Funktionales Programmieren

Teil 12

Carl Philipp Reh

Universität Siegen

24. November 2023

Typurteile

Wir wollen nun formal definieren, welche Ausdrücke welche Typen bekommen können. Wir schreiben $\Gamma \vdash e : \tau$ für den Fall, dass $e \in \text{Exp}$ den Typ $\tau \in \text{Type}$ unter der Typumgebung $\Gamma \in \text{TEnv}$ haben kann. Dies nennt man auch ein *Typurteil*.

Analog zu normalen Umgebungen fordern wir, dass jede Typumgebung einen Eintrag für die fundamentalen Operationen wie $(+)$ hat, also zum Beispiel soll für jedes Γ gelten, dass $\Gamma((+)) = \text{Int} \to \text{Int} \to \text{Int}$.

Wir definieren nun $\Gamma \vdash e : \tau$ induktiv, indem wir *Typregeln* angeben. Die Basisfälle sind folgende:

Für $x \in \text{Var}$ gilt $\Gamma \vdash x : \tau$, falls $\Gamma(x) = \tau$.

Für $i \in \mathbb{Z}$ gilt $\Gamma \vdash i : \text{Int}$.

Für einen Wertkonstruktor $\underline{\text{C}}$ vom Typ τ gilt $\Gamma \vdash \underline{\text{C}} : \tau$.

Typregeln

Induktionsschritte schreiben wir als Bruch. Das, was über dem Strich steht, sind die Voraussetzungen, und das, was unter dem Strich steht, kann daraus geschlossen werden.
Für die Applikation fordern wir, dass auf der linken Seite ein passender Funktionstyp steht:

$$\frac{\Gamma \vdash e_1 : \tau' \to \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash (e_1\ e_2) : \tau}$$

Bei **let** $x = e_1$ **in** e_2 müssen wir den Typ für x sowohl in e_1, als auch in e_2 binden. Der Ergebnistyp ist dann der von e_2:

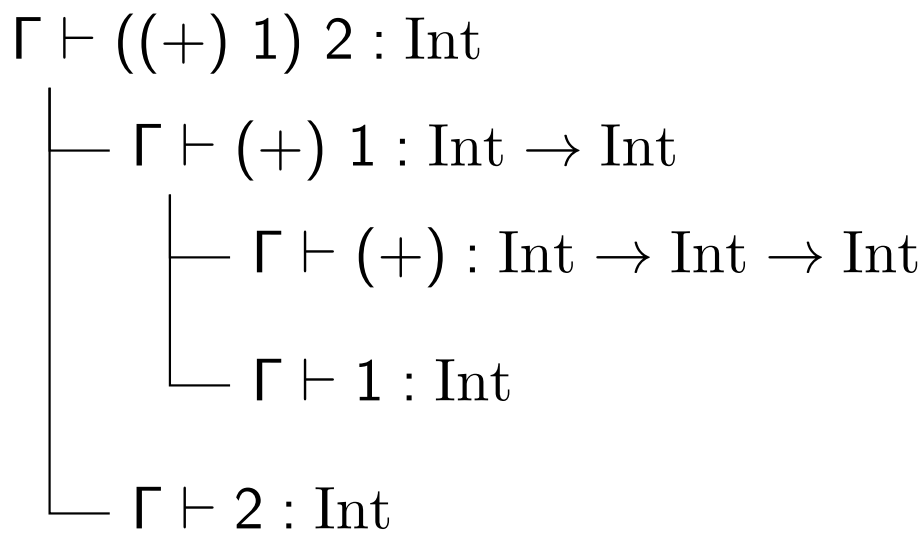
$$\frac{\Gamma \oslash [x/\tau'] \vdash e_1 : \tau' \quad \Gamma \oslash [x/\tau'] \vdash e_2 : \tau}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau}$$

Ein Lambda-Ausdruck erhält einen Funktionstyp bei passendem Parametertyp:

$$\frac{\Gamma \oslash [x/\tau'] \vdash e : \tau}{\Gamma \vdash \backslash x \texttt{->}\ e : \tau' \to \tau}$$

Typregeln

Eine *Typherleitung* ist eine Folge von Typurteilen, die sich aus den Typregeln ergeben. Wir schreiben diese als Baum auf. Beispiel:



Ein Ausdruck $e \in \text{Exp}$, für den es ein $\Gamma \in \text{TEnv}$ und $\tau \in \text{Type}$ gibt mit $\Gamma \vdash e : \tau$, heißt auch *wohlgetypt*.
Nicht wohlgetypte Ausdrücke sind dann solche, für die wir keine Typherleitung finden können. Zum Beispiel ist $(+)\ 1\ \underline{\text{True}}$ nicht wohlgetypt.

Typregeln

Für Case-Ausdrücke betrachten wir folgendes Beispiel:

$$\textbf{case}\ e\ \textbf{of}\ \{\underline{\mathrm{MkPair}}\ x\ y\ {->}\ x + y\},$$

wobei $\underline{\mathrm{MkPair}}\colon \mathrm{Int} \to \mathrm{Int} \to \mathrm{IntPair}$. Wir müssen zum einen prüfen, dass $\Gamma \vdash e : \mathrm{IntPair}$, also dass der Typ von e auf den Ergebnistyp des Konstruktors passt. Zum anderen müssen wir überprüfen, dass $x + y$ unter der Umgebung $\Gamma \oslash [x/\mathrm{Int}, y/\mathrm{Int}]$ wohlgetypt ist. Hier gilt $\Gamma \oslash [x/\mathrm{Int}, y/\mathrm{Int}] \vdash x + y : \mathrm{Int}$.

Wir wollen allgemein definieren, wann Folgendes gilt:

$$\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ \{\underline{\mathrm{C}}\ x_1\ \ \ldots\ \ x_n\ {->}\ e'\} : \tau$$

Dazu muss zunächst $\Gamma \vdash e : \tau'$ gelten. Der Ergebnistyp von $\underline{\mathrm{C}}$ muss zu τ' passen, also $\Gamma \vdash \underline{\mathrm{C}}\colon \tau_1 \to \cdots \to \tau_n \to \tau'$. Außerdem müssen die Paramtertypen von $\underline{\mathrm{C}}$ zu $x_1, \ldots, x_n$ passen, also $\Gamma \oslash [x_1/\tau_1, \ldots, x_n/\tau_n] \vdash e' : \tau$.

Typregel für Case

Bei der allgemeinen Typregel für Case, wo es mehrere Patterns geben kann, müssen wir die Forderung, die wir an einen einzelnen Pattern gestellt haben, entsprechend wiederholen:

$$\frac{\begin{array}{l}\Gamma \vdash e : \tau' \\ \Gamma \vdash \underline{\mathrm{C}_1} : \tau_{1,1} \to \cdots \to \tau_{1,m_1} \to \tau' \\ \Gamma \oslash [x_{1,1}/\tau_{1,1}, \ldots, x_{1,m_1}/\tau_{1,m_1}] \vdash e_1 : \tau \\ \ldots \\ \Gamma \vdash \underline{\mathrm{C}_n} : \tau_{n,1} \to \cdots \to \tau_{n,m_n} \to \tau' \\ \Gamma \oslash [x_{n,1}/\tau_{n,1}, \ldots, x_{n,m_n}/\tau_{n,m_n}] \vdash e_n : \tau\end{array}}{\begin{array}{l}\textbf{case } e \textbf{ of } \{\underline{\mathrm{C}_1}\ x_{1,1}\ \ldots\ x_{1,m_1} -> e_1 \\ \qquad\qquad ; \cdots ; \\ \qquad\qquad \underline{\mathrm{C}_n}\ x_{n,1}\ \ldots\ x_{n,m_n} -> e_n\} : \tau\end{array}}$$

Beispiel für eine Typherleitung

Als Beispiel betrachten wir folgende Typherleitung für einen Ausdruck mit mehreren Patterns:

$$\Gamma \vdash \textbf{case}\ (\underline{\text{Succ}}\ \underline{\text{Zero}})\ \textbf{of}\ \{\underline{\text{Zero}} \mathrel{-\!\!>} \underline{\text{Zero}}\ ;\ \underline{\text{Succ}}\ x \mathrel{-\!\!>} x\} : \text{Nat}$$

- $\Gamma \vdash (\underline{\text{Succ}}\ \underline{\text{Zero}}) : \text{Nat}$
  - $\Gamma \vdash \underline{\text{Succ}} : \text{Nat} \to \text{Nat}$
  - $\Gamma \vdash \underline{\text{Zero}} : \text{Nat}$
- $\Gamma \vdash \underline{\text{Zero}} : \text{Nat}$
- $\Gamma \vdash \underline{\text{Zero}} : \text{Nat}$
- $\Gamma \vdash \underline{\text{Succ}} : \text{Nat} \to \text{Nat}$
- $\Gamma \circledcirc [x/\text{Nat}] \vdash x : \text{Nat}$

Typinferenz

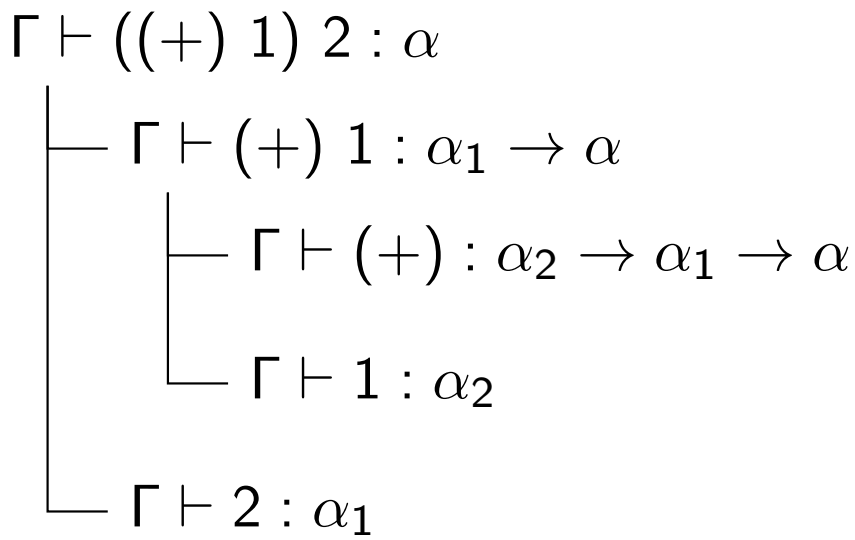
Wir werden nun einen Algorithmus implementieren, der herausfindet, ob $\Gamma \vdash e : \tau$ gilt. Die Typregeln eignen sich nicht direkt dazu, weil man an gewissen Stellen „raten“ muss, zum Beispiel bei der Applikation: Wenn man zeigen will, dass $\Gamma \vdash (e_1\ e_2) : \tau$, muss man zeigen, dass $\Gamma \vdash e_1 : \tau' \to \tau$ und $\Gamma \vdash e_2 : \tau'$, aber für welches τ'?

Wir lösen dieses Problem des „Ratens“, indem wir τ' erst einmal „offenlassen“. Dies geschieht durch Verwenden einer *Typvariable* α, also $\Gamma \vdash e_1 : \alpha \to \tau$. Formal sind die Typvariablen die Menge $\mathrm{TVar} = \{\alpha, \alpha_1, \alpha_2, \dots\}$, wobei wir TVar zu Type hinzunehmen.

Statt zu fragen, ob $\Gamma \vdash e : \tau$ für ein konkretes τ gilt, werden wir auch hier eine Typvariable verwenden, also $\Gamma \vdash e : \alpha$. Das Bestimmen des „besten“ Werts für α nennt man *Typinferenz*.

Typinferenz

Für unser erstes Beispiel sieht das dann so aus:



Unsere bisherigen Typregeln erlauben nur, dass $\Gamma \vdash (+) : \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}$, und $\Gamma \vdash 1 : \mathrm{Int}$ bzw. $\Gamma \vdash 2 : \mathrm{Int}$. Statt diese Regeln zu modifizieren, notieren wir uns die Typgleichungen $\alpha_2 \to \alpha_1 \to \alpha = \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}$, $\alpha_2 = \mathrm{Int}$ und $\alpha_1 = \mathrm{Int}$.
Formal ist eine *Typgleichung* von der Form $\tau = \tau'$, wobei $\tau, \tau' \in$ Type, und ein *Typgleichungssystem* ist eine Menge von Typgleichungen.

Typinferenz

Unsere Aufgabe wird es sein, solch ein Typgleichungssystem zu *lösen*. Intuitiv geschieht dies, indem man Typvariablen so ersetzt, dass alle Typgleichungen von der Form $\tau = \tau$ sind, also zum Beispiel $\alpha = \mathrm{Int}$ wird zu $\mathrm{Int} = \mathrm{Int}$, indem man α durch Int ersetzt. Einen Algorithmus, der die „beste“ Lösung findet, werden wir in der nächsten Vorlesung kennenlernen.

Wenn wir eine Typgleichung $\tau = \tau'$ notieren müssen, schreiben wir dies als $\{\tau = \tau'\}$ mit in die Typregel. Die neuen Typregeln sehen dann folgendermaßen aus:

Für $x \in \mathrm{Var}$ haben wir $\Gamma \vdash x : \tau \ \{\tau = \tau'\}$, falls $\Gamma(x) = \tau'$.

Für $i \in \mathbb{Z}$ haben wir $\Gamma \vdash i : \tau \ \{\tau = \mathrm{Int}\}$.

Für einen Wertkonstruktor $\underline{\mathrm{C}}$ vom Typ τ' haben wir $\Gamma \vdash \underline{\mathrm{C}} : \tau \ \{\tau = \tau'\}$.

Typinferenz

Bei der Applikation ersetzen wir, wie besprochen, den Parameter-Typ durch eine neue Typvariable:

$$\frac{\Gamma \vdash e_1 : \alpha \to \tau \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash (e_1\ e_2) : \tau}$$

Bei Let-Ausdrücken gehen wir analog vor:

$$\frac{\Gamma \oslash [x/\alpha] \vdash e_1 : \alpha \quad \Gamma \oslash [x/\alpha] \vdash e_2 : \tau}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau}$$

Bei Lambda-Ausdrücken können wir nicht mehr fordern, dass der Ergebnistyp ein Funktionstyp ist. Daher fügen wir zwei neue Typvariablen (für den Parameter und für das Ergebnis) ein:

$$\frac{\Gamma \oslash [x/\alpha] \vdash e : \alpha'}{\Gamma \vdash \backslash x \texttt{->}\ e : \tau\ \{\tau = \alpha \to \alpha'\}}$$

Beispiel für Typinferenz

Für unser vorheriges Beispiel sähe eine Typinferenz dann so aus:

$$\Gamma \vdash ((+)\ 1)\ 2 : \alpha$$

- $\Gamma \vdash (+)\ 1 : \alpha_1 \to \alpha$
  - $\Gamma \vdash (+) : \alpha_2 \to \alpha_1 \to \alpha$
    $\{\alpha_2 \to \alpha_1 \to \alpha = \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}\}$
  - $\Gamma \vdash 1 : \alpha_2\ \{\alpha_2 = \mathrm{Int}\}$
- $\Gamma \vdash 2 : \alpha_1\ \{\alpha_1 = \mathrm{Int}\}$

Damit erhalten wir folgendes Typgleichungssystem:

$$\begin{aligned}\{\alpha_2 \to \alpha_1 \to \alpha &= \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int},\\ \alpha_2 &= \mathrm{Int},\\ \alpha_1 &= \mathrm{Int}\}\end{aligned}$$

Typinferenz

Bei Case-Ausdrücken ersetzen wir einfach alle Typen, die wir bisher „raten“ mussten, durch neue Typvariablen:

$$\frac{\begin{array}{l}\Gamma \vdash e : \alpha \\ \Gamma \vdash \underline{\mathrm{C}_1} : \alpha_{1,1} \to \cdots \to \alpha_{1,m_1} \to \alpha \\ \Gamma \oslash [x_{1,1}/\alpha_{1,1}, \ldots, x_{1,m_1}/\alpha_{1,m_1}] \vdash e_1 : \tau \\ \ldots \\ \Gamma \vdash \underline{\mathrm{C}_n} : \alpha_{n,1} \to \cdots \to \alpha_{n,m_n} \to \alpha \\ \Gamma \oslash [x_{n,1}/\alpha_{n,1}, \ldots, x_{n,m_n}/\alpha_{n,m_n}] \vdash e_n : \tau \end{array}}{\begin{array}{l}\textbf{case } e \textbf{ of } \{\underline{\mathrm{C}_1}\ x_{1,1}\ \ldots\ x_{1,m_1} \mathrel{->} e_1 \\ \qquad ; \ldots ; \\ \qquad \underline{\mathrm{C}_n}\ x_{n,1}\ \ldots\ x_{n,m_n} \mathrel{->} e_n\} : \tau\end{array}}$$

Weiteres Beispiel für Typinferenz

$\Gamma \vdash \textbf{case}\ (\underline{\text{Succ}}\ \underline{\text{Zero}})\ \textbf{of}\ \{\underline{\text{Zero}} -> \underline{\text{Zero}}\ ;\ \underline{\text{Succ}}\ x -> x\} : \alpha$

- $\Gamma \vdash (\underline{\text{Succ}}\ \underline{\text{Zero}}) : \alpha_1$
  - $\Gamma \vdash \underline{\text{Succ}} : \alpha_2 \to \alpha_1\ \{\alpha_2 \to \alpha_1 = \text{Nat} \to \text{Nat}\}$
  - $\Gamma \vdash \underline{\text{Zero}} : \alpha_2\ \{\alpha_2 = \text{Nat}\}$
- $\Gamma \vdash \underline{\text{Zero}} : \alpha_1\ \{\alpha_1 = \text{Nat}\}$
- $\Gamma \vdash \underline{\text{Zero}} : \alpha\ \{\alpha = \text{Nat}\}$
- $\Gamma \vdash \underline{\text{Succ}} : \alpha_3 \to \alpha_1\ \{\alpha_3 \to \alpha_1 = \text{Nat} \to \text{Nat}\}$
- $\Gamma \oslash [x/\alpha_3] \vdash x : \alpha\ \{\alpha_3 = \alpha\}$

Wir erhalten damit das Typgleichungssystem:

$$\{\alpha_2 \to \alpha_1 = \text{Nat} \to \text{Nat},\ \alpha_2 = \text{Nat},\ \alpha_1 = \text{Nat},$$
$$\alpha = \text{Nat},\ \alpha_3 \to \alpha_1 = \text{Nat} \to \text{Nat},\ \alpha_3 = \alpha\}$$