Karlsruher Institut für Technologie

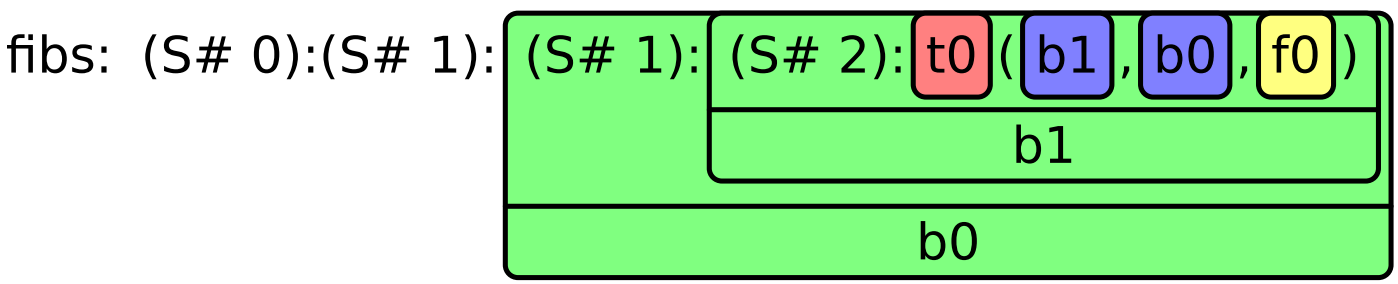
Visualisierung von Lazy Evaluation und Sharing

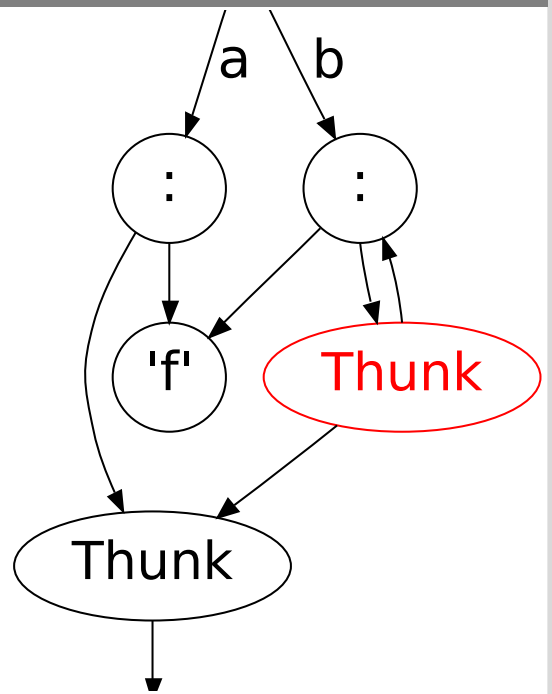
Vortrag zur Bachelorarbeit

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)
```



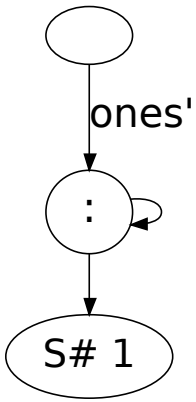


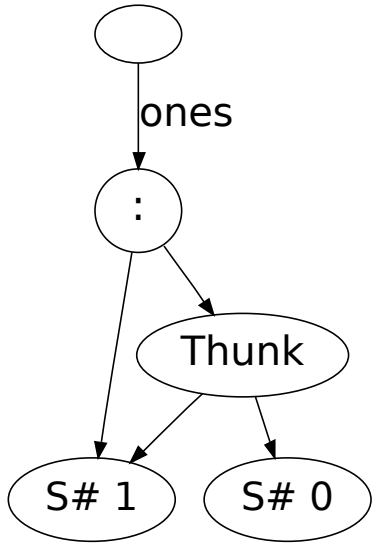
Motivation

- Manche Konzepte in Haskell schwer zu verstehen

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

- Visualisierung kann helfen:



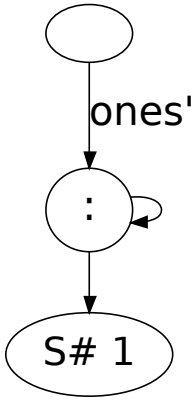


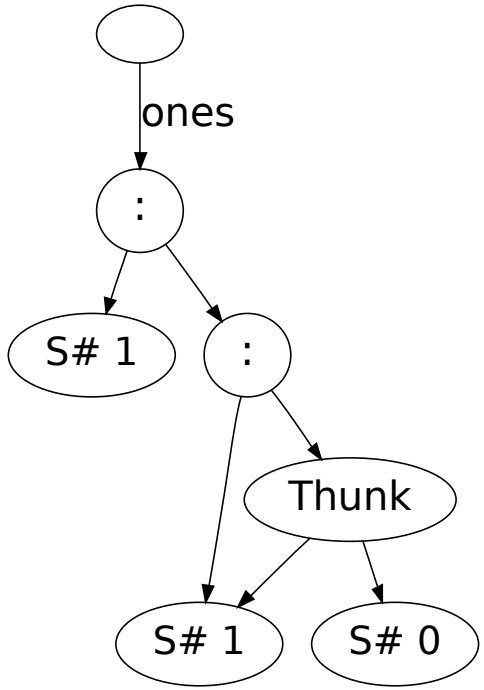
Motivation

- Manche Konzepte in Haskell schwer zu verstehen

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

- Visualisierung kann helfen:



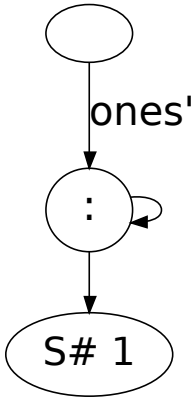


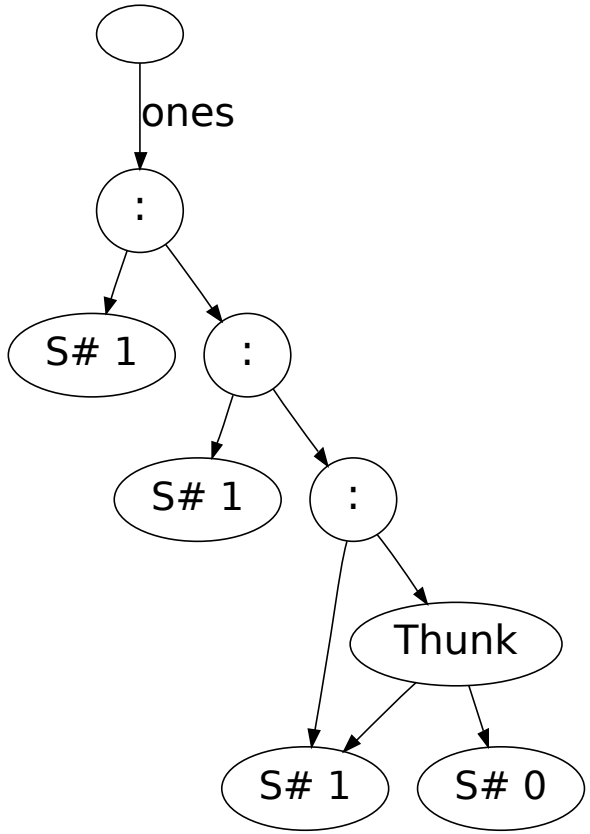
Motivation

- Manche Konzepte in Haskell schwer zu verstehen

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

- Visualisierung kann helfen:



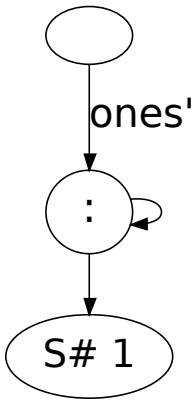


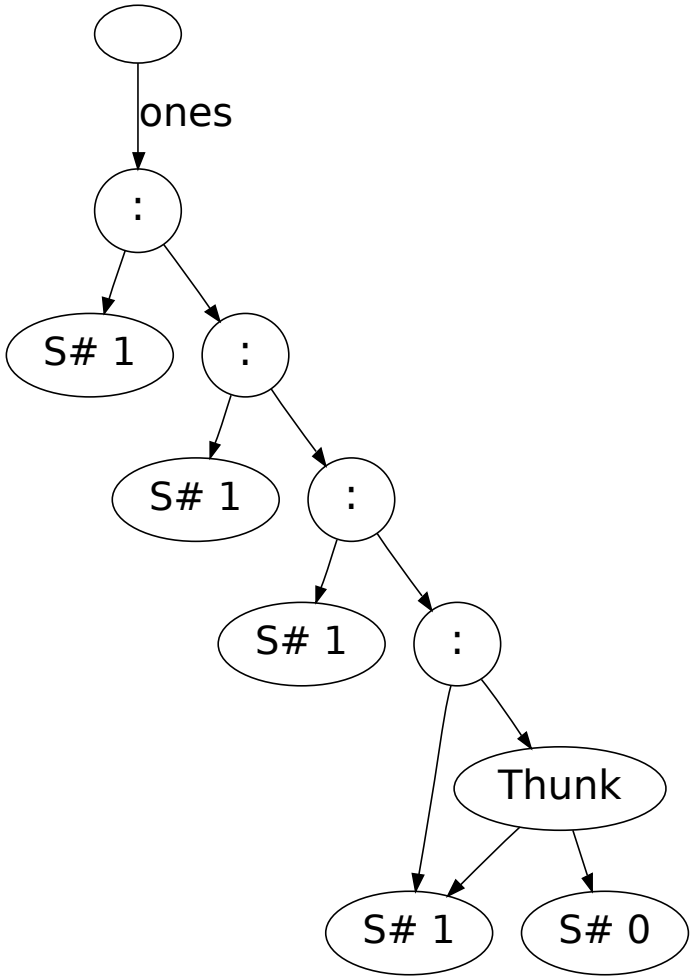
Motivation

- Manche Konzepte in Haskell schwer zu verstehen

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

- Visualisierung kann helfen:



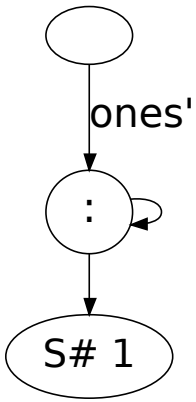


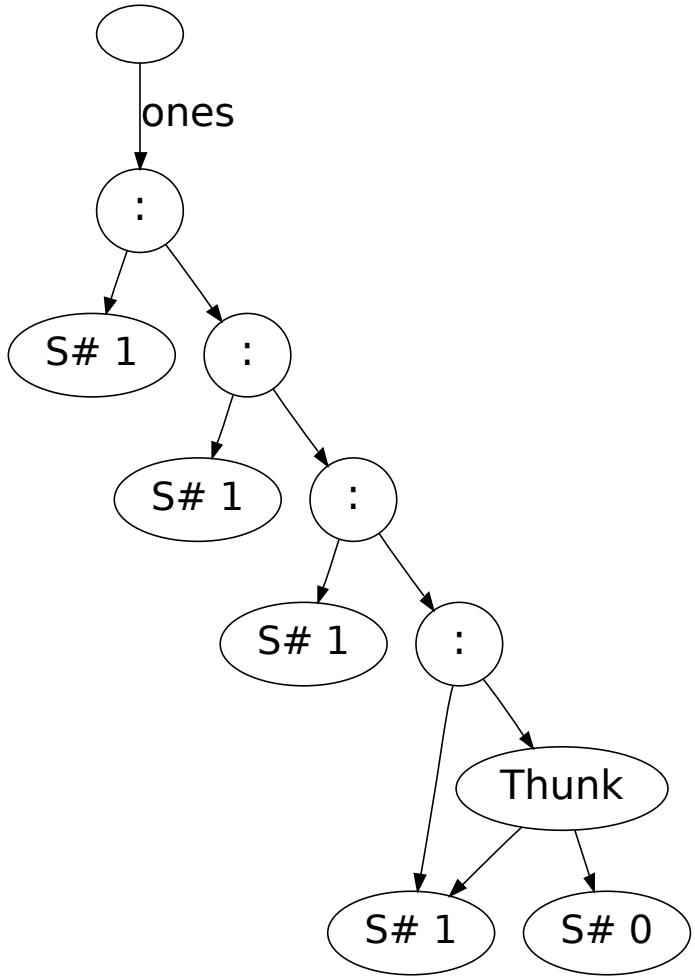
Motivation

- Manche Konzepte in Haskell schwer zu verstehen

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

- Visualisierung kann helfen:





- Ziel: Sharing und Lazy Evaluation intuitiv sichtbar machen

Überblick

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

Haskell

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

Lazy Evaluation

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

$$\begin{aligned} \mathit{squares} &= \mathit{map}\ (\uparrow 2)\ [1\,..] \\ \mathit{smallSquares} &= \mathit{takeWhile}\ (\leqslant 20)\ \mathit{squares} \end{aligned}$$

- Modularität: Aufteilen in Generator und Selektor

Lazy Evaluation

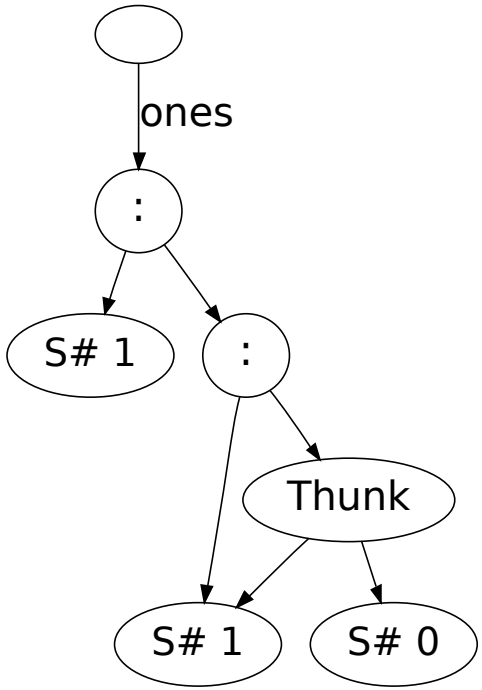
- Auch bekannt als call-by-need
- Ausdrücke so spät wie möglich auswerten $\Rightarrow$ Wenn benötigt
- Spart automatisch unnötige Berechnungen:

$$\begin{aligned} \mathit{squares} &= \mathit{map}\ (\uparrow 2)\ [1\,..] \\ \mathit{smallSquares} &= \mathit{takeWhile}\ (\leqslant 20)\ \mathit{squares} \end{aligned}$$

- Modularität: Aufteilen in Generator und Selektor

- Beispiel aus der Motivation:

$$\mathit{ones} = [1, 1\,..]$$



Sharing

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

$$\begin{aligned} \mathit{squares} &= \mathit{map}\ (\uparrow 2)\ [1\,..] \\ \mathit{smallSquares} &= \mathit{takeWhile}\ (\leqslant 20)\ \mathit{squares} \end{aligned}$$

- Spart mehrfaches Auswerten:

$$\begin{aligned} &\mathit{doubleTotal} = \mathit{total} + \mathit{total} \\ &\quad \textbf{where}\ \mathit{total} = \mathit{sum}\ [1\,..\,100] \end{aligned}$$

Sharing

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

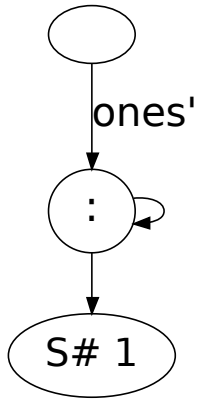
$$\begin{aligned} \mathit{squares} &= \mathit{map}\ (\uparrow 2)\ [1\,..] \\ \mathit{smallSquares} &= \mathit{takeWhile}\ (\leqslant 20)\ \mathit{squares} \end{aligned}$$

- Spart mehrfaches Auswerten:

$$\begin{aligned} &\mathit{doubleTotal} = \mathit{total} + \mathit{total} \\ &\quad \textbf{where}\ \mathit{total} = \mathit{sum}\ [1\,..\,100] \end{aligned}$$

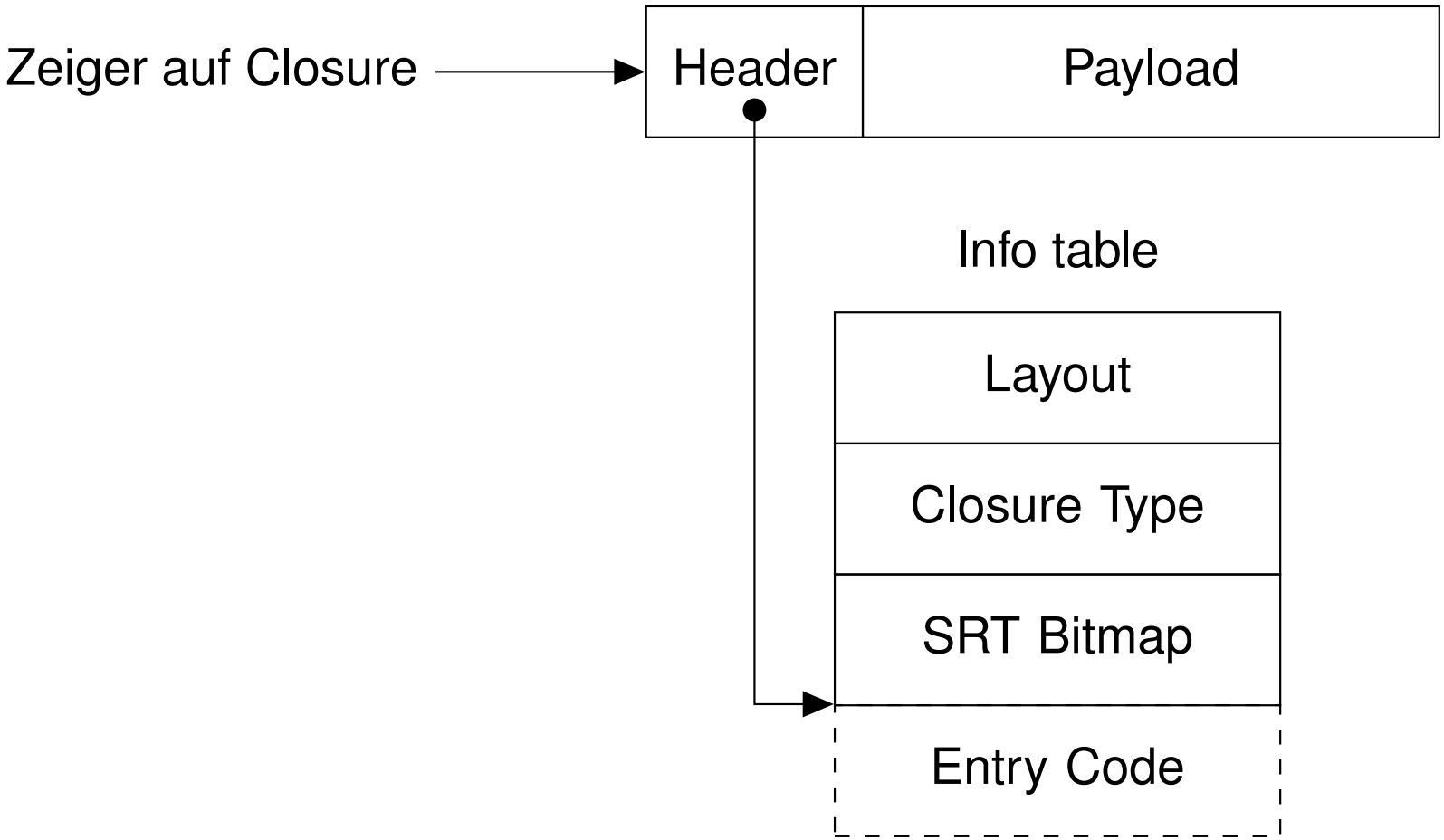
- Beispiel aus der Motivation:

$$\mathit{ones}' = 1 : \mathit{ones}'$$



GHC Heap: Layout

- Alle dynamischen Objekte auf Heap
- Einheitliches Layout als Closure



GHC Heap: Closures

Karlsruher Institut für Technologie

Wichtige Closure-Typen:

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

GHC Heap: Closures

Wichtige Closure-Typen:

- Data Constructor

$$\textbf{data}\ \mathit{Maybe}\ a = \mathit{Nothing} \mid \mathit{Just}\ a$$

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

GHC Heap: Closures

Wichtige Closure-Typen:

- Data Constructor
- Function und Partial Application

$$square\ x = x * x$$

- Thunk und General Application
- Byte Code Object

GHC Heap: Closures

Wichtige Closure-Typen:

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

$$\mathit{head}\ [1, 2, 3]$$

- Byte Code Object

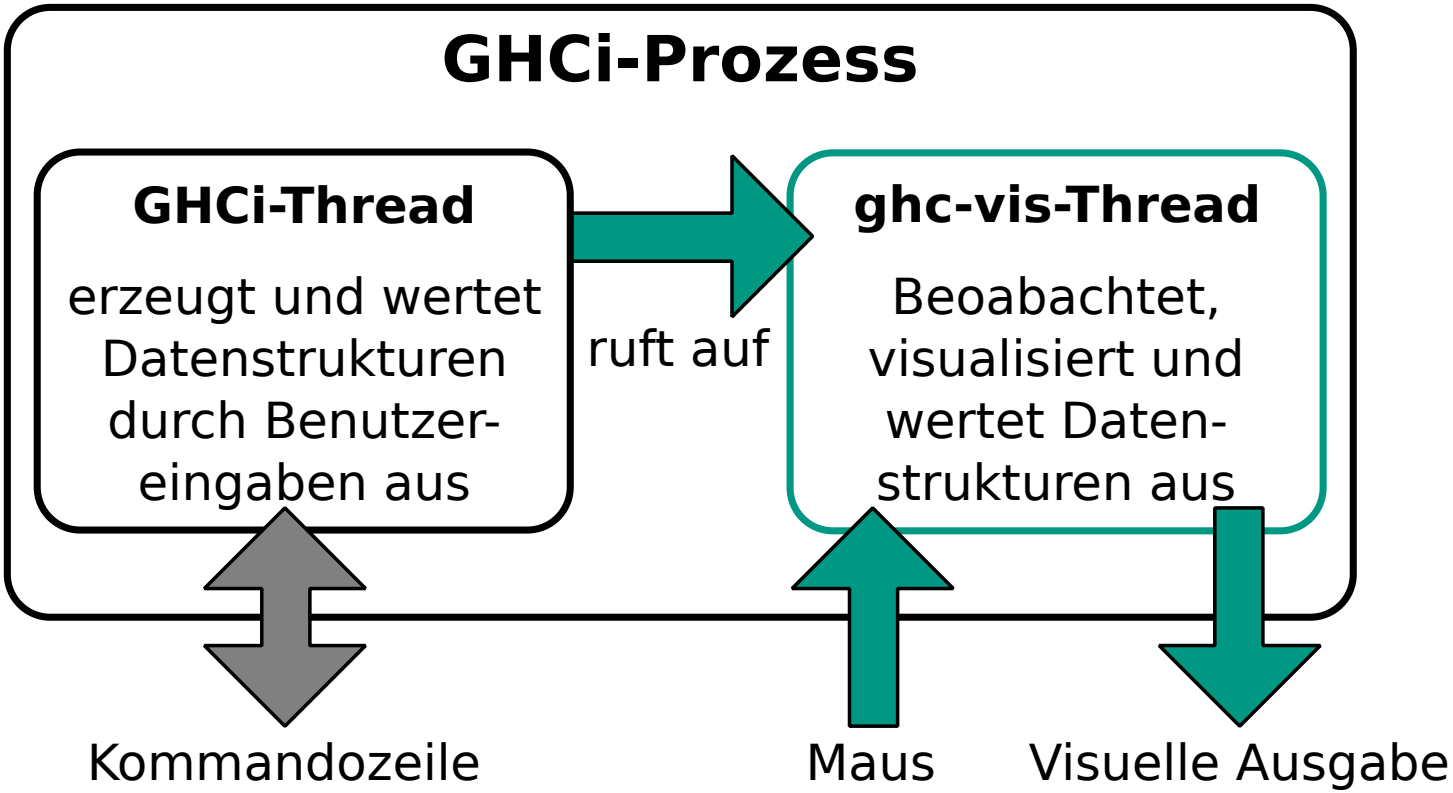
ghc-vis

1. Grundlagen

2. ghc-vis: Tool zur Visualisierung
   - Design
   - Ansichten
   - Implementierung

3. Evaluierung

Design

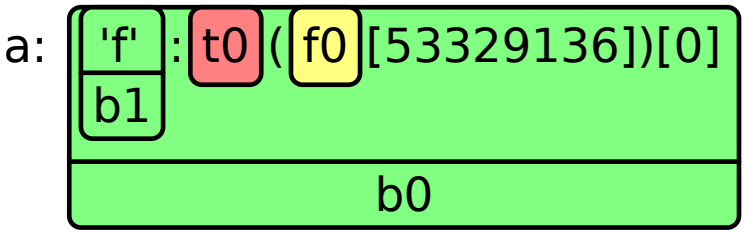

GHCi-Prozess
GHCi-Thread
erzeugt und wertet Datenstrukturen durch Benutzer-eingaben aus
ruft auf
ghc-vis-Thread
Beoabachtet, visualisiert und wertet Daten-strukturen aus
Kommandozeile
Maus
Visuelle Ausgabe

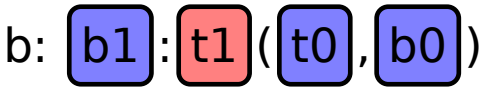
Design: Heap-Map

- 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

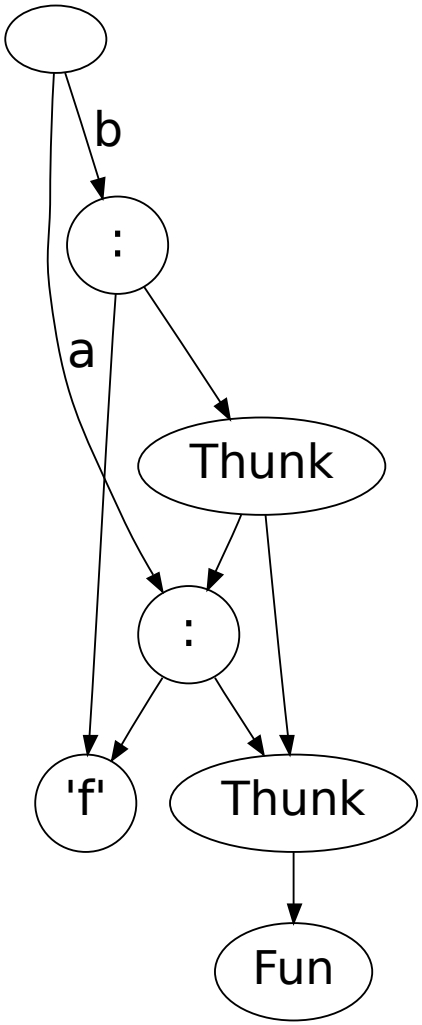
Ansichten

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





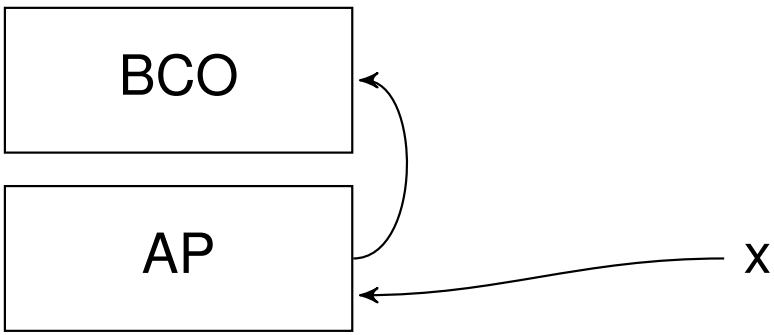
Lineare Ansicht



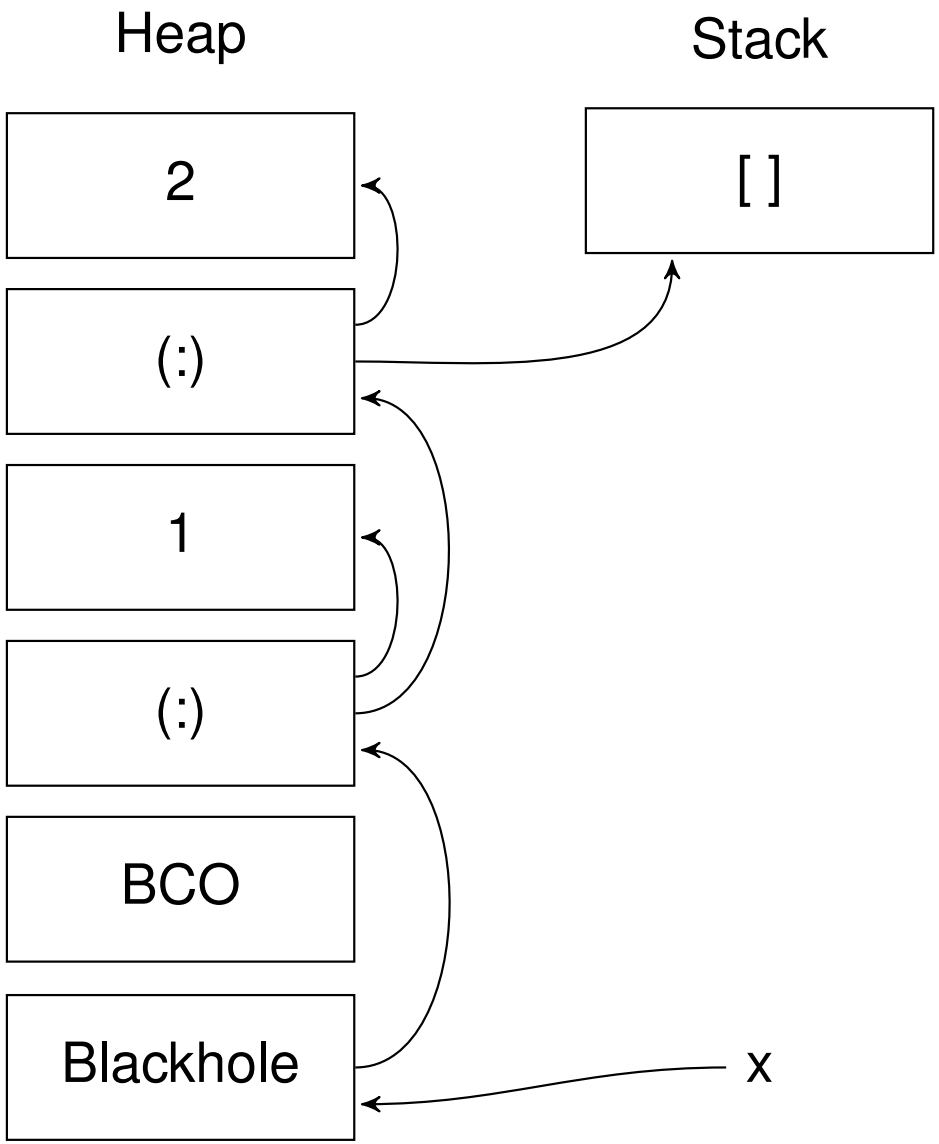
Graph-Ansicht

Implementierung: Garbage Collection

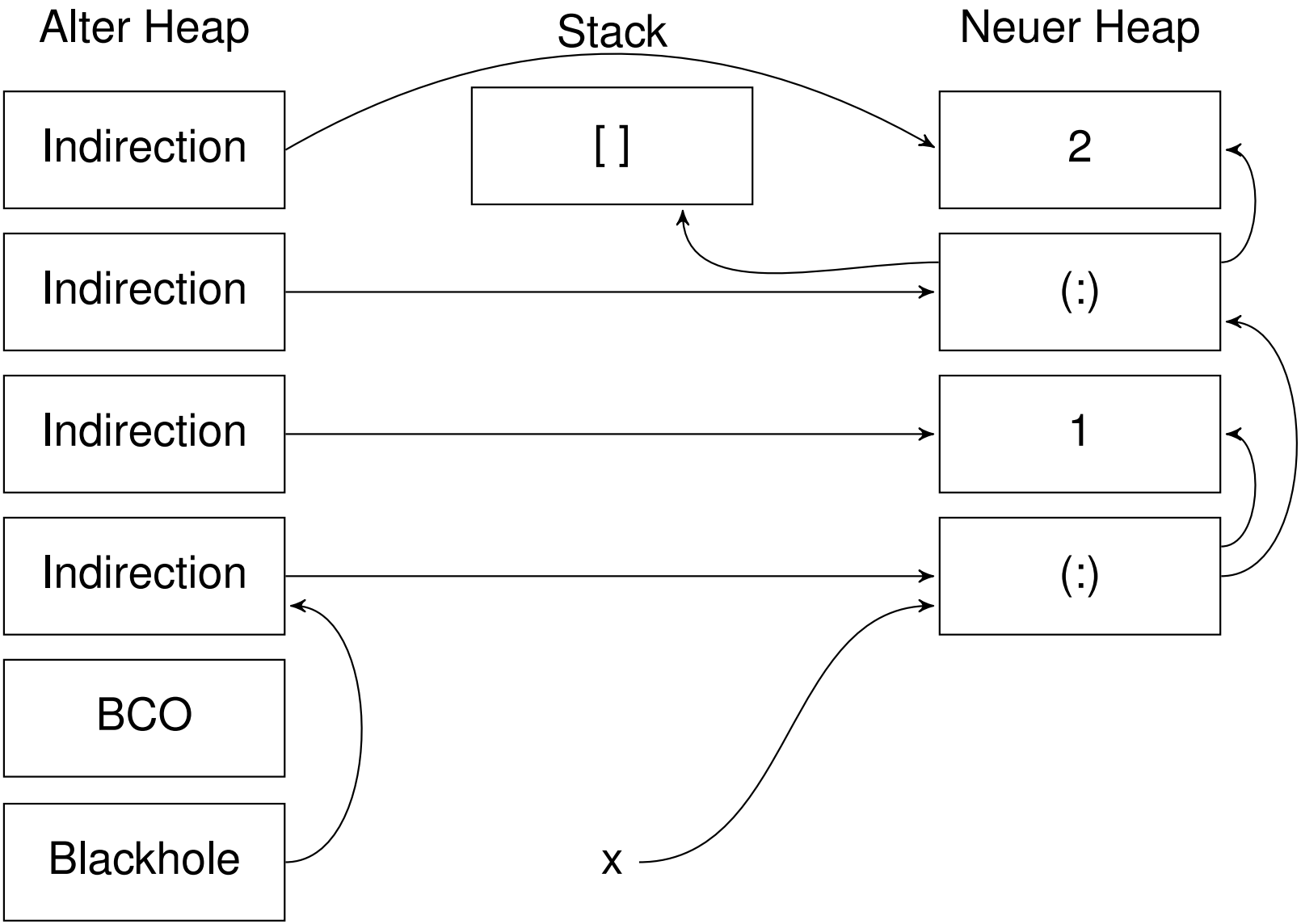
Heap



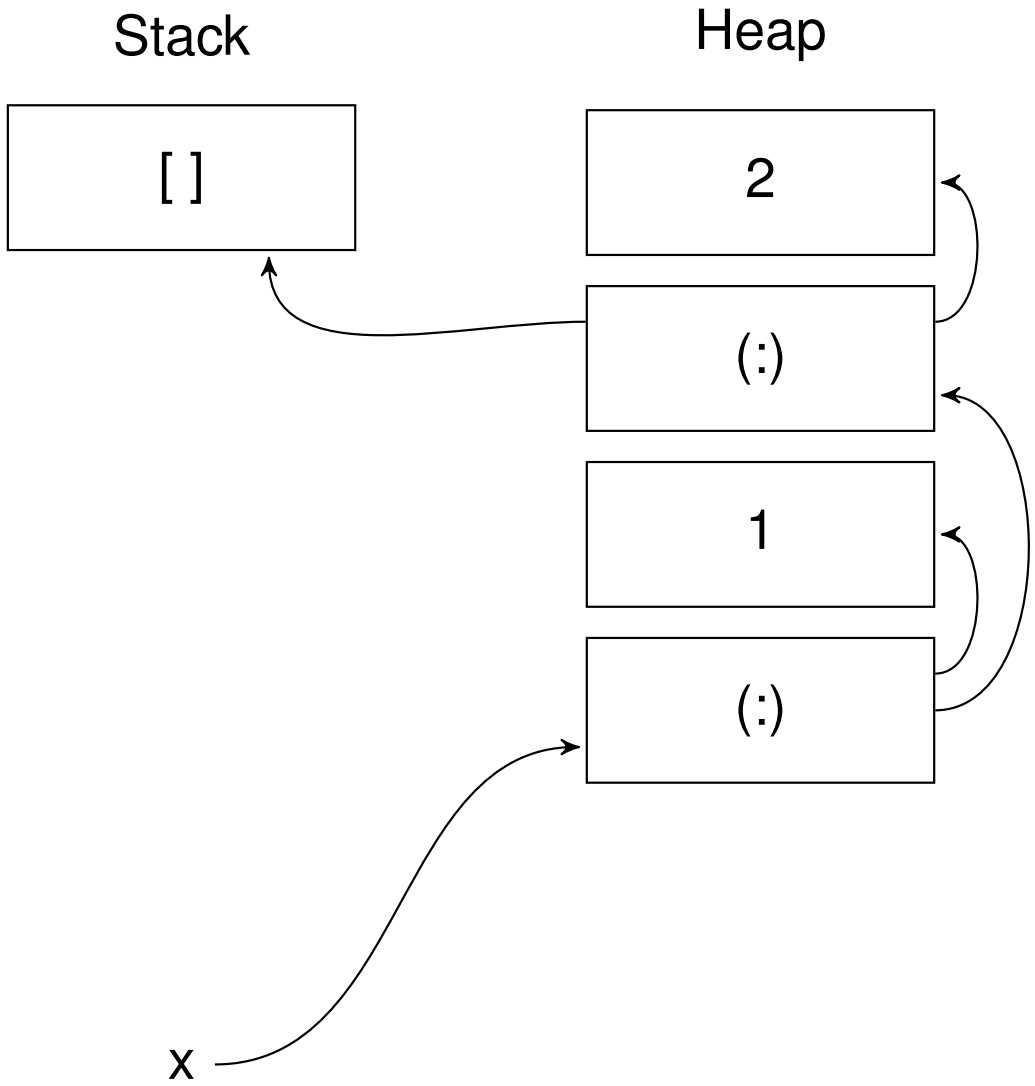
Implementierung: Garbage Collection



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}))
,...]
```

Implementierung: Garbage Collection

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}))
,...]
```

Implementierung: Garbage Collection

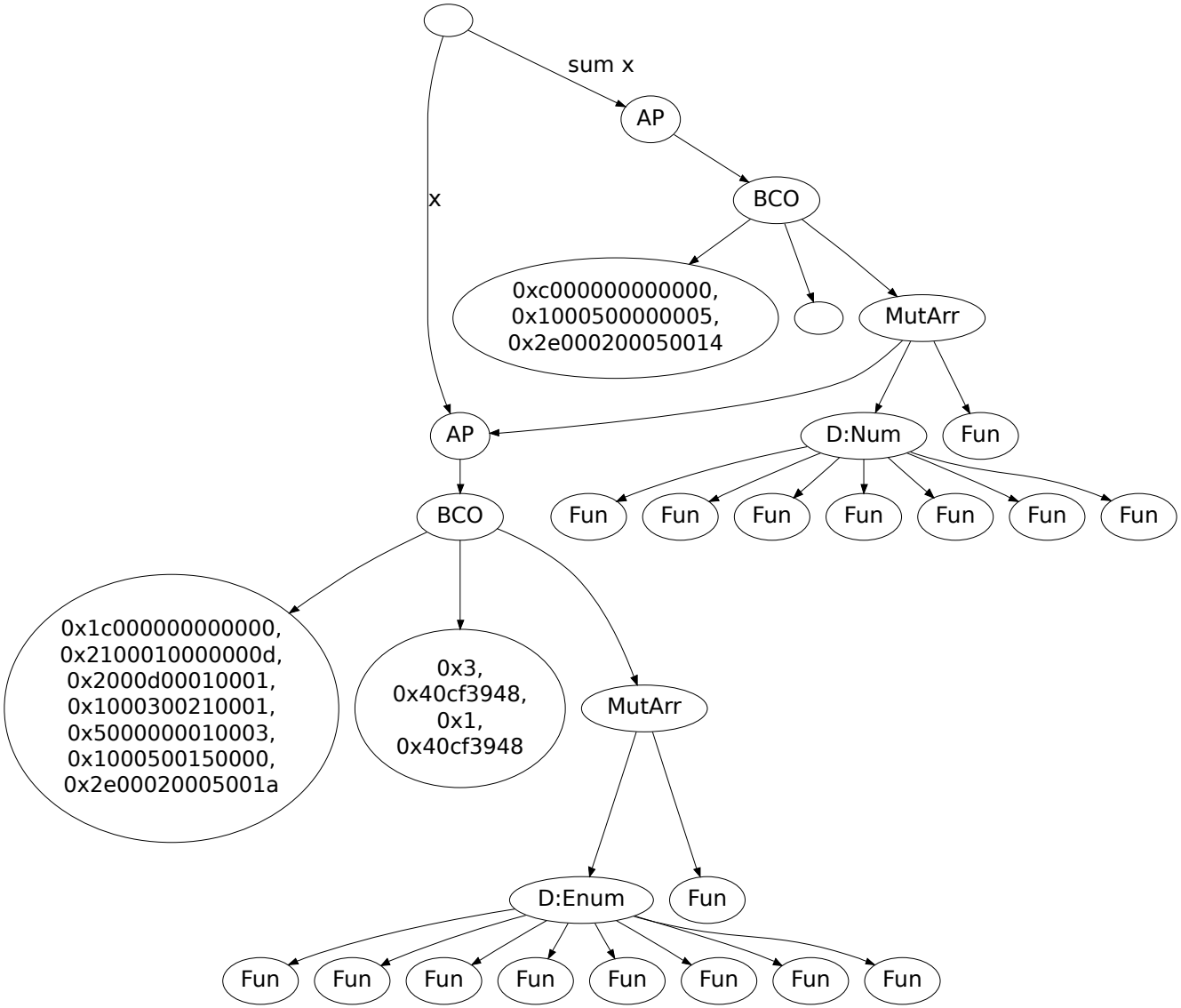
Ausschnitt aus derselben Heap-Map nach Garbage Collection:

```
[...
,(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

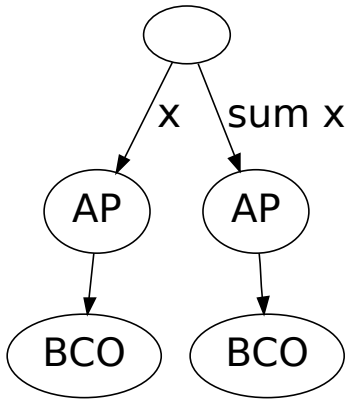
Implementierung: Byte Code Objects

Vollständige BCO-Informationen:



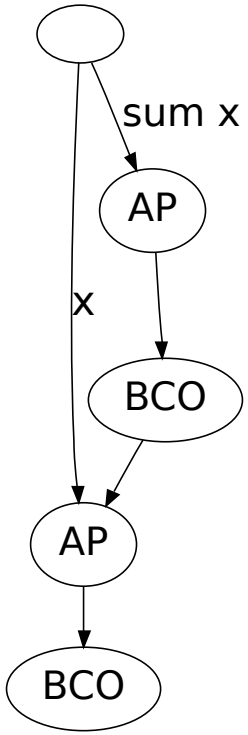
Implementierung: Byte Code Objects

Keine BCO-Informationen:



Implementierung: Byte Code Objects

⇒ Relevante BCO-Informationen auswählen:



Evaluierung

Vergleich: :print

- :print Bestandteil von GHCi
- Kein Sharing
  $\Rightarrow$ Keine unendlichen Datenstrukturen

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

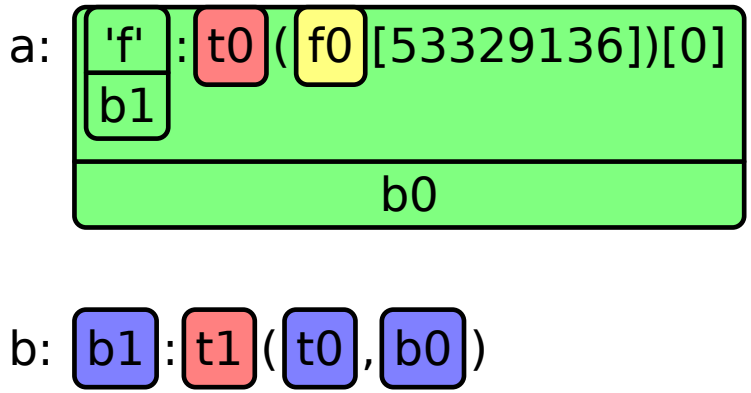
```
ghci> :print a
a = 'f' : (_t1::[Char])
ghci> :print b
b = 'f' : (_t2::[Char])
```

Vergleich: :print

- :print Bestandteil von GHCi
- Kein Sharing
  $\Rightarrow$ Keine unendlichen Datenstrukturen

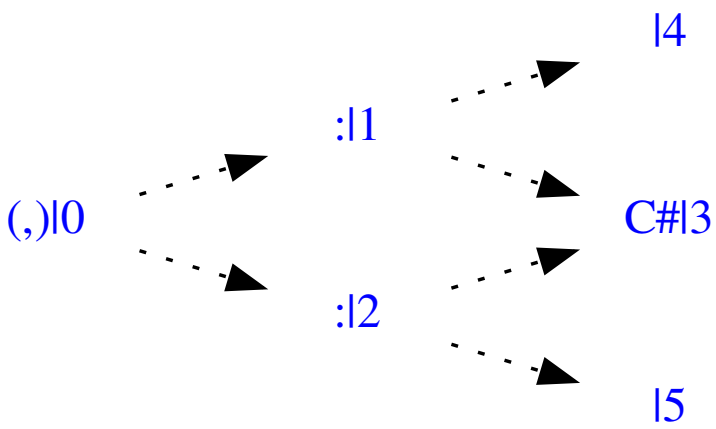
```
ghci> let a = "foo"
ghci> let b = a ++ a
ghci> head b
'f'
```

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



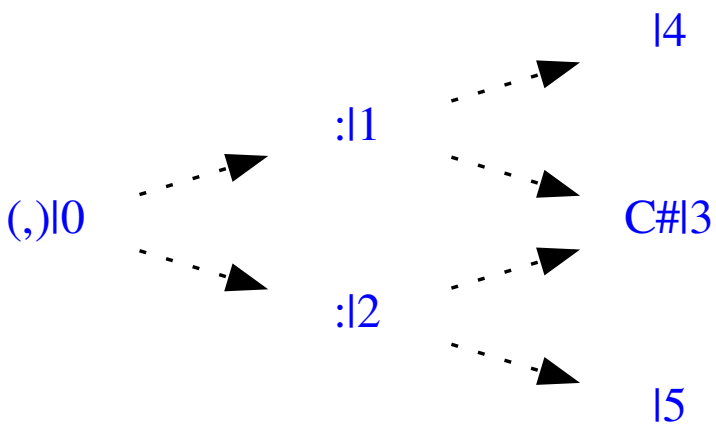
Vergleich: vacuum

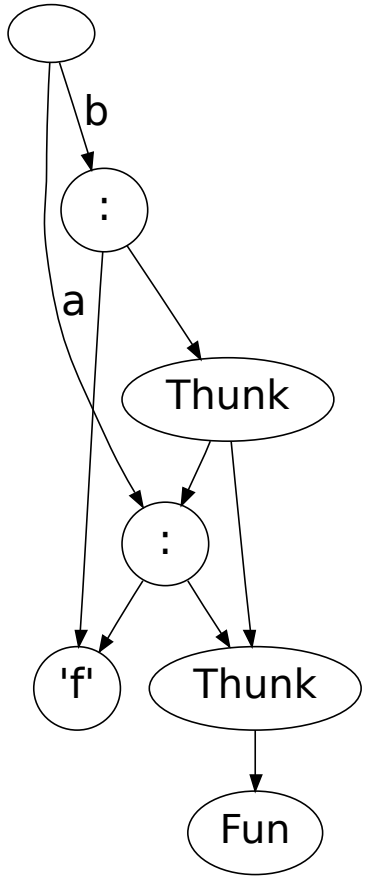
- Kein Sharing hinter unausgewerteten Ausdrücken
- Nicht interaktiv



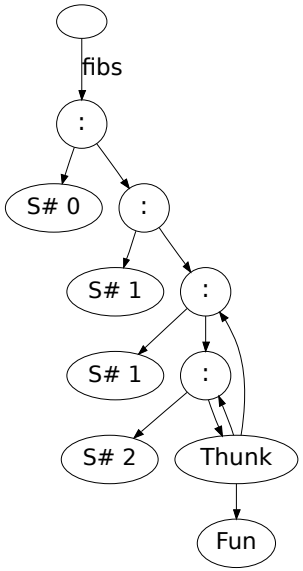
Vergleich: vacuum

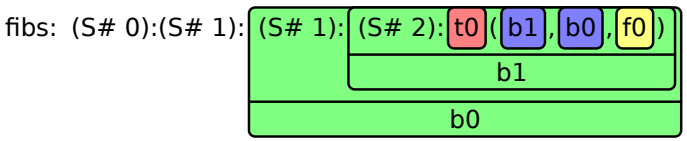
- Kein Sharing hinter unausgewerteten Ausdrücken
- Nicht interaktiv

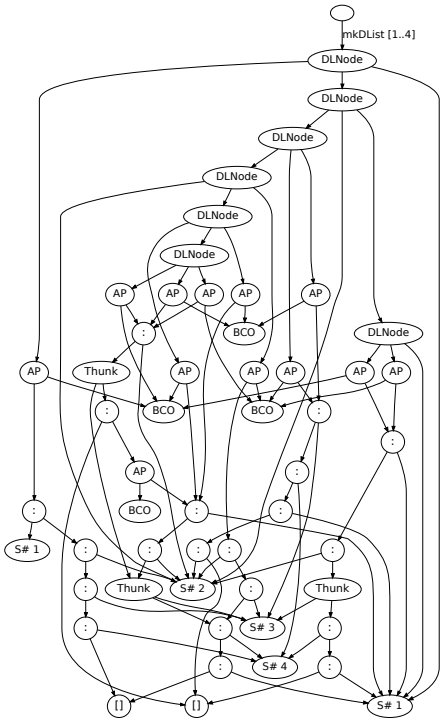


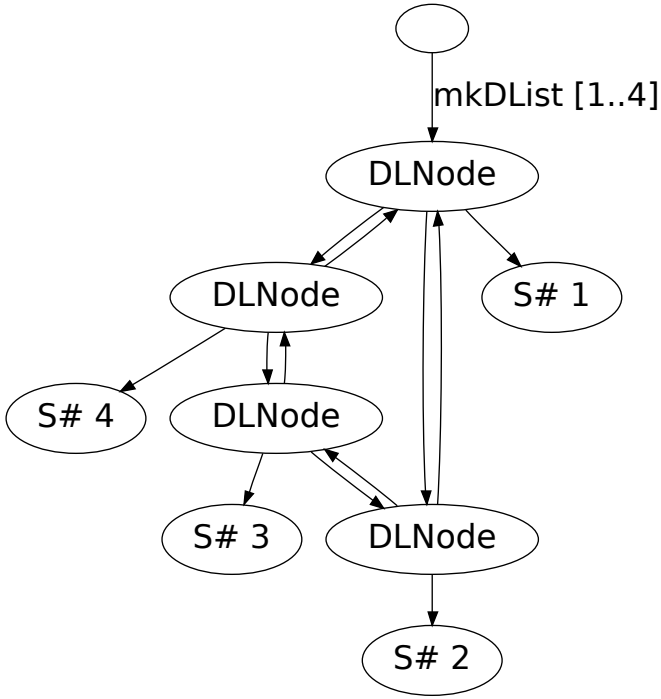


Demonstration









Abschluss

- 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

GHC Heap

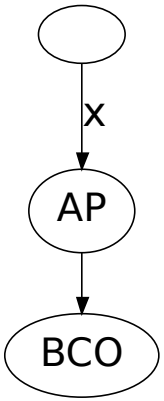
- Two-Space Stop-and-Copy Collector
- Alle benötigten Closures aus einer Hälfte des Heaps in andere kopieren
- Pointer in Closures verfolgen um weitere Closures zu finden

GHC Heap: Auswertung

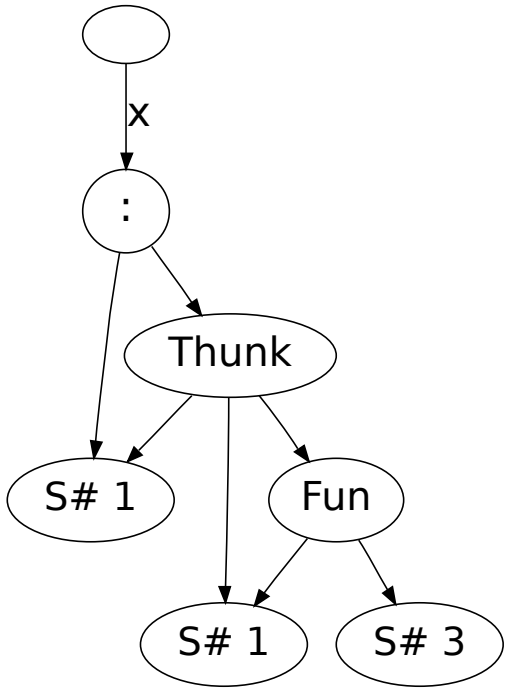
- Closure wird durch Weak-Head Normal Form ersetzt

$$x = [1..3]$$

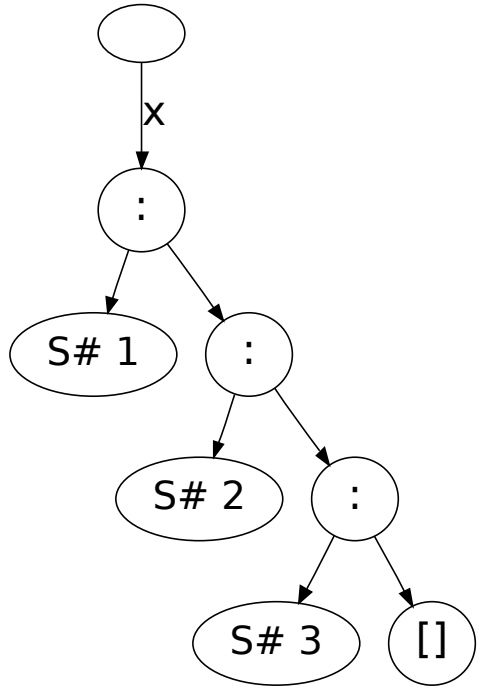
Unausgewertet



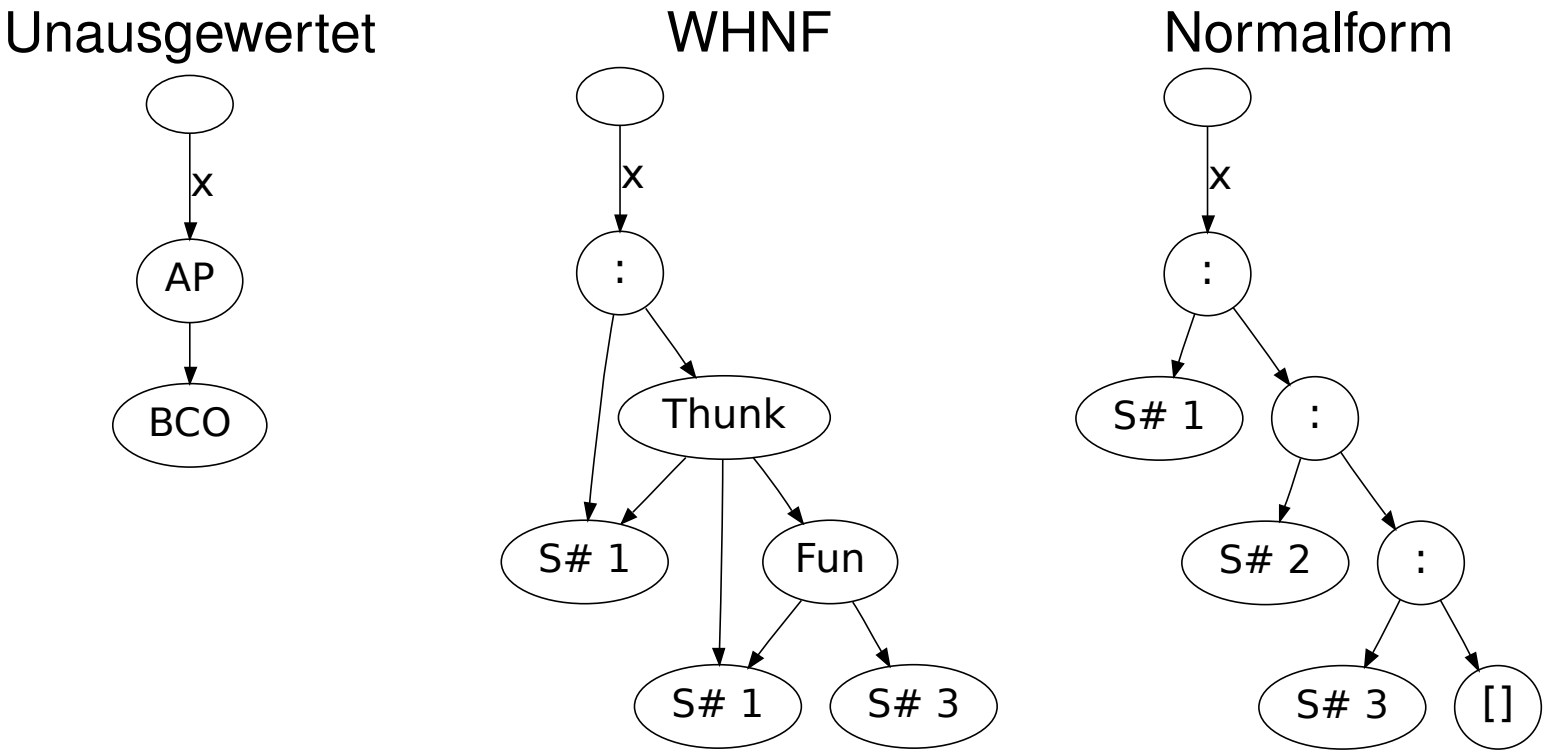
WHNF



Normalform

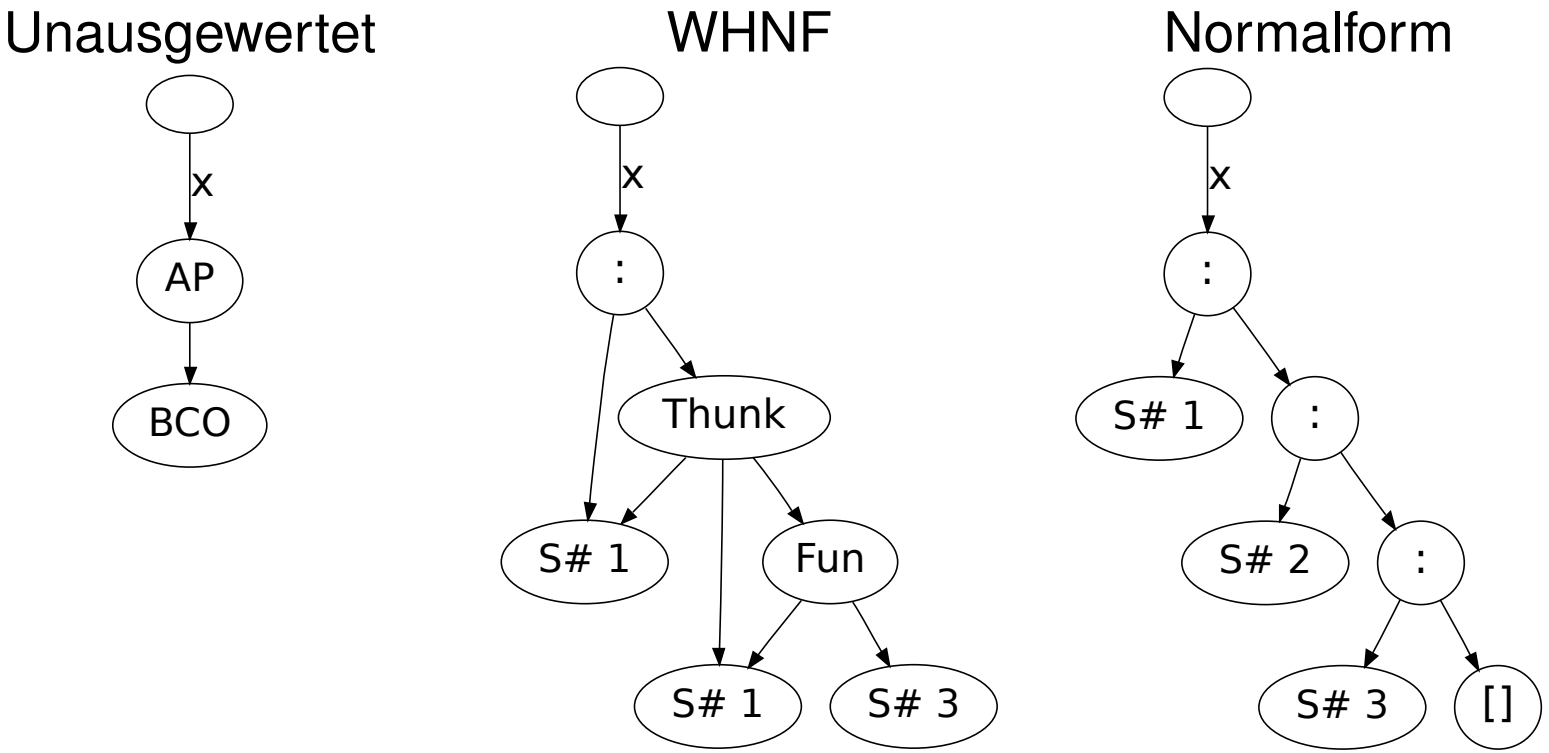


GHC Heap: Auswertung



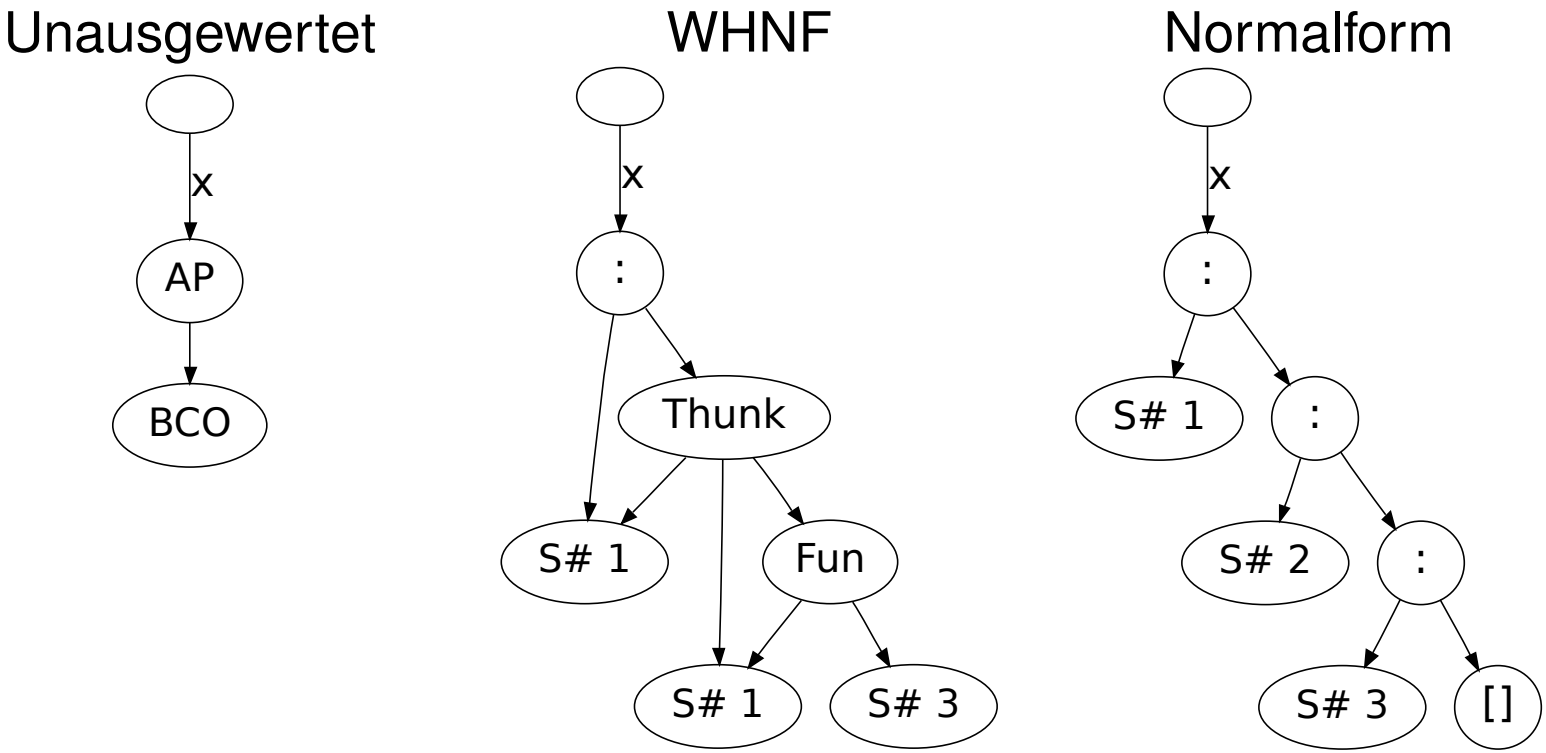
- Data Constructor
- Function Closure und Partial Application
- Thunk und General Application

GHC Heap: Auswertung



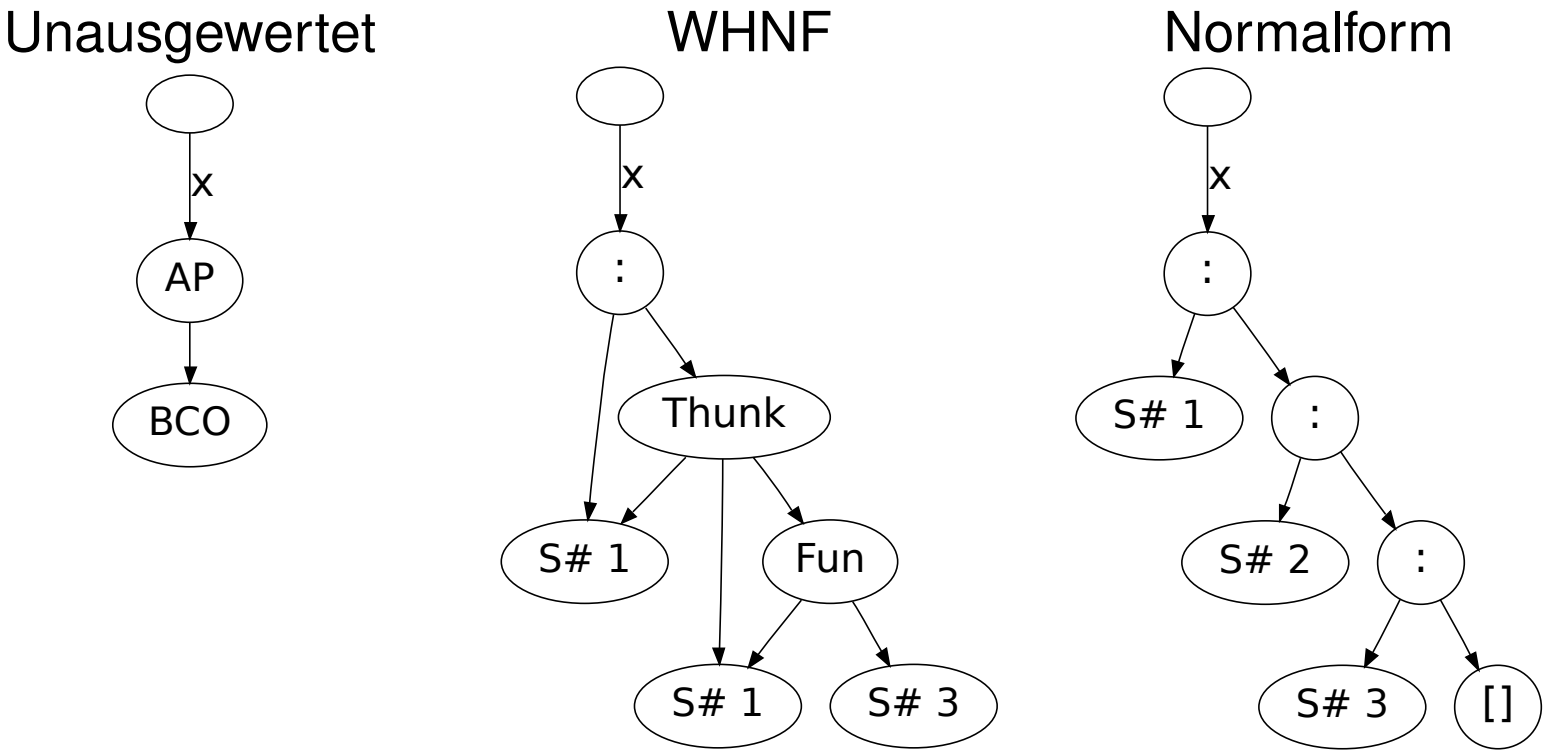
- Data Constructor $\Rightarrow$ WHNF
- Function Closure und Partial Application
- Thunk und General Application

GHC Heap: Auswertung



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

GHC Heap: Auswertung



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