

Visualisierung von Lazy Evaluation und Sharing

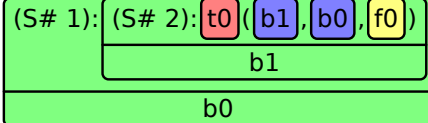
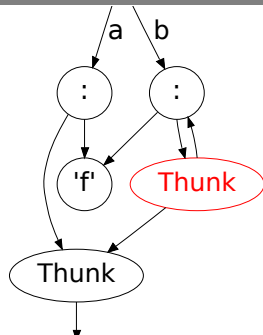
Vortrag zur Bachelorarbeit

Dennis Felsing | 2012-10-30

INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION

```
λ> let a = "foo"
λ> let b = a ++ b
λ> let fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

fibs: (S# 0):(S# 1):(S# 1):(S# 2):t0((b1, b0, f0))

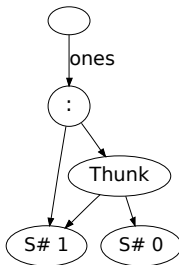
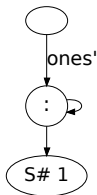
Motivation

- Manche Konzepte in **Haskell** schwer zu verstehen

$ones = [1, 1 ..]$

$ones' = 1 : ones'$

- Visualisierung** kann helfen:



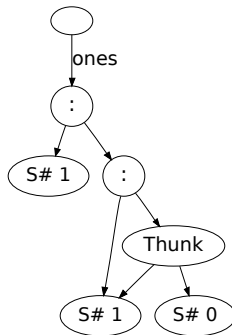
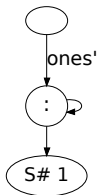
Motivation

- Manche Konzepte in **Haskell** schwer zu verstehen

$ones = [1, 1 ..]$

$ones' = 1 : ones'$

- Visualisierung** kann helfen:



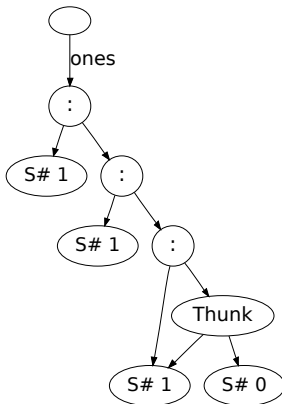
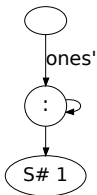
Motivation

- Manche Konzepte in **Haskell** schwer zu verstehen

$ones = [1, 1 ..]$

$ones' = 1 : ones'$

- Visualisierung** kann helfen:

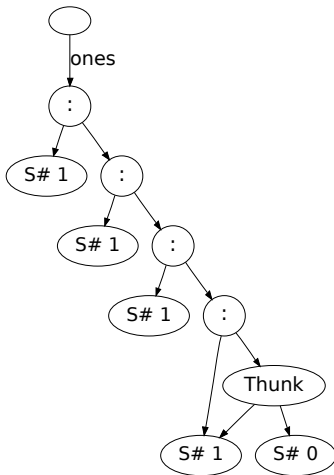
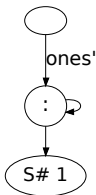


- Manche Konzepte in **Haskell** schwer zu verstehen

$ones = [1, 1 ..]$

$ones' = 1 : ones'$

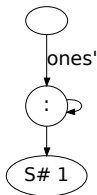
- Visualisierung** kann helfen:



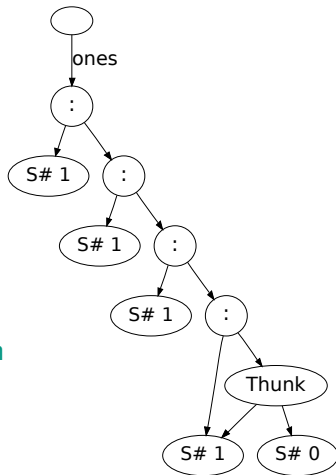
- Manche Konzepte in **Haskell** schwer zu verstehen

$ones = [1, 1 ..]$
 $ones' = 1 : ones'$

- Visualisierung** kann helfen:



- Ziel: **Sharing** und **Lazy Evaluation** intuitiv sichtbar machen



- 1 Grundlagen
 - Haskell
 - Lazy Evaluation
 - Sharing
 - GHC Heap
- 2 ghc-vis: Tool zur Visualisierung
- 3 Evaluierung

- Funktionale Programmiersprache
- **Reinheit**: Keine Seiteneffekte
- \Rightarrow **Referenzielle Transparenz**: Gleiche Ausdrücke werten zu gleichem Wert aus, Zeitpunkt der Auswertung spielt keine Rolle

- Auch bekannt als **call-by-need**
- Ausdrücke **so spät wie möglich** auswerten \Rightarrow Wenn benötigt
- Spart automatisch unnötige Berechnungen:

squares = *map* ($\uparrow 2$) [1..]

smallSquares = *takeWhile* (≤ 20) *squares*

- Modularität: Aufteilen in **Generator und Selektor**

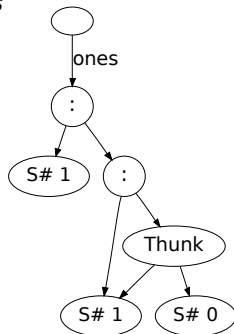
- Auch bekannt als **call-by-need**
- Ausdrücke **so spät wie möglich** auswerten \Rightarrow Wenn benötigt
- Spart automatisch unnötige Berechnungen:

squares = *map* (\uparrow^2) [1..]

smallSquares = *takeWhile* (≤ 20) *squares*

- Modularität: Aufteilen in **Generator und Selektor**
- Beispiel aus der Motivation:

ones = [1, 1..]



- Werte sind **unveränderlich**
- \Rightarrow Müssen nicht kopiert werden
- Stattdessen vorhandene Werte **wiederverwenden**:

squares = *map* (\uparrow^2) [1..]

smallSquares = *takeWhile* (≤ 20) *squares*

- Spart mehrfaches Auswerten:

doubleTotal = *total* + *total*
where *total* = *sum* [1..100]

- Werte sind **unveränderlich**
- \Rightarrow Müssen nicht kopiert werden
- Stattdessen vorhandene Werte **wiederverwenden**:

squares = *map* (\uparrow^2) [*1..*]

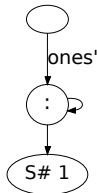
smallSquares = *takeWhile* (≤ 20) *squares*

- Spart mehrfaches Auswerten:

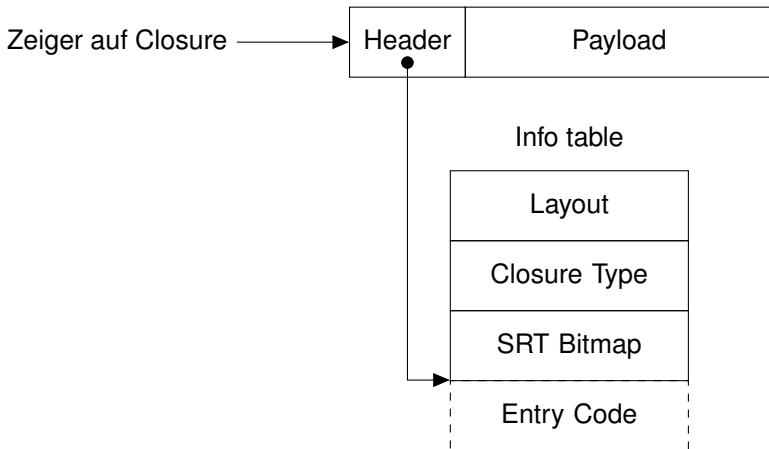
doubleTotal = *total* + *total*
where *total* = *sum* [*1..100*]

- Beispiel aus der Motivation:

ones' = 1 : *ones'*



- Alle dynamischen Objekte auf Heap
- Einheitliches Layout als Closure



Wichtige Closure-Typen:

- Data Constructor
- Function und Partial Application
- Thunk und General Application
- Byte Code Object

Wichtige Closure-Typen:

- Data Constructor

data *Maybe a = Nothing | Just a*

- Function und Partial Application
- Thunk und General Application
- Byte Code Object

Wichtige Closure-Typen:

- Data Constructor
- Function und Partial Application

square $x = x * x$

- Thunk und General Application
- Byte Code Object

Wichtige Closure-Typen:

- Data Constructor
- Function und Partial Application
- Thunk und General Application

head [1, 2, 3]

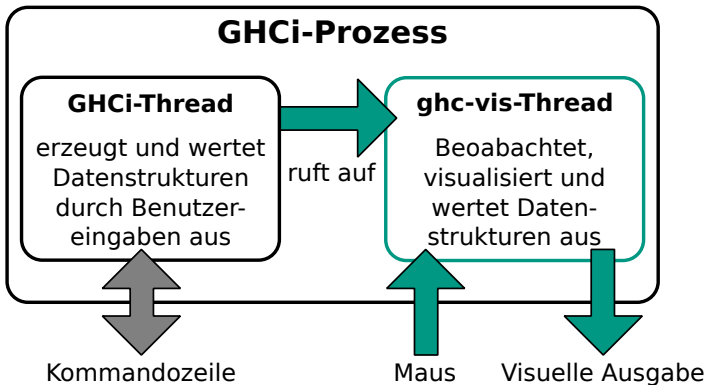
- Byte Code Object

1 Grundlagen

2 ghc-vis: Tool zur Visualisierung

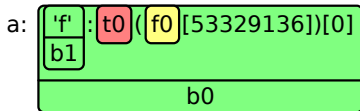
- Design
- Ansichten
- Implementierung

3 Evaluierung

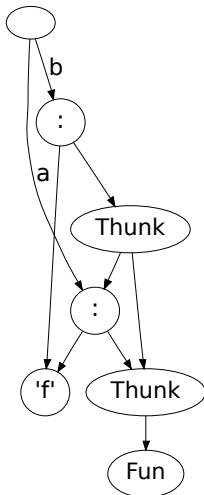


- ghc-vis benötigt Repräsentation des relevanten Teils des GHC-Heaps: **Heap-Map**
- Folge rekursiv Zeigern in Closures
- Stoppe bei bereits bekannten Closures
- Ansichten werden aus Heap-Map generiert

```
ghci> let a = "foo"  
ghci> let b = a ++ a  
ghci> head b  
'f'
```

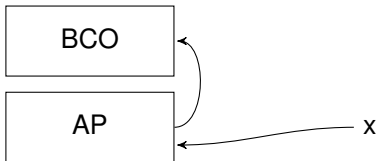


Lineare Ansicht

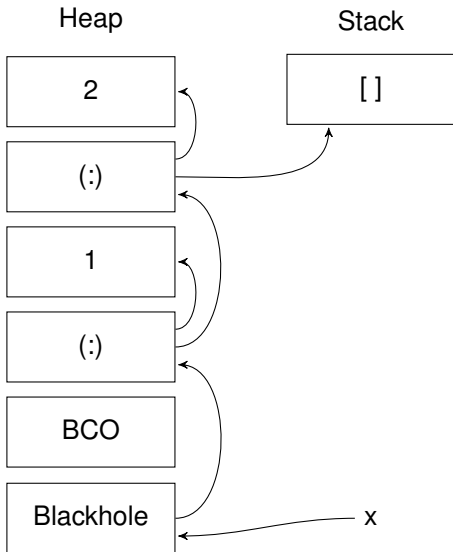


Graph-Ansicht

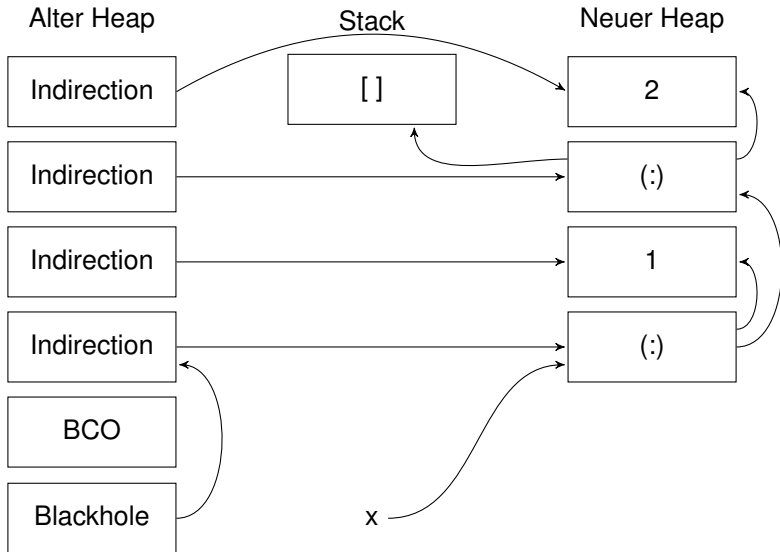
Heap



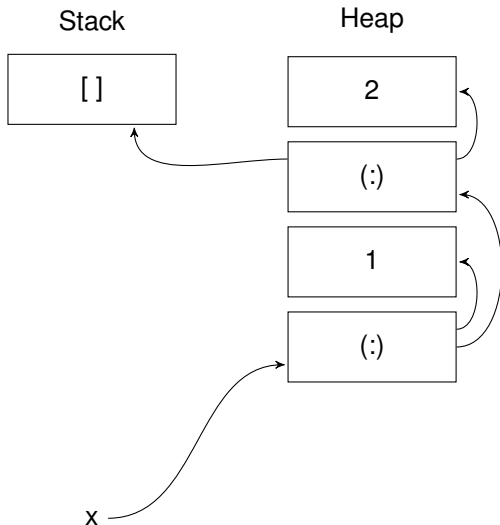
Implementierung: Garbage Collection



Implementierung: Garbage Collection



Implementierung: Garbage Collection



Ausschnitt aus der Heap-Map vor Garbage Collection:

```
[...  
, (0x00007f732e8dbc40/2, (Nothing, ConsClosure {  
  ptrArgs = [0x00007f732e8dbce0, 0x00007f732e8dbcc8/2],  
  dataArgs = [], name = ":"}))  
, (0x00007f732dac5380, (Just "x", BlackholeClosure {  
  indirectee = 0x00007f732e8dbc40/2}))  
, ...]
```

Ausschnitt aus **derselben** Heap-Map nach Garbage Collection:

```
[...  
, (0x00007f732dcad4b0/2, (Nothing, ConsClosure {  
  ptrArgs = [0x00007f732dcad890, 0x00007f732dcad878/2],  
  dataArgs = [], name = ":"}))  
, (0x00007f732dcad4b0/2, (Just "x", BlackholeClosure {  
  indirectee = 0x00007f732dcad4b0/2}))  
, ...]
```

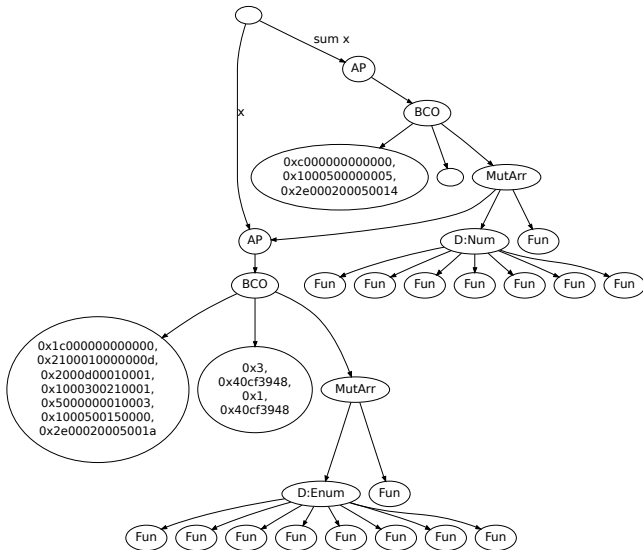
Ausschnitt aus **derselben** Heap-Map nach Garbage Collection:

```
[...  
, (0x00007f732dcad4b0/2, (Nothing, ConsClosure {  
  ptrArgs = [0x00007f732dcad890, 0x00007f732dcad878/2],  
  dataArgs = [], name = ":"}))  
, (0x00007f732dcad4b0/2, (Just "x", BlackholeClosure {  
  indirectee = 0x00007f732dcad4b0/2}))  
, ...]
```

Lösung: Garbage Collection vor Erzeugen der Ansicht erzwingen

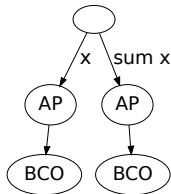
Implementierung: Byte Code Objects

Vollständige BCO-Informationen:



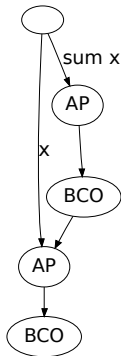
Implementierung: Byte Code Objects

Keine BCO-Informationen:



Implementierung: Byte Code Objects

⇒ **Relevante** BCO-Informationen auswählen:



- 1 Grundlagen
- 2 ghc-vis: Tool zur Visualisierung
- 3 Evaluierung**
 - Vergleich
 - Demonstration

- `:print` Bestandteil von GHCi
- Kein Sharing
⇒ Keine unendlichen Datenstrukturen

```
ghci> let a = "foo"  
ghci> let b = a ++ a  
ghci> head b  
'f'
```

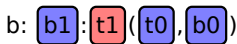
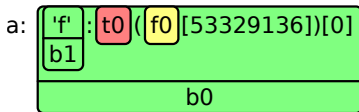
```
ghci> :print a  
a = 'f' : (_t1::[Char])  
ghci> :print b  
b = 'f' : (_t2::[Char])
```

Vergleich: :print

- `:print` Bestandteil von GHCi
- Kein Sharing
 - ⇒ Keine unendlichen Datenstrukturen

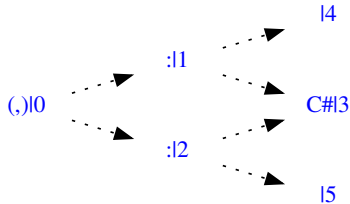
```
ghci> let a = "foo"  
ghci> let b = a ++ a  
ghci> head b  
'f'
```

```
ghci> :print a  
a = 'f' : (_t1::[Char])  
ghci> :print b  
b = 'f' : (_t2::[Char])
```



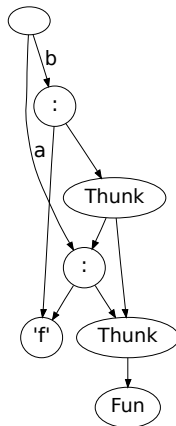
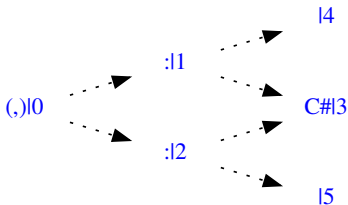
Vergleich: vacuum

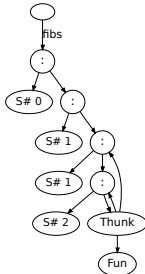
- Kein Sharing hinter unausgewerteten Ausdrücken
- Nicht interaktiv



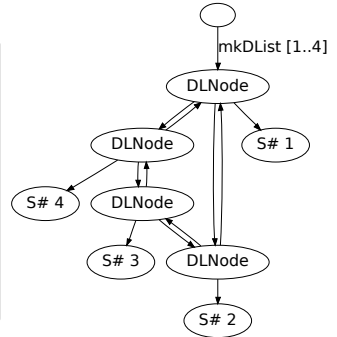
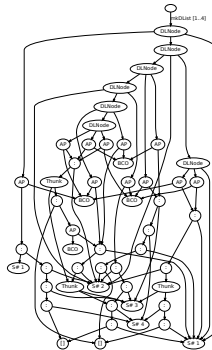
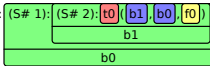
Vergleich: vacuum

- Kein Sharing hinter unausgewerteten Ausdrücken
- Nicht interaktiv





fibs: (S# 0):(S# 1):



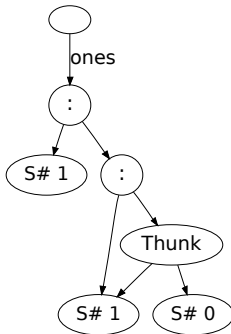
- Live Datenstrukturen beobachten
- Integriert in GHCi
- Verständnis **Lazy Evaluation** und **Sharing**
- Einsatz in Lehre und Entwicklung in Haskell

Zukünftige Arbeit:

- Besser nach relevanten Datenstrukturen filtern
- Informationen über Typen
- Substrukturen aus ghc-vis in GHCi als Variablen

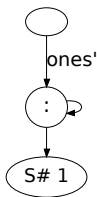
- Auch bekannt als **call-by-need**
- Ausdrücke **so spät wie möglich** auswerten \Rightarrow Wenn benötigt
- Spart automatisch unnötige Berechnungen
- Beispiel aus der Motivation:

ones = [1, 1 ..]



- Werte sind **unveränderlich**
- \Rightarrow Müssen nicht kopiert werden
- Stattdessen vorhandene Werte **wiederverwenden**
- Beispiel aus der Motivation:

$$ones' = 1 : ones'$$

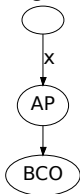


- Two-Space Stop-and-Copy Collector
- Alle benötigten Closures aus einer Hälfte des Heaps in andere kopieren
- Pointer in Closures verfolgen um weitere Closures zu finden

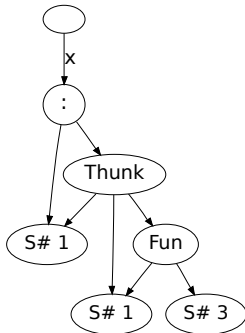
- Closure wird durch **Weak-Head Normal Form** ersetzt

$x = [1..3]$

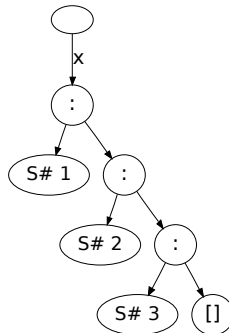
Unausgewertet



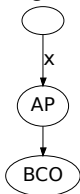
WHNF



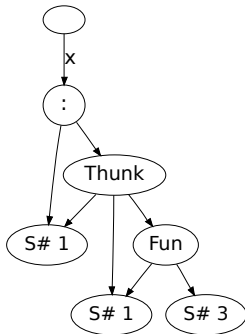
Normalform



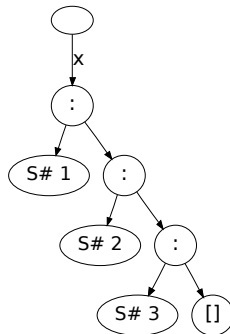
Unausgewertet



WHNF

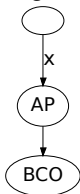


Normalform

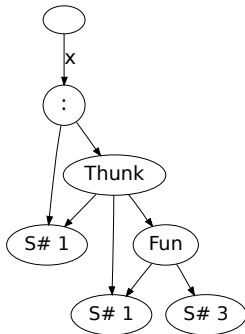


- Data Constructor
- Function Closure und Partial Application
- Thunk und General Application

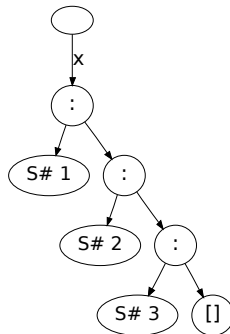
Unausgewertet



WHNF

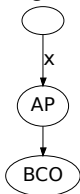


Normalform

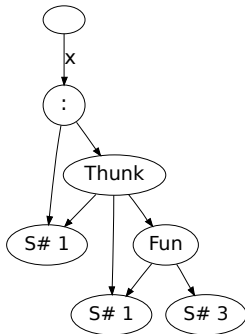


- Data Constructor \Rightarrow WHNF
- Function Closure und Partial Application
- Thunk und General Application

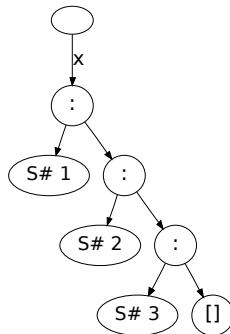
Unausgewertet



WHNF

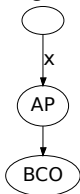


Normalform

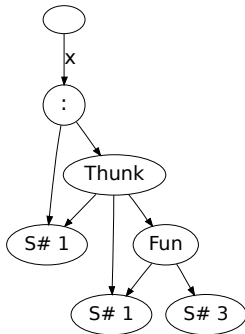


- Data Constructor \Rightarrow WHNF
- Function Closure und Partial Application \Rightarrow WHNF
- Thunk und General Application

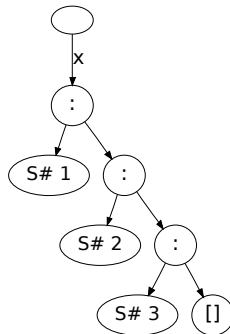
Unausgewertet



WHNF



Normalform



- **Data Constructor** \Rightarrow WHNF
- **Function Closure** und Partial Application \Rightarrow WHNF
- **Thunk** und General Application \Rightarrow Unausgewertet