# Kurs języka Haskell 2024/25

LISTA NR 10 (TERMIN: 27.01.2025, godz 5:00)

**Uwaga:** Wszystkie rozwiązania należy umieścić w jednym module o nazwie `Lista10` zachowując sygnatury zgodne z szablonem rozwiązań zamieszczonym w SKOS-ie.

**Zadanie 1.** Rozważ typ drzew binarnych etykietowanych w węzłach:

```
data Tree a = Node a (Tree a) (Tree a) | Leaf
```

Zainstaluj ten typ w klasie HasShape, a następnie zdefiniuj algebry

```
sumTreeAlg  :: Shape (Tree Int) Int → Int
prodTreeAlg :: Shape (Tree Int) Int → Int
```

definiujące odpowiednio sumę i produkt wartości etykiet przez katamorfizm. Na przykład:

```
ghci> ex1 = Node 5 (Node 2 Leaf Leaf) (Node 10 Leaf Leaf)
ghci> cata sumTreeAlg ex1
17
ghci> cata prodTreeAlg ex1
100
```

**Zadanie 2.** W przykładzie z poprzedniego zadania obliczaliśmy wartości dwóch katamorfizmów na tej samej strukturze danych. Gdybyśmy chcieli zamknąć tę funkcjonalność w jednej funkcji, moglibyśmy napisać:

```
sumProd :: Tree Int → (Int, Int)
sumProd t = (cata sumTreeAlg t, cata prodTreeAlg t)
```

Łatwo zauważyć sposobność optymalizacji: każdy z tych katamorfizmów przechodzi przez strukturę, więc w sumie musimy zwinąć drzewo dwa razy. A można przejść przez drzewo tylko raz, dla każdego konstruktora produkując parę rezultatów.

Zaimplementuj tę funkcjonalność w funkcji

```
parCata :: (HasShape d)
        ⇒ (Shape d a → a)
        → (Shape d b → b)
        → d
        → (a, b)
```

która daje taki sam rezultat jak

```
parCata f g d = (cata f d, cata g d)
```

ale przechodzi przez `d` tylko jeden raz.

[1] https://en.wikipedia.org/wiki/Stern-Brocot_tree

**Zadanie 3.** Rozwiąż problem repmin dla drzew etykietowanych w węzłach z Zadania 1. używając funkcji parCata z poprzedniego zadania dla algebry definiującej katamorfizm obliczający minimalną wartość w drzewie i algebry definiującej katamorfizm podstawiający pewną wartość w drzewie. Dla uproszczenia uznajemy, że typ etykiet to **Int**, a minimalną wartością w drzewie Leaf jest **maxBound**.

*Uwaga:* Po rozwiązaniu tego zadania spójrz jeszcze raz na napisany kod i doceń to, co właśnie się stało. Napisał(e/a/u)ś dwie odrębne funkcje (do oblicania minimum i do podstawiania wartości), zupełnie jak w pierwszym, naiwnym, przechodzącym drzewo dwa razy rozwiązaniu repmina z trzeciego wykładu. Nagrodą za to, że umiesz wyrazić te funkcje jako katamorfizmy, jest to, że repmin właściwie rozwiązał się sam.

**Zadanie 4.** Zaimplementuj drzewo Sterna–Brocota[1] reprezentowane przy użyciu typu z Zadania 1., tworzone przy pomocy anamorfizmu. Ułamki reprezentuj jako znormalizowane pary wartości typu **Integer**.

**Zadanie 5.** Sortowanie bąbelkowe polega na tym, że:

- Zaczynając od końca listy, porównujemy dwa sąsiednie elementy. Jeśli drugi (czyli ten bliżej końca listy) jest mniejszy, zamieniamy je miejscami.
- Po przebąbelkowaniu całej listy wiemy, że w jej głowie znajduje się najmniejszy element. Wystarczy więc rekurencyjnie uruchomić algorytm dla ogona.

Zaimplementuj algorytm sortowania bąbelkowego, w którym pojdyncze „przebąbelkowanie" się przez listę jest katamorfizmem, a „rekurencyjne wywołanie dla ogona" realizowane jest anamorfizmem.

*Wskazówka:* Najwygodniejszym typem dla algebry definiującej bąbelkowanie jest

```
(Ord a) ⇒ Shape [a] (Shape [a] [a]) → Shape [a] [a]
```

**Zadanie 6.** Przedstaw algorytm quicksort jako hylomorfizm:

- Część „ana" buduje drzewo BST zgodnie z zasadami quicksorta: elementy niewiękdsze od piwota idą na lewo, a większe – na prawo.
- Część „kata" spłaszcza drzewo do listy.