# Typinferenz in Haskell

Inferenzregel:

$$\frac{f :: A \to B \qquad e :: A}{(f\ e) :: B}$$

# Typinferenz in Haskell

Inferenzregel:

$$\frac{f :: A \to B \qquad e :: A}{(f\ e) :: B}$$

Man bemerke die Analogie zur logischen Inferenzregel

# Typinferenz in Haskell

Inferenzregel:

$$\frac{f :: A \to B \qquad e :: A}{(f\ e) :: B}$$

Man bemerke die Analogie zur logischen Inferenzregel

Modus Ponens: $\dfrac{A \to B \quad A}{B}$

# Typinferenz in Haskell

Inferenzregel:

$$\frac{f :: A \to B \qquad e :: A}{(f\ e) :: B}$$

Man bemerke die Analogie zur logischen Inferenzregel

Modus Ponens: $\dfrac{A \to B \quad A}{B}$

Beispiel:

```
even      :: Int -> Bool
3         :: Int
(even 3)  :: ?
```

$$\frac{\overbrace{even}^{f} :: \overbrace{Int}^{A} \to \overbrace{Bool}^{B} \qquad \overbrace{3}^{e} :: \overbrace{Int}^{A}}{\underbrace{(even\ 3)}_{(fe)} :: \underbrace{Bool}_{B}}$$

# Typinferenz für polymorphe Typen

$$\frac{f :: A \to B \quad e :: A'}{(f\ e) :: ?}$$

# Typinferenz für polymorphe Typen

$$\frac{f :: A \to B \quad e :: A'}{(f\ e) ::?}$$

Beispiel:

```
len               :: [a] -> Int
[1 .. 10]         :: [Int]
(len [1 .. 10]) :: ?
```

$$\frac{\overbrace{len}^{f} :: \overbrace{[a]}^{A} \to \overbrace{Int}^{B} \qquad \overbrace{[1..10]}^{e} :: \overbrace{[Int]}^{A'}}{\underbrace{len[1..10]}_{(f\ e)} :: \underbrace{Int}_{B'}}$$

für $\sigma = \{a \mapsto \texttt{Int}\}$: $\sigma(A) = \sigma([a]) = [Int] = \sigma(A')$, $\sigma(B) = Int = \sigma(B')$

# Typinferenz für polymorphe Typen

$$\frac{f :: A \to B \quad e :: A'}{(f\ e) ::?}$$

Beispiel:

```
len              :: [a] -> Int
[1 .. 10]        :: [Int]
(len [1 .. 10]) :: ?
```

$$\frac{\overbrace{len}^{f} :: \overbrace{[a]}^{A} \to \overbrace{Int}^{B} \qquad \overbrace{[1..10]}^{e} :: \overbrace{[Int]}^{A'}}{\underbrace{len[1..10]}_{(f\ e)} :: \underbrace{Int}_{B'}}$$

für $\sigma = \{a \mapsto \texttt{Int}\}$: $\sigma(A) = \sigma([a]) = [Int] = \sigma(A')$, $\sigma(B) = Int = \sigma(B')$
allgemeine Inferenzregel:
für allgemeinstem Unifikator $\sigma$ der Typausdrücke (Terme) $A$ und $A'$
(Substitution mit $\sigma(A) = \sigma(A')$)

$$\frac{f :: \sigma(A) \to \sigma(B) \quad e :: \sigma(A')}{(f\ e) :: \sigma(B)}$$

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::  ?`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` `?`
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` ?
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
    `twice :: (Int -> Int) -> Int -> Int`
    also `(twice (\ x -> 3*x) )::`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` ?
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
    `twice :: (Int -> Int) -> Int -> Int`
    also `(twice (\ x -> 3*x) )::Int -> Int`
  - Pos 00: `( (.) double ) ::` ??

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` `?`
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
    `twice :: (Int -> Int) -> Int -> Int`
    also `(twice (\ x -> 3*x) )::` `Int -> Int`
  - Pos 00: `( (.) double ) :: ??`
    zu `double :: Int -> Int` passende Instanz von `(.)` mit

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` `?`
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
    `twice :: (Int -> Int) -> Int -> Int`
    also `(twice (\ x -> 3*x) )::` `Int -> Int`
  - Pos 00: `( (.) double ) ::` `??`
    zu `double :: Int -> Int` passende Instanz von `(.)` mit
    $\sigma(\mathsf{b}) = \sigma(\mathsf{c}) = \mathtt{Int}$:
    `(.) :: (Int -> Int) -> (a -> Int) -> a -> Int`
    also `( (.) double ) ::`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der

- Blätter:
  Pos 000: `(.) :: (b -> c) -> (a -> b) -> a -> c`
  Pos 001: `double :: Int -> Int`
  Pos 010: `twice :: (a -> a) -> a -> a`
  Pos 011: `(\ x -> 3*x) :: Int -> Int`
  Pos 1: `2 :: Int`
- inneren Knoten:
  - Pos 01: `(twice ( \ x -> 3 * x ))::` `?`
    zu `(\ x -> 3*x) :: Int -> Int`
    passende Instanz von `twice` mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
    `twice :: (Int -> Int) -> Int -> Int`
    also `(twice (\ x -> 3*x) )::` `Int -> Int`
  - Pos 00: `( (.) double ) :: ??`
    zu `double :: Int -> Int` passende Instanz von `(.)` mit
    $\sigma(\mathsf{b}) = \sigma(\mathsf{c}) = \mathtt{Int}$:
    `(.) :: (Int -> Int) -> (a -> Int) -> a -> Int`
    also `( (.) double ) ::` `(a -> Int) -> a -> Int`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`
  zu `twice (\ x -> 3*x):: Int -> Int`
  passende Instanz von
  `( (.) double ) :: (a -> Int) -> a -> Int`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`
  zu `twice (\ x -> 3*x):: Int -> Int`
  passende Instanz von
  `( (.) double ) :: (a -> Int) -> a -> Int`
  mit $\sigma(\mathtt{a}) = \mathtt{Int}$:
  `( (.) double ) :: (Int -> Int) -> Int -> Int`
  also `((.) double (twice (\ x -> 3*x))) ::`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`
  zu `twice (\ x -> 3*x):: Int -> Int`
  passende Instanz von
  `( (.) double ) :: (a -> Int) -> a -> Int`
  mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
  `( (.) double ) :: (Int -> Int) -> Int -> Int`
  also `((.) double (twice (\ x -> 3*x))) :: Int -> Int`
- Pos $\varepsilon$ (Wurzel):
  `(((.) double (twice (\ x -> 3*x))) 2) :: ?`

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`
  zu `twice (\ x -> 3*x):: Int -> Int`
  passende Instanz von
  `( (.) double ) :: (a -> Int) -> a -> Int`
  mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
  `( (.) double ) :: (Int -> Int) -> Int -> Int`
  also `((.) double (twice (\ x -> 3*x))) :: Int -> Int`
- Pos $\varepsilon$ (Wurzel):
  `(((.) double (twice (\ x -> 3*x))) 2) :: ?`
  `((.) double (twice (\ x -> 3*x))) :: Int -> Int`
  und `2 :: Int` ergibt

# `(double . (twice ( \ x -> 3*x))) 2 ) :: ?`

Typen der inneren Knoten

- Pos 0: `((.) double (twice (\ x -> 3*x))) :: ?`
  zu `twice (\ x -> 3*x):: Int -> Int`
  passende Instanz von
  `( (.) double ) :: (a -> Int) -> a -> Int`
  mit $\sigma(\mathsf{a}) = \mathtt{Int}$:
  `( (.) double ) :: (Int -> Int) -> Int -> Int`
  also `((.) double (twice (\ x -> 3*x))) :: Int -> Int`
- Pos $\varepsilon$ (Wurzel):
  `(((.) double (twice (\ x -> 3*x))) 2) :: ?`
  `((.) double (twice (\ x -> 3*x))) :: Int -> Int`
  und `2 :: Int` ergibt
  `(((.) double (twice (\ x -> 3*x))) 2 ) :: Int`

also `(( double . (twice (\ x -> 3*x))) 2 ) :: Int`

# `(twice twice succ 0) ::  ?`

Typen der

- Blätter:
  Pos 000 und 001: `twice :: (a -> a) -> a -> a`
  Pos 01: `succ :: Int -> Int`
  Pos 1: `0 :: Int`
- inneren Knoten:
  - Pos 00: `twice twice :: ?`
    Pos 001: `twice :: (a -> a) -> (a -> a)`
    passende Instanz des `twice` an Pos 000 mit $\sigma(a) = (a \to a)$ :

# `(twice twice succ 0) :: ?`

Typen der

- Blätter:
  Pos 000 und 001: `twice :: (a -> a) -> a -> a`
  Pos 01: `succ :: Int -> Int`
  Pos 1: `0 :: Int`
- inneren Knoten:
  - Pos 00: `twice twice :: ?`
    Pos 001: `twice :: (a -> a) -> (a -> a)`
    passende Instanz des `twice` an Pos 000 mit $\sigma(a) = (a \to a)$ :
    `twice :: ((a -> a) -> (a -> a)) -> (a -> a) -> (a -> a)`
    also `twice twice :: (a -> a) -> (a -> a)`
  - Pos 0: `twice twice succ :: ?`
    ...
  - Pos $\varepsilon$ (Wurzel): `twice twice succ 0 :: ?`
    ...