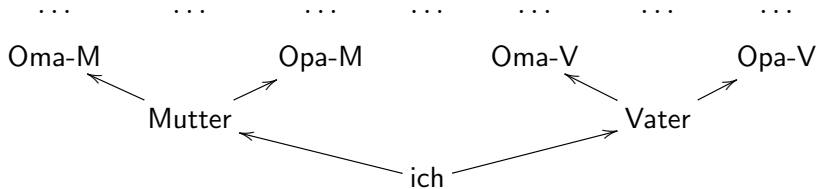


Grundlagen der Programmierung 2

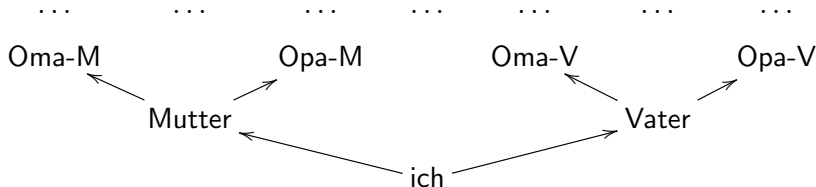
Haskell: Auswertung

Prof. Dr. Manfred Schmidt-Schauß

Sommersemester 2018

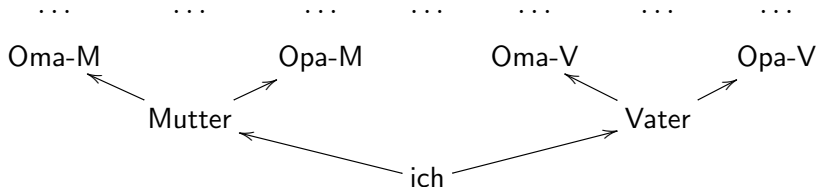


Aufgabe: Wieviele Vorfahren hat ein Mensch nach n Generationen?



Aufgabe: Wieviele Vorfahren hat ein Mensch nach n Generationen?

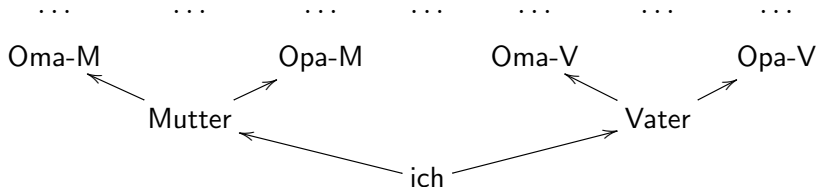
```
anzahlInGeneration n = if n == 0 || n == 1 then 2  
                        else 2 * anzahlInGeneration (n-1)
```



Aufgabe: Wieviele Vorfahren hat ein Mensch nach n Generationen?

```
anzahlInGeneration n = if n == 0 || n == 1 then 2  
                        else 2 * anzahlInGeneration (n-1)
```

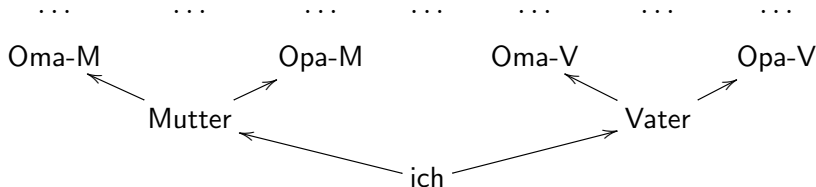
```
anzahlVorfahren n =  
  if n == 0 then 0  
  else anzahlInGeneration (n) + anzahlVorfahren (n-1)
```



Aufgabe: Wieviele Vorfahren hat ein Mensch nach n Generationen?

```
anzahlInGeneration n = if n == 0 || n == 1 then 2  
                        else 2 * anzahlInGeneration (n-1)
```

```
anzahlVorfahren n =  
  if n == 0 then 0  
  else anzahlInGeneration (n) + anzahlVorfahren (n-1)
```



Aufgabe: Wieviele Vorfahren hat ein Mensch nach n Generationen?

```
anzahlInGeneration n = if n == 0 || n == 1 then 2  
                        else 2 * anzahlInGeneration (n-1)
```

```
anzahlVorfahren n =  
  if n == 0 then 0  
  else anzahlInGeneration (n) + anzahlVorfahren (n-1)
```

Jetzt die Definition dazu:

f, g, f_i seien Haskell-definierte Funktionen.

f **referenziert** g **direkt**, **wenn** g im Rumpf von f vorkommt.

f **referenziert** g **wenn**
 f referenziert direkt g ,
 oder es gibt Funktionen f_1, \dots, f_n ,
 so dass gilt: f referenziert direkt f_1 ,
 f_1 referenziert direkt f_2, \dots ,
 f_n referenziert direkt g .
 f ist **direkt rekursiv**, **wenn** f sich selbst direkt referenziert.
 f ist **rekursiv**, **wenn** f sich selbst referenziert.

Verschränkte Rekursion: wenn f die Funktion g referenziert
 und g die Funktion f .
 (auch für allgemeinere Fälle)

```
quadrat x          = x*x  
quadratsumme x y = (quadrat x) + (quadrat y)
```

quadratsumme **referenziert direkt** die Funktionen `quadrat` und
die (eingebaute) Funktion `+`
quadratsumme **referenziert** die Funktionen `{quadrat, *, +}`

Die Funktion `quadratsumme` ist somit **nicht rekursiv**

$$\begin{aligned} 0! &:= 1 \\ n! &:= n * (n - 1)! && \text{wenn } n > 0 \end{aligned}$$

Anwendung: $n!$ ist die Anzahl aller Permutationen der Elemente einer n -elementigen Menge.

Beispiel:

Menge mit 3 Elementen $\{A, B, C\}$:

6 Permutationen: $ABC, ACB, BAC, BCA, CAB, CBA$

$$3! = 3 * 2! = 3 * 2 * 1! = 3 * 2 * 1 * 0! = 3 * 2 * 1 * 1 = 6$$

rekursive Definition:

```
fakultaet:: Integer -> Integer
fakultaet x  = if x <= 0 then 1
               else x*(fakultaet (x-1))
```

Die Funktion fakultaet ist **rekursiv**, da sie im Rumpf der Definition sich selbst referenziert

zwei Fälle sind beim Entwurf rekursiver Funktionen zu beachten

der **Basisfall**: keine weitere Rekursion

der **Rekursionsfall**: neuer rekursiver Aufruf

Terminierung der Aufrufe für alle Eingabefälle sollte man prüfen.
... Natürlich auch **Korrektheit**.


Am **Beispiel** fakultaet:

der **Basisfall**: Ergebnis: 0 wenn das Argument $x \leq 1$ ist.


der **Rekursionsfall**: Ergebnis: $x * (\text{fakultaet } (x-1))$, wenn $x > 1$ ist.

Terminierung:

- Argumente werden mit jedem rekursiven Aufruf kleiner
fakultaet x ruft fakultaet $(x-1)$ auf für $x \geq 1$.
- Der Basisfall hat das kleinste Argument

```
*Main> fakultaet 3 
```

```
6
```

```
*Main> fakultaet 40 
```

```
815915283247897734345611269596115894272000000000
```

```
fakultaet_nt:: Integer -> Integer
fakultaet_nt x  = if x == 0 then 1
                  else x*(fakultaet_nt (x-1))
```

Diese Funktion **terminiert nicht** bei negativen Eingaben:
fakultaet_nt (-5) ruft fakultaet_nt (-6) auf usw.

```
ggt :: Integer -> Integer -> Integer
ggt a b =    if    b == 0
              then a
              else ggt b (rem a b)
```

Annahme: $a, b \geq 0$

Basisfall: ?

Rekursionsfall: ?

Terminierung: ?

```
ggt :: Integer -> Integer -> Integer
ggt a b =    if    b == 0
              then a
              else ggt b (rem a b)
```

Annahme: $a, b \geq 0$

Basisfall: $b = 0$

Rekursionsfall: ?

Terminierung: ?

```
ggt :: Integer -> Integer -> Integer
ggt a b =    if    b == 0
              then a
              else ggt b (rem a b)
```

Annahme: $a, b \geq 0$

Basisfall: $b = 0$

Rekursionsfall: $b > 0$

Terminierung: ?


```
ggt :: Integer -> Integer -> Integer
ggt a b =    if    b == 0
              then a
              else ggt b (rem a b)
```

Annahme: $a, b \geq 0$

Basisfall: $b = 0$

Rekursionsfall: $b > 0$

Terminierung: $b > (a \text{ 'rem' } b) \geq 0;$
also ...

Auswertung von Haskell-Programmen

operationale Semantik von Haskell:

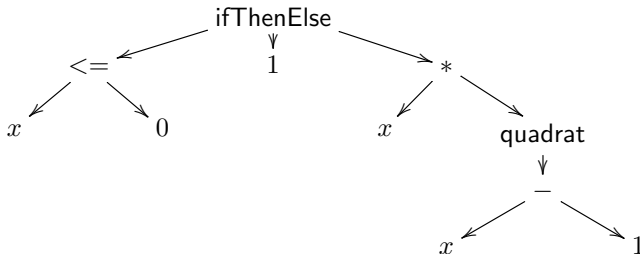
Transformationen eines Ausdrucks
(bzw. des main-Ausdrucks des Programms)

bis ein einfacher Ausdruck erreicht ist.

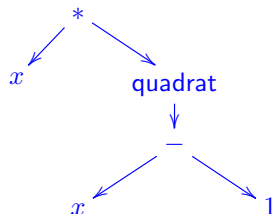
⇒ **Transformations-Semantik**

Ausdrücke sind implizit durch Syntaxbäume eindeutig dargestellt.

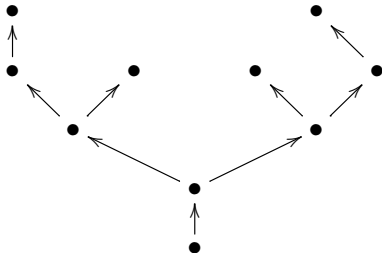
if x <= 0 then 1 else x*(quadrat (x-1))



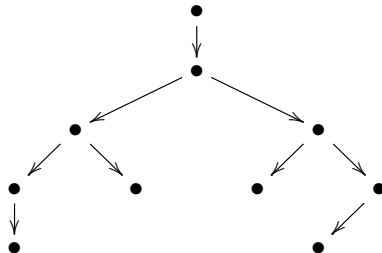
x*(quadrat (x-1))



Baum



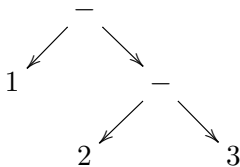
Baum (auf dem Kopf stehend)



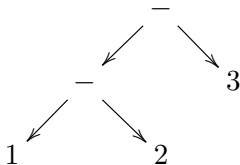


Zwei Syntaxbäume zum Ausdruck

1-2-3:



$(1 - (2 - 3))$



$((1 - 2) - 3)$

Nur einer passt zum Ausdruck 1-2-3 in Haskell.

Eindeutige Festlegung der Ausführung durch eine Strategie:

zwei Möglichkeiten (u.a.):

- **normale Reihenfolge der Auswertung**,
(Haskell benutzt eine verbesserte Variante)
- **applikative Reihenfolge der Auswertung**,
(z.B. in Python, ML, OCaml, Lisp)

Prinzip der **Berechnung** in Haskell:

Auswertung = Folge von Transformationen eines Ausdrucks

- in **normaler Reihenfolge (Normalordnung)**
- unter Benutzung der Funktions-Definitionen bis ein Basiswert erreicht ist

Fehler-Möglichkeiten für Transformationsfolgen:

- **Transformation bleibt stecken, aber kein Basiswert wird erreicht**
(wird verhindert durch Typisierung)
- **Transformationsfolge terminiert nicht**

$f\ x_1 \dots x_n = R_f$

sei die Haskell-Definition von f

Auswertung:

$$(f\ t_1 \dots t_n) \rightarrow (R_f[t_1/x_1, \dots t_n/x_n])$$

paralleles (unabhängiges) Ersetzen
von x_i durch die Argumente t_i für $i = 1, \dots, n$

$f\ x_1 \dots x_n = R_f$

sei die Haskell-Definition von f

Auswertung:

$(f\ t_1 \dots t_n) \rightarrow (R_f[t_1/x_1, \dots t_n/x_n])$

paralleles (unabhängiges) Ersetzen
von x_i durch die Argumente t_i für $i = 1, \dots, n$

Beispiel

`quadrat x = x*x`

R_{quadrat} ist hier: `x*x`

`quadrat 11`

$f\ x_1 \dots x_n = R_f$ sei die Haskell-Definition von f

Auswertung:

$$(f\ t_1 \dots t_n) \rightarrow (R_f[t_1/x_1, \dots t_n/x_n])$$

paralleles (unabhängiges) Ersetzen
von x_i durch die Argumente t_i für $i = 1, \dots, n$

Beispiel

`quadrat x = x*x` R_{quadrat} ist hier: `x*x`

`quadrat 11` $\rightarrow x*x[11/x] = 11*11$

Zahl + Zahl Ergebnis durch „Ausrechnen“
 mittels vorprogrammierter Operationen
Zahl * Zahl genauso
...

Beachte: Wenn *s op t* arithmetisch ausgewertet werden soll,
dann müssen zuerst die Ausdrücke *s* und *t*
zu Zahlen ausgewertet werden!

Programm:

```
main =  quadrat 5  
quadrat x = x*x
```

Auswertung als Folge von Transformationen:

main

Programm:

```
main = quadrat 5  
quadrat x = x*x
```

Auswertung als Folge von Transformationen:

main

→ (Definitions-Einsetzung)

quadrat 5

Programm:

```
main = quadrat 5  
quadrat x = x*x
```

Auswertung als Folge von Transformationen:

main

→ (Definitions-Einsetzung)

quadrat 5

→ (Definitions-Einsetzung)

5 * 5

Programm:

```
main = quadrat 5  
quadrat x = x*x
```

Auswertung als Folge von Transformationen:

main

→ (Definitions-Einsetzung)

quadrat 5

→ (Definitions-Einsetzung)

5 * 5

→ (arithmetische Auswertung)

25

UND

$v \ \&\& \ w$

ODER

$v \ || \ w$

NICHT

$\text{not } w$

Skönnte man so definieren:

```
x && y = if x then y      else False
x || y = if x then True  else y
not x  = if x then False else True
```

Fallunterscheidung (if-Reduktion)

Definition:

$$\begin{aligned}(\text{if True then } e_1 \text{ else } e_2) &\rightarrow e_1 \\(\text{if False then } e_1 \text{ else } e_2) &\rightarrow e_2\end{aligned}$$

Beachte

(if b then e_1 else e_2)-Auswertung

erfordert automatisch, dass zuerst b ausgewertet werden muss.

Wir nennen eine Transformation auch *Reduktion* und eine Folge von Programmtransformationen auch *Reduktionsfolge* (oder *Auswertung*).

Beachte: Reduktionen / Transformationen sind zunächst überall im Ausdruck erlaubt.

Erst eine *Auswertungs-Strategie* macht die Auswertung eindeutig.

Motivation für diese Transformationssemantik:

- **eindeutige Festlegung der (Haskell-)Auswertung**
- **Umgang mit nichtterminierenden Argumenten**
- **Unendliche Listen**
- **diese kommt ohne Übersetzung/Compilation aus**

Motivation für diese Transformationssemantik:

- **eindeutige Festlegung der (Haskell-)Auswertung**
- **Umgang mit nichtterminierenden Argumenten**
- **Unendliche Listen**
- **diese kommt ohne Übersetzung/Compilation aus**

Motivation für Normalordnung (s.u.):

- **korrekte Programmtransformationen**
- **einfache Programm-Logik**
- **Hohe Modularität wird ermöglicht**
- **Implizite Parallelität wird unterstützt**

```
x && y = if x then y      else False
x || y = if x then True   else y
bot     = bot
```

Beispiel-Auswertungen dazu

True && True → if True then True else False → True

True && False → if True then False else False → False

True || True → if True then True else True → True

True || False → if True then True else False → True

True && bot → if True then bot else False → bot → bot ...

True || bot → if True then True else bot → True

bot terminiert nicht bei Auswertung!

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)`

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)`

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5)$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5)$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5)$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5)$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{array}{l} \text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) \end{array}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)` \xrightarrow{D} `(4 + 5) * (4 + 5)` \xrightarrow{A} `9 * (4 + 5)` \xrightarrow{A} `9 * 9` \xrightarrow{A} 81
`quadrat(4 + 5)`

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \rightarrow 81 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)` \xrightarrow{D} `(4 + 5) * (4 + 5)` \xrightarrow{A} `9 * (4 + 5)` \xrightarrow{A} `9 * 9` \xrightarrow{A} 81
`quadrat(4 + 5)` \rightarrow `(4 + 5) * (4 + 5)` \rightarrow `(4 + 5) * 9` \rightarrow `9 * 9` \rightarrow 81
`quadrat(4 + 5)`

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)` \xrightarrow{D} `(4 + 5) * (4 + 5)` \xrightarrow{A} `9 * (4 + 5)` \xrightarrow{A} `9 * 9` \xrightarrow{A} 81
`quadrat(4 + 5)` \rightarrow `(4 + 5) * (4 + 5)` \rightarrow `(4 + 5) * 9` \rightarrow `9 * 9` \rightarrow 81
`quadrat(4 + 5)`

3 verschiedene Auswertungen für `quadrat (4+5)` :

$\text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81$
 $\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \rightarrow 81$
 $\text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9$

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)` \xrightarrow{D} `(4 + 5) * (4 + 5)` \xrightarrow{A} `9 * (4 + 5)` \xrightarrow{A} `9 * 9` \xrightarrow{A} 81
`quadrat(4 + 5)` \rightarrow `(4 + 5) * (4 + 5)` \rightarrow `(4 + 5) * 9` \rightarrow `9 * 9` \rightarrow 81
`quadrat(4 + 5)` \rightarrow **`quadrat 9`**

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{aligned} \text{quadrat}(4 + 5) &\xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) &\rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \rightarrow 81 \\ \text{quadrat}(4 + 5) &\rightarrow \text{quadrat } 9 \rightarrow 9 * 9 \end{aligned}$$

3 verschiedene Auswertungen für `quadrat (4+5)` :

`quadrat(4 + 5)` \xrightarrow{D} `(4 + 5) * (4 + 5)` \xrightarrow{A} `9 * (4 + 5)` \xrightarrow{A} `9 * 9` \xrightarrow{A} 81
`quadrat(4 + 5)` \rightarrow `(4 + 5) * (4 + 5)` \rightarrow `(4 + 5) * 9` \rightarrow `9 * 9` \rightarrow 81
`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow **9 * 9**

3 verschiedene Auswertungen für `quadrat (4+5)` :

<code>quadrat(4 + 5)</code>	\xrightarrow{D}	<code>(4 + 5) * (4 + 5)</code>	\xrightarrow{A}	<code>9 * (4 + 5)</code>	\xrightarrow{A}	<code>9 * 9</code>	\xrightarrow{A}	<code>81</code>
<code>quadrat(4 + 5)</code>	\rightarrow	<code>(4 + 5) * (4 + 5)</code>	\rightarrow	<code>(4 + 5) * 9</code>	\rightarrow	<code>9 * 9</code>	\rightarrow	<code>81</code>
<code>quadrat(4 + 5)</code>	\rightarrow	<code>quadrat 9</code>	\rightarrow	<code>9 * 9</code>	\rightarrow	<code>81</code>		

3 verschiedene Auswertungen für `quadrat (4+5)` :

$$\begin{array}{l} \text{quadrat}(4 + 5) \xrightarrow{D} (4 + 5) * (4 + 5) \xrightarrow{A} 9 * (4 + 5) \xrightarrow{A} 9 * 9 \xrightarrow{A} 81 \\ \text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow (4 + 5) * 9 \rightarrow 9 * 9 \rightarrow 81 \\ \text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9 \rightarrow 9 * 9 \rightarrow 81 \end{array}$$

Beobachtungen:

- Das Ergebnis ist gleich
- Die Anzahl der Reduktionen kann verschieden sein

Wichtige *Reduktionsstrategien*

Applikative Reihenfolge:

Argumentauswertung vor Definitionseinsetzung

Normale Reihenfolge:

Definitionseinsetzung vor Argumentauswertung

Verzögerte Reihenfolge:

Normale Reihenfolge mit Sharing

(informell)

Satz

Sei t ein Ausdruck in Haskell.

Wenn

$$t \xrightarrow{*,appl.R.} a,$$

$$t \xrightarrow{*,norm.R.} b$$

$$t \xrightarrow{*,verz.R.} c$$

und a, b, c sind Basiswerte (d.h. Integer, Zahlen, ..., Boolesche Werte).

Dann gilt $a = b = c$.

$t \xrightarrow{*,\dots} a$ bedeutet: in 0,1,2, 3,.. Reduktionsschritten.

- Argumentauswertung vor Definitionseinsetzung
wird in den meisten Programmiersprachen benutzt
z.B. in Python, C, ML,
- wird **nicht** in Haskell verwendet.
- Diskussion der Vor- / Nachteile später ...

Werte den Ausdruck t **applikativ** aus! (dadurch wird die Auswertung eindeutig)

- t ist Basiswert. fertig.

- $t \equiv s_1 t_1$,

Wenn s_1 kein Funktionsname und keine Anwendung,
dann werte s_1 applikativ aus (**rekursiv**)

- $t \equiv f t_1 \dots t_n$.

Wenn $ar(f) \leq n$, dann (**rekursiv**) applikativ auswerten:

t_i , $1 \leq i \leq ar(f)$ von links nach rechts.

Wenn $ar(f) \leq n$ und alle t_i Basiswerte, dann Definitionseinsetzung

Wenn $n < ar(f)$, dann fertig: keine Reduktion.

- $t \equiv \text{if } b \text{ then } e_1 \text{ else } e_2$.

Wenn b Basiswert, dann if-Reduktion

Wenn b kein Basiswert, dann werte b applikativ aus (**rekursiv**)

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

quadrat (`quadrat ((quadrat 2) + (quadrat 3))`)

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

Wo im Ausdruck **applikativ** auswerten?

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

`quadrat (quadrat ((quadrat 2) + (quadrat 3)))`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

quadrat(4 + 5)

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` → `quadrat 9`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

$\text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9 \rightarrow 9 * 9$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow **`9 * 9`**


```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

$\text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9 \rightarrow 9 * 9 \rightarrow 81$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

$\text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9 \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

$\text{quadrat}(4 + 5) \rightarrow \text{quadrat } 9 \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow \text{quadrat}(\text{quadrat } 5)$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)` \rightarrow
`quadrat 25`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)` \rightarrow
`quadrat 25`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)` \rightarrow
`quadrat 25` \rightarrow `25 * 25`


```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)` \rightarrow
`quadrat 25` \rightarrow 25 * 25

```
quadrat x = x*x
```

2 Beispiel-Auswertungen (applikative Reihenfolge)

`quadrat(4 + 5)` \rightarrow `quadrat 9` \rightarrow `9 * 9` \rightarrow 81

`quadrat(quadrat(2 + 3))` \rightarrow `quadrat(quadrat 5)` \rightarrow
`quadrat 25` \rightarrow `25 * 25` \rightarrow 625

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

fakultaet 4

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

fakultaet 4

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
```



```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
4*(fakultaet 3)
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
4*(fakultaet 3)
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
4*(fakultaet 3)
4*(if 3 <= 1 then 1 else 3*(fakultaet (3-1)))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

applikativ

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet (4-1))
if False then 1 else 4*(fakultaet (4-1))
4*(fakultaet (4-1))
4*(fakultaet 3)
4*(if 3 <= 1 then 1 else 3*(fakultaet (3-1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
```



```
4*(if False then 1 else 3*(fakultaet (3-1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
```

```
4*(3*(fakultaet (3-1)))
```

```
4*(3* (fakultaet 2))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
```

```
4*(3*(fakultaet (3-1)))
```

```
4*(3* (fakultaet 2))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
```



```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet (2-1))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet (2-1))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet (2-1))))  
4*(3*(2*(fakultaet 1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))  
4*(3*(fakultaet (3-1)))  
4*(3* (fakultaet 2))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet (2-1))))  
4*(3*(2*(fakultaet 1)))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1)))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1)))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
```



```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
(4*6)
```

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
(4*6)
```



```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1)))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
(4*6)
```

24

```
4*(if False then 1 else 3*(fakultaet (3-1)))
4*(3*(fakultaet (3-1)))
4*(3* (fakultaet 2))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet (2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet (2-1))))
4*(3*(2*(fakultaet 1))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
(4*6)
```

24

18 Auswertungsschritte

```
main = const 5 (fakultaet 4)
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
const x y = x
```

$\text{main} \xrightarrow{1} \text{const } 5 \text{ (fakultaet } 4) \xrightarrow{18} \text{const } 5 \text{ 24} \xrightarrow{1} 5$

Anzahl der Reduktionen: 20

- Definitionseinsetzung vor Argumentauswertung
- wird in Haskell benutzt

werte t in **normaler** Reihenfolge aus. Fälle:

- t ist Basiswert. fertig.
- $t \equiv s_1 t_1$,
Wenn s_1 kein Funktionsname und keine Anwendung.
Dann normale R. auf s_1
- $t \equiv f t_1 \dots t_n$ und f keine eingebaute Funktion,
Wenn $ar(f) \leq n$, dann Definitionseinsetzung auf $f t_1 \dots t_{ar(f)}$.
Wenn $ar(f) > n$: keine Reduktion.
- $t \equiv f t_1 \dots t_n$ und f ist eingebaute Funktion
Wenn $ar(f) \leq n$ und Argumente von f keine Basiswerte,
dann normale R. auf $ar(f)$ Argumente von links nach rechts.
Wenn $ar(f) \leq n$, und $ar(f)$ Argumente von f sind Basiswerte,
dann eingebaute Funktion aufrufen.
Wenn $ar(f) > n$: keine Reduktion.
- $t \equiv \text{if } b \text{ then } e_1 \text{ else } e_2$.
Wenn b Basiswert, dann if-Reduktion
Wenn b kein Basiswert, dann normale R. auf b

Wo im Ausdruck auswerten (entsprechend normaler Reihenfolge) ?

$(\text{quadrat } (2 + 3)) * (\text{quadrat } (2 + 3))$

Wo im Ausdruck auswerten (entsprechend normaler Reihenfolge) ?

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

Wo im Ausdruck auswerten (entsprechend normaler Reihenfolge) ?

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

Wo im Ausdruck auswerten (entsprechend normaler Reihenfolge) ?

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

Wo im Ausdruck auswerten (entsprechend normaler Reihenfolge) ?

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

`(quadrat (2 + 3)) * (quadrat (2 + 3))`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

`quadrat(4 + 5)`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5)$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

`quadrat(4 + 5) → (4 + 5) * (4 + 5)`

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5)$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5)$


```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3))$


```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * ((2 + 3) * (2 + 3))$


```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * ((2 + 3) * (2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * ((2 + 3) * (2 + 3)) \rightarrow$

$25 * (5 * (2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * ((2 + 3) * (2 + 3)) \rightarrow$
 $25 * (5 * (2 + 3))$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * ((2 + 3) * (2 + 3)) \rightarrow$
 $25 * (5 * (2 + 3)) \rightarrow$
 $25 * (5 * 5)$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * ((2 + 3) * (2 + 3)) \rightarrow$
 $25 * (5 * (2 + 3)) \rightarrow$
 $25 * (5 * 5)$

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

```
quadrat (quadrat(2 + 3))           →  
(quadrat(2 + 3)) * (quadrat(2 + 3)) →  
((2 + 3) * (2 + 3)) * (quadrat(2 + 3)) →  
(5 * (2 + 3)) * (quadrat(2 + 3))   →  
(5 * 5) * (quadrat(2 + 3))         →  
25 * (quadrat(2 + 3))              →  
25 * ((2 + 3) * (2 + 3))           →  
25 * (5 * (2 + 3))                 →  
25 * (5 * 5)                       →  
25 * 25
```

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$

$(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$

$(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * (\text{quadrat}(2 + 3)) \rightarrow$

$25 * ((2 + 3) * (2 + 3)) \rightarrow$

$25 * (5 * (2 + 3)) \rightarrow$

$25 * (5 * 5) \rightarrow$

25 * 25

```
quadrat x = x*x
```

2 Beispiel-Auswertungen

$\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$

$\text{quadrat}(\text{quadrat}(2 + 3)) \rightarrow$
 $(\text{quadrat}(2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $((2 + 3) * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * (2 + 3)) * (\text{quadrat}(2 + 3)) \rightarrow$
 $(5 * 5) * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * (\text{quadrat}(2 + 3)) \rightarrow$
 $25 * ((2 + 3) * (2 + 3)) \rightarrow$
 $25 * (5 * (2 + 3)) \rightarrow$
 $25 * (5 * 5) \rightarrow$
 $25 * 25 \rightarrow$

625


```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

fakultaet 4

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

fakultaet 4

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet(4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

fakultaet 4

if 4 <= 1 then 1 else 4*(fakultaet(4-1))

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet(4-1))
if False then 1 else 4*(fakultaet(4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet(4-1))
if False then 1 else 4*(fakultaet(4-1))
```

```
main = fakultaet 4
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```

Auswertung (in normaler Reihenfolge:)

```
fakultaet 4
if 4 <= 1 then 1 else 4*(fakultaet(4-1))
if False then 1 else 4*(fakultaet(4-1))
4*(fakultaet(4-1))
```


`4*(fakultaet(4-1))`

$4 * (\text{fakultaet}(4-1))$

```
4*(fakultaet(4-1)))
```

```
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))
```

```
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))
```

```
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))
```

```
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
```



```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(4-1)*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if(3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet((((4-1)-1)-1)))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet((((4-1)-1)-1)))))
```



```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if(3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet((((4-1)-1)-1)))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet((((4-1)-1)-1)))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))  
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*((4-1)*(fakultaet((4-1)-1)))  
4*(3*(fakultaet((4-1)-1)))  
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))  
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))  
4*(3*((4-1)-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
```



```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(fakultaet(((4-1)-1)-1))))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))
```

```
4*(fakultaet(4-1)))
4*(if (4-1)<= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if 3 <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
4*(if False then 1 else (4-1)*(fakultaet((4-1)-1)))
4*((4-1)*(fakultaet((4-1)-1)))
4*(3*(fakultaet((4-1)-1)))
4*(3*(if ((4-1)-1) <= 1 then 1 else (((4-1)-1)*(fakultaet(((4-1)-1)-1)))))
4*(3*(if (3-1)<= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if 2 <= 1 then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(if False then 1 else ((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(((4-1)-1)*(fakultaet(((4-1)-1)-1))))
4*(3*((3-1)*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(fakultaet(((4-1)-1)-1))))
4*(3*(2*(if (((4-1)-1) <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))
```

$4 * (3 * (2 * (\text{if } ((4 - 1) - 1) \leq 1 \text{ then } 1 \dots)))$

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))
```



```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))
```



```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)  
4*6
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)  
4*6
```

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)  
4*6
```

24

```
4*(3*(2*(if (((4-1)-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if ((3-1)-1) <= 1 then 1 ...)))  
4*(3*(2*(if 2-1 <= 1 then 1 ...)))  
4*(3*(2*(if 1 <= 1 then 1 ...)))  
4*(3*(2*(if True then 1 ...)))  
4*(3*(2*1))  
4*(3*2)  
4*6
```

24

Das sind 24 Auswertungsschritte

```
main = const 5 (fakultaet 4)
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
const x y = x
```

`main` $\xrightarrow{1}$ `const 5 (fakultaet 4)` $\xrightarrow{1}$ 5

Anzahl der Reduktionen: 2

(20 bei applikativer Reihenfolge)

```
quadrat x = x*x
```

3 Auswertungen für `quadrat (4+5)` :

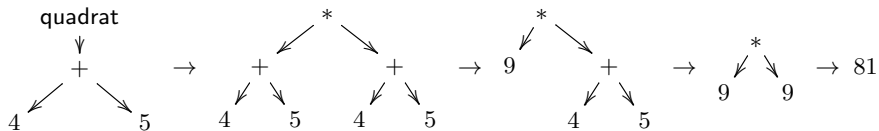
- ① `quadrat(4 + 5) → (4 + 5)*(4+5) → 9*(4 + 5) → 9 * 9 → 81`
normale Reihenfolge der Auswertung
- ② `quadrat(4 + 5) → (4+5)*(4 + 5) → (4 + 5)*9 → 9 * 9 → 81`
Irgendeine Auswertung
- ③ `quadrat(4 + 5) → (quadrat 9) → 9 * 9 → 81`
applikative Reihenfolge der Auswertung

Definition *verzögerte Reihenfolge der Auswertung* (lazy reduction):

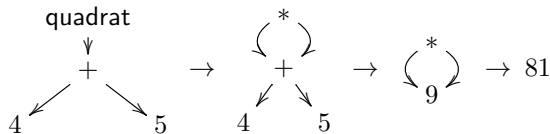
- wie normale Reihenfolge
- aber: **gerichteter Graph** statt **Syntax-Baum**
- Vermeidung von unnötiger Doppelauswertung durch gemeinsame Unterausdrücke (Sharing)
- Die gemeinsamen (d.h. shared) Unterausdrücke sind durch die Funktionsrümpfe festgelegt.

- 4 Reduktionen: (normale Reihenfolge)
 $\text{quadrat}(4 + 5) \rightarrow (4 + 5) * (4 + 5) \rightarrow 9 * (4 + 5) \rightarrow 9 * 9 \rightarrow 81$
- 3 Reduktionen (verzögerte Reihenfolge)
 $\text{quadrat}(4 + 5) \rightarrow (4 + 5)^{(1)} * (4 + 5)^{(1)} \rightarrow 9 * 9 \rightarrow 81$

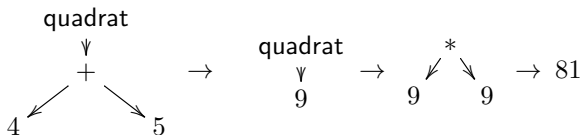
Normale Reihenfolge:



Verzögerte Reihenfolge:



Applikative Reihenfolge:



```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

fakultaet 4

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

fakultaet 4

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

fakultaet 4

if 4 <= 1 then 1 else 4*(fakultaet(4-1))

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

fakultaet 4

if 4 <= 1 then 1 else 4*(fakultaet(4-1))


```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

fakultaet 4

if 4 <= 1 then 1 else 4*(fakultaet(4-1))

if False then 1 else 4*(fakultaet(4-1))

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))  
4*(if (4-1) <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))  
4*(if (4-1) <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))  
4*(if (4-1) <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3 <= 1 then 1 else 3 * (fakultaet(3 -1)))
```

```
fakultaet x = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

Rot: die Stelle, die reduziert wird

Grün: die Stelle, die identisch mit der roten ist

```
fakultaet 4  
if 4 <= 1 then 1 else 4*(fakultaet(4-1))  
if False then 1 else 4*(fakultaet(4-1))  
4*(fakultaet(4-1))  
4*(if (4-1) <= 1 then 1 else (4-1)*(fakultaet((4-1)-1)))  
4*(if 3  <= 1  then 1 else 3  *(fakultaet(3  -1)))
```



```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1) <= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
```



```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet(2-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet(2-1))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4* (if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet(2-1))))  
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))  
4* (if False then 1 else 3*(fakultaet(3-1)))  
4*(3*(fakultaet(3-1)))  
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))  
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))  
4*(3*(if False then 1 else 2*(fakultaet(2-1))))  
4*(3*(2*(fakultaet(2-1))))  
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1))))  
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1))))))
```



```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4* (if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4* (if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4* (if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4* (if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
4*6
```



```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
4*6
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
4*6
```

```
4*(if 3 <= 1 then 1 else 3*(fakultaet(3-1)))
4*(if False then 1 else 3*(fakultaet(3-1)))
4*(3*(fakultaet(3-1)))
4*(3*(if (3-1)<= 1 then 1 else (3-1)*(fakultaet((3-1)-1))))
4*(3*(if 2 <= 1 then 1 else 2*(fakultaet(2-1))))
4*(3*(if False then 1 else 2*(fakultaet(2-1))))
4*(3*(2*(fakultaet(2-1))))
4*(3*(2*(if (2-1)<= 1 then 1 else (2-1)*(fakultaet ((2-1)-1)))))
4*(3*(2*(if 1 <= 1 then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*(if True then 1 else 1*(fakultaet(1-1)))))
4*(3*(2*1))
4*(3*2)
4*6
```

24

(18 Auswertungsschritte)

	verzögerte R.	applikative R.	normale R.
(fakultaet 4)	18	18	24
main	2	20	2

```
main = const 5 (fakultaet 4)
```

Es gilt: verzögerte Reduktion
hat **optimale Anzahl** von Reduktionsschritten. !

- Es gilt immer:

$$\# \text{ verzögerte R.} \leq \# \text{ normale R.}$$

$$\# \text{ verzögerte R.} \leq \# \text{ applikative R.}$$

- Im allgemeinen gilt:
 $\#$ applikative R. und $\#$ normale R. sind **unvergleichbar**
- Wenn alle Reduktionsschritte für das Ergebnis benötigt werden:

$$\# \text{ verzögerte R.} \leq \# \text{ applikative R.} \leq \# \text{ normale R.}$$

In deterministischen Programmiersprachen (z.B. Haskell) mit verzögerter (oder normaler) Reihenfolge der Auswertung gilt:

Reduktionen zur Compile-Zeit
sind **korrekte Programmtransformationen**
d.h. die Semantik bleibt erhalten

Das ist i.a. **falsch** in Programmiersprachen, die die applikative Reihenfolge verwenden.

Auswertungsprozesse

Wir betrachten jetzt **Auswertungsprozesse**, die durch **eine einzige** rekursive Funktion erzeugt werden

Wir betrachten bei der Analyse von Auswertungsprozessen nur die **applikative Reihenfolge**

Beispiel: Auswertung der rekursiven Fakultätsfunktion

$$\begin{aligned} 0! &:= 1 \\ n! &:= n * (n - 1)! \quad \text{wenn } n > 1 \end{aligned}$$

```
fakultaet x = if x <= 1 then 1
              else x*(fakultaet (x-1))
```


bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

(fakultaet 6)

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
```

```
(6 * (fakultaet (6-1)))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
```

```
(6 * (fakultaet (6-1)))
```

```
(6 * (5 * (fakultaet (5-1))))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
```

```
(6 * (fakultaet (6-1)))
```

```
(6 * (5 * (fakultaet (5-1))))
```

```
(6 * (5 * (4 * (fakultaet (4-1)))))
```

```
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
```


bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
(6 * (5 * 24))
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
(6 * (5 * 24))
(6 * 120)
```

bei **applikativer Reihenfolge** der Auswertung
(Nicht jeder Zwischenzustand ist angegeben)

```
(fakultaet 6)
(6 * (fakultaet (6-1)))
(6 * (5 * (fakultaet (5-1))))
(6 * (5 * (4 * (fakultaet (4-1)))))
(6 * (5 * (4 * (3 * (fakultaet (3-1))))))
(6 * (5 * (4 * (3 * (2 * (fakultaet (2-1)))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
(6 * (5 * 24))
(6 * 120)
720
```

(fakultaet 6) Auswertungsprozess ist linear rekursiv

- Charakteristisch:
- nur eine rekursive Funktionsanwendung in jedem Ausdruck der Reduktionsfolge
 - Zwischenausdrücke sind nicht beschränkt.

Iteriere folgende Regel:

Produkt	\Rightarrow	Produkt * Zähler
Zähler	\Rightarrow	Zähler + 1

```
fakt_iter produkt zaehler max =  
  if zaehler > max  
  then produkt  
  else fakt_iter (zaehler*produkt) (zaehler + 1) max  
  
fakultaet_lin n = fakt_iter 1 1 n
```

Verwendung lokaler Funktionen mit `let ...in ...`

```
fakultaet_lin n =  
  let fakt_iter produkt zaehler max =  
    if zaehler > max  
    then produkt  
    else fakt_iter (zaehler*produkt)  
                  (zaehler + 1)  
                  max  
  in  fakt_iter 1 1 n
```


Eine **Endrekursion** ist eine lineare Rekursion.

Zusätzlich muss gelten:

- In jedem Rekursionsaufruf:
der rekursive Aufruf
berechnet direkt den Rückgabewert
ohne Nachverarbeitung nach der Rekursion

Z.B. `funktion a b c` ruft auf `funktion d e f`
`aber nicht` `g * (funktion d e f)`

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

(fakultaet_lin 5)

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
```

```
(fakt_iter 1 1 5)
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
```

```
(fakt_iter 1 1 5)
```

```
(fakt_iter (1*1) (1+1) 5)
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
(fakt_iter 1 1 5)
(fakt_iter (1*1) (1+1) 5)
(fakt_iter (2*(1*1)) (2+1) 5)
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
```

```
(fakt_iter 1 1 5)
```

```
(fakt_iter (1*1) (1+1) 5)
```

```
(fakt_iter (2*(1*1)) (2+1) 5)
```

```
(fakt_iter (3*(2*(1*1))) (3+1) 5)
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
(fakt_iter 1 1 5)
(fakt_iter (1*1) (1+1) 5)
(fakt_iter (2*(1*1)) (2+1) 5)
(fakt_iter (3*(2*(1*1))) (3+1) 5)
(fakt_iter (4*(3*(2*(1*1)))) (4+1) 5)
```


Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
(fakt_iter 1 1 5)
(fakt_iter (1*1) (1+1) 5)
(fakt_iter (2*(1*1)) (2+1) 5)
(fakt_iter (3*(2*(1*1))) (3+1) 5)
(fakt_iter (4*(3*(2*(1*1)))) (4+1) 5)
(fakt_iter (5*(4*(3*(2*(1*1))))) (5+1) 5)
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
(fakt_iter 1 1 5)
(fakt_iter (1*1) (1+1) 5)
(fakt_iter (2*(1*1)) (2+1) 5)
(fakt_iter (3*(2*(1*1))) (3+1) 5)
(fakt_iter (4*(3*(2*(1*1)))) (4+1) 5)
(fakt_iter (5*(4*(3*(2*(1*1))))) (5+1) 5)
(5*(4*(3*(2*(1*1)))))
```

Auswertung von (fakultaet_lin 5) bei verzögerter Reihenfolge der Auswertung

```
(fakultaet_lin 5)
(fakt_iter 1 1 5)
(fakt_iter (1*1) (1+1) 5)
(fakt_iter (2*(1*1)) (2+1) 5)
(fakt_iter (3*(2*(1*1))) (3+1) 5)
(fakt_iter (4*(3*(2*(1*1)))) (4+1) 5)
(fakt_iter (5*(4*(3*(2*(1*1))))) (5+1) 5)
(5*(4*(3*(2*(1*1)))))
120
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

Iterativer Auswertungsprozess bei applikativer Auswertung:

(fakultaet_lin 6)

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
```

```
(fakt_iter 1 1 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
```

```
(fakt_iter 1 1 6)
```

```
(fakt_iter 1 2 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
```

```
(fakt_iter 1 1 6)
```

```
(fakt_iter 1 2 6)
```

```
(fakt_iter 2 3 6)
```


Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
```

```
(fakt_iter 1 1 6)
```

```
(fakt_iter 1 2 6)
```

```
(fakt_iter 2 3 6)
```

```
(fakt_iter 6 4 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
(fakt_iter 1 1 6)
(fakt_iter 1 2 6)
(fakt_iter 2 3 6)
(fakt_iter 6 4 6)
(fakt_iter 24 5 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
(fakt_iter 1 1 6)
(fakt_iter 1 2 6)
(fakt_iter 2 3 6)
(fakt_iter 6 4 6)
(fakt_iter 24 5 6)
(fakt_iter 120 6 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
(fakt_iter 1 1 6)
(fakt_iter 1 2 6)
(fakt_iter 2 3 6)
(fakt_iter 6 4 6)
(fakt_iter 24 5 6)
(fakt_iter 120 6 6)
(fakt_iter 720 7 6)
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
(fakt_iter 1 1 6)
(fakt_iter 1 2 6)
(fakt_iter 2 3 6)
(fakt_iter 6 4 6)
(fakt_iter 24 5 6)
(fakt_iter 120 6 6)
(fakt_iter 720 7 6)
720
```

Iterativer Auswertungsprozess bei applikativer Auswertung:

```
(fakultaet_lin 6)
(fakt_iter 1 1 6)
(fakt_iter 1 2 6)
(fakt_iter 2 3 6)
(fakt_iter 6 4 6)
(fakt_iter 24 5 6)
(fakt_iter 120 6 6)
(fakt_iter 720 7 6)
720
```

Iterativer Prozess:

Charakteristisch: ist eine Endrekursion
 Argumente sind Basiswerte
 (bzw. Größe des Gesamtausdrucks bleibt beschränkt.)
 optimierte Rückgabe des Wertes

imperative Programmiersprachen	Endrekursion i.a. nicht optimiert. d.h. Wert wird durch alle Stufen der Rekursion zurückgegeben
Haskell	Endrekursion ist optimiert. am Ende wird der Wert unmittelbar zurückgegeben.

Deshalb braucht man in imperativen Programmiersprachen:

Iterationskonstrukte

`for ...do,` `while,` `repeat ...until.`

Diese entsprechen iterativen Auswertungsprozessen

Bei verzögerter Auswertung:

Eine rekursive Funktion f ist iterativ, wenn
 $f\ t_1 \dots t_n$ für Basiswerte t_i
bei applikativer Reihenfolge der Auswertung
einen iterativen Prozess ergibt.

Viele (nicht alle) linear rekursive Funktionen
kann man zu iterativen umprogrammieren.
Zum Beispiel: `fakultaet` zu `fakultaet_lin`

Beispiel Berechnung der Fibonacci-Zahlen

1, 1, 2, 3, 5, 8, 13, 21, ...

$$Fib(n) := \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{sonst} \end{cases}$$

```
fib n = if n <= 0 then 0
       else if n == 1 then 1
       else fib (n-1) + fib(n-2)
```

in applikativer Reihenfolge:

Der Auswertungs-Prozess ergibt folgende Zwischen-Ausdrücke:

fib 5

fib 4 + fib 3

(fib 3 + fib 2) + fib 3

((fib 2 + fib 1) + fib 2) + fib 3

(((fib 1 + fib 0) + fib 1) + fib 2) + fib 3

((1+0) + fib 1) + fib 2) + fib 3

((1 + fib 1) + fib 2) + fib 3

((1+1) + fib 2) + fib 3

(2 + fib 2) + fib 3

(2 + (fib 1 + fib 0)) + fib 3

.....

fib 5

fib 4 + fib 3

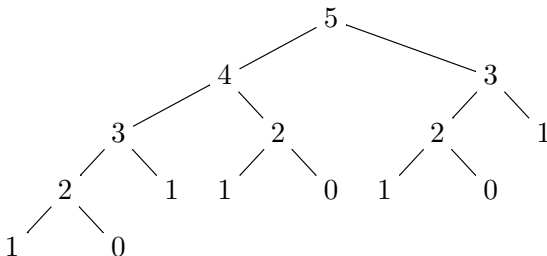
(fib 3 + fib 2) + fib 3

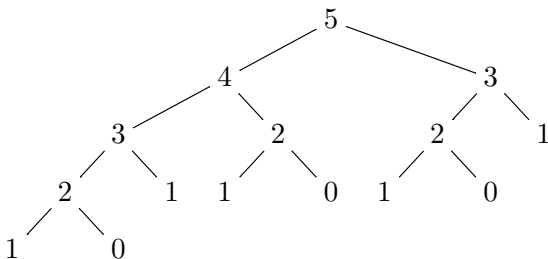
((fib 2 + fib 1) + fib 2) + fib 3

((fib 1 + fib 0) + fib 1) + fib 2) + fib 3

((1+0) + fib 1) + fib 2) + fib 3

((1 + fib 1) + fib 2) + fib 3





Das ist **Baumrekursion**

Charakteristisch: Ausdrücke in der Reduktionsfolge

- können unbegrenzt wachsen
- können mehrere rekursive Aufrufe enthalten
- Aber: nicht geschachtelt
(d.h. die Argumente eines rekursiven Aufrufs
enthalten keine rekursiven Aufrufe)

Ist der **allgemeine** Fall
wird normalerweise selten benötigt, da i.a. nicht effizient
berechenbar. **Beispiel:** Die **Ackermannfunktion**

```
----- Ackermanns Funktion -----  
ack 0 y                                = 1  
ack 1 0                                = 2  
ack x 0 | x >= 2                        = x+2  
ack x y | x > 0 && y > 0                = ack (ack (x-1) y) (y-1)
```

benutzt folgende **Programmier-Technik** in Haskell:


Mehrere Definitionsgleichungen **einer** Funktion

Reihenfolge: der Fallabarbeitung

von oben nach unten wird probiert, welche Definitionsgleichung
passt:

- 1) Argumente anpassen
- 2) Bedingung rechts vom | prüfen

```
----- Ackermanns Funktion "optimiert" ----  
ackopt 0 y = 1  
ackopt 1 0 = 2  
ackopt x 0 = x+2  
ackopt x 1 = 2*x  
ackopt x 2 = 2^x  
ackopt x y | x > 0 && y > 0 =  
            ackopt (ackopt (x-1) y) (y-1)
```

```
*Main> anzahlStellen (ackopt 5 3)   
19729
```

19729 ist die Anzahl der Dezimalstellen

des Ergebnisses $(\text{ackopt } 5 \ 3) = 2^{65536} = 2^{2^{2^2}}$

- sehr schnell wachsende Funktion
- man kann nachweisen: man braucht geschachtelte Baum-Rekursion
um ack zu berechnen
- hat Anwendung in der Komplexitätstheorie

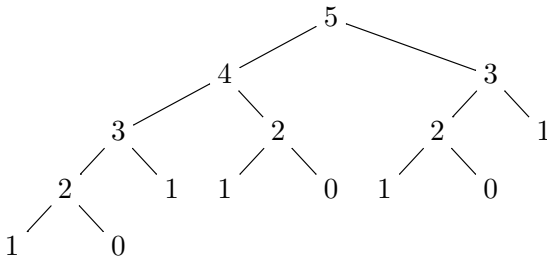
„impliziert“ ↑	geschachtelt baumrekursiv	mehrere rekursive Unterausdrücke auch in den Argumenten der rekursiven Unterausdrücke erlaubt
	baumrekursiv	mehrere rekursive Unterausdrücke erlaubt, aber Argumente der rekursiven Unterausdrücke ohne weitere Rekursion
	linear rekursiv	maximal ein rekursiver Unterausdruck
	endrekursiv	linear rekursiv und Gesamtergebn ist Wert des rekursiven Unterausdrucks
	iterativ	endrekursiv und Argumente des rekursiven Unterausdrucks sind Basiswerte

Im folgenden: Analyse auf Laufzeit und Speicherbedarf von:

- fib
- fakultaet
- ggt

Beispiel

Berechnung von (fib 5)



fib 3 wird 2 mal berechnet

fib 2 wird 3 mal berechnet

fib 1 wird 5 mal berechnet

Genauer und Allgemeiner:

Bei Berechnung von `fib n` für $n \geq 2$
wird `fib(1)` jeweils `(fib n)`-mal berechnet

$$(\text{fib } n) \approx \frac{\Phi^n}{\sqrt{5}} \quad \text{wobei } \Phi = \frac{1+\sqrt{5}}{2} \approx 1.6180 \text{ (goldener Schnitt)}$$

Fazit:

Reduktionen für `fib(n)` ist **exponentiell**
d.h. die Laufzeit von `fib` ist exponentiell in n

Beobachtung: zur Berechnung von $\text{fib}(n)$ benötigt man nur die Werte $\text{fib}(i)$ für $1 \leq i \leq n$.

Idee: Berechnung einer Wertetabelle für fib .

Verbesserte Variante: aus fib_{n-1} und fib_{n-2} berechne fib_n ohne Doppelberechnung von Funktionswerten.

Paar von Zahlen:

$$(\text{fib}_{n-1}, \text{fib}_{n-2}) \rightarrow ((\underbrace{\text{fib}_{n-1} + \text{fib}_{n-2}}_{= \text{fib}_n}), \text{fib}_{n-1})$$

Rechenvorschrift: $(a, b) \rightarrow (a + b, a)$

```
fib_lin n          = (fib_iter 1 0 n)
fib_iter a b zaehler = if zaehler <= 0
                        then b
                        else fib_iter (a + b)
                                      a
                                      (zaehler - 1)
```

Prozess für (fib_lin 5)

(fib_lin 5)

```
(fib_lin 5)
```

```
(fib_iter 1 0 5)
```



```
(fib_lin 5)  
(fib_iter 1 0 5)  
(fib_iter 1 1 4)
```

```
(fib_lin 5)  
(fib_iter 1 0 5)  
(fib_iter 1 1 4)  
(fib_iter 2 1 3)
```

```
(fib_lin 5)  
(fib_iter 1 0 5)  
(fib_iter 1 1 4)  
(fib_iter 2 1 3)  
(fib_iter 3 2 2)
```

```
(fib_lin 5)
(fib_iter 1 0 5)
(fib_iter 1 1 4)
(fib_iter 2 1 3)
(fib_iter 3 2 2)
(fib_iter 5 3 1)
```

```
(fib_lin 5)
(fib_iter 1 0 5)
(fib_iter 1 1 4)
(fib_iter 2 1 3)
(fib_iter 3 2 2)
(fib_iter 5 3 1)
5
```

Für `(fib_lin n)` gilt:

- ist operational äquivalent zu `fib`
- benötigt linear viele Reduktionen abhängig von n
- Größe der Ausdrücke ist beschränkt
- Platzbedarf ist konstant (d.h. unabhängig) von n .
(unter Vernachlässigung der Darstellungsgröße der Zahlen)

erzeugt **iterativen Auswertungsprozess** (applikative R.)

Abschätzung und Messung des Ressourcenbedarfs
von Haskell-Programmen,
bei verzögerter Auswertung:

Zeit: Anzahl der Transformationsschritte

Platz: (Gesamt-Speicher): Maximale Größe der Ausdrücke

Arbeitsspeicher: Maximale Größe der Ausdrücke
(ohne die Eingabe)

arithmetische und Boolesche Operationen
= 1 Transformationsschritt.

Angabe des Ressourcenbedarf eines Algorithmus
in Abhängigkeit von der Größe der Eingabe.

Notation für Algorithmus alg bei Eingabe der Größe n :

$\text{red}_{\text{alg}}(n)$ maximale Anzahl der Reduktionen bei verzögerter
Auswertung für alle Eingaben der Größe n .

$\text{Platz}_{\text{alg}}(n)$ Platzbedarf: maximale Größe der Ausdrücke (des
gerichteten Graphen) bei verzögerter Auswertung
für alle Eingaben der Größe n .
Die Eingaben nicht mitzählen


```
fib n = if n <= 0 then 0
        else if n == 1 then 1
            else fib (n-1) + fib(n-2)
```

$red_{\text{fib}}(n) \approx 1.6^n$ (einfach exponentiell)

Bezugsgröße: Zahl n

Achtung: Die Komplexitätsberechnung verwendet
i.a. die Speicher-Größe der Eingabe
(d.h. Anzahl der Stellen einer Zahl)

```
fakultaet x  = if x <= 1 then 1  
              else x*(fakultaet (x-1))
```

fakultaet n benötigt $5 * (n - 1) + 3$ Reduktionsschritte.
bei verzögerter Reihenfolge der Auswertung

(Kann man durch Beispielauswertungen raten)
Z.B. fakultaet 4 benötigt 18 Reduktionsschritte.

Nachweis mit vollständiger Induktion

Nachzuweisen ist: fakultaet $(n-1)$ benötigt $5 * (n - 2) + 4$
für $n \geq 2$ bei verzögerter Reihenfolge der Auswertung:

Induktions-Basis:

```
fakultaet (2-1)
if (2-1) <= 1 then 1 else ...
if 1 <= 1 then 1 else ...
if True then 1 else ...
1
```

Das sind 4 Reduktionsschritte.

Da $5 * (2 - 1 - 1) + 4 = 4$, gilt die Formel in diesem Fall.

Nachzuweisen ist: fakultaet (n-1) benötigt $5 * (n - 2) + 4$ für $n > 2$
bei verzögerter Reihenfolge der Auswertung:

Induktions-Schritt:

```
fakultaet (n-1)
if (n-1) <= 1 then ...
if n1 <= 1 then ...    -- n1 ist Basiswert > 1
if False then ...
n1*fakultaet (n1-1)    -- Berechnung von n2 geht
                        -- als Ind.-hypothese ein
n1*n2                  -- Produkt-Berechnung zaehlt noch dazu
n3
```

Das sind $5 + 5 * (n1 - 2) + 4 = 5 * (n - 2) + 4$ Reduktionsschritte

Nachzuweisen ist: fakultaet n benötigt $5 * (n - 1) + 3$ für $n > 1$
bei verzögerter Reihenfolge der Auswertung:

Induktions-Schritt:

```
fakultaet n
if n <= 1 then ...      n ist > 1
if False then ...
n*fakultaet (n-1)  -- s.o.: 5*(n-2)+4 Schritte
n*n2              -- Produkt-Berechnung zaehlt noch dazu
n3
```

Das sind $4 + 5 * (n - 2) + 4 = 5 * (n - 1) + 3$ Reduktionsschritte

Für $n = 4$ ergibt das $5 * 3 + 3 = 18$ Reduktionsschritte für
fakultaet 4.

Beachte: breite Streuung des Ressourcenbedarfs ist möglich für die Menge aller Eingaben einer bestimmten Größe.

Komplexitäten von Platz und Zeit:

Ressourcenbedarf verschiedene Varianten:

im *schlimmsten Fall* (worst-case)

im *besten Fall* (best-case)

Minimum der Anzahl der Reduktionen
bzw. Minimum der Größe der Ausdrücke.

im *Mittel* (average case)

Welche Verteilung?

Als Beispiel hatten wir das Ergebnis:

fakultaet (n-1) benötigt $5 * (n - 1) + 3$ Reduktionen für $n > 2$.

Abschätzung von $5 * (n - 1) + 3$ nach oben (als Funktion von n):

$$5 * (n - 1) + 3 \leq 6 \cdot n$$

Geschrieben als $\lambda n. (5 * (n - 1) + 3) \in O(n)$.

! Multiplikative und additive Konstanten werden ignoriert.

$O(1)$	konstant
$O(\log(n))$	logarithmisch
$O(n)$	linear
$O(n * \log(n))$	fastlinear (oder auch $n \cdot \log n$)
$O(n^2)$	quadratisch
$O(n^3)$	kubisch
$O(n^k)$	polynomiell
$O(2^n)$	exponentiell

n ist die Größe der Eingabe (i.a Bit-Anzahl)

$\text{ggT}(a, b)$ (Euklids Algorithmus)

*Teile a durch b ; ergibt Rest r ,
wenn $r = 0$, dann $\text{ggT}(a, b) := b$
wenn $r \neq 0$, dann berechne $\text{ggT}(b, r)$.*

Beispiel $\text{ggT}(30, 12) = \text{ggT}(12, 6) = 6$

```
ggT a b =    if b == 0
              then a
              else ggT b (rem a b)
```

SATZ (Lamé, 1845):

Wenn der Euklidische ggt-Algorithmus k Schritte benötigt,
dann ist die kleinere Zahl der Eingabe $\geq fib(k)$.

Platz- und Zeitbedarf von ggt: $O(\log(n))$

Begründung:

Wenn n die kleinere Zahl ist und der Algorithmus k Schritte benötigt,

dann ist $n \geq fib(k) \approx 1.6180^k$

Also ist Anzahl der Rechenschritte $k = O(\log(n))$

Aufruf	Zeitaufwand - Abschätzung	
Arithmetische Operationen als $O(1)$ angenommen		
fakultaet n	$O(n)$	
fib n	$O(1, 62^n)$	
fib_lin n	$O(n)$	
ggt $m\ n$	$O(\log(\max(m, n)))$	
$m + n$	$O(1)$	$m, n :: \text{Int}$
$m * n$	$O(1)$	$m, n :: \text{Int}$
quadratsumme $m\ n$	$O(1)$	$m, n :: \text{Int}$
Arithmetische Operationen auf großen Zahlen		
$m + n$	$O(\log(m + n))$	$m, n :: \text{Integer}$
$m * n$	$O(\log(m + n))$	$m, n :: \text{Integer}$
quadratsumme $m\ n$	$O(\log(m + n))$	$m, n :: \text{Integer}$