Egzamin z Programowania Funkcyjnego - OCaml

Czas: 180 minut | Punkty: 100

Zadanie 1: Zmienne Wolne i Związane (10 pkt)

W poniższych wyrażeniach OCaml podkreśl **wolne** wystąpienia zmiennych i narysuj strzałki od **związanych** wystąpień do ich definicji.

a) (2 pkt)

```ocaml
let f x = x + y in
let y = 5 in
f (y + 1)
```

b) (3 pkt)

```ocaml
let rec map f lst =
  match lst with
  | [] -> []
  | h :: t -> f h :: map f t
in map
```

c) (3 pkt)

```ocaml
fun x ->
  let y = x + z in
  fun z -> x + y + z
```

d) (2 pkt)

```ocaml
let x = 10 in
let f = fun y -> x + y in
let x = 20 in
f x
```

Zadanie 2: Typowanie Funkcji (15 pkt)

Podaj najogólniejszy typ dla poniższych wyrażeń lub napisz **BRAK TYPU**.

a) (2 pkt) `fun f x -> f (f x)`

b) (2 pkt) `fun x y z -> if x then y else z`

c) (3 pkt) `fun f g x -> f x && g x`

d) (3 pkt) `let rec length lst = match lst with [] -> 0 | _::t -> 1 + length t in length`

e) (2 pkt) `fun x -> (x 1, x "hello")`

f) (3 pkt) `fun lst -> match lst with [x; y] -> x + y | _ -> 0`

Zadanie 3: Funkcje dla Typów (12 pkt)

Dla podanych typów napisz funkcje o podanych sygnaturach:

a) Typ: `'a list * 'a list` **(4 pkt)**

- `merge_lists : 'a list * 'a list -> 'a list` (łączy dwie listy)
- `lists_equal : 'a list * 'a list -> bool` (sprawdza czy listy mają tę samą długość)

b) Typ: `('a -> 'b) * ('b -> 'c)` **(4 pkt)**

- `compose : ('a -> 'b) * ('b -> 'c) -> ('a -> 'c)` (składa funkcje)
- `apply_both : ('a -> 'b) * ('b -> 'c) -> 'a -> 'b * 'c`

c) Typ: `'a option option` **(4 pkt)**

- `flatten_option : 'a option option -> 'a option`
- `is_some_some : 'a option option -> bool`

Zadanie 4: Zasady Indukcji (8 pkt)

a) (4 pkt) Sformułuj zasadę indukcji strukturalnej dla typu:

```
ocaml
```

```ocaml
type 'a tree =
  | Empty
  | Node of 'a tree * 'a * 'a tree
```

b) (4 pkt) Sformułuj zasadę indukcji dla typu:

```ocaml
type expr =
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
```

Zadanie 5: Dowody Indukcyjne (10 pkt)

a) (5 pkt) Udowodnij przez indukcję strukturalną: `List.length (List.rev lst) = List.length lst`

b) (5 pkt) Dla typu `'a tree` z zadania 4a, udowodnij: `tree_height (mirror_tree t) = tree_height t`

gdzie:

```ocaml
let rec mirror_tree = function
  | Empty -> Empty
  | Node(l, x, r) -> Node(mirror_tree r, x, mirror_tree l)
```

Zadanie 6: Gramatyki Bezkontekstowe (10 pkt)

a) (5 pkt) Dla gramatyki:

```
Expr ::= Num | Expr "+" Expr | Expr "*" Expr | "(" Expr ")"
Num ::= [0-9]+
```

Zdefiniuj typ AST w OCamlu i pokaż jak reprezentować wyrażenie `"2 + 3 * 4"`.

b) (5 pkt) Napisz gramatykę dla prostych instrukcji:

- przypisanie: `x = expr`
- if: `if expr then stmt else stmt`
- blok: `{ stmt; stmt; ... }`
- while: `while expr do stmt`

Zadanie 7: Interpreter (25 pkt)

Zaimplementuj interpreter dla języka z następującymi konstrukcjami:

```ocaml
type expr =
  | Num of int
  | Bool of bool
  | Var of string
  | Add of expr * expr
  | Sub of expr * expr
  | Eq of expr * expr
  | Lt of expr * expr
  | And of expr * expr
  | Or of expr * expr
  | If of expr * expr * expr
  | Let of string * expr * expr
  | Fun of string * expr
  | App of expr * expr
  | Fix of string * expr    (* rekurencja *)

type value = VNum of int | VBool of bool | VFun of string * expr * env
and env = (string * value) list
```

a) (15 pkt) Zaimplementuj funkcję `eval : env -> expr -> value`

b) (5 pkt) Pokaż jak reprezentować i wykonać funkcję rekurencyjną:

```ocaml
let rec factorial n = if n = 0 then 1 else n * factorial (n-1)
```

c) (5 pkt) Dodaj obsługę wyjątków - rozszerz typy i interpreter o:

- `Raise of string`
- `Try of expr * string * expr`

Zadanie 8: Funkcje z Fold (10 pkt)

Zaimplementuj używając **tylko** `List.fold_left`, `List.fold_right` lub `tree_fold`:

a) (3 pkt) `all_pairs : 'a list -> 'b list -> ('a * 'b) list`

Wszystkie pary elementów z dwóch list.

b) (3 pkt) `group_by : ('a -> 'b) -> 'a list -> ('b * 'a list) list`

Grupuje elementy według wyniku funkcji.

c) (4 pkt) `tree_paths : 'a tree -> 'a list list`

Wszystkie ścieżki od korzenia do liści w drzewie.

Zadanie 9: Kompilator do Maszyny Stosowej (10 pkt)

Dla wyrażeń arytmetycznych:

```ocaml
type simple_expr =
  | SNum of int
  | SAdd of simple_expr * simple_expr
  | SMul of simple_expr * simple_expr

type instruction =
  | Push of int
  | Add
  | Mul

type machine = { stack: int list }
```

a) (5 pkt) Zaimplementuj kompilator:

```ocaml
compile : simple_expr -> instruction list
```

b) (3 pkt) Zaimplementuj maszynę wirtualną:

```ocaml
execute : instruction list -> machine -> int
```

c) (2 pkt) Pokaż kompilację wyrażenia `2 + 3 * 4` i jego wykonanie.

Zadanie 10: Zadanie Dodatkowe (5 pkt)

Zaimplementuj funkcję `optimize : simple_expr -> simple_expr`, która wykonuje podstawowe optymalizacje:

- `0 + x` → `x`
- `x + 0` → `x`
- `1 * x` → `x`
- `x * 0` → `0`

Powodzenia! 🍀