Rekonstruktion von Typen

Proseminar „Grundlagen höherer Programmiersprachen“ (WS 02/03)

- Polymorphismus, Typinferenz
- Substitution von Typen
- Constraint-Based Typing
- Unifikation

Roland Eckl, ecklr@informatik.uni-muenchen.de
Ludwig-Maximilians-Universität München, Institut für Informatik
http://www.cip.informatik.uni-muenchen.de/~ecklr/

1. Allgemeines

Polymorphismus

Polymorphe Typsysteme erlauben anstatt der strikten Festlegung auf bestimmte Typen die Verwendung von Typvariablen. Der Körper einer polymorphen Funktion kann also mit Argumenten verschiedenen Typs ausgeführt werden.

Typinferenz

Unter Typinferenz versteht man das Folgern eines bestimmten Typs für einen gegebenen Ausdruck, obwohl der Typ nicht explizit angegeben wurde. Stattdessen kann man unter Zuhilfenahme eines Algorithmus auf die richtigen Typen rückschließen.

Da Typinferenz jedoch sehr stark von dem jeweiligen Typsystem abhängig ist, kann ein passender Algorithmus zum Teil auch nur schwer oder gar nicht gefunden werden. Gute Typinferenz zeichnet sich daher in der Regel dadurch aus, dass sie immer den allgemeinsten Typ für einen nicht getypten Ausdruck findet.

Die hier letztlich benutzte Typinferenz setzt sich aus den Constraint Typing Rules, der Unifikation und der Substitution von Typen zusammen.

Anwendungsgebiete für Typinferenz

- „Berechnung“ der möglichen Eingabe- und Rückgabe-Typen. (Vergleiche hierzu beispielsweise mit *SML* oder etwa *Gofer*, *Haskell*)
- Die Ausführbarkeit von Algorithmen kann bestimmt werden. Sind diese also in den gegebenen Kombinationen aus Typen und nicht näher spezifizierten Variablen überhaupt lauffähig. (Einsatz von Typinferenz in Compilern)

2. Substitution von Typen

Formal gesehen ist die Typsubstitution σ eine endliche Abbildung von Typvariablen auf Typen. Es werden die Typvariablen also durch (in der Praxis konkrete) Typen ersetzt.

$\sigma = [X \mapsto T, Y \mapsto U]$ ist beispielsweise die Schreibweise für die Substitution, die X mit T verbindet und Y mit U.

$dom(\sigma)$ ist die Menge aller Typvariablen, die in σ jeweils auf der linken Seite stehen.

$range(\sigma)$ ist die Menge aller Typen, die in σ jeweils auf der rechten Seite stehen und zur Ersetzung der Variablen dienen.

Definition für die Anwendung der Substitution auf einen Typ:

$$\sigma(\mathrm{X}) = \begin{cases} \mathrm{T} & \text{if } (\mathrm{X} \mapsto \mathrm{T}) \in \sigma \\ \mathrm{X} & \text{if } \mathrm{X} \notin dom(\sigma) \end{cases}$$

$$\sigma(\mathrm{Nat}) = \mathrm{Nat}$$

$$\sigma(\mathrm{Bool}) = \mathrm{Bool}$$

$$\sigma(\mathrm{T}_1 \to \mathrm{T}_2) = \sigma\,\mathrm{T}_1 \to \sigma\,\mathrm{T}_2$$

Sind sowohl σ als auch γ als Substitution gegeben, können diese folgendermaßen kombiniert werden:

$$\sigma \circ \gamma = \begin{bmatrix} \mathrm{X} \mapsto \sigma(\mathrm{T}) & \text{für jedes } (\mathrm{X} \mapsto \mathrm{T}) \in \gamma \\ \mathrm{X} \mapsto \mathrm{T} & \text{für jedes } (\mathrm{X} \mapsto \mathrm{T}) \in \sigma \text{ mit } \mathrm{X} \notin dom(\gamma) \end{bmatrix}$$

Beachte: $(\sigma \circ \gamma)\mathrm{S} = \sigma(\gamma\,\mathrm{S})$

Zwei mögliche Sichtweisen für Terme t:

- Typvariablen bleiben während der Typüberprüfung abstrakt.
  Der Term $\lambda f : X \to X . \lambda a : X . f(fa)$ hat so zum Beispiel den Typ $(X \to X) \to X \to X$, und bei jeder Ersetzung von X durch einen konkreten Typ bleibt der Term wohlgetypt.
  (*parametric polymorphism*, eine Funktion wird unabhängig von Typ der Parameter ausgeführt)

- Selbst bei nicht wohlgetypten Termen kann überprüft werden, ob diese nicht durch angemessene Werte in wohlgetypte Terme umgeformt werden können.
  So ist beispielsweise der Term $\lambda f : Y . \lambda a : X . f(fa)$ in dieser Form nicht typbar. Aber durch Ersetzung von Y mit Nat $\to$ Nat und X mit Nat wird daraus $\lambda f : \text{Nat} \to \text{Nat} . \lambda a : \text{Nat} . f(fa)$, vom Typ $(\text{Nat} \to \text{Nat}) \to \text{Nat} \to \text{Nat}$.

Für einen Kontext Γ und einen Term $\mathbf{t}$ ist (σ, T) eine Lösung.

3. Constraint-Based Typing

Mit den Regeln der nachfolgenden Seite lassen sich aus einem Kontext Γ und einem Term $\mathbf{t}$ so genannte constraints berechnen, d.h. Gleichungen zwischen Ausdrücken von Typen in der Form $\{ S_i = T_i \ ^{i \in 1..n} \}$

Es werden hierbei jedoch keine constraints geprüft, sondern lediglich für die spätere Auswahl aufgelistet. So wird beispielsweise eine Anwendung $t_1\ t_2$ mit $\Gamma \vdash t_1 : T_1$ und $\Gamma \vdash t_2 : T_2$ nicht darauf hin überprüft, ob t_1 als Rückgabewert $T_2 \to T_{12}$ besitzt, sondern es wird stattdessen eine „frische" Variable X gewählt, und der Eintrag lautet $T_1 = T_2 \to X$

Eine Substitution σ unifiziert eine Gleichung $S = T$ wenn die Instanzen σS und σT identisch sind.

$FV(T)$ ist die Menge aller in T vorkommenden Typen.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T \mid_{\varnothing} \{\}} \text{ CT-VAR} \qquad \frac{}{\Gamma \vdash 0 : \text{Nat} \mid_{\varnothing} \{\}} \text{ CT-ZERO}$$

$$\frac{}{\Gamma \vdash \text{true} : \text{Bool} \mid_{\varnothing} \{\}} \text{ CT-TRUE} \qquad \frac{}{\Gamma \vdash \text{false} : \text{Bool} \mid_{\varnothing} \{\}} \text{ CT-FALSE}$$

$$\frac{\Gamma \vdash t_1 : T \mid_X C \quad C' = C \cup \{ T = \text{Nat} \}}{\Gamma \vdash \text{succ } t_1 : \text{Nat} \mid_X C'} \text{ CT-SUCC} \qquad \frac{\Gamma \vdash t_1 : T \mid_X C \quad C' = C \cup \{ T = \text{Nat} \}}{\Gamma \vdash \text{pred } t_1 : \text{Nat} \mid_X C'} \text{ CT-PRED}$$

$$\frac{\Gamma \vdash t_1 : T \mid_X C \quad C' = C \cup \{ T = \text{Nat} \}}{\Gamma \vdash \text{iszero } t_1 : \text{Bool} \mid_X C'} \text{ CT-ISZERO} \qquad \frac{\Gamma, x : T_1 \vdash t2 : T_2 \mid_X C}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2 \mid_X C} \text{ CT-ABS}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : T_1 \mid_{X_1} C_1 \\ \Gamma \vdash t_2 : T_2 \mid_{X_2} C_2 \qquad \Gamma \vdash t_3 : T_3 \mid_{X_3} C_3 \\ X_1, X_2, X_3 \text{ nonoverlapping} \\ C' = C_1 \cup C_2 \cup C_3 \cup \{ T_1 = \text{Bool}, T_2 = T_3 \}\end{array}}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T_2 \mid_{X_1 \cup X_2 \cup X_3} C'} \text{ CT-IF}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : T_1 \mid_{X_1} C_1 \qquad \Gamma \vdash t_2 : T_2 \mid_{X_2} C_2 \\ X_1 \cap X_2 = X_1 \cap \text{FV}(T_2) = X_2 \cap \text{FV}(T_1) = \varnothing \\ X \notin X_1, X_2, T_1, T_2, C_1, C_2, t_1 \text{ or } t_2 \\ C' = C_1 \cup C_2 \cup \{ T_1 = T_2 \to X \}\end{array}}{\Gamma \vdash t_1 \, t_2 : X \mid_{X_1 \cup X_2 \cup \{X\}} C'} \text{ CT-APP}$$

4. Unifikation

Um die Lösungen aus den *constraint sets* zu berechnen, bedienen wir uns der Unifikation nach einer Idee von Hindley und Miller. Auf diese Weise lässt sich feststellen, ob eine nicht-leere Lösungsmenge gegeben ist, und um dann daraus das „beste" Element zu bestimmen.

```
unify(C)=  if C = ∅, then []
           else let { S = T } ∪ C′ = C in
                if S = T
                     then unify(C′)
                else if S = X and X ∉ FV(T)
                     then unify([X↦T]C′)∘[X↦T]
                else if T = X and X ∉ FV(T)
                     then unify([X↦S]C′)∘[X↦S]
                else if S = S₁→S₂ and T = T₁→T₂
                     then unify(C′ ∪ {S₁ = T₁, S₂ = T₂})
                else
                     fail
```

- `unify(C)` endet für alle constraints C, entweder durch Abbruch (fail) bei nicht gegebener Unifizierbarkeit, oder durch die Rückgabe eines *principal unifiers*, der eine Substitution für alle constraints ermöglicht.
- Wenn `unify(C)` = σ, dann ist σ ein Unifikator von C.
- Wenn ζ ein Unifikator von C ist, dann ist `unify(C)` = σ, wobei $\zeta \sqsubseteq \sigma$.