Praktische Informatik 3 WS 10/11

Christoph Lüth
Bernd Gersdorf
Christian Maeder
René Wagner
Dennis Walter
Diedrich Wolter
Simon Ulbricht

5. Übungsblatt

Ausgabe: 02.12.10
Abgabe: 20.12.10

5.1 *Typableitungen* — *10 Punkte*

Leiten Sie für die folgenden Ausdrücke wie in der Vorlesung vom 01.12.2010 gezeigt mithilfe der Ableitungsregeln aus dem Skript ein allgemeinstes *typing judgment* ("Typurteil") im folgenden Variablenkontext ab:

$$\Gamma = \mathit{True} :: \mathit{Bool}, \mathit{False} :: \mathit{Bool}, \mathit{zs} :: [\mathit{Int}], \\ \mathit{tail} :: [b] \to [b], \mathit{cons} :: c \to [c] \to [c], \mathit{head} :: [d] \to d, \mathit{null} :: [e] \to \mathit{Bool} \quad (1)$$

Instanziieren Sie die Typvariablen b bis e jeweils mittels einer geeigneten Substitution. (je 3 Punkte für 1. und 2., 4 Punkte für 3.)

1. `if True then null zs else False`
2. `(\x →x) (\x →x)`
3. `let cat = \xs → \ys → if null xs then ys else cons (head xs) (cat (tail xs) ys) in cat`

Beispiel: Wir ersetzen e durch Int in Γ und nennen diesen neuen Kontext Γ_η (wir wenden also die Substitution $\eta = \{e \mapsto \mathit{Int}\}$ auf alle Typen in Γ an, wodurch insbesondere $\Gamma_\eta(\mathit{null}) = [\mathit{Int}] \to \mathit{Bool}$ gilt).

Dann zeigen wir $\Gamma_\eta \vdash$ `(\xs →null xs) zs` $:: \mathit{Bool}$.

Nr.	Judgment		Regel[Vorbedingungen]
1	$\Gamma_\eta, \mathit{xs} :: [\mathit{Int}] \vdash$ `xs`	$:: [\mathit{Int}]$	Var
2	$\Gamma_\eta, \mathit{xs} :: [\mathit{Int}] \vdash$ `null`	$:: [\mathit{Int}] \to \mathit{Bool}$	Var
3	$\Gamma_\eta, \mathit{xs} :: [\mathit{Int}] \vdash$ `null xs`	$:: \mathit{Bool}$	App[2, 1]
4	$\Gamma_\eta \vdash$ `\xs →null xs`	$:: [\mathit{Int}] \to \mathit{Bool}$	Abs[3]
5	$\Gamma_\eta \vdash$ `zs`	$:: [\mathit{Int}]$	Var
6	$\Gamma_\eta \vdash$ `(\xs →null xs) zs`	$:: \mathit{Bool}$	App[4,5]

5.2 *Substitution* — *2 Punkte*

In dieser Aufgabe betrachten wir Typterme, die Funktionen, Typvariablen sowie die Basistypen `Int` und `Bool` darstellen können. Typterme werden demzufolge anhand der folgenden Grammatik gebildet:

$$\begin{array}{rcll} t & ::= & a \mid b \mid c \mid \ldots & \text{(Variablen)} \\ & \mid & \mathit{Bool} & \\ & \mid & \mathit{Int} & \\ & \mid & t_1 \to t_2 & \text{(Funktionstypen)} \end{array} \quad (2)$$

t_1, t_2 stehen hierbei wiederum –rekursiv– für beliebige Typterme. Beispiele für Typterme sind also a, Int, $\mathit{Int} \to \mathit{Bool}$, $\mathit{Bool} \to b$, oder auch $(a \to b) \to c$.

Wenden Sie "per Hand" die folgenden Substitutionen auf die dazugehörigen Typterme (kurz: Typen) an. Geben Sie jeweils nur das Ergebnis der Substitution an.

Beispiel: Typ: $a \to b \to c$. Substitution: $\eta = \{a \mapsto b, c \mapsto (\mathit{Int} \to \mathit{Int})\}$. Ergebnis der Anwendung, d.h. $(a \to b \to c)_\eta$ ergibt $b \to b \to (\mathit{Int} \to \mathit{Int})$.

1. Typ: $a \to a$. Substitution: $\eta_1 = \{a \mapsto (\mathit{Int} \to c), b \mapsto c\}$
2. Typ: $((a \to a) \to a) \to b$. Substitution: $\eta_2 = \{a \mapsto \mathit{Bool}, b \mapsto ((c \to c) \to c)\}$.

3. Typ: $(a \to c) \to b \to d$. Substitution: $\eta_3 = \{a \mapsto Int, c \mapsto Bool, b \mapsto d\}$

5.3 *Unifikation* — *8 Punkte*

Finden Sie ebenfalls "per Hand" durch Anwendung der Rechenvorschriften des Unifikationsalgorithmus $\mathcal{U}$ (s. Skript) Unifikatoren für die folgenden Paarlisten von Typtermen. Die Paare repräsentieren übrigens Gleichheitsbedingungen zwischen Typen, die nicht aus der Luft gegriffen wurden (sogenannte *air conditions*), sondern sie entstehen bei der Typinferenz der dazu angegebenen Haskell-Ausdrücke.

Beispiel: Für den Ausdruck $1 + 1$ (mit $(+) :: Int \to Int \to Int$) ergeben sich die geforderten Typgleichheiten $b \to a \to z = Int \to Int \to Int$, $b = Int$ und $a = Int$.

- Die Eingabe für $\mathcal{U}$ ist also $[(b \to a \to z, Int \to Int \to Int), (b, Int), (a, Int)]$.
- Wir berechnen $\mathcal{U}((b \to a \to z, Int \to Int \to Int) : [(b, Int), (a, Int)])$ und landen im Fall 3: $t = b \to a \to z$ und $t' = Int \to Int \to Int$. Wir müssen also $\mathcal{U}((b, Int) : (a, Int) : (z, Int) : [(b, Int), (a, Int)])$ berechnen.[1]
- Nun ist Fall 1 zutreffend: $t = b$ und $t' = Int$. Der Occurs-Check gelingt, wir setzen $\eta_1 = \{b \mapsto Int\}$ und berechnen $\mathcal{U}(((a, Int) : (z, Int) : [(b, Int), (a, Int)])_{\eta_1})$, also $\mathcal{U}((a, Int) : (z, Int) : [(Int, Int), (a, Int)])$.
- Wieder trifft Fall 1 zu: $t = a$ und $t' = Int$. Setze $\eta_2 = \{a \mapsto Int\}$ und berechne $\mathcal{U}(((z, Int) : [(Int, Int), (a, Int)])_{\eta_2})$, d.h. $\mathcal{U}((z, Int) : [(Int, Int), (Int, Int)])$.
- Zu guter Letzt: $t = z$ und $t' = Int$, $\eta_3 = \{z \mapsto Int\}$. Für die verbleibende Liste ergibt sich

$$\mathcal{U}([(Int, Int), (Int, Int)]) = \{\}.$$

- Die rekursiven Aufrufe zurückverfolgend erhalten wir somit $\eta = (\eta_3 \circ \eta_2) \circ \eta_1$ bzw. ausgewertet $\{a \mapsto Int, b \mapsto Int, z \mapsto Int\}$ als allgemeinsten Unifikator. z entspricht hierbei übrigens gerade dem zunächst mutig angenommenen Typen des Gesamtausdrucks `1 + 1`, der also mittels η zu *Int* instanziiert werden muss.

□

1. Berechnen Sie wie oben demonstriert den Unifikator

$$\mathcal{U}([(a, Int \to Bool), (b, Bool \to Bool), (b \to a \to z, (d \to c) \to (e \to d) \to e \to c)])$$

   der sich bei der Typinferenz für `not . even` (mit den Typen von `not` und `(.)` wie im Haskell-Prelude und `even :: Int →Bool`) ergibt.

2. Nun berechnen Sie $\mathcal{U}([(z, a \to b), (c \to b, a), (c, a)])$, das sich bei der Typinferenz für (`\ x →x x`) ergibt.

Beachten Sie, dass es nicht für jede Liste von Typgleichheiten einen Unifikator gibt! (Jeweils 2 Punkte)

Die Berechnung von $\mathcal{U}([(z, a \to b), (b, c \to d), (d, e \to f), (h \to g \to f, a), (h, e), (g, c)])$ — entstanden für den Ausdruck `\f →\x →\y →f y x`, vielen besser bekannt als `flip` — möchte niemand händisch durchführen. Vervollständigen Sie daher den teilweise vorgegebenen Unifikationsalgorithmus `unify` aus der Vorlage `Vorlage-Blatt05.hs`, die Sie auf der PI3-Webseite finden, und wenden Sie ihn auf die obige Eingabe (ebenfalls vordefiniert) an. Sie müssen hierfür ausnahmsweise nur Quellcode-Kommentare erstellen und dürfen auf Testfälle verzichten. (4 Punkte)

Gutes Gelingen!

[1] Der Typkonstruktor C ist in diesem Fall der $\to$, d.h. der Konstruktor für den Funktionstyp. Genau genommen müssten wir diesen Fall zwei mal betrachten, für $(b \to (a \to z))$ und dann für $(a \to z)$, was wir hier abkürzenderweise auslassen. Sie dürfen die n-fache Applikation ebenfalls wie einen einzigen n-stelligen Funktionstypen behandeln.