Mathematisches Institut
Heinrich-Heine Universität Düsseldorf
Dr. Alexander Bentkamp, Jon Eugster

WiSe 2023/24
5. Dezember 2023

Computer-gestützte Beweisführung

Übungsblatt 7

In allen Aufgaben meinen wir mit „beweisen“ die Erstellung formaler Beweisterme. Verwenden Sie keine Taktiken, die nicht in der Vorlesung oder im Skript eingeführt wurden. Zur besseren Klausurvorbereitung empfehlen wir, diese Übungen ohne Lean zu bearbeiten.

Aufgabe 1. Geben Sie eine Typderivation mit den Regeln aus Abschnitt 3.7 des Skripts für die folgenden Terme an:

(a) $\Pi X : \mathsf{Type}.\ \mathsf{False}$
(b) $\Pi X : \mathsf{Type}.\ X$
(c) $\Pi h : \mathsf{False}.\ \mathsf{Prop}$
(d) $\Pi h : \mathsf{False}.\ \mathsf{True}$

Lösung.

(a) Zunächst bemerken wir, dass Type für Sort_1 steht und dass False vom Typ Prop ist, was für Sort_0 steht. Außerdem ist $\operatorname{imax}(2,0) = 0$.

$$\dfrac{\dfrac{}{\vdash\ \mathsf{Sort}_1 : \mathsf{Sort}_2}\,\text{Sort} \qquad \dfrac{}{X : \mathsf{Sort}_1\ \vdash\ \mathsf{False} : \mathsf{Sort}_0}\,\text{Konst}}{\vdash\ \Pi X : \mathsf{Sort}_1.\ \mathsf{False} : \mathsf{Sort}_0}\,\text{Pi}$$

$\Pi X : \mathsf{Type}.\ \mathsf{False}$ ist also vom Typ Sort_0, also Prop.

(b) Wieder steht Type für Sort_1. Außerdem ist $\operatorname{imax}(2,1) = 2$.

$$\dfrac{\dfrac{}{\vdash\ \mathsf{Sort}_1 : \mathsf{Sort}_2}\,\text{Sort} \qquad \dfrac{}{X : \mathsf{Sort}_1\ \vdash\ X : \mathsf{Sort}_1}\,\text{Var}}{\vdash\ \Pi X : \mathsf{Sort}_1.\ X : \mathsf{Sort}_2}\,\text{Pi}$$

$\Pi X : \mathsf{Type}.\ X$ ist also vom Typ Sort_2, also Type_1.

(c) False ist vom Typ Prop und Prop steht für Sort_0. Außerdem ist $\operatorname{imax}(0,1) = 1$.

$$\dfrac{\dfrac{}{\vdash\ \mathsf{False} : \mathsf{Sort}_0}\,\text{Konst} \qquad \dfrac{}{h : \mathsf{False}\ \vdash\ \mathsf{Sort}_0 : \mathsf{Sort}_1}\,\text{Sort}}{\vdash\ \Pi h : \mathsf{False}.\ \mathsf{Sort}_0 : \mathsf{Sort}_1}\,\text{Pi}$$

$\Pi h : \mathsf{False}.\ \mathsf{Prop}$ ist also vom Typ Sort_1, also Type.

Bitte wenden!

(d) False und True sind vom Typ Prop, also Sort_0. Außerdem ist $\mathrm{imax}(0,0) = 0$.

$$\dfrac{\dfrac{}{\vdash\ \mathsf{False} : \mathsf{Sort}_0}\,\text{Konst} \qquad \dfrac{}{h : \mathsf{False} \vdash\ \mathsf{True} : \mathsf{Sort}_0}\,\text{Konst}}{\vdash\ \Pi h : \mathsf{False}.\ \mathsf{True} : \mathsf{Sort}_0}\,\text{Pi}$$

$\Pi h : \mathsf{False}.\ \mathsf{True}$ ist also vom Typ Sort_0, also Prop.

Aufgabe 2. Beweisen Sie das Gesetz von Peirce (gesprochen wie englisch „purse“)

$$\textbf{theorem}\ \ \mathsf{peirces\_law}\ (p\ q : \mathsf{Prop}) : ((p \to q) \to p) \to p := \ldots$$

mithilfe des Axioms des ausgeschlossenen Dritten

$$\textbf{axiom}\ \ \mathsf{em} : \forall p : \mathsf{Prop}.\ p \vee \neg p$$

Tipp: Machen Sie eine Fallunterscheidung auf $\mathsf{em}\ p$ mittels **match** $\mathsf{em}\ p$ **with**.

Lösung.

theorem $\mathsf{peirces\_law}\ (p\ q : \mathsf{Prop}) : ((p \to q) \to p) \to p :=$
match $\mathsf{em}\ p$ **with**
| $\mathsf{Or.inl}\ (hp : p) \Rightarrow \lambda(h : (p \to q) \to p).\ hp$
| $\mathsf{Or.inr}\ (hnp : \neg p) \Rightarrow \lambda(h : (p \to q) \to p).\ h\ (\lambda hp : p.\ \mathsf{False.elim}\ (hnp\ hp))$

Aufgabe 3. Russel's Barbier-Paradoxon lautet:

> Du kannst einen Barbier definieren als „jemanden, der all jene, und nur jene, rasiert, die sich nicht selbst rasieren“. Die Frage ist: Rasiert der Barbier sich selbst?

Beweisen Sie, dass ein solcher Barbier zu einem Widerspruch führt. Sei hierzu $\mathsf{Person} : \mathsf{Type}$ der Typ aller Personen und $\mathsf{barbier} : \mathsf{Person}$. Sei $\mathsf{rasiert} : \mathsf{Person} \to \mathsf{Person} \to \mathsf{Prop}$ eine Konstante, wobei „$\mathsf{rasiert}\ x\ y$“ bedeuten soll, dass Person x Person y rasiert.

theorem $\mathsf{barbier\_paradoxon}\ (h : \forall(x : \mathsf{Person}).\ \mathsf{rasiert\ barbier}\ x \leftrightarrow \neg\ \mathsf{rasiert}\ x\ x) :$
$\mathsf{False} := \ldots$

Tipp: Nutzen Sie das Axiom em. Wenden Sie es auf die Aussage $\mathsf{rasiert\ barbier\ barbier}$ an.

Lösung.

```
theorem barbier_paradoxon (h : ∀(x : Person). rasiert barbier x ↔ ¬ rasiert x x) :
  False :=
have h' : rasiert barbier barbier ↔ ¬ rasiert barbier barbier := h barbier
match em (rasiert barbier barbier) with
| Or.inl (hb : rasiert barbier barbier) ⇒
  have hnb : ¬ rasiert barbier barbier := Iff.mp h' hb
  show False from hnb hb
| Or.inr (hnb : ¬ rasiert barbier barbier) ⇒
  have hb : rasiert barbier barbier := Iff.mpr h' hnb
  show False from hnb hb
```

Oder ohne have/show-Syntax:

```
theorem barbier_paradoxon (h : ∀(x : Person). rasiert barbier x ↔ ¬ rasiert x x) :
  False :=
match em (rasiert barbier barbier) with
| Or.inl (hb : rasiert barbier barbier) ⇒ Iff.mp (h barbier) hb hb
| Or.inr (hnb : ¬ rasiert barbier barbier) ⇒ hnb (Iff.mpr (h barbier) hnb)
```

Aufgabe 4. Auf Übungsblatt 5 haben wir das induktive Prädikat Even definiert:

```
inductive Even : Nat → Prop :=
| base : Even 0
| step : ∀(n : Nat). Even n → Even (succ (succ n))
```

(a) Definieren Sie analog ein Prädikat Odd, das die ungeraden Zahlen $1, 3, 5, ...$ identifiziert. Verwenden Sie dazu nicht das Prädikat Even.

```
inductive Odd : Nat → Prop :=
| base : ???
| step : ???
```

(b) Beweisen Sie, dass Odd 5 gilt.

(c) Beweisen Sie das folgende Theorem. Schreiben Sie dabei stets Even.base / Odd.base / Even.step / Odd.step anstelle von base / step, um Verwechslungen zu vermeiden. Tipp: Induktion.

```
theorem odd_succ_of_even (n : Nat) : Even n → Odd (succ n)
| Even.base ⇒
  show Odd (succ 0) from ???
| Even.step (m : Nat) (hm : Even m) ⇒
  show Odd (succ (succ (succ m))) from ???
```

(d) Beweisen Sie das folgende Theorem.

```
theorem even_succ_of_odd (n : Nat) : Odd n → Even (succ n)
???
```

Lösung.

(a)

```
inductive Odd : Nat → Prop :=
| base : Odd 1
| step : ∀(n : Nat). Odd n → Odd (succ (succ n))
```

(b)

```
theorem odd_5 : Odd 5 :=
  Odd.step 3 (Odd.step 1 Odd.base)
```

(c)

```
theorem odd_succ_of_even (n : Nat) : Even n → Odd (succ n)
| Even.base ⇒
  show Odd (succ 0) from Odd.base
| Even.step (m : Nat) (hm : Even m) ⇒
  show Odd (succ (succ (succ m))) from
  Odd.step (succ m) (odd_succ_of_even m hm)
```

(d)

```
theorem even_succ_of_odd (n : Nat) : Odd n → Even (succ n)
| Odd.base ⇒
  show Even (succ 1) from Even.step 0 Even.base
| Odd.step (m : Nat) (hm : Odd m) ⇒
  show Even (succ (succ (succ m))) from
  Even.step (succ m) (even_succ_of_odd m hm)
```

Abgabe: 12. Dezember 2023, bis 16.30 Uhr auf Ilias
(Oder bei technischen Problemen per Email an `jon.eugster@hhu.de`)
Abgabe zu zweit ist erlaubt.