Konzepte von Programmiersprachen
3. Bindung

Inhalt

Grundlegendes
Vereinbarungen
Blöcke

Typbenennung / Typeinführung in ML

Typbenennung

```
- type complex = real * real ;
> type complex = real * real
- val i = (0.0,1.0) ;
> val i = (0.0,1.0): real * real
```

Typeinführung

```
- datatype complex = compl of real * real;
> type complex
  con compl : real * real->complexKonstruktor-Funktion
- val i = (0.0,1.0) ;
> val i = (0.0,1.0): real * real
```

Ada: siehe Übung

Exkurs: Typäquivalenz

Wann definieren $s = S$ und $t = T$ *äquivalente* Typen?

Beispiele

Name	=	Cardinal → Char
String	=	Cardinal → Char
Person	=	Name × Cardinal
Car	=	String × Cardinal
Person	≡	Car ? String ≡ Name ?

Exkurs: Typäquivalenz

Wann definieren $s = S$ und $t = T$ *äquivalente* Typen?

Beispiele

Name	=	Cardinal → Char
String	=	Cardinal → Char
Person	=	Name × Cardinal
Car	=	String × Cardinal
Person	≡	Car ? String ≡ Name ?

***strukturelle* Äquivalenz**

wenn S und T die *gleiche Wertemenge* definieren

***namentliche* Äquivalenz**

wenn $s = S$ und $t = T$ *an derselben Stelle* definiert wurden und also $s = t$ gilt

Konstantenbenennung

val $k = E$

ML

```
- datatype intlist = nil | cons of int * intlist
- val primes = cons(3, cons(5, cons(7, nil)))
```


Variableneinführung

var v = **new** T

ML

```
- val ivar = ref (0.0, 1.0);
```


Variablenbenennung

var $v = V$

ML

```
- val myvar = ivar;
> val myvar = ref(0.0,1.0): real * real;
```

Aliase erschweren das Verstehen von Programmen

Funktionsbenennung / Prozedurbenennung

fun $f = (P) \rightarrow T$ => E **proc** $p = (P)$ => C

ML

```
- fun even (N:int) = (n mod 2) = 0;
- % oder: val even = fn (N:int) => (n mod 2) = 0;
> val even = fn int->bool

- fun read (N:int ref) = … ; ();
> val read = fn int ref->unit
```

Bindung und Benutzung von Namen

bindendes Auftreten

die Stelle, an der er gebunden wird

benutzendes Auftreten

eine Stelle, an der er gemäß der Bindung benutzt wird

Bindung	Benutzung
Typbenennung Typeinführung	*Typausdrücke* in Vereinbarungen von Typen, Variablen, Funktionen, ...
Konstantenbenennung	*Ausdrücke* in Feldtypen, Zuweisungen, ...
Variablenbenennung Variableneinführung	*Ausdrücke* und *Zuweisungen* in Konstantenbenennungen, Variableneinführungen, ...
Funktionsbenennung Prozedurbenennung	*Ausdrücke* *Befehle*

zusammengesetzte Vereinbarungen

die Abarbeitung einer Vereinbarung liefert eine Bindung

elab: $D \times (Id \to G_\perp) \to Id \times G_\perp$ Notation: $[\![D]\!]\gamma = \gamma'$

$[\![$ **Kind** $x = G]\!]\gamma = (x, g(\textbf{Kind}, G))$

Die Abarbeitung mehrerer Vereinbarungen *kombiniert* Bindungen

aber wie?

- sequentiell
- parallel
- rekursiv

Kombination: parallel und sequenziell

$D_1 \,||\, D_2$ $D_1 ; D_2$

$[\![D_1 \,||\, D_2]\!]\gamma = \gamma \oplus \{ [\![D_1]\!]\gamma, [\![D_2]\!]\gamma \}$ $[\![D_1 ; D_2]\!]\gamma = [\![D_2]\!](\gamma \oplus [\![D_1]\!]\gamma)$

ML

```
val sin = fn (x: real) => ...
and cos = fn (x: real) => ... ;
val tan = fn (x: real) => sin(x) / cos(x)
```


rekursive Kombination

rec D

$[\![$ **rec** $D]\!]\gamma = \gamma' \cong \gamma' \oplus [\![D]\!]\gamma'$

ML

```
val rec fac =
   fn (x: int) =>if n > 0 then fac(n-1) else 1
```

$\gamma \Rightarrow \gamma \oplus (\mathtt{fac}, \perp) \Rightarrow \gamma \oplus (\mathtt{fac}, \mathrm{Int} \to \mathrm{Int})$

Blockbefehle

let D **in** C

ML

```
if m > n thenlet val t = ref(!m)
             in m := !n; n := !t
             end
```


Blockausdrücke

let D **in** E

ML

```
let var s = (a + b + c) * 0.5
in sqrt(s * (s - a) * s - b) * (s - c)
end
```


Blockvereinbarungen

let D **in** D'

ML

```
local
 fun multiple (n: int, d: int) = (n mod d = 0)
in
 fun leap (y: int) = (multiple(y,4)
                     andalso not multiple(y, 100))
                     orelse  multiple(y,400)
end
```


das *Qualifikationsprinzip*

Definition

Für jedes Programmstück B
sollen *lokal* Vereinbarungen getroffen werden können

Gültigkeit versus Lebensdauer von Variablen

Beispiel (ML)

```
let val odds = ref(cons(1, nil));
    val primes = ref(nil)
in  val odds = ref(cons(3, cons(5, cons(7, nil))));
    primes := ref(2, !odds)
end
```


statische und dynamische Bindung

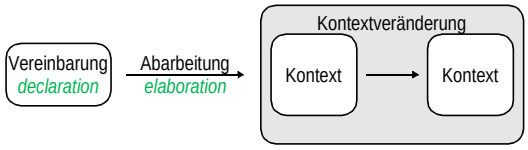
wann werden nicht-lokale Namen gebunden?

Beispiel (ML)

```
let val n = 1
in  let m = n * n
    in  let n = 1000
        in m
        end
    end
end
```

Welchen Wert hat `m` im innersten Block?

die Bindungs-Façette



Programmgrößen werden an Namen gebunden

syntaktischer Bereich: *Namen*

semantischer Bereich: *Bindung*

die Abarbeitung von Vereinbarung verändert den Kontext

syntaktischer Bereich: *Vereinbarungen*

semantischer Bereich: *Kontext, Bindungsänderungen*