Metody Programowania — Egzamin

Czas: 180 minut
Punktacja: 17 pkt = 3.0, 21 pkt = 3.5, 25 pkt = 4.0, 29 pkt = 4.5, 33 pkt = 5.0

Zadanie 1. (3 pkt) — Zmienne wolne i związane

W poniższych wyrażeniach w języku OCaml podkreśl **wolne wystąpienia** zmiennych. Dla każdego **związanego wystąpienia** zmiennej, narysuj strzałkę od tego wystąpienia do wystąpienia wiążącego je.

a)

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

b)

```ocaml
fun f ->
  let x = f 3 in
  fun y -> x + f y + w
```

c)

```ocaml
let rec factorial n =
  if n <= 1 then 1
  else n * factorial (n - 1)
in factorial k
```

d)

```ocaml
fun g ->
  let h = fun x -> g x + y in
  fun y -> h (y + 1)
```

e)

```ocaml
let x = 10 in
let rec loop y =
  if y > 0 then x + loop (y - 1)
  else z
in loop x
```

Zadanie 2. (3 pkt) — Typowanie funkcji

Dla poniższych wyrażeń w języku OCaml podaj ich **najogólniejszy typ**, lub napisz **„BRAK TYPU"**, gdy wyrażenie się nie typuje.

a) `fun f g x -> f (g x) (g x)` : _____

b) `fun lst -> List.fold_left (fun acc x -> x :: acc) [] lst` : _____

c) `fun x -> x x` : _____

d) `fun f -> fun opt -> match opt with | Some x -> f x | None -> None` : _____

e) `fun pred lst -> List.fold_left (fun (yes, no) x -> if pred x then (x::yes, no) else (yes, x::no)) ([], []) lst` : _____

Zadanie 3. (3 pkt) — Typ → Funkcja

Napisz **dowolną poprawną funkcję** realizującą każdy z poniższych typów:

a) `('a -> 'b -> 'c) -> 'b -> 'a -> 'c` : _____

b) `('a -> bool) -> 'a list -> ('a list * 'a list)` : _____

c) `'a option -> ('a -> 'b) -> 'b option` : _____

Zadanie 4. (3 pkt) — Alpha-konwersja

Przeprowadź **alpha-konwersję** dla poniższych wyrażeń tak, aby każda zmienna miała unikalną nazwę. Użyj nazw `a`, `b`, `c`, `d`, ... w kolejności potrzeb.

a)

```ocaml
fun x ->
  let x = x + 1 in
  fun y ->
    let x = y * 2 in
    x + y
```

b)

```ocaml
let rec f x =
  let f = fun y -> x + y in
  f (f x)
```

c)

```ocaml
fun g ->
  let h = fun g -> g + 1 in
  fun h -> g (h 5)
```

Zadanie 5. (4 pkt) — Indukcja strukturalna

Rozważmy typ drzew binarnych z wartościami w liściach:

```ocaml
type 'a btree =
  | Leaf of 'a
  | Node of 'a btree * 'a btree
```

a) (1 pkt) Sformułuj **zasadę indukcji strukturalnej** dla typu `'a btree`.

b) (3 pkt) Udowodnij indukcyjnie, że dla dowolnego drzewa `t : 'a btree`:

```
leaf_count t = node_count t + 1
```

gdzie:

```ocaml
let rec leaf_count = function
  | Leaf _ -> 1
  | Node(l, r) -> leaf_count l + leaf_count r

let rec node_count = function
  | Leaf _ -> 0
  | Node(l, r) -> 1 + node_count l + node_count r
```

Zadanie 6. (2 pkt) — Monady

a) (1 pkt) Dla typu `'a option = None | Some of 'a`, napisz funkcje:

```ocaml
let return x = _______________
let bind m f = _______________
```

b) (1 pkt) Używając tylko `bind` i `return`, napisz funkcję:

```ocaml
let map2 f opt1 opt2 = _______________
```

która aplikuje funkcję `f` do wartości z dwóch opcji (zwraca `None` jeśli którakolwiek to `None`).

Zadanie 7. (3 pkt) — Funkcje fold

Zaimplementuj poniższe funkcje używając **wyłącznie** `List.fold_left` lub `List.fold_right`:

a) `let max_element lst = _____`

(zwraca maksymalny element z niepustej listy int)

b) `let zip_with_index lst = _____`

(dodaje indeksy: `[a;b;c]` → `[(0,a);(1,b);(2,c)]`)

c) `let take_while pred lst = _____`

(bierze elementy z początku dopóki spełniają predykat)

d) `let insert_sorted x lst = _____`

(wstawia element x we właściwe miejsce w posortowanej liście)

e) `let group_by_parity lst = _____`

(dzieli listę int na parzyste i nieparzyste: zwraca `(parzyste, nieparzyste)`)

Zadanie 8. (2 pkt) — Fold na drzewach

Dla typu `type 'a tree = Empty | Node of 'a tree * 'a * 'a tree`, zaimplementuj:

a) `let tree_fold f init t = _____`
(ogólna funkcja fold dla drzew)

b) Używając `tree_fold`, napisz:

```ocaml
let tree_to_list t = _____ (* in-order traversal *)
let tree_max t = _____ (* maksymalny element, None dla Empty *)
```

Zadanie 9. (3 pkt) — Gramatyki bezkontekstowe

a) (1 pkt) Napisz gramatykę bezkontekstową dla języka:

```
L = {a^n b^m a^n | n,m ≥ 0}
```

(równa liczba `a` na początku i końcu, dowolnie dużo `b` w środku)

b) (2 pkt) Napisz gramatykę dla wyrażeń arytmetycznych z operatorami `+`, `*`, `-` (unarny minus), nawiasami i liczbami, gdzie:

- `-` (unarny) ma najwyższy priorytet
- `*` ma wyższy priorytet niż `+`
- operatory binarne są lewostronnie łączne

Zadanie 10. (4 pkt) — Operacje na drzewach BST

Dla typu `type 'a tree = Empty | Node of 'a tree * 'a * 'a tree`:

a) (1 pkt) `let tree_search x t = _____`
(sprawdza czy element x jest w BST)

b) (2 pkt) `let tree_insert x t = _____`
(wstawia element x do BST)

c) (1 pkt) `let tree_min t = _____`
(zwraca najmniejszy element z BST, `None` dla `Empty`)

Zadanie 11. (4 pkt) — Interpreter

Uzupełnij definicje typów dla interpretera języka funkcyjnego:

```ocaml
type expr =
  | Int of int
  | Bool of bool
  | Var of string
  | Add of expr * expr
  | Sub of expr * expr
  | Mult of expr * expr
  | Eq of expr * expr
  | Lt of expr * expr
  | And of expr * expr
  | Or of expr * expr
  | Not of expr
  | If of expr * expr * expr
  | Let of string * expr * expr
  | Fun of string * expr
  | App of expr * expr
  | Funrec of string * string * expr
  | Pair of expr * expr
  | Fst of expr
  | Snd of expr
  | Fix of string * expr

type value =
  | _____________
  | _____________
  | _____________
  | _____________

type env = _____________

let rec eval e env =
  (* Zaimplementuj interpreter *)
  _____________
```

Zadanie 12. (3 pkt) — Kompilator maszyny stosowej

a) (2 pkt) Dla maszyny stosowej z instrukcjami:

```ocaml
type instr =
  | Push of int
  | Add | Sub | Mult
  | Dup                (* duplikuje top stosu *)
  | Swap               (* zamienia dwa top elementy *)
  | MakePair           (* (a,b) z dwóch elementów *)
  | Fst | Snd          (* pierwszy/drugi element pary *)

type value = VInt of int | VPair of value * value
```

Napisz funkcję: `let eval_stack instrs stack = _____`

b) (1 pkt) Skompiluj wyrażenie `(3 + 5, 2 * 4)` do listy instrukcji.

Zadanie 13. (2 pkt) — All variables bound

Dla języka z zadania 11, napisz funkcję sprawdzającą czy wszystkie zmienne w wyrażeniu są związane:

```ocaml
let rec all_vars_bound (bound_vars : string list) (e : expr) : bool =
  (* Sprawdza czy wszystkie zmienne w e są w bound_vars lub związane lokalnie *)
  _____
```

Zadanie 14. (2 pkt) — Max stack height

a) (1 pkt) Dla maszyny stosowej z zadania 12, napisz funkcję obliczającą maksymalną wysokość stosu podczas wykonania programu (zaczynając z pustym stosem):

```ocaml
let max_stack_height (instrs : instr list) : int = _____
```

b) (1 pkt) Oblicz `max_stack_height` dla programu z zadania 12b.

Zadanie 15. (3 pkt) — Type checker 2

Rozważmy język z listami i pattern matchingiem:

ocaml

```ocaml
type expr2 =
  | Int of int
  | Nil
  | Cons of expr2 * expr2
  | Match of expr2 * expr2 * string * string * expr2
  (* Match(lst, nil_case, head_var, tail_var, cons_case) *)
  (* match lst with [] -> nil_case | h::t -> cons_case *)

type typ2 =
  | TInt
  | TList of typ2

type tenv2 = (string * typ2) list
```

Zaimplementuj type checker:

ocaml

```ocaml
let rec type_check2 (tenv : tenv2) (e : expr2) : typ2 =
  match e with
  | Int _ -> _______________
  | Nil -> _______________
  | Cons(e1, e2) -> _______________
  | Match(lst, nil_case, h_var, t_var, cons_case) -> _______________
```

Zadanie 16. (3 pkt) — Type checker

Zadanie 16. (3 pkt) — Type checker

Dla języka z zadania 11, zaimplementuj type checker:

```ocaml
type typ =
  | TInt | TBool
  | TFun of typ * typ
  | TPair of typ * typ

type tenv = (string * typ) list

let rec type_check tenv e =
  match e with
  | Int _ -> _____________
  | Bool _ -> _____________
  | Var x -> _____________
  | Add(e1, e2) -> _____________
  | Pair(e1, e2) -> _____________
  | Fst e -> _____________
  | Fix(f, e) -> _____________
  | (* ... reszta przypadków ... *)
```

Łącznie: 44 punkty