Computer-gestützte Beweisführung

Klausur I

Heinrich-Heine-Universität Düsseldorf
Wintersemester 2023/24

- **Die Klausur dauert 90 Minuten. Sie besteht aus 8 Aufgaben. Pro Aufgabe sind 10 Punkte zu erwerben.**
- Öffnen Sie die Klausur erst, wenn der Klausurbeginn angesagt wurde!
- Es sind keine Hilfsmittel (Taschenrechner, Mitschriften, Notizen, Skript, Handy, etc.) außer mechanischen Armbanduhren und Weckern zugelassen.
- Bitte prüfen Sie die folgenden Angaben und korrigieren Sie sie gegebenenfalls:

  Name:
  Matrikelnr.:

  Bitte legen Sie Ihren Studierendenausweis und Ihren Lichtbildausweis vor sich auf das Pult, damit Ihre Identität während der Klausur geprüft werden kann.
- Bearbeiten Sie die Aufgaben jeweils auf den dafür vorgesehenen Seiten. Wenn Sie zusätzliche Blätter benötigen, geben Sie uns bitte ein Zeichen. Schreiben Sie auf zusätzliche Blätter Ihren Namen und welche Aufgabe Sie bearbeiten.
- Sie können einzelne Klausurbögen mit „bitte nicht werten“ kennzeichnen, aber nicht mitnehmen. Die Klausuraufgaben werden wir veröffentlichen.
- In allen Aufgaben meinen wir mit „definieren“ und „beweisen“ die Erstellung formaler Definitionen und Beweisterme im Calculus of Inductive Constructions. Beschränken Sie sich auf die Taktiken, die in der Vorlesung eingeführt wurden.

Viel Erfolg!

Aufgabe	1	2	3	4	5	6	7	8	$\sum$	
Punkte										/80

Spickzettel Teil I: Typisierungsregeln

Einfach getypter λ-Kalkül.

$$\text{Konst}\ \frac{}{\Gamma\ \vdash\ \mathsf{c} : T} \quad \text{wenn } \mathsf{c} \text{ eine Konstante von Typ } T \text{ ist}$$

$$\text{Var}\ \frac{}{\Gamma\ \vdash\ x : \Gamma(x)}$$

$$\text{Anw}\ \frac{\Gamma\ \vdash\ t : S \to T \quad \Gamma\ \vdash\ s : S}{\Gamma\ \vdash\ t\ s : T}$$

$$\text{Lam}\ \frac{\Gamma,\ x : S\ \vdash\ t : T}{\Gamma\ \vdash\ (\lambda x : S.\ t) : S \to T}$$

Calculus of Inductive Constructions.

$$\text{Konst}\ \frac{}{\Gamma\ \vdash\ \mathsf{c} : T} \quad \text{wenn } \mathsf{c} \text{ eine Konstante von Typ } T \text{ ist}$$

$$\text{Var}\ \frac{}{\Gamma\ \vdash\ x : \Gamma(x)}$$

$$\text{Anw}\ \frac{\Gamma\ \vdash\ t : \Pi x : S.\ T \qquad \Gamma\ \vdash\ s : S}{\Gamma\ \vdash\ t\ s : T[x := s]}$$

$$\text{Lam}\ \frac{\Gamma,\ x : S\ \vdash\ t : T \qquad \Gamma\ \vdash\ S : \mathsf{Sort}_u}{\Gamma\ \vdash\ (\lambda x : S.\ t) : (\Pi x : S.\ T)}$$

$$\text{Sort}\ \frac{}{\Gamma\ \vdash\ \mathsf{Sort}_u : \mathsf{Sort}_{u+1}}$$

$$\text{Pi}\ \frac{\Gamma\ \vdash\ S : \mathsf{Sort}_u \qquad \Gamma,\ x : S\ \vdash\ T : \mathsf{Sort}_v}{\Gamma\ \vdash\ \Pi x : S.\ T : \mathsf{Sort}_{\mathrm{imax}(u,v)}}$$

$$\text{DefGl}\ \frac{\Gamma\ \vdash\ t : T}{\Gamma\ \vdash\ t : S} \text{ wenn } T \equiv S$$

wobei

$$\mathrm{imax}(u, v) = \begin{cases} 0 & \text{wenn v} = 0 \\ \max(u, v) & \text{sonst} \end{cases}$$

Spickzettel Teil II: Wichtige Definitionen

Logische Prädikate.

inductive And $(p\ q : \mathsf{Prop}) : \mathsf{Prop} :=$
| And.intro : $p \to q \to \mathsf{And}\ p\ q$

inductive Or $(p\ q : \mathsf{Prop}) :=$
| Or.inl : $p \to \mathsf{Or}\ p\ q$
| Or.inr : $q \to \mathsf{Or}\ p\ q$

inductive Iff $(p\ q : \mathsf{Prop}) : \mathsf{Prop} :=$
| Iff.intro : $(p \to q) \to (q \to p) \to \mathsf{Iff}\ p\ q$

inductive True : Prop :=
| True.intro : True

inductive False : Prop :=
[Induktiver Typ ohne Konstruktoren]

def Not $(p : \mathsf{Prop}) : \mathsf{Prop} := p \to \mathsf{False}$

inductive Exists.$\{u\}$ $\{X : \mathsf{Type}_u\}$ $(p : X \to \mathsf{Prop}) : \mathsf{Prop} :=$
| Exists.intro : $\forall x : X.\ p\ x \to \mathsf{Exists}\ p$

inductive Eq.$\{u\}$ $\{X : \mathsf{Type}_u\}$ $(x : X) : X \to \mathsf{Prop} :=$
| rfl : Eq $x\ x$

Eliminationsregeln.

And.left : $\Pi\{p\ q : \mathsf{Prop}\}.\ \mathsf{And}\ p\ q \to p$ And.right : $\Pi\{p\ q : \mathsf{Prop}\}.\ \mathsf{And}\ p\ q \to p$

Or.elim : $\Pi\{p\ q\ r : \mathsf{Prop}\}.\ \mathsf{Or}\ p\ q \to (p \to r) \to (q \to r) \to r$

False.elim : $\Pi\{p : \mathsf{Prop}\}.\ \mathsf{False} \to p$

Iff.mp : $\forall\{p\ q : \mathsf{Prop}\}.\ \mathsf{Iff}\ p\ q \to (p \to q)$ Iff.mpr : $\forall\{p\ q : \mathsf{Prop}\}.\ \mathsf{Iff}\ p\ q \to (q \to p)$

Exists.elim.$\{u\}$: $\Pi\{X : \mathsf{Type}_u\}\ \{p : X \to \mathsf{Prop}\}\ \{q : \mathsf{Prop}\}.$
Exists $p \to (\forall x : X.\ p\ x \to q) \to q$

Eq.subst : $\Pi\{X : \mathsf{Type}\}\ \{x\ y : X\}\ \{p : X \to \mathsf{Prop}\}.\ \mathsf{Eq}\ x\ y \to p\ x \to p\ y$

Induktive Typen.

inductive Bool : Type :=
| false : Bool
| true : Bool

inductive Nat : Type :=
| zero : Nat
| succ : Nat → Nat

inductive List $(X : \mathsf{Type}) : \mathsf{Type} :=$
| nil : List X
| cons : $X \to \mathsf{List}\ X \to \mathsf{List}\ X$

Aufgabe 1 (Einfach getypter λ-Kalkül)**.** Sind die folgenden Terme wohlgetypt im einfach getypten λ-Kalkül? Wenn nein, begründen Sie kurz. Wenn ja, geben Sie den Typ des Terms an und ergänzen Sie den Typ jeder durch λ gebundenen Variable. (Sie brauchen hier keine Typderivation aufschreiben.) Es seien A und B Basistypen und $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, $\mathsf{g} : \mathsf{B} \to \mathsf{A}$ und $\mathsf{a} : \mathsf{A}$ seien Konstanten.

(a) $(\lambda x.\ \mathsf{g}\ (\mathsf{g}\ x))\ (\mathsf{f}\ \mathsf{a})$
(b) $(\lambda y\ x.\ y\ x)\ \mathsf{g}$

Lösung.

(a) Nein, denn g erwartet ein Argument vom Typ B und $\mathsf{g}\ x$ kann nur Typ A haben.
(b) $((\lambda(y : \mathsf{B} \to \mathsf{A})\ (x : \mathsf{B}).\ y\ x)\ \mathsf{g}) : \mathsf{B} \to \mathsf{A}$

Aufgabe 2 (Typderivation)**.** Erstellen Sie eine Typderivation für $\lambda x : \mathsf{Prop}.\ \mathsf{Not}\ x$ im Calculus of Inductive Constructions, wobei $\mathsf{Not} : \mathsf{Prop} \to \mathsf{Prop}$ eine Konstante ist. Der Spickzettel am Anfang der Klausur kann hier hilfreich sein.

Lösung. Der Ausdruck Prop steht für Sort_0 und der Ausdruck $\mathsf{Prop} \to \mathsf{Prop}$ steht für $\Pi x : \mathsf{Sort}_0.\ \mathsf{Sort}_0$. Somit:

$$\dfrac{\dfrac{\dfrac{}{x : \mathsf{Sort}_0 \vdash\ \mathsf{Not} : (\Pi x : \mathsf{Sort}_0.\ \mathsf{Sort}_0)}\,\text{Konst} \quad \dfrac{}{x : \mathsf{Sort}_0 \vdash\ x : \mathsf{Sort}_0}\,\text{Var}}{x : \mathsf{Sort}_0 \vdash\ \mathsf{Not}\ x : \mathsf{Sort}_0}\,\text{Anw} \quad \dfrac{}{\vdash\ \mathsf{Sort}_0 : \mathsf{Sort}_1}\,\text{Sort}}{\vdash\ (\lambda x : \mathsf{Sort}_0.\ \mathsf{Not}\ x) : (\Pi x : \mathsf{Sort}_0.\ \mathsf{Sort}_0)}\,\text{Lam}$$

Aufgabe 3 (Induktive Typen und Pattern-Matching)**.**

(a) Definieren Sie eine „kleiner-gleich“ Funktion $\mathsf{kg} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Bool}$, die zu zwei natürlichen Zahlen genau dann true zurückgibt, wenn die erste Zahl kleiner als oder gleich der zweiten ist. (Achtung: Der Rückgabewert soll Bool sein, nicht Prop!)

(b) Definieren Sie eine Funktion $\mathsf{sortiert} : \mathsf{List\ Nat} \to \mathsf{Bool}$, die genau dann ‘true‘ zurückgibt, wenn eine gegebene Liste von natürlichen Zahlen aufsteigend sortiert ist. Die Funktion $\mathsf{sortiert}$ soll also für $[1, 1, 33, 230]$ true zurückgeben, aber für $[4, 2, 7]$ false.

Als gegeben können Sie dabei annehmen und verwenden:

- die Funktion kg aus Teil (a)
- eine Funktion $\mathsf{and} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$, die genau dann true zurückgibt, wenn ihre beiden Argumente true sind.

Tipp: Unterscheiden Sie drei Fälle: die leere Liste, eine Liste mit einem Element, und eine Liste mit mindestens zwei Elementen.

Lösung.

(a)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{kg} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Bool} \\
&\mid \mathsf{zero},\ \mathsf{zero} \Rightarrow \mathsf{true} \\
&\mid \mathsf{zero},\ \mathsf{succ}\ y \Rightarrow \mathsf{true} \\
&\mid \mathsf{succ}\ x,\ \mathsf{zero} \Rightarrow \mathsf{false} \\
&\mid \mathsf{succ}\ x,\ \mathsf{succ}\ y \Rightarrow \mathsf{kg}\ x\ y
\end{aligned}
$$

(b)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{sortiert} : \mathsf{List\ Nat} \to \mathsf{Bool} \\
&\mid \mathsf{nil} \Rightarrow \mathsf{true} \\
&\mid \mathsf{cons}\ x\ \mathsf{nil} \Rightarrow \mathsf{true} \\
&\mid \mathsf{cons}\ x\ (\mathsf{cons}\ y\ zs) \Rightarrow \mathsf{and}\ (\mathsf{kg}\ x\ y)\ (\mathsf{sortiert}\ (\mathsf{cons}\ y\ zs))
\end{aligned}
$$

Aufgabe 4 (Grundlegende Beweise)**.** Beweisen Sie folgende Theoreme. Der Spickzettel am Anfang der Klausur kann hier hilfreich sein.

(a)

```
theorem false_iff_not_true : False ↔ ¬True := ...
```

(b)

```
theorem not_implies_of_and_not (p q : Prop) :
  (p ∧ ¬q) → ¬(p → q) := ...
```

Lösung.

(a)

```
theorem false_iff_not_true : False ↔ ¬True :=
Iff.intro
  (λ(hf : False) (ht : True). hf)
  (λ(hnt : ¬True). hnt True.intro)
```

(b)

```
theorem not_implies_of_and_not (p q : Prop) :
  (p ∧ ¬q) → ¬(p → q) :=
λ (h : p ∧ ¬q) (h1 : p → q). And.right h (h1 (And.left h))
```

Aufgabe 5 (Induktive Prädikate)**.** Gegeben sei das folgende induktive Prädikat, das $\leq$ auf den natürlichen Zahlen repräsentiert:

```
inductive LE : Nat → Nat → Prop
| refl : ∀(n : Nat). LE n n
| step : ∀{n m : Nat}. LE n m → LE n (succ m)
```

(a) Beweisen Sie, dass $0 \leq 2$:

```
theorem zero_le_two : LE zero (succ (succ zero)) := ...
```

(b) Beweisen Sie, dass $0 \leq n$ für jedes n gilt:

```
theorem zero_le (n : Nat) : LE zero n := ...
```

Lösung.

(a)

```
theorem zero_le_two : LE zero (succ (succ zero)) :=
  LE.step (LE.step (LE.refl zero))
```

(b)

```
theorem zero_le (n : Nat) : LE zero n :=
  match n with
  | zero ⇒
    show LE zero zero from
    LE.refl zero
  | succ m ⇒
    show LE zero (succ m) from
    LE.step (zero_le m)
```

Aufgabe 6 (Gleichungen)**.** Gegeben seien die folgenden Funktionen pred, die eine natürliche Zahl auf ihren Vorgänger abbildet (und 0 auf 0), und sub, die zwei natürliche Zahlen subtrahiert (oder 0 zurückgibt, wenn die erste Zahl kleiner ist als die zweite):

```
def pred : Nat → Nat
| zero ⇒ zero
| succ x ⇒ x

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

Beweisen Sie, dass sub immer 0 zurückgibt, wenn das erste Argument 0 ist:

```
theorem zero_sub (y : Nat) : sub zero y = zero := ...
```

Tipp: Verwenden Sie die Definitionen von sub und pred und gebrauchen Sie Induktion.

Lösung.

```
theorem zero_sub (y : Nat) : sub zero y = zero :=
  match y with
  | zero ⇒ rfl
  | succ y ⇒
    calc sub zero (succ y)
    _ = pred (sub zero y)    := rfl
    _ = pred zero    := by rw [zero_sub y]
    _ = zero    := rfl
```

Aufgabe 7 (Rekursoren)**.** Definieren Sie die folgenden Funktionen mithilfe von Rekursoren statt Pattern-Matching. Geben Sie auch die impliziten Argumente der Rekursoren an.

(a)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{or} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}\\
&\mid \mathsf{false}, \mathsf{false} \Rightarrow \mathsf{false}\\
&\mid \mathsf{true}, \mathsf{false} \Rightarrow \mathsf{true}\\
&\mid \mathsf{false}, \mathsf{true} \Rightarrow \mathsf{true}\\
&\mid \mathsf{true}, \mathsf{true} \Rightarrow \mathsf{true}
\end{aligned}
$$

Verwenden Sie hier den Rekursor von Bool:

$$
\begin{aligned}
&\mathsf{Bool.rec}.\{u\} : \Pi\{\mathit{motive} : \mathsf{Bool} \to \mathsf{Sort}_u\}.\\
&\quad \mathit{motive}\ \mathsf{false} \to \mathit{motive}\ \mathsf{true} \to \Pi t : \mathsf{Bool}.\ \mathit{motive}\ t
\end{aligned}
$$

(b)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{fak} : \mathsf{Nat} \to \mathsf{Nat}\\
&\mid \mathsf{zero} \Rightarrow 1\\
&\mid \mathsf{succ}\ n \Rightarrow \mathsf{fak}\ n * (\mathsf{succ}\ n)
\end{aligned}
$$

Verwenden Sie hier den Rekursor von Nat:

$$
\begin{aligned}
&\mathsf{Nat.rec}.\{u\} : \Pi\{\mathit{motive} : \mathsf{Nat} \to \mathsf{Sort}_u\}.\\
&\quad \mathit{motive}\ \mathsf{zero} \to\\
&\quad (\Pi n : \mathsf{Nat}.\ \mathit{motive}\ n \to \mathit{motive}\ (\mathsf{succ}\ n)) \to\\
&\quad \Pi t : \mathsf{Nat}.\ \mathit{motive}\ t
\end{aligned}
$$

Lösung.

(a)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{or}\ (x\ y : \mathsf{Bool}) : \mathsf{Bool} :=\\
&\quad @\mathsf{Bool.rec}\ (\lambda\_.\ \mathsf{Bool})\ y\ \mathsf{true}\ x
\end{aligned}
$$

(b)

$$
\begin{aligned}
&\textbf{def}\ \ \mathsf{fak}\ (x : \mathsf{Nat}) : \mathsf{Nat} :=\\
&\quad @\mathsf{Nat.rec}\ (\lambda\_.\ \mathsf{Nat})\ 1\ (\lambda n\ r.\ r * (\mathsf{succ}\ n))\ x
\end{aligned}
$$

Aufgabe 8 (Anwendungen)**.**

(a) Der folgende Compiler, den wir in der Vorlesung verifiziert haben, setzt arithmetische Ausdrücke in Instruktionen für eine Stapelmaschine um:

def comp : AExp → List Instr

| num n ⇒ [LOADI n]

| var x ⇒ [LOAD x]

| plus a b ⇒ comp a ++ comp b ++ [ADD]

Für welchen arithmetischen Ausdruck liefert der Compiler die folgenden Instruktionen?

[LOADI 1, LOADI 2, LOAD "x", ADD, ADD]

(b) Geben Sie einen Syntaxbaum für den Satz „Sofia lacht“ und eine Schritt-für-Schritt-Berechnung von ⟦Sofia lacht⟧ an (in β-reduzierter Form). Wenn nicht eindeutig, geben Sie an, welche syntaktische Struktur gemeint ist, z.B. ⟦Sofia (EN)⟧ statt ⟦Sofia⟧ . Verwenden Sie dabei die folgenden Regeln:

EN ⟶ Sofia	⟦**EN**⟧ := (sofia : Entität)
IV ⟶ lacht	⟦**IV**⟧ := (lacht : Entität → Prop)
NP ⟶ **EN**	⟦**NP**⟧ := λv : Entität → Prop. v ⟦**EN**⟧
VP ⟶ **IV**	⟦**VP**⟧ := ⟦**IV**⟧
S ⟶ **NP VP**	⟦**S**⟧ := ⟦**NP**⟧ ⟦**VP**⟧

(c) Teilbarkeit ist definiert als:

def teilt (a b : Nat) : Prop := $\exists x$: Nat. $a * x = b$

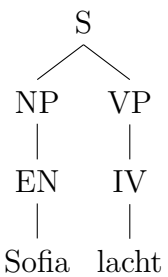
Beweisen Sie:

theorem kein_vielfaches (a b c : Nat) (hab : ¬ teilt a b) : ¬ ($a * c = b$) := ...

Lösung.

(a) 1 + (2 + x)

(b)



$$
\begin{aligned}
[\![\text{Sofia lacht}]\!] &= [\![\text{Sofia (NP)}]\!]\ [\![\text{lacht (VP)}]\!] \\
&= (\lambda v : \mathsf{Entität} \to \mathsf{Prop}.\ v\ [\![\text{Sofia (EN)}]\!])\ [\![\text{lacht (IV)}]\!] \\
&= (\lambda v : \mathsf{Entität} \to \mathsf{Prop}.\ v\ \mathsf{sofia})\ \mathsf{lacht} \\
&\equiv \mathsf{lacht}\ \mathsf{sofia}
\end{aligned}
$$

(c)

theorem kein_vielfaches $(a\ b\ c : \mathsf{Nat})\ (hab : \neg\ \mathsf{teilt}\ a\ b) : \neg\ (a * c = b) :=$
$\lambda h.\ hab\ (\mathsf{Exists.intro}\ c\ h)$