Metody programowania

Egzamin poprawkowy

8 września 2015

Liczba punktów	Ocena
0 – 16	2.0
17 – 20	3.0
21 – 22	3.5
23 – 24	4.0
25 – 26	4.5
27 – 30	5.0

W każdym pytaniu testowym proszę wyraźnie zaznaczyć dokładnie jedną odpowiedź. Jeśli zostanie zaznaczona więcej niż jedna odpowiedź, to za wybraną zostanie uznana ta, która *nie jest* otoczona kółkiem. Każde pytanie jest warte 1 punkt. Czas trwania egzaminu: 120 minut.

Pytanie 1. Obliczenie celu

```
?- Jan = syn(Jan,Ewa).
```

a. ☐ zakończy się niepowodzeniem, bo żaden term nie może się zunifikować ze swoim właściwym podtermem.

b. ☐ zakończy się błędem, gdyż nikt nie może być swoim własnym ojcem.

c. ☐ zapętli się.

d. ■ zakończy się sukcesem.

Pytanie 2. Niech predykaty `p/1` i `q/1` będą zaprogramowane w czystym Prologu, tj. bez wykorzystania skutków ubocznych, odcięć itp. Wtedy jeśli cel

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

jest spełniony na nieskończenie wiele sposobów, to

a. ☐ cel `p(X)` jest spełniony na nieskończenie wiele sposobów.

b. ☐ cel `q(X)` jest spełniony na nieskończenie wiele sposobów.

c. ☐ dla żadnego termu t, który jest podstawiany pod zmienną `X` w wyniku spełnienia celu `p(X)` cel `q(`t`)` nie zawodzi.

d. ■ istnieje term t, dla którego oba cele `p(`t`)` i `q(`t`)` są spełnione na co najmniej jeden sposób.

Pytanie 3. Niech

```
empty([]).
p(X) :-
   empty(X),
   !,
   append(X,X,X).
q(X) :-
   !,
   append(X,X,X).
```

Wtedy

a. ■ odcięcie występujące w definicji predykatu p jest zielonym odcięciem.

b. □ odcięcie występujące w definicji predykatu q jest czerwonym odcięciem.

c. □ dla dowolnego termu t obliczenie celów p(t) i q(t) ma zawsze ten sam efekt (tj. predykaty p/1 i q/1 są równoważne).

d. □ usunięcie odcięcia z definicji predykatu p spowoduje, że będzie się on dla pewnych argumentów zapętlał.

Pytanie 4. Niech

```
p(_).
q(X) :-
   X = X.
r(X) :-
   X is X.
```

Wtedy dla dowolnego termu t efekt obliczenia celów

a. ■ p(t) i q(t) jest taki sam.

b. □ p(t) i r(t) jest taki sam.

c. □ q(t) i r(t) jest taki sam.

d. □ Żadna z powyższych odpowiedzi nie jest poprawna.

Pytanie 5. Obliczenie celu

```
?- append([1,2,3],X,X).
```

a. □ zapętli się.

b. □ zawiedzie.

c. ■ zakończy się pojedynczym sukcesem.

d. □ będzie mieć nieskończenie wiele sukcesów.

Pytanie 6. Rozważmy predykat

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

i zapytanie

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

a. ☐ Obliczenie zawiedzie.

b. ☐ Maszyna prologowa wygeneruje odpowiedź `X = [a,b,a]`, a po nawrocie zawiedzie.

c. ☐ Maszyna prologowa wygeneruje odpowiedź `X = [a,b,a,b,a]`, a po nawrocie zawiedzie.

d. ■ Obliczenie będzie mieć nieskończenie wiele sukcesów.

Pytanie 7. Skoro

```
member(H,[H|_]).
member(X,[_|T]) :-
   member(X,T).
```

to obliczenie celu

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

a. ☐ zawiedzie.

b. ☐ zapętli się.

c. ☐ zakończy się pojedynczym sukcesem.

d. ■ będzie mieć nieskończenie wiele sukcesów.

Pytanie 8. Rozważmy predykat

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

Obliczenie celu

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

a. ☐ zawiedzie.

b. ☐ zapętli się.

c. ■ będzie mieć pojedynczy sukces, a po nawrocie zapętli się.

d. ☐ będzie mieć nieskończenie wiele sukcesów.

Pytanie 9. Obliczenie celu

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

a. ☐ zawiedzie.

b. ☐ zapętli się.

c. ■ zakończy się pojedynczym sukcesem.

d. ☐ będzie mieć nieskończenie wiele sukcesów.

Pytanie 10. Niech *c* będzie pewnym celem w czystym Prologu (tj. nie powoduje odcięć, skutków ubocznych itp.). Wówczas cele *c* i `\+ \+` *c* są równoważne (ich obliczenie daje ten sam efekt) jeżeli cel *c*

a. ☐ jest termem zamkniętym.

b. ☐ jest spełniony co najwyżej raz.

c. ☐ jest spełniony co najmniej raz.

d. ■ zawodzi.

Pytanie 11. Niech

```
p(0).
p(N) :-
   M is N+1,
   p(M).
```

Wówczas obliczenie celu

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

a. ■ zakończy się pojedynczym sukcesem, a po nawrocie zapętli się.

b. ☐ zakończy się pojedynczym sukcesem, a po nawrocie wystąpi błąd `ERROR: p/1: Arguments are not sufficiently instantiated`.

c. ☐ będzie miało ten sam efekt, co obliczenie celu `p(X)`.

d. ☐ będzie mieć nieskończenie wiele sukcesów.

Pytanie 12. Obliczenie celu

```
?- not(member(X,[a,b,c])), X=d.
```

a. ■ zakończy się niepowodzeniem.

b. ☐ zakończy się pojedynczym sukcesem, w którym $\texttt{X} = \texttt{d}$.

c. ☐ zakończy się pojedynczym sukcesem, w którym `X` pozostanie nieukonkretnioną zmienną.

d. ☐ będzie mieć trzy sukcesy.

Pytanie 13. Wynikiem zapytania

```
?- 6*6 is 36.
```

jest

a. ☐ pojedynczy sukces,

b. ☐ zapętlenie,

c. ■ niepowodzenie,

d. ☐ błąd arytmetyczny.

Pytanie 14. Niech

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

Cel

```
?- p.
```

a. ☐ zawiedzie.

b. ☐ będzie spełniony na jeden sposób.

c. ■ będzie spełniony na nieskończenie wiele sposobów.

d. ☐ zapętli się.

Pytanie 15. Cel

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

a. ■ zawiedzie,

b. ☐ będzie spełniony na jeden sposób,

c. ☐ będzie spełniony na nieskończenie wiele sposobów,

d. ☐ zapętli się.

Pytanie 16. Kompilacja programu

```
data Czlowiek = Syn Czlowiek Czlowiek | Jan | Ewa
Jan = Syn(Jan,Ewa)
```

a. ☐ zakończy się błędem składniowym, gdyż definicje rekurencyjne mogą dotyczyć jedynie funkcji, a nie wartości algebraicznych typów danych, takich jak `Czlowiek`.

b. ■ zakończy się błędem typu.

c. ☐ zakończy się poprawnie, ale otrzymany program zapętli się.

d. ☐ zakończy się poprawnie, ale otrzymany program zakończy się błędem `Irrefutable pattern failed for pattern Jan`.

Pytanie 17. W zasięgu deklaracji

```
data Board coord = Board coord coord
```

wyrażenie `Board (1,2)` ma typ

a. ☐ `Board Int`

b. ☐ `Board Integer`

c. ☐ `Num a => a → Board a`

d. ■ `(Num a, Num b) => (a,b) → Board (a,b)`

Pytanie 18. Deklaracja

```
newtype MyMon a = MyMon a Integer
instance Monad MyMon where
   return x = MyMon x 0
   MyMon x r >>= f = MyMon y (r+s) where MyMon y s = f x
```

a. ■ jest niepoprawna składniowo.

b. □ nie jest poprawną definicją monady, gdyż typ `MyMon` nie spełnia równości

$$m \texttt{ >>= return } = m,$$

którą musi spełniać każda monada.

c. □ nie jest poprawną definicją monady, gdyż typ `MyMon` nie spełnia równości

$$\texttt{return } x \texttt{ >>= } f = f\ x,$$

którą musi spełniać każda monada.

d. □ jest poprawną definicją monady.

Pytanie 19. Niech

```
const x _ = x
```

Jeżeli typ `M :: * → *` jest monadą, to dla dowolnych wartości m :: `M` a, n :: `M` b, x :: b i funkcji f :: $b \to$ `M` c zachodzi równość

a. □ m >>= `const` $= m$.

b. □ `const` x >>= $f = f\ x$.

c. ■ (m >>= `const` n) >>= $f = m$ >>= `const` (n >>= f).

d. □ Nie zachodzi żadna z powyższych równości.

Pytanie 20. Niech

```
1 -* _ = 0
x -* y = (x-1) * y
(*-) = flip (-*)
```

Wtedy zachodzi równość

a. □ `foldr (-*) = foldl (-*)`.

b. □ `foldr (-*) = foldl (*-)`.

c. □ `foldr (-*) 0 = foldl (*-) 1`.

d. ■ Żadna z powyższych równości nie zachodzi.

Pytanie 21. W zasięgu deklaracji

```
class C t where
   m :: a → t a
```

wyrażenie `m 'a'` ma typ

a. □ `C t => Char`.

b. □ `C Char => t`.

c. ■ `C t => t Char`.

d. □ `C Char => t Char`.

Pytanie 22. Niech

```
data T = T T
newtype S = S S
```

Liczba różnych wartości typu `T` jest

a. ☐ równa 1.

b. ☐ równa 2.

c. ■ nieskończona.

d. ☐ równa liczbie różnych wartości typu `S`.

Pytanie 23. Niech

```
data T a = T a (T a)
f (T a t) = T a (f t)
```

Wtedy

a. ☐ funkcja `f` jest *non-strict*.

b. ■ funkcja `f` jest funkcją identyczności.

c. ☐ funkcja `f` ma pustą dziedzinę.

d. ☐ wywołanie funkcji `f` na dowolnej wartości typu `T a` zapętli się.

Pytanie 24. Równość xs ++ $xs = xs$

a. ☐ zachodzi dla dowolnej listy skończonej xs.

b. ☐ nie zachodzi dla żadnej listy skończonej xs.

c. ■ zachodzi dla dowolnej listy częściowej xs.

d. ☐ nie zachodzi dla żadnej listy częściowej xs.

Pytanie 25. Przypuśćmy, że dla pewnej funkcji f :: `Bool → Bool` zachodzi równość

$$f \,.\, f = \texttt{id}.$$

Wtedy

a. ☐ $f = \texttt{id}$.

b. ■ f jest *strict*.

c. ☐ f jest *non-strict*.

d. ☐ f jest funkcją stałą.

Pytanie 26. Niech

```
x = [ n*y | y ← x, n ← [2,3]]
```

Wartością zmiennej `x` jest

a. ☐ lista skończona `[2,3]`.

b. ☐ lista częściowa `2:3:`$\bot$.

c. ☐ lista nieskończona `[2,3,4,6,6,9,8,12,12,18,...]`.

d. ■ $\bot$.

Pytanie 27. Typem funkcji

```
f x y = f y x
```

jest

a. ■ $a \to a \to b$.

b. ☐ $a \to b \to a$.

c. ■ $a \to a \to$ `undefined`.

d. ☐ Funkcja `f` nie posiada typu.

Uwaga: `undefined` jest równie dobrą nazwą dla zmiennej typowej jak b!

Pytanie 28. Wskaż wyrażenie, które nie jest równe `[2,4,6]`.

a. ☐ `do { x ← [1,2,3]; return$ 2*x }`

b. ☐ `[1,2,3] >>= return . (2*)`

c. ☐ `[1,2,3] >>= (:[]).(2*)`

d. ■ `foldr ((:).(2*)) [1,2,3] []`

Pytanie 29. Niech

```
fork (f,g) x = (f x, g x)
```

Wskaż parę wyrażeń, które nie są równe.

a. ☐
```
foldr (const (+1)) 0
length
```

b. ☐
```
flip div 2 . uncurry (*) . fork (id,(1+)) . max 0
\ n -> foldr (+) 0 [1..n]
```

c. ■
```
let f n | n <= 1 = 1 | otherwise = n * f (n-1) in f
\ n -> foldr (*) 0 [1..n]
```

d. ☐
```
(>>=(:[]))
map id
```

Pytanie 30. 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) ]`.