Max Wisniewski , Alexander Steen

Tutor: Ansgar Schneider

Aufgabe 1 *Typüberprüfung*

Bestimmen Sie die Typen der folgenden Funktionen.

(i) $\lambda f\, x.(f\, x) + 1$
Lösung:
Die ersten beiden Hinweise, die wir haben, ist, dass wir eine Konstante $1 \in K^{\mathbb{N}_\perp}$ und eine Funktion $+ : [\mathbb{N}_\perp \to \mathbb{N}_\perp \to \mathbb{N}_\perp]$. Da wir f in die $+$ Funktion stecken, muss der Rückgabetyp $\mathbb{N}_\perp$ sein. Über die Eingabe müssen wir nicht mehr wissen nur, dass f eine Variable nimmt und das x daher diesen Typ haben muss.

$\lambda f\, x.(f\, x) + 1 \ : \ [[D \to \mathbb{N}_\perp] \to \mathbb{N}_\perp \to \mathbb{N}_\perp]$

Nun setzten wir die Variablen ein und überprüfen.
Sei $f \in X^{[D\to\mathbb{N}_\perp]}$ und $x \in X^D$.
Dann ist das einsetzen Korrekt, da $(\lambda f\, x.(f\, x) + 1) fx = (f\, x) + 1 \ : \ \mathbb{N}_\perp \to \mathbb{N}_\perp$
$1 \in K^{\mathbb{N}_\perp}$ das gilt also, nun überprüfen wir, ob $f(\, x) \ : \ \mathbb{N}$ erfüllt.
$f \in X^{[D\to\mathbb{N}_\perp} x \in X^D$, daher ist $fx \ : \ \mathbb{N}_\perp$.

Der Typ ist daher korrekt.

(ii) $\lambda(x, y) f\ .\ f\, x\, y$
Lösung:
Sei $x \in X^{D_1}$, $y \in X^{D_2}$ und $f \in X^{D_3}$.
Die Funktion $h = \lambda(x, y) f\ .\ f\, x\, y \ : \ D_1 \times D_2 \to D_3 \to D_4$. Wir müssen also überprüfen, was D_1, D_2, D_3 ist und welchen Rückgabetyp wir erhalten.

Setzten wir $h\,(x, y) f$ ein erhalten wir:
$f\, x\, y \ : \ D_4$.
Damit wir nun am Ende ein Element von einem Typ erhalten (hier hätten auch 3 Atome stehen können).
Daher muss f eine Funktion sein, die beide Elemente x, y aufnehmen kann.
$\Rightarrow D_3 = D_1 \to D_2 \to D_5$. Und da wir nichts anderes tun ist auch $D_4 = D_5$.
Weiter können wir nun nichts mehr sagen, also gilt:
$h \ : \ D_1 \times D_2 \to [D_1 \to D_2 \to D_4] \to D_4$.

(iii) $\lambda f.(f \lambda y.y)$
Lösung:
Sei $f \in X^{D_1}$. Dann hat die Funktion den folgenden Typ $h = \lambda f.(f \lambda y.y) \ : \ D_1 \to D_2$.

Nun setzten wir unser f ein und erhalten
$(f \lambda y.y) \ : \ D_2$.
Nun muss nach selben Überlegungen wie oben das f die Funktion $\lambda y.y \ : \ [D_3 \to D_3]$ schlucken

können.

Daher braucht ist der Typ $f\ :\ [[D_3 \to D_3] \to D_4]$.
Da dies nun der letzte Schritt ist muss $D_4 = D_2$ sein, da $(f\lambda y.y)\ :\ D_2$ gelten muss.

Die FUnktion hat also den folgenden Typ (umbenennung der Typklassen):
$(h = \lambda f.(f\lambda y.y)\ :\ [[T \to T] \to S] \to S$

Aufgabe 2 *Faltung*

Der Faltungsoperator $\underline{\text{lit}}$ sei informall bestimmt durch:
$\underline{\text{lit}} = f(x_1, ...x_n)x_{n+1} = fx_1(fx_2(...(fx_nx_{n+1})))$

(i) Bestimmen Sie den Typ von $\underline{\text{lit}}$.
Lösung:
Wir sehen zunächst, dass f 3 Eingaben erhält. Die zweite davon ist eine Liste.
Wir vergeben n f den Typ D_1, an die Liste $(x_1, ..., x_n)\ :\ D_2^*$ und an x_{n+1} den Typ D_3.

Setzen wir alle Parameter in die informelle Definition ein, sehen wir, dass es sich bei f um eine Funktion handelt, die 2 Parameter annimmt.
$\Rightarrow D_1 = [D_4 \to D_5 \to D_6]$.

Der zweite Parameter ist in der ersten Iteration x_{n+1} und danach ist der Rückgabewert von f die zweite Eingabe. Daher muss gelten $D_6 = D_5 = D_2$.

Der erste Parameter ist immer ein Element der Liste $D_2 = D_4$ und damit endet die Vorhersage, die wir treffen können.

$\underline{lit}\ :\ [A \to B \to B] \to A^* \to B \to B.$

(ii) Definieren Sie den Operator $\underline{\text{lit}}$ im getypten λ - Kalkül unter Verwendung der Gleichungsschreibweise.
Lösung:

$$\underline{lit} = \underline{fix}\left(\lambda F.\lambda f\, l\, x.\underline{empty}\, l \to x, f(\underline{hd}\, l)(F\, f\, (\underline{tail}\, l)\, x)\right)$$

(iii) Definieren Sie eine Funktion f im getpyten λ - Kalkül, so dass

$$f(x_1, ..., x_n)x = \begin{cases} wahr & \text{, falls } \exists i \leq n\ :\ x = x_i \\ false & \text{, sonst} \end{cases}$$

Lösung:
Wir verwenden die Funktionen $\underline{eq}$ für Gleichheit und $\underline{and}$ für die Verundung.

$$f = \lambda xl.\, \underline{lit}(\lambda x_i\, a.\, a\underline{or}(eqxx_i))l\underline{false}$$

Werten wir das ganze nun aus, bekommen wir in der Faltung die Auswertung
$fx < x_1, .., x_n >= (x == x_1) \vee (x == x_2) \vee ... \vee (x == x_n) \vee false.$

Das letzte False ist überflüssig und wir wissen, dass sobald eines der Literale True ergibt, muss das ganze True sein. Daher ist die Funktion vollständig.

Darüber hinaus ist sie *sound*, da false zurück kommt, wenn keiner der Vergleiche gut geht, oder die Methode im *eq* einen Fehler wirft, wenn die Typen nicht stimmen.

(iv) Bearbeiten Sie (i)-(iii) für
$\underline{lit'} f x_1 \,(x_2, ..., x_{n+1}) = (f((...(f x_1 x_2)...x_{n+1})$

Lösung:
Die Definition machte wie in der Aufgabe gestellt kein Sinn, dher haben wir die übliche $foldl$ Definition gewählt.

a. Wir vergeben an die Eingabeparameter wieder die Typen $f \;:\; D_1, x_1 \;:\; D_2$ und $(x_2, ..., x_{n+1}) \;:\; D_3^*$.

Immer innsersten wird f zunächst auf $x_1 \;:\; D_2$ und $x_2 \;:\; D_3$ angwandt. Daher muss f den Typ $[D_2 \to D_3 \to D_4]$ besitzen.
Als in den folgenden Schritten, wird immer wieder der Rückgabewert als erstes in die FUnktion gesteckt und als zweites ein weiteres Listenelement
$\Rightarrow D_2 = D_4$ und auf der obersten Ebene, wird nur einmal f aufgerufen, daher muss der Rückgabewert von $\underline{lit'}$ gleich D_4 sein.

$\underline{lit'} \;:\; [A \to B \to A] \to A \to B^* \to A$

b. Als nächstes Definieren wir das ganze:

$$\underline{lit'} = \underline{fix}\left(\lambda F.\lambda f\, x\, y.\, \underline{empty}\, y \to x\;,\; f\;(F\; f\; x\;(\underline{init}\, y))(\underline{last} y)\right)$$

Dabei ist $\underline{init} < x_1, ..., x_n > = < x_1, ..., x_{n-1} >$ und tail $\underline{tail} < x_1, ..., x_n > = x_n$ wie üblich definiert.
Wenn die Liste die Länge 1 hat, dann geben wir $f(x_1\, x_2)$.
Bei größerer Anzahl immer $f(f(f(..f(x_1\, x_2)x_3)x_4)....)$.

c. Bei der Definition von f ändert sich nichts, da die Operationen sowohl kommutativ als auch assoziativ ist, ist $foldl$ und $foldr$ äquivalent. (ALP 1).

$$f = \lambda xl.\, \underline{lit}(\lambda a\, x_i.\, a\underline{or}(eqxx_i))\underline{false}l$$

Die Auswertung ist im Gegensatz zu (iii) mit
$f\; < x_1, ..., x_n >\; x = x_1 == x \vee (x_2 == x \vee (....(x_n == n)))$
nur anders geklammert, also $f\, x\; < x_1, .., x_n > = x_n == x \vee (x_{n-1} == x \vee (...(x_1 == x_i)))$.

Da wir nun wie gesagt $\vee$ kommutativ und assoziativ ist, tut die Funktion das selbe.

Aufgabe 3 *Repeat*

Erweitern Sie die Syntax von WHILE um die Anweisung der Form

$$\underline{REPEAT}\ C\ \underline{UNTIL}\ B$$

und definieren Sie dazu eine geeignete denotationelle Semantik.

Lösung:

Zunächst fügen wir wie schon einmal das Konstrukt von REPEAT zur Syntax hinzu:

$C'\ :\ C|\underline{REPEAT}\ C\ \underline{UNTIL}\ B$

Wir definieren die Semantik von REPEAT analog zur WHILE Schleife.

$$C[\underline{REPEAT}\,C\,\underline{UNTIL}\,B] = C[C] \star B[B] \star cond(C[\underline{REPEAT}\,C\,\underline{UNTIL}\,B], \lambda z.z)$$