Programowanie L

Egzamin zasadniczy

17 czerwca 2013

Liczba punktów	Ocena
0 – 14	2.0
15 – 17	3.0
18 – 20	3.5
21 – 23	4.0
24 – 26	4.5
27 – 30	5.0

W każdym pytaniu proszę wyraźnie zaznaczyć dokładnie jedną odpowiedź. Jeśli zostanie zaznaczona więcej niż jedna odpowiedź, to za wybraną zostanie uznana ta, która *nie jest* otoczona kółkiem. Czas trwania egzaminu: 120 minut.

Pytanie 1. Rozważmy następujący program prologowy:

```
p(a,b).
p(b,c).
p(c,a).
q(X,X).
q(X,Y) :-
   p(X,Z),
   q(Z,Y).
```

i zapytanie

```
?- q(X,X).
```

☐ a. Powyższy cel zapętli się.

☐ b. Obliczenie tego celu zakończy się niepowodzeniem.

☐ c. Powyższy cel jest spełniony na nieskończenie wiele sposobów, a pierwszym z nich jest podstawienie `X = a`.

■ d. Powyższy cel jest spełniony na nieskończenie wiele sposobów, ale żadnym z nich nie jest postawienie `X = b`.

Pytanie 2. Rozważmy następujący program prologowy:

```
p(a).
p(b).
q(c).
```

i zapytanie

```
?- \+ \+ p(X), q(X).
```

☐ a. Powyższy cel zawiedzie.

■ b. Powyższy cel jest spełniony na jeden sposób.

☐ c. Powyższy cel jest spełniony na dwa sposoby.

☐ d. Powyższy cel jest spełniony na nieskończenie wiele sposobów.

Pytanie 3. Rozważmy następujący cel w Prologu:

```
?- \+ X \= 1, Y is 2 * X.
```

☐ a. Powyższy cel zakończy się niepowodzeniem.

☐ b. Powyższy cel zakończy się sukcesem, w którym `X=1` i `Y=2`.

☐ c. Powyższy cel zapętli się.

■ d. Powyższy cel zakończy się błędem wykonania.

Pytanie 4. Ktoś twierdzi, że:

- Jeśli coś się porusza, to jest zwierzęciem.
- Człowiek się porusza.
- Jeśli coś się porusza i nie jest ani człowiekiem, ani zwierzęciem, to jest UFO.
- Wierzbicki jest człowiekiem.

Powyższe informacje możemy zapisać w postaci klauzul prologowych:

```
zwierze(X) :- rusza_sie(X).
rusza_sie(X) :- czlowiek(X).
ufo(X) :- rusza_sie(X), \+ czlowiek(X), \+ zwierze(X).
czlowiek(wierzbicki).
```

Jaki będzie efekt obliczenia celu:

```
?- ufo(X).
```

■ a. Powyższy cel zawiedzie (znaczy: UFO nie istnieje).

☐ b. Powyższy cel zakończy się sukcesem, w którym `X` pozostanie nieukonkretnioną zmienną (znaczy: UFO jest wszędzie).

☐ c. Powyższy cel zakończy się sukcesem, w którym `X = wierzbicki` (znaczy: Wierzbicki to UFO).

☐ d. Powyższy cel zapętli się (znaczy: nie należy zadawać głupich pytań).

Pytanie 5. Przypuśćmy, że predykat `p/1` nie powoduje żadnych skutków ubocznych i że cel `p(X)` jest spełniony na nieskończenie wiele sposobów. Wtedy cel `p(X),p(X)`

☐ a. jest spełniony na nieskończenie wiele sposobów,

■ b. jest spełniony na co najmniej jeden sposób, ale po nawrocie z pierwszego sukcesu może się zapętlić,

☐ c. może zawieść,

☐ d. działa tak samo, jak cel `p(X)`.

Pytanie 6. Niech

```
q(X) :-
   !,
   X=1.
```

Cel

```
?- q(Y), fail.
```

■ a. zawiedzie,

☐ b. zakończy się sukcesem, w którym `Y` pozostanie nieukonkretnioną zmienną,

☐ c. zakończy się sukcesem, w którym `Y=1`,

☐ d. zapętli się.

Pytanie 7. Niech

```
append([],X,X).
append([H|T],X,[H|S]) :-
   append(T,X,S).
```

Cel

```
?- append(X,[],X).
```

☐ a. zawiedzie,

☐ b. zapętli się,

☐ c. będzie spełniony na jeden sposób,

■ d. będzie spełniony na nieskończenie wiele sposobów.

Pytanie 8. Niech

```
len([],0).
len([_|T],N+1) :-
   len(T,N).
```

Cel

```
?- len([adam,jan|piotr],N).
```

■ a. zawiedzie,

☐ b. zakończy się sukcesem, w którym `N=3`,

☐ c. zakończy się sukcesem, w którym `N=((0+1)+1)+1`,

☐ d. zapętli się.

Pytanie 9. Obliczenie celu

```
?- X is 1, X is 2.
```

■ a. będzie miało taki sam efekt, jak obliczenie celu

```
?- X = 1, X = 2.
```

☐ b. zakończy się pojedynczym sukcesem,

☐ c. zakończy się dwoma sukcesami,

☐ d. zakończy się błędem arytmetycznym.

Pytanie 10. Rozważmy taki predykat `p/0`, że obliczenie celu `p` daje dokładnie dwa sukcesy. Wówczas obliczenie celu

```
?- p, p.
```

☐ a. da dokładnie dwa sukcesy,

■ b. da co najwyżej cztery sukcesy,

☐ c. zawiedzie,

☐ d. zapętli się.

Pytanie 11. Obliczenie celu

```
?- [H|_] = .(jan,adam).
```

☐ a. kończy się niepowodzeniem, ponieważ term po prawej stronie nie jest listą,

☐ b. kończy się dwoma sukcesami, w których, kolejno, `H=jan` i `H=adam`,

■ c. kończy się pojedynczym sukcesem,

☐ d. zapętla się.

Pytanie 12. Niech

```
p :- p.
p.
```

Cel

```
?- p.
```

☐ a. zawiedzie,

☐ b. będzie spełniony na jeden sposób,

☐ c. będzie spełniony na nieskończenie wiele sposobów,

■ d. zapętli się.

Pytanie 13. Niech

```
p :- !, p.
```

Cel

```
?- p.
```

☐ a. zawiedzie,

☐ b. będzie spełniony na jeden sposób,

☐ c. będzie spełniony na nieskończenie wiele sposobów,

■ d. zapętli się.

Pytanie 14. Rozważmy predykat

```
x(0) --> "".
x(N+1) --> "a", x(N), "a".
```

Obliczenie celu

```
?- x(X,Y,Z).
```

☐ a. kończy się błędem „Undefined procedure x/3. However, there are definitions for x/1".

☐ b. kończy się niepowodzeniem.

☐ c. kończy się pojedynczym sukcesem.

■ d. ma nieskończenie wiele sukcesów.

Pytanie 15. Rozważmy predykat

```
eq(f(X),f(Y)) :- !, eq(X,Y).
eq(X,X).
```

☐ a. Predykat eq/2 działa tak samo, jak predykat =/2.

■ b. Istnieją termy t_1 i t_2, dla których cel eq(t_1, t_2) zapętla się.

☐ c. Istnieją termy t_1 i t_2, dla których obliczenie celu eq(t_1, t_2) kończy się więcej niż jednym sukcesem.

☐ d. Obliczenie celu `eq(f(a),f(b))` kończy się sukcesem.

Pytanie 16. Niech

```
f, g :: () → ()
f () = ()
g _ = ()
```

i, jak zwykle, $(f \cdot g)x = f(g\ x)$. Wtedy:

☐ a. g f = (),

☐ b. f · g = f,

■ c. $\mathtt{g} \cdot \mathtt{f} = \mathtt{f} \cdot \mathtt{g}$,

☐ d. żadna z powyższych równości nie zachodzi.

Pytanie 17. Niech

```
times :: Integer → Integer → Integer
0 `times` _ = 0
n `times` m = n*m
```

Wtedy:

☐ a. `foldl times 1` $=$ `foldr times 1`,

☐ b. `foldl (flip times) 1` $=$ `foldr times 1`,

☐ c. `foldl times 1` $=$ `foldr (flip times) 1`,

■ d. żadna z powyższych równości nie zachodzi.

Pytanie 18. Równość `tail` $xs = xs$ zachodzi:

☐ a. dla pewnej listy skończonej xs,

☐ b. dla pewnej listy częściowej xs,

■ c. dla pewnej listy nieskończonej xs,

☐ d. nie zachodzi dla żadnej listy.

Pytanie 19. Funkcja f jest *strict*, jeśli

☐ a. $f \perp \neq f\ x$ dla pewnego $x \neq \perp$,

☐ b. $f \perp \neq f\ x$ dla każdego $x \neq \perp$,

☐ c. $f \perp \neq \perp$,

■ d. $f \perp = \perp$.

Pytanie 20. Typem wyrażenia `head []` jest

■ a. `a`

☐ b. `[a]`

☐ c. `undefined`

☐ d. `[a] => a`

Pytanie 21. Niech e :: `Monad` $m \Rightarrow m$ `Integer`. Wyrażenie

$$e \texttt{ >>= } (\lambda\, x \rightarrow \texttt{return \$ } (2 * x))$$

jest równe

☐ a. e `>>=` (`return $` $(2*)$)

■ b. e `>>=` (`return` $\cdot\ (2*)$)

☐ c. e `>>=` $(2*)$

☐ d. e `>>=` (`return` $(2*)$)

Pytanie 22. Niech

```
data Kolor = Czerwony | Zielony
```

Wtedy jeśli dla pewnej funkcji f :: `Kolor` $\to$ `Kolor` jest:

☐ a. f `Czerwony` $\neq \bot$ i f `Zielony` $\neq \bot$, to f jest *strict*.

■ b. f `Czerwony` $\neq f$ `Zielony`, to f jest *strict*.

☐ c. $f \bot \neq \bot$, to f jest *strict*.

☐ d. $f \bot \neq$ `Czerwony` lub $f \bot \neq$ `Zielony`, to f jest *strict*.

Pytanie 23. Niech

```
f x y = y (f x)
```

Najogólniejszym typem funkcji `f` jest:

☐ a. $a \to b \to a$,

☐ b. $a \to (a \to b) \to b$,

☐ c. $a \to (b \to a) \to a$,

■ d. Funkcja `f` nie posiada typu.

Pytanie 24. Niech e, f :: `Monad` $m \Rightarrow m$ `Integer`. Wyrażenie

$$\texttt{do}\ \{\ x \leftarrow e\texttt{;}\ f\texttt{;}\ \texttt{return}\ x\ \}$$

jest równe:

☐ a. e >>= f

☐ b. do $\{\ e\texttt{;}\ f\ \}$

■ c. e >>= $(\lambda\ x \to$ do $\{\ f\texttt{;}\ \texttt{return}\ x\ \})$

☐ d. e >>= $(\lambda\ x \to f$ >>= `return` $x)$

Pytanie 25.

☐ a. Istnieją 4 różne wartości typu `Bool` $\to$ `Bool`, spośród których 3 daje się zdefiniować w Haskellu.

☐ b. Istnieje 16 różnych wartości typu `Bool` $\to$ `Bool`, spośród których 9 daje się zdefiniować w Haskellu.

■ c. Istnieje 27 różnych wartości typu `Bool` $\to$ `Bool`, spośród których 11 daje się zdefiniować w Haskellu.

☐ d. Istnieje 27 różnych wartości typu `Bool` $\to$ `Bool`, spośród których 16 daje się zdefiniować w Haskellu.

Pytanie 26. Niech

```
or1, or2 :: Bool → Bool → Bool
 True `or1` True  = True
 True `or1` False = True
False `or1` True  = True
False `or1` False = False
 True `or2` _     = True
    _ `or2` True  = True
False `or2` False = False
```

Wtedy

☐ a. `or1` = `or2`

☐ b. (`or1`) = (`or2`)

☐ c. `foldl or1 True` = `foldl or2 True`

☐ d. `foldr or1 True` = `foldr or2 True`

■ e. wszystkie powyższe odpowiedzi są fałszywe

Pytanie 27. Niech

```
data Tree a = Node a [Tree a]
inftree :: Tree Integer
inftree = head ts where
   ts = map (λ (Node n rs) -> Node (n+1) (tail rs)) (Node 0 ts : ts)
firstSons, sonsOfRoot :: Tree a → [a]
firstSons (Node x xs) = x : firstSons (head xs)
sonsOfRoot (Node x xs) = map (λ (Node y _) → y) xs
zeros :: [Integer]
zeros = 0 : zeros
```

Wtedy:

■ a. `firstSons inftree` = `sonsOfRoot inftree`

☐ b. `firstSons inftree` = `[1..]`

☐ c. `firstSons inftree` = `zeros`

☐ d. `sonsOfRoot inftree` = `[1..]`

Pytanie 28. Wartością wyrażenia

```
[ (m,n) | m ← [1,2], n ← [3,4] ]
```

jest

☐ a. `[ (1,3), (2,4) ]`,

☐ b. `( [1,2], [3,4] )`,

■ c. `[ (1,3), (1,4), (2,3), (2,4) ]`,

☐ d. `[ (1,3), (2,3), (1,4), (2,4) ]`.

Pytanie 29. Niech

```
xs :: [Integer]
xs = 1 : map (2*) xs
```

Wówczas `xs`

☐ a. zawiera błąd typowy,

☐ b. nie posiada wartości (próba jego obliczenia kończy się zapętleniem),

☐ c. ma taką samą wartość, jak wyrażenie `map (2*) [1..]`, ale jest mniej efektywne,

■ d. ma taką samą wartość, jak wyrażenie `map (2^) [0..]`, ale jest bardziej efektywne.

Pytanie 30. Dla żadnego typu instancja wyrażenia `return 'a' >>= return`

■ a. nie jest równa `'a'`,

☐ b. nie jest równa `"a"`,

☐ c. nie jest równa `return 'a'`,

☐ d. Wyrażenie nie posiada typu.