Metody programowania

Egzamin poprawkowy

17 września 2014

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 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. W pytaniach otwartych proszę czytelnie wpisać odpowiedź wewnątrz prostokąta. Każde pytanie testowe jest warte 1 punkt, każde pytanie otwarte — 2 punkty. Czas trwania egzaminu: 120 minut.

Pytanie 1. Rozważmy predykat:

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

Dla jakich kombinacji zmiennych `X`, `Y` i `Z` wywołanie `max(X,Y,Z)` nie zakończy się błędem?

☐ a. `X` i `Z` są nieukonkretnionymi zmiennymi, a `Y` jest termem arytmetycznym.

☐ b. `Y` jest nieukonkretnioną zmienną, $X = 1$ i $Z = a$.

■ c. `X` jest nieukonkretnioną zmienną, $Y = 1$ i $Z = a$.

☐ d. `X` jest nieukonkretnioną zmienną, $Y = 1$ i $Z = 1$.

Pytanie 2. Rozważmy predykat:

```
p([], 0).
p([H|T], N) :-
   length(H, M),
   p(T, K),
   N is M+K.
```

Jaki będzie wynik zapytania `?- p([[1,2]],N).`

☐ a. `N = 0`.

☐ b. `N = 1`.

■ c. `N = 2`.

☐ d. `N = 3`.

Pytanie 3. Jaki jest wynik zapytania

```
?- 2*2 is 4.
```

☐ a. Pojedynczy sukces.

■ b. Niepowodzenie.

☐ c. Zapętlenie.

☐ d. `ERROR: is/2: Arguments are not sufficiently instantiated`.

Pytanie 4. Jaki jest wynik zapytania

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

■ a. Pojedynczy sukces, przy czym `V = [[]]`.

☐ b. Pojedynczy sukces, przy czym `V = []`.

☐ c. Pojedynczy sukces, przy czym `V` jest nieukonkretnioną zmienną.

☐ d. Niepowodzenie.

Pytanie 5. Rozważmy program:

```
a --> "1".
a --> a, a.
```

Jaki będzie wynik zapytania `?- a("111","").`

☐ a. `Undefined procedure: a/2. However, there are definitions for: a/0`.

■ b. Sukces, a po wymuszeniu nawrotu program zapętli się.

☐ c. Nieskończenie wiele sukcesów.

☐ d. Niepowodzenie.

Pytanie 6. Rozważmy zapytanie

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

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

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

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

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

Pytanie 7. 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 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. *Podciągiem* listy `X` nazywamy w tym zadaniu taką listę `Y`, że istnieją takie listy `P` i `S`, że lista `X` jest wynikiem konkatenacji list `P`, `Y` i `S`. Dane są listy `L1` i `L2`. Napisz w Prologu zapytanie „czy listy `L1` i `L2` zawierają wspólny podciąg długości 3?". Na przykład listy `[1,2,3,4,5,6]` oraz `[9,3,4,5,6,8]` zawierają wspólny podciąg długości 3, a listy `[1,2,3,4,5,6]` i `[1,2,6,3,4,7]` — nie. W zapytaniu możesz użyć jedynie predykatów `append/3` i `=/2`.

Pytanie 10. Binarne rozwinięcia liczb nieujemnych reprezentujemy w Prologu w postaci list cyfr `0` i `1` w kolejności od najmniej do najbardziej znaczącej. Zaprogramuj predykat `succ/2` wyznaczający następnik liczby w tej reprezentacji. Mamy więc np.:

```
?- succ([0,1,0,1],L).
L = [1, 1, 0, 1].

?- succ([1,1,0,1],L).
L = [0, 0, 1, 1].

?- succ([1,1,1,1],L).
L = [0, 0, 0, 0, 1].
```

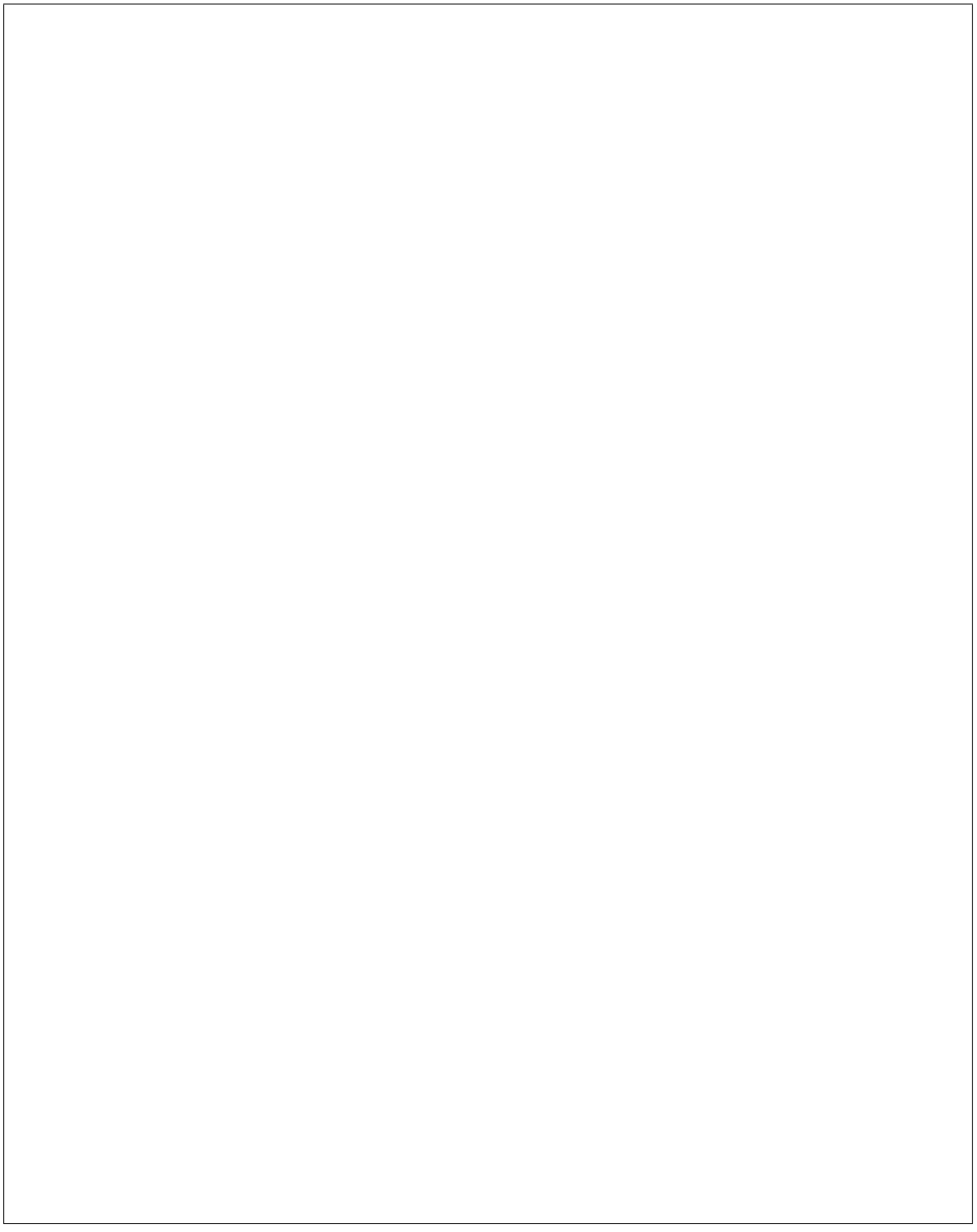
Pytanie 11. Zaprogramuj w Prologu predykat `p(?L)` spełniony wówczas, gdy reszta z dzielenia długości listy `L` przez 4 wynosi 3. Nie wolno używać arytmetyki ani żadnych pomocniczych predykatów.

Pytanie 12. Narysuj prologowe drzewo przeszukiwania dla celu

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

gdzie

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



Pytanie 13. Oto definicja funkcji `reverse`:

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

gdzie

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

Równość

```
reverse (reverse xs) = xs
```

zachodzi dla

☐ a. wszystkich list `xs`.

■ b. niektórych list częściowych `xs`.

☐ c. niektórych nieskończonych list `xs`.

☐ d. wszystkich częściowych list `xs`.

Pytanie 14. Oto definicja funkcji `map`:

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

Wyrażenie

```
map ⊥ []
```

jest równe

☐ a. ⊥.

■ b. `[]`.

☐ c. `map ⊥ ⊥`.

☐ d. `[⊥]`.

Pytanie 15. Które z poniższych wyrażeń nie posiada typu w Haskellu?

☐ a. `(2, tail, [])`

■ b. `[2, tail, []]`

☐ c. `2 : tail []`

☐ d. `tail $ 2 : []`

Pytanie 16. Rozważmy następującą sygnaturę:

```
f :: Integer → Integer → Integer
```

Wówczas `f` jest funkcją, która

☐ a. jako argumenty przyjmuje parę liczb całkowitych i zwraca w wyniku liczbę całkowitą.

☐ b. przyjmuje jako argument liczbę całkowitą i zwraca w wyniku parę liczb całkowitych.

■ c. przyjmuje jako argument liczbę całkowitą i zwraca w wyniku funkcję, która przyjmuje jako argument liczbę całkowitą i zwraca w wyniku liczbę całkowitą.

☐ d. przyjmuje jako argument funkcję, która przyjmuje jako argument liczbę całkowitą i zwraca w wyniku liczbę całkowitą, i zwraca w wyniku liczbę całkowitą.

Pytanie 17. 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 18. 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 19. 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 20. Wyrażenie

```
(undefined,undefined)
```

☐ a. jest równe `undefined`.

■ b. jest różne od `undefined`.

☐ c. jest równe `λ _ → undefined`.

☐ d. ma wartość $\bot$.

Pytanie 21. Notacja „do" w Haskellu umożliwia czytelne zapisywanie programów, ale jest jedynie „cukrem syntaktycznym", tj. konstrukcją, którą można już na poziomie parsowania zastąpić bardziej pierwotnymi mechanizmami języka. Przepisz podane niżej funkcje używając jedynie standardowych funkcji działających na listach i nie korzystając z notacji „do" ani metod klas `Monad` i `MonadPlus`.

`f :: [a] → [[a]]`
`f [] = return []`
`f (x:xs) =`
`  do`
`  ys ←f xs`
`  return (x:ys) ‘mplus‘`
`  return ys`	

`g :: [a] → [(a,[a])]`
`g [] = mzero`
`g (x:xs) =`
`  return (x,xs) ‘mplus‘ do`
`  (y,ys) ← g xs`
`  return (y,x:ys)`	

`h :: [a] → [[a]]`
`h [] = return []`
`h xs = do`
`  (y,ys) ← g xs`
`  zs ← h ys`
`  return (y:zs)`	

Pytanie 22. Zdefiniuj w Haskellu funkcję

$$\texttt{foldr2} :: (a \to b \to c \to c) \to c \to [a] \to [b] \to c$$
$$\texttt{foldr2}\ f\ c\ [a_1, \ldots, a_n]\ [b_1, \ldots, b_n] \ = \ f\ a_1\ b_1\ (f\ a_2\ b_2 \ldots (f\ a_n\ b_n\ c) \ldots)$$

Pytanie 23. Używając funkcji `foldr2` opisanej następująco:

$$\texttt{foldr2} :: (a \to b \to c \to c) \to c \to [a] \to [b] \to c$$
$$\texttt{foldr2}\ f\ c\ [a_1, \ldots, a_n]\ [b_1, \ldots, b_n] \ = \ f\ a_1\ b_1\ (f\ a_2\ b_2 \ldots (f\ a_n\ b_n\ c) \ldots)$$

uzupełnij poniższą definicję funkcji

$$\texttt{zipWith} :: (a \to b \to c) \to [a] \to [b] \to [c]$$
$$\texttt{zipWith}\ f\ [a_1, \ldots, a_n]\ [b_1, \ldots, b_n] \ = \ [f\ a_1\ b_1, f\ a_2\ b_2, \ldots, f\ a_n\ b_n]$$

```
zipWith f = foldr2  ..........................................................
```

Brudnopis