Theorie der Programmierung SS 2015

Übungsblatt 4

Rev.8927

Abgabe der Lösungen: Tutorium in der Woche 06.07.-10.07.

Eine Annahme

Wenn nicht anders angegeben, bezeichnet $\Gamma \vdash s : \alpha$ die Situation, dass eine entsprechende Typinferenz im System F à la Curry existiert.

Übung 1 Produkte in System F (à la Curry)

Wir kodieren das *kartesische Produkt* der Typen a und b in System F unter Verwendung des Typs $(a \times b) := \forall\, r\ .\ (a \to b \to r) \to r$.

1. Für diese Kodierung ist die "Konstruktions-Funktion" *pair*, welche aus zwei Elementen der Typen a bzw. b ein Paar des Typs $(a \times b)$ konstruiert, wie folgt definiert:

   $pair = \lambda x\ y\ .\ (\lambda f\ .\ f\ x\ y)$

   Zeigen Sie, dass $\vdash$ *pair* : $\forall\, a\ b\ .\ a \to b \to (a \times b)$ in System F gilt.

2. Geben Sie zu jeder der folgenden Funktionssignaturen eine Implementierung der jeweiligen Funktion (d.h. einen λ-Term) an und zeigen Sie, dass Ihre Implementierung den benötigten Typ hat:

   (a) *fst* : $\forall\, a\ b\ .\ (a \times b) \to a$

   (b) *snd* : $\forall\, a\ b\ .\ (a \times b) \to b$

3. Schreiben Sie unter Verwendung der obigen Funktionen eine weitere Funktion

   swap : $\forall\, a\ b\ .\ (a \times b) \to (b \times a)$

   und zeigen Sie, dass sie den korrekten Typ hat. Finden Sie also einen λ-Term s, so dass $\Gamma \vdash s$: $\forall\, a\ b\ .\ (a \times b) \to (b \times a)$, wobei

   $\Gamma := \{pair\colon \forall\, a\ b.a \to b \to (a \times b),\ fst\colon \forall\, a\ b.(a \times b) \to a,\ snd\colon \forall\, a\ b.(a \times b) \to b\}$

Übung 2 Koprodukte in System F (à la Curry)

Wir kodieren das Koprodukt der Typen a und b unter Verwendung des folgenden Typs:
$(a + b) := \forall\, r.(a \to r) \to (b \to r) \to r$.

Für diese Kodierung sind die "Konstruktions-Funktionen" *cLeft* und *cRight* wie folgt definiert:

$cLeft = \lambda x.\lambda f\ g.f\ x$
$cRight = \lambda y.\lambda f\ g.g\ y$

1. Zeigen Sie, dass $\vdash$ *cLeft* : $\forall\ a\ b\ .\ a \to (a + b)$ und $\vdash$ *cRight* : $\forall\ a\ b\ .\ b \to (a + b)$.

2. Schreiben Sie eine Funktion des Typs $\forall\ a\ b\ c.((a \times b) + c) \to (a \to c) \to c$ und zeigen Sie, dass Ihre Funktion den verlangten Typ hat (d.h., geben Sie eine entsprechende Typinferenz an). Wie viele *unterschiedliche* Funktionen diesen Typs gibt es? ("unterschiedlich" im Sinne der berechneten Funktion, nicht "unterschiedlich" im Sinne der Struktur des λ-Terms!)

Übung 3 Listen in System F (à la Curry)

Listen können in System F unter Verwendung des folgenden Typs kodiert werden:

$$\textbf{List}\ a := \ \forall\ r.\ r \to (a \to r \to r) \to r$$

In diesem Fall ergeben sich die folgenden "Konstruktor-Funktionen":

$$\begin{aligned} nil &= \lambda u\ f.u \\ cons &= \lambda x\ l.\lambda u\ f.\ f\ x\ (l\ u\ f) \end{aligned}$$

Für einen gegebenen (und durch *nil* und *cons* konstruierten) Term t des Typs **List** a verhält sich der Term $t\ u\ (\lambda x\ l.s)$ also genau so wie eine Funktion f für die für alle x (des Typs a) und alle l (des Typs **List** a) gilt:

$$\begin{aligned} f\ nil &\to^*_{\beta\delta} u \\ f\ (cons\ x\ l) &\to^*_{\beta\delta} s[l \mapsto f\ l] \end{aligned}$$

1. Zeigen Sie, dass $\vdash$ $nil : \forall a.\textbf{List}\ a$ und $\vdash$ $cons : \forall a.a \to \textbf{List}\ a \to \textbf{List}\ a$.

2. Schreiben Sie eine Funktion *length*, welche die Länge einer Liste berechnet. Es soll gelten:

   $$\begin{aligned} length\ nil &\to^*_{\beta\delta} zero \\ length\ (cons\ x\ l) &\to^*_{\beta\delta} succ\ (length\ l) \end{aligned}$$

   Zeigen Sie, dass $\Gamma_0 \vdash length : \forall a.\textbf{List}\ a \to \mathbb{N}$, wobei $\Gamma_0 = \{zero : \mathbb{N}, succ : \mathbb{N} \to \mathbb{N}\}$.

3. Schreiben Sie eine Funktion zur Listenkonkatenation. Das heißt, schreiben Sie eine Funkton *append*, so dass:

   $$\begin{aligned} append\ nil &\to^*_{\beta\delta} r \\ append\ (cons\ x\ l)\ r &\to^*_{\beta\delta} cons\ x\ (append\ l\ r) \end{aligned}$$

   Zeigen Sie außerdem, dass $\Gamma_0 \vdash append : \forall a.\textbf{List}\ a \to \textbf{List}\ a \to \textbf{List}\ a$, wobei:

   $$\Gamma_0 = \{\, nil : \forall a.\textbf{List}\ a, cons : \forall a.a \to \textbf{List}\ a \to \textbf{List}\ a\}$$

4. Schreiben Sie eine Funktion *join*, die eine Liste von Listen als Argument erwartet und die durch (von links nach rechts erfolgende) Konkatenation der einzelnen Listen entstehende Gesamtliste berechnet. Das heißt, für Ihre Funktion soll gelten:

   $$\begin{aligned} join\ nil &\to^*_{\beta\delta} nil \\ join\ (Cons\ x\ l) &\to^*_{\beta\delta} append\ x\ (join\ l) \end{aligned}$$

   Zeigen Sie, dass $\Gamma_0 \vdash join : \forall a.\textbf{List}\ (\textbf{List}\ a) \to \textbf{List}\ a$, wobei

   $$\Gamma_0 = \{\, nil : \forall a.\textbf{List}\ a, append : \forall a.\textbf{List}\ a \to \textbf{List}\ a \to \textbf{List}\ a\}$$

Übung 4 System F à la Church

1. Definieren Sie die Konstruktor-Funktionen *nil* und *cons* aus Übung 3 als äquivalente Terme des jeweils gleichen Typs in System F à la Church.

2. Implementieren Sie die Funktionen *length*, *append* und *join* als äquivalente Terme des jeweils gleichen Typs in System F à la Church.

Übung 5 *Binäre Bäume in System F* *(10 Punkte)*

In ähnlicher Weise wie für Listen kodieren wir den Typ binärer Bäume mit beschrifteten *Blättern* in System F mittels **BinTree** $a := \forall s.(a \to s) \to (s \to s \to s) \to s$.

1. Definieren Sie die "Konstruktor-Funktionen" *leaf* und *bin* derart, dass eine beliebige Funktion über binären Bäumen in der folgenden Form definiert werden kann:

   $f\ (leaf\ x) \to^*_{\beta\delta} g\ x$
   $f\ (bin\ l\ r) \to^*_{\beta\delta} h\ (f\ l)\ (f\ r)$

   wobei g und h geeignete Funktionen sind.

2. Zeigen Sie, dass $\vdash$ *leaf* : $\forall a.a \to$ **BinTree** a und $\vdash$ *bin* : $\forall a.$**BinTree** $a \to$ **BinTree** $a \to$ **BinTree** a.

Übung 6 *A surprisingly typeable term* *(10 Punkte)*

1. Zeigen Sie, dass $\vdash \lambda xy\,.\,xx : (\forall a.a \to a) \to b \to (\forall a.a \to a)$.

2. Gibt es einen Typ α, so dass $\vdash (\lambda xy\,.\,xx)(\lambda xy\,.\,xx) : \alpha$? Falls ja, geben Sie solch ein α und die zugehörige Typinferenz an; falls nein, begründen Sie, warum solch ein Typ nicht existiert.