Programowanie 2009 Programming 2009

Egzamin podstawowy poprawkowy Basic repetition exam

8 września 2009 September 8, 2009

Czas trwania egzaminu: 180 minut. Oceny:

Duration of the exam: 180 minutes. Grades:

punkty/points	international	ocena/grade
0–14	0– 8	2.0
15–17	9–10	3.0
18–20	11–12	3.5
21–23	13–14	4.0
24–26	15–16	4.5
27–30	17	5.0

Studenci studiów międzynarodowych nie rozwiązują zadań *oznaczonych gwiazdką.

International students do not solve the problems marked with *the asterisk.

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

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

```
?- \+ X*X is 9.
```

Zadanie 2 (1 pkt). 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 predykatu `append/3`.

Problem 2 (1 p). Given two lists `L1` and `L2`. Write in Prolog the query "do the lists `L1` and `L2` share a common substring of length 3?". For example the lists `[1,2,3,4,5,6]` and `[9,3,4,5,6,8]` share a common substring of length 3, but the lists `[1,2,3,4,5,6]` and `[1,2,6,3,4,7]` — do not. You can use the predicate `append/3` in your query.

Zadanie 3 (1 pkt). Zaprogramuj w Prologu predykat `concat/2` spinający listę list w jedną listę:

Problem 3 (1 p). Define in Prolog a predicate `concat/2` that concatenates a list of lists into a single list:

```
?- concat([[1],[2,3,4],[[5],6]],L).
L = [1, 2, 3, 4, [5], 6].
```

Możesz użyć predykatu `append/3`. Zadbaj by Twój predykat nie generował nieużytków a rekursja była ogonowa.

You can use the predicate `append/3`. Ensure that your predicate generates no garbage and is tail recursive.

Zadanie 4 (2 pkt). Oto prosta łamigłówka: w poniższym wyrażeniu

Problem 4 (2 p). Here is a simple puzzle: in the expression below

```
1 + 2 - 3 * 4 - 5 // 6 - 7 + 8 * 9
```

(gdzie `//` oznacza dzielenie całkowite) wstaw nawiasy tak, by wartość wyrażenia wyniosła 1. Aby móc sprawdzać jakie wartości może przyjąć wyrażenie napisz predykat `result/2`, który unifikuje swój drugi argument z wartościami pierwszego argumentu otrzymanymi przy różnych rozstawieniach nawiasów. Wyrażenie reprezentujemy w postaci listy liczb i atomów `+/0`, `-/0`, `*/0` i `// /0`. Na przykład:

(where `//` stands for integer division) put parentheses so that the value of this expression becomes 1. In order to be able to discover possible values of an expression write a predicate `result/2` which unifies its second argument with the values of its first argument computed under various placements of parentheses. We represent the expression as a list of numbers and atoms `+/0`, `-/0`, `*/0` and `// /0`. For example:

```
?- result([1,-,2,*,3,-,4],N).
N = 3 ;
N = -1 ;
N = 1 ;
N = -9 ;
N = -7 ;
false.
```

Można używać predykatów: `append/3`, `member/2`, `=../2` oraz `is/2`. Nie wolno definiować predykatów pomocniczych. Definicja predykatu `result/2` powinna zawierać co najwyżej dwie klauzule. *Wskazówka:*

It is allowed to use predicates `append/3`, `member/2`, `=../2`, and `is/2`. It is not allowed to define auxiliary predicates. The definition of the predicate `result/2` should consist of at most two clauses. *Hint:*

```
?- E =.. [+,1,2], N is E.
E = 1+2,
N = 3.
```

Zadanie 5 (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. Mówimy, że dwa drzewa są *izomorficzne*, jeżeli można w niektórych wierzchołkach zamienić kolejność synów tak, by drzewa stały się identyczne. Zaprogramuj predykat `izo(?L,?R)` spełniony wówczas, gdy drzewa `L` i `R` są izomorficzne. Nie wolno używać predykatów pomocniczych.

Problem 5 (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. We say that two trees are *isomorphic* if it is possible to change the order of sons in some vertices so that the trees become identical. Define a predicate `izo(?L,?R)` satisfied when the trees `L` and `R` are isomorhic. It is not allowed to use auxiliary predicates.

Zadanie 6 (1 pkt). Niech ⟨*simple instruction*⟩ i ⟨*logical expression*⟩ będą symbolami nieterminalnymi występującymi w definicji składni pewnego języka programowania. Instrukcja w tym języku to instrukcja prosta lub instrukcja warunkowa o następującej składni: zaczyna się słowem kluczowym `if` po którym następuje wyrażenie logiczne, słowo kluczowe `then` i instrukcja. Następnie występuje zero lub więcej ciągów składających się ze słowa kluczowego `elif`, wyrażenia logicznego, słowa kluczowego `then` i instrukcji. Następnie opcjonalnie występuje słowo kluczowe `else` po którym następuje instrukcja. Instrukcja warunkowa jest zakończona słowem kluczowym `fi`. Napisz produkcje gramatyki BNF dla symbolu nieterminalnego ⟨*instruction*⟩.

Problem 6 (1 p). Let ⟨*simple instruction*⟩ and ⟨*logical expression*⟩ be nonterminal symbols occurring in a definition of the syntax of some programming language. Instruction in this language is either a simple instruction or the conditional instruction which has the following syntax: it starts with the keyword `if` followed by a logical expression, the keyword `then` and an instruction. Then there are zero or more sequences consisting of the keyword `elif`, a logical expression, the keyword `then` and an instruction. Then there is optionally the keyword `else` and an instruction. The conditional instruction is terminated with the keyword `fi`. Write productions of the BNF grammar for the nonterminal symbol ⟨*instruction*⟩.

Zadanie 7 (2 pkt). Napisz zbiór produkcji gramatyki bezkontekstowej nad alfabetem terminalnym $\{a, b\}$ i alfabetem nieterminalnym $\{S\}$ generującej język złożony ze słów mających tyle samo wystąpień liter a i b.

Problem 7 (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 consisting of words having the same number of occurrences of the letters a and b.

Zadanie 8 (1 pkt). Do zbioru wyrażeń arytmetycznych języka „While" dodajemy operatory postinkrementacji i postdekrementacji:

Problem 8 (1 p). We add the postincrementation and postdecrementation operators to the set of arithmetic expressions of the "While" language:

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

Podaj reguły semantyki małych kroków (strukturalnej semantyki operacyjnej) dla wyrażeń arytmetycznych tego języka.

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

Zadanie 9 (1 pkt). Napisz reguły semantyki operacyjnej wielkich kroków (semantyki naturalnej) dla instrukcji

Problem 9 (1 p). Write the rules of the big-step operational semantics (natural semantics) for the instruction

`do` c `while` b

Wykonanie tej instrukcji polega na cyklicznym wykonywaniu instrukcji c. Pętla jest zakończona, gdy po wykonaniu instrukcji c nie 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 not satisfied after the execution of the statement c.

Zadanie 10 (3* pkt). Przypomnijmy, że

Problem 10 (3* p). Recall that

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl  _  c   []   = c
foldl (*) c (x:xs) = foldl (*) (c*x) xs
const :: a -> b -> a
flip :: (a -> b -> c) -> (b -> a -> c)
(.) :: (b -> c) -> (a -> b) -> (a -> c)
```

Uzupełnij poniższe definicje funkcji w Haskellu.

Complete the definitions of the Haskell functions presented below.

```
length = foldl
```

```
reverse = foldl
```

```
sum = foldl
```

Zadanie 11 (4* pkt). Wiedząc, że

Problem 11 (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 ::
```

Zadanie 12 (2^* pkt). Podaj definicję klasy `MonadPlus`.

Problem 12 (2^* p). Give the definition of the `MonadPlus` class.

Uczyń typ `[]` instancją klasy `MonadPlus`.

Make the type `[]` the instance of the `MonadPlus` class.

Zadanie 13 (2 pkt). Do zbioru wyrażeń arytmetycznych języka „While" dodajemy funkcję wczytującą liczby ze strumienia wejściowego:

Problem 13 (2 p). We add the function that reads numbers from the input stream to the set of arithmetic expressions of the "While" language:

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

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 $((\alpha \to \beta \to \alpha) \to \gamma) \to \gamma$.

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

Zadanie 15 (3* pkt). Liczby naturalne będziemy reprezentować za pomocą tzw. liczebników Churcha:

Problem 15 (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 dodawanie, mnożenie i potęgowanie.

Give lambda terms computing the addition, multiplication and exponentiation.

Zadanie 16 (1 pkt). Podaj najsłabszy warunek wstępny dla programu

Problem 16 (1 p). Give the weakest precondition for the program

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

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

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

Zadanie 17 (3 pkt). Udekoruj poniższy program asercjami tak, by otrzymać dowód częściowej poprawności tego programu względem podanej specyfikacji.

Problem 17 (3 p). Decorate the program below with assertions to get the proof of partial correctness of this program with respect to the given specification.

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

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

```
R := 2;
```

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

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

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

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

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

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

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

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

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

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

```
done
```

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

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