

PROGRAMOWANIE W LOGICE

Gramatyki metamorficzne

(Lista 6)

Przemysław Kobyłański

Wstęp

brak

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 )

```

Na poprzedniej liście w zadaniu 1 należało napisać skaner czytający strumień znaków i oddający listę tokenów (wg. powyższej gramatyki).

Napisz gramatykę metamorficzną o początkowym symbolu nieterminalnym `program(PROGRAM)`, która analizuje listę tokenów z poprzedniej listy zadań, sprawdza czy tworzą one poprawny składniowo program w języku **Imperator**, przy czym jeśli program jest poprawny, to oddaje term `PROGRAM` postaci zdefiniowanej poniżej¹:

```

PROGRAM = [ ]
PROGRAM = [ INSTRUKCJA | PROGRAM ]

INSTRUKCJA = assign(ID, WYRAŻENIE)
INSTRUKCJA = read(ID)
INSTRUKCJA = write(WYRAŻENIE)
INSTRUKCJA = if(WARUNEK, PROGRAM)
INSTRUKCJA = if(WARUNEK, PROGRAM, PROGRAM)
INSTRUKCJA = while(WARUNEK, PROGRAM)

WYRAŻENIE = id(ID)
WYRAŻENIE = int(NUM)
WYRAŻENIE = WYRAŻENIE + WYRAŻENIE
WYRAŻENIE = WYRAŻENIE - WYRAŻENIE
WYRAŻENIE = WYRAŻENIE * WYRAŻENIE
WYRAŻENIE = WYRAŻENIE / WYRAŻENIE
WYRAŻENIE = WYRAŻENIE mod WYRAŻENIE

WARUNEK = WYRAŻENIE == WYRAŻENIE
WARUNEK = WYRAŻENIE <= WYRAŻENIE
WARUNEK = WYRAŻENIE < WYRAŻENIE
WARUNEK = WYRAŻENIE > WYRAŻENIE

```

¹Termy takie omówiono na wykładzie prezentując interpreter języka **Imperator**.

```

WARUNEK = WYRAŻENIE =< WYRAŻENIE
WARUNEK = WYRAŻENIE >= WYRAŻENIE
WARUNEK = WARUNEK ; WARUNEK
WARUNEK = WARUNEK , WARUNEK

```

Przykład

Założmy, że w pliku `ex1.prog` znajduje się następujący program:

```

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

```

Wynik analizy składniowej powyższego programu:

```

?- open('ex1.prog', read, X), scanner(X, Y), close(X),
   phrase(program(PROGRAM), Y).
X = <stream>(0x7f8a01714ef0),
Y = [key(read), id('N'), sep(;;), id('SUM'), sep(:=), int(0),
     sep(;;), key(while), id(...)|...],
PROGRAM = [read('N'), assign('SUM', int(0)), while(id('N')>int(0),
            [assign('SUM', id('SUM')+id('N')), assign('N',
            id('N')-int(1))]), write(id('SUM'))]

```

Zadanie 2 (3 pkt)

Napisz predykat `wykonaj(NazwaPliku)`, który otwiera strumień znaków (nazwa pliku podana w argumencie wywołania), skanuje z niego tokeny (zadanie 1 z listy 5), następnie parserem analizuje listę tokenów (zadanie 1 z listy 6) a na końcu interpretuje program reprezentowany termem uzyskanym w parserze (interpreter przedstawiono na wykładzie 6).

Przykład

Założmy, że w pliku `ex1.prog` znajduje się następujący program:

```

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

```

Przykład uruchomienia powyższego programu:

```
?- wykonaj('ex1.prog').
|: 10.
55
true .
```

```
?- wykonaj('ex1.prog').
|: 100.
5050
true .
```

Zadanie 3 (2 pkt)

- Napisz gramatykę metamorficzne akceptujące słowa z języka $a^n b^n$, gdzie $n \geq 0$.
- Napisz gramatykę metamorficzne akceptujące słowa z języka $a^n b^n c^n$, gdzie $n \geq 0$.
- Napisz gramatykę metamorficzne akceptujące słowa z języka $a^n b^{fib(n)}$, gdzie $n \geq 0$ a $fib(n)$ jest n -tym wyrazem ciągu Fibonacciego.

Zdefiniowano następującą gramatykę metamorficzną:

```
p([ ]) —> [ ].
p([X | Xs]) —> [X], p(Xs).
```

Jak jest zależność między listami L1, L2 i L3, jeśli spełniają one warunek `phrase(p(L1), L2, L3)`?