MP23 @ II UWr 22 marca 2023 r.

Lista zadań nr 5

Poniższe zadania należy rozwiązać przy użyciu języka Plait.

Zadanie 1.

Zaimplementuj procedury o podanych poniżej typach. Procedury nie powinny pętlić się ani zgłaszać błędów.

```
('a 'b -> 'a)
(('a 'b -> 'c) ('a -> 'b) 'a -> 'c)
((('a -> 'a) -> 'a) -> 'a)
(('a -> 'b) ('a -> 'c) -> ('a -> ('b * 'c)))
(('a -> (Optionof ('a * 'b))) 'a -> (Listof 'b))
```

Zadanie 2.

Napisz i uzasadnij typy procedur zdefiniowanych poniżej.

```
(define (apply f x) (f x))
(define (compose f g) (lambda (x) (f (g x))))
(define (flip f) (lambda (x y) (f y x)))
(define (curry f) (lambda (x) (lambda (y) (f x y))))
```

Zadanie 3. (2 pkt)

Napisz i uzasadnij typy poniższych, wykorzystujących procedury z poprzedniego zadania, wyrażeń.

```
(curry compose)
((curry compose) (curry compose))
((curry compose) (curry apply))
((curry apply) (curry compose))
(compose curry flip)
```

Zadanie 4.

Zaimplementuj procedurę `perms`, obliczającą wszystkie permutacje zadanej listy (w dowolnej kolejności). Na przykład:

```
> (perms '(1 2 3))
'((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

Zadanie 5. (2 pkt)

Dla drzew binarnych z wykładu zdefiniuj procedurę `process-tree` o następującym typie:

```
(('a 'b 'c 'b -> 'b) ; procedura dla węzła
 ('a -> 'b)          ; procedura dla liścia
 ('a 'c -> 'a)       ; akumulacja dla lewej gałęzi
 ('a 'c -> 'a)       ; akumulacja dla prawej gałęzi
 'a                  ; wartość akumulatora
 (Tree 'c)
 ->
 'b)
```

Procedurę tę można rozumieć jak procedurę `fold-tree` rozbudowaną o akumulację wzdłuż gałęzi.

Wykorzystując zdefiniowaną procedurę, zaimplementuj znane z poprzedniej listy zadań procedury `bst?` oraz `sum-paths`. Przypomnijmy:

- `(bst? t)` – predykat sprawdzający, czy zadane drzewo spełnia własność BST.
- `(sum-paths t)` – produkuje drzewo o tym samym kształcie co `t`, w którym etykietą danego węzła jest suma wartości węzłów na ścieżce od korzenia drzewa do tego węzła.

Zadanie 6.

Zaproponuj typ danych *rose trees* – drzew zawierających elementy w liściach, których węzły mogą posiadać dowolną liczbę poddrzew (w tym zerową). Zaimplementuj procedurę wypisującą wszystkie elementy drzewa rose tree w porządku przeszukiwania w głąb, od lewych do prawych poddrzew. Zadbaj o to, aby nie generować nieużytków.

Zadanie 7.

Poniższy typ danych definiuje formuły rachunku zdań:

```
(define-type Prop
  (var [v : String])
  (conj [l : Prop] [r : Prop])
  (disj [l : Prop] [r : Prop])
  (neg [f : Prop]))
```

Zaimplementuj procedurę `free-vars`, zwracającą listę wszystkich zmiennych wolnych występujących w formule, bez powtórzeń:

```
free-vars : (Prop -> (Listof String))
```

Ponieważ w formułach rachunku zdań zmienne nie są wiązane, wszystkie wystąpienia zmiennych w tych formułach są wolne.

Zadanie 8.

Język Plait dostarcza implementację słowników, na którą składają się, między innymi, następujące procedury:

```
hash     : ((Listof ('a * 'b)) -> (Hashof 'a 'b))
hash-ref : ((Hashof 'a 'b) 'a -> (Optionof 'b))
hash-set : ((Hashof 'a 'b) 'a 'b -> (Hashof 'a 'b))
```

Słowniki typu `(Hashof String Boolean)` możemy rozumieć jako wartościowania zmiennych występujących w formule. Zaimplementuj procedurę obliczającą wartość formuły dla zadanego wartościowania:

```
eval : ((Hashof String Boolean) Prop -> Boolean)
```

Zadanie 9. (2 pkt)

Zaimplementuj procedurę `tautology?` sprawdzającą, czy zadana formuła rachunku zdań jest tautologią. Przypomnijmy, że formuła jest tautologią wtedy i tylko wtedy, gdy jej wartością jest prawda dla dowolnego wartościowania występujących w niej zmiennych.