



Universität Karlsruhe (TH)

Institut für Innovatives Rechnen und Programmstrukturen (IPD)

Informatik I WS 2002/03

Dozent: Prof. Dr.rer.nat. G. Goos

Übungsleiter: Markus L. Noga

<http://eins.info.uni-karlsruhe.de>

goos@ipd.info.uni-karlsruhe.de

noga@ipd.info.uni-karlsruhe.de

Übungsblatt 9

Ausgabe: 13.12.2002

Abgabe: 20.12.2002 14.00 Uhr

Einwurf im Keller des Informatik-Hauptbaus (Geb. 50.34)

Aufgabe 1: Church-Booleans (9 T- Punkte)

Die boole'schen Wahrheitswerte repräsentieren eine Auswahl zwischen zwei Alternativen. Im Lambda-Kalkül werden sie als Funktionen dargestellt, die eines ihrer Argumente auswählen:

$$true = (\lambda x.(\lambda y.(x)))$$

$$false = (\lambda x.(\lambda y.(y)))$$

Auch Operatoren auf Wahrheitswerten werden als Funktion dargestellt. Die Negation hat folgende Form:

$$neg = (\lambda x.(x false true))$$

1.1 Funktionen (3 Punkte)

Definieren Sie Funktionen *nand* und *nor*, welche die logischen Operation $\bar{\wedge} \equiv \neg(a \wedge b)$ bzw. $\bar{\vee} \equiv \neg(a \vee b)$ umsetzen.

1.2 Ausdrücke (2 Punkt)

Formen Sie den Ausdruck $a \wedge \neg(b \vee \neg c)$ in einen äquivalenten Ausdruck über $\{\bar{\wedge}, \bar{\vee}, true, false, a, b, c\}$ um und drücken Sie ihn als Funktion im Lambdakalkül aus.

1.3 Reduktion (4 Punkte)

Wenden Sie die Funktion auf eine Repräsentation der Belegung $a = true, b = false, c = true$ im Lambdakalkül an und reduzieren Sie maximal unter Angabe der Reduktionsart.

Aufgabe 2: Church-Zahlen (11 T- Punkte)

Auch natürliche Zahlen werden im Lambda-Kalkül als Funktionen dargestellt. Die Funktion *i* wendet ihr erstes Argument *i*-mal auf das zweite Argument an:

$$0 = (\lambda f.(\lambda x.(x)))$$

$$1 = (\lambda f.(\lambda x.(fx)))$$

$$2 = (\lambda f.(\lambda x.(f(fx))))$$

$$3 = (\lambda f.(\lambda x.(f(f(fx))))))$$

$$\vdots = \vdots$$

Die Nachfolgerfunktion *succ* ist wie folgt definiert:

$$succ = (\lambda n.(\lambda f.(\lambda x.(f(n f x))))))$$

Der Nachfolger von 1 ergibt sich zu:

$$\begin{aligned} succ\ 1 &= (\lambda n.(\lambda f.(\lambda x.(f(n f x)))))(\lambda f.(\lambda x.(f x))) \\ &\equiv^\alpha (\lambda n.(\lambda f.(\lambda x.(f(n f x)))))(\lambda g.(\lambda y.(g y))) \\ &\equiv^\beta (\lambda f.(\lambda x.(f((\lambda g.(\lambda y.(g y))) f x)))) \\ &\equiv^\beta (\lambda f.(\lambda x.(f((\lambda y.(f y)) x)))) \\ &\equiv^\beta (\lambda f.(\lambda x.(f(f x)))) = 2 \end{aligned}$$

2.1 Funktionen (6 Punkte)

Definieren Sie Funktionen *plus*, *times* und *exp*, welche die mathematischen Operatoren $a + b$, $a \times b$ und a^b im Lambda-Kalkül umsetzen (Hinweis: Die Definition von *add* in Goos Band I, S. 217 ist fehlerhaft). Definieren Sie eine Funktion *isNull*, welche die Funktion $x == 0$ umsetzt und den entsprechenden Church-Boolean liefert.

2.2 Ausdrücke (1 Punkt)

Drücken Sie den Ausdruck $a + b \times c^2 == 0$ als Funktion im Lambdakalkül aus.

2.3 Reduktion (4 Punkte)

Wenden Sie die Funktion auf eine Repräsentation der Belegung $a = 1, b = 0, c = 2$ im Lambda-Kalkül an und reduzieren Sie maximal unter Angabe der Reduktionsart.

Aufgabe 3: Church-Booleans in Haskell (18 P- Punkte)

3.1 Funktionen (8 Punkte)

Implementieren sie die Werte *true* und *false* sowie die Operationen *and*, *or*, *neg*, *nand* und *nor* aus Aufgabe 1 als Funktionen `ctrue`, `cfalse`, `cand`, `cor`, `cneg`, `cnand` und `cnor` in Haskell. Definieren Sie eine weitere Funktion `cbshow`, die Church-Booleans auf den Haskell-Datentyp `Bool` abbildet.

Verwenden Sie dabei ausschließlich die Haskell-Notation $(\lambda f \rightarrow g)$ für Lambda-Ausdrücke $(\lambda f.(g))$.

3.2 Prüfung (2 Punkte)

Prüfen Sie die Übereinstimmung von $a \wedge \neg(b \vee \neg c)$ und Ihrer Umformung dieses Terms aus Aufgabe 1.2 für alle Belegungen der Variablen im Haskell-Interpreter mit Hilfe der Funktionen aus der vorigen Teilaufgabe.

3.3 Typen (8 Punkte)

Gegeben sei der Datentyp `type ChurchBoolean a = a -> a -> a`. Typisieren Sie die Funktionen der ersten Teilaufgabe. Platzhalter `a` sind nur als Argument an `ChurchBoolean` zulässig.

Aufgabe 4: Church-Zahlen in Haskell (22 P- Punkte)

4.1 Funktionen (10 Punkte)

Implementieren sie die Werte 0,1,2 und 3 sowie die Operationen *succ*, *plus*, *times*, *exp* und *isNull* aus Aufgabe 2 als Funktionen `c0`, `c1`, `c2`, `c3`, `csucc`, `cplus`, `ctimes`, `cexp` und `cnisnull` in Haskell. Anders als in 1.1 soll `cnisnull` den Haskell `Bools` zurückliefern. Definieren Sie eine weitere Funktion `cnshow`, die Church-Zahlen auf den Haskell-Datentyp `Int` abbildet.

Verwenden Sie dabei ausschließlich die Haskell-Notation $(\lambda f \rightarrow g)$ für Lambda-Ausdrücke $(\lambda f.(g))$.

4.2 Prüfung (2 Punkte)

Prüfen Sie Ihre Implementierung anhand des Ausdrucks $a + b \times c^2 == 0$ aus Aufgabe 2.2. Werten Sie die entsprechende Funktion für alle Belegungen $a, b, c \in \{0, 1, 2, 3\}$ aus. Können Sie eine äquivalente aussagenlogische Formel angeben, die nur den Operator `==` verwendet?

4.3 Typen (10 Punkte)

Gegeben sei der Datentyp `type ChurchNumeral a = (a -> a) -> a -> a`. Typisieren Sie die Funktionen der ersten Teilaufgabe. Platzhalter `a` sind nur als Argument an `ChurchBoolean` und als Ergebnis der Funktion `cnshow` zulässig.