PARADIGME DE PROGRAMARE

Curs 12

Antrenament pentru testul grilă.

Funcționale

Se dă următorul program Racket:

```
(define L '(1 2 3 4 5 6 7 8))
(foldl (lambda (elem acc) (if (odd? elem) acc (cons elem acc))) '() L)
(foldr (lambda (elem acc) (if (even? elem) (cons elem acc) acc)) '() L)
(filter odd? L)
(map (lambda (x) (if (even? x) x)) L)
```

Aplicația cărei funcții se va evalua la lista '(2 4 6 8)?

(a) foldl
(b) filter
(c) foldr
(d) map

Funcționale

Se dă următorul program Racket:

```
(define L '(1 2 3 4 5 6 7 8))
(foldl (lambda (elem acc) (if (odd? elem) acc (cons elem acc))) '() L)
(foldr (lambda (elem acc) (if (even? elem) (cons elem acc) acc)) '() L)
(filter odd? L)
(map (lambda (x) (if (even? x) x)) L)
```

Aplicația cărei funcții se va evalua la lista '(2 4 6 8)?

(a) foldl
(b) filter
(c) foldr
(d) map

Funcționale

La ce se va evalua următoarea expresie Haskell?

```
filter id $ zipWith (<) [1, 2, 3, 4] [4, 3, 2, 1]
```

(a) [True, True, False, False]

(b) [True, True]

(c) [False, False]

(d) [False, False, True, True]

Funcționale

La ce se va evalua următoarea expresie Haskell?

```
filter id $ zipWith (<) [1, 2, 3, 4] [4, 3, 2, 1]
```

(a) [True, True, False, False]

(b) [True, True]

(c) [False, False]

(d) [False, False, True, True]

Funcționale

La ce se va evalua următoarea expresie Haskell?

```
take 4 $ iterate (\x -> (head x + 1) : x) [1]
```

(a) [[1], [2,1], [3,2,1], [4,3,2,1]]

(b) [[1], [2], [3], [4]]

(c) [[1], [1,2], [1,2,3], [1,2,3,4]]

(d) [1,2,3,4]

Funcționale

La ce se va evalua următoarea expresie Haskell?

```
take 4 $ iterate (\x -> (head x + 1) : x) [1]
```

(a) [[1], [2,1], [3,2,1], [4,3,2,1]]

(b) [[1], [2], [3], [4]]

(c) [[1], [1,2], [1,2,3], [1,2,3,4]]

(d) [1,2,3,4]

Funcționale

La ce se va evalua următoarea expresie Racket?

```
(foldl (lambda(x y) (x y (x y))) 2 (list * +))
```

(a) 16

(b) 8

(c) 2

(d) 6

Funcționale

La ce se va evalua următoarea expresie Racket?

```
(foldl (lambda(x y) (x y (x y))) 2 (list * +))
```

(a) 16

(b) 8

(c) 2

(d) 6

Funcționale

La ce se va evalua următoarea expresie Racket?

```
(apply map append '(((1 2)) ((3 4))))
```

(a) (1 3 2 4)

(b) ((1 3) (2 4))

(c) ((1 2 3 4))

(d) (1 2 3 4)

Funcționale

La ce se va evalua următoarea expresie Racket?

```
(apply map append '(((1 2)) ((3 4))))
```

(a) (1 3 2 4)

(b) ((1 3) (2 4))

(c) ((1 2 3 4))

(d) (1 2 3 4)

Domeniul de vizibilitate a variabilelor

Ce va afișa următorul program Racket?

```
(define x 1)
(define y 2)
(let* ([x 42] [y (+ y 1)] [z (+ x 1)])
      (+ x y z))
```

(a) Programul intră în buclă infinită

(b) 6

(c) 47

(d) 88

Domeniul de vizibilitate a variabilelor

Ce va afișa următorul program Racket?

```
(define x 1)
(define y 2)
(let* ([x 42] [y (+ y 1)] [z (+ x 1)])
      (+ x y z))
```

(a) Programul intră în buclă infinită

(b) 6

(c) 47

(d) 88

Domeniul de vizibilitate a variabilelor

Ce va afișa următorul program Racket?

```
(define x 2)
(let ([x 1] [y 2] [f (delay (lambda (y) (+ x y)))])
       (let ([x 5])
              ((force f) x)))
```

(a) 3

(b) 4

(c) 5

(d) 7

Domeniul de vizibilitate a variabilelor

Ce va afișa următorul program Racket?

```
(define x 2)
(let ([x 1] [y 2] [f (delay (lambda (y) (+ x y)))])
      (let ([x 5])
            ((force f) x)))
```

(a) 3

(b) 4

(c) 5

(d) 7

Închideri funcționale

Ce va întoarce următorul program Racket?

```
(define f (lambda () f))
(f)
```

(a) O eroare
(b) O promisiune
(c) O funcție
(d) Programul intră în buclă infinită

Închideri funcționale

Ce va întoarce următorul program Racket?

```
(define f (lambda () f))
(f)
```

(a) O eroare
(b) O promisiune
(c) O funcție
(d) Programul intră în buclă infinită

Închideri / promisiuni / funcții nestricte

Care dintre următoarele expresii va produce eroare în Racket?

(a) (or #f (/ 1 0) #t)
(b) (lambda (x) (car cdr))
(c) (delay (/ 1 0))
(d) (lambda () (/ 1 0))

Închideri / promisiuni / funcții nestricte

Care dintre următoarele expresii va produce eroare în Racket?

(a) (or #f (/ 1 0) #t)
(b) (lambda (x) (car cdr))
(c) (delay (/ 1 0))
(d) (lambda () (/ 1 0))

Închideri funcționale

Fie definițiile următoarelor funcții în Racket:

```
(define f (lambda (x) (f x)))
(define (g x) (lambda () (g x)))
```

Care este asemănarea/diferența dintre apelurile (f 10) și (g 10)?

(a) Ambele apeluri vor cicla la infinit
(b) Primul va cicla la infinit, iar al doilea va întoarce o funcție
(c) Primul va întoarce o funcție, iar al doilea va cicla la infinit
(d) Ambele apeluri vor întoarce o funcție

Închideri funcționale

Fie definițiile următoarelor funcții în Racket:

```
(define f (lambda (x) (f x)))
(define (g x) (lambda () (g x)))
```

Care este asemănarea/diferența dintre apelurile (f 10) și (g 10)?

(a) Ambele apeluri vor cicla la infinit

(b) Primul va cicla la infinit, iar al doilea va întoarce o funcție

(c) Primul va întoarce o funcție, iar al doilea va cicla la infinit

(d) Ambele apeluri vor întoarce o funcție

Promisiuni

Ce rezultat va întoarce următorul program Racket:

```
(define x 10)
(define pr (lambda (y) (delay (+ x y))))
(+ (force (pr 2)) (force (pr 3)) (force (pr 5)))
```

(a) 36

(b) 40

(c) Eroare

(d) 10

Promisiuni

Ce rezultat va întoarce următorul program Racket:

```
(define x 10)
(define pr (lambda (y) (delay (+ x y))))
(+ (force (pr 2)) (force (pr 3)) (force (pr 5)))
```

(a) 36

(b) 40

(c) Eroare

(d) 10

Promisiuni

Ce va afișa următorul program Racket:

```
(define x 10)
(define y (delay (+ x 10)))
(let ([x 20])
       (force y))
(define x 30)
(force y)
```

(a) 20 40

(b) 30 40

(c) 30 30

(d) 20 20

Promisiuni

Ce va afișa următorul program Racket:

```
(define x 10)
(define y (delay (+ x 10)))
(let ([x 20])
       (force y))
(define x 30)
(force y)
```

(a) 20 40

(b) 30 40

(c) 30 30

(d) 20 20

Evaluare

Care din următoarele expresii NU va produce eroare?

Racket: `(define x (/ (/ 10 2) (let ([z 0]) z)))`

Haskell: `let x = (10 `div` 2) / 0`

Prolog: `X = Y / 0, Y is 10 / 2.`

(a) Doar cea scrisă în Haskell
(b) Cele scrise în Prolog și Haskell
(c) Cele scrise în Racket și Haskell
(d) Cele scrise în Racket și Prolog

Evaluare

Care din următoarele expresii NU va produce eroare?

Racket: `(define x (/ (/ 10 2) (let ([z 0]) z)))`

Haskell: `let x = (10 'div' 2) / 0`

Prolog: `X = Y / 0, Y is 10 / 2.`

(a) Doar cea scrisă în Haskell

(b) Cele scrise în Prolog și Haskell

(c) Cele scrise în Racket și Haskell

(d) Cele scrise în Racket și Prolog

Secțiuni

Având programul următor în Haskell (prima linie poate fi ignorată):

```
{-# LANGUAGE FlexibleInstances #-}
instance (Eq a, Num a) => Eq (a -> a) where
      f == g = (f 1) == (g 1)
```

Ce va afișa apelul următor: map (== (+ 1)) [(+ 1), (+ 2), (/ 2), (* 2)]

(a) [True, False, False, False]

(b) [True, False, False, True]

(c) Eroare

(d) [False, False, False, False]

Secțiuni

Având programul următor în Haskell (prima linie poate fi ignorată):

```
{-# LANGUAGE FlexibleInstances #-}
instance (Eq a, Num a) => Eq (a -> a) where
      f == g = (f 1) == (g 1)
```

Ce va afișa apelul următor: map (== (+ 1)) [(+ 1), (+ 2), (/ 2), (* 2)]

(a) [True, False, False, False]

(b) [True, False, False, True]

(c) Eroare

(d) [False, False, False, False]

Secțiuni

Ce rezultat are codul Haskell de mai jos?

```
(+) ((10/) (-5)) $ (2-) 1
```

(a) 2.5

(b) -2.5

(c) 5

(d) -1

Secțiuni

Ce rezultat are codul Haskell de mai jos?

```
(+) ((10/) (-5)) $ (2-) 1
```

(a) 2.5

(b) -2.5

(c) 5

(d) -1

Punct și dolar

Fie următoarele definiții Haskell:

```
f1 = filter odd . map (+1)
f2 = map (+1) . filter even
f3 = filter odd $ map (+1)
f4 = map (+1) $ filter even
```

Care din următoarele afirmații este adevărată?

(a) f1 [2,5..10] == f2 [2,5..10]

(b) f1 [2,5..10] == f3 [2,5..10] și f2 [2,5..10] == f4 [2,5..10]

(c) f1 [2,5..10] == f2 [3,6..11] și f3 [2,5..10] == f4 [3,6..11]

(d) f1 [2,5..10] == f2 [1,4..9] și f3 [2,5..10] == f4 [1,4..9]

Punct și dolar

Fie următoarele definiții Haskell:

```
f1 = filter odd . map (+1)
f2 = map (+1) . filter even
f3 = filter odd $ map (+1)
f4 = map (+1) $ filter even
```

Care din următoarele afirmații este adevărată?

(a) f1 [2,5..10] == f2 [2,5..10]

(b) f1 [2,5..10] == f3 [2,5..10] și f2 [2,5..10] == f4 [2,5..10]

(c) f1 [2,5..10] == f2 [3,6..11] și f3 [2,5..10] == f4 [3,6..11]

(d) f1 [2,5..10] == f2 [1,4..9] și f3 [2,5..10] == f4 [1,4..9]

TDA

Se dă tipul de date Haskell:

```
data MyUnion = UInt Int | UFloat Float | Complex Float Float deriving
(Show, Eq)
```

Care din următoarele expresii va genera o eroare la compilare?

(a) x = UInt 42

(b) x = UFloat 42.0

(c) x = MyUnion

(d) x = Complex 42.0

TDA

Se dă tipul de date Haskell:

```
data MyUnion = UInt Int | UFloat Float | Complex Float Float deriving
(Show, Eq)
```

Care din următoarele expresii va genera o eroare la compilare?

(a) x = UInt 42

(b) x = UFloat 42.0

(c) x = MyUnion

(d) x = Complex 42.0

Scheme de tip

Ce tip are următoarea funcție în Haskell?

```
f x y z = fst (z, [x, y])
```

(a) a -> b -> c -> c

(b) a -> a -> a -> a

(c) a -> a -> b -> b

(d) a -> b -> a -> b

Scheme de tip

Ce tip are următoarea funcție în Haskell?

```
f x y z = fst (z, [x, y])
```

(a) a -> b -> c -> c

(b) a -> a -> a -> a

(c) a -> a -> b -> b

(d) a -> b -> a -> b

Scheme de tip

Ce tip are expresia Add . Inv?

```
data Point = Point Int Int | Add Point Point | Inv Point
```

(a) Point -> Point

(b) Point -> Point -> Point

(c) Int -> Int - > Point

(d) Eroare de sinteză de tip

Scheme de tip

Ce tip are expresia Add . Inv?

```
data Point = Point Int Int | Add Point Point | Inv Point
```

(a) Point -> Point

(b) Point -> Point -> Point

(c) Int -> Int - > Point

(d) Eroare de sinteză de tip

Scheme de tip

Care este tipul următoarei expresii în Haskell?

```
map (+) [1,2,3]
```

(a) Num a => [a -> a]

(b) Num a => [a]

(c) Num a => a

(d) Eroare de sinteză de tip

Scheme de tip

Care este tipul următoarei expresii în Haskell?

```
map (+) [1,2,3]
```

(a) Num a => [a -> a]

(b) Num a => [a]

(c) Num a => a

(d) Eroare de sinteză de tip

Scheme de tip

Care este tipul următoarei funcții în Haskell?

```
f x y z = x . y $ z
```

(a) (b -> c) -> (s -> b) -> s -> c

(b) Eroare de sinteză de tip

(c) (b -> c) -> (b -> c) -> b -> c

(d) (b -> s) -> (c -> b) -> s -> c

Scheme de tip

Care este tipul următoarei funcții în Haskell?

```
f x y z = x . y $ z
```

(a) (b -> c) -> (s -> b) -> s -> c

(b) Eroare de sinteză de tip

(c) (b -> c) -> (b -> c) -> b -> c

(d) (b -> s) -> (c -> b) -> s -> c

Scheme de tip

Care este tipul următoarei funcții în Haskell?

```
f x y z = map x . y $ z
```

(a) (c -> b) -> (d -> a -> c) -> d -> [a] -> [b]

(b) (c -> a -> b) -> (d -> c) -> d -> [a] -> [b]

(c) (c -> b) -> (a -> c) -> [a] -> [b]

(d) (a -> b) -> (c -> [a]) -> c -> [b]

Scheme de tip

Care este tipul următoarei funcții în Haskell?

```
f x y z = map x . y $ z
```

(a) (c -> b) -> (d -> a -> c) -> d -> [a] -> [b]

(b) (c -> a -> b) -> (d -> c) -> d -> [a] -> [b]

(c) (c -> b) -> (a -> c) -> [a] -> [b]

(d) (a -> b) -> (c -> [a]) -> c -> [b]

Polimorfism și clase

Care este semnificația codului următor?

```
class Num a => Fractional a where
```

(a) Fractional a este un tip de date care aparține clasei Num a

(b) Clasa Num a poate fi instanțiată numai de tipuri de numere fracționare

(c) Dacă tipul a este membru al clasei Num, atunci el este și membru al clasei Fractional

(d) Pentru ca tipul a să fie membru al clasei Fractional, el trebuie să fie membru al clasei Num

Polimorfism și clase

Care este semnificația codului următor?

```
class Num a => Fractional a where
```

(a) Fractional a este un tip de date care aparține clasei Num a

(b) Clasa Num a poate fi instanțiată numai de tipuri de numere fracționare

(c) Dacă tipul a este membru al clasei Num, atunci el este și membru al clasei Fractional

(d) Pentru ca tipul a să fie membru al clasei Fractional, el trebuie să fie membru al clasei Num

Polimorfism și clase

Ce fel de polimorfism utilizează următoarea funcție în Haskell?

```
f :: Ord a => a -> a -> b -> b -> b
f x y a b = if x < y then a else b
```

(a) Exclusiv parametric

(b) Exclusiv ad-hoc

(c) Atât parametric cât și ad-hoc

(d) Funcția nu este polimorfică

Polimorfism și clase

Ce fel de polimorfism utilizează următoarea funcție în Haskell?

```
f :: Ord a => a -> a -> b -> b -> b
f x y a b = if x < y then a else b
```

(a) Exclusiv parametric

(b) Exclusiv ad-hoc

(c) Atât parametric cât și ad-hoc

(d) Funcția nu este polimorfică

Ce întoarce codul?

Care din următoarele expresii Haskell generează lista numerelor impare de la 1 la 10?

(a) `[ x | x <- [1 ..], x < 11, x `mod` 2 == 1 ]`

(b) `[ x | x <- [1 .. 10], not $ even x ]`

(c) `[ odd x | x <- [1 .. 10] ]`

(d) `[ x | x > 0, x < 11, odd x ]`

Ce întoarce codul?

Care din următoarele expresii Haskell generează lista numerelor impare de la 1 la 10?

(a) `[ x | x <- [1 ..], x < 11, x `mod` 2 == 1 ]`

(b) `[ x | x <- [1 .. 10], not $ even x ]`

(c) `[ odd x | x <- [1 .. 10] ]`

(d) `[ x | x > 0, x < 11, odd x ]`

Ce afișează codul?

Ce afișează (prin funcția display) codul de mai jos? (fără rezultatul final)

```
(define (f n)
(cond [(< n 2) 1]
      [else (display n) (+ (f (- n 2)) (f (- n 4)))]))
(f 8)
```

(a) 8 6 4 2 2 4 2

(b) 8 6 6 4 4 4 2 2 2 2

(c) 8 6 4 4 2 2 2

(d) 8 6 4 2 4 2 2

Ce afișează codul?

Ce afișează (prin funcția display) codul de mai jos? (fără rezultatul final)

```
(define (f n)
(cond [(< n 2) 1]
      [else (display n) (+ (f (- n 2)) (f (- n 4)))]))
(f 8)
```

(a) 8 6 4 2 2 4 2

(b) 8 6 6 4 4 4 2 2 2 2

(c) 8 6 4 4 2 2 2

(d) 8 6 4 2 4 2 2

Unificare

Câte soluții va avea următoarea interogare în Prolog?

```
append([X | Y], Z, [1, 2, 3]).
```

(a) O infinitate

(b) 4

(c) 1

(d) 3

Unificare

Câte soluții va avea următoarea interogare în Prolog?

```
append([X | Y], Z, [1, 2, 3]).
```

(a) O infinitate

(b) 4

(c) 1

(d) 3

Unificare și instanțiere

Fie următoarele definiții în Prolog pentru funcția de maxim:

```
max(X, Y, X):- X > Y, !.
max(_, Y, Y).
```

Ce afișează următoarele două interogări?

max(3, 2, 2).
max(2, 3, 2).

(a) false și false

(b) true și true

(c) false și true

(d) true și false

Unificare și instanțiere

Fie următoarele definiții în Prolog pentru funcția de maxim:

```
max(X, Y, X):- X > Y, !.
max(_, Y, Y).
```

Ce afișează următoarele două interogări?

max(3, 2, 2).
max(2, 3, 2).

(a) false și false

(b) true și true

(c) false și true

(d) true și false

Unificare și instanțiere

Câte soluții întoarce Prolog pentru interogarea următoare și ce valori are variabila Z?

```
member(X, [1,2,3]), Y == X, !, Z is X + Y.
```

(a) Trei: 2, 4, 6

(b) Zero (eșuează)

(c) Una: 2

(d) Nouă: 2, 3, 4, 3, 4, 5, 4, 5, 6

Unificare și instanțiere

Câte soluții întoarce Prolog pentru interogarea următoare și ce valori are variabila Z?

```
member(X, [1,2,3]), Y == X, !, Z is X + Y.
```

(a) Trei: 2, 4, 6

(b) Zero (eșuează)

(c) Una: 2

(d) Nouă: 2, 3, 4, 3, 4, 5, 4, 5, 6

Unificare și instanțiere

Care dintre următoarele două interogări vor fi satisfăcute (vor întoarce true)?

?- X == Y, X = Y.

?- X = Y, X == Y.

(a) A doua

(b) Ambele

(c) Niciuna

(d) Prima

Unificare și instanțiere

Care dintre următoarele două interogări vor fi satisfăcute (vor întoarce true)?

?- X == Y, X = Y.

?- X = Y, X == Y.

(a) A doua

(b) Ambele

(c) Niciuna

(d) Prima

Cut

Care este diferența dintre:

```
p :- a, b.
p :- c.
```

și

```
p :- a, !, b.
p :- c.
```

(a) Niciuna

(b) Primul este p = a ∧ b ∧ c, al doilea este p = a ∧ (b ∨ c)

(c) Primul este p = (a ∧ b) ∨ c, al doilea este p = (a ∧ b) ∨ (¬a ∧ c)

(d) Primul este p = (a ∧ b) ∨ c, al doilea este p = (a ∧ b) ∨ (b ∧ c)

Cut

Care este diferența dintre:

```
p :- a, b.
p :- c.
```

și

```
p :- a, !, b.
p :- c.
```

(a) Niciuna

(b) Primul este p = a ∧ b ∧ c, al doilea este p = a ∧ (b ∨ c)

(c) Primul este p = (a ∧ b) ∨ c, al doilea este p = (a ∧ b) ∨ (¬a ∧ c)

(d) Primul este p = (a ∧ b) ∨ c, al doilea este p = (a ∧ b) ∨ (b ∧ c)

Cut

Considerând următoarele fapte Prolog:

```
p1(1). p1(2). p1(3). p1(4).
p2(2). p2(3).
```

Ce va afișa (exceptând false) interogarea

p1(X), p2(Y), write(X), write(Y), X >= Y, !, fail.

(a) 123223323423

(b) 1122

(c) 12322

(d) 121322

Cut

Considerând următoarele fapte Prolog:

```
p1(1). p1(2). p1(3). p1(4).
p2(2). p2(3).
```

Ce va afișa (exceptând false) interogarea

p1(X), p2(Y), write(X), write(Y), X >= Y, !, fail.

(a) 123223323423

(b) 1122

(c) 12322

(d) 121322

Cut

Cu ce se va solda încercarea de satisfacere a scopului p(X)?

```
p(X) :- !, q(X), r(X).
p(1). p(2).
q(3). q(4).
r(5).
```

(a) X = 1; X = 2.

(b) X = 1.

(c) false.

(d) X = 2.

Cut

Cu ce se va solda încercarea de satisfacere a scopului p(X)?

```
p(X) :- !, q(X), r(X).
p(1). p(2).
q(3). q(4).
r(5).
```

(a) X = 1; X = 2.

(b) X = 1.

(c) false.

(d) X = 2.

Cut

Cu ce se va solda încercarea de satisfacere a scopului p(X)?

```
p(X) :- !, q(X), r(X).
q(1). q(2).
r(1). r(2).
```

(a) X = 1; X = 2.

(b) X = 1.

(c) false.

(d) X = 2.

Cut

Cu ce se va solda încercarea de satisfacere a scopului p(X)?

```
p(X) :- !, q(X), r(X).
q(1). q(2).
r(1). r(2).
```

(a) X = 1; X = 2.

(b) X = 1.

(c) false.

(d) X = 2.

Negația ca eșec

Câte rezultate va avea următoarea interogare? (de câte ori va fi satisfăcută?)

?- L1 = [1,2,3,4,5], L2 = [1,3,5], \+ (member(X, L2), \+ member(X, L1)).

(a) Niciodată

(b) O dată

(c) De două ori

(d) De trei ori

Negația ca eșec

Câte rezultate va avea următoarea interogare? (de câte ori va fi satisfăcută?)

?- L1 = [1,2,3,4,5], L2 = [1,3,5], \+ (member(X, L2), \+ member(X, L1)).

(a) Niciodată

(b) O dată

(c) De două ori

(d) De trei ori

Metapredicate

Fie următoarele fapte în cadrul unui program Prolog:

```
p(1). p(2). p(4).
q(1). q(4). q(8).
```

Ce va afișa următoarea interogare?

?- setof(X/Y, (p(X), q(Y), not(q(X))), L).

(a) L = [2/1, 2/4, 2/8].

(b) L = [2.0, 0.5, 0.25].

(c) L = [1/4, 1/8, 2/1, 2/4, 2/8, 4/1, 4/8].

(d) L = [0.25, 0.125, 2.0, 0.5, 4.0].

Metapredicate

Fie următoarele fapte în cadrul unui program Prolog:

```
p(1). p(2). p(4).
q(1). q(4). q(8).
```

Ce va afișa următoarea interogare?

?- setof(X/Y, (p(X), q(Y), not(q(X))), L).

(a) L = [2/1, 2/4, 2/8].

(b) L = [2.0, 0.5, 0.25].

(c) L = [1/4, 1/8, 2/1, 2/4, 2/8, 4/1, 4/8].

(d) L = [0.25, 0.125, 2.0, 0.5, 4.0].

Metapredicate

Fie următoarele fapte și reguli în cadrul unui program Prolog:

```
foo(2, 3, 1). foo(2, 3, 6).
foo(1, 2, 3). foo(1, 3, 4). foo(1, 2, 5).
p(X) :- bagof(C, (foo(A, B, C), C mod 2 > 0), [X, Y]).
```

Ce va afișa interogarea p(X)?

(a) X = 3; X = 5.

(b) X = 1; X = 3.

(c) X = 3.

(d) X = 1.

Metapredicate

Fie următoarele fapte și reguli în cadrul unui program Prolog:

```
foo(2, 3, 1). foo(2, 3, 6).
foo(1, 2, 3). foo(1, 3, 4). foo(1, 2, 5).
p(X) :- bagof(C, (foo(A, B, C), C mod 2 > 0), [X, Y]).
```

Ce va afișa interogarea p(X)?

(a) X = 3; X = 5.

(b) X = 1; X = 3.

(c) X = 3.

(d) X = 1.

Metapredicate

Fie următoarele fapte în cadrul unui program Prolog:

```
a(1). b(1).
a(2). b(2).
a(3). b(3).
```

Ce va afișa următoarea interogare?

?- findall(X/Y, (a(X), !, b(Y); (a(Y), b(X))), L).

(a) L = [1/1, 2/2, 3/3].

(b) L = [1/1, 2/2, 3/3, 1/1, 2/2, 3/3].

(c) L = [1/2, 1/3, 2/1, 2/3, 3/1, 3/2].

(d) L = [1/1, 1/2, 1/3].

Metapredicate

Fie următoarele fapte în cadrul unui program Prolog:

```
a(1). b(1).
a(2). b(2).
a(3). b(3).
```

Ce va afișa următoarea interogare?

?- findall(X/Y, (a(X), !, b(Y); (a(Y), b(X))), L).

(a) L = [1/1, 2/2, 3/3].

(b) L = [1/1, 2/2, 3/3, 1/1, 2/2, 3/3].

(c) L = [1/2, 1/3, 2/1, 2/3, 3/1, 3/2].

(d) L = [1/1, 1/2, 1/3].