

PROGRAMOWANIE W LOGICE

Wejście i wyjście

(Lista 5)

Przemysław Kobylański

Wstęp

Aby rozłożyć term na funktor i listę jego argumentów należy użyć predykatu `=..` w następujący sposób

```
Term =.. [Funktory | ListaArgumentów]
```

Stałe traktowane są jako funktory zero-argumentowe, natomiast jeśli term jest zmienną, to nie da się go rozłożyć:

```
?- f(a, b) =.. [F | A].  
F = f,  
A = [a, b].
```

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

```
?- X =.. [f, 1, a].  
X = f(1, a).
```

```
?- a =.. X.  
X = [a].
```

```
?- X =.. L.  
ERROR: =../2: Arguments are not sufficiently instantiated  
?- 10 =.. X.  
X = [10].
```

```
?- 2.5 =.. X.  
X = [2.5].
```

Zadania

Zadanie 1 (5 pkt)

Poniżej przedstawiono gramatykę BNF języka **Imperator**:

```
PROGRAM ::=
PROGRAM ::= INSTRUKCJA ; PROGRAM

INSTRUKCJA ::= IDENTYFIKATOR := WYRAŻENIE
INSTRUKCJA ::= read IDENTYFIKATOR
INSTRUKCJA ::= write WYRAŻENIE
INSTRUKCJA ::= if WARUNEK then PROGRAM fi
INSTRUKCJA ::= if WARUNEK then PROGRAM else PROGRAM fi
INSTRUKCJA ::= while WARUNEK do PROGRAM od

WYRAŻENIE ::= SKŁADNIK + WYRAŻENIE
WYRAŻENIE ::= SKŁADNIK - WYRAŻENIE
WYRAŻENIE ::= SKŁADNIK

SKŁADNIK ::= CZYNNIK * SKŁADNIK
SKŁADNIK ::= CZYNNIK / SKŁADNIK
SKŁADNIK ::= CZYNNIK mod SKŁADNIK
SKŁADNIK ::= CZYNNIK

CZYNNIK ::= IDENTYFIKATOR
CZYNNIK ::= LICZBA_NATURALNA
CZYNNIK ::= ( WYRAŻENIE )

WARUNEK ::= KONIUNKCJA or WARUNEK
WARUNEK ::= KONIUNKCJA

KONIUNKCJA ::= PROSTY and KONIUNKCJA
KONIUNKCJA ::= PROSTY

PROSTY ::= WYRAŻENIE = WYRAŻENIE
PROSTY ::= WYRAŻENIE /= WYRAŻENIE
PROSTY ::= WYRAŻENIE < WYRAŻENIE
PROSTY ::= WYRAŻENIE > WYRAŻENIE
PROSTY ::= WYRAŻENIE >= WYRAŻENIE
PROSTY ::= WYRAŻENIE <= WYRAŻENIE
PROSTY ::= ( WARUNEK )
```

Identyfikatory są słowami złożonymi z wielkich liter.

Napisz predykat `scanner(Strumień, Tokeny)`, który czyta strumień znaków i zamienia go na listę tokenów, przy czym tokeny powinny mieć następującą postać:

key(SŁOWO_KLUCZOWE) gdzie SŁOWO_KLUCZOWE jest jednym ze słów kluczowych języka Imperator: read, write, if, then, else, fi, while, do, od, and, or, mod;

int(LICZBA_NATURALNA) gdzie LICZBA_NATURALNA jest, jak sama nazwa wskazuje, liczbą naturalną;

id(ID) gdzie ID jest nazwą zmiennej będącą słowem złożonym z wielkich liter;

sep(SEPARATOR) gdzie SEPARATOR jest jednym z separatorów języka **Imperator**: ';;', '+', '-', '*', '/', '(', ')', '<', '>', '=<', '>=', ':=', '=', '/='.

Przykład

Załóżmy, że w pliku `ex1.prog` znajduje się następujący tekst:

```
read N;
SUM := 0;
while N > 0 do
    SUM := SUM + N;
    N := N - 1;
od;
write SUM;
```

Przykład wykonania skanowania na powyższym pliku:

```
?- open('ex1.prog', read, X), scanner(X, Y), close(X), write(Y).
[key(read),id(N),sep(;),id(SUM),sep(:=),int(0),sep(;),key(while),
id(N),sep(>),int(0),key(do),id(SUM),sep(:=),id(SUM),sep(+),id(N),
sep(;),id(N),sep(:=),id(N),sep(-),int(1),sep(;),key(od),sep(;),
key(write),id(SUM),sep(;;)]
X = <stream>(0x7f8f0b61e0c0),
Y = [key(read), id('N'), sep(;), id('SUM'), sep(:=), int(0),
    sep(;), key(while), id(...)|...] .
```

Uwaga

To zadanie trzeba rozwiązać, bo na następnych listach będziemy z niego korzystać.

Zadanie 2 (3 pkt)

Będziemy rysować szachownice złożone z ciemnych i jasnych pól:

+-----+	+-----+
:~::~ pole	~ pole
:~::~ ciemne	~ jasne
+-----+	+-----+

Na polach szachownicy mogą być umieszczone hetmany:

+-----+	+-----+
:###: pole	### pole
:###: ciemne	### jasne
+-----+	+-----+

Napisz predykat `board(L)`, którego parametrem jest lista liczb `L`. Rysuje on szachownice o `N` wierszach i `N` kolumnach, gdzie `N` jest długością listy `L`.

Liczba na i -tej pozycji listy `L`, gdzie $i \in \{1, 2, \dots, N\}$, jest numerem wiersza (liczonym od dołu planszy), w którym stoi hetman ustawiony w i -tej kolumnie.

Uwaga

Pamiętaj, że pole w lewym dolnym rogu szachownicy jest zawsze ciemne.

Przykład

Na rysunku 1 przedstawiono przykład użycia predykatu `board/1`. W przykładzie tym wykorzystano prezentowany na wykładzie predykat `hetmany/2`, który znajduje ustawienie niebijących się hetmanów.

Zadanie 3* (2 pkt)

Na prologowe terminy można patrzeć jak na drzewa. Liśćmi takiego drzewa są stałe i zmienne występujące w nim, natomiast wierzchołki wewnętrzne odpowiadają funktorom.

Na rysunku 2 przedstawiono w postaci drzewa przykładowy term `f1(f2(a2, a3), a1, f3(a4))`.

Obchodzenie drzewa reprezentującego term rozpoczyna się w jego korzeniu.

Do poruszania się po drzewie służą następujące polecenia:

- **in** – przejście z bieżącego wierzchołka do pierwszego jego syna,
- **out** – powrót do ojca bieżącego wierzchołka,
- **next** – przejście do następnego brata,
- **prev** – przejście do poprzedniego brata.

Powyższe polecenia przedstawiono na rysunku 3.

Polecenia będziemy zapisywać krócej pierwszą literą ich nazwy.

Na rysunku 4 przedstawiono przykładowe obejście wszystkich wierzchołków drzewa z rysunku 2 w kolejności wykonywania poleceń: **ii nonniooo**.

Napisz predykat `browse(Term)`, który pozwala obchodzić drzewo odpowiadające termowi `Term`.

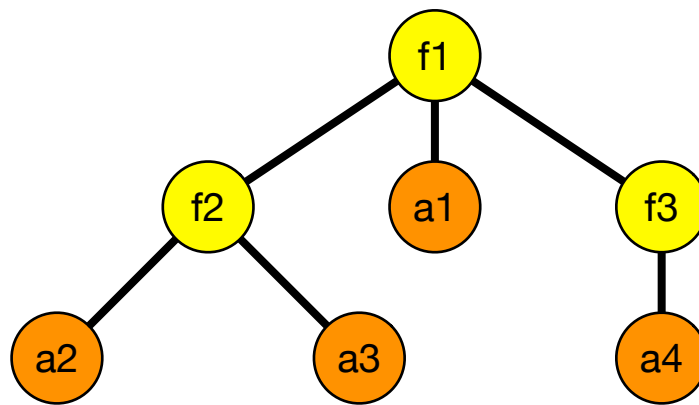
Za każdym razem gdy jesteśmy w wierzchołku, drukowany jest term w nim zakorzeniony (drukuj tak aby stałe o nazwach rozpoczynających się wielkimi literami były ujęte w cudzysłowy).

```

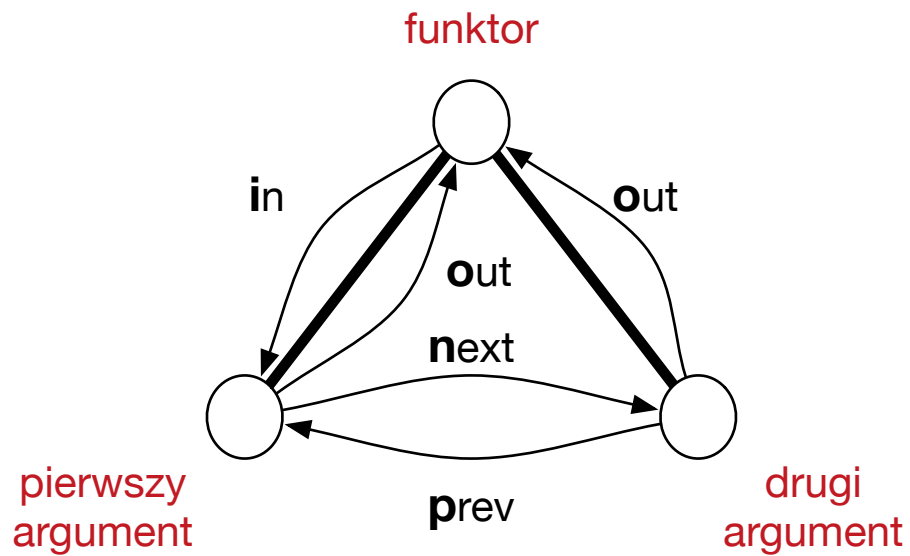
?- hetmany(12, X), board(X).
+-----+-----+-----+-----+-----+-----+-----+
| |:::| |:::| |:::| |:::| |:::| |:::|
| |:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| |:::| |:::| |:::| ### |:::| |:::|
|:::| |:::| |:::| |:::| ### |:::| |:::|
+-----+-----+-----+-----+-----+-----+
| |:::| |:::| ### |:::| |:::| |:::| |:::|
| |:::| |:::| ### |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| |:::| |:::| |:::| |:::| |:::|
|:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
| |:::| |:::| |:::| |:::| |:::| |:::|
| |:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| |:::| |:::| |:::| |:::| ### |:::|
|:::| |:::| |:::| |:::| |:::| ### |:::|
+-----+-----+-----+-----+-----+-----+
| |:::| |:::| |:::| |:::| ### |:::| |:::|
| |:::| |:::| |:::| |:::| ### |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| |:::| |:::| |:::| |:::| |:::|
|:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
| |:::| |:::| |:::| |:::| |:::| |:::|
| |:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| ### |:::| |:::| |:::| |:::| |:::|
|:::| ### |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
| |:::| |:::| |:::| |:::| ### |:::| |:::|
| |:::| |:::| |:::| |:::| ### |:::| |:::|
+-----+-----+-----+-----+-----+-----+
|:::| |:::| |:::| |:::| |:::| |:::|
|:::| |:::| |:::| |:::| |:::| |:::|
+-----+-----+-----+-----+-----+-----+
X = [1, 3, 5, 8, 10, 12, 6, 11, 2]...

```

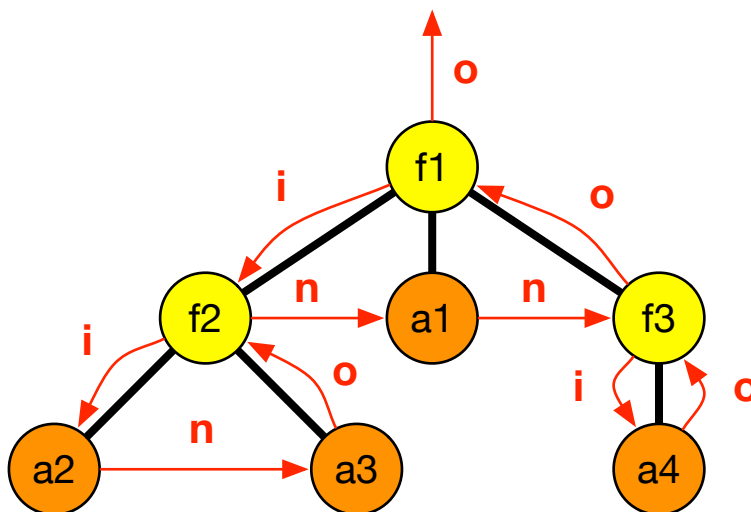
Rysunek 1: Przykład szachownicy dla $N = 12$.



Rysunek 2: Term $f1(f2(a2, a3), a1, f3(a4))$.



Rysunek 3: Polecenia do poruszania się po drzewie.



Rysunek 4: Obchodzenie drzewa zgodnie z poleceniami: **ii nonnii ooo**.

Po wydrukowaniu termu predykat powinien czytać stałą odpowiadającą pierwszej literze polecenia i przejść w drzewie dalej zgodnie z tym poleceniem.

Jeśli wpisano złą stałą albo nie można wykonać danego polecenia (np. **i** gdy jesteśmy w liściu), to pozostajemy w bieżącym wierzchołku i jeszcze raz drukujemy zakorzeniony w nim term.

Przykład

Oto przykładowy dialog jak w sytuacji przedstawionej na rysunku 4:

```
?- browse(f1(f2(a2, a3), a1, f3(a4))).  
f1(f2(a2,a3),a1,f3(a4))  
command: i.  
f2(a2,a3)  
command: |: i.  
a2  
command: |: n.  
a3  
command: |: o.  
f2(a2,a3)  
command: |: n.  
a1  
command: |: n.  
f3(a4)  
command: |: i.  
a4  
command: |: o.  
f3(a4)  
command: |: o.  
f1(f2(a2,a3),a1,f3(a4))  
command: |: o.  
  
true .
```