Kapitel 10

Polymorphismus

Einführendes Beispiel: Polymorphe Listen.

- JAVA-Polymorphismus `Object` $\geq A$.

  ```
  class List {
    ...
    int    length();
    void   cons(Object a);
    void   append(List l);
    Object front();
  }
  ```

- Nachteil: Hat man homogene Listen, z.B., `Integer` Listen, muss man trotzdem immer casten: `(Integer)l.front()`.

- F-Polymorphismus in JAVA 1.5 behebt dieses Problem:

  ```
  class List<A> {
    ...
    int    length();
    void   cons(A a);
    void   append(List<A> l);
    A      front();
  }
  ```

- Polymorphismus in Haskell:

  ```
  data List a = Nil
              | Cons a (List a)

  map :: forall a b. (a -> b) -> List a -> List b
  map f Nil        = Nil
  map f (Cons a l) = Cons (f a) (map f l)
  ```

Wir studieren nun den Polymorphismus in Rein-Form, d.h., im λ-Kalkül.

10.1 System F, Curry-Stil

Syntaktische Kategorien.

$$
\begin{array}{llll}
\Lambda \ni r,s,t & ::= & x \mid \lambda x t \mid r\,s & \text{ungetypte Terme} \\
\mathsf{Ty} \ni A,B,C & ::= & A \to B \mid \forall X A \mid X & \text{einfache und polymorphe Typen} \\
\mathsf{Cxt} \ni \Gamma & ::= & \diamond \mid \Gamma, x{:}A \mid \Gamma, X & \text{Typisierungs-Kontexte}
\end{array}
$$

Typisierung.

$$
\text{TY-VAR}\ \frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \text{TY-ABS}\ \frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \lambda x t : A \to B} \qquad \text{TY-APP}\ \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\,s : B}
$$

$$
\text{TY-GEN}\ \frac{\Gamma, X \vdash t : A}{\Gamma \vdash t : \forall X A} \qquad \text{TY-INST}\ \frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[B/X]}
$$

Übung 10.1 *Geben Sie Typisierungsherleitungen für*

1. $\vdash \lambda x x : \forall X.\, X \to X$
2. $\vdash \lambda x.\, x\, x : (\forall X X) \to \forall X X.$

Beispiel 10.2 (Nicht typisierbare Terme) *Die Terme* Ω *und* $\underline{2}\,\underline{2}\,\mathsf{K}$ *sind in System F nicht typisierbar.*

Typisierung ist unentscheidbar (Wells, 1994). Wohlgetypte Terme sind stark normalisierend, jedoch nicht umgekehrt (Beispiel: $\underline{2}\,\underline{2}\,\mathsf{K}$). Es gilt Typerhaltung unter β-Reduktion, jedoch nicht unter η-Reduktion.

Beispiel 10.3 (η-Reduktion erhält Typen nicht) *Sei* $X \notin \mathsf{FV}(A)$. *Folgende Typisierung ist gültig:*

$$y{:}A \to \forall X B \vdash \lambda x.\, y\, x : \forall X.\, A \to B$$

Jedoch hat y *nicht den Typ* $\forall X.\, A \to B$.

10.1.1 Imprädikative Kodierungen

Einige Konstruktionen im ungetypten λ-Kalkül, wie z.B. Church-Ziffern und Tupel, können wir in System F typisieren.

$$
\begin{array}{lcllc}
\mathsf{N} & = & \forall X.\, (X \to X) \to X \to X & & \\
\underline{n} & = & \lambda f x.\, f^n\, x & : & \mathsf{N} \\
\underline{+} & = & \lambda n m f x.\, n\, f\, (m\, f\, x) & : & \mathsf{N} \\
\underline{\cdot} & = & \lambda n m f.\, n\, (m\, f) & : & \mathsf{N} \\
\mathsf{exp} & = & \lambda n m.\, n\, m & : \mathsf{N} &
\end{array}
$$

Übung 10.4 *Programmieren Sie die Ackermann-Funktion in System F!*

Lösung 5 $\mathsf{ack} = \lambda n.\, n\, (\lambda r \lambda m.\, m\, r\, \underline{1})\, (\lambda m.\, m \underline{+1})$.

Im Allgemeinen kann man eine Datenstruktur imprädikativ kodieren, wenn sie einen *universalen Eliminator* hat, d.h., jede Verwendung der Datenstruktur kann mit dem u.E. programmiert werden. Der universale Eliminator für Paare ist split.

$$\begin{array}{lcl} \mathsf{split} & : & \forall AB.\, A \times B \to \forall C.\, (A \to B \to C) \to C \\ \mathsf{split}\, (\mathsf{pair}\, a\, b)\, k & \longrightarrow & k\, a\, b \\ \mathsf{fst} & = & \lambda p.\, \mathsf{split}\, p\, \mathsf{T} \\ \mathsf{snd} & = & \lambda p.\, \mathsf{split}\, p\, \mathsf{F} \end{array}$$

Nun nimmt man als Definition für $A \times B$ den Zieltyp $\forall C.\, (A \to B \to C) \to C$ des universalen Eliminators.

$$\begin{array}{lcl} A \times B & = & \forall C.\, (A \to B \to C) \to C \\[1ex] \mathsf{pair} & : & \forall AB.\, A \to B \to A \times B \\ \mathsf{pair} & = & \lambda ab.\, \lambda k.\, k\, a\, b \\[1ex] \mathsf{split} & : & \forall AB.\, A \times B \to \forall C.\, (A \to B \to C) \to C \\ \mathsf{split} & = & \lambda xx \end{array}$$

Der universale Eliminator für Summentypen ist $\mathsf{case} : \forall AB.\, A + B \to \forall C.\, (A \to C) \to (B \to C) \to C$. Also funktioniert folgende Kodierung.

$$\begin{array}{lcl} A + B & = & \forall C.\, (A \to C) \to (B \to C) \to C \\[1ex] \mathsf{inl} & : & \forall AB.\, A \to A + B \\ \mathsf{inl} & = & \lambda a.\, \lambda f g.\, f\, a \\[1ex] \mathsf{inr} & : & \forall AB.\, B \to A + B \\ \mathsf{inr} & = & \lambda b.\, \lambda f g.\, g\, b \\[1ex] \mathsf{case} & : & \forall AB.\, A + B \to \forall C.\, (A \to C) \to (B \to C) \to C \\ \mathsf{case} & = & \lambda xx \end{array}$$

Der universale Eliminator für Listen ist foldr:

$$\begin{array}{lcl} \mathsf{foldr} & : & \forall A.\, \mathsf{List}\, A \to \forall B.\, (A \to B \to B) \to B \to B \\ \mathsf{foldr}\, [a_1, \ldots, a_n]\, (\otimes)\, e & \longrightarrow & a_1 \otimes (a_2 \otimes \ldots (a_n \otimes e) \ldots) \\ \mathsf{foldr}\, \mathsf{nil}\, (\otimes)\, e & \longrightarrow & e \\ \mathsf{foldr}\, (\mathsf{cons}\, a\, l)\, (\otimes)\, e & \longrightarrow & a \otimes \mathsf{foldr}\, l\, (\otimes)\, e \end{array}$$

Wir kodieren Listen also als

$$
\begin{array}{lcl}
\mathsf{List}\,A & = & \forall B.\,(A \to B \to B) \to B \to B \\[1ex]
\mathsf{nil} & : & \forall A.\,\mathsf{List}\,A \\
\mathsf{nil} & = & \lambda f e.\, e \\[1ex]
\mathsf{cons} & : & \forall A.\,A \to \mathsf{List}\,A \to \mathsf{List}\,A \\
\mathsf{cons} & = & \lambda a l.\,\lambda f e.\, f\,a\,(l\,f\,e) \\[1ex]
\mathsf{foldr} & = & \lambda x x
\end{array}
$$

Übung 10.5 *Implementieren Sie* map *und* append *mit dieser Listen-Kodierung.*

Eine andere Sichtweise auf die imprädikative Kodierung ist: Sei X ein Datentyp mit den Konstruktoren $c_1 : F_1(X) \ldots c_n : F_n(X)$. Dann ist $\forall X.\, F_1(X) \to \cdots \to F_n(X) \to X$ die imprädikative Kodierung von X.

Übung 10.6 *Machen Sie sich diese Sichtweise an den Beispielen* N, $\times$, $+$ *und* $\mathsf{List}\,A$ *klar.*

Übung 10.7 *Kodieren Sie den Typ* $\mathsf{BTree}\,A$ *der Binärbäume mit Knotenbeschriftungen in* A *imprädikativ.*