

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	Σ
Punkte									/80

Spickzettel Teil I: Typisierungsregeln

Einfach getypter λ -Kalkül.

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

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

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

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

Calculus of Inductive Constructions.

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

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

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

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

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

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

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

wobei

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

Spickzettel Teil II: Wichtige Definitionen

Logische Prädikate.

```

inductive And (p q : Prop) : Prop :=
| And.intro : p → q → And p q

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

inductive Or (p q : Prop) :=
| Or.inl : p → Or p q
| Or.inr : q → Or p q

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

def Not (p : Prop) : Prop := p → False

inductive Iff (p q : Prop) : Prop :=
| Iff.intro : (p → q) → (q → p) → Iff p q

inductive Exists.{u} {X : Typeu} (p : X → Prop) : Prop :=
| Exists.intro : ∀ x : X. p x → Exists p

inductive Eq.{u} {X : Typeu} (x : X) : X → Prop :=
| rfl : Eq x x

```

Eliminationsregeln.

```

And.left : Π{p q : Prop}. And p q → p
And.right : Π{p q : Prop}. And p q → q

Or.elim : Π{p q r : Prop}. Or p q → (p → r) → (q → r) → r

False.elim : Π{p : Prop}. False → p

Iff.mp : ∀{p q : Prop}. Iff p q → (p → q)
Iff.mpr : ∀{p q : Prop}. Iff p q → (q → p)

Exists.elim.{u} : Π{X : Typeu} {p : X → Prop} {q : Prop}.
  Exists p → (∀ x : X. p x → q) → q

Eq.subst : Π{X : Type} {x y : X} {p : X → Prop}. Eq x y → p x → p y

```

Induktive Typen.

```

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

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

inductive List (X : Type) : Type :=
| nil : List X
| cons : X → List X → 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

$$t \text{ true true} \equiv \text{true}$$

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

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

$$t \text{ false false} \equiv \text{false}$$

Max Mustermann

Aufgabe 1
(Seite 5)

Max Mustermann

Aufgabe 1
(Seite 6)

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

Max Mustermann

Aufgabe 2
(Seite 8)

Max Mustermann

Aufgabe 2
(Seite 9)

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

diagonale f $a \equiv f$ a a und **diagonale** g $b \equiv g$ b b

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

Max Mustermann

Aufgabe 3
(Seite 11)

Aufgabe 4 (Induktive Typen und Pattern-Matching).

- (a) Definieren Sie eine Funktion $\text{or} : \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}$, 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):

```
def pred : Nat → Nat
| 0 ⇒ 0
| succ x ⇒ x
```

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)

Max Mustermann

Aufgabe 4
(Seite 14)

Max Mustermann

Aufgabe 4
(Seite 15)

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

(a)

theorem forall_swap ($p : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Prop}$) ($h : \forall x y. p x y$) :
 $\forall y x. p x y := \dots$

(b)

theorem by_cases ($p q : \text{Prop}$) : $(p \rightarrow q) \rightarrow (\neg p \rightarrow q) \rightarrow q := \dots$

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

Max Mustermann

Aufgabe 5
(Seite 17)

Max Mustermann

Aufgabe 5
(Seite 18)

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

```

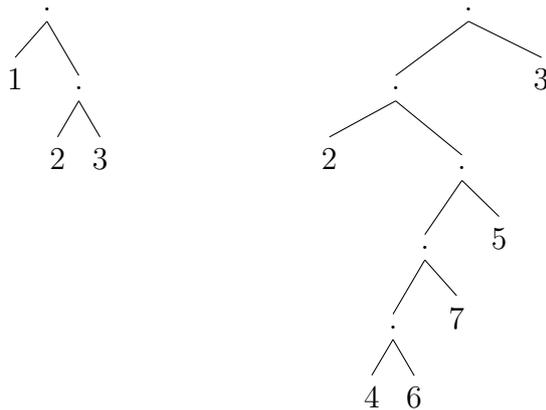
inductive BTree :=
  | leaf : Nat → BTree
  | branch : BTree → BTree → BTree
  
```

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 ($t : \text{BTree}$) : Entartet $t \rightarrow$ Entartet (mirror t)

| base ($n : \text{Nat}$) \Rightarrow

show Entartet (mirror (leaf n)) **from**

| @left ($n : \text{Nat}$) ($x : \text{BTree}$) ($hx : \text{Entartet } x$) \Rightarrow

show Entartet (mirror (branch x (leaf n))) **from**

| @right ($n : \text{Nat}$) ($y : \text{BTree}$) ($hy : \text{Entartet } y$) \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.

Max Mustermann

Aufgabe 7 (Gleichungen). Gegeben seien die folgenden Funktionen `length`, die die Länge einer Liste zurückgibt, und `countdown`, die eine Liste der Form $[n - 1, n - 2, \dots, 0]$ 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` ($x : \text{Nat}$) : `length (countdown x) = x := ...`

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

Max Mustermann

Aufgabe 7
(Seite 24)

Max Mustermann

Aufgabe 7
(Seite 25)

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

(a)

```
def isZero : Nat → Bool
| zero ⇒ true
| succ x ⇒ false
```

Verwenden Sie hier den Rekursor von Nat:

```
Nat.rec.{u} : Π{motive : Nat → Sortu}.
  motive zero →
  (Πn : Nat. motive n → motive (succ n)) →
  Πt : Nat. motive t
```

(b)

```
def umdrehen {X : Type} : List X → List X
| nil ⇒ nil
| cons x xs ⇒ verbinde (umdrehen xs) (cons x nil)
```

Hierbei kann die Funktion `verbinde : Π{X : Type}. List X → List X → List X` als gegeben angenommen werden. Verwenden Sie den Rekursor von List:

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

Max Mustermann

Aufgabe 8
(Seite 27)

Max Mustermann

Aufgabe 8
(Seite 28)