

Proseminar „Programmiersprachen“

ML-Polymorphismus

Marc Hoffmann

29.11.2006

Betreut von Prof. T. Nipkow

Technische Universität München

Motivation

- Der Lambda-Kalkül hat in seiner reinen Form keine Typen. Ein Term muss nur den syntaktischen Vorgaben genügen.
- Bei der Evaluierung mancher Terme kann man jedoch in einen **festgefahrenen Zustand** [stuck state] geraten, weil keine weiteren Evaluierungsregeln anwendbar sind.
- Die Verwendung von Typen soll dem entgegenwirken: alle Ausdrücke, denen ein Typ zugeordnet werden kann, **werten stets komplett aus**.
- Weitere Konsequenz: **nicht alle Terme sind typisierbar**.

Teil I: Explizite Typbestimmung

- Bei einer **Abstraktion** (Funktion) wird für die gebundene Variable ein Typ angegeben. Notation: $\lambda x : T. t$
- Für alle Variablen x im Term t wird dann der Typ T angenommen. Können wir jetzt zeigen, dass der Term t den Typ T' annimmt, so hat die Abstraktion insgesamt den Typ $T \rightarrow T'$.
- Beachte: T' ist **abhängig von** T .
- Beispiel: Es sei $f = \lambda x. iszero\ x$ mit $x : Nat$. Dann hat f den Typ $Nat \rightarrow Bool$.

Umgebung und Notationen

- Wir möchten uns die Annahmen über Variablen sinnvoll merken:
- Eine **Umgebung** ist eine Sequenz (Menge) von Tupeln $x : T$. Dabei ist x eine **Variable** und T der ihr **zugeordnete Typ**. Neue Einträge werden rechts nach einem Komma eingefügt. Die leere Umgebung wird oft ausgelassen.

→ Notationen:

$$\Gamma \vdash x : T$$

$$\Gamma \vdash \lambda x : T_1. t : T_1 \rightarrow T_2$$

$$\Gamma \vdash t_1 t_2 : T$$

Umbenennung von Bezeichnern

→ Werden in einem Term zwei Variablen gebunden, die den gleichen Bezeichner benutzen, so benennen wir einen davon um. Statt

$$\lambda x : T_1 . \lambda y : T_2 . (\lambda x : T_3 . x y) x$$

schreiben wir z.B.

$$\lambda x : T_1 . \lambda y : T_2 . (\lambda z : T_3 . z y) x$$

→ Dies ist insbesondere immer möglich, da der Zeichenvorrat (abzählbar) unendlich ist. Wir sagen auch, dass die Ausdrücke eindeutig „modulo Wahl der Bezeichner“ sind.

Ableitungsregeln

- Für die Bestimmung des Typs eines Terms werden **Ableitungsregeln** [inference rules] genutzt, um den Typ zu inferieren. Folgender Sachverhalt wird dabei für die **Abstraktion** ausgedrückt:

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x : T_1 . t : T_1 \rightarrow T_2} \text{ (T-ABS)}$$

- Die Abstraktion $\Gamma \vdash \lambda x : T_1 . t$ gibt an, dass für alle x in t der Typ T_1 angenommen werden soll,
- d.h. $x : T_1$ wird in die Umgebung für t eingefügt.
- Können wir nun (mittels weiterer Ableitungen) zeigen, dass t in der Umgebung $\Gamma, x : T_1$ den Typ T_2 hat,
- So hat der Ausdruck $\Gamma \vdash \lambda x : T_1 . t$ den Typ $T_1 \rightarrow T_2$.

Ableitungsregeln

- Die Ausdrücke oberhalb des Trennstriches werden **Prämissen** genannt. Sind diese erfüllt, so folgt der Typ des Ausdrucks unterhalb. Aus dem unteren Ausdruck folgen auch die oberen; dies formt also eine **Äquivalenz**.
- Für Variablen ist die zugehörige Ableitungsregel einfach - der Typ einer Variablen ist gerade der, welcher in der **momentanen** Umgebung vorkommt:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-VAR)}$$

- Alle Variablen kommen auch in der Umgebung vor, da der Gesamtterm geschlossen ist, d.h. alle Variablen durch (T-ABS) in die Umgebung aufgenommen wurden.

Ableitungsregeln

→ Für die Abstraktion gestaltet sich die Ableitungsregel folgendermaßen:

$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 t_2 : T_{12}} \text{ (T-APP)}$$

- Wir gehen von einer Applikation $t_1 t_2$ aus.
- Können wir in der Umgebung Γ zeigen, dass t_1 den Typ $T_{11} \rightarrow T_{12}$ und t_2 den Typ T_{11} hat (die Typen also „zueinander passen“),
- So ist der Typ der Applikation $t_1 t_2$ schließlich T_{12} .
- Auch hier sind weitere Ableitungsschritte für die Bestimmung der Typen von t_1 und t_2 notwendig.

Ableitungsbäume

- Durch die Verwendung der Ableitungsregeln entsteht ein **Ableitungsbaum**:
Die Ableitungsregel (T-ABS) erzeugt einen Kindknoten, (T-APP) erzeugt zwei Kindknoten und (T-VAR) stellt ein Blatt dar.
- Die Ableitungsregeln beschreiben ein **algorithmisches Verfahren**, mit dem der **monomorphe Typ** sukzessiv bestimmt werden kann.

Folgende wichtige Eigenschaften sind beweisbar:

- Der Typ ist **für eine gegebene Umgebung eindeutig**; weiterhin ist der zugehörige Ableitungsbaum ebenfalls eindeutig.
- Die Auswertung (Evaluierung) eines Ausdrucks **ändert den Typ nicht**.

Bool als Typ

$$\Gamma \vdash \text{true} : \text{Bool} \quad (\text{T-TRUE})$$

$$\Gamma \vdash \text{false} : \text{Bool} \quad (\text{T-FALSE})$$

→ In jeder Umgebung sind *true* und *false* vom Typ *Bool*. Diese Regeln sind **Prämissenfrei** und erzeugen Blätter. Der Aufbau der *if*-Ableitung ist:

$$\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (\text{T-IF})$$

→ **Gegeben:** $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3$

→ Können wir in der Umgebung Γ zeigen, dass, dass t_1 den Typ *Bool*, t_2 sowie t_3 den gleichen Typ *T* haben,

→ So ist der Typ der unteren Zeile ebenfalls *T*.

Beispiel

→ Was ist der Typ des folgenden Ausdrucks?

$$\vdash (\lambda x : Bool. x) true$$

→ **Lösung:**

$$\begin{array}{c}
 \frac{x : Bool \in x : Bool}{x : Bool \vdash x : Bool} \text{T-VAR} \\
 \frac{\vdash \lambda x : Bool . x : Bool \rightarrow Bool}{\vdash (\lambda x : Bool . x) true : Bool} \text{T-ABS} \quad \frac{}{\vdash true : Bool} \text{T-TRUE} \\
 \hline
 \vdash (\lambda x : Bool . x) true : Bool \quad \text{T-ABS}
 \end{array}$$

Zwischenbilanz

- Wir können für Ausdrücke, welche mit **konkreten Typen** annotiert sind, einen (eventuellen) **monomorphen Typ algorithmisch bestimmen**.
- Gibt es im Ableitungsbaum eine Stelle, an der die Typbestimmung nicht weiter fortschreiten kann, so ist der Gesamtterm **nicht typisierbar**.
- Was fehlt noch, um Polymorphismus zu verwirklichen? Ein polymorpher Ausdruck beschreibt im Allgemeinen eine **Menge von Typen**, für die der Ausdruck typkorrekt ist. Wir möchten diese Menge von typkorrekten Instanziierungen ebenfalls beschreiben und einen **allgemeinsten Typ algorithmisch bestimmen** können.

Teil II: Implizite Typbestimmung

- Es werden **Typvariablen** (z.B. X) statt konkreter Typen verwendet. Diese sind als Platzhalter zu verstehen und werden bei einer Instanziierung durch einen konkreten Typ ersetzt.
- Allgemein beschreiben wir Typersetzung mit einer **Substitution** σ .
- Beispielsweise beschreibt $\sigma = [X \mapsto Bool]$ eine Substitutionsregel, die alle Vorkommnisse von X durch $Bool$ ersetzt (auf der rechten Seite sind Typvariablen auch möglich).
- Die Anwendung $\sigma(X \rightarrow X)$ ergibt dann $Bool \rightarrow Bool$.

Substitutionsregeln

$$\sigma(X) = \begin{cases} T & , \text{ falls } (X \mapsto T) \in \sigma \\ X & , \text{ sonst} \end{cases}$$

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

$$\sigma(T_1 \rightarrow T_2) = \sigma T_1 \rightarrow \sigma T_2$$

$$\sigma(x_1 : T_1, \dots, x_n : T_n) = (x_1 : \sigma T_1, \dots, x_n : \sigma T_n)$$

$$\sigma(\lambda x : X. t : Y) = \lambda x : \sigma(X). \sigma(t) : \sigma(Y)$$

$$\sigma(t_1 t_2 : X) = \sigma(t_1) \sigma(t_2) : \sigma(X)$$

$$\sigma(\mathit{if } t_1 \mathit{ then } t_2 \mathit{ else } t_3 : T) = \mathit{if } \sigma(t_1) \mathit{ then } \sigma(t_2) \mathit{ else } \sigma(t_3) : \sigma(T)$$

σt

→ Substitutionen sind **eindeutig**.

→ Die Substitution **erhält den Typ**: Aus $\Gamma \vdash t : T$ folgt $\sigma\Gamma \vdash \sigma t : \sigma T$.

Lösung von (Γ, t)

→ Zu einer gegebenen Umgebung Γ und einem Term t ist **eine Lösung** von (Γ, t) **ein Paar** (σ, T) , so dass $\sigma\Gamma \vdash \sigma t : T$

→ **Beispiel:** Es seien $t = f a$ sowie $\Gamma = f : X, a : Y$. Dann sind

$$([X \mapsto Y \rightarrow Bool], Bool)$$

$$([X \mapsto Bool \rightarrow Bool, Y \mapsto Bool], Bool)$$

$$([X \mapsto Y \rightarrow Bool \rightarrow Bool], Bool \rightarrow Bool)$$

einige Lösungen von (Γ, t) .

Typinferenz mit Bedingungen

- Der bisherige Ansatz beschreibt zwar **deklarativ** alle Lösungen; diese können wir aber nicht bestimmen oder konstruieren.
- Die **Typinferenz mit Bedingungen** ist ein Verfahren, welches die gleichen Lösungen bestimmt und **algorithmischer Natur** ist. Zu einem Term t und einer Umgebung Γ wird ein Typ S und Bedingungen \mathcal{C} berechnet.
- Mit \mathcal{C} bezeichnen wir eine **Menge von Bedingungen** $\{S_i = T_i\}_{i \in 1 \dots n}$.
- Eine Substitution σ **unifiziert** eine Gleichung $S = T$ dann, wenn σS und σT identisch sind.
- Eine Substitution σ **unifiziert** \mathcal{C} dann, wenn σ jede Gleichung aus \mathcal{C} unifiziert.

Erzeugung von Bedingungen

→ Statt die bei jeder Ableitung notwendigen Zusammenhänge zwischen den Prämissen sofort zu prüfen, werden diese **in den Bedingungen vermerkt**.

Beispiel: Für eine Applikation $t_1 t_2$ mit $\Gamma \vdash t_1 : T_1$ und $\Gamma \vdash t_2 : T_2$ wurde bisher geprüft, ob $T_1 = T_2 \rightarrow R$ gilt. Nun wird mit einer noch unbenutzten **Typvariablen** X in den Bedingungen \mathcal{C} aufgezeichnet, dass

$$T_1 = T_2 \rightarrow X$$

Schließlich wird für die Applikation der Typ X zurückgegeben. Die Applikation hat also den Typ X , falls $T_1 = T_2 \rightarrow X$ gilt.

Ein Paar (σ, T) heißt **Lösung** von $(\Gamma, t, S, \mathcal{C})$ falls $\sigma S = T$ und σ die Bedingungen \mathcal{C} alle erfüllt (d.h. σ ist **Unifikator** von \mathcal{C}).

Ableitungsregeln mit Bedingungen

Notation: $\Gamma \vdash t : T \mid \mathcal{C}$ heißt, der Term t hat Typ T in der Umgebung Γ , falls die Bedingungen \mathcal{C} erfüllt sind. T ist im Allgemeinen eine Typvariable.

→ Die Ableitungsregeln werden folgendermaßen **mit Bedingungen erweitert**:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T \mid \{\}} \quad (\text{CT-VAR})$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \mid \mathcal{C}}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \rightarrow T_2 \mid \mathcal{C}} \quad (\text{CT-ABS})$$

→ Und **ohne Typannotation** (X ist eine noch unbenutzte Typvariable):

$$\frac{\Gamma, x : X \vdash t_1 : T \mid \mathcal{C}}{\Gamma \vdash \lambda x . t_2 : X \rightarrow T \mid \mathcal{C}} \quad (\text{CT-ABSINF})$$

Ableitungsregeln mit Bedingungen

→ Und die weiteren Ableitungsregeln analog...

$$\frac{\Gamma \vdash t_1 : T_1 \mid \mathcal{C}_1 \quad \Gamma \vdash t_2 : T_2 \mid \mathcal{C}_2 \quad \mathcal{C}' = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{T_1 = T_2 \rightarrow X\}}{\Gamma \vdash t_1 t_2 : X \mid \mathcal{C}'} \text{ (CT-APP)}$$

$$\Gamma \vdash \text{true} : \text{Bool} \mid \{\}$$

(CT-TRUE)

$$\Gamma \vdash \text{false} : \text{Bool} \mid \{\}$$

(CT-FALSE)

$$\frac{\Gamma \vdash t_1 : T_1 \mid \mathcal{C}_1 \quad \Gamma \vdash t_2 : T_2 \mid \mathcal{C}_2 \quad \Gamma \vdash t_3 : T_3 \mid \mathcal{C}_3 \quad \mathcal{C}' = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \{T_1 = \text{Bool}, T_2 = T_3\}}{\Gamma \vdash f t_1 \text{ then } t_2 \text{ else } t_3 : T_2 \mid \mathcal{C}'} \text{ (CT-IF)}$$

Beispiel

→ Finde S und C , so dass

$$\vdash \lambda x : X . \lambda y : Y . \lambda z : Z . (x z)(y z) : S \mid C$$

→ **Lösung:**

$$\begin{array}{c}
 \frac{x : X \in \Gamma \quad z : Z \in \Gamma}{\Gamma \vdash x : X \quad \Gamma \vdash z : Z} \quad \frac{y : Y \in \Gamma \quad z : Z \in \Gamma}{\Gamma \vdash y : Y \quad \Gamma \vdash z : Z} \\
 \frac{\Gamma \vdash x : X \quad \Gamma \vdash z : Z}{C_1 := \{X = Z \rightarrow A\}} \quad \frac{\Gamma \vdash y : Y \quad \Gamma \vdash z : Z}{C_2 := \{Y = Z \rightarrow B\}} \\
 \frac{\Gamma \vdash (x z) : A \mid C_1 \quad \Gamma \vdash (y z) : B \mid C_2}{C' := \{X = Z \rightarrow A\} \cup \{Y = Z \rightarrow B\} \cup \{A = B \rightarrow C\}} \text{CT-APP} \\
 \frac{\Gamma := x : X, y : Y, z : Z \vdash (x z)(y z) : C \mid C'}{\Gamma := x : X, y : Y \vdash \lambda z : Z . (x z)(y z) : Z \rightarrow C \mid C'} \text{CT-ABS} \\
 \frac{x : X, y : Y \vdash \lambda z : Z . (x z)(y z) : Z \rightarrow C \mid C'}{x : X \vdash \lambda y : Y . \lambda z : Z . (x z)(y z) : Y \rightarrow (Z \rightarrow C) \mid C'} \text{CT-ABS} \\
 \frac{x : X \vdash \lambda y : Y . \lambda z : Z . (x z)(y z) : Y \rightarrow (Z \rightarrow C) \mid C'}{\vdash \lambda x : X . \lambda y : Y . \lambda z : Z . (x z)(y z) : X \rightarrow (Y \rightarrow (Z \rightarrow C)) \mid C'} \text{CT-ABS}
 \end{array}$$

Unifikation

→ Wir möchten aus dem resultierenden Paar (S, C) einen **allgemeinsten Typ** folgern, der die Menge der Lösungen direkt beschreibt.

→ Was ist **Unifikation**? Man betrachte folgende Mengen von Bedingungen:

$$\{X = Bool, Y = X \rightarrow X\} [X \mapsto Bool, Y \mapsto Bool \rightarrow Bool]$$

$$\{Bool \rightarrow Bool = X \rightarrow Y\} [X \mapsto Bool, Y \mapsto Bool]$$

$$\{Y = Bool \rightarrow Y\} \text{ **Nicht unifizierbar (bzw. nur mit rekursiven Typen)**}$$

→ Welche Substitution erfüllt die jeweiligen Bedingungen „am einfachsten“?

→ Die Unifikation leistet im Wesentlichen das, was im Beispiel ersichtlich ist:

Sie erzeugt eine **allgemeinste Substitution**, die alle Bedingungen erfüllt.

Unifikationsalgorithmus

```

unify( $\mathcal{C}$ ) =
  if  $\mathcal{C} = \emptyset$  then []
  else let  $\{S = T\} \cup \mathcal{C}' = \mathcal{C}$  in
    if  $S = T$  then
      unify( $\mathcal{C}'$ )
    else if  $S = X$  and  $X \notin \text{FV}(T)$  then
      unify( $[X \mapsto T]\mathcal{C}'$ )  $\circ$   $[X \mapsto T]$ 
    else if  $T = X$  and  $X \notin \text{FV}(S)$  then
      unify( $[X \mapsto S]\mathcal{C}'$ )  $\circ$   $[X \mapsto S]$ 
    else if  $S = S_1 \rightarrow S_2$  and  $T = T_1 \rightarrow T_2$  then
      unify( $\mathcal{C}' \cup \{S_1 = T_1, S_2 = T_2\}$ )
    else fail
  
```

„wähle eine Bedingung $S = T$ aus der Menge \mathcal{C} , wobei $\mathcal{C}' = \mathcal{C} \setminus \{S = T\}$ ist“

Allgemeinster Typ

- Der Unifikationsalgorithmus terminiert immer. Er bricht ab, falls keine Unifikation möglich ist, und gibt sonst die **allgemeinste Substitution** zurück, welche die Bedingungen erfüllt. Diese hat die Eigenschaft, dass sie aus einer minimalen Anzahl von Einzelsubstitutionen besteht.
- Wenden wir diese **allgemeinste Substitution** auf das S an, welches wir durch Verwendung der Typinferenz mit Bedingungen erhielten, so liefert uns dies den **allgemeinsten Typ T** von t .
- Der **allgemeinste Typ** ist so erstellt, dass er die Bedingungen „minimal“ erfüllt. Er dokumentiert sozusagen die **Struktur des Typs**. Aus dem **allgemeinsten Typ** sind **alle Lösungen generierbar**.

Fortsetzung des Beispiels

→ Im Beispiel hatten wir den folgenden Ausdruck inferiert:

$$\vdash \lambda x : X . \lambda y : Y . \lambda z : Z . (x z)(y z) : S \mid \mathcal{C}$$

→ Dabei hatten wir folgende Werte ermittelt:

$$S = X \rightarrow (Y \rightarrow (Z \rightarrow C)) = X \rightarrow Y \rightarrow Z \rightarrow C$$

$$\mathcal{C} = \{X = Z \rightarrow A, Y = Z \rightarrow B, A = B \rightarrow C\}$$

→ Die Anwendung der **Typinferenz mit Bedingungen** stellt **Schritt 1** dar.

→ Wir berechnen nun die **allgemeinste Substitution** (den **allgemeinsten Unifikator**) mit dem **Unifikationsalgorithmus** für die Menge \mathcal{C} aus. Dies wäre dann **Schritt 2**.

Fortsetzung des Beispiels

$$\begin{aligned}
 \text{unify}(C) &= \text{unify}([X \mapsto Z \rightarrow A]\{Y = Z \rightarrow B, A = B \rightarrow C\}) \circ [X \mapsto Z \rightarrow A] \\
 &= \text{unify}(\{Y = Z \rightarrow B, A = B \rightarrow C\}) \circ [X \mapsto Z \rightarrow A] \\
 &= \text{unify}([Y \mapsto Z \rightarrow B]\{A = B \rightarrow C\}) \circ [Y \mapsto Z \rightarrow B] \circ [X \mapsto Z \rightarrow A] \\
 &= \text{unify}(\{A = B \rightarrow C\}) \circ [Y \mapsto Z \rightarrow B, X \mapsto Z \rightarrow A] \\
 &= \text{unify}([A \mapsto B \rightarrow C]\{\}) \circ [A \mapsto B \rightarrow C] \circ [Y \mapsto Z \rightarrow B, X \mapsto Z \rightarrow A] \\
 &= \text{unify}(\{\}) \circ [A \mapsto B \rightarrow C, Y \mapsto Z \rightarrow B, X \mapsto Z \rightarrow A] \\
 &= [] \circ [A \mapsto B \rightarrow C, Y \mapsto Z \rightarrow B, X \mapsto Z \rightarrow A] \\
 &= [A \mapsto B \rightarrow C, Y \mapsto Z \rightarrow B, X \mapsto Z \rightarrow A] =: \sigma
 \end{aligned}$$

→ Schließlich erhalten wir

$$\begin{aligned}
 \sigma S &= [A \mapsto B \rightarrow C, Y \mapsto Z \rightarrow B] (Z \rightarrow A) \rightarrow Y \rightarrow Z \rightarrow C \\
 &= [A \mapsto B \rightarrow C] (Z \rightarrow A) \rightarrow (Z \rightarrow B) \rightarrow Z \rightarrow C \\
 &= (Z \rightarrow B \rightarrow C) \rightarrow (Z \rightarrow B) \rightarrow Z \rightarrow C
 \end{aligned}$$

was der **allgemeinsten Typ** ist. Die Anwendung von σ auf S ist **Schritt 3**.

Fortsetzung des Beispiels

→ Der allgemeinste Typ ist also $(Z \rightarrow B \rightarrow C) \rightarrow (Z \rightarrow B) \rightarrow Z \rightarrow C$

Zum Vergleich: Die Auswertung in Ocaml liefert für

```
fun x -> fun y -> fun z -> (x z) (y z);;
```

die Signatur

```
- : ('a -> 'b -> 'c) -> ('a -> 'b) -> 'a -> 'c = <fun>
```

→ **Bis auf Umbenennung** der Typvariablen sind die Typen also **Äquivalent**.

Fazit: Ist ein (polymorpher) Ausdruck typisierbar, so können wir durch die eben erwähnten drei Schritte einen allgemeinsten Typ inferieren. Mit diesem lässt sich bei Verwendung des Ausdrucks prüfen, ob die Instanziierung der Typen das **Schema erhält**; ist dies gegeben, so ist die Instanz typkorrekt.

Danke für Ihre Aufmerksamkeit!

Sind noch **Fragen** offen?