

# Lista zagadnień nr 11

## Przed zajęciami

Tematem przewodnim bieżącego tygodnia jest mechanizm **kontraktów**. Na wykładzie omówiliśmy **kontrakty dla funkcji**, **kontrakty dla struktur**, **kontrakty parametryczne** oraz **kontrakty rekurencyjne**, poznaliśmy też pojęcie **polaryzacji** pozycji w kontrakcie dla funkcji. Przed zajęciami należy zapoznać się z kodem źródłowym z wykładu oraz przejrzeć **rozdział 7** dokumentacji Racketa **The Racket Guide** (<https://docs.racket-lang.org/guide/>).

## Na zajęciach

### Ćwiczenie 1.

Napisz procedurę `suffixes`, zwracającą wszystkie sufiksy listy podanej jako argument. Napisz dla tej procedury odpowiedni kontrakt parametryczny.

### Ćwiczenie 2.

Poniższa procedura ma za zadanie obliczyć listę wszystkich podlist listy podanej jako argument:

```
(define (sublists xs)
  (if (null? xs)
      (list null)
      (append-map
        (lambda (ys) (cons (cons (car xs) ys) ys))
        (sublists (cdr xs)))))
```

Niestety, procedura ta zawiera błąd:

```
> (sublists '(1 2))
'((1 2) 2)
> (sublists '(1 2 3))
'((1 2 3) 2 3 (1 . 3) . 3)
```

Napisz kontrakt parametryczny dla tej procedury, który odrzuci błędne wyniki. Popraw procedurę, aby działała zgodnie z założeniem oraz spełniała swój kontrakt.

### Ćwiczenie 3.

Wskaż w poniższych kontraktach wystąpienia pozytywne i negatywne. Zaimplementuj procedury spełniające te kontrakty.

```
(parametric->/c [a b] (-> a b a))
(parametric->/c [a b c] (-> (-> a b c) (-> a b) a c))
(parametric->/c [a b c] (-> (-> b c) (-> a b) (-> a c)))
(parametric->/c [a] (-> (-> (-> a a) -> a) -> a))
```

### Ćwiczenie 4.

Zaimplementuj procedurę spełniającą poniższy kontrakt:

```
(parametric->/c [a b] (-> a b))
```

*Podpowiedź:* Kod procedury nie jest w stanie wygenerować wartości oznakowanej b. Co można zrobić, żeby mimo tego nigdy nie naruszyć kontraktu?

### Ćwiczenie 5.

Poniższy kod implementuje procedurę łączącą w sobie cechy `foldl` i `map`:

```
(define (foldl-map f a xs)
  (define (it a xs ys)
    (if (null? xs)
        (cons (reverse ys) a)
        (let [(p (f (car xs) a))]
          (it (cdr p)
              (cdr xs)
              (cons (car p) ys)))))
  (it a xs null))
```

Pierwszy argument powinien być procedurą przyjmującą dwa argumenty, oznaczające (w kolejności) bieżący element listy oraz bieżący akumulator, zaś zwracającą parę złożoną z nowego elementu listy oraz nowej wartości akumulatora. Pozostałe dwa argumenty powinny zawierać startową wartość akumulatora oraz listę elementów do przetworzenia. Procedura `foldl-map` zwraca parę złożoną z listy wynikowej i końcowej wartości akumulatora.

Przykładowe wywołanie procedury, obliczające sumy częściowe:

```
(foldl-map (lambda (x a) (cons a (+ a x))) 0 '(1 2 3))
```

Napisz kontrakt parametryczny dla tej definicji. Zastosuj w kontrakcie jak najwięcej (prawidłowo użytych) parametrów.

### **Ćwiczenie 6.**

Na wykładzie szóstym pojawił się interpreter prostych wyrażeń arytmetycznych ze zmiennymi i let-wyrażeniami, wykorzystujący model podstawieniowy. Napisz kontrakt rekurencyjny  $\text{expr}/c$  spełniony przez prawidłowo zbudowane wyrażenia arytmetyczne (nie trzeba sprawdzać, czy wystąpienia zmiennych są związane). Napisz kontrakty dla procedur `subst` i `eval`.