

## 8. Typinferenz und Polymorphismus

Bisher: explizit getypte Programme

- in Wertdeklarationen: Typangabe für deklarierte Variablen
- in Muster: Typangaben für jede Mustervariable

### 8.1 Typinferenz

SML verlangt keine explizite Typangabe (mit einigen Ausnahmen).

**Typinferenz** versucht, Typinformationen über Variablen aus dem Kontext zu berechnen.

#### Beispiele:

- $\text{fn } n:\text{int} \Rightarrow n + 1$   
Typangabe `int` ist überflüssig, weil die Funktionsdefinition nur dann typkorrekt ist, wenn im definierenden Ausdruck,  $n + 1$ ,  $n$  den Typ `int` hat.
- $\text{fn } s:\text{string} \Rightarrow "0"^\wedge s$   
Definierender Ausdruck  $"0"^\wedge s$  erzwingt, dass  $s$  den Typ `string` hat.

Manche Funktionen erlauben mehr als einen Argumenttyp

#### Beispiel:

Identitätsfunktion  $\text{fn } x \Rightarrow x$

Der definierende Ausdruck  $x$  erzwingt keinerlei Typeinschränkung. Die Identitätsfunktion verhält sich für Argumente aller Typen gleich. Solche Funktionen heißen **polymorph**.

Typ der Identitätsfunktion:  $t \rightarrow t$  wenn Argumenttyp  $t$  ist.

Formalisierung: **Typschema**, Ausdrücke in **Typkonstruktoren** und **Typvariablen**

**Typkonstruktoren:**  $\rightarrow$  (2-stellig), `int` (0-stellig)

**Typvariablen:** schreibt man in ML `'a`, `'b`, `'c`  
man spricht sie traditionell  $\alpha$ ,  $\beta$ ,  $\gamma$

Typvariablen (meist allquantifiziert) stehen für beliebige Variablen.

Eine **Instanz** eines Typschemas erhält man, indem man Typvariablen des Schemas **konsistent** durch Typschemata ersetzt. **Konsistente Ersetzung** heißt, alle Vorkommen einer Typvariablen werden durch das gleiche Typschema ersetzt.

#### Beispiele:

- Typschema  $'a \rightarrow 'a$  hat (u.a.) folgende Instanzen:  
 $\text{int} \rightarrow \text{int}$   
 $(\text{int} \rightarrow 'a) \rightarrow (\text{int} \rightarrow 'a)$   
 $\text{char} \rightarrow \text{char}$   
 $(\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$

$(\text{int}, \text{char}) \rightarrow (\text{int}, \text{char})$   
keine Instanz ist  $\text{int} \rightarrow \text{real}$

- Typschema  $'a \rightarrow 'b$  hat z.B. die Instanzen  
 $\text{int} \rightarrow \text{real}$   
 $\text{int} \rightarrow \text{int}$

### Beispiele polymorpher Funktionen:

- $\text{fn } (x, y) \Rightarrow x + 1$   
Typeinschränkung für  $x$ :  $\text{int}$   
Typ:  $(\text{int} * 'a) \rightarrow \text{int}$
- $\text{fn } (x, y) \Rightarrow x$   
Typ:  $'a * 'b \rightarrow 'a$

Gäben wir der Funktion das Typschema  $'a * 'a \rightarrow 'a$ , so wäre das zu restriktiv, würde einige durch die Funktionsdefinition erlaubte Typen ausschließen.

Andererseits wäre das Typschema  $'a * 'b \rightarrow 'c$  falsch, weil die Bedingung „Typ des ersten Arguments gleich Typ des Ergebnisses“ nicht ausgedrückt wäre.

$'a * 'b \rightarrow 'a$  ist **der allgemeinste Typ** (das allgemeinste Typschema) für diese Funktion.

Eigenschaft von SML: (Fast) jeder Ausdruck hat einen allgemeinsten Typ (principal type)

**Typinferenzer** berechnet für alle Funktionen den allgemeinsten Typ, unter dem alle Anwendungen der Funktion typkorrekt sind.

Ein Vorkommen eines Ausdrucks bzw. eine Anwendung der Funktion bekommt eine Instanz des allgemeinen Typs zugeordnet.

### Beispiele:

- $(\text{fn } x \Rightarrow x) (0)$   
erzwingt für dieses Vorkommen den Typ  $\text{int} \rightarrow \text{int}$
- $((\text{fn } x \Rightarrow x) (\text{fn } x \Rightarrow x)) (0)$   

$\downarrow$   
 $(\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$

$\downarrow$   
 $\text{int} \rightarrow \text{int}$

### Typinferenz (2-stufiger Prozess)

Frage: unter welchen Typbedingungen sind alle Ausdrücke typkorrekt?

#### 1. Aufstellen von Bedingungen (constraints)

Beispiele:

- $e_1 e_2$   
Bedingungen:  $e_1$  muss funktionalen Typ haben  
 $e_2$  muss vom Argumenttyp von  $e_1$  sein
- $\text{if } e_1 \text{ then } e_2 \text{ else } e_3$   
Bedingung:  $e_1$  muss Typ  $\text{bool}$  sein und  $e_2$  und  $e_3$  müssen den gleichen Typ haben

#### 2. Lösen der Bedingung (constraint satisfaction)

Analog zum Lösen von Gleichungssystemen, aber mit Rechnen über Typschemata

#### Ergebnisse:

1. **überbestimmt:** Typfehler, keine Lösung
2. **unterbestimmt:** mehrere Lösungen  
eventuell explizite Typangabe erforderlich

### 3. **eindeutig bestimmt:** es gibt genau eine Lösung

Problem bei Fehlerdiagnose: Fehlerentdeckung in der 2.Phase nicht unbedingt dort, wo der Fehler verursacht wurde

Empfehlung: gut platzierte explizite Typangaben

### **Inferenzregeln für Typinferenz**

Die meisten Inferenzregeln zur Typberechnung können übernommen werden, beispielsweise [scon].

Bindungen in Typumgebungen sind jetzt an **Typschemata**. [vid] schlägt in der Typumgebung für Variablen nach.

$$[\text{ifthenelse}] \quad \frac{\Gamma \vdash e_1 : \text{bool} \quad \wedge \quad \Gamma \vdash e_2 : t \quad \wedge \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t}$$

#### Bisherige Interpretation:

rechne Typen von  $e_2$  und  $e_3$  aus, teste auf Gleichheit und im Erfolgsfall ordne diesen Typ dem if-Ausdruck zu.

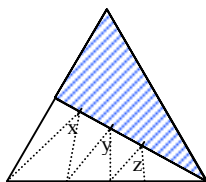
#### Jetzt:

Regel wird benutzt, um die Bedingungen zu bestimmen, unter denen  $e_2$  und  $e_3$  den gleichen Typ haben.

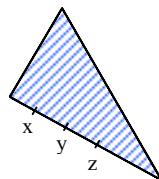
Seien  $t_2$  bzw.  $t_3$  die Typschemata von  $e_2$  bzw.  $e_3$ . **Angleichung** (unification) „macht  $t_2$  und  $t_3$  gleich“, d.h. instantiiert Typvariablen in  $t_2$  und  $t_3$  so, dass das gleiche Typschema resultiert.

#### **Musterabgleich:**

Wert



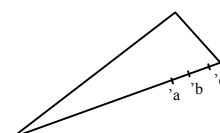
Muster



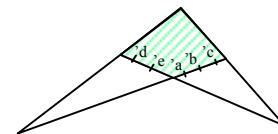
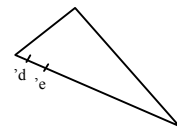
Mustervariablen werden gebunden an Teilwerte

#### **Typschema:**

$t_2$



$t_3$



#### **Typangleichung:**

Variablen in beiden Typschemata gebunden

#### **Beispiel:**

$\text{fn } (x, y, z) \Rightarrow$

$\text{if } z > 0 \text{ then } (x, 1) \text{ else } ("a", y)$

Was macht der Typinferenzer beim definierenden Vorkommen von Variablen im

- fn-Ausdruck
- Wertdeklaration



- im **imperativen** Teil von SML: Auswertung eines Ausdrucks verändert eventuell die Wertumgebung. Deshalb: mehrfaches Auswerten eines Ausdrucks führt eventuell zu verschiedenen Werten

Deshalb Einschränkung:

Variablen dürfen nur an polymorph getypte **Werte** gebunden werden

**Beispiele:**

- val  $J = II$       Fehlermeldung “data polymorphism”
- fun  $Jx = IIx$       o.k., Funktionen sind Werte

## 8.3 Überladung

übernommen aus der Mathematik (für arithmetische Operatoren)

eventuell macht es explizite Typangabe erforderlich

$+$  :  $\text{int} \times \text{int} \rightarrow \text{int}$

$+$  :  $\text{real} \times \text{real} \rightarrow \text{real}$

Typinferenzer benutzt Kontext, um Überladung aufzulösen. Wieviel Kontext?

Direkt umfassende Funktionsdefinition

mosml betrachtet größeren Kontext