Lista zadań nr 1 (12.06) - Egzamin Podobny do Oryginału

Zadanie 1: Zmienne Wolne/Związane/Wiążące i Zasięg

W poniższych wyrażeniach OCaml wskaż, do której definicji (let) odwołuje się każde użycie zmiennej. Określ też finalną wartość całego wyrażenia.

a)

```ocaml
let a = 7 in
let f x = a + x in
let a = 3 in
f (a + 2)
```

b)

```ocaml
let rec fib n =
  if n <= 1 then n
  else fib (n-1) + fib (n-2) in
let fib = fun x -> x * 2 in
fib 5
```

c)

```ocaml
let x = "outer" in
let g y =
  let x = y ^ "_inner" in
  fun z -> x ^ "_" ^ z ^ "_" ^ y in
let h = g "middle" in
let x = "newest" in
h "end"
```

d)

```ocaml
let rec even n = if n = 0 then true else odd (n-1)
and odd n = if n = 0 then false else even (n-1) in
let even = fun x -> false in
odd 3
```

Zadanie 2: Typowanie Wyrażeń

Dla poniższych wyrażeń podaj ich najogólniejszy typ, lub napisz „BRAK TYPU".

a) `fun f x y -> f (x + 1) (y - 1)`

b) `fun lst -> match lst with [] -> 0 | h :: t -> h`

c) `let rec apply_twice f x = f (f x) in apply_twice`

d) `fun x -> (x 1, x "hello")`

e) `let f = fun x -> x in let g = f f in g 42`

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

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

h) `fun f g x -> f x || g x`

Zadanie 3: Funkcje z fold_left/fold_right

Zaimplementuj używając wyłącznie `List.fold_left` lub `List.fold_right`:

a) `exists (p: 'a -> bool) (lst: 'a list) : bool` - czy istnieje element spełniający predykat

b) `find_index (elem: 'a) (lst: 'a list) : int option` - znajduje indeks pierwszego wystąpienia

c) `take (n: int) (lst: 'a list) : 'a list` - bierze pierwsze n elementów

d) `partition (p: 'a -> bool) (lst: 'a list) : 'a list * 'a list` - dzieli na spełniające i niespełniające

e) `zip (l1: 'a list) (l2: 'b list) : ('a * 'b) list` - łączy dwie listy w pary

Zadanie 4: Definicje Typów i Funkcje

Część A: Drzewa binarne

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

Zaimplementuj:

- `tree_size (t: 'a btree) : int` - liczba węzłów

- `tree_depth (t: 'a btree) : int` - głębokość drzewa
- `in_order (t: 'a btree) : 'a list` - traversal in-order
- `tree_fold (f: 'b -> 'a -> 'b -> 'b) (acc: 'b) (t: 'a btree) : 'b`

Część B: Listy z długością

```ocaml
type 'a sized_list = {
  data: 'a list;
  length: int;
}
```

Zaimplementuj funkcje zachowujące niezmiennik (poprawną długość):

- `sl_empty : 'a sized_list`
- `sl_cons (x: 'a) (sl: 'a sized_list) : 'a sized_list`
- `sl_append (sl1: 'a sized_list) (sl2: 'a sized_list) : 'a sized_list`

Zadanie 5: Interpreter

Dany typ wyrażeń:

```ocaml
type expr =
  | Int of int
  | Bool of bool
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Eq of expr * expr        (* równość *)
  | Lt of expr * expr        (* mniejszość *)
  | If of expr * expr * expr
  | Let of string * expr * expr

type value = VInt of int | VBool of bool
type env = (string * value) list
```

Zaimplementuj:

- `eval (env: env) (e: expr) : value` - interpreter
- `free_vars (e: expr) : string list` - zmienne wolne (bez duplikatów)

Zadanie 6: Maszyna Stosowa

```ocaml
type instruction =
  | Push of int
  | Add | Mul | Sub
  | Load of string    (* załaduj zmienną *)
  | Store of string   (* zapisz do zmiennej *)
  | Jump of int       (* skok bezwarunkowy *)
  | JumpIfZero of int (* skok jeśli 0 na stosie *)

type machine_state = {
  stack: int list;
  vars: (string * int) list;
  pc: int;  (* program counter *)
}
```

Zaimplementuj:

- `execute_instruction (instr: instruction) (state: machine_state) : machine_state`
- `run_program (program: instruction array) (initial_state: machine_state) : machine_state`

Zadanie 7: Kompilator Wyrażeń

Dla typu `expr` z zadania 5, zaimplementuj kompilator do instrukcji maszyny stosowej:

- `compile (e: expr) : instruction list` - kompiluje wyrażenie do listy instrukcji

Przykład: `Add(Int 3, Int 5)` → `[Push 3; Push 5; Add]`

Zadanie 8: Gramatyki i AST

Część A: Gramatyka dla prostych list:

```
List ::= "[" "]" | "[" Elem (";" Elem)* "]"
Elem ::= num | List
```

Zdefiniuj typ AST i napisz prostą funkcję parsującą stringi.

Część B: Rozszerz interpreter z zadania 5 o listy:

```ocaml
type expr =
  | (* poprzednie konstruktory *)
  | List of expr list
  | Head of expr      (* pierwszy element *)
  | Tail of expr      (* reszta listy *)
  | IsEmpty of expr   (* czy lista pusta *)
```

Zadanie 9: Funkcje do Typów (NOWY RODZAJ!)

Dla podanych typów, napisz funkcje o podanych sygnaturach:

a) Typ: `'a option`

- `option_map : ('a -> 'b) -> 'a option -> 'b option`
- `option_bind : 'a option -> ('a -> 'b option) -> 'b option`
- `option_fold : ('a -> 'b) -> 'b -> 'a option -> 'b`

b) Typ: `'a * 'b`

- `pair_map : ('a -> 'c) -> ('b -> 'd) -> ('a * 'b) -> ('c * 'd)`
- `curry : (('a * 'b) -> 'c) -> 'a -> 'b -> 'c`
- `uncurry : ('a -> 'b -> 'c) -> ('a * 'b) -> 'c`

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

- `compose : ('b -> 'c) -> ('a -> 'b) -> ('a -> 'c)`
- `flip : ('a -> 'b -> 'c) -> ('b -> 'a -> 'c)`
- `const : 'a -> 'b -> 'a`

d) Typ: `'a list list`

- `concat_all : 'a list list -> 'a list`
- `transpose_matrix : 'a list list -> 'a list list`
- `group_by_length : 'a list list -> (int * 'a list list) list`

Zadanie 10: Dowody Indukcyjne

Na listach:

- Udowodnij: `List.rev (List.rev lst) = lst`
- Udowodnij: `List.length (l1 @ l2) = List.length l1 + List.length l2`

Na drzewach (dla typu z zadania 4A):

- Udowodnij: `tree_size (mirror_tree t) = tree_size t`
- Udowodnij: `List.length (in_order t) = tree_size t`

Zadanie 11: Typowanie z Polimorfizmem

Podaj typ lub uzasadnij brak typu:

a) `let id x = x in (id 1, id "hello", id [])`

b) `let compose f g x = f (g x) in compose`

c) `let rec fix f = f (fix f) in fix`

d) `fun f -> let x = f 1 in let y = f "a" in (x, y)`

Zadanie 12: Parser Wyrażeń

Napisz prosty parser dla wyrażeń arytmetycznych ze stringów:

```ocaml
type token = NUM of int | PLUS | MULT | LPAREN | RPAREN | EOF

val tokenize : string -> token list
val parse_expr : token list -> expr * token list
```

Gramatyka:

```
expr ::= term ('+' term)*
term ::= factor ('*' factor)*
factor ::= num | '(' expr ')'
```