Übung zur Vorlesung

Automatisierte Logik und Programmierung I

Prof. Chr. Kreitz

Universität Potsdam, Theoretische Informatik — Wintersemester 2016/17

Blatt 4 — Abgabetermin: 08.12.2016

Das vierte Übungsblatt soll dazu dienen, sich mit Typisierung, Typ-Inferenz und der Argumentation in Normalisierungsbeweisen auseinanderzusetzen.
Wir werden mögliche Lösungen zu Beginn der Veranstaltung am 08.12.2016 besprechen.

Achtung: Am 15. Dezember 2016 finden keine Vorlesungen statt.
Die nächste Veranstaltung ist am 5. Januar 2017

Aufgabe 4.1 (Typisierung)

Geben Sie, wo möglich, eine Typisierung für die folgenden Terme an und beweisen Sie diese mit dem Refinement Kalkül für die einfache Typentheorie.

4.1–a $\lambda \mathtt{t}.\lambda \mathtt{y}.\ \mathtt{t\ y\ y}$
4.1–b $(\lambda \mathtt{x}.\lambda \mathtt{y}.\ \mathtt{x\ y})\ (\lambda \mathtt{z}.\ \mathtt{z})$
4.1–c $\lambda \mathtt{f}.\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x\ x}))\ (\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{x\ x}))$
4.1–d $\lambda \mathtt{y}.\lambda \mathtt{g}.\ (\lambda \mathtt{x}.\ \mathtt{x}^3\ \mathtt{y})\ (\lambda \mathtt{z}.\ \mathtt{g\ z\ z})$

Welches Ergebnis würde eine Anwendung des Hindley–Milner–Algorithmus auf diese Terme liefern?

4.1-e Zeigen Sie durch Induktion, daß die Church–Numerals ($\overline{n} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\ \mathtt{x}$) für alle $n \in \mathbb{N}$ mit $(\mathtt{X}\to\mathtt{X})\to\mathtt{X}\to\mathtt{X}$ typisiert werden können.

Aufgabe 4.2 (Lehren aus der Leere ziehen)

Der Beweis der starken Normalisierbarkeit typisierbarer λ-Terme setzt implizit voraus, daß Datentypen nicht leer sind.

Nehmen Sie an, es gäbe einen leeren Datentyp $\{\}$ und das explizite Urteil $x \in \{\}$ *gilt niemals*. Zeigen Sie anhand eines Beispiels, daß es nun typisierbare Terme gibt, die nicht stark normalisierbar sind.

Untersuchen Sie anhand dieses Beispiels, an welcher Stelle der Beweis der starken Normalisierbarkeit fehlschlägt, wenn Typisierung mit leeren Datentypen erlaubt ist.

Aufgabe 4.3 (Hindley–Milner Algorithmus in der Praxis)

In vielen funktionalen Programmiersprachen wie ML oder Haskell wird eine erweiterte Version des Hindley–Milner Typechecking Algorithmus eingesetzt um den Datentyp eines gegebenen Ausdrucks der Sprache zu bestimmen.

Wie müsste der Hindley–Milner Algorithmus erweitert werden, wenn neben dem einfachen Funktionenraum auch die folgenden Datenstrukturen zum Typsystem der Sprache gehören.

- Den Typ $\mathbb{B}$ der booleschen Werte zusammen mit den Elementen T und F und der Analyseoperation $\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t$.
- Das Produkt $S \times T$ zweier Datentypen S und T zusammen mit dem Element $\langle s, t\rangle$ (Paarbildung) und der Analyseoperation $\mathsf{match}\ p\ \mathsf{with}\ \langle x, y\rangle \mapsto e$.
- Den Typ $\mathbb{N}$ der natürlichen Zahlen mit Element 0, der Nachfolgeroperation $\mathsf{s}(i)$, den arithmetischen Ausdrücken $i{+}j$, $i{-}j$, $i{*}j$, i/j, $i\ \mathsf{mod}\ j$, und einem induktiven Analyseoperator $\mathsf{PR}[base; h](i)$[1].
- Den Typ $T\ \mathsf{list}$ der Listen über dem Datentyp T zusammen mit Element $[]$, Operation $t{::}l$ (t wird an den Anfang der Liste l gehängt) und einem induktiven Analyseoperator $\mathsf{list\_ind}[base; h](l)$[2].

[1] Oft geschrieben als `f(`i`), where f(x) = if x=0 then` $base$ `else` h`(x,f(x-1))`
[2] Oft geschrieben als `f(`l`), where f(x) = if x=[] then` $base$ `else let x=hd::tl in` h`(hd,tl,f(tl))`