Algebraiske datatyper med fokus på algebra

- Hvorfor heter det
  - *Algebraiske* datatyper?
  - Sumtyper?
  - Produkttyper?
- Hvordan kan vi bruke matte til å forenkle datatyper?

Typer og datatyper

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

```
data MinDatatype a = Konstruktor a

v :: Datatype Int
v = Konstruktor 5
```

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

Produkt-typer

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

```
data BoolPair = Coord Bool Bool

data Person = Person String Int Bool
```

- Antall unike verdier : multiplisere unike verdier til type

Når er to typer like?

- Hva vil det si at at noe er likt?
- Mange forskjellige varianter av likhet
- Her : toR typer er like hviss de har like mange unike verdier
    - Samme kardinalitet (kardinalitet)

```
data Bool = True | False

data HeiHopp = Hei | Hopp

data Melk = Lett | Hel | Skummet
```

Isomorfe

- Samme struktur
- Mappe fram og tilbake med to funksjon
    - en-til-en-korrespondanse
- Isomorfe hviss 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 Melk : 2+3 : 5

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

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

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

Maybe

- Hva blir da Maybe?

```
data Maybe a = Nothing | Just a
```

- Det blir + 1

```
1+1        = 2
Maybe () ~= Bool


Maybe a  :  a + 1

Either () a ~= Maybe a
```

Identitetselementer - 0

- Har vi en type med kardinalitet 0? Ja!

```haskell
-- Ingen konstruktører! Ingen verdier!
data Void

absurd :: Void -> a
abusrd v = case v of {}
```

```haskell
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?

En semiring er en algebraisk struktur 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)

Hva mangler vi?

Semiring!

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

```
(Void,a) ~= Void

toL (v,_) = a
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 distribusjon er ca helt lik)

Funksjoner

- Hva med funksjoner?
- Hva blir kardinaliteten for

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

En funksjon `a -> b` har kardinalitet

$$b^a$$

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

Potensregler

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

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

toR cb2a = \c b -> cb2a (c,b)
toL c2b2a = \(c,b) -> c2b2a c b
```

- Lligger i standardlib :

```
curry   :: ((c,b)  -> a) -> (c -> b -> a)
uncurry :: (c -> b -> a) -> ((c,b)  -> a)
```

Potensregler

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

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

$$a^0 = 1$$

```
Void -> a ~= ()

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

Forenkle datatyper

Konklusjon

- Fascinerende sammenheng mellom aritmetikk og algebraiske datatyper
- Forenkle og manipulere
- Hva med minus og deling? 🤔🤯