Programowanie w Logice

Gramatyki metamorficzne

Przemysław Kobylański
na podstawie [CM2003] i [SS1994]

Gramatyki metamorficzne

Gramatyki bezkontekstowe

- Gramatyką bezkontekstową nazywamy uporządkowaną czwórkę $G = \langle \Sigma, N, S, P \rangle$, gdzie
  - Σ jest skończonym i niepustym alfabetem (symbole terminalne)
  - N jest skończonym i niepustym zbiorem symboli nieterminalnych
  - $S \in N$ jest wyróżnionym nieterminalem nazywanym symbolem początkowym
  - P jest skończonym zbiorem reguł produkcji postaci $n \to w$, gdzie $n \in N$ oraz $w \in (\Sigma \cup N)^*$
- Jeśli $u = u'nu''$, dla $u', u'' \in (\Sigma \cup N)^*$, oraz $n \to w \in P$, to mówimy, że słowo $v = u'wu''$ jest w jednym kroku wyprowadzalne ze słowa u, co zapisujemy $u \Rightarrow v$.

Gramatyki metamorficzne

Gramatyki bezkontekstowe

- Niech $\overset{*}{\Rightarrow}$ będzie domknięciem tranzytywnym relacji wyprowadzenia $\Rightarrow$.
- Mówimy, że słowo $w \in \Sigma^*$ jest wyprowadzalne w gramatyce G, jeśli $S \overset{*}{\Rightarrow} w$.
- Język wszystkich słów wyprowadzalnych w gramatyce G oznaczamy przez $L(G)$:

$$L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}.$$

- Symbolem ε będziemy oznaczać słowo puste.

Gramatyki metamorficzne

Gramatyki bezkontekstowe

Example (Język palindromów)

Niech $G = \langle \Sigma, N, S, P \rangle$, gdzie

- $\Sigma = \{a, b\}$
- $N = \{S\}$
- $P = \{S \to \varepsilon, S \to a, S \to b, S \to aSa, S \to bSb\}$

Wówczas język $L(G)$ składa się z palindromów nad alfabetem $\{a, b\}$:

$$L(G) = \{\varepsilon, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, \ldots\}$$

Example (Język poprawnie rozstawionych nawiasów)

Niech $G = \langle \Sigma, N, S, P \rangle$, gdzie

- $\Sigma = \{(,)\}$
- $N = \{S\}$
- $P = \{S \to \varepsilon, S \to (S)S\}$

Wówczas język $L(G)$ składa się ze słów w których poprawnie rozstawiono nawiasy:

$$L(G) = \{\varepsilon, (), ()(), (()), ()()(), ()(()), (())(), (()()), ((())), \ldots\}$$

Example (Prosta gramatyka zdań języka angielskiego)

Niech $G = \langle \Sigma, N, \mathit{zdanie}, P \rangle$, gdzie

- $\Sigma = \{\mathit{the}, \mathit{apple}, \mathit{man}, \mathit{eats}, \mathit{sings}\}$
- $$N = \left\{ \begin{array}{l} \mathit{zdanie}, \\ \mathit{fraza\_rzecz}, \\ \mathit{fraza\_czas}, \\ \mathit{przedimek}, \\ \mathit{rzeczownik}, \\ \mathit{czasownik} \end{array} \right\}$$

Example (cd.)

- $$P = \left\{ \begin{array}{l} \mathit{zdanie} \to \mathit{fraza\_rzecz}\ \mathit{fraza\_czas}, \\ \mathit{fraza\_rzecz} \to \mathit{przedimek}\ \mathit{rzeczownik}, \\ \mathit{fraza\_czas} \to \mathit{czasownik}\ \mathit{fraza\_rzecz}, \\ \mathit{fraza\_czas} \to \mathit{czasownik}, \\ \mathit{przedimek} \to \mathit{the}, \\ \mathit{rzeczownik} \to \mathit{apple}, \\ \mathit{rzeczownik} \to \mathit{man}, \\ \mathit{czasownik} \to \mathit{eats}, \\ \mathit{czasownik} \to \mathit{sings} \end{array} \right\}$$

Example (cd.)

Wówczas język $L(G)$ składa się z kilkunastu prostych zdań języka angielskiego:

$$L(G) = \left\{ \begin{array}{l} \mathit{the\ apple\ eats\ the\ apple}, \\ \mathit{the\ apple\ eats\ the\ man}, \\ \mathit{the\ apple\ sings\ the\ apple}, \\ \mathit{the\ apple\ sings\ the\ man}, \\ \mathit{the\ apple\ eats}, \\ \mathit{the\ apple\ sings}, \\ \mathit{the\ man\ eats\ the\ apple}, \\ \mathit{the\ man\ eats\ the\ man}, \\ \mathit{the\ man\ sings\ the\ apple}, \\ \mathit{the\ man\ sings\ the\ man}. \\ \mathit{the\ man\ eats}, \\ \mathit{the\ man\ sings} \end{array} \right\}$$

Gramatyki metamorficzne

Analiza składniowa w Prologu

Spróbujmy napisać analizator w postaci predykatów akceptujących listy słów:

```
zdanie(X) :-
    append(Y, Z, X),
    fraza_rzecz(Y), fraza_czas(Z).

fraza_rzecz(X) :-
    append(Y, Z, X),
    przedimek(Y), rzeczownik(Z).

fraza_czas(X) :-
    append(Y, Z, X),
    czasownik(Y), fraza_rzecz(Z).
fraza_czas(X) :-
    czasownik(X).
```

Gramatyki metamorficzne

Analiza składniowa w Prologu

```
przedimek([the]).
rzeczownik([apple]).
rzeczownik([man]).
czasownik([eats]).
czasownik([sings]).
```

Przykłady analizy i syntezy zdań:

```
?- zdanie([the, man, eats, the, apple]).
true ;
false.

?- zdanie(X).
X = [the, apple, eats, the, apple] ;
X = [the, apple, eats, the, man] ;
ERROR: Out of global stack
   Exception: (8) fraza_czas(_G21) ?
```

Gramatyki metamorficzne

Analiza składniowa w Prologu

Zapiszemy predykaty w ten sposób aby wydzielały w liście prefiks wyprowadzalny z danego nieterminala i oddawały pozostały sufiks listy:

```
zdanie(X, Z) :-
    fraza_rzecz(X, Y), fraza_czas(Y, Z).

fraza_rzecz(X, Z) :-
    przedimek(X, Y), rzeczownik(Y, Z).

fraza_czas(X, Z) :-
    czasownik(X, Y), fraza_rzecz(Y, Z).
fraza_czas(X, Y) :-
    czasownik(X, Y).
```

Gramatyki metamorficzne

Analiza składniowa w Prologu

```
przedimek([the | X], X).
rzeczownik([apple | X], X).
rzeczownik([man | X], X).
czasownik([eats | X], X).
czasownik([sings | X], X).
```

Przykład syntezy zdań:

```
?- zdanie(X, []).
X = [the, apple, eats, the, apple] ;
X = [the, apple, eats, the, man] ;
X = [the, apple, sings, the, apple] ;
X = [the, apple, sings, the, man] ;
X = [the, apple, eats] ;                X = [the, apple, sings] ;
X = [the, man, eats, the, apple] ;
X = [the, man, eats, the, man] ;
X = [the, man, sings, the, apple] ;
X = [the, man, sings, the, man] ;
X = [the, man, eats] ;                  X = [the, man, sings].
```

Gramatyki metamorficzne

Produkcje gramatyki jako klauzule

W Prologu produkcje gramatyki można zapisywać wprost w programie:

```
zdanie --> fraza_rzecz, fraza_czas.
fraza_rzecz --> przedimek, rzeczownik.
fraza_czas --> czasownik, fraza_rzecz.
fraza_czas --> czasownik.
przedimek --> [the].
rzeczownik --> [apple].
rzeczownik --> [man].
czasownik --> [eats].
czasownik --> [sings].
```

Symbole nieterminalne zapisuje się w takiej postaci jak nazwy predykatów, symbole terminalne podaje się między nawiasami kwadratowymi, strzałkę zapisuje się trzema znakami `-->`.

Gramatyki metamorficzne

Produkcje gramatyki jako klauzule

Jeżeli ten sam nieterminal występuje w więcej niż jednej produkcji, to prawe ich strony można połączyć średnikiem (alternatywa):

```
zdanie --> fraza_rzecz, fraza_czas.
fraza_rzecz --> przedimek, rzeczownik.
fraza_czas --> czasownik, fraza_rzecz; czasownik.
przedimek --> [the].
rzeczownik --> [apple]; [man].
czasownik --> [eats]; [sings].
```

Gramatyki metamorficzne

Produkcje gramatyki jako klauzule

Do analizy i syntezy służą predykaty `phrase/2` i `phrase/3`. W pierwszym przypadku ma on postać `phrase(+Nieterminal, ?ListaTerminali)`:

```
?- phrase(zdanie, [the, man, eats, the, apple]).
true .

?- findall(X, phrase(zdanie, X), L).
L = [[the,apple,eats,the,apple],[the,apple,eats,the,man],
     [the,apple,sings,the,apple],[the,apple,sings,the,man],
     [the,apple,eats],[the,apple,sings],
     [the,man,eats,the,apple],[the,man,eats,the,man],
     [the,man,sings,the,apple],[the,man,sings,the,man],
     [the,man,eats],[the,man,sings]]
```

Gramatyki metamorficzne

Produkcje gramatyki jako klauzule

W drugim przypadku ma on postać `phrase(+Nieterminal, ?ListaTerminali, ?ListaPozostałychTerminali)`, przy czym trzecim parametrem odbiera się sufiks, który nie jest wyprowadzony z podanego nieterminala:

```
?- phrase(fraza_rzecz, [the, man, eats, the, apple], X).
X = [eats, the, apple].

?- phrase(zdanie, [the, man, eats, the, apple], X).
X = [] .
```

Gramatyki metamorficzne

Produkcje gramatyki jako klauzule

Example (Język poprawnie rozstawionych nawiasów)

W SWI-Prologu, począwszy od wersji 7, aby łańcuch znaków traktowany był w postaci listy kodów należy umieścić go między odwróconymi apostrofami (w starszych wersjach między cudzysłowami).

```
s2 --> ‘‘.
s2 --> ‘(‘, s2, ‘)‘, s2.
```

Dialog:

```
?- phrase(s2, ‘(())‘).
true .

?- phrase(s2, ‘())(‘).
false.
```

Gramatyki metamorficzne

Parametry nieterminali

W Prologu nieterminale mogą mieć parametry, dzięki którym nie tylko mogą przeprowadzać analizę składniową ale również dokonywać przekształcenia sparsowanego słowa do postaci prologowego termu (stąd nazwa gramatyki metamorficzne).

Example (Kody regularnych drzew binarnych)

Regularne drzewa binarne będziemy zapisywać w postaci termów zbudowanych z funktora `f/2` (wierzchołki wewnętrzne) i stałej `a` (liście).
Drzewo kodować będziemy w postaci ciągu zer i jedynek w sposób następujący:

- Liść `a` kodujemy cyfrą 0.
- Drzewo `f(X, Y)` o dwóch poddrzewach `X` i `Y`, kodujemy w postaci ciągu złożonego kolejno z cyfry 1, następnie kodu poddrzewa `X` i na końcu kodu poddrzewa `Y`.

Gramatyki metamorficzne

Parametry nieterminali

Example (cd.)

Gramatyka metamorficzna dla kodów regularnych drzew binarnych:

```
kod(a) --> ‘0‘.
kod(f(X, Y)) --> ‘1‘, kod(X), kod(Y).
```

Dialog:

```
?- phrase(kod(X), ‘1100100‘).
X = f(f(a, a), f(a, a)) .

?- phrase(kod(f(f(a, a), a)), X), format(’~s~n’, [X]).
11000
X = [49, 49, 48, 48, 48].
```

Gramatyki metamorficzne

Cele umieszczane w produkcjach

Do produkcji gramatyki można wstawiać prologowe cele umieszczając je w nawiasach klamrowych.

Example (Liczby binarne)

```
bin(X) --> bin(0, X).
bin(X, X) --> ‘‘.
bin(X, Z) --> ‘0‘, {Y is 2*X}, bin(Y, Z).
bin(X, Z) --> ‘1‘, {Y is 2*X+1}, bin(Y, Z).
```

Dialog:

```
?- phrase(bin(N), ‘101010‘).
N = 42 .
?- phrase(bin(N), ‘11111111‘).
N = 255 .
```

Gramatyki metamorficzne

Cele umieszczane w produkcjach

Example (Gramatyka metamorficzna głównych liczebników)

```
number(0) --> [zero].
number(N) --> xxx(N).
xxx(N) --> digit(D), [hundred], rest_xxx(N1),
           {N is D*100 + N1}.
xxx(N) --> xx(N).
rest_xxx(0) --> [].
rest_xxx(N) --> [and], xx(N).
xx(N) --> digit(N).
xx(N) --> teen(N).
xx(N) --> tens(T), rest_xx(N1), {N is T+N1}.
rest_xx(0) --> [].
rest_xx(N) --> digit(N).
```

Gramatyki metamorficzne

Cele umieszczane w produkcjach

Example (cd.)

```
digit(1) --> [one].   digit(2) --> [two].
digit(3) --> [three]. digit(4) --> [four].
digit(5) --> [five].  digit(6) --> [six].
digit(7) --> [seven]. digit(8) --> [eight].
digit(9) --> [nine].

teen(10) --> [ten].      teen(11) --> [eleven].
teen(12) --> [twelve].   teen(13) --> [thirteen].
teen(14) --> [fourteen]. teen(15) --> [fifteen].
teen(16) --> [sixteen].  teen(17) --> [seventeen].
teen(18) --> [eighteen]. teen(19) --> [nineteen].
```

Gramatyki metamorficzne

Cele umieszczane w produkcjach

Example (cd.)

```
tens(20) --> [twenty]. tens(30) --> [thirty].
tens(40) --> [forty].  tens(50) --> [fifty].
tens(60) --> [sixty].  tens(70) --> [seventy].
tens(80) --> [eighty]. tens(90) --> [ninety].
```

Dialog:

```
?- phrase(number(N), [thirteen]).
N = 13 .

?- phrase(number(N), [twenty, five]).
N = 25 .

?- phrase(number(N), [six, hundred, and, sixty, six]).
N = 666 .
```

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

- Rzeczownikowi własnemu odpowiada prologowa stała np. `john`.
- Rzeczownikowi odpowiada jednoargumentowy predykat wyrażający własność np. `man(X)`.
- Czasownikowi nieprzechodniemu odpowiada jednoargumentowy predykat wyrażający własność np. `lives(X)`.
- Czasownikowi przechodniemu odpowiada dwuargumentowy predykat wyrażający relację np. `loves(X, Y)`.
- Przedimkowi **every** odpowiada kwantyfikator ogólny.
- Przedimkowi **a** odpowiada kwantyfikator szczegółowy (egzystencjalny).
- Fraza rzeczownikowa może zawierać klauzulę względną zaczynającą się od słowa **that**, która precyzuje rzeczownik podając jego dodatkowe własności i relacje.

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

```
:- op(900, xfx, =>).
:- op(800, xfy, &).
:- op(550, xfy, :).

zdanie(P) -->
    fraza_rzecz(X, P1, P), fraza_czas(X, P1).

fraza_rzecz(X, P1, P) -->
    przedimek(X, P2, P1, P), rzeczownik(X, P3),
    klauzula_wzgl(X, P3, P2).
fraza_rzecz(X, P, P) -->
    rzeczownik_wlasny(X).

fraza_czas(X, P) -->
    czasownik_przech(X, Y, P1), fraza_rzecz(Y, P1, P).
fraza_czas(X, P) -->
    czasownik_nieprzech(X, P).
```

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

```
klauzula_wzgl(X, P1, P1&P2) -->
    [that], fraza_czas(X, P2).
klauzula_wzgl(_, P, P) -->
    [].

przedimek(X, P1, P2, all(X):(P1=>P2)) -->
    [every].
przedimek(X, P1, P2, exists(X):(P1&P2)) -->
    [a].

rzeczownik(X, man(X)) -->
    [man].
rzeczownik(X, woman(X)) -->
    [woman].
```

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

```
rzeczownik_wlasny(john) -->
    [john].

czasownik_przech(X, Y, loves(X,Y)) -->
    [loves].
czasownik_przech(X, Y, knows(X,Y)) -->
    [knows].

czasownik_nieprzech(X, lives(X)) -->
    [lives].
```

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

Dialog:

```
?- phrase(zdanie(X), [john, lives]).
X = lives(john).

?- phrase(zdanie(X), [john, knows, john]).
X = knows(john, john) .

?- phrase(zdanie(X), [john, knows, a, woman]).
X = exists(G1): (woman(G1)&knows(john, G1)) .

?- phrase(zdanie(X), [john, loves, every, woman]).
X = all(G1): (woman(G1)=>loves(john, G1)) .

?- phrase(zdanie(X), [every, man, loves, a, woman]).
X = all(G1): (man(G1)=>exists(G2):
                        (woman(G2)&loves(G1, G2))) .
```

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

Dialog:

```
?- phrase(zdanie(X), [john, loves, every, woman,
                      that, loves, john]).
X = all(G1): (woman(G1)&loves(G1, john)=>loves(john, G1))
```

$$\forall_{G_1}(woman(G_1) \wedge loves(G_1, john) \rightarrow loves(john, G_1))$$

```
?- phrase(zdanie(X), [every, man, loves, every, woman,
                      that, knows, john]).
X = all(G1): (man(G1)=>all(G2):
              (woman(G2)&knows(G2, john)=>loves(G1, G2)))
```

$$\forall_{G_1}(man(G_1) \rightarrow \forall_{G_2}(woman(G_2) \wedge knows(G_2, john) \rightarrow loves(G_1, G_2)))$$

Gramatyki metamorficzne

Tłumaczenie zdań na formuły rachunku predykatów

Dialog:

```
?- phrase(zdanie(X), [john, loves, a, woman,
                      that, knows, a, man,
                      that, knows, john]).
X = exists(G1):
      ((woman(G1)&exists(G2):
                  ((man(G2)&knows(G2, john))&
                    knows(G1, G2)))&loves(john, G1))
```

$$\exists_{G_1}(woman(G_1) \wedge \exists_{G_2}(man(G_2) \wedge knows(G_2, john) \wedge knows(G_1, G_2)) \wedge loves(john, G_1))$$

- G_1 pewna kobieta, którą kocha John
- G_2 pewien mężczyzna, który zna Johna i którego zna kobieta G_1