

# Algebraiske datatyper med fokus på algebra

- Typeteori del 2
  - Nå : tall istedenfor bevis
- Hvorfor heter det
  - *Algebraiske* datatyper?
  - Sumtyper?
  - Produkttyper?
- Hvordan kan vi bruke matte til å forenkle datatyper?
- Kode i Haskell og kanskje noen nye ord? :))

# Typar og datatyper

- Datatyper : typer som er bygget opp av algebraiske datatyper
- `a` : typeparameter (som `<A>` i Java osv)

```
data MinDatatype a = Konstruktor a
```

```
v :: MinDatatype Int  
v = Konstruktor 5
```

```
-- Unit i Kotlin  
data () = ()
```

# Sum-typer

- Enten det ene eller det andre

```
data Bool = False | True
```

```
data Dag = Mandag | Tirsdag | Onsdag | Torsdag | Fredag | Loerdag | Soendag
```

- Antall unike verdier : summere unike verdier til hver konstruktør
  - Kalles for kardinaliteten
- Hva er kardinaliteten til Bool og Dag?

# Produkt-typer

- Kombinerer flere verdier til en sammensatt verdi
  - Det ene OG det andre

```
data BoolPair = BoolPair Bool Bool
--alle muligheter
b1 = BoolPair False False
b2 = BoolPair False True
b3 = BoolPair True False
b4 = BoolPair True True
```

- Antall unike verdier : multiplisere unike verdier til hvert felt
- Kardinalitet : 4

# Når er to typer like?

- Hva vil det si at noe er likt?
- Mange forskjellige varianter av likhet
- Her : to typer er like hvis og bare hvis de har like mange unike verdier
  - Samme kardinalitet
  - Isomorfe

```
data Bool = True | False
```

```
data HeiHopp = Hei | Hopp
```

```
data Melk = Lett | Hel | Skummet
```

# Isomorfe

- To typer er isomorfe hvis det finnes en isomorfisme 🙄
- Samme struktur
- Mappe fram og tilbake med to funksjoner
  - en-til-en-korrespondanse
- Isomorfe hvis og bare hvis de har like mange unike verdier

```
a ~ = b
toR :: a -> b
toL :: b -> a

-- fram og tilbake skal ende opp med samme verdi
toL (toR a) == a
toR (toL b) == b
```

# Isomorfe eksempler

```
data Bool = False | True
data HeiHopp = Hei | Hopp
```

```
Bool ~ = HeiHopp
```

```
toR False = Hei
toR True = Hopp
toL Hei = False
toL Hopp = True
```

```
(Int, Bool) ~ = (Bool, Int)
```

```
toR (i,b) = (b,i)
toL (b,i) = (i,b)
```

- Ikke isomorfe: Bool og (Bool, Bool)

# Standard generisk produkttype : (a,b)

- Tuppel
- Representerer \*
- (Bool,Bool) :  $2*2 = 4$

```
data (a,b) = (a,b)
```

```
(a,b) ~= (b,a) --kommutativ
```

```
toR = toL = \(x,y) -> (y,x) -- swap
```

```
(a,(b,c)) ~= ((a,b),c) -- assosiativ
```

```
toR (a,(b,c)) = ((a,b),c)
```

```
toL ((a,b),c) = (a,(b,c))
```



## \* Standard generisk sumtype - Either

- Representerer +
- `Either Bool () : 2+1 = 3`

```
data Either a b = Left a | Right b

Either a b ~ = Either b a --kommutativ
switchEither (Left x) = Right x
switchEither (Right x) = Left x
toR = toL = switchEither
```

```
(Either a (Either b c)) ~ = (Either (Either a b) c) -- assosiativ
-- ta den i hodet 🙄
```

```
Either () a ~ = Maybe a -- 1 + a
```

# Identitetselementer - 0

- Har vi en type med kardinalitet 0? Ja!

```
-- Ingen konstruktører! Ingen verdier!
```

```
data Void
```

```
absurd :: Void -> a
```

```
absurd v = case v of {}
```

```
Either Void a ~ = a
```

```
toR (Left v) = absurd v
```

```
toR (Right a) = a
```

```
toL a = Right a
```

```
-- samme andre veien
```

```
Either a Void ~ = a
```

- `Void` er identitetselementet til `Either`  
på samme måte 0 er det til +

# Identitetselementer - 1

```
data () = ()
```

- Nøyaktig en verdi

- 

$$a * 1 = 1$$

```
(a, ()) ~ = a
```

```
toR (a, ()) = a
```

```
toL a = (a, ())
```

```
-- samme andre veien
```

```
(a, ()) ~ = a
```

- `()` er identitetselementet til `(,)` (Tuple)  
på samme måte 1 er det til `*`

# Semiring? Hva mangler?

En semiring er en algebraisk struktur  $(+, *, 0, 1)$  som oppfyller følgende

- $+$  : Addisjon
  - Assosiativ :  $(a+b) + c = a + (b+c)$
  - Identitetselement :  $a+0 = 0 = 0+a$
  - Kommutativ :  $a+b = b+a$
- $*$  : Multiplikasjon
  - Assosiativ :  $(a+b) + c = a + (b+c)$
  - Identitetselement :  $a * 1 = a = 1 * a$
  - Absorberingselement :  $a * 0 = 0 = 0 * a$
- Distribusjon:
  - $a * (b + c) = (a * b) + (a * c)$
  - $(a + b) * c = (a * c) + (b * c)$

# One semiring to bind them

- Absorberingselement :  $a * 0 = 0 = 0 * a$

```
(Void, a) ~ = Void
```

```
toL (v, _) = v  
toR v = absurd v
```

- Distribusjon :  $a * (b + c) = (a * b) + (a * c)$

```
(a, Either b c) ~ = Either (a, b) (a, c)
```

```
toL (a, Left b) = Left (a, b)  
toL (a, Right c) = Right (a, c)  
toR (Left (a, b)) = (a, Left b)  
toR (Right (a, c)) = (a, Right c)
```

(Den andre distribusjonen er veldig lik)

# Hva med funksjoner?

- Rene funksjoner er mapping fra input til output
- Hva blir kardinaliteten for

```
() -> Bool  
Bool -> ()  
Bool -> Bool  
Maybe Bool -> Bool
```

- Altså hvor mange forskjellige mappings fra  
a til b kan man lage

En funksjon `a -> b` har kardinalitet

$$b^a$$

```
() -> Bool -- 2
Bool -> () -- 1
Bool -> Bool -- 4
Maybe Bool -> Bool -- 8
```

# Potensregler

$$a^b * a^c = a^{b+c}$$

$(b \rightarrow a, c \rightarrow a) \sim \text{Either } b \text{ } c \rightarrow a$

$$a^0 = 1$$

$\text{Void} \rightarrow a \sim ()$

$\text{toR } \_ = ()$

$\text{toL } () = \backslash v \rightarrow \text{absurd } v$



# Potensregler - har sammenheng med currying?

- Currya- funksjoner og tupla-funksjoner er like kraftfulle
- Ligger i standardlib :

```
curry    :: ((c,b)  -> a) -> (c -> (b -> a))
curry cb2a = \c b -> cb2a (c,b)
uncurry  :: (c -> b -> a) -> ((c,b)  -> a)
uncurry cba = \(c,b) -> cba c b
```

# Potensregler - har sammenheng med currying? 🤯

$$a^{b*c} = (a^b)^c$$

```
(b,c) -> a ~ = c -> (b -> a)
-- siden (,) er kommutativ
(c,b) -> a ~ = c -> (b -> a)
```


```
toR = curry
toL = uncurry
```

# Forenkle datatyper 1 : sponset av Tine™

```
data Melk = Hel | Lett | Skummet
-- Vi vil forenkle :
(Melk, Melk, Bool -> Melk)
--
3*3 * 3^2 = 3^3
--
Melk -> Melk
☕ -> ☕
🥛 -> 🥛

-- eller
(Melk, Melk, Melk, Melk)
(🥛, 🥛, 🥛, 🥛)
```

## Forenkle datatyper 2

- Forenkle en hårete  type :  $(b \rightarrow x \rightarrow a, c \rightarrow x \rightarrow a)$

```
--(a^x)^b * (a^x)^c
(b -> x -> a, c -> x -> a)
-- currying , a^(b*c) = (a^b)^c
((b, x) -> a, (c, x) -> a)
-- potensregel : a^b * a^c = a^(b*c)
Either (b,x) (c,x) -> a
-- Distribusjon -- a*b + a*c = c * (a+b)
(Either b c, x) -> a
-- a^((b+c) * x)
```

- til en koselig  $(\text{Either } b \text{ } c, x) \rightarrow a$

# Konklusjon og litt ekstra snacks 🍭

- Bedre forståelse av algebraiske datatyper og terminologi
- Fascinerende sammenheng mellom aritmetikk og algebraiske datatyper
- Forenkle og manipulere

## Finnes selvfølgelig mer :))

- Rekursive typer

```
data List a = Null | Cons a (List a)
gir L(x)    = 1 + x * L(x)
```

- Derivering og Taylor series
- Hva med minus og deling? 🤔👨‍🍳