Woche 2 – Übersicht, Tricks & Aufgaben

(Disclaimer: Die hier vorzufindenden Notizen haben keinerlei Anspruch auf Korrektheit oder Vollständigkeit und sind nicht Teil des offiziellen Vorlesungsmaterials. Alle Angaben sind ohne Gewähr.)

Anmerkungen zu den Code-Expert Abgaben:

- Insgesamt sehr schön gelöst. Mir sind keine besonderen Probleme aufgefallen.

Anmerkungen zu den Moodle Abgaben:

```haskell
-- Insgesamt gab es zwei Probleme, die sehr häufig aufgetreten sind:

Erstes Problem:
-- Probleme bei der Abgabe:

-- next (a,b) = (b, a+b) erfordert kein sofortiges Evaluieren der Argumente.

-- (Falls ihr euch jetzt fragt, wieso das bei fac in der Übungsstunde anders war:)
-- Bei fac (3-2) mussten wir bspw. "3-2" sofort evaluieren, da es ein PATTERN MATCHING gab, hier ist
-- es egal, welche Werte die Expressions a und b haben

next :: Num b => (b, b) -> (b, b)
next (a,b) = (b, a+b)

func :: Num b => (b, b) -> b
func x = fst(next x)

-- Wenn wir jetzt func (1/0, 2) ausführen,
-- bekommen wir ohne Probleme 2.0 zurück - obwohl wir durch 0 teilen. Warum?



Zweites Problem:
-- Wie wird hier evaluiert: (1 + 1 + fibLouis 2) + (1 + 1) ? (Was ist der nächste Schritt?)

-- Outermost: "(if possible first try to) evaluate the function, then evaluate its arguments." (- Sofia)
-- Was das heisst: Wir evaluieren den äussersten Operator/Funktion zuerst
{-
Beispiel:

((1+1) + fibLouis (3-2)) + fibLouis (4-2)

Leftmost: Also zuerst "((1+1) + fibLouis (3-2))" evaluieren
Outermost: Also zuerst das Plus zwischen (1+1) und fibLouis (3-2) evaluieren
Also:
Leftmost: Also zuerst 1+1 evaluieren
Also:
= 2 + fibLouis (3-2)
= ...
-}
```

Alpha-Conversion:

Kleiner First-Order-Logic (ich werde von nun an **FOL** schreiben) Recap:

Wir arbeiten mit Termen und Formeln. Terme sind entweder Variablen $x \in V$ oder (multivariate) Funktionen $f^n(t_1, \dots, t_n)$ wobei $t_1, \dots, t_n$ selbst Terme sind (Für $n = 0$ handelt es sich um Konstanten, bspw. "4").

Formeln sind:

$$\bot$$

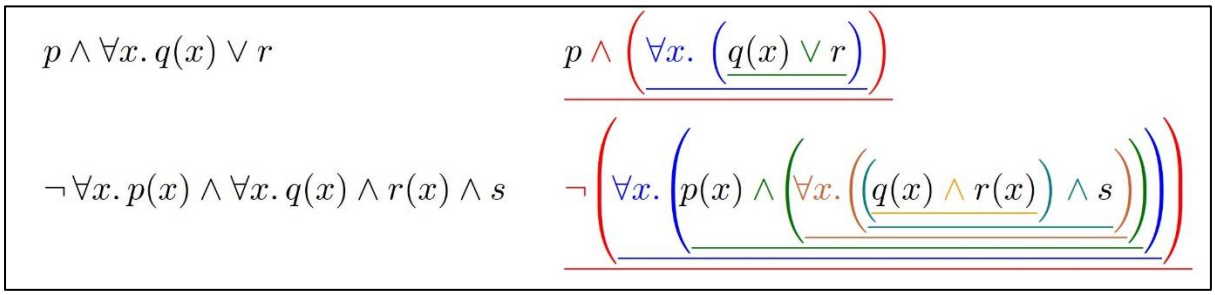
$$p^n(t_1, \dots, t_n), \qquad p \in P \text{ (Prädikate)}$$

$$A \circ B, \qquad A, B \in Formeln, \qquad \circ \in \{\wedge, \vee, \rightarrow\}$$

$$Qx.A, \qquad A \in Formeln, \quad x \in V, \quad Q \in \{\exists, \forall\}$$

Wichtig: **Quantifier reichen so weit wie möglich, also entweder bis zum Ende einer "Zeile" oder bis zu einer schliessenden Klammer ")"!!!**

Hier dazu ein hilfreiches Bild aus der VL:



Gem. Vorlesung gilt für alle Variablen, dass sie entweder "bound" oder "free" sind:

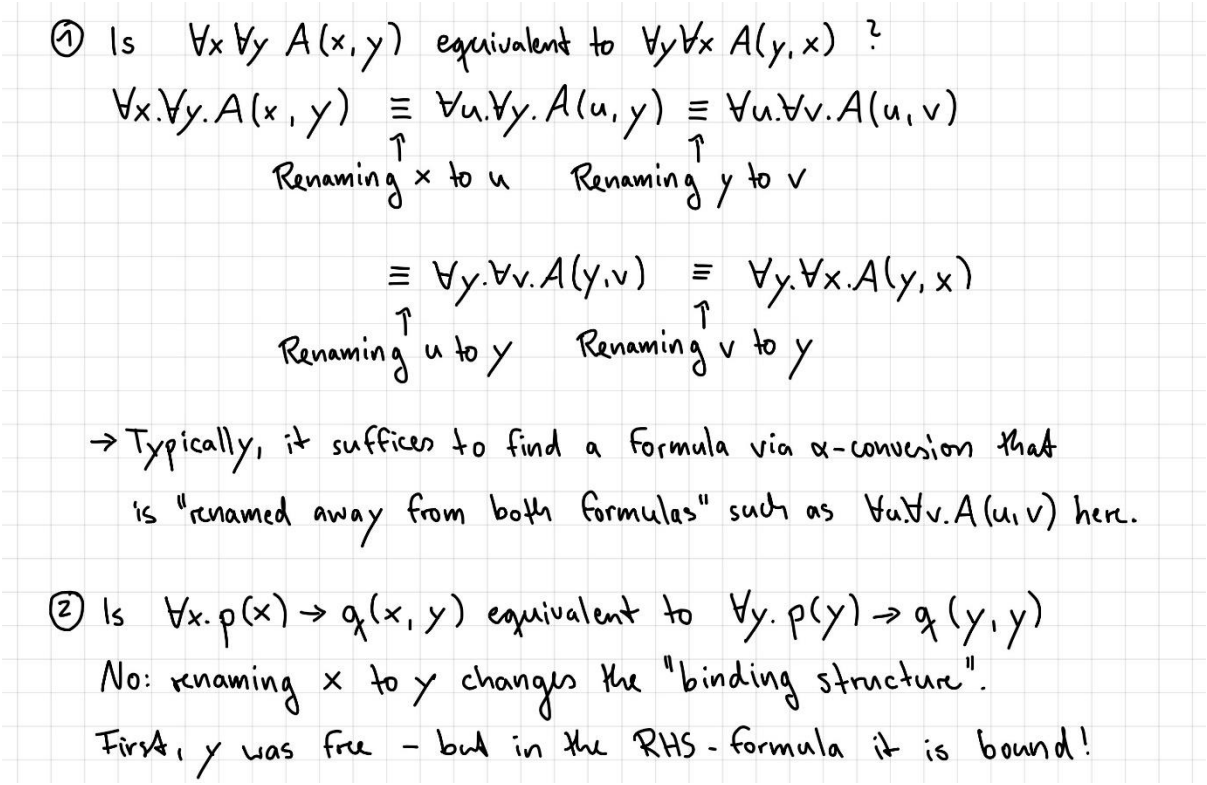
$$(q(x) \vee \exists x.\, \forall y.\, p(f(x), z) \wedge q(a)) \vee \forall x.\, r(x, z, g(x))$$

In diesem Beispiel aus der Vorlesung (Quelle: ND-Slides) sind die blauen Variablen "bound", da sie in einer Subformel A der Form $\exists x.\, B$ bzw. $\forall x.\, B$ vorkommen und die roten Variablen "free" (denen in einer Interpretation noch ein Wert zugewiesen werden muss, siehe Diskrete Mathematik).

Solange wir nichts an der "binding structure" der Formel (also welche Variablen bound und welche free sind und durch welche Quantifier die bound variables gebunden werden) ändern, dürfen wir die Namen von bound Variables beliebig verändern. (→ Merke: "Binding Structure": Welche Variablen bound, welche free sind und durch welche Quantifiers sie gebunden werden!!)

Hier ein paar Aufgaben (von den Slides, die wir in der Übungsstunde gelöst haben):

(Nächste Seite)



(Wozu machen wir das eigentlich? Unter anderem für das bessere Verständnis von ND-Proofs und Lambda-Expressions in Haskell (später mehr dazu)).

Substitution:

Wir schreiben $A[x \to t]$ für das Substituieren eines Terms t für eine "free" Variable x, also für das Ersetzen aller Vorkommnisse von x in A durch t.

Einfaches Beispiel: Sei $A \equiv \forall x.p(x) \vee z$. Dann ist $A[z \to y] \equiv \forall x.p(x) \vee y$ oder $A[z \to f(v)] \equiv \forall x.p(x) \vee f(v)$.

Wichtig: Die "free" Variablen in dem Term t müssen auch nach der Substitution "free" bleiben! Sehen können wir das Anhand unseres Beispiels von oben: Für $t \coloneqq x$ gilt, dass x "free" ist, also gibt es bei der Subsitution $A[z \to x] \equiv \forall x.p(x) \vee x$ offensichtlich ein Problem, da das "zweite" x nun "bound" ist!

➔ Lösung: Erst α-conversion anwenden, dann Substituieren!

Hier also: Erst α-conversion: $A \equiv \forall v.p(v) \vee z$ und dann substituieren: $A[z \to x] \equiv \forall v.p(v) \vee x$.

Erweiterung unserer ND-Regeln für FOL:

Zunächst müssen wir uns noch einmal daran erinnern, dass Quantifier so weit wie möglich (also bis zum Ende einer Zeile oder bis zu einer schliessenden Klammer) reichen, das heisst wir könnten die Regel →-I **nicht** für die Formel $\forall x.P \to Q$ anwenden, da hier eigentlich $\forall x.(P \to Q)$ steht. (Achtung: Das wurde in anderen Kursen (Diskrete Mathematik) eventuell anders gelehrt!).

Wir führen vier neue Regeln (je eine Elimination und eine Introduction) für die beiden Quantifier, die im Gegensatz zu Prädikatenlogik bei der FOL neu sind, ein:

$$\frac{}{\Gamma, A \vdash A}\ axiom \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{-}I \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to\text{-}E$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot\text{-}E \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg\text{-}I \qquad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash B}\ \neg\text{-}E$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-}I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-}EL \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-}ER$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-}IL \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-}IR$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\text{-}E$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ \forall\text{-}I^{*} \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x \mapsto t]}\ \forall\text{-}E$$

$$\frac{\Gamma \vdash A[x \mapsto t]}{\Gamma \vdash \exists x.A}\ \exists\text{-}I \qquad \frac{\Gamma \vdash \exists x.A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ \exists\text{-}E^{**}$$

Side conditions:

* x *does not occur free in any formula in* Γ*;*
** x *does not occur free in any formula in* Γ *nor in* B*.*

Einzelne kurze Erklärungen (Ich erkläre die Regeln immer "von oben nach unten", also in der Reihenfolge, wie man einen ND-Proof lesen – aber nicht schreiben – würde) zu den interessanten 3:

Zunächst betrachten wir die Regeln $\forall - I^{*}$ und $\exists - E^{**}$: Wenn aus unseren Annahmen folgt, dass A gilt, können wir herleiten, dass aus unseren Annahmen folgt, dass A für alle x gilt bzw. Wenn aus unseren Annahmen folgt, dass ein x existiert, sodass A gilt, und aus unseren Annahmen und A folgt, dass B gilt, dann können wir herleiten, dass aus unseren Annahmen B folgt.

> Wichtig: Warum müssen die jeweiligen Nebenbedingungen gelten? Hier gibt es eine sehr schöne Erklärung: https://math.stackexchange.com/questions/1819976/natural-deduction-introduction-of-universal-quantifier-and-elimination-of-exist (zweite Antwort, @dankness)
> Kurz gesagt: Ohne die NB kann es passieren, dass wir aus einer "Aussage" über ein spezifisches Objekt (die freie Variable x) eine allgemeine Aussage schliessen.

Nun betrachten wir die Regel $\forall - E$: Wenn aus unseren Annahmen folgt, dass A für alle x gilt, dann können wir herleiten, dass aus unseren Annahmen folgt, dass A auch gilt, wenn man x durch einen beliebigen Term substituiert (da A ja für alle x gilt). Achtung: Eventuell muss man zunächst α-conversion anwenden, um eine Änderung der binding structure zu vermeiden. Beispiel von Sofia's Slides:

$$\frac{\Gamma \vdash \forall x.\forall y.\, p(x, y)}{(\forall y.\, p(x, y))[x/f(y)] \equiv \forall z.\, p(f(y), z)}\ \forall\text{-E}$$

So schreiben wir die Formel um

Das hier haben wir "anfangs"

Hier ein Beispiel von Sofias Slides, wie wir die Regel $\forall - E$ häufig verwenden:

Wir wollen zeigen: $\forall y.\, p(y) \vdash p(f(u))$ – aber wie?
Mit Substitution! Denn $p(f(u)) \equiv p(y)\,[y \mapsto f(u)]$
Also:

$$\dfrac{\dfrac{}{\forall y.\, p(y) \vdash \forall y.\, p(y)}\ \text{Axiom}}{\forall y.\, p(y) \vdash p(f(u)) \equiv p(y)[y \mapsto f(u)]}\ \forall\text{-E}$$

↳ Wenn wir die Regel $\forall$-E betrachten, ist hier also $A \equiv p(y)$ und $[x \mapsto t]$ entspricht $[y \mapsto f(u)]$ und wir können die Regel anwenden!

Und hier nun die Aufgaben von den Slides mit Lösungen (die wir in der Übungsstunde gelöst haben):

① Beweise $\forall x.\, \bot \to p(x)$

Zunächst schreiben wir um: $\forall x.\, \bot \to p(x) \equiv \forall x.\, (\bot \to p(x))$, $\Gamma := \varnothing$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \bot \vdash \bot}\ \text{Axiom}}{\Gamma, \bot \vdash p(x)}\ \bot\text{-E}}{\Gamma \vdash \bot \to p(x)}\ \to\text{-I}}{\Gamma \vdash \forall x.\, (\bot \to p(x))}\ \forall\text{-I}^{*}$$

(*) x is not free in Γ

② Beweise $\forall x.\, p(x),\ p(t) \to q \vdash q$

Wir def.: $\Gamma := \forall x.\, p(x),\ p(t) \to q$

$$\dfrac{\dfrac{}{\Gamma \vdash p(t) \to q}\ \text{Axiom} \qquad \dfrac{\dfrac{}{\Gamma \vdash \forall x.\, p(x)}\ \text{Axiom}}{\Gamma \vdash p(t) \equiv p(x)\,[x \to t]}\ \forall\text{-E}}{\Gamma \vdash q}\ \to\text{-E}$$

③ Beweise $(\forall x.\forall y.\ p(x,y)) \to (\forall a.\forall b.\ p(a,b))$

Wir def.: $\Gamma := \forall x.\forall y.\ p(x,y)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \forall x.\forall y.\ p(x,y)}\,\text{Axiom}}{\Gamma \vdash \forall y.\ p(a,y) \equiv \forall y.\ p(x,y)[x \mapsto a]}\,\forall\text{-E}}{\Gamma \vdash p(a,b) \equiv p(a,y)[y \mapsto b]}\,\forall\text{-E}}{\Gamma \vdash \forall b.\ p(a,b)}\,\forall\text{-I}^{2}}{\Gamma \vdash \forall a.\forall b.\ p(a,b)}\,\forall\text{-I}^{1}}{\vdash (\forall x.\forall y.\ p(x,y)) \to (\forall a.\forall b.\ p(a,b))}\,\to\text{-I}$$

(1) a not free in Γ

(2) b not free in Γ

④ Beweise $\exists x.P \vdash \neg(\forall x.\neg P)$

Wir def.: $\Gamma := \exists x.P,\ \forall x.\neg P$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \exists x.P}\,\text{Axiom} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash \forall x.\neg P}\,\text{Axiom}}{\Gamma, P \vdash \neg P}\,\forall\text{-E} \qquad \dfrac{}{\Gamma, P \vdash P}\,\text{Axiom}}{\Gamma, P \vdash \bot}\,\neg\text{-E}}{\exists x.P,\ \forall x.\neg P \vdash \bot}\,\exists\text{-E}^{**}}{\exists x.P \vdash \neg(\forall x.\neg P)}\,\neg\text{-I}$$

(**) x not free in any formula in Γ or $\bot$ (trivial)

⑤ Beweise $(\exists z.\ Q(z) \wedge P(z)) \to \exists w.\ Q(w)$

Wir def.: $\Gamma := \exists z.\ Q(z) \wedge P(z)$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \exists z.\ Q(z) \wedge P(z)}\,\text{Axiom} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, Q(z) \wedge P(z) \vdash Q(z) \wedge P(z)}\,\text{Axiom}}{\Gamma, Q(z) \wedge P(z) \vdash Q(z)}\,\wedge\text{-EL}}{\Gamma, Q(z) \wedge P(z) \vdash \exists w.\ Q(w)}\,\exists\text{-I}}{\Gamma \vdash \exists w.\ Q(w)}\,\exists\text{-E}^{**}}{\vdash (\exists z.\ Q(z) \wedge P(z)) \to \exists w.\ Q(w)}\,\to\text{-I}$$

(**) z not free in Γ or $\exists w.\ Q(w)$

Haskell Rekursion – Fortgeschritten(er):

Die Funktionen, die wir in der Übungsstunde implementiert haben:

```haskell
-- Compute Length of a given List
myLen :: [a] -> Int
myLen [] = 0
myLen (x:xs) = 1 + myLen xs
```

```haskell
-- Add an Element at the End of a List
myAdd :: [a] -> a -> [a]
myAdd [] a = [a]
myAdd (x:xs) a = x : myAdd xs a
```

```haskell
-- Zip two Lists, e.g. myZip [1,2,3] "abc" = [(1,'a'),(2,'b'),(3,'c')]
-- If one list is longer, just ignore the rest of the longer list
myZip :: [a] -> [b] -> [(a,b)]
myZip [] _ = []
myZip _ [] = []
myZip (x:xs) (y:ys) = (x,y) : myZip xs ys
```

```haskell
-- Find maximum of List:
myMaxList :: (Ord a) => [a] -> a
myMaxList [] = error "Maximum of empty list"
myMaxList (x:xs) = aux xs x
  where
    aux [] x = x
    aux (y:ys) x
     | y > x     = aux ys y
     | otherwise = aux ys x
```

```haskell
-- Reverse of a List
myreverse :: [a] -> [a]
myreverse [] = []
myreverse (x:xs) = (myreverse xs) ++ [x]
```

```haskell
-- Replicator (replicator 5 'a' = "aaaaa")
replicator :: (Num i, Ord i) => i -> a -> [a]
replicator 0 a = []
replicator i a = a : (replicator (i-1) a)
```

```haskell
-- Split String into List of Strings, split at char c, e.g.
-- split 'b' "aaabaaabaaa" = ["aaa","aaa","aaa"]
split :: Char -> String -> [String]
split c word = aux c word [] []
  where
    aux c [] myW result = (result ++ [myW])
    aux c (x:xs) myW result
     | c == x           = aux c xs [] (result ++ [myW])
     | otherwise        = aux c xs (myW ++ [x]) result
```

```haskell
-- myFilter, given arguments a test and a list should only leave
-- elements in the list that pass the test, e.g.
-- myFilter (>3) [1,2,3,4,5] = [4,5]
myFilter :: (a->Bool) -> [a] -> [a]
myFilter test [] = []
myFilter test (x:xs)
    | test x = x : myFilter test xs
    | otherwise = myFilter test xs
```