt m1 ha* $\Longrightarrow$ *moralisch welt* (*MaximeDisj m1 m2*) *ha*›
lemma *moralisch-MaximeDisj2*:
‹*moralisch welt m2 ha* $\Longrightarrow$ *moralisch welt* (*MaximeDisj m1 m2*) *ha*›

lemma *moralisch-MaximeDisj-False*:
‹*moralisch welt* (*MaximeDisj m1* (*Maxime* (λ- -. *False*))) *ha* $\longleftrightarrow$ *moralisch welt m1 ha*›

lemma *moralisch-MaximeDisj-True*:
‹*moralisch welt* (*MaximeDisj m1* (*Maxime* (λ- -. *True*))) *ha*›

Negation.

fun *MaximeNot* :: ‹(*'person*, *'welt*) *maxime* $\Rightarrow$ (*'person*, *'welt*) *maxime*›
where
‹*MaximeNot* (*Maxime m*) = *Maxime* (λ*p h*. $\neg$ *m p h*)›

lemma *okay-MaximeNot*: ‹*okay* (*MaximeNot m*) *p h* $\longleftrightarrow$ $\neg$ *okay m p h*›

lemma *okay-DeMorgan*:
‹*okay* (*MaximeNot* (*MaximeConj m1 m2*)) *p h*
$\longleftrightarrow$ *okay* (*MaximeDisj* (*MaximeNot m1*) (*MaximeNot m2*)) *p h*›

lemma *moralisch-DeMorgan*:
‹*moralisch welt* (*MaximeNot* (*MaximeConj m1 m2*)) *ha*
$\longleftrightarrow$ *moralisch welt* (*MaximeDisj* (*MaximeNot m1*) (*MaximeNot m2*)) *ha*›

7 Schleier des Nichtwissens

In diesem Abschnitt werden wir, basierend auf der Idee von Rawls Schleier des Nitchwissens, definieren, was eine wohlgeformte Handlungsabsicht und eine wohlgeformte Maxime sind.

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: λ*ich welt. if ich = Alice then Do-A welt else Do-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: λ*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: λ*ich welt. enteignen ' {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, 'welt) wp-swap* = ‹*'person* ⇒ *'person* ⇒ *'welt* ⇒ *'welt*›

Ein jeder *('person, 'welt) wp-swap* sollte mindestens folgendes erfüllen:

definition *wps-id* :: ‹*('person, 'welt) wp-swap* ⇒ *'welt* ⇒ *bool*›
where
‹*wps-id wps welt* ≡ ∀ *p1 p2. wps p2 p1 (wps p1 p2 welt)* = *welt*›

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

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, 'welt) wp-swap ⇒ 'welt ⇒ ('person, 'welt) 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*)›

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

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

fun *wohlgeformte-handlungsabsicht-gegenbeispiel*
:: ‹('person, 'welt) wp-swap ⇒ 'welt ⇒ ('person, 'welt) 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, 'welt) wp-swap ⇒ 'welt ⇒
      ('person, 'welt) maxime ⇒ ('person, 'welt) 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.

```
beispiel
  ‹¬ 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*, *'welt*) *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, 'welt) maxime ⇒ ('person, 'welt) wp-swap ⇒ 'welt ⇒ 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* ⟹
  wpsm-kommutiert m wps welt ⟹
    okay m p2 (*handeln p1* (*wps p1 p2 welt*) *ha*)
    ⟷
    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
  ‹∀ *ha*. *wohlgeformte-handlungsabsicht wps welt ha* ∧
      (∀ *p1 p2*. *okay m p2* (*handeln p1* (*wps p1 p2 welt*) *ha*)
          ⟷
          okay m p1 (*handeln p2 welt ha*)) ⟹
    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*
  :: ‹*'welt handlung* ⇒ (*'person*, *'welt*) *wp-swap* ⇒ (*'person*, *'welt*) *maxime* ⇒ *bool*›
where
  ‹*wohlgeformte-maxime-auf h wps m* ≡
    ∀ *p1 p2*. *okay m p1 h* ⟷ *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*, *'welt*) *wp-swap* ⇒ (*'person*, *'welt*) *maxime* ⇒ *bool*›
where
  ‹*wohlgeformte-maxime wps m* ≡
    ∀ *h*. *wohlgeformte-maxime-auf h wps m*›

Beispiel:

beispiel ‹*wohlgeformte-maxime swap* (*Maxime* (λ*ich h*. (*vorher h*) *ich* ≤ (*nachher h*) *ich*))›

8 Kategorischer Imperativ

In diesem Abschnitt werden wir den kategorischen Imperativ modellieren.

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

- $\mathit{moralisch}::'\mathit{welt} \Rightarrow ('\mathit{person}, '\mathit{welt})\ \mathit{maxime} \Rightarrow ('\mathit{person}, '\mathit{welt})\ \mathit{handlungsabsicht} \Rightarrow \mathit{bool}$

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:

- $'\mathit{welt} \Rightarrow ('\mathit{person}, '\mathit{welt})\ \mathit{maxime} \Rightarrow \mathit{bool}$

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*
 :: ‹$('\mathit{person}, '\mathit{welt})\ \mathit{handlungsabsicht} \Rightarrow '\mathit{welt} \Rightarrow ('\mathit{person}, '\mathit{welt})\ \mathit{maxime} \Rightarrow \mathit{bool}$›
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, 'welt) wp-swap ⇒ 'welt ⇒ ('person, 'welt) 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::'welt ⇒ ('person, 'welt) 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* $\Longrightarrow$
kategorischer-imperativ-auf ha welt m›

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

beispiel
‹*m = Maxime* (λ- -. *False*) $\Longrightarrow$
kategorischer-imperativ-auf ha welt m $\longrightarrow$ *moralisch welt m ha*
$\Longrightarrow$ *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* $\longleftrightarrow$
($\forall$ *ha p1 p2 ich*.
wohlgeformte-handlungsabsicht wps welt ha $\wedge$ *ausfuehrbar ich welt ha* $\wedge$
okay m ich (*handeln ich welt ha*)
$\longrightarrow$ *okay m p1* (*handeln p2 welt ha*))›

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

lemma *kategorischer-imperativ-auf2*:
‹*kategorischer-imperativ-auf ha welt m* $\longleftrightarrow$
moralisch welt m ha $\vee$ $\neg$($\exists$ *p*. *ausfuehrbar p welt ha* $\wedge$ *okay m p* (*handeln p welt ha*))›

corollary
‹*kategorischer-imperativ-auf ha welt m* $\longleftrightarrow$
moralisch welt m ha $\vee$ ($\forall$ *p*. *ausfuehrbar p welt ha* $\longrightarrow$ $\neg$ *okay m p* (*handeln p welt ha*))›

Man könnte auch sagen, damit eine Handlungsabsicht den kategorischen Imperativ erfüllt, muss die Handlungsabsicht entweder für alle moralisch sein, oder die Handlungsabsicht (soweit ausführbar) ist für alle nicht okay.

lemma *katimp-eq-entweder-moralisch-oder-nicht-okay*:
assumes ‹$\exists$ *p*. *ausfuehrbar p welt ha*›
shows
‹*kategorischer-imperativ-auf ha welt m* $\longleftrightarrow$
(*moralisch welt m ha* $\oplus$ ($\forall$ *p*. *ausfuehrbar p welt ha* $\longrightarrow$ $\neg$ *okay m p* (*handeln p welt ha*)))›

Also entweder ist die Absicht *für alle* okay, oder sie ist *für alle* nicht okay.

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:

beispiel ‹*kategorischer-imperativ wps welt* (*Maxime* (*λich h. False*))›

Allerdings kann mit so einer Maxime nie etwas moralisch sein.

beispiel ‹¬ *moralisch welt* (*Maxime* (*λich h. False*)) *h*›

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

beispiel ‹*kategorischer-imperativ wps welt* (*Maxime* (*λich h. True*))›

Allerdings ist mit so einer Maxime alles moralisch.

beispiel ‹*moralisch welt* (*Maxime* (*λich 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 ‹∀ *ha*. *moralisch welt maxime ha* ⟹ *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*))›

Sollte eine Handlungsabsicht nicht ausführbar, sein erfüllt sie immer den kategorischen Imperativ.

lemma *nicht-ausfuehrbar-katimp*:
‹∀ *p*. ¬ *ausfuehrbar p welt ha* ⟹ *kategorischer-imperativ-auf ha welt 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* :: ‹′*welt handlung* ⇒ *bool*›
assumes *mhg*: ‹∀ *p*. *maxime-und-handlungsabsicht-generalisieren wps welt* (*Maxime* (λ*ich*::′*person*. *P*)) *ha p*›
and *maxime-erlaubt-untaetigkeit*: ‹∀ *p*. *ist-noop* (*handeln p welt ha*) ⟶ *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*:
‹∀ *p1 p2 welt*. *wps p1 p2 welt* = *wps p2 p1 welt*›
and *wps-id*:
‹∀ *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
  :: ‹'welt ⇒ ('person::enum, 'welt) handlungsabsicht ⇒ 'welt handlung list›
where
  ‹alle-moeglichen-handlungen welt ha ≡ [handeln p welt ha. p ← (Enum.enum::'person list)]›
```

```
lemma set-alle-moeglichen-handlungen:
  ‹set (alle-moeglichen-handlungen welt ha) = {handeln p welt ha | p. True}›
```

```
record ('person, 'welt) beipiel =
  bsp-erfuellte-maxime :: ‹bool›
  bsp-erlaubte-handlungen :: ‹('person, 'welt) handlungsabsicht list›
  bsp-verbotene-handlungen :: ‹('person, 'welt) handlungsabsicht list›
  bsp-uneindeutige-handlungen :: ‹('person, 'welt) handlungsabsicht list›
```

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

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 genau dann ein Beispiel, wenn alles wohlgeformt ist.

lemma ‹($\exists$ *bsp. erzeuge-beispiel wps welt has m* = *Some bsp*) $\longleftrightarrow$
($\forall$ *ha* $\in$ *set has. wohlgeformte-handlungsabsicht wps welt ha*) $\wedge$
($\forall$ *h* $\in$ *set* [*h. ha* $\leftarrow$ *has*, *h* $\leftarrow$ *alle-moeglichen-handlungen welt ha*]. *wohlgeformte-maxime-auf h wps m*)›
beispiel
‹*erzeuge-beispiel swap* (λ*p*::*person. 0*::*int*) [*Handlungsabsicht* (λ*p w. Some w*)] (*Maxime* (λ*ich w. True*))
=
Some
⦇
bsp-erfuellte-maxime = *True*,
bsp-erlaubte-handlungen = [*Handlungsabsicht* (λ*p w. Some w*)],
bsp-verbotene-handlungen = [],
bsp-uneindeutige-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 *'welt handlung*, nur die Welt vorher und nachher, ohne Absicht) aufzuschreiben. Allerdings erzeugt das ohne die Absicht (i.e. (*'person*, *'welt*) *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*, *'welt*) *maxime* $\Rightarrow$ *'welt* $\Rightarrow$ (*'person*, *'welt*) *handlungsabsicht* $\Rightarrow$ *bool*›
where
‹*ex-erfuellbare-instanz m welt ha* $\equiv$
$\exists$ *ich. ausfuehrbar ich welt ha* $\wedge$ *okay m ich* (*handeln ich welt ha*)›

8.5.1 Konjunktion

lemma *MaximeConjI*:
‹*kategorischer-imperativ-auf ha welt m1* $\wedge$ *kategorischer-imperativ-auf ha welt m2* $\Longrightarrow$
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* $\Longrightarrow$
kategorischer-imperativ-auf ha welt (*MaximeConj m1 m2*) $\Longrightarrow$
kategorischer-imperativ-auf ha welt m1 $\wedge$ *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* $\Longrightarrow$
kategorischer-imperativ-auf ha welt (*MaximeConj m1 m2*) $\longleftrightarrow$
kategorischer-imperativ-auf ha welt m1 $\wedge$ *kategorischer-imperativ-auf ha welt m2*›

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

lemma *kategorischer-imperativ-auf-MaximeConj-True*:
‹*kategorischer-imperativ-auf ha welt* (*MaximeConj m1* (*Maxime* (λ- -. *True*)))
$\longleftrightarrow$ *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* $\wedge$ *kategorischer-imperativ-auf ha welt m1*) $\vee$
(*ex-erfuellbare-instanz m2 welt ha* $\wedge$ *kategorischer-imperativ-auf ha welt m2*) $\Longrightarrow$
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.

beispiel
‹*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*))
    ⟷ *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 *'welt glueck-messen* = ‹*'welt 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 ‹(λ*h*::*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* (−∞)) = −∞›

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

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

9.1 Goldene Regel und Utilitarismus im Einklang

In §4.1 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, 'welt*) *maxime* ⇒ (*'person, 'welt*) *handlungsabsicht* ⇒ *bool*›
where
‹*goldene-regel-als-gesinnungsethik maxime handlungsabsicht* ≡
∀ *welt. moralisch welt maxime handlungsabsicht*›

definition *utilitarismus-als-verantwortungsethik*
:: ‹*'welt glueck-messen ⇒ 'welt 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, 'welt*) *maxime* ⇒ (*'welt handlung ⇒ bool*)› **where**
‹*maximeNeutralisieren* (*Maxime m*) = (*λwelt.* ∀ *p::'person. m p welt*)›

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

definition *maxime-als-nutzenkalkuel*
:: ‹(*'person, 'welt*) *maxime* ⇒ *'welt 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*, *'welt*) *maxime* ⇒ *'welt glueck-messen*›
where
‹*maxime-als-summe-wohlergehen* (*Maxime m*) =
(λ*welt*. ∑*p*∈*bevoelkerung*. (*case m p welt*
of True ⇒ *1*
| *False* ⇒ − ∞))›
theorem
fixes *maxime* :: ‹(*'person*, *'welt*) *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*, *'welt*) *handlungsabsicht* sehr einfach in eine *'welt handlung* in unserem theoretischen Modell überführen können. Die widerspricht Kants Grundannahme, dass die Folgen einer Handlungsabsicht unvorhersehbar sind.

10 Zahlenwelt Helper

In diesem Abschnitt definieren wir Hilfsfunktionen für kommende "Zahlenwelt" Beispiele.

Wir werden Beispiele betrachten, in denen wir Welten modellieren, in denen jeder Person eine Zahl zugewiesen wird, Typ *person* ⇒ *int*. Diese Zahl kann zum Beispiel der Besitz oder Wohlstand einer Person sein, oder das Einkommen. Dabei ist zu beachten, dass Gesamtbesitz und Einkommen (über einen kurzen Zeitraum) recht unterschiedliche Sachen modellieren, jedoch den gleichen Typen in unserem Modell haben werden.

Hier sind einige Hilfsfunktionen um mit dem Typ *person* ⇒ *int* allgemein zu arbeiten.

Default: Standardmäßig hat jede Person *0*:

definition *DEFAULT* :: ‹*person* ⇒ *int*› (€) **where**
  ‹*DEFAULT* ≡ λ*p*. *0*›

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

Beispiel mit fancy Syntax:

beispiel ‹(€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)) *Bob* = *3*›

Das Beispiel liest sich wie folgt. Die Welt (€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)) :: *person* ⇒ *int* ist eine Funktion von *person* nach *int*. Wir rufen diese Funktion mit den Parameter *Bob* auf. Das Ergebnis ist *3*.

Die Funktion (€(*Alice* := *4*, *Carol* := *4*)) lässt sich auch mit Hilfe folgender Hilfsfunktionen als eine Menge von Tupeln darstellen.

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

Folgende Syntaxabkürzungen erlauben es uns eine einfachere Notation einzuführen, um den Besitz einer Person zu erhöhen oder zu verringern.

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*))›

beispiel ‹⟦€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)(*Bob* += *4*)⟧ *Bob* = *7*›
beispiel ‹⟦€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)(*Bob* −= *4*)⟧ *Bob* = −*1*›

Erhöhen und verringern heben sich auf.

beispiel fixes *n*:: ‹*int*› **shows** ‹⟦⟦*f*(*p* += *n*)⟧(*p* −= *n*)⟧ = *f*›

Folgende Funktion wählt diskriminierungsfrei eine *′person* eindeutig anhand Ihres Besitzes aus.

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*)›

Folgende Hilfsdefinition definiert eindeutig das Element in einer Einelementigen Menge, wenn dieses existiert.

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*›

```
beispiel ‹the-single-elem {a} = Some a›
beispiel ‹the-single-elem {1::nat,2} = None›
beispiel ‹the-single-elem {} = None›
```

Hier sehen wir unser Shallow Embedding: Unsere Definition *the-single-elem* lässt sich komplett auf bereits existierende Konzepte in HOL reduzieren.

```
lemma the-single-elem:
  ‹the-single-elem s = (if is-singleton s then Some (Set.the-elem s) else None)›

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*: ‹$as \neq [] \longleftrightarrow (\exists\, a.\ a \in set\ as)$›

lemma *enum-class-not-empty-list*: ‹$enum\text{-}class.enum \neq []$›

lemma *alles-kaputt-machen-code-help*:
‹$(\lambda\text{-}.\ Min\ (range\ x) - 1) = (\lambda\text{-}.\ min\text{-}list\ (map\ x\ enum\text{-}class.enum) - 1)$›

swap funktioniert auch auf Mengen.

lemma ‹$(swap\ Alice\ Carol\ id)\ `\ \{Alice, Bob\} = \{Carol, Bob\}$›

11 Beispiel: Zahlenwelt

In diesem Abschnitt werden wir ein Beispiel sehen.

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:

beispiel ‹$gesamtbesitz\ (Zahlenwelt\ (€(Alice := 4,\ Carol := 8))) = 12$›
beispiel ‹$gesamtbesitz\ (Zahlenwelt\ (€(Alice := 4,\ Carol := 4))) = 8$›

Mein persönlicher Besitz:

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

Beispiel:

beispiel ‹$meins\ Carol\ (Zahlenwelt\ (€(Alice := 8,\ Carol := 4))) = 4$›

Um den `SchleierNichtwissen.thy` zu implementieren:

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

Beispiel:

beispiel ‹$zahlenwps\ Alice\ Carol\ (Zahlenwelt\ (€(Alice := 4,\ Bob := 6,\ Carol := 8)))$
$= (Zahlenwelt\ (€(Alice := 8,\ Bob := 6,\ Carol := 4)))$›

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

definition ‹$initialwelt \equiv Zahlenwelt\ (€(Alice := 5,\ Bob := 10,\ Carol := -3))$›

11.1 Ungültige Handlung

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

```
beispiel ‹¬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*.

```
beispiel‹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)))›
beispiel‹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)))›
beispiel ‹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-erschaffen*:
‹*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 *stehlen* :: ‹*int ⇒ int ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt option*› **where**
‹*stehlen beute opfer-nach-besitz dieb (Zahlenwelt besitz) =*
map-option Zahlenwelt (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-reset*:
‹*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))*›
beispiel ‹*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*›

Die Beispielhandlungsabsichten, die wir betrachten wollen.

definition ‹*handlungsabsichten* ≡ [
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen 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*)›

Die Eigenschaft *maxime-und-handlungsabsicht-generalisieren* wird von den meisten Handlungsabsichten erfüllt. Jedoch nicht von *reset*. Das nicht-wohlgeformte *stehlen-nichtwf* erfüllt sie allerdings schon.

lemma ‹*ha* ∈ {
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen-nichtwf 5 Bob*),
Handlungsabsicht (*stehlen 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* nicht:

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

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

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

Folgendes Beispiel zeigt, dass die *maxime-zahlenfortschritt* den kategorischen Imperativ auf unseren *handlungsabsichten* nicht erfüllt (zu sehen an dem *False* im *bsp-erfuellte-maxime.*). Die Handlungsabsichten *BeispielZahlenwelt.stehlen* als auch *reset* sind nicht mit der Maxime vereinbar. Für die übrigen Handlungsabsichten ist das Ergebnis aber intuitiv:

- *erschaffen* ist erlaubt.
- da *unmoeglich* im Nichtstuen endet, ist dies auch erlaubt.
- *alles-kaputt-machen* ist verboten.

beispiel ‹*erzeuge-beispiel*
zahlenwps (*Zahlenwelt* (€(*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*)))
handlungsabsichten
maxime-zahlenfortschritt =
Some
⦇
bsp-erfuellte-maxime = *False*,
bsp-erlaubte-handlungen = [
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht unmoeglich],
bsp-verbotene-handlungen = [*Handlungsabsicht alles-kaputt-machen*],
bsp-uneindeutige-handlungen = [
Handlungsabsicht (*stehlen 5 10*),
Handlungsabsicht reset]⦈›

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* (*stehlen 5 10*))›

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

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

11.5 Maxime für allgemeinen Fortschritt

Wir können die vorherige 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 *True* im *bsp-erfuellte-maxime*.

Die Handlungsabsichten werden eingeordnet wie erwartet:

- *erschaffen* ist gut, *unmoeglich* (bedeutet: Nichtstun) ist auch okay.
- *BeispielZahlenwelt.stehlen*, *reset*, *alles-kaputt-machen* ist schlecht.

beispiel ‹*erzeuge-beispiel*
zahlenwps (*Zahlenwelt* (€(*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*)))
handlungsabsichten
maxime-altruistischer-fortschritt =
Some
⦇
bsp-erfuellte-maxime = *True*,
bsp-erlaubte-handlungen = [
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht unmoeglich],
bsp-verbotene-handlungen = [
Handlungsabsicht (*stehlen 5 10*),
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen],
bsp-uneindeutige-handlungen = []⦈›

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 ‹$ha \in$ *set handlungsabsichten* $\Longrightarrow$
kategorischer-imperativ-auf ha welt maxime-altruistischer-fortschritt›

11.6 Maxime für strikten individuellen Fortschritt

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

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

Folgendes ernüchterndes Beispiel zeigt, die Maxime *individueller-strikter-fortschritt* erfüllt nicht den kategorischen Imperativ. Entweder erlaubt die Maxime keine Assuage über eine Handlungsabsicht, oder die Handlungsabsicht ist verboten.

beispiel ‹*erzeuge-beispiel*
zahlenwps (*Zahlenwelt* (€(*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*)))
handlungsabsichten
(*Maxime individueller-strikter-fortschritt*) =
Some
⦇
bsp-erfuellte-maxime = *False*,
bsp-erlaubte-handlungen = [],
bsp-verbotene-handlungen = [
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich],
bsp-uneindeutige-handlungen = [
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen 5 10*),
Handlungsabsicht reset]
⦈›

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

lemma ‹$\neg$ *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 dabei nicht erhöht:

beispiel
‹⦇
dbg-opfer = *Bob*, *dbg-taeter* = *Alice*,
dbg-handlung = *handeln Alice initialwelt* (*Handlungsabsicht* (*erschaffen 5*))
⦈
∈ *debug-maxime id 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:

beispiel ‹*moralisch initialwelt*
(*Maxime* (λ*ich*. *globaler-strikter-fortschritt*)) (*Handlungsabsicht* (*erschaffen 5*))›

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

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

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

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

Folgendes Beispiel zeigt, dass die Maxime *globaler-strikter-fortschritt* den kategorischen Imperativ erfüllen kann. Die Handlungsabsichten sind fast intuitiv in erlaubt und verboten eingeordnet.

- *erschaffen 5* ist erlaubt.
- *BeispielZahlenwelt.stehlen*, *reset*, *alles-kaputt-machen* sind verboten. Allerdings ist auch *unmoeglich* verboten, da die Maxime Untätigkeit verbietet. Dieser letzte Fall ist unschön.

beispiel ‹*erzeuge-beispiel*
zahlenwps (*Zahlenwelt* (€(*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*)))
handlungsabsichten
(*Maxime* (λ*ich*. *globaler-strikter-fortschritt*)) =
Some
⦇

bsp-erfuellte-maxime = *True*,
bsp-erlaubte-handlungen = [*Handlungsabsicht* (*erschaffen 5*)],
bsp-verbotene-handlungen = [
Handlungsabsicht (*stehlen 5 10*),
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich],
bsp-uneindeutige-handlungen = []⦈›

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:

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

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

beispiel ‹*moralisch initialwelt*
(*Maxime* (λ*ich*. *globaler-fortschritt*)) (*Handlungsabsicht* (*stehlen-nichtwf 5 Bob*))›
beispiel ‹*moralisch initialwelt*
(*Maxime* (λ*ich*. *globaler-fortschritt*)) (*Handlungsabsicht* (*stehlen 5 10*))›

Folgendes Beispiel zeigt, dass die Maxime *globaler-fortschritt* den kategorischen Imperativ erfüllen kann. Die Handlungsabsichten sind meiner Meinung nach intuitiv (basierend auf der globalen Betrachtung der Maxime) in erlaubt und verboten eingeordnet.

- *erschaffen* ist erlaubt. Auch *BeispielZahlenwelt.stehlen* ist erlaubt, da dabei "doem Kollektiv" kein Besitz verloren geht. Untätigkeit wird wieder über *unmoeglich* erlaubt.
- *reset* und *alles-kaputt-machen* sind verboten.

beispiel ‹*erzeuge-beispiel*
zahlenwps (*Zahlenwelt* (€(*Alice* := *5*, *Bob* := *10*, *Carol* := −*3*)))
handlungsabsichten
(*Maxime* (λ*ich*. *globaler-fortschritt*)) =
Some
⦇
bsp-erfuellte-maxime = *True*,
bsp-erlaubte-handlungen = [
Handlungsabsicht (*erschaffen 5*),
Handlungsabsicht (*stehlen 5 10*),

Handlungsabsicht unmoeglich],
bsp-verbotene-handlungen = [
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen],
bsp-uneindeutige-handlungen = []⦈›

Auch allgemein lässt ich beweisen, dass diese Maxime für sehr viele Handlungsabsichten den kategorischen Imperativ erfüllt.

theorem
‹$\forall$ *p. maxime-und-handlungsabsicht-generalisieren zahlenwps welt*
(*Maxime* (λ*ich. globaler-fortschritt*)) *ha p* $\Longrightarrow$
wohlgeformte-handlungsabsicht zahlenwps welt ha $\Longrightarrow$
kategorischer-imperativ-auf ha welt (*Maxime* (λ*ich*::*person. globaler-fortschritt*))›

Sollte man das Kollektiv höher stellen als das Individuum und damit effektiv das Recht auf Privateigentum ablehnen (was ich persönlich nicht unterstütze), so ist *globaler-fortschritt* eine Maxime mit schönen Eigenschaften.

11.9 Ungültige Maxime

Es ist verboten, in einer Maxime eine spezielle Person hardzucoden. Technisch könnte so eine Maxime den *kategorischer-imperativ-auf* erfüllen. Dies wollen wir aber nicht, da dies gegen die Gleichbehandlung aller Menschen verstoßen würde. Das Ergebnis wären verdrehte Moralbewertungen, welche moralische Entscheidungen ausschließlich basierend auf egoistischen Bedürfnissen der hardgecodeten Personen basieren.
Beispielsweise könnten wir *individueller-fortschritt* nicht mehr parametrisiert verwenden, sondern einfach *Alice* reinschreiben:

beispiel ‹*individueller-fortschritt Alice*
= (λ*h. case h of Handlung vor nach* $\Rightarrow$ (*meins Alice vor*) $\leq$ (*meins Alice nach*))›

Solch eine Maxime ist allerdings nicht wohlgeformt.

beispiel ‹$\neg$ *wohlgeformte-maxime-auf*
(*handeln Alice initialwelt* (*Handlungsabsicht* (*stehlen 5 10*))) *zahlenwps*
(*Maxime* (λ*ich. individueller-fortschritt Alice*))›

Sobald wir aufhören *Alice* hardzucoden, wird die Maxime wohlgeformt.

beispiel ‹*wohlgeformte-maxime-auf*
(*handeln Alice initialwelt* (*Handlungsabsicht* (*stehlen 5 10*))) *zahlenwps*
(*Maxime* (λ*ich. individueller-fortschritt ich*))›

Unser *erzeuge-beispiel* verweigert die Arbeit auf nicht-wohlgeformten Maximen.

11.10 Uneindeutige Handlungen

Bis jetzt haben wir den Schuldigen immer bei der Maxime gesucht, wenn der kategorische Imperativ nicht erfüllt war und wir somit über bestimmte Handlungsabsichten keine Aussage treffen konnten. Gibt es jedoch auch Handlungsabsichten welche vermutlich unabhängig von jeder durchdachten Maxime keine Bewertung im Sinne des kategorischen Imperativs erlauben?

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*)›
beispiel ‹*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*)›

Die Handlungsabsicht *collatzh* generalisiert nicht mit der *maxime-zahlenfortschritt*. Dies ist keine große Überraschung, da *reset* auch nicht mit dieser Maxime generalisiert hat und wir die Maxime auch für ungeeignet befunden haben.

beispiel
‹¬ *maxime-und-handlungsabsicht-generalisieren*
zahlenwps (*Zahlenwelt* (€(*Alice* := *2*, *Bob* := *3*)))
maxime-zahlenfortschritt (*Handlungsabsicht collatzh*) *Alice*›

Für unsere hochgelobte *maxime-altruistischer-fortschritt* hingegen haben wir noch kein Beispiel einer Handlungsabsicht gesehen, welche nicht mit ihr generalisiert hat. Dies wirft die Frage auf: "gibt es überhaupt wohlgeformte Handlungsabsichten, welche nicht mit *maxime-altruistischer-fortschritt* generalisieren?" Die Antwort liefert *collatzh*.

beispiel
‹¬ *maxime-und-handlungsabsicht-generalisieren*
zahlenwps (*Zahlenwelt* (€(*Alice* := *2*, *Bob* := *3*)))
maxime-altruistischer-fortschritt (*Handlungsabsicht collatzh*) *Alice*›

Wir haben *collatzh* bis jetzt immer bei der Bewertung von Maximen ausgeschlossen. Das Ergebnis vorweg: Ein kategorischer Imperativ, egal welche vielversprechende Maxime, gilt nicht für die Handlungsabsicht *collatzh*.

beispiel
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht collatzh*)

initialwelt maxime-zahlenfortschritt›
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht collatzh*)
initialwelt maxime-altruistischer-fortschritt›
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht collatzh*)
initialwelt (*Maxime* (*λich. globaler-fortschritt*))›

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. Oder anders ausgedrückt: können wir ein allgemeines Gesetz wollen, welches eine Aussage über die Handlungsabsicht *collatzh* macht? Ich argumentiere, dass wir solch ein Gesetz nicht wollen, da

- Würden wir nur die Auswirkung von *collatzh* betrachten, (also die resultierende *'welt handlung*, nicht die *Handlungsabsicht collatzh*::(*person*, *zahlenwelt*) *handlungsabsicht*) so kann diese Auswirkung durchweg positiv sein, und wir möchten etwas positives nicht verbieten.
- Jedoch hat die Handlungsabsicht auch negative Charakterzüge, da wir billigend in Kauf nehmen, dass Besitz vernichtet werden könnte. Daher möchten wir diese Absicht auch nicht uneingeschränkt erlauben. Besonders deutlich wird dies bei folgender zugespitzten Handlungsabsicht, welche billigend die komplette Vernichtung allen Besitzes in Kauf nehmen würde.

definition *uneindeutiger-charakter*:: ‹*person* ⇒ *zahlenwelt* ⇒ *zahlenwelt option*› **where**
‹*uneindeutiger-charakter* ≡
(*λich welt. if meins ich welt mod 2 = 0*
then alles-kaputt-machen ich welt
else erschaffen 5 ich welt)›

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

beispiel
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht uneindeutiger-charakter*)
initialwelt maxime-zahlenfortschritt›
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht uneindeutiger-charakter*)
initialwelt maxime-altruistischer-fortschritt›
‹¬ *kategorischer-imperativ-auf* (*Handlungsabsicht uneindeutiger-charakter*)
initialwelt (*Maxime* (*λich. globaler-fortschritt*))›

Mir gefällt, dass der (extensionale) kategorische Imperativ prinzipiell sagt, dass wir die Handlungsabsicht *uneindeutiger-charakter* nicht in einem allgemeinen Gesetz behandeln können, da die potenziellen positiven Auswirkungen im starken Gegensatz zu der potenziell destruktiven zugrundeliegenden Absicht stehen.

Wenn wir allerdings ausnutzen, dass Handlungsabsichten partiell sein können, und so den guten und den schlechten Charakter in eigenständige Handlungsabsichten separieren, so können wir wieder allgemeines Aussage über die beiden Handlungsabsichten machen.

```
definition partiell-guter-charakter:: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹partiell-guter-charakter ≡
    (λich welt. if meins ich welt mod 2 = 0
              then None
              else erschaffen 5 ich welt)›

definition partiell-schlechter-charakter:: ‹person ⇒ zahlenwelt ⇒ zahlenwelt option› where
  ‹partiell-schlechter-charakter ≡
    (λich welt. if meins ich welt mod 2 = 0
              then alles-kaputt-machen ich welt
              else None)›

beispiel ‹erzeuge-beispiel
    zahlenwps (Zahlenwelt (€(Alice := 5, Bob := 10, Carol := −3)))
    [Handlungsabsicht partiell-guter-charakter, Handlungsabsicht partiell-schlechter-charakter]
    maxime-altruistischer-fortschritt
= Some
  ⦇
   bsp-erfuellte-maxime = True,
   bsp-erlaubte-handlungen = [Handlungsabsicht partiell-guter-charakter],
   bsp-verbotene-handlungen = [Handlungsabsicht partiell-schlechter-charakter],
   bsp-uneindeutige-handlungen = []
  ⦈›
```

12 Änderungen in Welten

In diesem Abschnitt werden wir Änderungen in Welten, und darauf basierend, Abmachungen modellieren.

Bei einer Änderung keine eine Person entweder etwas verlieren oder gewinnen. Dieses einfache Modell ist natürlich auf unsere Zahlenwelten angepasst, in der normalerweise Typ *'etwas* ein *int* ist.

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

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

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

```
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*›

beispiel ‹*betroffene [Verliert Alice (2::int), Gewinnt Bob 3, Gewinnt Carol 2, Verliert Eve 1]*
= [Alice, Bob, Carol, Eve]›
beispiel ‹*betroffene [Verliert Alice (5::nat), Gewinnt Bob 3, Verliert Eve 7]*
= [Alice, Bob, Eve]›
beispiel ‹*betroffene [Verliert Alice (5::nat), Gewinnt Alice 3]*
= [Alice, Alice]›

12.1 Deltas

Deltas, d.h. Unterschiede zwischen Welten. Ein Delta ist eine Liste von Änderungen. Wir definieren das **type-synonym** delta als die Funktion, welche solch eine Liste berechnet, gegeben die Handlung welche die Veränderung hervorruft.

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

Eine Liste von Änderungen lässt sich ausführen.

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)⟧*›

Die lokale Variable *bes::'person ⇒ 'etwas* stellt dabei den aktuellen Besitz dar. Die Ausgabe der Funktion ist der modifizierte Besitz, nachdem die Änderung ausgeführt wurde.

beispiel
‹*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))›

beispiel
‹*aenderung-ausfuehren*
[Verliert Alice (2::int), Verliert Alice 6]
(€(Alice:=8, Bob:=3, Eve:= 5))
=
(€(Bob:=3, Eve:= 5))›

Im vorherigen Beispiel verliert *Alice* alles. Da sie nun den €-Wert von *0* besitzt, wird ihre Besitz nicht angezeigt.

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 vom Typ $'person \rightharpoonup 'etwas$, da diese eindeutig einer $'person$ ein $'etwas$ zuordnet. Dies funktioniert allerdings nur, wenn $'etwas::\{uminus,plus\}$ mit Plus und Minus dargestellt werden kann, um *Gewinnt* und *Verliert* darzustellen. Allerdings ist auch diese Darstellung nicht eindeutig, da z.B. $[Alice \mapsto 0::'a] = Map.empty$ semantisch gilt, solange $0::'a$ ein neutrales Element ist. Deshalb stellen wir eine Abmachung als eine totale Funktion vom Typ $'person \Rightarrow ('etwas::\{uminus, plus, zero\})$ dar. Der Term $(\lambda\text{-}.\ 0::'a)(Alice := 3::'a, Bob := -\ (3::'a))$ bedeutet *Alice* bekommt 3, *Bob* verliert 3.

type-synonym $('person, 'etwas)$ *abmachung* = ‹$'person \Rightarrow 'etwas$›

Folgende Funktion konvertiert eine Liste von Änderungen in ein Abmachung. Persönlich finde ich es schöner eine Liste von Änderungen aufzuschreiben, mathematisch ist eine Abmachung allerdings überlegen. Folgende Funktion sorgt dafür, dass wir Abmachungen dennoch als Liste von Änderungen aufschreiben können, dann allerdings mit der Abmachung weiterrechnen.

```
fun to-abmachung
  :: ‹('person, 'etwas::{ord,zero,plus,minus,uminus}) aenderung list ⇒ ('person, 'etwas) abmachung›
where
  ‹to-abmachung [] = (λp. 0)›
| ‹to-abmachung (delta # deltas) =
    ⟦(to-abmachung deltas)(betroffen delta += aenderung-val delta)⟧›
```

```
beispiel ‹[to-abmachung [Gewinnt Alice (3::int)], to-abmachung [Gewinnt Alice 3, Verliert Bob 3]]
        = [(λp.0)(Alice := 3), (λp.0)(Alice := 3, Bob := −3)]›
```

Personen, welche von einer Abmachung betroffen sind.

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

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

Eine Abmachung lässt sich ausführen. Dabei wird effektiv die gegebene *besitz* Funktion upgedated.

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

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

Es ist equivalent eine Abmachung oder die entsprechende Änderungsliste auszuführen.

lemma *abmachung-ausfuehren-aenderung*:
fixes *abmachung* :: ‹('*person*::*enum*, '*etwas*::*ordered-ab-group-add*) *abmachung*›
shows ‹*abmachung-ausfuehren abmachung* = *aenderung-ausfuehren* (*abmachung-to-aenderung abmachung*)›

12.3 Konsens

Laut https://de.wikipedia.org/wiki/Konsens#Konsens_im_Rechtssystem lässt sich Konsens wie folgt 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*›

Folgendes Beispiel liest sich wie folgt:
(λ-. [])(*Alice* := [*to-abmachung* [*Gewinnt Alice 3*], *to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*]], *Bob* := [*to-abmachung* [*Gewinnt Alice 3*, *Verliert Bob 3*]])
Alice stimmt folgendem zu:

- *Alice* bekommt 3.
- *Alice* bekommt 3 und *Bob* muss 3 abgeben.

Bob stimmt folgendem zu:

- *Alice* bekommt 3 und *Bob* muss 3 abgeben.

Wir könnten also sagen, dass Konsens zwischen *Alice* und *Bob* herrscht, dass 3 Besitz von *Bob* auf *Alice* übergehen. Zusätzlich wäre es in diesem Beispiel auch okay für *Alice*, wenn sie 3 Besitz erhalten würde, ohne dass *Bob* 3 Besitz verliert.

Folgendes Prädikat prüft, ob für eine gegebene Abmachung Konsens herrscht.

```
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)›
```

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 (λp k. k(p := remove1 abmachung (k p))) (abmachungs-betroffene abmachung) kons›
```

```
beispiel
  ‹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
    = (λp. if p ∈ set (abmachungs-betroffene a) then remove1 a (kons p) else (kons p))›
```

Folgendes Prädikat prüft ob eine Abmachung korrekt aus dem Konsens entfernt wurde. Dies sollte normalerweise direkt nachdem eine Abmachung eingelöst wurde geschehen.

```
definition konsens-wurde-entfernt
  :: ‹('person::enum, 'etwas::zero) abmachung ⇒
      ('person, 'etwas) globaler-konsens ⇒ ('person, 'etwas) globaler-konsens ⇒ bool›
where
  ‹konsens-wurde-entfernt abmachung konsens-vor konsens-nach ≡
    ∀ betroffene-person ∈ set (abmachungs-betroffene abmachung).
       mset (konsens-vor betroffene-person) = mset (abmachung#(konsens-nach betroffene-person))›
```

Wir müssen hier Multisets (*mset*) verwenden, da eine Abmachung sowohl mehrfach vorkommen kann aber nur einmal eingelöst wird und die Reihenfolge in welcher die Abmachungen angeordnet sind egal ist.

Folgendes gilt nicht *konsens-wurde-entfernt a konsens* (*konsens-entfernen a konsens*), da *konsens-entfernen* nur einen existierenden Konsens entfernt. Sollte der gegebene Konsens nicht existieren passiert nichts!

beispiel
‹$\mathit{konsens} = (\lambda\text{-}.\ []) \Longrightarrow a = \mathit{to\text{-}abmachung}\ [\mathit{Gewinnt\ Alice}\ (3{::}\mathit{int}),\ \mathit{Verliert\ Bob}\ 3]$
$\Longrightarrow \neg\ \mathit{konsens\text{-}wurde\text{-}entfernt\ a\ konsens}\ (\mathit{konsens\text{-}entfernen\ a\ konsens})$›

Wenn wir allerdings Konsens haben, dann verhalten sich *konsens-wurde-entfernt* und *konsens-entfernen* doch wie erwartet.

lemma *konsens-wurde-entfernt-konsens-entfernen*:
‹$\mathit{enthaelt\text{-}konsens\ a\ konsens} \Longrightarrow \mathit{konsens\text{-}wurde\text{-}entfernt\ a\ konsens}\ (\mathit{konsens\text{-}entfernen\ a\ konsens})$›

Gegeben eine Handlung berechnet folgende Funktion die Abmachung, aus der diese Handlung resultiert haben könnte.

definition *reverse-engineer-abmachung*
:: ‹$('\mathit{person}{::}\mathit{enum} \Rightarrow '\mathit{etwas}{::}\mathit{linordered\text{-}ab\text{-}group\text{-}add})\ \mathit{handlung} \Rightarrow ('\mathit{person},\ '\mathit{etwas})\ \mathit{abmachung}$›
where
‹$\mathit{reverse\text{-}engineer\text{-}abmachung}\ h \equiv$
$\mathit{fold}\ (\lambda p\ \mathit{acc}.\ \mathit{acc}(p := (\mathit{nachher}\ h\ p) - (\mathit{vorher}\ h\ p)))\ \mathit{Enum.enum}\ (\lambda\text{-}.\ 0)$›

Sollte die Abmachung vom Typ (*person*, *int*) *abmachung* sein, ist dies eindeutig.

lemma *reverse-engineer-abmachung-delta-num-fun*:
‹$\mathit{reverse\text{-}engineer\text{-}abmachung}\ h = \mathit{to\text{-}abmachung}\ (\mathit{delta\text{-}num\text{-}fun}\ h)$›
lemma *reverse-engineer-abmachung*:
‹$\mathit{reverse\text{-}engineer\text{-}abmachung}\ (\mathit{Handlung\ welt\ welt}') = a \longleftrightarrow \mathit{abmachung\text{-}ausfuehren\ a\ welt} = \mathit{welt}'$›

13 Beispiel: Zahlenwelt2

In diesem Abschnitt werden wir ein weiteres Beispiel sehen.

Dieses Beispiel ist ähnlich zum Beispiel Zahlenwelt in Abschnitt 11. Allerdings führen wir einige Erweiterungen ein:

- Jeder Person wird weiterhin ihr Besitz zugeordnet.
- Neben dem Besitz gibt es auch ein Modell von Konsens. Dabei soll Konsens die Liste aller bereits getroffenen Abmachungen darstellen, bzw modellieren, zu was die Leute bereit wären. So lässt sich beispielsweise Schenken (Besitzübergang mit Konsens) von Stehlen (Besitzübergang ohne Konsens) unterscheiden.
- Es gibt eine spezielle Entität, nämlich den Staat. Diese Entität ist nicht in der Menge der natürlichen Personen enthalten. Dies erlaubt es z.B. den Staat in Handlungsabsichten hardzucoden und gleichzeitig eine wohlgeformte Handlungsabsicht zu haben. TODO: machen

- Als weitere spezielle Entität wird die Umwelt eingeführt.

```
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›

beispiel ‹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 einvernehmlich :: ‹zahlenwelt handlung ⇒ bool›
where
  ‹einvernehmlich h ≡
    let abmachung = reverse-engineer-abmachung (map-handlung besitz h)
    in enthaelt-konsens abmachung (vorher h)
       ∧ konsens-wurde-entfernt abmachung (konsens (vorher h)) (konsens (nachher h))›
```

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

```
lemma ‹einvernehmlich (handeln p welt (Handlungsabsicht (λp w. Some w)))›

beispiel
  ‹einvernehmlich (handeln Alice initialwelt
    (Handlungsabsicht (λp w. Some
       (w⦇ besitz := ⟦⟦(besitz w)(Alice += 3)⟧(Bob −= 3)⟧,
            konsens := konsens-entfernen (to-abmachung [Gewinnt Alice (3::int), Verliert Bob 3]) (konsens w)
⦈)))))›
beispiel ‹¬ einvernehmlich (handeln Alice initialwelt
```

(*Handlungsabsicht* (*λp w. Some* (*w*⦇ *besitz* := ⟦⟦(*besitz w*)(*Alice* += *3*)⟧(*Bob* −= *3*)⟧ ⦈)))))›
beispiel ‹¬ *einvernehmlich* (*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*) ⦈›

beispiel ‹*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›

beispiel ‹*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*
⦈›

beispiel ‹*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*
⦈›

beispiel ‹*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.

```
beispiel ‹¬ 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-existierende-abmachung-einloesen:
  ‹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.

```
beispiel ‹¬ 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.

```
beispiel ‹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.

```
beispiel ‹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 *einvernehmlich*. Das *reverse-engineer-abmachung* macht also das Richtige.

```
lemma einvernehmlich-existierende-abmachung-einloesen:
 ‹einvernehmlich (handeln p welt (Handlungsabsicht existierende-abmachung-einloesen))›

fun stehlen :: ‹int ⇒ int ⇒ person ⇒ zahlenwelt ⇒ zahlenwelt option› where
 ‹stehlen beute opfer-nach-besitz dieb welt =
      map-option (λb. welt⦇besitz := b⦈) (Zahlenwelt.stehlen beute opfer-nach-besitz dieb (besitz welt))›

beispiel‹stehlen 3 10 Alice initialwelt =
Some ⦇
 besitz = (€(Alice := 8, Bob := 7, 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
⦈›
```

BeispielZahlenwelt2.stehlen und *existierende-abmachung-einloesen* können ununterscheidbar sein, was den *besitz* betrifft. Der Hauptunterschied ist, ob *konsens* eingelöst wurde oder nicht.

beispiel
‹*besitz* (*the* (*stehlen 3 10 Alice initialwelt*)) =
besitz (*the* (*existierende-abmachung-einloesen Bob initialwelt*))›
‹*konsens* (*the* (*stehlen 3 10 Alice initialwelt*)) ≠
konsens (*the* (*existierende-abmachung-einloesen Bob initialwelt*))›

Ressourcen können nicht aus dem Nichts erschaffen werden. Diese Handlungsabsicht entnimmt der Natur und weist einer Person zu.

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* ⦈)›

Diese Handlungsabsicht weist allen Personen ein Besitz von 0 zu. Dies vernichtet allen Besitz. Personen mit Schulden (negativem Besitz) könnten jedoch profitieren.

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

Die Handlungsabsicht die alles kaputt macht. Die Handlungsabsicht sucht sich den minimalen Besitz aller Personen und weist allen Personen Eins weniger zu. Damit haben alle Personen definitiv weniger als zuvor.

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*)⦈)›

Die unmögliche Handlungsabsicht, welche immer scheitert.

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

Die Beispielhandlungsabsichten, die wir betrachten wollen.

definition ‹*handlungsabsichten* ≡ [
Handlungsabsicht (*abbauen 5*),
Handlungsabsicht (*stehlen 3 10*),
Handlungsabsicht existierende-abmachung-einloesen,
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen,
Handlungsabsicht unmoeglich
]›

lemma *wfh-handlungsabsichten*:

‹$ha \in$ *set handlungsabsichten* $\Longrightarrow$ *wohlgeformte-handlungsabsicht zahlenwps welt ha*›

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

definition *maxime-altruistischer-fortschritt* :: ‹(*person*, *zahlenwelt*) *maxime*› **where**
‹*maxime-altruistischer-fortschritt* $\equiv$
Maxime (λ*ich h*. $\forall$ *pX*. *individueller-fortschritt pX h*)›

beispiel ‹*erzeuge-beispiel*
zahlenwps initialwelt
handlungsabsichten
maxime-altruistischer-fortschritt
= *Some*
⦇
bsp-erfuellte-maxime = *False*,
bsp-erlaubte-handlungen = [
Handlungsabsicht (*abbauen 5*),
Handlungsabsicht unmoeglich],
bsp-verbotene-handlungen = [
Handlungsabsicht (*stehlen 3 10*),
Handlungsabsicht reset,
Handlungsabsicht alles-kaputt-machen],
bsp-uneindeutige-handlungen = [
Handlungsabsicht existierende-abmachung-einloesen]
⦈›

definition *maxime-hatte-konsens* :: ‹(*person*, *zahlenwelt*) *maxime*› **where**
‹*maxime-hatte-konsens* $\equiv$ *Maxime* (λ*ich h*. *einvernehmlich h*)›

beispiel ‹$\forall h \in$ *set* (*alle-moeglichen-handlungen initialwelt* (*Handlungsabsicht existierende-abmachung-einloesen*)).
wohlgeformte-maxime-auf
h zahlenwps
maxime-hatte-konsens›

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

beispiel ‹*erzeuge-beispiel*
zahlenwps initialwelt
[*Handlungsabsicht existierende-abmachung-einloesen*]
maxime-hatte-konsens
= *Some*

```
  ⦇
   bsp-erfuellte-maxime = True,
   bsp-erlaubte-handlungen = [Handlungsabsicht existierende-abmachung-einloesen],
   bsp-verbotene-handlungen = [],
   bsp-uneindeutige-handlungen = []⦈›

beispiel ‹erzeuge-beispiel
  zahlenwps initialwelt
  [Handlungsabsicht (abbauen 5),
   Handlungsabsicht (stehlen 3 10),
   Handlungsabsicht reset,
   Handlungsabsicht alles-kaputt-machen,
   Handlungsabsicht unmoeglich]
  maxime-altruistischer-fortschritt
= Some
  ⦇
   bsp-erfuellte-maxime = True,
   bsp-erlaubte-handlungen = [
      Handlungsabsicht (abbauen 5),
      Handlungsabsicht unmoeglich],
   bsp-verbotene-handlungen = [
      Handlungsabsicht (stehlen 3 10),
      Handlungsabsicht reset,
      Handlungsabsicht alles-kaputt-machen],
   bsp-uneindeutige-handlungen = []⦈›


beispiel ‹erzeuge-beispiel
  zahlenwps initialwelt
  handlungsabsichten
  (MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens)
= Some
  ⦇
   bsp-erfuellte-maxime = True,
   bsp-erlaubte-handlungen = [
      Handlungsabsicht (abbauen 5),
      Handlungsabsicht existierende-abmachung-einloesen,
      Handlungsabsicht unmoeglich],
   bsp-verbotene-handlungen = [
      Handlungsabsicht (stehlen 3 10),
      Handlungsabsicht reset,
      Handlungsabsicht alles-kaputt-machen],
   bsp-uneindeutige-handlungen = []⦈›
```

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*:
‹∀ *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›

Folgendes Theorem zeigt, dass das *MaximeDisj* Konstrukt in jeder Welt funktioniert.

theorem
‹*ex-erfuellbare-instanz maxime-altruistischer-fortschritt welt ha* ∧
(∀ *p*. *maxime-und-handlungsabsicht-generalisieren*
zahlenwps welt maxime-altruistischer-fortschritt ha p)
∨
ex-erfuellbare-instanz maxime-hatte-konsens welt ha ∧
(∀ *p*. *maxime-und-handlungsabsicht-generalisieren*
zahlenwps welt maxime-hatte-konsens ha p) ⟹
wohlgeformte-handlungsabsicht zahlenwps welt ha ⟹
kategorischer-imperativ-auf ha welt
(*MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens*)›

Wir könnten zusätzlich noch eine Maxime einführen, welche besagt, dass die Umwelt nicht zerstört werden darf.

definition *maxime-keine-umweltzerstoerung* :: ‹(*person*, *zahlenwelt*) *maxime*› **where**
‹*maxime-keine-umweltzerstoerung* ≡
Maxime (λ- *h*. *umwelt* (*vorher h*) ≤ *umwelt* (*nachher h*))›

Folgendes Beispiel ist wie die vorherigen Beispiel. Zusätzlich fügen wir jedoch noch *maxime-keine-umweltzerstoerung* via *MaximeConj* hinzu.

beispiel‹*erzeuge-beispiel*
zahlenwps initialwelt
handlungsabsichten
(*MaximeConj* (*MaximeDisj maxime-altruistischer-fortschritt maxime-hatte-konsens*)
maxime-keine-umweltzerstoerung)
= *Some*
⦇
bsp-erfuellte-maxime = *True*,

```
  bsp-erlaubte-handlungen = [
    Handlungsabsicht existierende-abmachung-einloesen,
    Handlungsabsicht unmoeglich],
  bsp-verbotene-handlungen = [
    Handlungsabsicht (abbauen 5),
    Handlungsabsicht (stehlen 3 10),
    Handlungsabsicht reset,
    Handlungsabsicht alles-kaputt-machen],
  bsp-uneindeutige-handlungen = []⦈›
```

Das Ergebnis ist fast wie in vorherigen Beispielen. Allerdings ist *abbauen* nun Teil der verbotenen Handlungsabsichten, da dabei Umwelt abgebaut wird.

14 Einkommensteuergesetzgebung

In diesem Abschnitt werden wir eine versuchen die Grundlagen der Einkommenssteuergesetzgebung zu modellieren.

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

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

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

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

Eine konkrete *steuer* Funktion wird noch nicht gegeben.

```
locale steuer-defs =
  fixes steuer :: ‹nat ⇒ nat› — Funktion: Einkommen -> Steuer
begin
  definition brutto :: ‹nat ⇒ nat› where
    ‹brutto einkommen ≡ einkommen›
  definition netto :: ‹nat ⇒ nat› where
    ‹netto einkommen ≡ einkommen − (steuer einkommen)›
  definition steuersatz :: ‹nat ⇒ percentage› where
    ‹steuersatz einkommen ≡ percentage ((steuer einkommen) / einkommen)›
end
```

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

```
definition beispiel-25prozent-steuer :: ‹nat ⇒ nat› where
  ‹beispiel-25prozent-steuer e ≡ nat ⌊real e ∗ (percentage 0.25)⌋›

beispiel
  ‹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* $\geq$ *einkommen-b* $\Longrightarrow$ *steuer einkommen-a* $\geq$ *steuer einkommen-b*›

and *leistung-lohnt-sich*:
‹*einkommen-a* $\geq$ *einkommen-b* $\Longrightarrow$ *netto einkommen-a* $\geq$ *netto einkommen-b*›

— Ein Existenzminimum wird nicht versteuert. Zahl Deutschland 2022, vermutlich sogar die falsche Zahl.
and *existenzminimum*:
‹*einkommen* $\leq$ *9888* $\Longrightarrow$ *steuer einkommen = 0*›

begin

end

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

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

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

beispiel
‹*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:

```
beispiel ‹einkommenssteuer 10 = 0›
beispiel ‹einkommenssteuer 10000 = 0›
```

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

```
beispiel ‹einkommenssteuer 14000 = floor ((14000−10347)∗0.14)›
beispiel ‹einkommenssteuer 14000 = 511›
```

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

```
beispiel ‹einkommenssteuer 20000 = 1857›
beispiel ‹einkommenssteuer 20000 =
```

floor ((14926−10347)∗0.14 + (20000−14926)∗0.2397)›

Höhere Einkommen führen zu einer höheren Steuer:

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

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

interpretation *steuersystem*
  where *steuer* = ‹*einkommenssteuer*›

15 Beispiel: Steuern

In diesem Abschnitt kombinieren wir das vorhergehende Modell der Einkommensteuergesetzgebung mit dem kategorischen Imperativ um ein Beispiel über moralische Aussagen über Steuern zu schaffen.

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).

Die Steuerlast sagt, wie viel Steuern gezahlt werden.

fun *steuerlast* :: ‹*person* ⇒ *steuerwelt handlung* ⇒ *int*› **where**
  ‹*steuerlast p* (*Handlung vor nach*) = ((*get-einkommen vor*) *p*) − ((*get-einkommen nach*) *p*)›

Das Einkommen vor Steuer wird brutto genannt.

fun *brutto* :: ‹*person* ⇒ *steuerwelt handlung* ⇒ *int*› **where**
  ‹*brutto p* (*Handlung vor nach*) = (*get-einkommen vor*) *p*›

Das Einkommen nach Steuer wird netto genannt.

fun *netto* :: ‹*person* ⇒ *steuerwelt handlung* ⇒ *int*› **where**
  ‹*netto p* (*Handlung vor nach*) = (*get-einkommen nach*) *p*›

Beispiele

beispiel ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* (€(*Alice*:=*8*))) (*Steuerwelt* (€(*Alice*:=*5*)))) = *3*›
beispiel ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* (€(*Alice*:=*8*))) (*Steuerwelt* (€(*Alice*:=*0*)))) = *8*›
beispiel ‹*steuerlast Bob* (*Handlung* (*Steuerwelt* (€(*Alice*:=*8*))) (*Steuerwelt* (€(*Alice*:=*5*)))) = *0*›
beispiel ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* (€(*Alice*:=−*3*))) (*Steuerwelt* (€(*Alice*:=−*4*)))) = *1*›
beispiel ‹*steuerlast Alice* (*Handlung* (*Steuerwelt* (€(*Alice*:=*1*))) (*Steuerwelt* (€(*Alice*:=−*1*)))) = *2*›

Folgende Menge beinhaltet alle Personen die mehr verdienen als ich.

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

beispiel ‹*mehrverdiener Alice*
    (Handlung (Steuerwelt (€(Alice:=8, Bob:=12, Eve:=7))) (Steuerwelt (€(Alice:=5))))
    = {Alice, Bob}›

Folgende Maxime versucht Steuergerechtigkeit festzuschreiben:

```
definition maxime-steuern :: ‹(person, steuerwelt) maxime› where
  ‹maxime-steuern ≡ Maxime
      (λich handlung.
          (∀ p∈mehrverdiener ich handlung.
              steuerlast ich handlung ≤ steuerlast p handlung)
        ∧ (∀ p∈mehrverdiener ich handlung.
              netto ich handlung ≤ netto p handlung)
        )›
```

Wenn die Steuerfunktion monoton ist, dann können wir auch einen sehr eingeschränkten kategorischen Imperativ zeigen.

```
lemma katimp-auf-handlungsabsicht-monoton:
  ‹(⋀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 Beispiel: Keiner Zahlt Steuern

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

```
beispiel ‹moralisch (Steuerwelt (€(Alice:=8, Bob:=3, Eve:= 5)))
                maxime-steuern (Handlungsabsicht (λich welt. Some welt))›
```

15.2 Beispiel: Ich zahle 1 Steuer

Das funktioniert nicht:

```
definition ‹ich-zahle-1-steuer ich welt ≡
  Some (Steuerwelt ⟦(get-einkommen welt)(ich −= 1)⟧)›
beispiel ‹¬ moralisch (Steuerwelt (€(Alice:=8, Bob:=3, Eve:= 5)))
                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 so Etwas jemals explizit gefordert haben.

15.3 Beiepiel: Jeder zahle 1 Steuer

Jeder muss steuern zahlen: funktioniert.

Das *ich* wird gar nicht verwendet, da jeder Steuern zahlt.

definition ‹*jeder-zahle-1-steuer ich welt* ≡
Some (*Steuerwelt* ((λ*e*. *e* − *1*) ∘ (*get-einkommen welt*)))›
beispiel ‹*moralisch* (*Steuerwelt* (€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)))
maxime-steuern (*Handlungsabsicht jeder-zahle-1-steuer*)›

15.4 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 *Steuerwelt* (€(*Alice* := *8*, *Bob* := *3*, *Eve* := *5*)) zahlt keiner Steuern.

beispiel ‹*ist-noop*
(*handeln Alice*(*Steuerwelt* (€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)))
(*Handlungsabsicht jeder-zahlt-einkommenssteuer*))›

beispiel ‹*moralisch* (*Steuerwelt* (€(*Alice*:=*8*, *Bob*:=*3*, *Eve*:= *5*)))
maxime-steuern (*Handlungsabsicht jeder-zahlt-einkommenssteuer*)›

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

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

Unser Beispiel erfüllt auch den kategorischen Imperativ.

beispiel ‹*erzeuge-beispiel*
steuerwps (*Steuerwelt* (€(*Alice*:=*10000*, *Bob*:=*14000*, *Eve*:= *20000*)))
[*Handlungsabsicht jeder-zahlt-einkommenssteuer*]
maxime-steuern
=

```
Some
  ⦇
   bsp-erfuellte-maxime = True,
   bsp-erlaubte-handlungen = [Handlungsabsicht jeder-zahlt-einkommenssteuer],
   bsp-verbotene-handlungen = [],
   bsp-uneindeutige-handlungen = []
  ⦈›
```

15.5 Vereinfachtes Deutsches Steuersystem vs. die Steuermaxime

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

```
lemma steuersystem-imp-maxime:
  ‹steuersystem steuersystem-impl ⟹
      (∀ 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:
  ‹∀ einkommen. steuersystem-impl einkommen ≤ einkommen ⟹
   ∀ einkommen. einkommen ≤ 9888 ⟶ steuersystem-impl einkommen = 0 ⟹
   ∀ welt. moralisch welt maxime-steuern
          (Handlungsabsicht (λp w. Some (jeder-zahlt steuersystem-impl p w)))
    ⟹ 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 Tugend 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 ⇒ nat*, mit zwei weiteren Annahmen gilt, dass *steuersystem* und *maxime-steuern* in der *jeder-zahlt* Implementierung äquivalent sind.

```
theorem
  fixes steuersystem-impl :: ‹nat ⇒ nat›
  assumes steuer-kleiner-einkommen: ‹∀ einkommen. steuersystem-impl einkommen ≤ einkommen›
      and existenzminimum: ‹∀ einkommen. einkommen ≤ 9888 ⟶ steuersystem-impl einkommen = 0›
    shows
     ‹(∀ welt. moralisch welt maxime-steuern
             (Handlungsabsicht (λp w. Some (jeder-zahlt steuersystem-impl p w))))
      ⟷
      steuersystem steuersystem-impl›
```

Da jede Steuersystemimplementierung welche *steuersystem* erfüllt auch moralisch ist (lemma *steuersystem-imp-maxime*), erfüllt damit auch jedes solche System den kategorischen Imperativ.

corollary *steuersystem-imp-kaptimp*:
‹*steuersystem steuersystem-impl* $\Longrightarrow$
  kategorischer-imperativ-auf
    (*Handlungsabsicht* ($\lambda p\ w$. *Some* (*jeder-zahlt steuersystem-impl p w*)))
    welt
    maxime-steuern›

Und daraus folgt, dass auch *jeder-zahlt-einkommenssteuer* den kategorischen Imperativ erfüllt.

corollary
‹*steuersystem steuersystem-impl* $\Longrightarrow$
  kategorischer-imperativ-auf
    (*Handlungsabsicht jeder-zahlt-einkommenssteuer*)
    welt
    maxime-steuern›

References

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