

5. Funktionen

Funktionen (Definition & Applikation)

Syntax:

```

exp ::= ...
      fn match
match ::= mrule
mrule ::= pat => exp
appexp ::= ...
         appexp atexp
ty ::= ...
      ty1 → ty2

```

Typberechnung:

(*app*)

$$\frac{\Gamma \oplus \{x \mapsto (val, t)\} \vdash e : t'}{\Gamma \vdash fn(x:t) \Rightarrow e : t \rightarrow t'} [fun]$$

**Funktionen sind
Werte in ML !**

$$\frac{\Gamma \vdash e_1 : t \rightarrow t' \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1(e_2) : t'}$$

Bereiche:

$Z \cup Q \cup B \cup \Sigma \cup \Sigma^* \cup \mathbf{Fkt}$

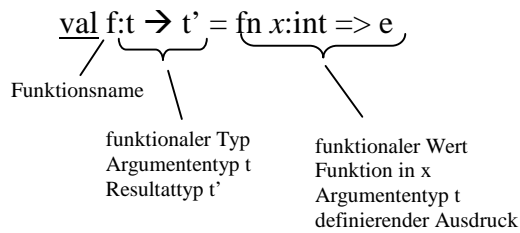
$\mathbf{Fkt} ::= \text{vid } x \text{ exp } x \text{ WU (Wertumgebung)}$

Semantik:

$$\overline{\rho \vdash fn : x \Rightarrow e \Downarrow (x, e, \rho)}$$

$$\frac{\rho \vdash e_1 \Downarrow (x, e, \rho') \quad \rho \vdash e_2 \Downarrow v \quad \rho' \oplus \{x \mapsto v\} \vdash e \Downarrow v'}{\rho \vdash e_1 e_2 \Downarrow v'}$$

$$\frac{\overline{\rho \vdash fn(x : \text{int}) \Rightarrow x \Downarrow (x, x, \rho)} \quad \overline{\rho \vdash 1 \Downarrow 1} \quad \frac{\rho \oplus \{x \mapsto (val, 1)\}(1) = 1}{\rho \oplus \{x \mapsto (val, 1)\} \vdash x \Downarrow 1}}{\rho \vdash (fn(x : \text{int}) \Rightarrow x)(1) \Downarrow 1}$$

Funktionsdeklaration:**Beispiel:**

val quadrat : int \rightarrow int = fn n : int \Rightarrow n*n

val hochvier : int \rightarrow int = fn n : int \Rightarrow quadrat (quadrat(n))

Abkürzende Schreibweise:

fun f (x:t) : t' = e

Funktionsanwendung:

$e_1 e_2$

Typ: e_1 hat funktionalen Typ $t \rightarrow t'$

e_2 hat den Typ t

$e_1 e_2$ hat den Typ t'

Auswertung:

1. Auswertung e_1 zu einem funktionalen Wert fn $x : t \Rightarrow e$
2. Auswertung von e_2
3. Binden Wert von e_2 an x
4. Wertet e in der neuen Wertumgebung aus

semantischer Bereich: $\text{Fkt} = \text{Vid} \times \text{Exp} \times \text{WU}$

Bemerkung:

Überabestrategie von SML: call-by-value

Argumente werden azsgewertet an die Funktion übergeben.

Alternative: Übergabe von e_2 (unausgewertet) und Auswertung erst wenn Wert von x benötigt wird. call-by-name (call-by-need, lazy evaluation)

Gültigkeitsbereiche:

fn $x : t \Rightarrow e$ führt ein definirendes Vorkommen von x ein. Sein Gültigkeitsbereich ist e . Mit diesem definierenden Vorkommen ist noch kein Wert verbunden. Diese Bindung geschieht erst bei der Funktionsanwendung (Schritt 3 von oben)

Beispiel:

```

val x : real = 1.0
fun f (x : real) : real = x + x
fun f (y : real) : real = x + y

```

Konsistente Umbenennung von Funktionsparametern

Bedeutung von Funktionsdeklarationen hängt nicht vom Namen der Parameter ab.

$\left. \begin{array}{l} \text{fun } f(x : \text{real}) : \text{real} = e \\ \text{fun } f(y : \text{real}) : \text{real} = e \end{array} \right\} \text{ bedeutet dieselbe Funktion}$

...außer ein anderes definierendes Vorkommen von y wird “gefangen” im Gültigkeitsbereich des neuen definierenden Vorkommens.

Beispiel:

```

val x : real = 2.0
fun h (y : real) : real = x + y

```

e

Umbenennung von y :

- in z ok.
- in x nicht ok.

Freies Vorkommen:

Variable x hat in Ausdruck e ein **freies Vorkommen** wenn es ein Vorkommen von x in e gibt, welches nicht in einem Gültigkeitsbereich eines definierenden Vorkommens von x in e liegt.

Die Bedeutung von Funktionsdeklaration hängt offensichtlich von der Bindung der freien Variablen ab.

obiges Beispiel: h bezeichnet die Funktion, die auf ihre Argumente 2.0 addiert.

Anderer Kontext für die Deklaration vorhanden:

```

val x : real = 4.0
fun h (y : real) : real = x + y

```

Jetzt würde h 4.0 auf seine Argumente addieren.

Die Wertumgebung ρ in dem Tripel (f, e, ρ) für eine Funktion f enthält die Bindung der freien Variablen von e bei der Deklaration von f .

Prinzip: „statische Bindung“