Programowanie 2010

Egzamin poprawkowy

20 września 2010

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

punkty	ocena
0–12	2.0
13–16	3.0
17–20	3.5
21–24	4.0
25–28	4.5
29–36	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).`	
Jaka będzie druga odpowiedź maszyny (po nawrocie) na powyższe zapytanie? (Wpisz kreskę, jeśli nawrót jest niemożliwy).	
Jaka będzie pierwsza odpowiedź maszyny na zapytanie `?- p(1).`	
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”).	

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

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

Zadanie 3 (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 `binarnie/1` który generuje napisy złożone ze znaków `0` i `1`, będące binarnymi reprezentacjami kolejnych dodatnich liczb całkowitych (bez zer nieznaczących):

```
"1"
"10"
"11"
"100"
"101"
...
```

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

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

`flip (.)`	
`(.) flip`	
`s s k`	
`s k s`	
`k s s`	
`s k k`	

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 z pola szachownicy (x, y) goniec może w jednym kroku przejść na pole (x', y') (goniec porusza się po przekątnych). 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.

Zadanie 6 (2 pkt). Oto łamigłówka: w skrajne kratki wpisz takie dwie dodatnie liczby całkowite, że ich suma wynosi 100, a w środkową znak operacji arytmetycznej `+`, `-` lub `*` tak, by zachodziła równość:

$$\square\,\square\,\square = 68.$$

Napisz zapytanie prologowe, którego wynikiem będzie rozwiązanie powyższej łamigłówki:

```
?- <wymyśl zapytanie> .
X = 84,
Y = 16,
F = (-),
Z = 84-16 ;
false.
```

Nie wolno definiować żadnych predykatów. Wolno korzystać z predyktów standardowych `member/2`, `is/2` i `=../2` oraz niestandardowego predykatu `between(+X,+Y,?Z)` podstawiającego pod `Z` kolejne liczby z przedziału od `X` do `Y`.

Zadanie 7 (2 pkt). Niech

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

Zaprogramuj w Haskellu funkcję

```
find :: Ord a => a -> Tree a -> Bool
```

sprawdzającą czy podana etykieta występuje w drzewie.

Zadanie 8 (3 pkt). Zaprogramuj w Haskellu algorytm *Mergesort* w postaci funkcji

```
msort :: Ord a => [a] -> [a]
```

Możesz użyć metody `<=` klasy `Ord`.

Zadanie 9 (2 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 Prologu predykat `fw/1` który generuje kolejne słowa Fibonacciego:

```
?- fw(X).
X = "a" ;
X = "b" ;
X = "ab" ;
X = "bab" ;
X = "abbab" ;
X = "bababbab" ;
...
```

Podobnie jak dla liczb Fibonacciego należy pamiętać dwie poprzednie wartości, by uniknąć wykładniczej liczby wywołań rekurencyjnych.