

Computer-gestützte Beweisführung Übungsblatt 5

In allen Aufgaben meinen wir mit „beweisen“ die Erstellung formaler Beweisterme im Sinne von Abschnitt 3.5 des Skripts. Zur besseren Klausurvorbereitung empfehlen wir, diese Übungen ohne Lean zu bearbeiten.

Aufgabe 1. Beweisen Sie:

(a)

theorem true_iff_not_false : True \leftrightarrow \neg False := ...

Tipp: Nutzen Sie `Iff.intro` und `True.intro`, die in der Vorlesung eingeführt wurden. Beachten Sie, dass $\neg s$ definitionell gleich $s \rightarrow$ `False` ist: Jede Funktion von s nach `False` ist also ein Element von $\neg s$.

(b)

theorem and_false (p : Prop) : (p \wedge False) \leftrightarrow False := ...

Tipp: Hier könnten `Iff.intro`, `And.right` und `False.elim` helfen. Beachten Sie, dass Sie mit einem Beweis $h : \text{False}$ einen Beweis `False.elim h : q` (genauer `@False.elim q h : q`) einer beliebigen Aussage q konstruieren können.

Aufgabe 2. Beweisen Sie:

(a)

theorem forall_implies (p q : Nat \rightarrow Prop) :

($\forall x. p\ x \rightarrow q\ x$) \rightarrow ($\forall x. p\ x$) \rightarrow ($\forall x. q\ x$) := ...

Tipp: Am Typ von p und q können wir erkennen, dass alle Variablen x in dieser Aufgabe vom Typ `Nat` sein sollen. Beachten Sie, dass \forall nur eine alternative Schreibweise für Π ist: Ein Element von $\forall x : \text{Nat}. s$ ist also eine Funktion von `Nat` nach s .

(b)

theorem not_exists_not_of_forall (p : Nat \rightarrow Prop) :

($\forall x. p\ x$) \rightarrow $\neg(\exists x. \neg p\ x)$:= ...

Tipp: Nutzen Sie `Exists.elim`.

Aufgabe 3. Wir definieren ein induktives Prädikat `Even` von geraden Zahlen. Es sollen also `Even 0`, `Even 2`, `Even 4`, ... beweisbar sein, aber nicht `Even 1`, `Even 3`, `Even 5`, ...

(a) Ergänzen Sie den fehlenden Konstruktor „`???`“:

```

inductive Even : Nat → Prop :=
  | ???
  | step : ∀(n : Nat). Even n → Even (succ (succ n))

```

(b) Beweisen Sie, dass 4 gerade ist:

```

theorem even_four : Even (succ (succ (succ (succ zero)))) := ...

```

Aufgabe 4. Auf einer Insel sind alle Einwohner entweder Ritter, die immer die Wahrheit sagen, oder Knappen, die immer lügen. Ein Besucher trifft die Einwohner Martin und Klaus.

Klaus sagt: „Martin ist ein Knappe.“

Martin sagt: „Klaus und ich sind Ritter.“

Wir wollen formal beweisen, dass Martin ein Knappe ist.

Sei dazu `Einwohner : Type` ein Typ der Einwohner der Insel und seien `klaus : Einwohner` und `martin : Einwohner`. Sei `ritter : Einwohner → Prop` eine Funktion, die angibt, ob ein Einwohner ein Ritter ist. Da jeder Einwohner entweder Ritter oder Knappe ist, können wir zur Vereinfachung „ \neg `ritter P`“ für „P ist ein Knappe“ schreiben. Einen Hinweis „P sagt: X“ können wir kodieren als: „`ritter P ↔ X`“.

Formal können wir diese Aufgabe also wie folgt fassen. Ergänzen Sie den Beweis.

```

theorem raetsel1
  (h1 : ritter klaus ↔ ¬ritter martin)
  (h2 : ritter martin ↔ (ritter martin ∧ ritter klaus)) :
  ¬ ritter martin := ...

```

Hierbei kodiert h_1 den Hinweis „Klaus sagt: ‚Martin ist ein Knappe.‘“ und h_2 den Hinweis „Martin sagt: ‚Klaus und ich sind Ritter.‘“

Abgabe: 28. November 2023, bis 16.30 Uhr auf Ilias
 (Oder bei technischen Problemen per Email an jon.eugster@hhu.de)
 Abgabe zu zweit ist erlaubt.