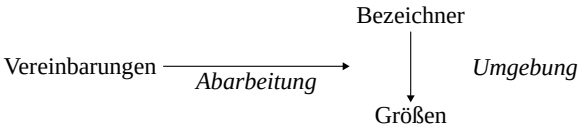
BINDUNG

die Bindungsfacette von Programmiersprachen



Inhalt

Umgebung

Bindung

eine Bezeichnung für eine Größe (*entity*) vereinbaren (*deklarieren*)

Umgebung (*environment*) eines Programmstücks

eine Menge von Bindungen,
oder eine partielle Abbildung von Bezeichnern auf Größen

E: Identifier → Entity

Annahme: kein Überladen (*overloading*)

Beispiel: Umgebungen in einem Pascal-Programm

```
program p;
const z = 0;
var c: Char;
procedure q;
   const c = 3.0e6;
   var   b : Boolean;
   begin
   ...
    end
begin
   ...
end.
```

b	→	**var** Boolean
c	→	**const** Real (3 000 000.0)
q	→	**procedure**
z	→	**const** Integer (0)

c	→	**var** Char
q	→	**procedure**
z	→	**const** Integer (0)

Benennbare Größen

Welche Arten von Größen können benannt werden?

in Pascal:
- primitive Werte und Zeichenketten (in Konstantendefinitionen)
- Variablen (in Variablenvereinbarungen)
- Prozeduren / Funktionen (in Prozedurvereinbarungen)
- Typen (in Typdefinitionen)

in Haskell:
- primitive und zusammengesetzte Werte, Funktionen (mit =)
- Typen, Typabkürzungen (mit **`data`**= und **`type`**)
- abstrakte Datentypen (mit **`abstype`**)

Gültigkeitsbereich (*scope*) einer Vereinbarung

der Teil eines Programms, in dem sie gilt

Block

Grenze für den Gültigkeitsbereich von Vereinbarungen

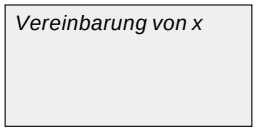
Gültigkeitsbereich *versus* Lebensdauer

Gültigkeitsbereich ist eine statische Eigenschaft (von Vereinbarungen)
Lebensdauer ist eine dynamische Eigenschaft (von Variablen)

Blockstruktur

monolithische Struktur (Cobol)

alle Vereinbarungen gelten
für das gesamte Programm
alle Bezeichner müssen
verschieden sein



flache Struktur (Fortran)

separate Gültigkeitsbereiche:
global fürs Programm
lokal für Unterprogramme

geschachtelte Struktur (Algol-60 ff.)

Blöcke können beliebig
geschachtelt werden
für jedes Programmstück können
lokale Namen vereinbart werden

Schachtelung ist nützlich ...

... reicht alleine aber noch nicht aus (siehe Kapselung)

... in modularen Sprachen nicht unbedingt nötig

Sichtbarkeit von Vereinbarungen

bindendes Vorkommen eines Bezeichners
eine Programmstelle, wo der Bezeichner vereinbart wird
(ein Bezeichner kann mehrmals vereinbart werden)

anwendendes Vorkommen eines Bezeichners
alle anderen Vorkommen des Bezeichners

Vereinbarungsprinzip
zu jedem anwendenden Vorkommen eines Bezeichners
muß es ein passendes bindendes Vorkommen geben

Sonderfall: vordefinierte Größen sind in einem
fiktiven Block vereinbart, der das Programm umschließt

Verdecken von Vereinbarungen
die lokale Vereinbarung eines Bezeichners
verdeckt eine globale Vereinbarung
desselben Bezeichners

Annahme
- statische Bindung (siehe weiter hinten)
- jede lokale Vereinbarung gilt im gesamten Block
- kein Überladen

statische / dynamische Bindung

Bindung nicht-lokaler Bezeichner in Prozeduren
statisch: an die bei der Prozedurvereinbarung gültige Vereinbarung
dynamisch: an die beim Prozeduraufruf gültige Vereinbarung

Beispiel

```
const s = 2;
var   h : Integer;
function scaled (d : Integer) : Integer;
   begin
   scaled := d * s
   end;
procedure … ;
   const s = 3;
   begin
   … scaled (h) …statisch:      scaled(h) = 2*h
                  dynamisch:    scaled(h) = 3*h
   end;
begin
... scaled (h) … scaled(h) = 2*h
end;
```

Folgen
statische Bindung kann zur Übersetzungszeit festgelegt werden
- ist einfach zu übersetzen

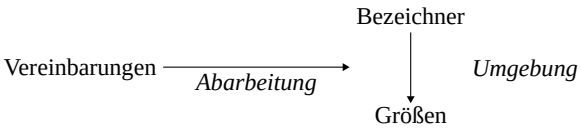
dynamische Bindung hängt vom Kontrollfluß des Programms ab
- Effekte sind schwerer vorherzusehen
- verträgt sich nicht mit statische Typisierung
  Beispiel: `const s = "a";`

VEREINBARUNG (*declaration*)

Was ist eine Vereinbarung?
Ein Programmstück, daß abgearbeitet (*elaboriert*) wird,
um Bindungen herzustellen
(und eventuell Zellen anzulegen o.ö.)

Arten von Vereinbarungen
- Definition
- Einführung
- unabhängige Kombination von Vereinbarungen
- sequenzielle Kombination von Vereinbarungen
- rekursive Vereinbarung

Definitionen sind einfache Vereinbarungen,
die nur Bindungen herstellen



Konstantendefinition

bindet einen Bezeichner an einen Wert

Beispiel:

```
const minint  = -maxint;

      (* statisch, aber in Pascal verboten *)
      letters = ['a' .. 'z', 'A' .. 'Z'];
      minchar = chr(0);
      halfpi  = 0.5 * pi;
```

der Wert muß statisch sein (warum eigentlich?)

Gegenbeispiel (Haskell):

```
f g x = positive v
where   v          = g x
        positive   = (> 0)
```

Funktionsdefinition (und Prozedurdefinition)

führt einen Bezeichner für eine Funktion (bzw. Prozedur) ein

Beispiel (Ada):

```
function even (n: Integer) return Boolean is
   begin
   return (n mod 2 = 0)
   end even;
```

Beispiel (Haskell)
Konstantendefinition eiens funktionalen Wertes

```
even = \ n -> (n mod 2 = 0)
```

mehr dazu unter Abstraktion

Typdefinition und -einführung

Typdefinition
ein neuer Name für einen existierenden Typ (ein Synonym)

Beispiel (Haskell):

```
type book   == (string, int)
type author == (string, int)
```

paßt gut zu struktureller Typäquivalenz
weil `book` ≡ `author`

Typeinführung
ein neuer Name für einen neuen Typ (verschieden von allen anderen)

Beispiel (Haskell):

```
data book'   ::= Bk string num
data author' ::= Au string num
```

paßt gut zu namentlicher Typäquivalenz
weil `book'` und `author'` verschieden sind

Typdefinition und -einführung in Pascal

jeder Typkonstruktor (**`record`**, **`array`**, **`set`**)
führt einen neuen Typ ein, der zunächst anonym bleibt

eine Typdefinition bindet einen
(neuen oder alten) Typen an einen Bezeichner

Beispiel

```
type Person =
   record
      name : packed array [1..10] of Character;
      age: Integer
   end;
type Title  = packed array [1..10] of Character;
     Int    = Integer
```

Dann gilt:

- Der Typ von `person.name` ist verschieden von `Title`
- `Int` ≡ `Integer`

Variablendefinition und -Vereinbarung

Variablendefinition (Umbenennung)
ein neuer Name für eine existierende Variable (ein alias)

Beispiele:

```
pop : Integer renames population(state); (Ada)
ref int pop = population[state]          (Algol-68)
val pop = population sub state           (ML)
```

Aliasse machen Programme schwerer verständlich
und härter zu überprüfen

Beispiel:

```
p (pop, population(state));
```

Variableneinführung
ein neuer Name für eine neu angelegte Variable

Beispiele:

```
count : Integer := 0;          (Ada)
ref int count = loc int := 0;  (Algol-68)
val count = ref 0              (ML)
```

Unabhängige und sequenzielle Kombination (v. V.)

unabhängige Kombination (*collateral*)

Form: D_1 **and** D_2

D_1 und D_2 werden unabhängig voneinander ausgearbeitet,

die entstehenden Bindungen werden dann vereinigt

Beispiel (ML):

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

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

falsch: `sin` noch nicht bekannt!

sequenzielle Kombination

Form: D_1 ; D_2

D_1 und D_2 werden nacheinander ausgearbeitet

Beispiel (ML):

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


rekursive Vereinbarungen

Rekursive Vereinbarungen

Form: **rec** *D*

die von *D* hergestellte Bindung gilt schon innerhalb von *D*

Beispiel (Pascal)

Typdefinitionen werden automatisch rekursiv kombiniert:

```
type Intlist = ^ Intnode
     Intnode = record
                 head : Integer;
                 tail : Intlist
               end;
```

Prozedurdefinitionen auch:

```
procedure expression; forward;
procedure primary;
   begin
   ... expression; ...
   end;
procedure expression;
   begin
   ... primary; ... expression; ...
   end;
```

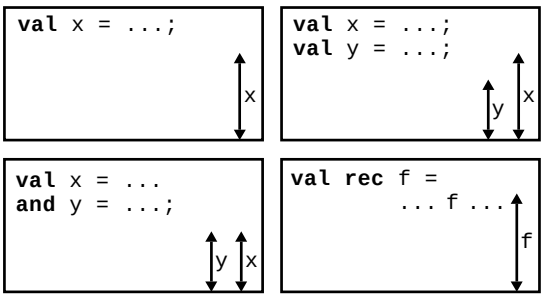
Beispiel (ML)

explizit rekursive Wertdefinition

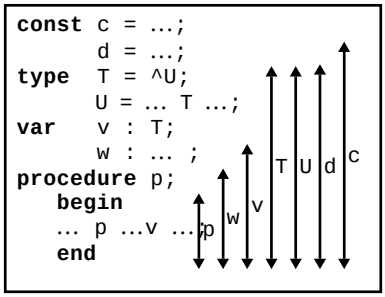
```
val rec fac =
    fn (x: int) => if n > 0 then fac(n-1)
                   else ...
```


Beginn des Gültigkeitsbereichs

Gültigkeitsbereiche in ML



Gültigkeitsbereiche in Pascal



Zusammenfassung

- Konstanten- und Variablenreinführung sequenziell, nicht rekursiv
- Typ- und Prozedurdefinition sequenziell, rekursiv

Blöcke

Blockbefehl

Form

declare *D* **begin** *C* **end** (*D* **begin** *C* **end** in Pascal)

vereinbare *D* lokal für die Befehle *C*

In Pascal: eingeschränkt für Programm und Prozeduren / Funktionen
In Algol- 68 und Ada: ein beliebiger Befehl

Blockausdruck

Form

let *D* **in** *E*

vereinbare *D* lokal für den Ausdruck *E*

Das Qualifikationsprinzip

Für jedes Programmstück, daß Berechnungen spezifiziert,
können lokale Vereinbarungen getroffen werden

Blockvereinbarung (ML)

vereinbare *D* lokal für die Vereinbarungen *F*

local *D* **in** *F* **end**

Bindungen in Ada

Was für Vereinbarungen gibt es?

Konstanten
Variablen
Prozeduren und Funktionen (auch generisch)
Typen und Untertypen
Repräsentationen
Marken
Pakete (auch generisch)
Ausnahmen
Aufgaben (*task*)
Einträge (*entry*)

Gibt es implizite Vereinbarungen?

in einer Zählschleife wird die Schleifenvariable vereinbart

Welche Arten von Größen können an Namen gebunden werden?

Werte (statische und dynamische)
Variablen
Prozeduren und Funktionen
Typen und Untertypen
Programmstellen (Marken und Schleifen)
Pakete
Ausnahmen
Aufgaben
Einträge

Reichweiten in Ada

Wo beginnt die Reichweite in einem Block?

lineare Sichtbarkeit, konsequent durchgehalten

die Bindung eines Bezeichner muß
textuell vor seiner ersten Anwendung stehen!

```
type cell; -- Vorwärtsspezifikation
type list is access cell;
type cell is  record
                head: Integer;
                tail: list;
              end record;
```

was sind hier *bindende* und *anwendende* Vorkommen?

Detailfragen

ab wann ist eine Vereinbarung verdeckt?

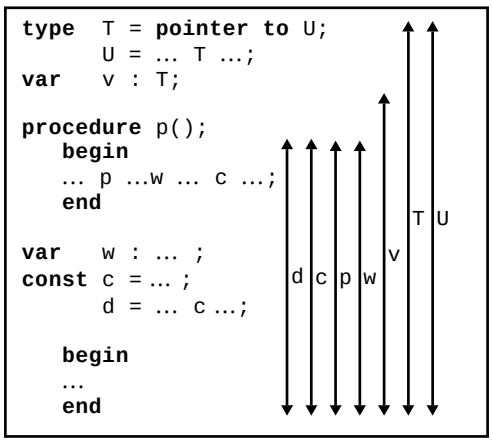
```
declare
   c : constant Integer := 0;
   ..
   declare
      c2 : constant Integer := .. c ..;
      c : Character := 'a';
   begin
   ...
   end;
end;
```

In Pascal ist das anwendende Vorkommen von **c** ein Fehler!

Gilt das *Qualifikationsprinzip*?

- es gibt keinen Block*ausdruck*
- Block*vereinbarung* kann mit Paketen realisiert werden

Übung: Beginn der Reichweite in Modula-2



Sichtbarkeitsregeln von Modula-2

Auszug aus N. Wirth: *Programming in Modula-2*