Precizări:

- Fiecare subiect are câte 10p. **Punctajul NU se acordă în absența justificării răspunsului!**
- Este suficientă rezolvarea a **10** subiecte din cele 12 pentru nota maximă.
- Punctajul suplimentar reprezintă **bonus**, ce poate compensa punctajul de pe parcurs.

1. Îmbogățiți λ-expresia $(x\ \ x)$, astfel încât expresia finală să conțină exact 2 apariții **legate** ale variabilei x.

   Soluție. .

   $(\lambda x.x\ \ x)$ sau $(x\ \ \lambda x.x)$. □

2. (20p!) Precizați cum decurg, pas cu pas, evaluările expresiilor Racket de mai jos, utilizând modelul de evaluare bazat pe **substituție** textuală. Există vreo diferență? Dacă da, în ce constă?

   (a) (10p)

   ```
   1 (and (> (+ 1 2) 5)
   2      (> (+ 2 3) 5))
   ```

   (b) (10p)

   ```
   1 (define (f x y)
   2   (and (> x 5)
   3        (> y 5)))
   4 (f (+ 1 2) (+ 2 3))
   ```

   Soluție. .

   (a) `(and (> (+ 1 2) 5) (> (+ 2 3) 5))`
   → `(and (> 3 5) (> (+ 2 3) 5))`
   → `(and #f (> (+ 2 3) 5))`
   → `#f`

   (b) `(f (+ 1 2) (+ 2 3))`
   → `(f 3 (+ 2 3))`
   → `(f 3 5)`
   → `(and (> 3 5) (> 5 5))`
   → `(and #f (> 5 5))`
   → `#f` □

3. Funcțiile Racket de mai jos determină **ultimul** element al unei liste nevide în două moduri diferite:

   ```
   1 (define (last1 L)
   2   (if (null? (cdr L))
   3       (car L)
   4       (last1 (cdr L))))
   5
   6 (define last2 (compose car reverse))
   ```

Scrieți un avantaj al folosirii lui `last1` în locul lui `last2`. Se presupune că `reverse` este implementată eficient.

Soluție. .

`last2` necesită construcția listei inversate, în timp ce `last1`, nu. □

4. În Haskell, funcționala `map` se poate extinde pentru a opera cu funcții parțiale, care întorc `Nothing` dacă nu sunt definite într-un anumit punct, în forma

```
mapMaybe :: (a  -> Maybe b)
         -> [a] -> [b]
```

unde constructorul de tip `Maybe` este definit ca

```
data Maybe a = Nothing | Just a
```

Diferența față de funcționala `map` standard constă în posibilitatea funcționalei `mapMaybe` de a *renunța* la elementele pentru care funcția de aplicat întoarce `Nothing`. De exemplu, expresia de mai jos se va evalua la `[40, 60]`.

```
mapMaybe (\x -> if x == 5
                then Nothing
                else Just $ x * 10)
         [4, 5, 6]
```

Implementați funcționala `mapMaybe`, folosind **exclusiv funcționale**. Rezolvările care nu respectă condiția **NU** vor fi punctate!

Soluție. .

```
mapMaybe :: Eq b => (a -> Maybe b) -> [a] -> [b]
mapMaybe f = map (\(Just x) -> x) . filter (/= Nothing) . map f
```

Constrângerea `(Eq b)` este revendicată de operatorul `(/=)`, fiind specifică acestei implementări. □

5. La ce se evaluează expresia Racket de mai jos? Justificați!

```
(let ([x 1])
  (lambda (x)
    (let ([x x])
      x)))
```

Soluție. .

Corpul construcției `let` este o funcție, care va constitui și valorea exspresiei. `let`-ul interior leagă o variabilă locală la parametrul funcției, și o întoarce. Valoarea va fi deci funcția identitate. □

6. Sintetizați tipul funcției Haskell de mai jos. Justificați!

```
f x y = [x y, y]
```

Soluție. .

```
f :: a -> b -> c
x :: d -> e
a = d -> e
c = [e]
b = d
b = e
f :: (e -> e) -> e -> [e]
```
□

7. Fie reprezentarea în Haskell a unei matrice pătratice sub forma unei liste de linii. Mai jos, sunt ilustrate o matrice și reprezentarea ei:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow [[1,2],[3,4]]$$

Supraîncărcați operatorul `(+)` pentru a realiza adunarea matricelor reprezentate ca mai sus. Pentru aceasta, instanțiați clasa `Num`:

```
class Num a where
    (+) :: a -> a -> a
```

Hint:

```
zipWith :: (a  ->  b  ->  c)
        -> [a] -> [b] -> [c]
```

Soluție. .

```
{-# LANGUAGE FlexibleInstances #-}

instance Num a => Num [[a]] where
    (+) = zipWith (zipWith (+))
```
□

8. Scrieți o propoziție din logica propozițională care să admită **exact** 42 de interpretări distincte. Justificați!

Soluție. .

Nu există o astfel de expresie, deoarece numărul de interpretări este 2^n, unde n este numărul de propoziții simple. □

9. Utilizați rezoluția propozițională pentru a demonstra că

$$\{a \vee b, a \Rightarrow c, b \Rightarrow c\} \models c.$$

Soluție. .

Adăugând negația concluziei, $\neg c$, la setul de ipoteze, și transformând în forma clauzală, se obțin clauzele: $\{a, b\}, \{\neg a, c\}, \{\neg b, c\}, \{\neg c\}$. De aici, se poate obține cu ușurință clauza vidă. □

10. Transcrieți în Prolog propoziția:

$$\neg\exists x.(p(x) \wedge \neg q(x)).$$

Soluție. .

Propoziția se poate rescrie în forma $\forall x.(\neg p(x) \vee q(x))$, respectiv $\forall x.(p(x) \Rightarrow q(x))$. În Prolog, aceasta se poate reprezenta prin:

```
1 q(X) :- p(X).
```

□

11. Ce șir se va afla pe banda mașinii Markov după execuția algoritmului de mai jos, presupunând că șirul inițial este `abcbc`?

```
1 abc -> ca
2 cc ->
3 -> .
```

Soluție. .

abcbc -1-> cabc -1-> cca -2-> a

□