Programowanie w Logice

Przykłady programów

Przemysław Kobylański

Przykłady programów

Interpreter prostego języka imperatywnego

- Język **Imperator**[1] jest prostym językiem imperatywnym.
- Jego składnię opisuje poniższa gramatyka BNF:

```
  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
```

[1] Nazwę zaproponował MKI.

Przykłady programów

Interpreter prostego języka imperatywnego

```
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 )
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
   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 )
```

Przykłady programów

Interpreter prostego języka imperatywnego

Example (Obliczenie sumy liczb A i B)

```
read A;
read B;
X := A;
Y := B;
while Y > 0 do
    X := X + 1;
    Y := Y - 1;
od;
write X;
```

Przykłady programów

Interpreter prostego języka imperatywnego

- Program reprezentowany jest listą złożoną z termów.
- Każdy term reprezentuje jedną instrukcję.
- Jeśli instrukcja jest złożona, to term zawiera jako podterm listę termów reprezentujących zagnieżdżone instrukcje.

```
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)
```

Przykłady programów

Interpreter prostego języka imperatywnego

Wyrażenie jest albo prostym wyrażeniem będącym zmienną lub liczbą naturalną albo wyrażeniem złożonym będącym sumą, różnicą, iloczynem lub ilorazem dwóch wyrażeń:

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

Przykłady programów

Interpreter prostego języka imperatywnego

Warunek jest relacją równości, różności, mniejszości, większości (słabej lub silnej) między wartościami dwóch wyrażeń albo alternatywą lub koniunkcją dwóch warunków:

```
WARUNEK = WYRAŻENIE =:= WYRAŻENIE
WARUNEK = WYRAŻENIE =\= WYRAŻENIE
WARUNEK = WYRAŻENIE < WYRAŻENIE
WARUNEK = WYRAŻENIE > WYRAŻENIE
WARUNEK = WYRAŻENIE =< WYRAŻENIE
WARUNEK = WYRAŻENIE >= WYRAŻENIE
WARUNEK = WARUNEK ; WARUNEK
WARUNEK = WARUNEK , WARUNEK
```

Przykłady programów

Interpreter prostego języka imperatywnego

Example (Sumowanie liczb od 1 do N)

```
program1([
    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'))]).
```

Przykłady programów

Interpreter prostego języka imperatywnego

Example (Obliczenie ilorazu D i reszty R z dzielenia M przez N)

```
program2([
    read('M'),
    read('N'),
    assign('D', int(0)),
    assign('R', id('M')),
    while(id('R') >= id('N'),
        [assign('D', id('D') + int(1)),
         assign('R', id('R') - id('N'))]),
    write(id('D')),
    write(id('R'))]).
```

Przykłady programów

Interpreter prostego języka imperatywnego

- Bieżący stan obliczeń zapisywać będziemy w postaci listy asocjacji.
- Każda asocjacja jest termem w postaci `ID = Wartość`, gdzie `ID` jest nazwą zmiennej a `Wartość` jest bieżącą wartością tej zmiennej.

```
% podstaw(+Stare, +ID, +Wartość, -Nowe)
podstaw([], ID, N, [ID = N]).
podstaw([ID=_ | AS], ID, N, [ID=N | AS]) :- !.
podstaw([ID1=W1 | AS1], ID, N, [ID1=W1 | AS2]) :-
    podstaw(AS1, ID, N, AS2).

% pobierz(+Asocjacje, +ID, -Wartość)
pobierz([ID=N | _], ID, N) :- !.
pobierz([_ | AS], ID, N) :-
    pobierz(AS, ID, N).
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
% wartość(+Wyrażenie, +Asocjacje, -Wartość)
wartość(int(N), _, N).
wartość(id(ID), AS, N) :-
    pobierz(AS, ID, N).
wartość(W1 + W2, AS, N) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N is N1 + N2.
wartość(W1 - W2, AS, N) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N is N1 - N2.
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
wartość(W1 * W2, AS, N) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N is N1 * N2.
wartość(W1 / W2, AS, N) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N2 =\= 0, N is N1 div N2.
wartość(W1 mod W2, AS, N) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N2 =\= 0,
    N is N1 mod N2.
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
% prawda(+Warunek, +Asocjacje)
prawda(W1 =:= W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 =:= N2.
prawda(W1 =\= W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 =\= N2.
prawda(W1 < W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 < N2.
prawda(W1 > W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 > N2.
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
prawda(W1 >= W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 >= N2.
prawda(W1 =< W2, AS) :-
    wartość(W1, AS, N1), wartość(W2, AS, N2),
    N1 =< N2.
prawda((W1, W2), AS) :-
    prawda(W1, AS),
    prawda(W2, AS).
prawda((W1; W2), AS) :-
    (   prawda(W1, AS),
        !
    ;   prawda(W2, AS)).
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
% interpreter(+Program, +Asocjacje)
interpreter([], _).
interpreter([read(ID) | PGM], ASSOC) :- !,
    read(N),
    integer(N),
    podstaw(ASSOC, ID, N, ASSOC1),
    interpreter(PGM, ASSOC1).
interpreter([write(W) | PGM], ASSOC) :- !,
    wartość(W, ASSOC, WART),
    write(WART), nl,
    interpreter(PGM, ASSOC).
interpreter([assign(ID, W) | PGM], ASSOC) :- !,
    wartość(W, ASSOC, WAR),
    podstaw(AS, ID, WAR, ASSOC1),
    interpreter(PGM, ASSOC1).
```

Przykłady programów

Interpreter prostego języka imperatywnego

```
interpreter([if(C, P) | PGM], ASSOC) :- !,
    interpreter([if(C, P, []) | PGM], ASSOC).
interpreter([if(C, P1, P2) | PGM], ASSOC) :- !,
    (   prawda(C, ASSOC)
    ->  append(P1, PGM, DALEJ)
    ;   append(P2, PGM, DALEJ)),
    interpreter(DALEJ, ASSOC).
interpreter([while(C, P) | PGM], ASSOC) :- !,
    append(P, [while(C, P)], DALEJ),
    interpreter([if(C, DALEJ) | PGM], ASSOC).

% interpreter(+Program)
interpreter(PROGRAM) :-
    interpreter(PROGRAM, []).
```

Przykłady programów

Interpreter prostego języka imperatywnego

Example (Sumowanie)

```
?- program1(X), interpreter(X).
|: 10.
55
X = [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'))].
```

Przykłady programów

Interpreter prostego języka imperatywnego

Example (Iloraz i reszta)

```
?- program2(X), interpreter(X).
|: 123.
|: 13.
9
6
X = [read('M'), read('N'), assign('D', int(0)),
     assign('R', id('M')), while(id('R')>=id('N'),
     [assign('D', id(...)+int(...)), assign('R', ...
      - ...)]), write(id('D')), write(id('R'))].

?- X is 9*13+6.
X = 123.
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

- Zakładamy, że predykaty zapisane są w postaci klauzul, których ciała zawierają tylko koniunkcje formuł atomowych (nie ma negacji, alternatyw i implikacji).
- Wszystkie predykaty są zdefiniowane przez nas (nie korzystamy z predykatów wbudowanych, które mogą być napisane np. w języku C).

Przykłady programów

Interpreter podzbioru Prologu w Prologu

Example (Konkatenacja list na liście)

```
app([], X, X).
app([X | L1], L2, [X | L3]) :-
    app(L1, L2, L3).

app([], []).
app([L1 | L2], L3) :-
    app(L1, L4, L3),
    app(L2, L4).
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

- Predykat `clause(H, B)` dostarcza głowę `H` i ciało `B` klauzuli.
- `H` jest unifikowane z głową a `B` z ciałem klauzuli.
- Fakt ma ciało równe `true`.

```
?- clause(app([], A, B), C).
A = B,
C = true.

?- clause(app(A, B, C), D).
A = [],
B = C,
D = true ;
A = [_G4888|_G4889],
C = [_G4888|_G4892],
D = app(_G4889, B, _G4892).
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

```
?- clause(app(A, B), C).
A = B, B = [],
C = true ;
A = [_G4870|_G4871],
C =  (app(_G4870, _G4874, B), app(_G4871, _G4874)).
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

- Interpreter Prologu w Prologu zapiszemy w postaci predykatu `udowodnij(Cel)`, gdzie `Cel` jest zadanym celem do udowodnienia.
- Celem może być warunek `true`, formuła atomowa lub koniunkcja dwóch celów.

```
udowodnij(true) :- !.
udowodnij((G1, G2)) :- !,
    udowodnij(G1),
    udowodnij(G2).
udowodnij(A) :-
    clause(A, B),
    udowodnij(B).
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

```
?- udowodnij(app(X, Y, [1, 2, 3])).
X = [],
Y = [1, 2, 3] ;
X = [1],
Y = [2, 3] ;
X = [1, 2],
Y = [3] ;
X = [1, 2, 3],
Y = [] ;
false.
```

Przykłady programów

Interpreter podzbioru Prologu w Prologu

```
?- udowodnij(app([1, 2, 3], X, [1, 2, 3, 4 | Y])).
X = [4|Y].

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

Przykłady programów

Unifikacja i reprezentacja termów

Idea algorytmu unifikacji termów:

```
unify(V,  T ) :- var(V), !, bind(V, T).
unify(T,  V ) :- var(V), !, bind(V, T).
unify(T1, T2) :- T1 =.. [Fun | Args1],
                 T2 =.. [Fun | Args2],
                 unify_args(Args1, Args2).

unify_args([], []).
unify_args([T1 | A1], [T2 | A2]) :-
    unify(T1, T2),
    unify_args(A1, A2).

bind(V, T) :- V = T.

?- unify(f(A, g(A)), f(a, B)).
A = a, B = g(a).
```

Przykłady programów

Unifikacja i reprezentacja termów

Reprezentacja termu w WAM (ang. Warren Abstract Machine):

A	str | A'	pod adresem *A* jest referencja do struktury pamiętanej pod adresem *A'*
A	ref | A'	pod adresem *A* jest referencja do adresu *A'*
A	fun/n	pod adresem *A* jest *n*-argumentowy funktor **fun**

Dwa termy **f(A, g(A))** i **f(a, B)** zajmują 11 komórek pamięci:

0	str	1	f(
1	f/2		
2	ref	2	A,
3	str	4	g(
4	g/1		
5	ref	2	A))
6	str	7	f(
7	f/2		
8	str	10	a,
9	ref	9	B)
10	a/0		

dref(A, X) do jakiego miejsca X prowadzi adres A?

```
?- dref(6, X).
X = 6.
?- dref(5, X).
X = 2.
```

bind(A, X) niech referencja pod adresem A prowadzi do miejsca X

Example (Unifikacja w WAM)

Reprezentacja komórek pamięci w postaci dynamicznych faktów:

```
:- dynamic store/2.

store(0, str(1)).    % f(
store(1, f/2).       %
store(2, ref(2)).    %   A,
store(3, str(4)).    %   g(
store(4, g/1).       %
store(5, ref(2)).    %     A))
store(6, str(7)).    % f(
store(7, f/2).       %
store(8, str(10)).   %   a,
store(9, ref(9)).    %   B)
store(10, a/0).      %
```


Example (Unifikacja w WAM cd.)

Pomocnicze predykaty:

```
dref(A, R) :-
        store(A, ref(V)),
        A =\= V, !,
        dref(V, R).
dref(A, A).

bind(A1, A2) :-
        retract(store(A1, _)),
        assert(store(A1, ref(A2))).
```


Example (Unifikacja w WAM cd.)

```
unify(A1, A2) :-
        dref(A1, D1),
        dref(A2, D2),
        unify2(D1, D2).

unify2(D, D) :- !.
unify2(D1, D2) :-
        store(D1, S1),  % S1 = ref(A1) lub S1 = str(A1)
        store(D2, S2),  % S2 = ref(A2) lub S2 = str(A2)
        S1 =.. [T1, A1],
        S2 =.. [T2, A2],
        unify3(D1, T1, A1, D2, T2, A2).
```


Example (Unifikacja w WAM cd.)

```
unify3(D1, ref, _,  D2, _,   _ ) :- !, bind(D1, D2).
unify3(D1, _,   _,  D2, ref, _ ) :- !, bind(D2, D1).
unify3(_,  _,   V1, _,  _,   V2) :- store(V1, F/N),
                                    store(V2, F/N),
                                    A1 is V1+1,
                                    A2 is V2+1,
                                    unify4(N, A1, A2).

unify4(0, _,  _ ) :- !.
unify4(N, A1, A2) :- N1 is N-1,
                     unify(A1, A2),
                     NA1 is A1+1,
                     NA2 is A2+1,
                     unify4(N1, NA1, NA2).
```

Example (Unifikacja w WAM cd.)

Przykładowe wywołanie:

```
?- unify(0, 6). % odpowiada f(A, g(A)) = f(a, B)
true.
```

Jaki jest wynik unifikacji? Jakie wartości przyjęły zmienne?

Example (Unifikacja w WAM cd.)

Zmiany zachodzące w komórkach pamięci:

0	str	1	f(
1	f/2		
2	ref	2	A,
3	str	4	g(
4	g/1		
5	ref	2	A))
6	str	7	f(
7	f/2		
8	str	10	a,
9	ref	9	B)
10	a/0		

bind(2, 8)

0	str	1	f(
1	f/2		
2	ref	8	a,
3	str	4	g(
4	g/1		
5	ref	2	a))
6	str	7	f(
7	f/2		
8	str	10	a,
9	ref	9	B)
10	a/0		

bind(9, 3)

0	str	1	f(
1	f/2		
2	ref	8	a,
3	str	4	g(
4	g/1		
5	ref	2	a))
6	str	7	f(
7	f/2		
8	str	10	a,
9	ref	3	g(a))
10	a/0		

Example (Unifikacja w WAM cd.)

Wynik unifikacji:

	0	str	1
	1	f/2	
A	2	ref	8
	3	str	4
	4	g/1	
	5	ref	2
	6	str	7
	7	f/2	
	8	str	10
B	9	ref	3
	10	a/0