06.12.05

Theorie der Programmierung I

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

Beweis: (b) Wenn unify(E) eine Substitution s liefert, dann ist s Lösung von E. Induktion über die Rekursionstiefe

1.) unify(∅) = [], klar, da jede Substitution Lösung von E = ∅

2.) unify($\{\tau=\tau\}\cup E$) = unify(E) liefert Substitution s, wenn unify(E) die Substitution s liefert. Nach Induktionsannahme ist s dann Lösung von E, also auch von $\{\tau=\tau\}\cup E$

3.) unify($\{\tau_1\rightarrow\tau_2=\tau_1'\rightarrow\tau_2'\}\cup E$) = unify($\{\tau_1\rightarrow\tau_1'=\tau_2\rightarrow\tau_2'\}\cup E$), nach Induktionsannahme ist die gelierferte Substitution s Lösung von $\tau_1=\tau_1'$, $\tau_2=\tau_2'$ und E, also auch Lösung von $\tau_1\rightarrow\tau_2=\tau_1'\rightarrow\tau_2'$ und E.

4.) $\text{unify}(\{\alpha=\tau\}\cup E) = \begin{cases} s_1s_2 \text{ mit } s_1=[\tau/\alpha] \text{ und } s_2=\text{unify}(Es_1), \text{ falls } \alpha\notin \text{tvar}(\tau) \\ \text{nicht unifizierbar, sonst} \end{cases}$

Es wird nur dann eine Substitution geliefert (nämlich s_1s_2), wenn $\alpha\notin\text{tvar}(\tau)$ und wenn unify(Es_1) die Substitution s_2 liefert. Nach Induktionsannahme ist s_2 dann Lösung von Es_1, d.h. $(Es_1)s_2$ (=$E(s_1s_2)$) besteht nur aus trivialen Gleichungen. Also ist s_1s_2 Lösung von E. Andererseits gilt:

- $\alpha s_1=s_1(\alpha)=\tau$
- $\tau s_1=\tau$ weil $\alpha\notin\text{tvar}(\tau)$
- und $s_1(\alpha') = \alpha'\ \forall\ \alpha'\neq\alpha$

d.h., s_1 löst auch $\alpha=\tau$ ➔ Erst recht ist s_1s_2 Lösung von $\alpha=\tau$ (denn die Gleichung, die schon erfüllt ist, kann durch eine weitere Substitution nicht ungültig werden). Also ist s_1s_2 Lösung von $\{\alpha=\tau\}\cup E$

5.) nichts zu zeigen (da keine Substitution zurückgeliefert wird)

c) zu zeigen: Wenn E eine Lösung s' hat, dann liefert unify(E) eine Substitution s mit $s\sqsubseteq s'$. Induktion über die Rekursionstiefe

1.) Klar, da [] die allgemeinste Substitution von allen ist (d.h. $[]\sqsubseteq s'$ für alle s')

2.) Wenn $\{\tau=\tau\}\cup E$ eine Lösung s' hat, dann ist s' Lösung von E, also gilt nach Induktionsannahme: unify(E) liefert $s\sqsubseteq s'$

3.) Wenn $\{\tau_1\rightarrow\tau_2=\tau_1'\rightarrow\tau_2'\}\cup E$ eine Lösung s' hat, dann ist s' auch Lösung von $\{\tau_1\rightarrow\tau_2=\tau_1'\rightarrow\tau_2'\}\cup E$, also gilt nach Induktionsannahme: Der rekursive Aufruf liefert ein $s\sqsubseteq s'$.

4.) Wenn $\{\alpha=\tau\}\cup E$ (bzw. $\{\tau=\alpha\}\cup E$) eine Lösung s' hat, dann ist s' Lösung von E und es gilt $s'(\alpha)=\tau s'$. Sei $s_1=[\tau/\alpha]$. Dann ist s' auch Lösung von Es_1, dann: wegen $s'(\alpha)=\tau s'$ gilt $Es'=(Es_1)s'$, d.h., wenn Es' nur aus trivialen Gleichungen besteht, dann gilt das auch für $(Es_1)s'$. Also gilt nach Induktionsannahme unify(Es_1) liefert $s_2\sqsubseteq s'$- Es bleibt zu zeigen, dass auch s_1s_2Es' gilt. s_2Es' bedeutet, es existiert ein s'' mit $s_2s''=s'$. Es folgt: $s_1s_2s''=s_1s'$. Also genügt zu zeigen: $s_1s'=s'$. Es gilt:

I. $(s_1s')(\alpha)=(s_1(\alpha))s'=\tau s'=s'(\alpha)$

II. für $\alpha'=\alpha$: $(s_1s')(\alpha')=(s_1(\alpha'))s'=\alpha's'=s'(\alpha')$.

Aus I. und II. ➔ $s_1s'=s'$

Noch zu zeigen: Wenn $\alpha\in\text{tvar}(\tau)$, dann hat $\{\alpha=\tau\}\cup E$ bzw. $\{\tau=\alpha\}\cup E$ <u>keine</u> Lösung. Angenommen es existiert eine Lösung s': Dann muss $s'(\alpha)=\tau s'$ gelten, aber da α echter Teiltyp von τ' ist, muss auch $s'(\alpha)$ echter Teiltyp von $\tau s'$ sein.

5.) Es ist zu zeigen, dass in allen anderen Fällen $\{\tau_1=\tau_2\}\cup E$ <u>keine</u> Lösung besitzt (➔ „nicht unifizierbar“ ist die richtige Antwort). Da Fall (3) nicht vorliegt, sind τ_1, τ_2 nicht beide Funktionstypen. Da Fall (4) nicht vorliegt, ist weder τ_1 noch τ_2 eine Typvariable. Also ist mindestens einer der beiden Typen weder Typvariable noch Funktionstyp, d.h. er muss Basistyp sein. Also müssten τ_1 und τ_2 syntaktisch gleich sein ➔ bereits Fall (2).

Nachtrag: **let-Ausdrücke**

Die Definition von tequs(Γ, e, α) wird auf let-Ausdrücke erweitert durch
tequs(Γ, let id=e_1 in e_2, α)=tequs(Γ, e_1, α_1)$\cup$tequs(Γ,e_2, α_2)$\cup\{\alpha=\alpha_2\}$

Bemerkung: let id = e1 in e2 lässt sich jetzt wieder (wie in $\mathcal{L}_2$, im Gegensatz zu $\mathcal{L}_2^t$) als syntaktischer Zucker für (λid. e_2) e_1 auffassen. Insbesondere lässt sich die Typregel (LET) mit Hilfe von (ABSTR') ableiten.

Der Typinferenzalgorithmus

Eingabe:
- Typumgebung Γ (evtl. mit Typvariablen)
- Ausdruck $e\in\mathcal{L}_2^t$ (d.h. ohne Typinformation)

Ablauf des Algorithmus:

1. Phase: Aufstellen der typgleichungen tequs(Γ, e, α), wobei α neue Typvariable ist (d.h. α kommt nicht in Γ vor)

2. Phase: Lösen der Typgleichungen mit dem Unifikationsalgorithmus ➔ liefert eine Substitution s, insbesondere einen Typ s(α)

Was erwarten wir von der Lösung s?

- Wenn Γ keine Typvariablen enthält (z.B. Γ=[]), dann sollte τ=s(α) allgemeinster Typ („primipal Type“) von e bezüglich Γ sein, d.h.
  - τ ist ein möglicher Typ von e bezüglich Γ, d.h. $\Gamma \triangleright e :: \tau$ ist gültiges Typurteil
  - τ ist allgemeiner als jeder andere mögliche Typ τ', d.h. wenn $\Gamma \triangleright e :: \tau'$ gültig ist, dann existiert eine Substitution s mit $\tau'=\tau s$
- Wenn Γ Typvariablen enthält, dann muss man auch die „Wirkung“ von s auf Γ berücksichtigen ➔ genauere Formulierung später

Schreibweisen:

Sei Γ Typumgebung, s Substitution

(a) tvar(Γ) = $\cup_{id=dom(\Gamma)}$ (Γ(id))
(b) Γs ist die Typumgebung mit: dom(Γs)=dom(Γ) und (Γs)(id)=(Γ(id))s $\forall$ id $\in$ dom(Γ)

Bsp.: Γ=[x: α, f: $\alpha\to\alpha$]
s=[int/α]
Γs=[x: int, f: int$\to$int]

Definition: (a) eine Substitution s ist **Lösung** von (Γ, e, α), wenn $\Gamma s \triangleright e :: s(\alpha)$ gültig ist
(b) s heißt **allgemeinste Lösung** von (Γ, e, α), wenn:
- s Lösung ist
- $s\sqsubseteq s'$ für jede andere Lösung s' gilt