Typfamilien

Funktionen auf Typen

Christian Höner zu Siederdissen
christian.hoener.zu.siederdissen@uni-jena.de

Theoretische Bioinformatik, Bioinformatik Uni Jena

Dezember 2023

Was sind Typfamilien

- Typfamilien erlauben es uns Familien von Typen zu definieren und diese mit mittels TypLevel-Funktionen zu assoziieren.
- Während Typklassen Operationen auf Werten definieren, werden hier Beziehungen zwischen Typen definiert.
- Damit haben wir Typlevel Polymorphismus.

Typfamilien

```haskell
type family List a :: *

type instance List () = Int

type instance List Int = [Int]
```

Typklassen für Funktionen auf Typfamilien

```
1  class ListFuns a where
2    lfcons :: a -> List a -> List a
3    lfuncons :: List a -> (a, List a)
4    lfnull :: a -> List a
5
6  instance ListFuns () where
7    lfcons _ k = k+1
8    lfuncons k | k > 0 = ((), k-1)
9    lfnull _ = 0
10
11 instance ListFuns Int where
12   lfcons a as = a : as
13   lfuncons (a:as) = (a,as)
14   lfnull _ = []

-- (1::Int)`lfcons` lfnull (undefined :: Int)== [1]
```

Probleme mit Typen

```
1   1 `lfcons` lfnull (undefined :: Int)
2
3  <interactive>:15:1: error:
4      - Could not deduce: List a0 ~ [Int]
5        arising from a type ambiguity check for
6        the inferred type for 'it'
7      from the context: (List a ~ [Int], ListFuns a, Num
           a)
8        bound by the inferred type for 'it':
9                   forall {a}. (List a ~ [Int], ListFuns
                        a, Num a) => List a
10       at <interactive>:15:1-36
11     The type variable 'a0' is ambiguous
```

Assoziierte Typfamilien

```
 1  class TFList a where
 2    type TFL a :: *
 3    tfcons :: a -> TFL a -> TFL a
 4    tfuncons :: TFL a -> (a, TFL a)
 5    tfnull :: a -> TFL a
 6
 7  instance TFList Int where
 8    type TFL Int = [Int]
 9    tfcons a as = a:as
10    tfuncons (a:as) = (a,as)
11    tfnull _ = []
```

Assoziierte Datenfamilien

```
 1  class DFList a where
 2    data DFL a :: *
 3    dfcons :: a -> DFL a -> DFL a
 4    dfuncons :: DFL a -> (a, DFL a)
 5    dfnull :: DFL a
 6
 7  instance DFList Int where
 8    newtype DFL Int = DFInt [Int]
 9    dfcons a (DFInt as) = DFInt (a:as)
10    dfuncons (DFInt (a:as)) = (a, DFInt as)
11    dfnull = DFInt []

-- (1 :: Int)`dfcons` dfnull == DFInt [1]
```