3. Lineáris algebra, és a tuple-k

Lineáris Algebra

- Fejezzük ki a vektor műveleteket a funkcionális primitívekkel!

Lineáris Algebra

- Skalárral szorzás, osztás:

```
fmap( mul(c), v)
fmap( div(c), v)
```

- Elemenkénti összeadás, kivonás:

```
zip( add, v, u)
zip( sub, v, u)
```

Lineáris Algebra

- Vektor négyzetes hossza:

```
fold( add, 0, (\x y->x+y*y), v)
```

- Skaláris szorzat:

```
fold( add, 0, zip( mul, v, u) )
```

- Diadikus szorzat:

```
fmap( \c->fmap( mul(c), v ) u)
```

Lineáris Algebra

- Mátrix-vektor szorzat:

```
fmap( \R -> fold( add, 0, zip( mul, R, u) ), M )
```

Lineáris Algebra

- Tuple-k:

Ha különböző típusú dolgokat kell tárolnunk, akkor kell egy tuple típus.

Probléma: a GPU-ra sokszor csak úgynevezett standard layout típust lehet kirakni, és az std::tuple nem ilyen.

Kell írni egy sajátot!

Hogyan? Pl.: Rekurzívan.

Lineáris Algebra

Alap tuple műveletek:

- Indexelés (fordítás idejű Int-el)
- Map
- Zip
- Fold

Példakód: tuple.h

A C++ logika mindegyikre ugyan az: van egy verzió, amit a user hív (pl. idx), utána az overloading-al (pl. idx_impl) szétválik a rekurzió induktív és lezáró esetére.

Lineáris Algebra

Vegyük észre, hogy a Tuple és a Vector Funktor is indexelhető.

Mi a közös bennük?

Mindkettő úgynevezett Naperian Functor, az rögzített alakú funktor:

```
class Functor f ⇒ Naperian f where
    type Pos f
    lookup :: f a → (Pos f → a)
    tabulate :: (Pos f → a) → f a
```

Lineáris Algebra

```
Naperian Functor:
    class Functor f ⇒ Naperian f where
        type Pos f
        lookup :: f a → (Pos f → a)
        tabulate :: (Pos f → a) → f a
```

John Naperian

`Pos f` az `f` Funktor pozíciói (indexei), ahol `a` típusú elemek találhatóak.

`Pos f → a` egy indexelő függvény (pozíció megy be, érték jön ki).

Ezt egy tárolóra hattatva lehet indexelni (lookup),
vagy ebből egy tárolót felépíteni (tabulate)

Forrás:
Jeremy Gibbons:
APLicative Programming with
Naperian Functors

Lineáris Algebra

```
Naperian Functor:
    class Functor f ⇒ Naperian f where
        type Pos f
        lookup :: f a → (Pos f → a)
        tabulate :: (Pos f → a) → f a
```

John Naperian

De miért Naperian?

Ha f és g Naperian Functor, akkor a kettő szorzatának Pozíció típusa:

$$\text{Pos}\,(f \times g) \simeq \text{Pos}\,f + \text{Pos}\,g$$

Továbbá:

$$\text{Pos}\,(f \circ g) \simeq \text{Pos}\,f \times \text{Pos}\,g$$

Lineáris Algebra

John Naperian

De miért Naperian?

Ha f és g Naperian Functor, akkor a kettő szorzatának Pozíció típusa:

$$\text{Pos}\,(f \times g) \simeq \text{Pos}\,f + \text{Pos}\,g$$

Továbbá:

$$\text{Pos}\,(f \circ g) \simeq \text{Pos}\,f \times \text{Pos}\,g$$

Szavakban:
Ha két Naperian Functort összeszorzunk, az indexeik összeadódnak (vagy az egyik indexeivel, vagy a másikéival tudunk elemeket kivenni a szorzatból)

Ha két ilyen tárolót komponálunk (egymásba ágyazunk: Vector<Vector<a>>), akkor az eredményt az indexeik szorzatával lehet indexelni.

Mi az a művelet, ami ilyen algebrai tulajdonságokkal rendelkezik?

Lineáris Algebra

De miért Naperian?

Ha f és g Naperian Functor, akkor a kettő szorzatának Pozíció típusa:

$$\text{Pos}\,(f \times g) \simeq \text{Pos}\,f + \text{Pos}\,g$$

Továbbá:

$$\text{Pos}\,(f \circ g) \simeq \text{Pos}\,f \times \text{Pos}\,g$$

John Naperian

Mi az a művelet, ami ilyen algebrai tulajdonságokkal rendelkezik?

A logaritmus:

$$\text{Log}\,(f \times g) \simeq \text{Log}\,f + \text{Log}\,g$$

Továbbá:

$$\text{Log}\,(f \circ g) \simeq \text{Log}\,f \times \text{Log}\,g$$

Azaz, `type Pos f` -et hívhatnánk `Log f`-nek is.

Lineáris Algebra

Mivel minden fix struktúrájú tároló (pl. std::array, std::tuple, a saját Vector ill. Tuple osztályunk stb., DE szemben a nem fix alakú tárolókkal std::vector, vagy az összegtípusokkal pl. lista) Naperian, ezért a map, zip, fold kifejezhető az indexeléssel.

L. példakód: indexible.h

A kulcs algebrai tulajdonság, hogy az egymásba ágyazott Naperian tárolók transzponálhatóak: $f\ (g\ a) \simeq g\ (f\ a)$

```
transpose :: (Naperian f, Naperian g) ⇒ f (g a) → g (f a)
transpose = tabulate . fmap tabulate . flip . fmap lookup . lookup
```

Ahol:

```
flip :: (a → b → c) → (b → a → c)
```