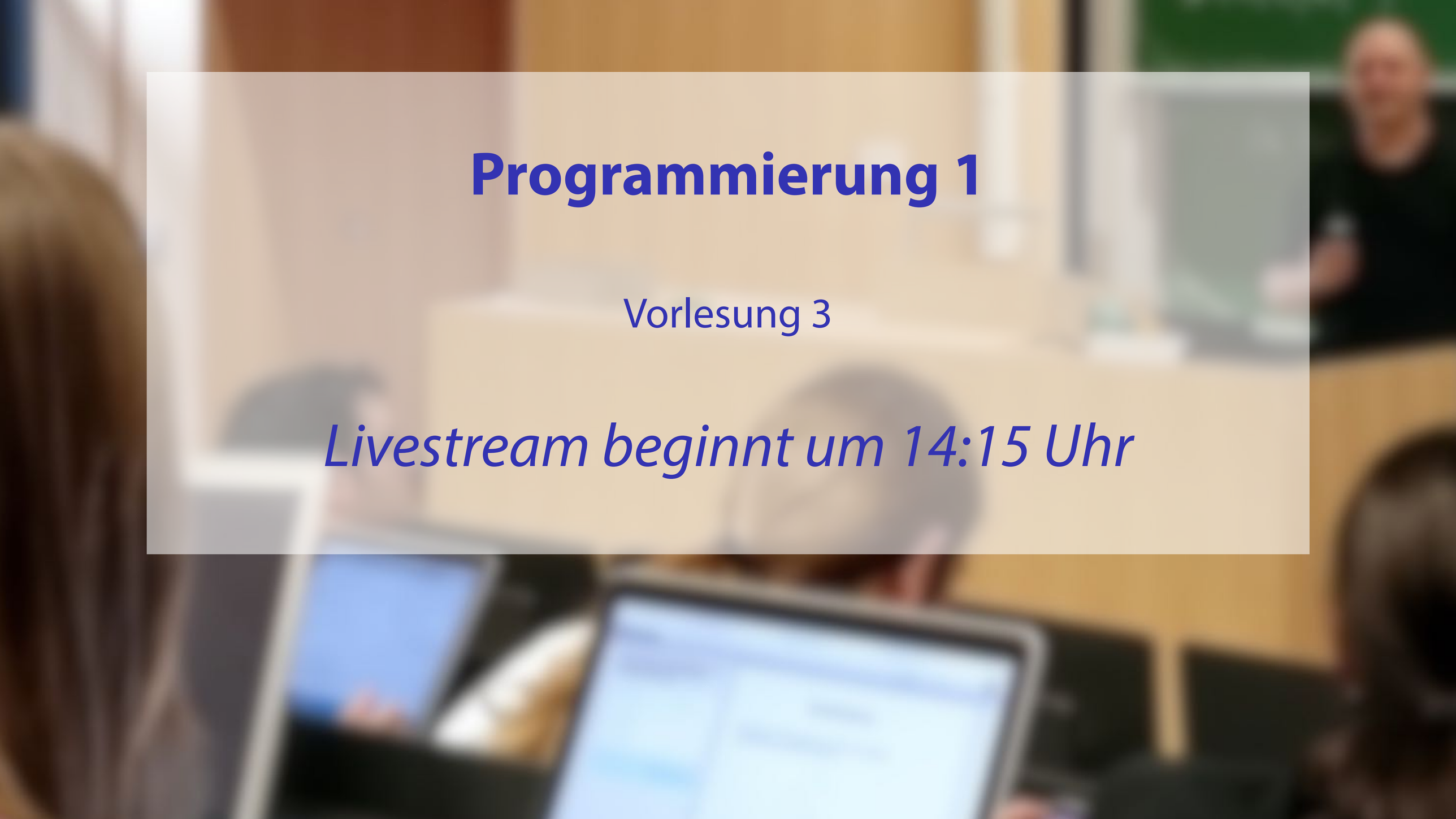


Programmierung 1

Vorlesung 3

Livestream beginnt um 14:15 Uhr

Schnellkurs, Teil 3
Programmiersprachliches

Programmierung 1

Rekursion in ML

```
fun potenz (x:int, n:int) : int =
    if n>0 then x*potenz(x,n-1) else 1
```

*val potenz : int * int → int*

- Rumpf enthält **Selbstanwendung** (auch: **rekursive Anwendung**)
- Bei der Deklaration rekursiver Funktionen geben wir den **Ergebnistyp** an.

Natürliche Quadratwurzel

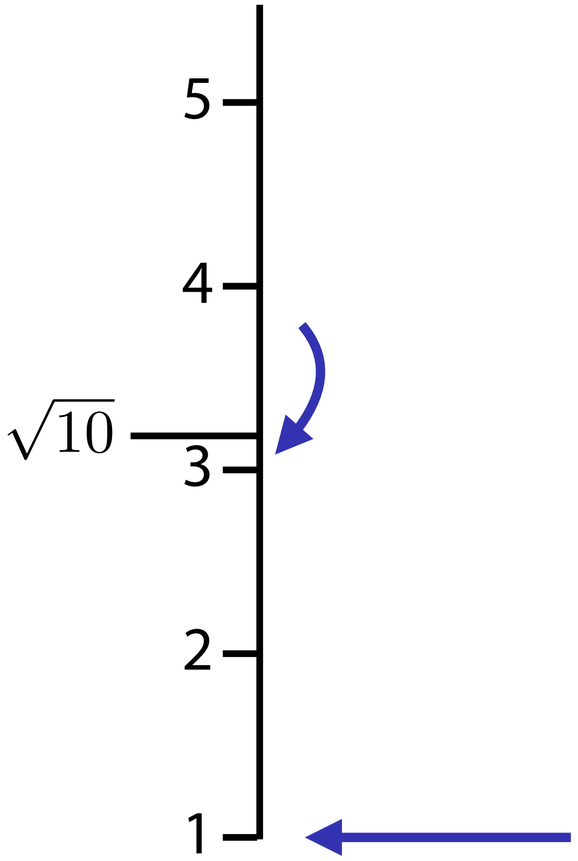
$$\lfloor \sqrt{n} \rfloor = \max\{ k \in \mathbb{N} \mid k^2 \leq n \}$$

Beispiele

- $\lfloor \sqrt{10} \rfloor = 3$
- $\lfloor \sqrt{9} \rfloor = 3$
- $\lfloor \sqrt{8} \rfloor = 2$

Berechnungsidee

$$\lfloor \sqrt{n} \rfloor = \max\{ k \in \mathbb{N} \mid k^2 \leq n \}$$



$4^2>10$ $\quad 4 = \min\{k \in \mathbb{N} \mid k^2 > 10\}$

$3^2\leq 10$ $\quad 3 = \max\{k \in \mathbb{N} \mid k^2 \leq 10\}$
$= \min\{k \in \mathbb{N} \mid k^2 > 10\} - 1$

$2^2\leq 10$

$1^2\leq 10$

Hilfsfunktion

$$\lfloor \sqrt{n} \rfloor = \max\{ k \in \mathbb{N} \mid k^2 \leq n \}$$
$$= \min\{ k \in \mathbb{N} \mid k^2 > n \} - 1$$

- **Hilfsfunktion** $\boxed{\lfloor \sqrt{n} \rfloor = w(1, n) - 1}$

$$w : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$w(k, n) = \min\{ i \in \mathbb{N} \mid i \geq k \text{ und } i^2 > n \}$$

$$w(k, n) = k \qquad \text{für } k^2 > n$$

$$w(k, n) = w(k + 1, n) \qquad \text{für } k^2 \leq n$$

Natürliche Quadratwurzel in ML

```
fun w (k:int,n:int) : int = if k*k>n then k else w(k+1,n)
```

*val w : int * int → int*

```
fun wurzel (n:int) = w(1,n)-1
```

val wurzel : int → int

- **Akkumulator** (kurz: **Akku**):
  zusätzliches Argument in Hilfsfunktion (hier: *k*)

Endrekursion

- Eine rekursive Prozedur heißt **endrekursiv**
  wenn das Ergebnis im Rekursionsfall
  unmittelbar durch eine **rekursive Anwendung**
  geliefert wird.

- endrekursiv:

```
fun w (k:int,n:int) : int =
      if k*k>n then k else w(k+1,n)
```

- nicht endrekursiv:

```
fun potenz (x:int, n:int) : int =
      if n>0 then x*potenz(x,n-1) else 1
```

Endrekursive Prozedur für Potenzfunktion

$$p : \mathbb{Z} \times \mathbb{Z} \times \mathbb{N} \to \mathbb{Z}$$

$$p(a, x, n) = a \cdot x^n$$

$$\boxed{p(1, x, n) = x^n}$$

$$p(a, x, 0) = a$$

$$p(a, x, n) = p(a \cdot x, x, n-1) \qquad \text{für } n > 0$$

```
fun p (a:int, x:int, n:int) : int =
          if n<1 then a else p(a*x,x,n-1)

fun potenz (x:int, n:int) = p(1,x,n)
```

Frage

Welche der folgenden Prozeduren sind endrekursiv?

fun f(x:int) : bool =

- if f x then g x else h x
- if g x then f x else h x
- if g x then h x else f x
- if g x then f x else f (x+1)

Festkomma und Gleitkommazahlen

- **Festkommazahlen:** ganze Zahlen
  SOSML : Typ int [-2^{30} , ... , 2^{30} -1]
  Bei Überlauf: Laufzeitfehler **Overflow**
- **Gleitkommazahlen:** $m \cdot 10^n$
  m: Mantisse
  n: Exponent
  Typ real
  Bei Überlauf: **Rundungsfehler**

Newtonsches Verfahren

$$a_0 = \frac{x}{2}$$

$$a_n = \frac{1}{2}(a_{n-1} + \frac{x}{a_{n-1}}) \quad \text{für n>0}$$

konvergiert gegen $\sqrt{x}$

```
fun newton (a:real, x:real, n:int) : real =
    if n<1 then a else newton (0.5*(a+x/a), x, n-1)
```

*val newton : real * real * int → real*

```
fun sqrt (x:real) = newton (x/2.0, x , 5)
```

val sqrt : real → real

Standardstrukturen

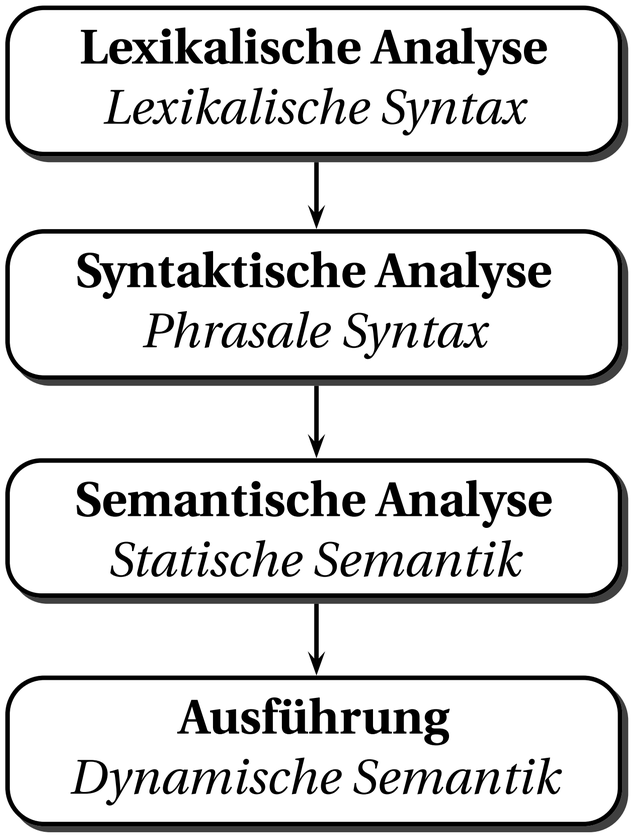
Math.sqrt	:	$real \rightarrow real$	$\sqrt{x}$
Math.sin	:	$real \rightarrow real$	
Math.asin	:	$real \rightarrow real$	
Math.exp	:	$real \rightarrow real$	e^x
Math.pow	:	$real * real \rightarrow real$	x^y
Math.ln	:	$real \rightarrow real$	
Math.log10	:	$real \rightarrow real$	
Real.fromInt	:	$int \rightarrow real$	
Real.round	:	$real \rightarrow int$	
Real.floor	:	$real \rightarrow int$	$\lfloor x \rfloor$ *Rundung nach unten*
Real.ceil	:	$real \rightarrow int$	$\lceil x \rceil$ *Rundung nach oben*
Math.pi	:	*real*	π
Math.e	:	*real*	e

ML stellt Standardprozeduren im Rahmen von Standardstrukturen bereit.

Bezeichner sind zusammengesetzt aus Bezeichner der Struktur und lokalem Bezeichner, z.B. Math.pi

Kapitel 2
Programmiersprachliches

Verarbeitungsphasen eines Interpreters



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

Beschreibt die Wortfolge ein **Programm**?

Ist das Programm **semantisch zulässig**?

Führe das Programm **aus**.

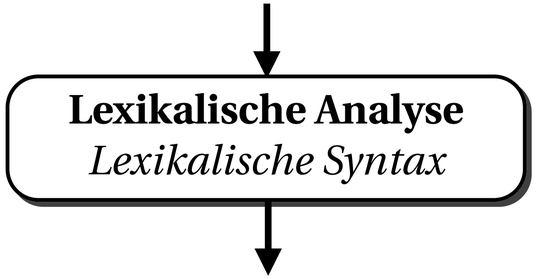
Syntax und Semantik

- **Syntax:** legt die Regeln für die **lexikalische und syntaktische Analyse** fest.
  - **lexikalische Syntax** regelt, wie Wörter aus Zeichen gebildet werden,
  - **phrasale Syntax** regelt, wie Phrasen aus Wörtern gebildet werden. (**Phrase:** Überbegriff für Programme, Deklarationen, Ausdrücke, Typen)
- **Semantik:** legt die Regeln für die **semantische Analyse** und die **Ausführung** fest.
  - **statische Semantik** formuliert Bedingungen, die semantisch zulässige Programme erfüllen müssen (z.B. Wohlgetyptheit).
  - **dynamische Semantik** beschreibt, welche Ergebnisse die Ausführung von Programmen liefern soll.

Zeichen und Wortdarstellung

Zeichendarstellung:

```
if x*y<15then(x,2+y)else paar(2*x-3)
```



Wortdarstellung:

if	x	*	y	<	15	then	(	x	,	2	+	y	)	else	paar	(	2	*	x	-	3	)

- Die **Zeichendarstellung** stellt eine Phrase als **Buchstabenfolge** dar. Leerzeichen (Zwischenraum, Tabulator, Zeilenwechsel) wo nötig.
- Die **Wortdarstellung** stellt eine Phrase als **Wortfolge** dar.

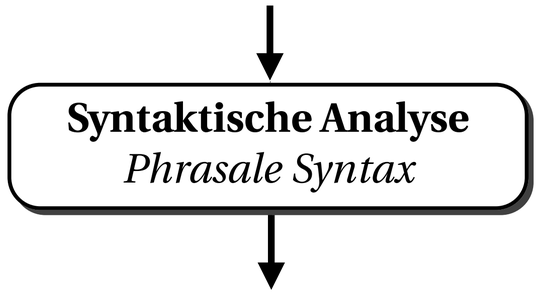
Wörter

- **Konstanten:** z.B. 3, ~56, 1.7, 1.7878E45, false, true, ().
- **Operatoren:** z.B. +, −, *, /, div, mod, <, <=, >, >=, =, <>, ~.
- **Schlüsselwörter:** z.B. val, fun, if , then, else, let, in, end, int, bool, unit, =, (,), : , ->, *, # und das Komma.
- **Bezeichner:** Wörter, die aus Buchstaben, Ziffern sowie den Zeichen "_" (Unterstrich) und "′" (Hochkomma) gebildet sind. **Müssen mit einem Buchstaben beginnen** und dürfen nicht zu einer der anderen Wortarten gehören. Zusammengesetzte Bezeichner durch Punkt verbunden, z.B. "Math.pi ".
- **Doppelrolle:** Vor dem Schlüsselwort if treten = und * als Schlüsselwörter auf, danach als Operatoren:
  z.B. fun f (t: int*int*int) = if #1t=0 then #2t else 2*(#3t)

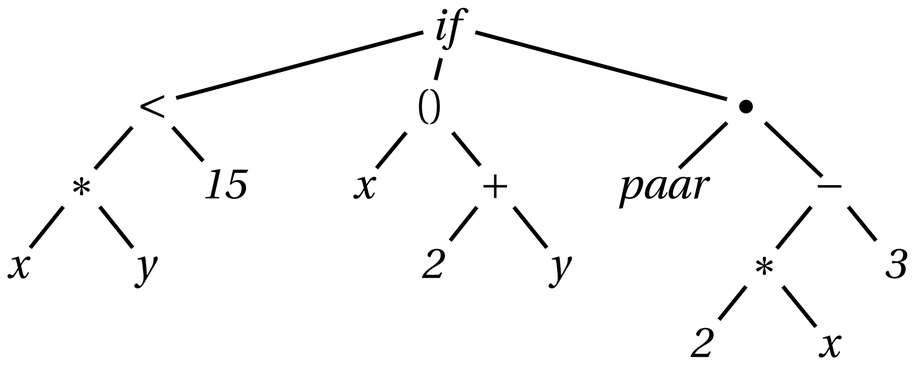
Baumdarstellung

Wortdarstellung:

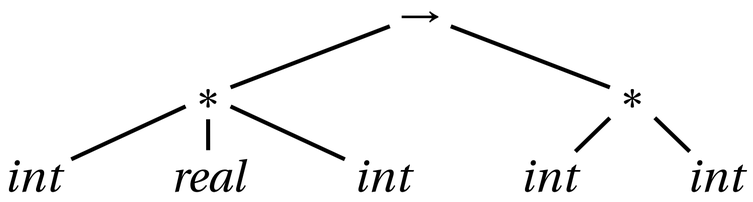
if	x	*	y	<	15	then	(	x	,	2	+	y	)	else	paar	(	2	*	x	-	3	)



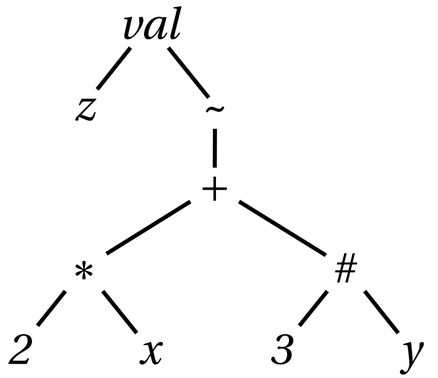
Baumdarstellung:



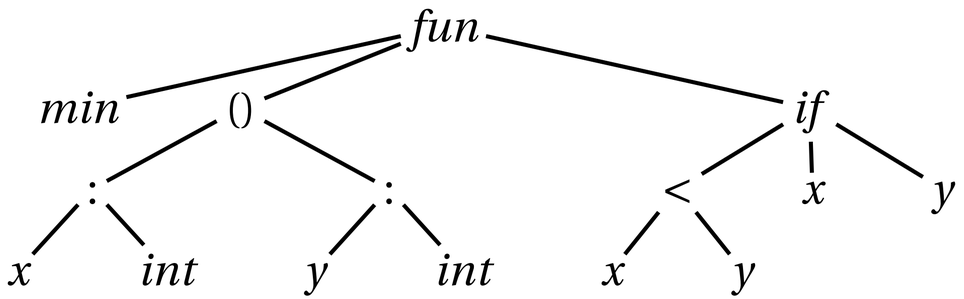
```
int * real * int -> int * int
```




```
val z = ~(2*x + #3y)
```




```
fun min (x:int, y:int) = if x<y then x else y
```



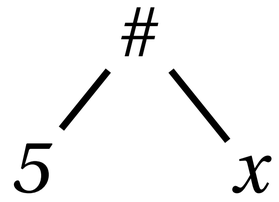
Phrasen

- Phrasen werden in **syntaktische Kategorien** unterteilt:
  u.a. **Ausdrücke**, **Deklarationen**, **Typen**
- Es gibt **atomare** und **zusammengesetzte Phrasen.**
  Die zur Bildung der zusammengesetzten Phrase verwendeten einfacheren Phasen heißen **Komponenten.**
- Syntaktische Kategorien können mit Hilfe von **syntaktischen Gleichungen** beschrieben werden.

Beispiel: Die Gleichung

$$\langle Projektion \rangle ::= \# \langle positive\ ganze\ Zahl \rangle \langle Ausdruck \rangle$$

bedeutet, dass die Projektion
durch den folgenden Baum dargestellt wird:



Syntaktische Gleichungen für Deklarationen

⟨*Programm*⟩ ::= ⟨*Deklaration*⟩ … ⟨*Deklaration*⟩

⟨*Deklaration*⟩ ::=
⟨*Val-Deklaration*⟩
| ⟨*Prozedurdeklaration*⟩

⟨*Prozedurdeklaration*⟩ ::=
fun ⟨*Bezeichner*⟩ ⟨*Argumentmuster*⟩ = ⟨*Ausdruck*⟩
| *fun* ⟨*Bezeichner*⟩ ⟨*Argumentmuster*⟩ : ⟨*Typ*⟩ = ⟨*Ausdruck*⟩

⟨*Typ*⟩ ::=
⟨*atomarer Typ*⟩
| ⟨*Prozedurtyp*⟩
| ⟨*Tupeltyp*⟩
| (⟨*Typ*⟩)

…

Syntaktische Gleichungen für Ausdrücke

⟨*Ausdruck*⟩ ::=
⟨*atomarer Ausdruck*⟩
| ⟨*Anwendung*⟩
| ⟨*Konditional*⟩
| ⟨*Tupelausdruck*⟩
| ⟨*Let-Ausdruck*⟩
| (⟨*Ausdruck*⟩)

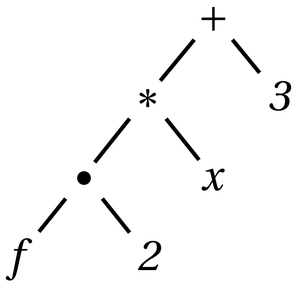
⟨*atomarer Ausdruck*⟩ ::=
⟨*Konstante*⟩
| ⟨*Bezeichner*⟩

⟨*Anwendung*⟩ ::=
⟨*Operatoranwendung*⟩
| ⟨*Prozeduranwendung*⟩
| ⟨*Projektion*⟩

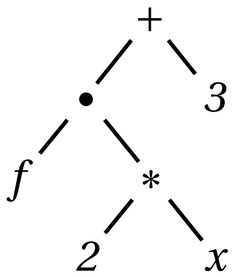
...

Klammern

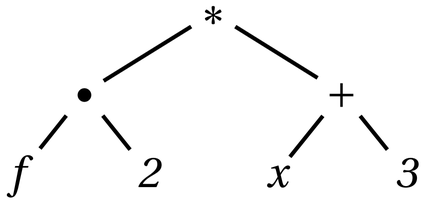
- Alle Phrasen können, mit Hilfe von Klammern,
  eindeutig durch **Wortdarstellungen** beschrieben werden:



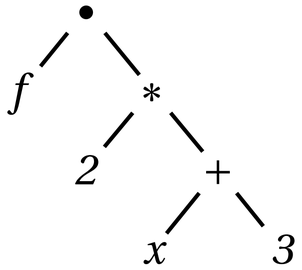
$f\ 2 * x + 3$



$f(2 * x) + 3$



$(f\ 2) * (x + 3)$



$f(2 * (x + 3))$

Klammersparregeln

$3 * 4 + 5$	$\rightsquigarrow$	$(3 * 4) + 5$	Punkt vor Strich
$2 + 3 + 4$	$\rightsquigarrow$	$(2 + 3) + 4$	Operatoranwendung klammert links
$2 - 3 + 4$	$\rightsquigarrow$	$(2 - 3) + 4$	
$f\ 3 + 4$	$\rightsquigarrow$	$(f\ 3) + 4$	Prozeduranwendung vor Operatoranwendung
$f\ g\ 3$	$\rightsquigarrow$	$(f\ g)\ 3$	Prozeduranwendung klammert links

$int * int \rightarrow int * int \quad \rightsquigarrow \quad (int * int) \rightarrow (int * int)$ Stern vor Pfeil

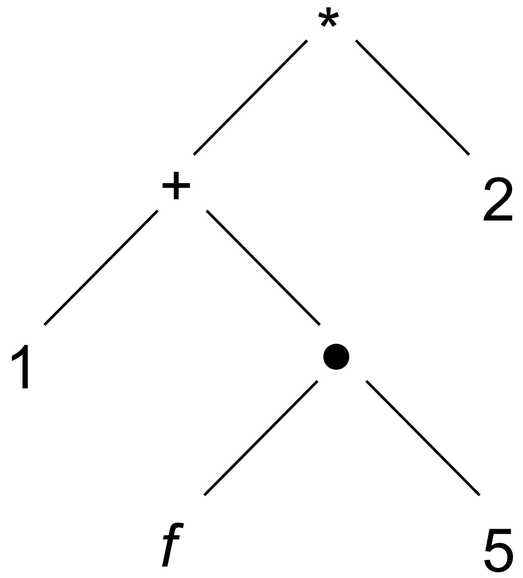
- überflüssige Klammern:

```
2*f~2+3      2*(f~2)+3      (2*(f~2))+3      ((2*(f~2))+3)
```

Frage

Geben Sie eine Zeichendarstellung für die durch die Baumdarstellung beschriebene Phrase an.

- (1 + f 5) * 2
- 1 + f 5 * 2
- 1 + (f 5) * 2
- 1 + (f 5 * 2)



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.

Frage

Welche der folgenden Deklarationen sind geschlossen?

- fun f(x:int) :int = f(x+y)
- val x = f(5)
- val x = let val y = 2 in y+1 end
- val x = let val x = 2 in x+1 end

Typregeln

$$\frac{e_1 : t_1 \qquad o : t_1 * t_2 \to t \qquad e_2 : t_2}{e_1\ o\ e_2 \ : \ t}$$

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 $t_1 * t_2 \to t$ 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.

www.prog1.saarland