ALGEBRAIC DATA TYPES (ADT)

Malte Neuss

ALGEBRAIC DATA TYPES

Numbers:

```
2 + 2 + 2 =  3 * 2
```

Types:

```
interface User = {
  isLoggedIn: Boolean
  email:      string
}
```

KINDS OF TYPES

- Product Type*
- Sum Type*
- Exponential Type*
- Recursive Type
- Linear Type
- Dependent Type ...

Make illegal state unrepresentable

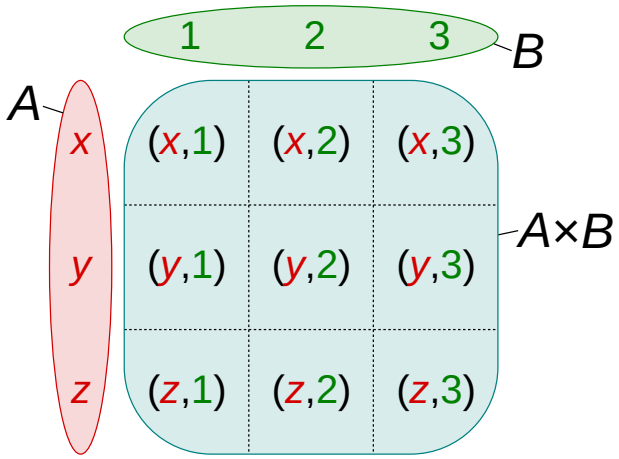
BASIC TYPES

```
type String = "" | "a" | "b".. // infty
type Int    = ..-1 | 0 | 1..   // 2^32
..
type Boolean = true | false // 2
type Unit    = unit         // 1
type Void                   // 0
```

PRODUCT TYPE

Ideally:

```
type ProductType = Factor x Factor
```



EXAMPLE 1

```
interface ProductType {
  factor1: Boolean; // 2
  factor2: Boolean; // 2
}
```

```
Boolean x Boolean
= (true,true) (false,false) (true,false) (false,true)
```

EXAMPLE 2

```
interface ProductType {
  factor1: Boolean; // 2
  factor2: Unit;    // 1
}
```

```
Boolean x Unit = (true,unit) (false,unit)    ~ Boolean
```

EXAMPLE 3

```
interface ProductType {
  factor1: Boolean; // 2
  factor2: Void;    // 0
}
```

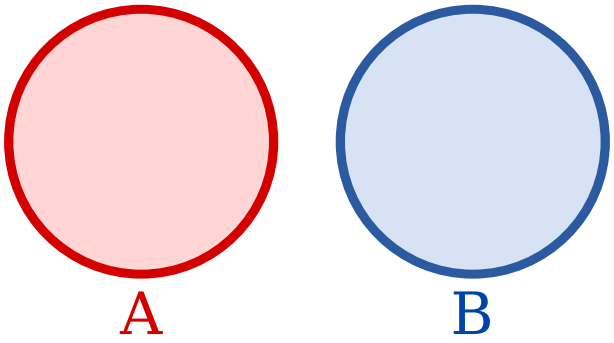
```
Boolean x Void = (true,?)                    ~ Void
```

SUM TYPE

Ideally:

```
type SumType = Summand + Summand
```

also Choice Type, Tagged Union, Discriminated Union



EXAMPLE 1

```
type SumType = Boolean | Unit
//     3            2      1
```

```
SumType = true false unit
```

```
type SumType = BoolTag Boolean | UnitTag Unit
```

EXAMPLE 2

```
type BoolOpt  = Boolean | Unit
```

```
type BoolOpt  = Boolean | None
```

```
type Option<T> = Some(T) | None
```

EXAMPLE 3

```
type SumType = Boolean | Void
//     2            2       0
```

```
SumType = true false                                    ~ Boolean
```

EXPONENTIAL TYPE

“Ideally:”

```
type ExpoType = Base^Exponent
```

EXAMPLE

```
//                      Exponent          Base
type Exponential =     Trilean   =>    Boolean
//     2^3                 3               2
//                       t1,t2,t3       true,false
```

TYPESCRIPT CHOICE TYPES

REAL EXAMPLE

TYPE ALIAS

```
type Euro = number;

function toNumber(e: Euro): number {
    return e;
}

function main() {
    toNumber(2); // allowed ↯
}
```

TYPE WRAPPER

```
type Euro =  { value: number };

function toNumber(e: Euro): number {
    return e;
}

function main() {
    toNumber(2); // error ✔
    toNumber( {value: 2} ); // allowed ✔, but overhead ↯
}
```

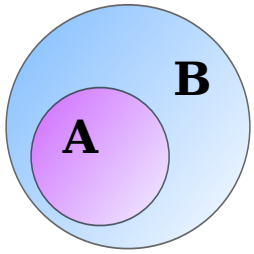
WORKAROUND

```ts
type Euro = number & { readonly __tag: unique symbol };

function toNumber(e: Euro): number {
    return e;
}

toNumber(2); // error ✓
toNumber(2 as Euro); // allowed ✓,  no overhead ✓
```

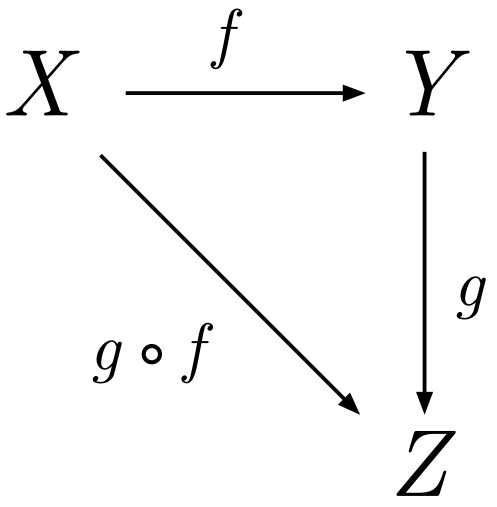
Source https://kubyshkin.name/posts/newtype-in-typescript/

TYPEWRAPPER IN DDD

```ts
type RawId       = string
type ValidatedId = string & { readonly __tag: unique symbol };

function validate(id: RawId): ValidatedId {
  // checks...
  return id as ValidatedId;
}

function fetchUser(id: ValidatedId) {..}

fetchUser("123"); // error
```



FURTHER STUDY



- Category Theory
  - Product Type
  - Sum Type
  - Exponential Type
  - Functor (map)
  - Monad (flatMap, Optional, Streams, RxJs)
- Haskell
- Rust

// reveal.js plugins