22.11.05

Theorie der Programmierung I

(Mitschrift von Lars Friedrich email@lars-friedrich-home.de)

Nachtrag: Man schreibt: $e :: \tau$, wenn $[] \triangleright e :: \tau$ und sagt: „e hat Typ τ“

Satz 6: (Eindeutigkeit des Typs)
Für jede Typumgebung Γ und jeden Ausdruck e existiert höchstens ein Typ τ mit $\Gamma \triangleright e :: \tau$

Beweis: siehe Übung (Induktion über die Größe von e)

Idee: In jeder Situation ist nur eine Typregel rückwärts anwendbar. Jede Typregel führt zu eindeutigen Prämissen $\hookrightarrow$ Induktionsannahme anwendbar.

Folgerung: Sei TEnv die Menge aller Typumgebungen. Dann kann man eine Funktion type: TEnv x Exp $\hookrightarrow$ Type definieren durch:

$$\text{type}(\Gamma, e) = \begin{cases} \tau, \text{ falls } \Gamma \triangleright e :: \tau \\ \text{undefiniert, falls kein solches } \tau \text{ existiert.} \end{cases}$$

Aus den Typregeln erhält man leicht einen Algorithmus zur Berechnung dieser Funktion:

$\text{type}(\Gamma, c) =$ der eindeutige Typ von c

$$\text{type}(\Gamma, id) = \begin{cases} \Gamma(id), \text{ falls } id \in \text{dom}(\Gamma) \\ \text{Undefiniert, sonst} \end{cases}$$

$$\text{type}(\Gamma, e_1e_2) = \begin{cases} \tau', \text{ wenn type } (\Gamma, e_1) = \tau \to \tau' \text{ und type } (\Gamma, e_2) = \tau \\ \text{undefiniert, sonst, d.h. falls type } (\Gamma, e_1) \text{ kein Funktionstyp oder type } (\Gamma, e_2) \text{ stimmt nicht mit dem Typ von type } (\Gamma, e_1) \text{ überein, oder einer der beiden undefiniert} \end{cases}$$

$$\text{type}(\Gamma, \lambda id.\tau\ e) = \begin{cases} \tau \to \tau, \text{ falls type } (\Gamma\ [\tau/id]\ e) = \tau' \\ \text{undefiniert, falls type } (\Gamma\ [\tau/id]\ e) \text{ undefiniert} \end{cases}$$

usw.
Das kann man leicht in ein Programm umsetzen.

Begriff: **Typüberprüfungs-(type checking)Algorithmus**

Laufzeit: linear in der Größe von e

Weitere Eigenschaften des Typsystems

Lemma 7: (Frei vorkommende Namen und Typurteile)
(a) Wenn $\Gamma \triangleright e :: \tau$, dann $\text{free}(e) \subseteq \text{dom}(\Gamma)$
(b) Wenn Γ, Γ' auf free(e) definiert sind und übereinstimmen, dann $\Gamma \triangleright e :: \tau \Leftrightarrow \Gamma' \triangleright e :: \tau$
(c) Wenn Γ' auf free(e) definiert ist und Γ' ist die Einschränkung auf free(e), dann gilt $\Gamma \triangleright e :: \tau \Leftrightarrow \Gamma' \triangleright e :: \tau$

Beweis: jeweils über die Größe von e

Intuition: (a) bedeutet, dass man die Typen der in e frei vrokommenden Namen kennen muss
(b) bedeutet, dass man nur die Typen der in e frei vrokommenden Namen kennen muss, d.h. Typen anderer Namen spielen keine Rolle

Lemma 8: (Frei vorkommende Namen und Substitution)
$\text{free}(e'[e/id]) \subseteq (\text{free}(e') \setminus \{id\}) \cup \text{free}(e)$

Intuitive Begründung: Durch die Substitution wird jeder freie Vorkommen von id durch e ersetzt, also verschwindet id und die frei vorkommenden Namen von e kommen hinzu

Ausnahme: $id \notin \text{free}(e')$, dann wird nichts ersetzt, deshalb gilt nur „$\subseteq$“ und nicht „=“.

Beweis: durch Induktion über die Größe von e

Lemma 9: (Substitution und Typurteile)

Wenn $\Gamma \triangleright e :: \tau$ und $\Gamma\ [\tau/id] \triangleright e' :: \tau'$ dann gilt: $\Gamma \triangleright e'[e/id] :: \tau'$

In Worten: Wenn e' vom Typ τ' ist, unter der Annahme, dass id vom Typ τ ist, dann kann man für id einen Ausdruck e von Typ τ einsetzen, ohne dass sich der Typ von e verändert.

Intuitive Begründung: Man betrachtet eine Herleitung für $\Gamma\ [\tau/id] \triangleright e' :: \tau'$. In dieser Herleitung wird bei jedem freien Vorkommen von id in e' Regel (ID) angewandt und führt zu $\Gamma\ [\tau/id] \triangleright id :: \tau$. Wenn wir an diesen Stellen jeweils die Herleitung für $\Gamma \triangleright e :: \tau$ einsetzen, so erhalten wir eine Herleitung für $\Gamma \triangleright e'\ [e/id] :: \tau'$. Genauer: Anstelle von Γ tritt evtl. eine Erweiterung Γ' von Γ. Die gebundene Umgebung bei der Substitution sorgt dafür, dass trotzdem alles gut geht.

Ziel: Beweis, dass für Berechnungen für wohlgetypte Ausdrücke nicht stecken bleiben.

1. Schritt: Bei jedem small step bleibt der Typ des Ausdrucks erhalten.

Dazu: „small steps für Ausdrücke von $\mathcal{L}^2$ definieren“

→ Die Regeln (BETA-V) und (UNFOLD) werden auf Ausdrücke mit Typinformationen erweitert.

(BETA-V): $(\lambda id{:}\tau.e) \rightarrow e[v/id]$

(UNFOLD): $\text{rec id: } \tau e \rightarrow e[\text{rec id: } \tau e/id]$

Satz 10: (Typerhaltungssatz, „Preservation”)

Wenn $\Gamma \triangleright e :: \tau$ und $e \rightarrow e'$, dann gilt $\Gamma \triangleright e' :: \tau$ (d.h. Wohlgetyptheit und der Typ eines Ausdrucks bleiben bei jedem small step erhalten)

Beweis: Induktion über die Größe von e und Fallunterscheidung nach der small step Regel

1. Fall: $op\ n_1 n_2 \rightarrow op^I(n_1, n_2)$ wegen (OP)
| Exp

Wenn $\Gamma \triangleright op\ n_1, n_2 :: int$, dann ist op arithmetischer Operator und $op^I(n_1, n_2) \in Int$.

Wenn $\Gamma \triangleright op\ n_1, n_2 :: bool$, dann ist op Vergleichsoperator und $op^I(n_1, n_2) \in Bool$.

2. Fall: $(\lambda id{:}\ \tau'.e)\ v \rightarrow e[v/id]$ wegen (BETA-V)

Wenn $\Gamma \triangleright (\lambda id{:}\ \tau'.e)\ v :: \tau$ gilt, dann kann es nur mit Typregel (APP) entstanden sein aus:

1) $\Gamma \triangleright \lambda id{:}\ \tau'.e :: \tau' \rightarrow \tau$
2) $\Gamma \triangleright v :: \tau'$

(1) kann nur mit (ABSTR) entstanden sein aus

3) $\Gamma\ [\tau'/id] \triangleright e :: \tau$
4) Aus (2) und (3) folgt nach Lemma 9: $\Gamma\ [v/id] :: \tau$

3. Fall: $e_1 e_2 \rightarrow e_1' e_2 :: \tau$ mit (APP-LEFT) aus $e_1 \rightarrow e_1'$

Wenn $\Gamma \triangleright e_1 e_2 :: \tau$, dann kann es nur mit (APP) entstanden sein aus:
1) $\Gamma \triangleright e_1 :: \tau' \rightarrow \tau$ und 2) $\Gamma \triangleright e_2 :: \tau'$

Nach Induktionsannahme folgt aus (1): $\Gamma \triangleright e_1' :: \tau' \rightarrow \tau$. Also folgt mit (APP): $\Gamma \triangleright e_1' e_2 :: \tau$

usw.

Satz 10 besagt schon, dass kein „Laufzeittypfehler“ auftreten kann. Noch z.z.: Es gibt auch keine andere Möglichkeit des „Steckenbleibens“.

Satz 11: (Existenz des Übergangschrittes, „Progress“)
Sei e :: τ (d.h. e abgeschlossener Typ von τ), dann gilt: Entweder e $\in$ Val oder es existiert ein small step e$\rightarrow$v mit v $\in$ Val$\cup$Exn.

Beweis: Induktion über die Größe von e und Fallunterscheidung über die Form von e

1. Fall: e = e_1e_2. Wenn e_1e_2 :: τ, dann kann das nur mit (APP) folgen aus:
1) e_1 :: τ'$\rightarrow\tau$ und 2) e_2 :: τ'

- Wenn $e_1 \notin$ Val, dann gilt Induktionsannahme wegen (1) $e_1\rightarrow e_1$' oder $e_1\rightarrow$exn. Also folgt mit (APP-LEFT) oder (APP-LEFT-EXN): $e_1e_2\rightarrow e_1'e_2$ oder $e_1e_2\rightarrow$exn.
- Wenn $e_1 \in$ Val und $e_2 \notin$ Val, analog mit (APP-RIGHT)
- Wenn $e_1=v_1$ und $e_2=v_2 \in$ Val, dann gibt es folgende Möglichkeiten für v_1 :: $\tau'\rightarrow\tau$:
  - v_1 = op n_1 :: int$\rightarrow\tau$ mit $\tau\in${Int, Bool}. Dann muss v_2 :: int gelten, d.h. $v_2=n_2 \in$ Int, also gilt (OP)
  - v_1 = op n_1 :: int$\rightarrow$int$\rightarrow\tau$. Dann muss wieder v_2=n $\in$ Int sein, v_1v_2=op n $\in$ Val. Widerspruch zur Vorraussetzung e $\notin$ Val.
  - λ id:: τ.e'. Dann gilt $v_1v_2=(\lambda$ id:: τ.e')$v_2\rightarrow^{(BETA\text{-}V)}$ e'[v/id]
  - usw.