1 Forelesning 14

1.1 Kinds

Kinds er en slags "typer for typer of typekonstruksjoner"

- Vanlige typer er av king *.
- Maybe of Set har typeargument: `kind * -> *`
- Map har to typeargument (key og value): `* -> * -> *`

Det finnes også høyere ordens kinds, f. eks

```
data Fix f = Fix (f (Fix f))
```

1.2 Typeklasser

1.2.1 Hva er en typeklasse

```
class (Bar t) => Foo t where
    foo :: t -> Maybe t
    xyzzy :: t -> Integer
    bar :: [t] -> String -> t

-- Law for Foo class:
-- For all ts and str:
-- xyzzy (bar ts str) == fromIntegral length ts
```

En typeklasse har:

- Et navn (Foo)
- Instansvariabler (t)
- Forkrav (Bar t)
- Spesifikasjon av funksjoner
- Uformelt: Lover som alle instanser bør oppfylle

1.2.2 map for Maybe?

```
addThreeMaybe :: (Num a) => Maybe a -> Maybe a
addThreeMaybe Nothing = Nothing
addThreeMaybe (Just a) = Just (a+3)
```

Skriv om `addThreeMaybe` til å bruke `maybe` funksjonen istedetfor pattern matching:

```
maybe :: b -> (a -> b) -> Maybe a -> b
```

Hva skal typene `a` og `b` være for `addThreeMaybe`?

```
addThreeMaybe = maybe Nothing (\a -> Just(a + 3))
```

1.2.3 Functor

```
class Functor f where
    fmap :: (a -> b) -> f a -> f b
```

Merk at `f` har kind `* -> *`

1.2.4 Funktorlovene

Instanser av `Functor` klassen forventes å oppfylle disse to lovene:

```
fmap (f . g) == fmap f . fmap g
fmap id == id
```

Disse følger fra intuisjonen fra tidligere. La oss sjekke at de holder hvis vi implementerer `fmap` for `Maybe`!

Eksempel. Either

◇