

# Programmierung 1

Vorlesung 4

*Livestream beginnt um 10:20 Uhr*

*Programmiersprachliches, Teil 2*  
*Höherstufige Prozeduren*

Programmierung 1

# Verarbeitungsphasen eines Interpreters

**Lexikalische Analyse**  
*Lexikalische Syntax*

Kann die eingegebene Zeichenfolge als **Folge von Wörtern** aufgefasst werden?

**Syntaktische Analyse**  
*Phrasale Syntax*

Beschreibt die Wortfolge ein **Programm?**

**Semantische Analyse**  
*Statische Semantik*

Ist das Programm **semantisch zulässig?**

**Ausführung**  
*Dynamische Semantik*

**Führe** das Programm **aus**.

# Semantische Zulässigkeit

- ▶ Ein semantisch zulässiges Programm muss **geschlossen** sein:

- ▶ Die **freien Bezeichner einer Phrase** sind die Bezeichner, die in der Phrase ein Auftreten haben, das **nicht** im Rahmen der Phrase **gebunden** ist.

- ▶ Phrasen **ohne** freie Bezeichner bezeichnet man als **geschlossen** und Phrasen **mit** freien Bezeichnern als **offen**.

- ▶ Beispiele:

```
fun q (x:int) = 3+(p x) offen
```

```
val x = x offen
```

```
fun f (x:int) : int = if x<1 then 1 else x*f(x-1)  
geschlossen
```

- ▶ Ein semantisch zulässiges Programm muss **wohlgetypt** sein.

# Typregeln

$$\frac{e_1 : t_1 \quad o : t_1 * t_2 \rightarrow t \quad e_2 : t_2}{e_1 o e_2 : t}$$

Die Regel besagt, dass eine Anwendung  $e_1 o e_2$  **wohlgetypt** ist und **den Typ  $t$  hat**, wenn

1. der linke Teilausdruck  $e_1$  den Typ  $t_1$  hat,
2. der Operator  $o$  den Typ  $t_1 * t_2 \rightarrow t$  hat, und
3. der rechte Teilausdruck  $e_2$  den Typ  $t_2$  hat.

**Beispiel:** Ausdruck  $x+5$

Wenn wir annehmen, dass  $x$  den Typ `int` hat, dann folgt, weil  $+$  den Typ `int * int → int` und `5` den Typ `int` hat, dass  $x+5$  wohlgetypt ist und den Typ `int` hat.

# Inferenzregeln

Allgemein hat eine **Inferenzregel** die Form

$$\frac{P_1 \quad \dots \quad P_n}{P}$$

$P_1 \quad \dots \quad P_n$  sind die **Prämissen** der Inferenzregel,  
 $P$  ist die **Konklusion**.

Die Regel besagt, dass die **Konklusion** gilt  
**wenn** die **Prämissen** gelten.

$$\frac{e_1 : \text{bool} \quad e_2 : t \quad e_3 : t}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t}$$

$$\frac{e_1 : t_1 \quad \cdots \quad e_n : t_n}{(e_1, \dots, e_n) : t_1 * \cdots * t_n}$$

$$\frac{e_1 : t_1 \rightarrow t_2 \quad e_2 : t_1}{e_1 e_2 : t_2}$$

# Ableitungsbaum

Ist der Ausdruck

*if if true then false else true then 10 else 2\*3*

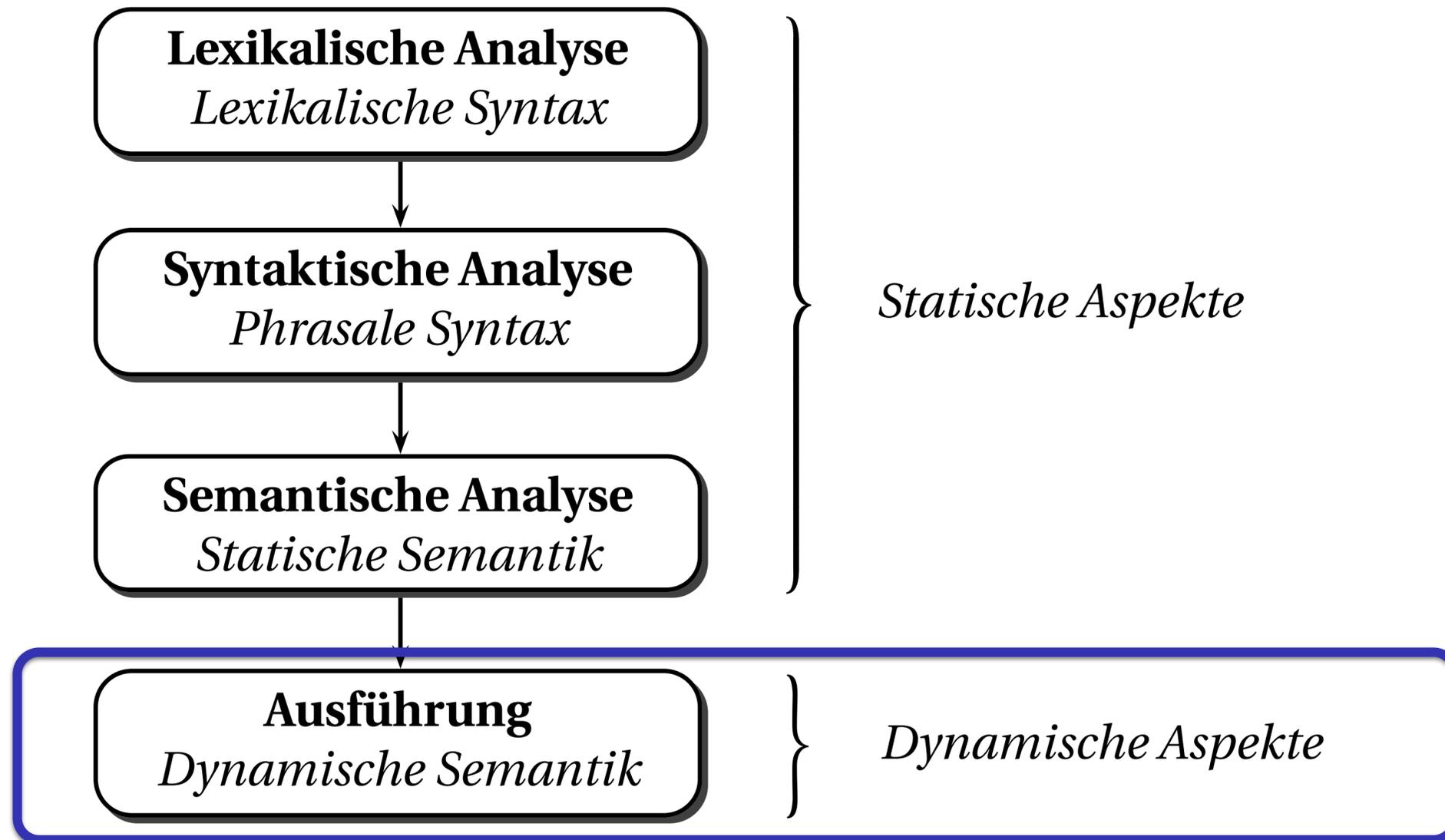
wohlgetypt und hat den Typ *int*?

$$\frac{\frac{\frac{}{true:bool} \quad \frac{}{false:bool} \quad \frac{}{true:bool}}{if\ true\ then\ false\ else\ true : bool} \quad \frac{}{10:int}}{if\ if\ true\ then\ false\ else\ true : bool} \quad \frac{\frac{}{2:int} \quad \frac{}{*:int*int \rightarrow int} \quad \frac{}{3:int}}{2*3 : int}}{if\ if\ true\ then\ false\ else\ true\ then\ 10\ else\ 2*3 : int}}$$

$$\frac{e_1 : t_1 \quad o : t_1 * t_2 \rightarrow t \quad e_2 : t_2}{e_1\ o\ e_2 : t}$$

$$\frac{e_1 : bool \quad e_2 : t \quad e_3 : t}{if\ e_1\ then\ e_2\ else\ e_3 : t}$$

# Verarbeitungsphasen eines Interpreters



# Umgebungen

- ▶ Eine **Umgebung** ist eine Sammlung von **Bezeichnerbindungen**.

Beispiel:  $[x:=5, y:=7]$

Der Bezeichner  $x$  ist an den Wert 5 gebunden,  
der Bezeichner  $y$  an den Wert 7.

- ▶ Um den **Wert** eines Ausdrucks mit **freien Bezeichnern** zu bestimmen, benötigen wir eine Umgebung:

$x*y$  hat den Wert 35 in der Umgebung  $[x:=5, y:=7]$

$x*y$  hat den Wert 40 in der Umgebung  $[x:=5, y:=8]$

# Tripeldarstellung von Prozeduren

**Prozedur = (Code, Typ, Umgebung)**

Die **Tripeldarstellung** einer **Prozedur** besteht aus

- ▶ dem **Code** (einer Prozedurdeklaration)
- ▶ dem **Typ** der Prozedur, und
- ▶ einer **Umgebung**, die die freien Bezeichner bindet.

**Beispiel:** `val a = 2*7`  
`fun p (x:int) = x+a`

Die Ausführung bindet  $p$  an die Prozedur

$$\left( \underbrace{\text{fun } p \ x = x + a}_{\text{Code}}, \underbrace{\text{int} \rightarrow \text{int}}_{\text{Typ}}, \underbrace{[a := 14]}_{\text{Umgebung}} \right)$$

# Adjunktion von Umgebungen

**Adjunktion** ist eine **Operation**  $V_1 + V_2$   
die zwei Umgebungen  $V_1, V_2$  zu einer Umgebung **kombiniert** :

$$[x:= 5, y:= 7] + [x:= 1, z:= 3] = [x:= 1, y:= 7, z:= 3]$$

$$[x:= 1, z:= 3] + [x:= 5, y:= 7] = [x:= 5, y:= 7, z:= 3]$$

**Adjunktion ist nicht kommutativ.**

Die Bindungen der rechten Umgebung überschreiben die Bindungen der linken Umgebung.

# Frage

Was ist  $[x:=1, y:=3] + [y:=4, z:=2]$  ?

- ▶  $[x:=1, z:=2]$
- ▶  $[y:=3, z:=2]$
- ▶  $[x:=1, y:=3, z:=2]$
- ▶  $[x:=1, y:=4, z:=2]$

# Ausführung (auch: Auswertung, Evaluation)

- ▶ **Ausführung von Programmen.** Gegeben ein Programm  $P$  und eine Umgebung  $V$ , bestimme die Umgebung, die  $P$  für  $V$  liefert.
- ▶ **Ausführung von Deklarationen.** Gegeben eine Deklaration  $D$  und eine Umgebung  $V$ , bestimme die Umgebung, die  $D$  für  $V$  liefert.
- ▶ **Ausführung von Ausdrücken.** Gegeben ein Ausdruck  $e$  und eine Umgebung  $V$ , bestimme den Wert, den  $e$  für  $V$  liefert.
- ▶ **Ausführung von Prozeduranwendungen.** Gegeben eine Prozedur  $p$  und einen Wert  $v$ , bestimme den Wert, den  $p$  für  $v$  liefert.

# Ausführung von Programmen

- ▶ **Aufbau eines Programms** (siehe syntaktische Gleichung):

$\langle \text{Programm} \rangle ::= \langle \text{Deklaration} \rangle \dots \langle \text{Deklaration} \rangle$

- ▶ **Ausführung eines Programms** in einer **Umgebung**  $V$ :

- ▶ Wenn das Programm **leer** ist,  
wird die Umgebung  $V$  geliefert.
- ▶ Wenn das Programm die **Form**  $DP$  hat,  
wird **zunächst** die **Deklaration**  $D$  in  $V$  ausgeführt.  
Wenn das die Umgebung  $V'$  liefert,  
wird das **Restprogramm**  $P$  in  $V'$  ausgeführt und  
das Programm liefert die so erhaltene Umgebung.

# Ausführung von Deklarationen

- ▶ **Aufbau einer Deklaration** (siehe syntaktische Gleichung):

$\langle \text{Deklaration} \rangle ::= =$

$\langle \text{Val-Deklaration} \rangle$

|  $\langle \text{Prozedurdeklaration} \rangle$

- ▶ **Ausführung einer Deklaration** in einer **Umgebung  $V$** :

1. Bei einer **Val-Deklaration**  $\text{val } M = e$  wird zuerst der Ausdruck  $e$  in  $V$  ausgeführt.

Wenn das den Wert  $v$  liefert, wird die Umgebung  $V'$  bestimmt, die die Variablen des Musters  $M$  gemäß  $v$  bindet.

Die Deklaration liefert dann die Umgebung  $V+V'$ .

# Ausführung von Deklarationen

- ▶ **Aufbau einer Deklaration** (siehe syntaktische Gleichung):

$\langle \text{Deklaration} \rangle ::= =$

$\langle \text{Val-Deklaration} \rangle$

|  $\langle \text{Prozedurdeklaration} \rangle$

- ▶ **Ausführung einer Deklaration** in einer **Umgebung**  $V$ :

2. Eine **Prozedurdeklaration**  $\text{fun } f \ M = e$  oder  $\text{fun } f \ M : t = e$  liefert die Umgebung  $V+[f:=(\text{fun } f \ M' = e, t', V')]$  wobei  $M'$ ,  $t'$  und  $V'$  wie folgt bestimmt sind:

$M'$  ergibt sich aus  $M$  durch Löschen der Typen;

$t'$  ist der für die deklarierte Prozedur ermittelte Typ;

$V'$  besteht aus den Bindungen von  $V$ ,

die die **freien Bezeichner** der Prozedurdeklaration binden.

# Beispiel

```
val a = 2*7
fun p (x:int) = x+a
fun q (x:int) = x + p x
```

Die Ausführung liefert:

```
a := 14
p := (fun p x = x + a, int → int, [a := 14])
q := (fun q x = x + p x, int → int,
      [p := (fun p x = x + a, int → int, [a := 14])])
```

# Beispiel

```
fun p (x:int) = x
fun q (x:int) = p x
fun p (x:int) = 2*x
val a = (p 5, q 5)
```

Die Ausführung liefert:

```
p := (fun p x = 2 * x, int → int, [])
q := (fun q x = p x, int → int, [p := (fun p x = x, int → int, [])])
a := (10, 5)
```

**statisches** (auch: **lexikalisches**) **Bindungsprinzip**:

**Der Rumpf einer Prozedur arbeitet immer mit den Bindungen, die bei der Ausführung der Deklaration der Prozedur vorlagen.**

# Ausführung von Ausdrücken

- ▶ **Aufbau eines Ausdrucks** (siehe syntaktische Gleichungen):

$\langle \text{Ausdruck} \rangle ::=$

$\langle \text{atomarer Ausdruck} \rangle$

    |  $\langle \text{Anwendung} \rangle$

    |  $\langle \text{Konditional} \rangle$

    |  $\langle \text{Tupelausdruck} \rangle$

    |  $\langle \text{Let-Ausdruck} \rangle$

    |  $( \langle \text{Ausdruck} \rangle )$

- ▶ **Ausführung eines Ausdrucks** in einer **Umgebung V**:

1. Die Ausführung eines Ausdrucks, der nur aus einer **Konstanten** besteht, liefert den durch die Konstante bezeichneten Wert.
2. Die Ausführung eines Ausdrucks, der nur aus einem **Bezeichner** besteht, liefert den von V für den Bezeichner gegebenen Wert.

# Ausführung von Ausdrücken

▸ **Ausführung eines Ausdrucks** in einer **Umgebung**  $V$ :

3. ...

4. ...

5. ...

6. Die Ausführung einer **Prozeduranwendung**  $e_1 e_2$  beginnt mit der Ausführung der Teilausdrücke  $e_1$  und  $e_2$  in  $V$ .

Wenn diese die **Prozedur**  $p$  und den **Wert**  $v$  liefern,

wird der **Prozeduraufruf**  $p v$  ausgeführt.

Die Prozeduranwendung liefert den so erhaltenen Wert.

# Ausführung von Prozeduraufrufen

Bei einem **Aufruf einer Prozedur**  $p = (\text{fun } f M = e, t, V)$   
mit einem Wert  $v$

- ▶ wird zuerst die Umgebung  $V'$  bestimmt, die die Variablen des **Musters**  $M$  gemäß dem Argument  $v$  bindet.
- ▶ Dann wird der **Prozedurrumpf**  $e$   
in der Umgebung  $(V + [f := p]) + V'$  ausgeführt.

Der Prozeduraufruf liefert den so erhaltenen Wert.

Die Bindung  $f := p$  ermöglicht **rekursive Prozeduraufrufe**.

# Ausführung von Prozeduraufrufen

## ▶ Ausführung eines Ausdrucks in einer Umgebung $V$ :

7. Die Ausführung eines **Konditionals** `if  $e_1$  then  $e_2$  else  $e_3$`  beginnt mit der Ausführung der **Bedingung**  $e_1$  in  $V$ .  
Wenn  $e_1$  den Wert `true` liefert, wird die **Konsequenz**  $e_2$  in  $V$  ausgeführt und das Konditional liefert den so erhaltenen Wert.  
Wenn  $e_1$  den Wert `false` liefert, wird die **Alternative**  $e_3$  in  $V$  ausgeführt und das Konditional liefert den so erhaltenen Wert.
8. Die Ausführung eines **Tupelausdrucks** `( $e_1, \dots, e_n$ )` beginnt mit der Ausführung der Teilausdrücke  $e_1, \dots, e_n$  in  $V$ . Wenn diese die Werte  $v_1, \dots, v_n$  liefern, liefert der Tupelausdruck das **Tupel** `( $v_1, \dots, v_n$ )`.
9. Die Ausführung eines **Let-Ausdrucks** `let  $P$  in  $e$  end` beginnt mit der Ausführung des **Programms**  $P$  in  $V$ . Wenn diese die Umgebung  $V'$  liefert, wird der **Ausdruck**  $e$  in  $V'$  ausgeführt. Der Let-Ausdruck liefert den so erhaltenen Wert.

# Semantische Äquivalenz

Zwei Programme sind **semantisch äquivalent**, wenn sie sich bezüglich statischer und dynamischer Semantik nach außen hin gleich verhalten.

`val h = 3*2` ist äquivalent zu `val h = 3+3`

`fun f(x:int) = 2+x` ist äquivalent zu `fun f(y:int) = 2+y`

`val z = (x-y) * (x-y)` ist äquivalent zu `val z = x*x - 2*x*y + y*y`

**Hierbei wird ein idealisierter Interpreter angenommen.**

Semantisch äquivalente Programme zeigen nicht notwendigerweise das gleiche Verhalten in SOSML.

## Kapitel 3

# Höherstufige Prozeduren

# Abstraktionen

**Abstraktionen** sind **Ausdrücke** die **Prozeduren** beschreiben.

## Beispiele:

```
fn (x:int) => x*x
```

```
fn : int → int
```

```
(fn (x:int) => x*x) 7
```

```
49 : int
```

```
fn (x:int, y:int) => x*y
```

```
fn : int * int → int
```

```
(fn (x:int, y:int) => x*y) (4,7)
```

```
28 : int
```

# Prozedurdeklarationen

Prozedurdeklaration:

```
fun f ... = ...
```

**abkürzende Schreibweise** für:

```
val f = fn ... => ...
```

Für **rekursive Prozeduren**:

```
val rec f = fn ... => ...
```

**Beispiel:**

```
fun p (x:int, n:int) :int
    = if n>0 then x*p(x,n-1) else 1
val rec p:int*int->int = fn (x:int, n:int)
    => if n>0 then x*p(x,n-1) else 1
```

# Kaskadierte Prozeduren

**Prozeduren** die **Prozeduren als Ergebnis** liefern, werden als **kaskadiert** bezeichnet.

## Beispiel:

```
fun mul (x:int) = fn (y:int) => x*y
```

```
val mul : int → (int → int)
```

```
mul 7
```

```
fn : int → int
```

```
it 3
```

```
21 : int
```

```
mul 7 5
```

```
35 : int
```

[www.prog1.saarland](http://www.prog1.saarland)