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

WiSe 2023/24
19. Dezember 2023

Computer-gestützte Beweisführung

Übungsblatt 9

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. Die folgende Funktion gibt die kleinere von zwei gegebenen natürlichen Zahlen zurück:

```
def min : Nat → Nat → Nat
| zero, zero ⇒ zero
| zero, succ y ⇒ zero
| succ x, zero ⇒ zero
| succ x, succ y ⇒ succ (min x y)
```

Beweisen Sie folgendes Theorem. Tipp: Machen Sie eine Fallunterscheidung auf x und auf y mit insgesamt vier Fällen.

```
theorem min_comm : ∀(x y : Nat). min x y = min y x := ...
```

Lösung.

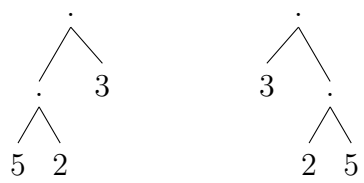
```
theorem min_comm : ∀(x y : Nat). min x y = min y x
| zero, zero ⇒ show min zero zero = min zero zero from rfl
| succ u, zero ⇒ show min (succ u) zero = min zero (succ u) from rfl
| zero, succ v ⇒ show min zero (succ v) = min (succ v) zero from rfl
| succ u, succ v ⇒
  calc min (succ u) (succ v)
  _ = succ (min u v) := rfl
  _ = succ (min v u) := by rw [min_comm u v]
  _ = min (succ v) (succ u) := rfl
```


Bitte wenden!

Aufgabe 2. Der folgende induktive Typ repräsentiert binäre Bäume mit natürlichen Zahlen an seinen Blättern:

```
inductive BTree :=
| leaf : Nat → BTree
| branch : BTree → BTree → BTree
```

So repräsentieren zum Beispiel die Elemente branch (branch (leaf 5) (leaf 2)) (leaf 3) und branch (leaf 3) (branch (leaf 2) (leaf 5)) die folgenden Bäume:



Die folgende Funktion spiegelt einen Baum an der vertikalen Achse. Zum Beispiel wird der linke der beiden obigen Bäume wird durch die Funktion zu dem rechten verarbeitet und der rechte zum linken.

```
def mirror : BTree → BTree
| leaf n ⇒ leaf n
| branch x y ⇒ branch (mirror y) (mirror x)
```

Beweisen Sie folgendes Theorem.

```
theorem mirror_mirror : ∀(z : BTree), mirror (mirror z) = z := ...
```

Lösung.

```
theorem mirror_mirror : ∀(z : BTree), mirror (mirror z) = z
| leaf n => show mirror (mirror (leaf n)) = leaf n from rfl
| branch x y =>
  calc mirror (mirror (branch x y))
  _ = mirror (branch (mirror y) (mirror x)) := rfl
  _ = branch (mirror (mirror x)) (mirror (mirror y)) := rfl
  _ = branch x (mirror (mirror y)) := by rw [mirror_mirror x]
  _ = branch x y := by rw [mirror_mirror y]
```

Aufgabe 3. Wir wollen zeigen, dass folgende Aussagen äquivalent sind, wenn wir das Axiom em des ausgeschlossenen Dritten nicht verwenden:

- Satz des ausgeschlossenen Dritten: $\forall p.\ p \vee \neg p$
- Eliminierung doppelter Negation: $\forall q.\ \neg\neg q \to q$

(a) Zunächst nur die eine Richtung:

theorem dn_of_em $(h : \forall(p : \mathsf{Prop}).\ p \vee \neg p) : \forall(q : \mathsf{Prop}).\ \neg\neg q \to q := ...$

(b) Die andere Richtung ist trickreich und wir geben daher den Beweis vor. Ergänzen Sie die vier Aussagen anstelle der „???“ in den show-Befehlen:

```
theorem em_of_dn (h : ∀(q : Prop). ¬¬q → q) : ∀(p : Prop). p ∨ ¬p :=
  λ (p : Prop). h (p ∨ ¬p) (
    show ??? from
    λ(hnpp : ¬(p ∨ ¬p)).
      show ??? from
      hnpp (
        show ??? from
        Or.inr (λhp : p.
          show ??? from
           hnpp (Or.inl hp)
        )
      )
  )
```

(c) Verwenden Sie die Theoreme aus den vorherigen beiden Teilaufgaben, um die Äquivalenz zu zeigen:

theorem dn_iff_em : $(\forall p : \mathsf{Prop}.\ p \vee \neg p) \leftrightarrow (\forall q : \mathsf{Prop}.\ \neg\neg q \to q) := ...$

Lösung.

(a)

```
theorem dn_of_em (h : ∀(p : Prop). p ∨ ¬p) : ∀(q : Prop). ¬¬q → q :=
  λ (q : Prop) (hnnq : ¬¬q).
    show q from
    match h q with
    | Or.inl (hq : q) ⇒ hq
    | Or.inr (hnq : ¬q) ⇒
      have hf : False := hnnq hnq
      False.elim hf
```

(b)

```
theorem em_of_dn (h : ∀(q : Prop). ¬¬q → q) : ∀(p : Prop). p ∨ ¬p :=
  λ (p : Prop). h (p ∨ ¬p) (
    show ¬¬(p ∨ ¬p) from
    λ(hnpp : ¬(p ∨ ¬p)).
      show False from
      hnpp (
        show p ∨ ¬p from
        Or.inr (λhp : p.
          show False from
          hnpp (Or.inl hp)
        )
      )
  )
```

(c)

```
theorem em_iff_dn : (∀p : Prop. p ∨ ¬p) ↔ (∀q : Prop. ¬¬q → q) :=
  Iff.intro dn_of_em em_of_dn
```

Aufgabe 4. Die folgende Funktion gibt die Summe einer Liste von natürlichen Zahlen zurück:

```
def sum : List Nat → Nat
| nil ⇒ 0
| cons x xs ⇒ x + sum xs
```

Definieren Sie diese Funktion mithilfe des Rekursors von List statt Pattern-Matching. Geben Sie auch die impliziten Argumente des Rekursors an.

```
List.rec.{u} : Π{X : Type}. Π{motive : List X → Sort_u} →
  motive nil →
  (Π(x : X) (xs : List X). motive xs → motive (cons x xs)) →
  (t : List X) → motive t
```

Lösung.

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{sum} : \mathsf{List\ Nat} \to \mathsf{Nat} := \\
&\lambda l.\ @\mathsf{List.rec\ Nat}\ (\lambda \_.\ \mathsf{Nat}) \\
&\qquad 0 \\
&\qquad (\lambda (x : A)\ (xs : \mathsf{List}\ A)\ (s : \mathsf{Nat}).\ x + s) \\
&\qquad l
\end{aligned}
$$

Abgabe: 12. Januar 2024, bis 16.30 Uhr auf Ilias
(Oder bei technischen Problemen per Email an `jon.eugster@hhu.de`)
Abgabe zu zweit ist erlaubt.