Programowanie L

Egzamin zasadniczy

29 maja 2012

Liczba punktów	Ocena
0 – 14	2.0
21 – 22	3.0
23 – 24	3.5
25 – 26	4.0
27 – 28	4.5
29 – 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 predykat:

```
p(X,X).
p(X,[a|Y]) :-
   p([b|X],Y).
```

i zapytanie

```
?- p([a,a,a], X).
```

☐ a. Maszyna prologowa wygeneruje odpowiedź `X = [a,a,a]`, a po nawrocie zapętli się.

■ b. Jedną z odpowiedzi będzie `X=[a,a,b,b,a,a,a]`.

☐ c. Maszyna prologowa wygeneruje cztery odpowiedzi, a po czwartym nawrocie zapętli się.

☐ d. Maszyna prologowa wygeneruje cztery odpowiedzi, a czwarty nawrót zakończy się niepowodzeniem.

Pytanie 2. Rozważmy predykat:

```
p([a|_]).
p([_|X]) :-
   p([a|X]).
```

i zapytanie

```
?- p(X).
```

☐ a. Jedną z odpowiedzi będzie podstawienie za `X` nieukonkretnionej zmiennej.

■ b. W każdej z odpowiedzi za `X` będzie podstawiona lista otwarta.

☐ c. W każdej z odpowiedzi za `X` będzie podstawiona lista, której głową jest atom `a`.

☐ d. Po pewnej liczbie nawrotów nastąpi niepowodzenie.

Pytanie 3. Rozważmy zapytanie

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

☐ a. Próbując spełnić ten cel maszyna prologowa zapętli się.

■ b. Jedyną odpowiedzią będzie `X = []`, a po nawrocie maszyna prologowa zapętli się.

☐ c. Jedyną odpowiedzią będzie `X = []`, a nawrót zakończy się niepowodzeniem.

☐ d. Jedną z odpowiedzi będzie lista cykliczna `X = [_|X]`.

Pytanie 4. Rozważmy zapytanie

```
?- append(X,Y,Z), reverse(X,Z).
```

☐ a. Jedną z odpowiedzi będzie `X = [_], Y = [], Z = [_]`.

☐ b. Próbując spełnić ten cel maszyna prologowa zapętli się.

☐ c. Próba spełnienia tego celu zakończy się niepowodzeniem.

■ d. Odpowiedzi będą takie same, jak na zapytanie

```
?- reverse(X,X), Y = [], Z = X.
```

Pytanie 5. Rozważmy zapytanie

```
?- X \= Y.
```

■ a. Powyższy cel zawiedzie.

☐ b. W kolejnych nawrotach za `X` i `Y` maszyna prologowa będzie podstawiać pary struktur, które się nie unifikują.

☐ c. Powyższy cel będzie spełniony na jeden sposób, a w odpowiedzi maszyna prologowa zunifikuje zmienne `X` i `Y`.

☐ d. Powyższy cel będzie spełniony na jeden sposób, a w odpowiedzi zmienne `X` i `Y` pozostaną nieukonkretnione.

Pytanie 6. Rozważmy predykaty

```
p(a).
p(b).

q([]).
q([X|Y]) :-
   p(X),
   q(Y).
```

i zapytanie

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

☐ a. Po skończonej liczbie nawrotów maszyna prologowa zapętli się.

☐ b. Po skończonej liczbie nawrotów wystąpi niepowodzenie.

☐ c. Wśród odpowiedzi pojawi się każda lista acykliczna, której elementami są atomy `a` i `b`.

■ d. Odpowiedzi nie zmienią się, jeśli z programu usuniemy klauzulę `p(b).`

Pytanie 7. Rozważmy predykaty

```
p(a).
p(f(X,Y)) :-
   p(X),
   p(Y).

f(a,a).
f(b,b).
```

i zapytanie

```
?- p(X).
```

☐ a. Jedną z odpowiedzi będzie podstawienie `X = f(f(a,a),f(a,a))`.

☐ b. Jedną z odpowiedzi będzie podstawienie `X = f(b,b)`.

☐ c. Jedną z odpowiedzi będzie podstawienie `X = b`.

■ d. Powyższy cel będzie spełniony na nieskończenie wiele sposobów.

Pytanie 8. Rozważmy zapytanie

```
?- \+ \+ member(X, [a]), X = b.
```

☐ a. Cel będzie spełniony na jeden sposób i odpowiedzią będzie `X = a`.

■ b. Cel będzie spełniony na jeden sposób i odpowiedzią będzie `X = b`.

☐ c. Próba spełnienia tego celu zakończy się niepowodzeniem.

☐ d. Próba spełnienia tego celu zakończy się zapętleniem.

Pytanie 9. Rozważmy gramatykę

```
x --> [1].
x --> x, [+], x.
```

i zapytanie

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

■ a. Odpowiedzi będzie nieskończenie wiele. Każde słowo z języka opisanego powyższą gramatyką zostanie podstawione za `X` w którejś z odpowiedzi.

☐ b. Odpowiedzi będzie nieskończenie wiele, ale nie wszystkie słowa z języka opisanego powyższą gramatyką zostaną zwrócone w odpowiedziach.

☐ c. Maszyna wygeneruje odpowiedź `X = [1]`, a po nawrocie zapętli się.

☐ d. Żadna z powyższych trzech odpowiedzi nie jest poprawna.

Pytanie 10. Rozważmy predykat

```
p([a]).
p([H|T]) :-
   !,
   p(T).
```

i zapytanie

```
?- p([b,b,a]).
```

☐ a. Wynikiem tego zapytania będzie niepowodzenie.

■ b. Wynikiem tego zapytania będzie sukces, a po nawrocie niepowodzenie.

☐ c. Usunięcie odcięcia z drugiej klauzuli spowoduje zmianę odpowiedzi na podane zapytanie.

☐ d. Kompilacja tego predykatu przebiegnie bez ostrzeżeń.

Pytanie 11. Rozważmy predykat

```
p([]).
p([a|X]) :-
  p(X),
  !.
```

i zapytanie

```
?- p(X).
```

☐ a. Odpowiedzi będzie nieskończenie wiele.

☐ b. Odcięcie w drugiej klauzuli jest „zielone", tj. usunięcie go nie zmieni zachowania predykatu (będzie on jedynie działał być może szybciej).

☐ c. Jedyną odpowiedzią będzie `X = []` i nie będzie nawrotów.

■ d. Jedną z odpowiedzi będzie `X = [a]`.

Pytanie 12. Rozważmy zapytanie

```
?- X + Y =:= 0.
```

(Predykat =:=/2 jest spełniony, gdy jego argumenty są równe w sensie arytmetycznym).

☐ a. Zapytanie zakończy się niepowodzeniem.

☐ b. Odpowiedź na to zapytanie będzie taka sama, jak na zapytanie

```
?- X + Y is 0.
```

■ c. Odpowiedź na to zapytanie będzie taka sama, jak na zapytanie

```
?- 0 is X + Y.
```

☐ d. W odpowiedzi pod `X` i `Y` zostaną podstawione przeciwne liczby całkowite.

Pytanie 13. Rozważmy predykat

```
p(N) :- N is 0.
p(N) :-
   p(N-1).
```

i zapytanie

```
?- p(-3).
```

☐ a. Obliczenie zakończy się sukcesem.

☐ b. Obliczenie zakończy się niepowodzeniem.

■ c. Obliczenie zapętli się.

☐ d. Obliczenie zakończy się błędem „Arguments are not sufficiently instantiated in arithmetic expression".

Pytanie 14. Rozważmy zapytanie

```
?- X is 4, Y is 2, X is 2*Y, Z is 2*X.
```

☐ a. Obliczenie zakończy się niepowodzeniem.

☐ b. Obliczenie zapętli się.

☐ c. Obliczenie zakończy się błędem.

■ d. Odpowiedzi na powyższe zapytanie i zapytanie

```
?- Y is 2, X is 4, Z is 2*X, X is 2*Y.
```

są takie same.

Pytanie 15. Rozważmy zapytanie

```
?- [[],[]|[]] = [[]|V].
```

☐ a. Obliczenie tego zapytania zakończy się niepowodzeniem.

☐ b. Wynikiem jest podstawienie `V = []`.

■ c. Wynikiem jest podstawienie `V = [[]]`.

☐ d. Wynikiem jest podstawienie `V = [[],[]]`.

Pytanie 16. Rozważmy funkcję

```
f :: Integer → Integer
f n = n * f $ n - 1
f 0 = 1
```

☐ a. Wartością wyrażenia `f 5` jest 120.

☐ b. Wartością wyrażenia `f 5` jest $\bot$.

☐ c. Obliczenie wyrażenia `f 5` zakończy się błędem.

■ d. Program jest niepoprawny.

Pytanie 17. Rozważmy program

```
f :: (Integer,Integer) → Integer → Integer
f (x,y) z
   | z>0 = z
   | otherwise = x*y
```

☐ a. Wartością wyrażenia `f undefined 42` jest 42.

■ b. Wartością wyrażenia `f undefined 42` jest $\bot$.

☐ c. Wartością wyrażenia `f undefined 42` jest $\bot * \bot$.

☐ d. Program jest niepoprawny.

Pytanie 18. Przyjmijmy, że

```
reverse :: [a] → [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

Równość

$$(\texttt{reverse . reverse})\ xs \ = \ xs$$

jest prawdziwa dla każdej

☐ a. listy $xs :: [a]$.

■ b. skończonej listy $xs :: [a]$.

☐ c. nieskończonej listy $xs :: [a]$.

☐ d. częściowej listy $xs :: [a]$.

Pytanie 19. Wyrażenie

```
undefined undefined
```

■ a. jest równe `undefined`.

☐ b. jest równe `λ _ → undefined`.

☐ c. jest różne od `undefined`.

☐ d. nie posiada typu.

Pytanie 20. Rozważmy program

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

Najogólniejszym typem funkcji `f` jest

☐ a. `(a → b) → a → b`.

☐ b. `((a → b) → a) → b → a`.

■ c. `(a → b) → ((a → b) → a) → b`.

☐ d. W definicji funkcji `f` występuje błąd typowy.

Pytanie 21. Rozważmy program

```
data T1 = C1^1 | C2^1 | ... | Cn^1
data T2 = C1^2 | C2^2 | ... | Cm^2
```

Typ `(T1,T2)` ma

☐ a. $n \cdot m$ wartości.

☐ b. $n \cdot m + 1$ wartości.

☐ c. $(n+1) \cdot (m+1)$ wartości.

■ d. $(n+1) \cdot (m+1) + 1$ wartości.

Pytanie 22. Niech

```
f :: Integer → Integer
f x = x
g x = λ y → y
h x y = f (g x y)
```

Wtedy

☐ a. `h = f . g`.

■ b. `h x = f . (g x)`.

☐ c. `h x y = (f . g) x y`.

☐ d. `h :: (Integer, Integer) -> Integer`.

Pytanie 23. Lista

```
[ x `div` 2 | x ← [1..100], x `mod` 3 > 0 ]
```

☐ a. ma 100 elementów.

■ b. ma 67 elementów.

☐ c. ma 51 elementów.

☐ d. jest równa liście `[1..50]`.

Pytanie 24. Niech

```
class C t where
   m :: t → t
```

Wtedy

☐ a. `m :: t → t`.

☐ b. `m :: C t → C t`.

☐ c. `m :: C => t → t`.

■ d. `m :: C t => t → t`.

Pytanie 25. Niech

```
a (x,y) = x
b (x,y) = y
f (x,y) z = (x z, y z)
g (x,y) = f (x . a, y . b)
h (x,y) (u,v) = (x u, y v)
```

Wtedy

☐ a. `g` = `h`.

■ b. `g, h :: (x → x', y → y') → (x,y) → (x',y')`.

☐ c. `g :: (x → x', y → y') → (x',y')`.

☐ d. definicja funkcji `g` zawiera błąd typowy.

Pytanie 26. Wyrażenie

```
do
   x ← [1,2,3]
   [x]
```

☐ a. ma typ `Monad m => m [Integer]`.

☐ b. nie posiada typu.

■ c. jest równe `[1,2,3]`.

☐ d. jest równe `[[1],[2],[3]]`.

Pytanie 27. Jednym z aksjomatów typu $t :: * \to *$ należącego do klasy `Monad` jest:

☐ a. $\forall m, n, p :: t\,a\ \ (m \texttt{ >>= } n) \texttt{ >>= } p = m \texttt{ >>= } (n \texttt{ >>= } p)$.

■ b. $\forall m :: t\,a\ \forall n :: a \to t\,b\ \forall p :: b \to t\,c\ \ (m \texttt{ >>= } n) \texttt{ >>= } p = m \texttt{ >>= } (\lambda x \to (n\ x) \texttt{ >>= } p)$.

☐ c. $\forall m :: t\,a\ \ \texttt{return} \texttt{ >>= } m = m$.

☐ d. żadna z powyższych równości.

Pytanie 28. Które z poniższych wyrażeń ma inną wartość niż pozostałe?

■ a. `[ 2,4..10 ]`.

☐ b. `[ 2*x | x ← [1..10], x 'mod' 2 == 0]`.

☐ c. `do { x ← [1..10]; if x 'mod' 2 == 0 then return (2*x) else [] }`

☐ d. `do { x ← [1..10]; guard$ x 'mod' 2 == 0; return$ 2*x }`.

Pytanie 29. Oto definicja funkcji `map`:

```
map :: (a → b) → [a] → [b]
map f [] = []
map f (x:xs) = f x : map f xs
```

Wyrażenie `map undefined []`

■ a. ma wartość `[]`.

☐ b. ma wartość $\bot$.

☐ c. ma wartość `[`$\bot$`]`.

☐ d. jest równe `map undefined undefined`.

Pytanie 30. Rozważmy funkcję

```
f xs@(x:_) = x : g xs where g (y:ys) = g (y:y:ys)
```

Jaka jest wartość wyrażenia `head $ f [0..]`?

■ a. `0`

☐ b. `[0]`

☐ c. `[]`

☐ d. obliczenie zapętli się.