Programowanie 2010

Egzamin podstawowy

28 czerwca 2010

Każdą kartkę należy podpisać na marginesie imieniem i nazwiskiem.
Czas trwania egzaminu: 120 minut. Punktacja:

punkty	ocena
0–22	2.0
23–26	3.0
27–30	3.5
31–34	4.0
35–38	4.5
39–46	5.0

Zadanie 1 (5 pkt). Oto program w Prologu:

```
p(0).
p(X) :-
   Y is X-1,
   p(Y),
   !.
```

Odpowiedz na pytania:

Jaka będzie pierwsza odpowiedź maszyny na zapytanie `?- p(X).`	X = 0
Jaka będzie druga odpowiedź maszyny (po nawrocie) na powyższe zapytanie? (Wpisz kreskę, jeśli nawrót jest niemożliwy).	BŁĄD ARYTMETYCZNY (NIEUKONKRETNIONA ZMIENNA)
Jaka będzie pierwsza odpowiedź maszyny na zapytanie `?- p(1).`	true.
Jaka będzie druga odpowiedź maszyny (po nawrocie) na powyższe zapytanie? (Wpisz kreskę, jeśli nawrót jest niemożliwy).	———
Czy któraś z czterech powyższych odpowiedzi ulegnie zmianie, jeśli z programu usuniemy odcięcie? (Wpisz „TAK” lub „NIE”).	TAK

Zadanie 2 (5 pkt). Napisz gramatykę DCG, która w wyniku przekształcenia za pomocą predykatu `expand_term/2` daje następujący parser w Prologu:

```
a(N,[N|T],S) :-
   number(N),
   S=T.
a(E1+E2,P,Q) :-
   a(E1,P,R),
   R = [+|S],
   a(E2,S,Q).
a(E,['('|T],R) :-
   a(E,T,S),
   S = [')'|R].
```

```
a(N) -->
   [N],
   {number(N)}.

a(E1+E2) -->
   a(E1),
   [+],
   a(E2).

a(E) -->
   ['('],
   a(E),
   [')'].
```

Zadanie 3 (5 pkt). Macierze możemy reprezentować w Prologu w postaci list wierszy, z których każdy jest listą elementów. Macierzy

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

odpowiada wówczas lista

$$[[a_{11}, \ldots, a_{1n}], \ldots, [a_{m1}, \ldots, a_{mn}]].$$

Zaprogramuj predykat `transpose(+M1,-M2)` wyznaczający macierz transponowaną podanej macierzy. Jeżeli argument `M1` nie jest poprawną reprezentacją macierzy (tj. nie jest listą $m \geq 1$ list tej samej długości $n \geq 1$), to efektem wywołania predykatu powinno być niepowodzenie. Nie wolno korzystać z żadnych predykatów standardowych z wyjątkiem odcięcia. Można definiować własne predykaty pomocnicze. Wolno zdefiniować nie więcej niż 7 klauzul.

```
lastRow([],[]).
lastRow([[H]|C],[H|R]):-
   lastRow(C,R).
oneRow([],[],[]).
oneRow([[H|T]|C],[H|R],[T|D]):-
   oneRow(C,R,D).
transpose(C,[R]):- lastRow(C,R),!.
transpose(C,[R|T]):-
   oneRow(C,R,S), transpose(S,T).
```

Zadanie 4 (6 pkt). Podaj typy podanych wyrażeń w Haskellu.

`flip (.)`	$(a \to b) \to (b \to c) \to (a \to c)$
`(.) flip`	$(a \to b \to c \to d) \to (a \to c \to b \to d)$
`s s k`	BRAK TYPU
`s k s`	$(a \to b \to c) \to (a \to b \to c)$
`k s s`	$(a \to b \to c) \to (a \to b) \to (a \to c)$
`s k k`	$a \to a$

gdzie

```
flip x y z = x z y
(x . y) z = x (y z)
k x _ = x
s x y z = x z (y z)
```

Zadanie 5 (6 pkt). Rozważmy graf skierowany posiadający 64 wierzchołki etykietowane parami liczb z przedziału [1..8]. Między wierzchołkami (x, y) oraz (x', y') występuje w tym grafie krawędź, jeżeli $|x - x'| + |y - y'| = 3$ oraz $x \neq x'$ i $y \neq y'$. Wierzchołki grafu reprezentujemy w Haskellu w postaci wartości typu

```
data Wierzcholek = Wierzcholek
   { wiersz :: Int
   , kolumna :: Int
   , nastepniki :: [Wierzcholek]
   }
```

gdzie *nastepnikami* danego wierzchołka nazywamy wszystkie wierzchołki połączone z nim krawędzią. Zbuduj w Haskellu strukturę cykliczną składającą się z 64 wartości typu `Wierzcholek` reprezentującą opisany wyżej graf. Udostępnij ją w postaci wartości

```
startowy :: Wierzcholek
```

która reprezentuje wierzchołek o etykiecie $(1, 1)$. Zadbaj o to, by w pamięci faktycznie powstały cykle i by utworzone wierzchołki zostały spamiętane tak, by wielokrotne ich odwiedzanie nie powodowało tworzenia

nowych wartości w pamięci. Nie wolno używać żadnych funkcji standardowych z wyjątkiem `abs`, `+`, `/=`, `==`. Wolno używać wyrażeń listowych (*list comprehensions*) i konstruktorów list. *Wskazówka 1:* przypomnij sobie cyklisty. *Wskazówka 2:* przypomnij sobie skoczka szachowego.

```
startowy = rog where
  pola (rog:_) = [Wierzcholek x y (nast x y) | x<-[1..8], y<-[1..8]]
  nast x y = [p | p2(Wierzcholek x' y' _) <- pola,
                  abs(x-x') + abs(y-y') == 3,
                  x /= x',
                  y /= y']
```

Zadanie 6 (5 pkt). Napisy reprezentujemy w Prologu w postaci list kodów ASCII. Aby nie trzeba było pamiętać tych kodów używamy notacji z cudzysłowami (np. `"abc"` oznacza listę `[97,98,99]`). Napisz predykat `slownik/1` który generuje napisy złożone z wielkich liter alfabetu w następującej kolejności: najpierw słowo puste, potem słowa jednoliterowe w kolejności słownikowej, następnie słowa dwuliterowe w kolejności słownikowej itd.:

```
""
"A"
"B"
...
"Z"
"AA"
"AB"
...
"AZ"
"BA"
...
"AAA"
...
```

Nie wolno używać predykatów standardowych z wyjątkiem `member/2`. Wolno definiować własne predykaty pomocnicze. Wolno zdefiniować nie więcej niż 3 klauzule.

```
slownik(X):-
    slownik([],X).

slownik(X,X).
slownik(X,Y):-
    slownik([L|X],Y),
    member(L,"ABCDEFGHIJKLMNOPQRSTUVWXYZ").
```

Zadanie 7 (5 pkt). Oto definicja pewnego predykatu `p/2` w Prologu:

```
pair(X,Y,(X,Y)).

p(X,Z) :-
   reverse(X,Y),
   maplist(pair,X,Y,Z).
```

gdzie standardowy predykat `maplist/4` jest zdefiniowany następująco:

```
maplist(_,[],[],[]).
maplist(P,[H1|T1],[H2,T2],[H3,T3]) :-
   call(P,H1,H2,H3),
   maplist(P,T1,T2,T3).
```

Wadą predykatu `p/2` jest to, że lista `X` jest przeglądana dwukrotnie — raz, by utworzyć listę odwrotną, i drugi raz, by połączyć elementy w pary. Zaprogramuj inną wersję predykatu `p/2`, która zwraca wynik po jednokrotnym przejściu przez podaną listę `X`. Wolno zdefiniować jeden predykat pomocniczy, co najwyżej pięcioargumentowy. Wolno zdefiniować co najwyżej trzy klauzule. Nie wolno korzystać z żadnych predykatów standardowych (w szczególności z `maplist/4` i `reverse/2`).

```
p([],[],[],A,A).
p([H1|T1],[H2|T2],[(H1,H2)|T3],A,Y):-
   p(T1,T2,T3,[H1|A],Y).
p(X,Z):-
   p(X,Y,Z,[],Y).
```

Zadanie 8 (2 pkt). Oto łamigłówka: w skrajne kratki wpisz po jednej cyfrze, a w środkową znak operacji arytmetycznej `+`, `-` lub `*` tak, by zachodziła równość:

□□□ = 14.

Napisz zapytanie prologowe, którego wynikiem będzie rozwiązanie powyższej łamigłówki:

```
?- ... E ... .
E = 2*7;
E = 5+9;
E = 6+8;
E = 7*2;
E = 7+7;
E = 8+6;
E = 9+5;
false.
```

Nie wolno definiować żadnych predykatów. Wolno korzystać z predyktów standardowych `member/2`, `is/2` i `=../2`.

```
?- L=[0,1,2,3,4,5,6,7,8,9], member(X,L), member(Y,L),
   member(D,[+,-,*]), E =.. [D,X,Y], 14 is E.
```

Zadanie 9 (2 pkt). Zaprogramuj w Haskellu algorytm *Mergesort* w postaci funkcji

`msortn :: Ord a => Int -> [a] -> [a]`

gdzie `msortn` n xs jest niemalejącą permutacją n pierwszych elementów listy xs. Możesz użyć funkcji `drop`, metody `<=` klasy `Ord` i operacji arytmetycznych.

```
msortn 0 _ = []
msortn 1 (x:_) = [x]
msortn n xs = msortn n2 xs `merge` msortn (n-n2) (drop n2 xs) where
  n2 = n `div` 2
  xs `merge` [] = xs
  [] `merge` ys = ys
  xs@(x:xs') `merge` ys@(y:ys') =
    | x <= y = x : (xs' `merge` ys)
    | otherwise = y : (xs `merge` ys')
```

Zadanie 10 (2 pkt). Niech

```
data Tree a = Node (Tree a) a (Tree a) | Leaf
```

Zaprogramuj w Haskellu funkcję

`insert :: Ord a => a -> Tree a -> Tree a`

wstawiającą podany element do podanego binarnego drzewa poszukiwań.

```
insert a Leaf = Node Leaf a Leaf
insert a (Node l b r)
    | a <= b = Node (insert a l) b r
    | otherwise = Node l b (insert a r)
```

Zadanie 11 (3 pkt). *Słowa Fibonacciego* F_n nad alfabetem $\{a, b\}$ definiujemy rekurencyjnie:

$$\begin{aligned} F_0 &= a \\ F_1 &= b \\ F_{n+2} &= F_n \cdot F_{n+1} \end{aligned}$$

gdzie „$\cdot$" jest konkatenacją słów. Zaprogramuj w Haskellu funkcję

`fw :: Int -> IO ()`

która tworzy wartość typu `IO ()` reprezentującą skutek uboczny polegający na wypisaniu n-tego słowa Fibonacciego do standardowego strumienia wyjściowego. Wolno użyć funkcji `putChar` i operacji monadycznych. Nie wolno tworzyć list. Podobnie jak dla liczb Fibonacciego należy pamiętać dwie poprzednie wartości, by uniknąć wykładniczej liczby wywołań rekurencyjnych.

```
fw = aux (putChar 'a') (putChar 'b') where
  aux f0 _ 0 = f0
  aux f0 f1 n = aux f1 (f0>>f1) (n-1)
```