1 Kontejnerová algebra

Definice: Paměť je posloupnost bajtů. Každý bajt má adresu z $\mathbb{N}_0$.

```
// Pamet je proste RAMka - pole bajtu
```

Definice: Cesta p je konečná, uspořádaná k-tice $(p_k, p_{k-1}, ..., p_1, p_0)$ pro $p_i \in \mathbb{N}_0$

```
// Je to posloupnost indexu ktera popisuje cestu
// skrz nejakou stromovou datovou strukturu:
char c = data[1][42][156][0]; // p = (1, 42, 156, 0)
```

Definice: P budeme značit univerzum všech cest p.

Definice: Velikost (míra) je funkční symbol μ, který danému symbolu přiřazuje přirozené číslo.

```
// Je to neco, co mi rekne, jak velky je neco jinyho
// Odpovida konstruktu:
sizeof(...)
```

Definice: Mějme množinu $P_k \subseteq P$, potom zobrazení $k : P_k \to \mathbb{N}$ se nazývá kontejner, pokud splňuje axiomy:

- $\mu(k) \in \mathbb{N}$ kontejner má předem známou konstantní velikost
- $k(p) \in [0, \mu(k) - 1]$ kontejner mapuje cesty do paměti, vejde se do své velikosti

Poznámka: $k(p)$ pro $p \notin P_k$ není definované.

Poznámka: k nemusí být prosté, takže velikost množiny P_k není omezená.

```
// Kontejner ma nejakou velikost a je to mapovani z cest
// do offsetu v pameti. Mapovani to nemusi byt proste
// a pro nektere cesty nemusi byt definovane. Mapovani ktere
// neni proste je napr. cyklicky buffer s indexem
// modulo velikost.
```

Poznámka: Nemá smysl definovat dimenzi kontejneru, jako délku cesty, protože ta může záviset na svých hodnotách. (`tuple<int, array<5, float>>` nemá dimenzi)

Definice: Kontejnerový generátor je funkce $\kappa(\alpha_1, ..., \alpha_n) \mapsto k$, která pro dané parametry α_i vrátí kontejner k.

Poznámka: Kontejner k je vlastně nulární kontejnerový generátor $k()$.

1.1 Primitivní typy

Člověk má tendenci chápat primitivní typ (`int`, `float`) jako základní stavební jednotku. To ale nejde, protože paměť je pole bajtů a veškeré offsety se v bajtech počítají. Takže je mnohem užitečnější nahlížet na primitivní typ jako na pole bajtů a celé struktuře přidat jednu dimenzi. (jako kdybychom mohli indexovat bajty v rámci primitivního typu). Jedině tak budou platit definice kontejnerů jako zobrazení do paměti. A `sizeof` beztak vrací počet bajtů. Takže z hlediska analýzy přidáme tuhle jednu nejspodnější dimenzi, ale z hlediska uživatele ji skryjeme a bude vždy rovna nule.

Prozatím tedy chápejme integer jako pole čtyřech bajtů:

```
int i = 42;
i[0] // 0
i[3] // 42
array<4, char> == int;
```

1.2 Příklady kontejnerů

```
// poznamky k syntaxi:
// zavorky <> jsou argumenty pro kontejnerovy generator
// zavorky [] jsou argumenty pro kontejner (cesta)
// podtrzitko _ znamena "zbytek cesty"
// velikost se musi definovat extra pomoci sizeof() = ...
//      (do budoucna udelat tridy s .map a .size ??)

// pole bajtu
bytes<n>[i, _] = i          // (podtrzitko se nepouzije)
sizeof(bytes<n>) = n

// nezapomen:
// bytes<3> je kontejner a sizeof(bytes<3>) = 3 jeho velikost
// bytes<n> je generator a sizeof(bytes<n>) = n jen
// schema pro to, jak pocitat velikost kontejneru

// primitivni typy (jen aliasy)
int = bytes<4>
double = bytes<8>

// pole jinych typu T delky n
array<n, T>[i, _] = sizeof(T) * i + T[_]     // "_" rekurze!
sizeof(array<n, T>) = sizeof(T) * n

// identita - kontejner co nedela nic
id<T>[_] = T[_]
sizeof(id<T>) = sizeof(T)
```

```
// matice po radcich
matrix<w, h, T>[i, j, _] = array<w, array<h, T>>[i, j, _]
sizeof(matrix<w, h, T>) = sizeof(array<w, array<h, T>>)

// cinska veta o zbytkach pro w, h nesoudelne
cvz<w, h, T>[i, _] = matrix<w, h, T>>[i % w, i % h, _]
sizeof(cvz<w, h, T>) = sizeof(matrix<w, h, T>>)

// z-krivka je podobna matici, jen prepocitava indexy primo

// tuple
tuple<Ts...>[i, _] = sum(sizeof(T) for T in Ts.slice(i - 1)) + Ti[_]
sizeof(tuple<Ts...>) = sum(sizeof(T) for T in Ts)

// kontejner ktery jen prehodi poradi prvnich dvou indexu
flipAxes<T>[i, j, _] = T[j, i, _]
sizeof(flipAxes<T>) = sizeof(T)

// to nam pomalu vede na transformace:
// nasledujici dva kontejnery maji stejny interface
// ale jine fyzicke ulozneni dat v pameti:
first = array<3, array<2, int>>
second = flipAxes<array<2, array<3, int>>>

for i in 0..3
    for j in 0..2
        first[i, j] == second[i, j] // !!!! see???
```