Programowanie 2009 Programming 2009

Egzamin podstawowy Basic exam

30 czerwca 2009 June 30, 2009

Czas trwania egzaminu: 180 minut. Oceny:

Duration of the exam: 180 minutes. Grades:

punkty/points	ocena/grade
0–14	2.0
15–17	3.0
18–20	3.5
21–23	4.0
24–26	4.5
27–30	5.0

Zadanie 1 (1 pkt). 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.

Problem 1 (1 p). Define in Prolog a predicate `p(?L)` satisfied when the remainder of the division of the length of the list `L` by 4 is 3. You are not allowed to use arithmetic or auxiliary predicates.

Zadanie 2 (1 pkt). Zaprogramuj w Prologu predykat `revapp/3` działający podobnie jak standardowy predykat `append/3` z tą różnicą, że pierwsza lista przed połączeniem z drugą zostaje odwrócona. Mamy więc np.

Problem 2 (1 p). Define in Prolog a predicate `revapp/3` that acts similarly to the standard predicate `append/3` with the difference that the first list is reversed before it is appended to the second one. We have for example:

```
?- revapp([1,2,3],[4,5],X).
X = [3, 2, 1, 4, 5].
```

Predykat powinien działać poprawnie przynajmniej w trybie `revap(+L1,+L2,?L3)`. Nie wolno używać arytmetyki ani żadnych pomocniczych predykatów.

The predicate should work correctly at least in the mode `revap(+L1,+L2,?L3)`. You are not allowed to use arithmetic or auxiliary predicates.

Zadanie 3 (1 pkt). 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.:

Problem 3 (1 p). We represent in Prolog binary representations of nonnegative integers using lists of digits `0` and `1` from the least to the most significant. Define a predicate `succ/2` that computes the successor of a number in this representation. We have for example:

```
?- 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].
```

Zadanie 4 (1 pkt). Drzewa binarne o etykietowanych wierzchołkach wewnętrznych reprezentujemy w Prologu w postaci struktur zbudowanych z atomu `leaf/0` i funktora `node/3`, którego pierwszy i trzeci argument są, odpowiednio, lewym i prawym poddrzewem, a drugi — etykietą wierzchołka. Zaprogramuj predykat `r(+T,?N)` spełniony wówczas, gdy etykieta `a/0` występuje `N` razy na każdej ścieżce od korzenia do liścia w drzewie `T`.

Problem 4 (1 p). We represent in Prolog binary trees with labelled internal nodes using structures built from the atom `leaf/0` and the functor `node/3`, whose the first and third argument is, respectively, the left and right subtree, and the second — the label of a node. Define a predicate `r(+T,?N)` satisfied when the label `a/0` occurs `N` times in any path from the root to the leaf in a tree `T`.

Zadanie 5 (1 pkt). Jaka jest odpowiedź maszyny prologowej na zapytanie

Problem 5 (1 p). What is the answer of the Prolog machine to the query

```
?- \+ append(X,Y,Z).
```

Zadanie 6 (2 pkt). Napisz zbiór produkcji gramatyki bezkontekstowej nad alfabetem terminalnym $\{a, b\}$ i alfabetem nieterminalnym $\{S\}$ generującej język

Problem 6 (2 p). Write the set of productions of a context-free grammar over the terminal alphabet $\{a, b\}$ and nonterminal alphabet $\{S\}$ that generates the language

$$\{w \in \{a, b\}^* : |w|_a \leq |w|_b\},$$

gdzie $|w|_x$ dla $x \in \{a, b\}$ oznacza liczbę wystąpień symbolu x w słowie w.

where $|w|_x$ for $x \in \{a, b\}$ stands for the number of occurrences of a symbol x in a word w.

Zadanie 7 (1 pkt). Napisz jednym zdaniem definicję pojęcia jednoznacznej gramatyki bezkontekstowej.

Problem 7 (1 p). Write a single sentence containing the definition of the notion of an unambiguous context-free grammar.

Zadanie 8 (1 pkt). Napisz reguły semantyki operacyjnej wielkich kroków (semantyki naturalnej) dla instrukcji

Problem 8 (1 p). Write the rules of the big-step operational semantics (natural semantics) for the instruction

`repeat` c `until` b

Wykonanie tej instrukcji polega na cyklicznym wykonywaniu instrukcji c. Pętla jest zakończona, gdy po wykonaniu instrukcji c jest spełniony warunek b.

Execution of this statement consists in repetitive execution of the statement c. The loop is completed when the condition b is satisfied after the execution of the statement c.

Zadanie 9 (1 pkt). Wyrażenia arytmetyczne języka „While” są opisane następującą gramatyką abstrakcyjną:

Problem 9 (1 p). Arithmetic expressions of the “While” language are described by the following abstract syntax:

$$\begin{array}{rcl} n & — & \text{integer literals} \\ x & — & \text{identifiers} \\ a & ::= & n \mid x \mid a_1 \oplus a_2 \\ \oplus & ::= & + \mid - \mid \times \mid \texttt{div} \mid \texttt{mod} \end{array}$$

Podaj reguły semantyki małych kroków (strukturalnej semantyki operacyjnej) dla wyrażeń arytmetycznych języka „While”.

Give the rules of the small-step semantics (structural operational semantics) for arithmetic expressions of the “While” language.

Zadanie 10 (3 pkt). Przypomnijmy, że

Problem 10 (3 p). Recall that

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr  _  c   []   = c
foldr (*) c (x:xs) = x * foldr (*) c xs
```

Uzupełnij poniższe definicje funkcji w Haskellu.

Complete the definitions of the Haskell functions presented below.

```
length = foldr
```

```
(++) = flip $ foldr
```

```
concat = foldr
```

Zadanie 11 (2 pkt). Podaj definicję klasy `Monad`.

Problem 11 (2 p). Give the definition of the `Monad` class.

Niech

Let

```
data Failable a = OK a | Error String
```

Uczyń typ `Failable` instancją klasy `Monad`.

Make the type `Failable` the instance of the `Monad` class.

Zadanie 12 (4 pkt). Wiedząc, że

Problem 12 (4 p). Knowing that

```
(f . g) x = f (g x)
f $ x = f x
flip f a b = f b a
```

wyznacz typy następujących wyrażeń w Haskellu:

find the types of the following Haskell expressions:

`(.)(.)(.)      ::`
`(.)($)(.)      ::`
`($)(.)($)      ::`
`flip flip flip ::`

Zadanie 13 (2 pkt). Do zbioru wyrażeń arytmetycznych języka „While” dodajemy operatory postinkrementacji i postdekrementacji:

Problem 13 (2 p). We add the postincrementation and postdecrementation operators to the set of arithmetic expressions of the “While” language:

$$\begin{array}{rcl} n & — & \text{integer literals} \\ x & — & \text{identifiers} \\ a & ::= & n \mid x \mid a_1 \oplus a_2 \mid x\texttt{--} \mid x\texttt{++} \\ \oplus & ::= & + \mid - \mid \times \mid \texttt{div} \mid \texttt{mod} \end{array}$$

Zdefiniuj dziedzinę denotacji powyższych wyrażeń arytmetycznych.

Define the domain of denotations of the arithmetic expressions above.

Zdefiniuj semantykę denotacyjną tych wyrażeń arytmetycznych.

Define the denotational semantics of those arithmetic expressions.

Zadanie 14 (1 pkt). Rozważmy lambda wyrażenia z typami prostymi. Podaj przykład zamkniętego lambda wyrażenia typu $((\alpha \to \alpha) \to \beta) \to \beta$.

Problem 14 (1 p). Consider the simply typed lambda terms. Give an example of a closed lambda term of type $((\alpha \to \alpha) \to \beta) \to \beta$.

Zadanie 15 (1 pkt). Podaj najsłabszy warunek wstępny dla programu

Problem 15 (1 p). Give the weakest precondition for the program

```
while N <> 0 do R := 2 * R; N := N - 1; done
```

z warunkiem końcowym $\mathtt{R} = 2^i$.

with the postcondition $\mathtt{R} = 2^i$.

Zadanie 16 (3 pkt). Udekoruj poniższy program asercjami tak, by otrzymać dowód częściowej poprawności tego programu względem podanej specyfikacji.

Problem 16 (3 p). Decorate the program below with assertions to get the proof of partial correctness of this program with respect to the given specification.

$\{\mathtt{N} = i\}$

$\Big\{$ .... $\Big\}$

```
R := 1;
```

$\Big\{$ .... $\Big\}$

$\Big\{$ .... $\Big\}$

```
while N <> 0 do
```

$\Big\{$ .... $\Big\}$

$\Big\{$ .... $\Big\}$

```
    R := 2 * R;
```

$\Big\{$ .... $\Big\}$

```
    N := N - 1;
```

$\Big\{$ .... $\Big\}$

$\Big\{$ .... $\Big\}$

```
done
```

$\Big\{$ .... $\Big\}$

$\{\mathtt{R} = 2^i\}$

Zadanie 17 (3 pkt). Liczby naturalne będziemy reprezentować za pomocą tzw. liczebników Churcha:

Problem 17 (3 p). We will represent natural numbers by so called Church numerals:

$$\underline{m} \quad = \quad \lambda f.\lambda x.f^{(m)}x,$$

gdzie

where

$$\begin{aligned} t^{(0)}s &= s, \\ t^{(n+1)}s &= t(t^{(n)}s), \end{aligned}$$

dla dowolnych termów t i s. Mówimy, że lambda wyrażenie t *oblicza* funkcję $f : \mathbb{N}^k \to \mathbb{N}$, jeżeli dla wszelkich liczb naturalnych $m_1, \ldots, m_k \in \mathbb{N}$ zachodzi

for any terms t and s. We say that a lambda term t *computes* a function $f : \mathbb{N}^k \to \mathbb{N}$ if for all natural numbers $m_1, \ldots, m_k \in \mathbb{N}$ one has

$$\underline{f(m_1, \ldots, m_k)} \quad =_\beta \quad t\ \underline{m_1} \ldots \underline{m_k}.$$

Podaj lambda wyrażenia obliczające następnik, dodawanie i mnożenie.

Give lambda terms computing the successor, addition and multiplication.

Zadanie 18 (1 pkt). Rozważmy sygnaturę złożoną z symboli $a/0$, $b/0$, $:/2$ i $+\!\!+/2$. Niech

Problem 18 (1 p). Consider a signature consisting of symbols $a/0$, $b/0$, $:/2$ and $+\!\!+/2$. Let

$$E_1 \quad = \quad \{a +\!\!+ x = a : x,\ b +\!\!+ x = b : x,\ (a : x) +\!\!+ y = a : (x +\!\!+ y),\ (b : x) +\!\!+ y = b : (x +\!\!+ y)\}$$

Przyjmujemy semantykę algebry początkowej. Udowodnij, że $(x +\!\!+ y) +\!\!+ z = x +\!\!+ (y +\!\!+ z)$.

We assume the initial algebra semantics. Prove that $(x +\!\!+ y) +\!\!+ z = x +\!\!+ (y +\!\!+ z)$.