Precizări:

- Primele 9 subiecte au fiecare câte 10p. Cele 3 subpuncte ale problemei au fiecare câte 10p. **Punctajul NU se acordă în absența justificării răspunsului!**
- Este suficientă rezolvarea a **10** itemi din cei 12 pentru nota maximă.
- Punctajul suplimentar reprezintă **bonus**, ce poate compensa punctajul de pe parcurs.

1. Fie definiția și aplicația funcției Racket de mai jos. Descrieți **pas cu pas** cum decurge evaluarea aplicației, utilizând modelul bazat pe substituție textuală.

```
(define (f x y)
  (lambda (z)
    (if (< x 10) (+ x y) z)))

> ((f (+ 0 1) (+ 2 3)) (+ 4 5))
```

Soluție. .

```
((f (+ 0 1) (+ 2 3)) (+ 4 5))
→ ((f 1 5) (+ 4 5))
→ ((lambda (z) (if (< 1 10) (+ 1 5) z)) (+ 4 5))
→ ((lambda (z) (if (< 1 10) (+ 1 5) z)) 9)
→ (if (< 1 10) (+ 1 5) 9)
→ (if true (+ 1 5) 9)
→ (+ 1 5)
→ 6
```
□

2. (V. sub. 1) Fie definiția și aplicația funcției Haskell de mai jos. Descrieți **pas cu pas** cum decurge evaluarea aplicației, utilizând modelul bazat pe substituție textuală.

```
f x y z = if x < 10 then x + y else z

> f (0 + 1) (2 + 3) (4 + 5)
```

Soluție. .

```
f (0 + 1) (2 + 3) (4 + 5)
→ if (0 + 1) < 10 then (0 + 1) + (2 + 3) else (4 + 5)
→ if 1 < 10 then 1 + (2 + 3) else (4 + 5)
→ if True then 1 + (2 + 3) else (4 + 5)
→ 1 + (2 + 3)
→ 1 + 5
→ 6
```
□

3. Fie următoarea funcție Racket:

```
(define (f x)
  (if (<= x 0) 0
      (- (f (- x 1)) 1))) ;  (f (- (f (- x 1)) 1))))
```

(a) Ce fel de recursivitate utilizează funcția?

(b) Ce fel de recursivitate utilizează funcția, dacă înlocuim ultima linie cu expresia comentată?

Soluție. .

(a) Pe stivă, întrucât se mai realizează operații după evaluarea aplicației recursive.

(b) Tot pe stivă, datorită aplicației recursive interne. □

4. Ce afișează prgramul Racket de mai jos?

```
1 (define (x x)
2   (let ([x x])
3     (x x)))
4
5 (x 0)
```

Soluție. .

Eroare, deoarece aparițiile lui `x` din linia 3 se referă la parametrul funcției, 0, care nu poate fi aplicat ca o funcție. □

5. Sintetizați tipul următoarei funcții Haskell:

```haskell
f u = map . u
```

Tipul operatorului de compunere este

```haskell
(.) :: (b -> c) -> (a -> b) -> a -> c
```

Soluție. .

```haskell
f :: d -> e
u :: d = a -> b
map :: (g -> h) -> [g] -> [h]
b = g -> h
c = [g] -> [h]
e = a -> c
f :: (a -> g -> h) -> a -> [g] -> [h]
```

□

6. Supraîncărcați în Haskell operatorul `(<=)` pe liste, astfel încât o listă este mai mică sau egală decât alta, dacă același lucru se poate afirma și despre sumele elementelor lor. Exemplu: `[1,2,10] <= [0, 1, 100, 3]`, întrucât `13 <= 104`.

Soluție. .

```haskell
instance (Num a, Ord a) => Ord [a] where
    xs <= ys = sum xs <= sum ys
```

□

7. Câte interpretări satisfac propozițiile α, respectiv β, în logica propozițională, unde $p_1, \ldots, p_{10}$ sunt propoziții simple?

$$\alpha = p_1 \vee \ldots \vee p_{10}$$
$$\beta = (p_1 \vee \ldots \vee p_5) \wedge (p_6 \vee \ldots \vee p_{10})$$

Soluție. .

Pentru n propoziții simple, există 2^n interpretări distincte.

Toate interpretările satisfac α, cu excepția celei care asociază Fals fiecărei propoziții simple. Numărul final este $2^{10} - 1$.

Similar se poate judeca și în cazul celor două paranteze din β. Numărul final este $(2^5 - 1)^2$. $\square$

8. Utilizând metoda rezoluției, demonstrați că:

$$\{\forall x.P(x) \vee \forall x.Q(x)\} \models \forall x.(P(x) \vee Q(x)).$$

Soluție. .

Pentru claritate, redenumim, variabilele, obținând:

$$\{\forall x.P(x) \vee \forall y.Q(y)\} \models \forall z.(P(z) \vee Q(z)).$$

Utilizăm reducerea la absurd, conform căreia adăugăm negația concluziei la mulțimea de propoziții: $\neg\forall z.(P(z) \vee Q(z)) \equiv \exists z.(\neg P(z) \wedge \neg Q(z))$. Astfel, în urma aplicării procedurii de transformare în forma normală conjunctivă, care presupune și înlocuirea variabilei cuantificate existențial z cu constanta c_z, se obțin clauzele $\{P(x), Q(y)\}, \{\neg P(c_z)\}, \{\neg Q(c_z)\}$. Aplicând rezoluția în raport cu substituția $\{x \leftarrow c_z, y \leftarrow c_z\}$, se obține clauza vidă. $\square$

9. Fie următorul program Prolog:

```
1 a(1). a(2).
2 b(1). b(2).
3
4 c(X, Y) :- a(X), b(Y).
5 c(3, 4).
6 c(4, 3).
```

Unde ar trebui plasat operatorul `cut`, astfel încât scopul următor să fie satisfăcut?

```
?- findall(_, c(X, Y), L), length(L, 4).
```

Soluție. .

```
c(X, Y) :- !, a(X), b(Y).
```

Astfel, se generează doar cele patru combinații între `a` și `b`, renunțându-se la definițiile din liniile 5 și 6. $\square$

10. PROBLEMA. Puteți utiliza oricare dintre cele două limbaje **funcționale** studiate.

    (a) Definiți reprezentarea unui arbore oarecare nevid, în care un nod poate avea oricâți copii.

    (b) Definiți o funcție care calculează înălțimea arborelui. Înălțimea unei frunze este 0. Utilizați cel puțin o funcțională.

    (c) Definiți o funcție care calculează numărul de frunze din arbore. Utilizați cel puțin o funcțională.

Soluție. .

Exemplificăm soluția în Haskell.

```haskell
data Tree a = Node a [Tree a]
    deriving (Eq, Show)

height :: Tree a -> Int
height (Node _ []) = 0
height (Node _ children) = 1 + maximum (map height children)

leaves :: Tree a -> Int
leaves (Node _ []) = 1
leaves (Node _ children) = sum (map leaves children)
```

□