08.12.05

Theorie der Programmierung I

(Mitschrift von Simon Schöling)

Definition: s heißt Lösung für (Γ, e, α), wenn $\Gamma s \triangleright e :: s(\alpha)$ gültig ist.

Lemma 15: Für alle Γ, e, α gilt:

(a) Jede Lösung s dieser tequs (Γ, e, α) ist auch eine Lösung für (Γ, e, α), d.h. $\Gamma s \triangleright e :: s(\alpha)$

(b) Wenn s eine Lösung für (Γ, e, α) ist, dann existiert eine Lösung s' von tequs (Γ, α, α), die sich von s nur auf den neuen Typvariablen der Gleichungen tequs (Γ, e, α) unterscheidet

Beweis: (a) Induktion über die Größe von e

e = c: tequs (Γ, c, α) = {α=τ} wpbei e :: Γ

Für eine Lösung der Gleichung gilt s(α) = τs=τ. Also gilt c :: s(α) und damit $\Gamma s \triangleright c :: s(\alpha)$ nach Typregel (CONST) | da tvar(τ)=∅

e = id: tequs (Γ, id, α) = (α = Γ(id))

Für eine Lösung s gilt s(α) = (Γ(id)s = (Γs)(id)). Also mit Typregel (ID): $\Gamma s \triangleright id :: s(\alpha)$

$e=e_1e_2$: tequs (Γ, e_1e_2, α) = tequs (Γ, e_1, α_1)∪tequs (Γ, e_2, α_2)∪{$\alpha_1=\alpha_2\rightarrow\alpha$}

Für jede Lösung s gilt: $s(\alpha_1) = s(\alpha_2) \rightarrow s(\alpha)$ und s ist Lösung der beiden Teil-Gleichungssysteme. Also gilt nach Induktionsannahme $\Gamma s \triangleright e_1 :: s(\alpha_1)$ und | = $s(\alpha_2)\rightarrow s(\alpha)$

$\Gamma s \triangleright e_2 :: s(\alpha_2)$ und damit folgt mit Typregel (APP): $\Gamma s \triangleright e1e2 :: s(\alpha)$

$e=\lambda id.e_1$: tequs (Γ, $\lambda id.e_1$, α) = tequs (Γ[α_0/id]), e_1, α_1)∪{α= $\alpha_0\rightarrow\alpha_1$}

Für eine Lösung s gilt: $s(\alpha) = s(\alpha_0)\rightarrow s(\alpha_1)$ und s ist Lösung von tequs (Γ[α_0/id]), e_1, α_1). Also gilt nach Induktionsannahme: $(\Gamma[\alpha_0/id])s \triangleright e_1 :: s(\alpha_1)$ und mit Typregel (ABSRTR') folgt:

| (Γs) [$s(\alpha_0)$/id]

$\Gamma s\ \lambda id.e_1 :: s(\alpha_0)\rightarrow s(\alpha_1)$ [=s(α)]

usw…

(b) Induktion über die Größe von e:

e = c: Wenn $\Gamma s \triangleright c :: s(\alpha)$ gültig ist, dann kann das nur mit (CONST) aus c :: s(α) folgen, d.h. es muss s(α)= τ sein, wobei τ der eindeutige Typ mit c::τ ist. Wegen tvar(τ) = ∅ folgt s(α)=τ=τs, d.h. s ist Lösung von {α=τ} 0 tequs (Γ, c, α)

e = id: $\Gamma s \triangleright id :: s(\alpha)$ kann nur gültig sein, wenn (Γs)(id) = s(α) ist (wegen Typregel (ID)) | (Γ(id)s)

Also ist s Lösung von {α=Γ(id)} = tequs (Γ, id, α)

$e = e_1e_2$: $\Gamma s \triangleright e_1e_2 :: s(\alpha)$ kann nur mit (APP) aus den Typurteilen $\Gamma s \triangleright e_1 :: \tau \rightarrow s(\alpha)$ und $\Gamma s \triangleright e_2 :: \tau$ folgen

Per Definition ist tequs (Γ, e_1e_2, α) = tequs (Γ, e_1, α_1) ∪ tequs (Γ, e_2, α_2) ∪ {$\alpha_1 = \alpha_2\rightarrow\alpha$} mit neuen Typvariablen α_1, α_2. Sei $s_1 = s[\tau\rightarrow s(\alpha)/\alpha_1]$. Dann gilt $\Gamma s_1 = \Gamma s$, weil s_1 und s auf tvar(Γ) übereinstimmen, also folgt $\Gamma s_1 \triangleright e_1 :: \tau\rightarrow s(\alpha)$ / $s_1'(\alpha_1)$. Nach Induktionsannahme existiert dann eine Lösung s_1' von tequs(Γ, e_1, α_1), die sich von s_1 nur in den neuen Typvariablen unterscheidet. Sei weiter $s_2 = s[\tau/\alpha_2]$. Dann folgt mit gleicher Argumentation: Es existiert eine Lösung s_2' von tequs(Γ, e_2, α_2), die sich von s_2 nur in den neuen Typvariablen unterscheidet. s_1' und s_2' lassen sich zu einer Substitution s'

„verschmelzen“, die – Lösung des gesamten Gleichungssystem – sich von s nur auf den neun Typvariablen unterscheidet.
usw…

Definition: Sei $T \subseteq TVar$ und $s: TVar \rightarrow Type$ eine Substitution. Dann sei $s \mid_T$ die Substitution mit:
$(s \mid_T)(\alpha) = s(\alpha)$ für alle $\alpha \in T$
$(s \mid_T)(\alpha) = \alpha$ für alle $\alpha \notin T$
insbesondere für $T = \{\alpha_1, \ldots, \alpha_n\}$ gilt: $s \mid_T = [s(\alpha_1)/\alpha_1, \ldots, s(\alpha_n)/\alpha_n]$

Satz 16 (Korrektheit des Typinferrenzalgorithmus)
Bei Eingabe (Γ, e) liefert der Typinferrentalgorithmus eine Substitution s, deren „Einschränkung“ $s \mid_{tvar(\Gamma)\cup\{\alpha\}}$ eine allgemeinste Lösung für (Γ, e, α) ist. Folgerung: Wenn $tvar(\Gamma)=\emptyset$, dann ist $s(\alpha)$ ein allgemeinster Typ von e bezüglich Γ.

Beweis: Nach Satz 11 liefert der Unifikationsalgorithmus eine allgemeinste Lösung s von $tequs(\Gamma, e, \alpha)$. Nach Lemma 15(a) ist s dann eine Lösung für (Γ, e, α), also ist auch $s \mid_{tvar(\Gamma)\cup\{\alpha\}}$ eine Lösung für (Γ, e, α). Bleibt zu zeigen: $s \mid_{tvar(\Gamma)\cup\{\alpha\}}$ ist allgemeiner als jede andere Lösung.
Sei s’ Lösung für (Γ, e, α). Dann existiert nach Lemma 15(b) eine Lösung s’’ von $tequs(\Gamma, e, \alpha)$, die sich von s’ nur auf den neuen Typvariablen unterscheidet. Da s allgemeinste Lösung von $tequs(\Gamma, e, \alpha)$ ist, gilt $s \sqsubseteq s''$. Da sich s’ und s’’ nur auf den neuen Typvariablen unterscheiden und $s \mid_{tvar(\Gamma)\sqsubseteq\{\alpha\}}$ auf den neuen Typvariablen „so allgemein wie möglich“ ist, folgt $s \mid_{tvar(\Gamma)\cup\{\alpha\}\sqsubseteq s'}$.

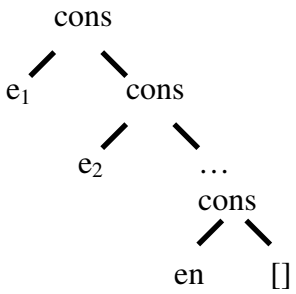
Fazit: $\mathcal{L}_2^{ti}$ ist eine typsichere Programmiersprache, die „benutzerfreundlicher“ ist als $\mathcal{L}_2^{t}$, denn:

- die wohlgetypten Ausdrücke von $\mathcal{L}_2^{ti}$ sind genau diejenigen, die sich durch „erase“ aus den wohlgetypten Ausdrücken von $\mathcal{L}_2^{t}$ ergeben → die Berechnung eines abgeschlossenen wohlgetypten Ausdrucks bleibt nie stecken.
- Der Typinterferenzalgorithmus erlaubt uns, Wohlgetyptheit (und damit Zugehörigkeit zu $\mathcal{L}_2^{ti}$ zu überprüfen).

Spracherweiterung: Paare und Listen

$\mathcal{L}_2^{ti}$ wird zu $\mathcal{L}_3^{ti}$ erweitert!
Erweiterung der kontextfreien Syntax:
Neue Konstanten:
c ::= [] | cons (constructor) | hd (head) | tl (tail) | is_empty | fst | snd
Neue Ausdrücke:
e ::= (e_1, e_2)

Syntaktischer Zucker:
$[e_1, \ldots, e_n]$ steht für $cons(e_1, cons(e_2, \ldots cons(e_n, [])))$
als Baum:



Neue Typen:

$\tau ::= \tau_1 * \tau_2$ (Produkttypen) | τ_1 list (Listentypen)

Neue Typregeln:

(EMPTY) [] :: τ list

(CONS) cons :: $\tau * \tau$ list → τ list

(HD) hd :: τ list → τ

(TL) tl :: τ list → τ list

(IS_EMPTY) is_empty :: τ list → bool

(FST) fst :: $\tau_1 * \tau_2 \rightarrow \tau_1$

(SND) snd :: $\tau_1 * \tau_2 \rightarrow \tau_2$

(PAIR) $\dfrac{\Gamma \triangleright l_1 :: \tau_1 \quad \Gamma \triangleright e_2 :: \tau_2}{\Gamma \triangleright (e_1, e_2) :: \tau_1 * \tau_2}$

Abgeleitete Regel:

(LIST) $\dfrac{\Gamma \triangleright e_1 :: \tau \ldots \Gamma \triangleright e_n :: \tau}{\Gamma \triangleright [e_1; \ldots; e_n) :: \tau \text{ list}}$

small step Semantik

Die Menge Val der Werte wird erweitert durch

$v ::= (v_1, v_2)$ | cons v_1