Material zur Vorlesung “Semantik von Programmiersprachen”

Prof. Gert Smolka und Martin Müller

1 Eine getypte funktionale Programmiersprache

TF ist die Einschränkung von F auf die sogenannten *wohlgetypten* Terme, die wir im folgenden definieren.

Typen. Wir nehmen eine unendliche Menge **TVar** von Typvariablen an, die wir als X, Y, Z schreiben. Die Typen von TF sind durch die folgende Grammatik definiert.

$$\tau \ ::= \ \mathbf{int} \mid \mathbf{bool} \mid \mathbf{unit} \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2 \mid X \mid \mu X.\tau$$

Die Menge der geschlossenen Typen (ohne freie Typvariablen) schreiben wir als **GTyp**.

Typregeln. Wir nehmen eine Abbildung $typ : \mathbf{Var} \to \mathbf{GTyp}$ an, die jeder Variablen einen eindeutigen Typ zuordnet.
Wir definieren eine Relation $: \subseteq \mathbf{Ter} \times \mathbf{GTyp}$ durch ein Regelsystem. Wenn $(t, \tau) \in :$ ist, schreiben wir $t : \tau$ und lesen “der Term t hat Typ τ”. Ein Term $t \in \mathbf{Ter}$ heißt wohlgetypt, wenn es $\tau \in \mathbf{GTyp}$ gibt, so daß $t : \tau$.

(int) $\quad n : \mathbf{int} \qquad n \in \mathbf{N}$

(var) $\quad x : \tau \qquad typ(x) = \tau$

(+) $\quad \dfrac{t_1 : \mathbf{int} \qquad t_2 : \mathbf{int}}{t_1 + t_2 : \mathbf{int}} \qquad (\times, -)$

(bool) $\quad \mathbf{true} : \mathbf{bool} \qquad \mathbf{false} : \mathbf{bool}$

($\leq$) $\quad \dfrac{t_1 : \mathbf{int} \qquad t_2 : \mathbf{int}}{t_1 \leq t_2 : \mathbf{bool}}$

(if) $\quad \dfrac{t_0 : \mathbf{bool} \qquad t_1 : \tau \qquad t_2 : \tau}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : \tau}$

(unit) $\quad () : \mathbf{unit}$

(,) $\quad \dfrac{t_1 : \tau_1 \qquad t_2 : \tau_1}{t_1, t_2 : \tau_1 \times \tau_2}$

(fst) $\quad \dfrac{t : \tau_1 \times \tau_2}{\mathbf{fst}\ t : \tau_1} \qquad$ (snd)

(inl) $\quad \dfrac{t : \tau_1}{\mathbf{inl}\ t : \tau_1 + \tau_2} \qquad$ (inr)

(case) $\quad \dfrac{t : \tau_1 + \tau_2 \qquad t_1 : \tau \qquad t_2 : \tau}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}\ x_1.t_1 \ [] \ \mathbf{inr}\ x_2.t_2 : \tau} \qquad typ(x_1) = \tau_1, typ(x_2) = \tau_2$

(lam) $\quad \dfrac{x : \tau_1 \qquad t : \tau_2}{\lambda x.t : \tau_1 \to \tau_2}$

(appl) $\quad \dfrac{t_1 : \tau_1 \to \tau_2 \qquad t_2 : \tau_1}{t_1\ t_2 : \tau_2}$

(rec) $\quad \dfrac{y : \tau \qquad \lambda x.t : \tau}{\mathbf{rec}\ y.\lambda x.t : \tau}$

$$(\mu\text{-}1) \quad \frac{t : \tau[\mu X.\tau/X]}{t : \mu X.\tau} \qquad (\mu\text{-}2) \quad \frac{t : \mu X.\tau}{t : \tau[\mu X.\tau/X]}$$

Proposition 1.1 (Typerhaltung) *Seien* $t, t' \in$ **GTer**. *Dann gilt*

$$\left.\begin{array}{l} t : \tau \\ t \rightarrow_1 t' \end{array}\right\} \Rightarrow t' : \tau$$

Beweis. Einfach. □

Proposition 1.2 (Freiheit von Typfehlern) *Sei* $t \in$ **GTer**. *Dann gilt*

$$\left.\begin{array}{l} t : \tau \\ t \textit{ nicht kanonisch} \end{array}\right\} \Rightarrow \exists t' : t \rightarrow_1 t'$$

Beweis. Einfach. □

2 Einfach getyptes $\mathbf{F}_0$

Wir betrachten die Menge $\mathbf{Typ}_0 \subseteq \mathbf{Typ}$, die durch die folgende Grammatik definiert ist.

$$\tau \;::=\; \mathbf{unit} \mid \tau_1 \rightarrow \tau_2$$

Die Elemente von $\mathbf{Typ}_0$ heißen auch *einfache Typen*. Jeder einfache Typ ist offenbar geschlossen. TF_0 ist die Einschränkung von F_0 auf wohlgetypte Terme über den Typen aus $\mathbf{Typ}_0$. Das Typsystem von TF_0 ist durch die folgenden Regeln definiert.

$$\begin{array}{ll} (\text{var}) & x : \tau \qquad typ(x) = \tau \\ (\text{lam}) & \dfrac{x : \tau_1 \qquad t : \tau_2}{\lambda x.t : \tau_1 \rightarrow \tau_2} \\ (\text{appl}) & \dfrac{t_1 : \tau_1 \rightarrow \tau_2 \qquad t_2 : \tau_1}{t_1\ t_2 : \tau_2} \end{array}$$

Satz. (Keine Divergenz) Sei $t \in \mathbf{GTer}_0$. Dann gilt

$$t : \tau \;\Rightarrow\; \exists c : t \rightarrow c$$

Beweis. Nicht-trivial (Siehe zum Beispiel [2, 3, 1]). □

Literatur

[1] H. P. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 118–310. Oxford University Press, 1992.

[2] W. W. Tait. Intensional Interpretation of Functionals of Finite Type. *Journal of Symbolic Logic*, (32), 1967.

[3] S. Thompson. *Type Theory and Functional Programming*. IEEE Computer Society Press, University of Kent at Canterbury, 1991.