Max Mustermann

Computer-gestützte Beweisführung

Probeklausur

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: **Max Mustermann**
  Matrikelnr.: **123456789**

  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

Max Mustermann

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

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

Max Mustermann

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 (Ungetypter λ-Kalkül)**.** Die Church-Booleans sind eine Kodierung der Booleans *true* und *false*. Wir bezeichnen im Folgenden den λ-Term $\lambda x\ y.\ x$ als *true* und den λ-Term $\lambda x\ y.\ y$ als *false*. Geben Sie einen λ-Term an, der „oder“ kodiert. Gesucht ist also ein λ-Term t, sodass

$$t\ \mathit{true}\ \mathit{true} \equiv \mathit{true}$$
$$t\ \mathit{true}\ \mathit{false} \equiv \mathit{true}$$
$$t\ \mathit{false}\ \mathit{true} \equiv \mathit{true}$$
$$t\ \mathit{false}\ \mathit{false} \equiv \mathit{false}$$

Lösung.

$$t = \lambda u\ v\ x\ y.\ u\ x\ (v\ x\ y)$$

oder

$$t = \lambda u\ v.\ u\ \mathit{true}\ v$$

Aufgabe 2 (Typderivation)**.** Erstellen Sie eine Typderivation für $\Pi X : \mathsf{Type}.\ \mathsf{List}\ X$ im Calculus of Inductive Constructions, wobei $\mathsf{List} : \mathsf{Type} \to \mathsf{Type}$ eine Konstante ist. Der Spickzettel am Anfang der Klausur kann hier hilfreich sein.

Lösung. Der Ausdruck $\Pi X : \mathsf{Type}.\ \mathsf{List}\ X$ steht für $\Pi X : \mathsf{Sort}_1.\ \mathsf{List}\ X$ und $\mathsf{Type} \to \mathsf{Type}$ steht für $\Pi Y : \mathsf{Sort}_1.\ \mathsf{Sort}_1$. Somit:

$$\dfrac{\dfrac{}{\vdash\ \mathsf{Sort}_1 : \mathsf{Sort}_2}\,\text{Sort} \qquad \dfrac{\dfrac{}{X : \mathsf{Sort}_1\ \vdash\ \mathsf{List} : \Pi Y : \mathsf{Sort}_1.\ \mathsf{Sort}_1}\,\text{Konst} \qquad \dfrac{}{X : \mathsf{Sort}_1\ \vdash\ X : \mathsf{Sort}_1}\,\text{Var}}{X : \mathsf{Sort}_1\ \vdash\ \mathsf{List}\ X : \mathsf{Sort}_1}\,\text{Anw}}{\vdash\ (\Pi X : \mathsf{Sort}_1.\ \mathsf{List}\ X) : \mathsf{Sort}_2}\,\text{Pi}$$

Denn $\mathrm{imax}(2, 1) = 2$.

Aufgabe 3 (Polymorphismus)**.** Definieren Sie eine polymorphe Funktion diagonale, sodass für $\mathsf{A} : \mathsf{Type}$, $\mathsf{B} : \mathsf{Type}$, $\mathsf{a} : \mathsf{A}$, $\mathsf{b} : \mathsf{B}$, $\mathsf{f} : \mathsf{A} \to \mathsf{A} \to \mathsf{B}$, $\mathsf{g} : \mathsf{B} \to \mathsf{B} \to \mathsf{A}$ gilt:

$$\mathsf{diagonale\ f\ a} \equiv \mathsf{f\ a\ a} \quad \text{und} \quad \mathsf{diagonale\ g\ b} \equiv \mathsf{g\ b\ b}$$

Hierbei soll die Funktion diagonale zwei implizite Argumente haben, die den Typ des zweiten Arguments und den Rückgabetyp festlegen.

Lösung.

$$\textbf{def}\ \ \mathsf{diagonale}\ \{X\ Y : \mathsf{Type}\}\ (z : X \to X \to Y)\ (x : X) : Y := z\ x\ x$$

Aufgabe 4 (Induktive Typen und Pattern-Matching)**.**

(a) Definieren Sie eine Funktion $\mathsf{or} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$, die genau dann true zurückgibt, wenn mindestens eines der Argumente true ist.

(b) Gegeben sei die Konstante pred, die eine natürliche Zahl auf ihren Vorgänger abbildet (und 0 auf 0):

$$\begin{aligned}&\textbf{def}\ \ \mathsf{pred} : \mathsf{Nat} \to \mathsf{Nat}\\ &\mid 0 \Rightarrow 0\\ &\mid \mathsf{succ}\ x \Rightarrow x\end{aligned}$$

Definieren Sie eine Funktion sub, die für zwei natürliche Zahlen x und y die Differenz $x - y$ zurückgibt. Falls diese Differenz negativ ist, wird 0 zurückgegeben. (Tipp: Verwenden Sie Rekursion auf dem zweiten Argument)

Lösung.

(a)

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

(b)

$$\begin{aligned}&\textbf{def}\ \ \mathsf{sub} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}\\ &\mid x, 0 \Rightarrow x\\ &\mid x, \mathsf{succ}\ y \Rightarrow \mathsf{pred}\ (\mathsf{sub}\ x\ y)\end{aligned}$$

Aufgabe 5 (Grundlegende Beweise)**.** Beweisen Sie folgende Theoreme. Der Spickzettel am Anfang der Klausur kann hier hilfreich sein.

(a)

theorem forall_swap $(p : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Prop})$ $(h : \forall x\ y.\ p\ x\ y) :$
$\forall y\ x.\ p\ x\ y := ...$

(b)

theorem by_cases $(p\ q : \mathsf{Prop}) : (p \to q) \to (\neg p \to q) \to q := ...$

Tipp: Verwenden Sie hier das Axiom des ausgeschlossenen Dritten $\mathsf{em} : \forall p : \mathsf{Prop}.\ p \vee \neg p$ und machen Sie eine Fallunterscheidung auf $\mathsf{em}\ p$ mittels **match** em p **with**.

Lösung.

(a)

theorem forall_swap $(p : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Prop})$ $(h : \forall x\ y.\ p\ x\ y) :$
$\forall y\ x.\ p\ x\ y := \lambda y\ x.\ h\ x\ y$

(b)

theorem by_cases $(p\ q : \mathsf{Prop}) : (p \to q) \to (\neg p \to q) \to q :=$
match em p **with**
| Or.inl $(hp : p) \Rightarrow \lambda\ (hpq : p \to q)\ (hnpq : \neg p \to q).\ hpq\ hp$
| Or.inr $(hnp : \neg p) \Rightarrow \lambda\ (hpq : p \to q)\ (hnpq : \neg p \to q).\ hnpq\ hnp$

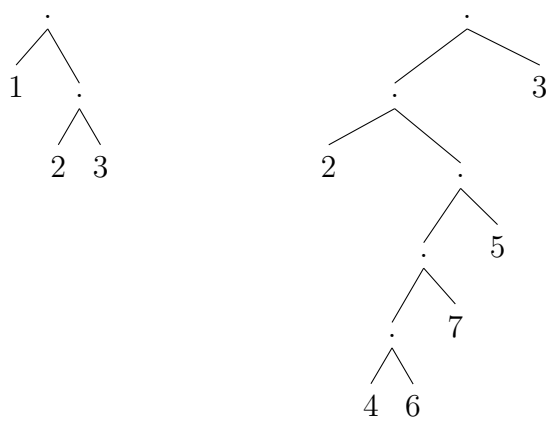
Aufgabe 6 (Induktive Prädikate)**.** Der folgende induktive Typ repräsentiert binäre Bäume mit natürlichen Zahlen an seinen Blättern:

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

Die folgende Funktion spiegelt einen Baum an der vertikalen Achse:

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

Wir nennen einen Baum entartet, wenn an jeder Verzweigung mindestens eine Seite ein Blatt ist. Hier sind zwei Beispiele:



Formal können wir dies definieren als:

inductive Entartet : BTree → Prop :=
| base : ∀(n : Nat). Entartet (leaf n)
| left : ∀(n : Nat) {x : BTree}. Entartet x → Entartet (branch x (leaf n))
| right : ∀(n : Nat) {y : BTree}. Entartet y → Entartet (branch (leaf n) y)

(a) Beweisen Sie, dass der Beispielbaum oben links entartet ist:

theorem entartet_beispiel :
Entartet (branch (leaf 1) (branch (leaf 2) (leaf 3))) := ...

(b) Vervollständigen Sie den folgenden Beweis, dass die Spiegelung eines entarteten Baumes wieder entartet ist:

theorem entartet_mirror $(t : \mathsf{BTree}) : \mathsf{Entartet}\ t \to \mathsf{Entartet}\ (\mathsf{mirror}\ t)$
| base $(n : \mathsf{Nat}) \Rightarrow$
show Entartet (mirror (leaf n)) **from**

| @left $(n : \mathsf{Nat})$ $(x : \mathsf{BTree})$ $(hx : \mathsf{Entartet}\ x) \Rightarrow$
show Entartet (mirror (branch x (leaf n))) **from**

| @right $(n : \mathsf{Nat})$ $(y : \mathsf{BTree})$ $(hy : \mathsf{Entartet}\ y) \Rightarrow$
show Entartet (mirror (branch (leaf n) y)) **from**

Tipps: Beachten Sie die Definition von mirror und die definitionellen Gleichheiten, die sie mit sich bringt. An zwei Stellen brauchen Sie Induktion.

Lösung.

(a)

```
theorem entartet_beispiel :
  Entartet (branch (leaf 1) (branch (leaf 2) (leaf 3))) :=
right 1 (right 2 (base 3))
```

(b)

```
theorem entartet_mirror (t : BTree) : Entartet t → Entartet (mirror t)
| base (n : Nat) ⇒
  show Entartet (mirror (leaf n)) from
  show Entartet (leaf n) from
  base n
| @left (n : Nat) (x : BTree) (hx : Entartet x) ⇒
  show Entartet (mirror (branch x (leaf n))) from
  show Entartet (branch (leaf n) (mirror x)) from
  right n (entartet_mirror x hx)
| @right (n : Nat) (y : BTree) (hy : Entartet y) ⇒
  show Entartet (mirror (branch (leaf n) y)) from
  show Entartet (branch (mirror y) (leaf n)) from
  left n (entartet_mirror y hy)
```

Aufgabe 7 (Gleichungen)**.** Gegeben seien die folgenden Funktionen length, die die Länge einer Liste zurückgibt, und countdown, die eine Liste der Form $[n-1, n-2, \ldots, 0]$ für gegebenes n zurückgibt:

```
def length {X : Type} : List X → Nat
| nil ⇒ zero
| cons x xs ⇒ succ (length xs)

def countdown : Nat → List Nat
| zero ⇒ nil
| succ x ⇒ cons x (countdown x)
```

Beweisen Sie:

```
theorem length_countdown (x : Nat) : length (countdown x) = x := ...
```

Tipp: Verwenden Sie die Definitionen von length und countdown und gebrauchen Sie Induktion.

Lösung.

```
theorem length_countdown (x : Nat) : length (countdown x) = x :=
match x with
| zero ⇒ rfl
| succ x ⇒
  calc length (countdown (succ x))
  _ = succ (length (countdown x))    := rfl
  _ = succ x    := by rw [length_countdown x]
```

Aufgabe 8 (Rekursoren)**.** Definieren Sie die folgenden Funktionen mithilfe von Rekursoren statt Pattern-Matching. Geben Sie auch die impliziten Argumente der Rekursoren an.

(a)

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

Verwenden Sie hier den Rekursor von Nat:

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

(b)

$$\begin{aligned}&\textbf{def}\ \mathsf{umdrehen}\ \{X : \mathsf{Type}\} : \mathsf{List}\ X \to \mathsf{List}\ X\\ &\mid \mathsf{nil} \Rightarrow \mathsf{nil}\\ &\mid \mathsf{cons}\ x\ xs \Rightarrow \mathsf{verbinde}\ (\mathsf{umdrehen}\ xs)\ (\mathsf{cons}\ x\ \mathsf{nil})\end{aligned}$$

Hierbei kann die Funktion $\mathsf{verbinde} : \Pi\{X : \mathsf{Type}\}.\ \mathsf{List}\ X \to \mathsf{List}\ X \to \mathsf{List}\ X$ als gegeben angenommen werden. Verwenden Sie den Rekursor von List:

$$\begin{aligned}&\mathsf{List.rec}.\{u\} : \Pi\{X : \mathsf{Type}\}.\ \Pi\{\mathit{motive} : \mathsf{List}\ X \to \mathsf{Sort}_u\} \to\\ &\quad \mathit{motive}\ \mathsf{nil} \to\\ &\quad (\Pi(x : X)\ (xs : \mathsf{List}\ X).\ \mathit{motive}\ xs \to \mathit{motive}\ (\mathsf{cons}\ x\ xs)) \to\\ &\quad \Pi t : \mathsf{List}\ X.\ \mathit{motive}\ t\end{aligned}$$

Lösung.

(a)

$$\begin{aligned}&\textbf{def}\ \mathsf{isZero}\ (x : \mathsf{Nat}) : \mathsf{Bool} :=\\ &\quad @\mathsf{Nat.rec}\ (\lambda\_.\ \mathsf{Bool})\ \mathsf{true}\ (\lambda n\ r.\ \mathsf{false})\ x\end{aligned}$$

(b)

$$\begin{aligned}&\textbf{def}\ \mathsf{umdrehen}\ \{X : \mathsf{Type}\}\ (t : \mathsf{List}\ X) : \mathsf{List}\ X :=\\ &\quad @\mathsf{List.rec}\ X\ (\lambda\_.\ \mathsf{List}\ X)\ \mathsf{nil}\ (\lambda x\ xs\ r.\ \mathsf{verbinde}\ r\ (\mathsf{cons}\ x\ \mathsf{nil}))\ t\end{aligned}$$