PATR, část 4

Gramatika 4

- Pořadí valencí v rámci je jiné než v předchozích verzích gramatiky: pro větný vzorec 'podmět sloveso předmět1 předmět2' jsme zatím měli v rámci pořadí 'předmět1 předmět2 podmět', zde je pořadí intuitivnější: 'podmět předmět1 předmět2'. Viz pravidlo 3.
- makra pro zápis opakujících se úseků gramatiky
- lexikální pravidla pro odvozování lexikálních kategorií
- pasivum

Makra

V předchozích verzích gramatiky se vyskytovaly v bezkontextových pravidlech: `s(S) ---> np(NP), vp(VP)`. Makra umožňují zapsat (definovat) omezení v gramatice jen jednou a pak se na ně odvolávat na různých místech programu. Můžeme jim dát i lingvistickou interpretaci a považovat je za zobecnění, která odpovídají principům gramatiky a vlastnostem tříd slov nebo frází. Tak například makro `hfp(Mother,HeadDaughter)` (HEAD FEATURE PRINCIPLE) vyjadřuje identitu hodnot atributů HEAD u matky a řídící dcery. Jeho definice je jednoduchá:

(1)

```
hfp(Mother, HeadDaughter) :-
    Mother/head *= HeadDaughter/head.
```

V pravidle se odkaz na definici makra provede takto:

(2)

```
%   sentence formation
1 # s(S) ---> np(NP), vp(VP) :: hfp(S, VP), finite(S),
                                VP/syncat/first *= NP,
                                VP/syncat/rest *= end.
```

V QPATRu makra musí mít parametry (musí být explicitně určeno, kterých objektů se definice týká). V definici makra může být uveden odkaz na jiné makro.

Makra lze použít všude, tedy v pravidlech (např. hfp), ve slovníku (u sloves v gramatice 4 jsou uvedena vždy dvě makra, první určuje morfologické kategorie a druhé valenci spolu se sémantickou interpretací) a v definicích jiných maker.

Některá makra použitá v gramatice 4:[1]

hfp/2 – Head Feature Principle

m3s/1 – maskulinum, 3. osoba singuláru (pro substantiva)

m3p/1 – maskulinum, 3. osoba plurálu (pro substantiva)

third_singular/1 – určitý tvar slovesa pro 3. osobu singuláru

plural/1 – určitý tvar slovesa pro všechny osoby plurálu

nonfinite/1 – neurčitý tvar slovesa

[1] Číslo za lomítkem vyjadřuje počet parametrů.

intransitive/2 – toto makro definuje hodnoty atributů TRANS a SYNCAT (tedy sémantickou interpretaci a rámec) pro nepřechodné sloveso

transitive/2 – totéž pro přechodné sloveso, definice tohoto makra se všemi dalšími volanými makry viz níže

raising_so/2 – "subject-to-object raising", správně by ale mělo být "subject-to-object equi", toto makro definuje sémantickou interpretaci a rámec slovesa, které vyžaduje doplnění dvěma předměty: NP a VP; závislé sloveso je přitom v infinitivu a jeho nevyjádřený podmět je koindexován s prvním předmětem řídícího slovesa

perfective/1 – definuje sémantickou interpretaci a rámec pomocného slovesa *have* pro perfektum, sloveso je typu "subject-to-subject raising": v sémantické interpretaci se podmět tohoto slovesa projeví jen jako argument významového slovesa

passive_voice/1 – definuje sémantickou interpretaci a rámec pomocného slovesa *be* pro pasivum, sloveso je typu "subject-to-subject raising": v sémantické interpretaci se podmět tohoto slovesa projeví jen jako argument významového slovesa

Definice některých maker jsou poměrně složité, neboť obsahují volání dalších maker:

(3)

```
transitive(F, TR)
   tv(F, TR)
      iv(F, TR)
         predicate(F, TR)
            F/head/trans/pred *= TR
         subject(F)
            np_comp1(F, G)
               comp1(F, G)
                  F/syncat/first *= G
               np(G)
                  category_label(G, np)
                     G/cat *= np
            G/head/trans *= F/head/trans/arg1
      direct_object(F)
         np_comp2(F, G)
            comp2(F, G)
               F/syncat/rest/first *= G
            np(G)
               category_label(G, np)
                  G/cat *= np
         G/head/trans *= F/head/trans/arg2
   two_comps(F)
      F/syncat/rest/rest *= end
```

Je-li TR = storm_, výsledkem volání makra transitive(F, TR) bude tato sestava rysů:

(4)
$$\begin{bmatrix} \text{head} & \begin{bmatrix}\text{trans} & \begin{bmatrix}\text{pred} & \text{storm\_}\\ \text{arg1} & \boxed{1}\\ \text{arg2} & \boxed{2}\end{bmatrix}\end{bmatrix}\\ \text{syncat} & \begin{bmatrix}\text{first} & \begin{bmatrix}\text{cat} & \text{np}\\ \text{head} & \begin{bmatrix}\text{trans} & \boxed{1}\end{bmatrix}\end{bmatrix}\\ \text{rest} & \begin{bmatrix}\text{first} & \begin{bmatrix}\text{cat} & \text{np}\\ \text{head} & \begin{bmatrix}\text{trans} & \boxed{2}\end{bmatrix}\end{bmatrix}\\ \text{rest} & \text{end}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Lexikální pravidla

Lexikální pravidla specifikují vztah mezi dvěma lexikálními kategoriemi, narozdíl od pravidel syntaktických, která specifikují vztah mezi dvěmi a více kategoriemi, z nichž alespoň jedna (matka) je kategorie frázová. Lze na ně pohlížet jako na zobecněné vyjádření vztahů mezi různými lexikálními kategoriemi (slovníkovými hesly), např. mezi tvary téhož paradigmatu, nebo jako na způsob, jakým lze z jednoho tvaru (třeba základního) odvodit tvar jiný. Lexikálními pravidly se v unifikačních přístupech může řešit flektivní a derivační morfologie, lexikální alternace a někdy i problémy tradičně považované za syntaktické (adjunkty, extrapozice). Alternativně lze použít hierarchické uspořádání lexikálních kategorií.

V QPATRu se lexikální pravidla formálně neliší od pravidel syntaktických. Nelze však upravovat tvary slov.[2] Jediné pravidlo lexikální pravidlo v gramatice 4 vyrobí z aktivního tvaru minulého příčestí homonymní pasivní tvar. Narozdíl od syntaktických pravidel zde neplatí head feature principle.

(5)
$$\begin{bmatrix} \text{head} & \begin{bmatrix}\text{form} & \text{passive}\\ \text{trans} & \boxed{2}\end{bmatrix}\\ \text{syncat} & \boxed{1}\end{bmatrix} \longrightarrow \begin{bmatrix} \text{head} & \begin{bmatrix}\text{form} & \text{pastparticiple}\\ \text{trans} & \boxed{2}\end{bmatrix}\\ \text{syncat} & \begin{bmatrix}\text{rest} & \boxed{1}\end{bmatrix}\end{bmatrix}$$

Tedy například z lexikální kategorie odpovídající minulému příčestí slovesa *stormed*

(6)
$$\begin{bmatrix} \text{cat} & \text{v}\\ \text{head} & \begin{bmatrix}\text{form} & \text{pastparticiple}\\ \text{trans} & \boxed{2}\begin{bmatrix}\text{pred} & \text{storm\_}\\ \text{arg1} & \boxed{3}\\ \text{arg2} & \boxed{4}\end{bmatrix}\end{bmatrix}\\ \text{syncat} & \begin{bmatrix}\boxed{1}\ \text{first} & \begin{bmatrix}\text{cat} & \text{np}\\ \text{head} & \begin{bmatrix}\text{trans} & \boxed{3}\end{bmatrix}\end{bmatrix}\\ \text{rest} & \begin{bmatrix}\text{first} & \begin{bmatrix}\text{cat} & \text{np}\\ \text{head} & \begin{bmatrix}\text{trans} & \boxed{4}\end{bmatrix}\end{bmatrix}\\ \text{rest} & \text{end}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

vznikne lexikální kategorie odpovídající trpnému příčestí téhož tvaru:

[2] S tvary slov lze pracovat ve formalismu DATR pro specifikaci hierarchického slovníku spolu s morfologií. Slovník ve formátu QDATR (viz odkaz na www stránce kursu) lze používat spolu s QPATRem.

(7) $$\begin{bmatrix} \text{cat} & \text{v} \\ \text{head} & \begin{bmatrix} \text{form} & \text{passive} \\ \text{trans} & \boxed{2} \begin{bmatrix} \text{pred} & \text{storm\_} \\ \text{arg1} & \boxed{3} \\ \text{arg2} & \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{syncat} & \begin{bmatrix} \boxed{1}\ \text{first} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{head} & \begin{bmatrix} \text{trans} & \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{rest} & \text{end} \end{bmatrix} \end{bmatrix}$$

Úkoly

Zkuste vytvořit jednoduchou gramatiku pro svůj mateřský nebo libovolný jiný jazyk kromě angličtiny. Můžete přitom vyjít z některé verze anglické gramatiky a soustředit se na řešení jen některých jevů.

Příloha 1: Gramatika 4 v kódu QPATR

```
/*********************************************************************/
/*  SHIEBER4.GRM                                                     */
/*                                                                   */
/*  demonstration grammar                                            */
/*  (based on demonstration grammar four with major modifications)  */
/*     subject-verb agreement                                        */
/*     complex subcategorization                                     */
/*     logical form construction                                     */
/*     lexical organization by templates (macros)                    */
/*     lexical rules                                                 */
/*                                                                   */
/*  Stuart M. Shieber, An Introduction to Unification-Based          */
/*  Approaches to Grammar. Stanford, 1986.                           */
/*********************************************************************/

% grammar rules *****************************************************

%   sentence formation
1 # s(S) ---> np(NP), vp(VP) :: hfp(S, VP), finite(S),
                                VP/syncat/first *= NP,
                                VP/syncat/rest *= end.

%   trivial verb phrase
2 # vp(VP) ---> v(V) :: hfp(VP, V), VP/syncat *= V/syncat.

%   complements
3 # vp(VP_1) ---> vp(VP_2), xp(XP) ::
                     hfp(VP_1, VP_2),
                     VP_2/syncat/first *= VP_1/syncat/first,
                     VP_2/syncat/rest/first *= XP,
                     VP_2/syncat/rest/rest  *= VP_1/syncat/rest.


% lexical rules *****************************************************

%   passive formation
```

```
4 # v(V1) ---> v(V2) ::
                np_comp2(V2, NP), past_participle(V2), passive(V1),
                V1/syncat *= V2/syncat/rest,
                V1/head/trans *= V2/head/trans.


% lexicon ***********************************************************

uther     lex  np(F) :: m3s(F), F/head/trans *= uther_.
cornwall  lex  np(F) :: m3s(F), F/head/trans *= cornwall_.
knights   lex  np(F) :: m3p(F), F/head/trans *= knights_.
%-------------------------------------------------------------------
sleeps    lex  v(F) :: third_singular(F), intransitive(F, sleep_).
sleep     lex  v(F) :: plural(F), intransitive(F, sleep_).
sleep     lex  v(F) :: nonfinite(F), intransitive(F, sleep_).
slept     lex  v(F) :: past_participle(F), intransitive(F, sleep_).
%-------------------------------------------------------------------
storms    lex  v(F) :: third_singular(F), transitive(F, storm_).
storm     lex  v(F) :: plural(F), transitive(F, storm_).
storm     lex  v(F) :: nonfinite(F), transitive(F, storm_).
stormed   lex  v(F) :: past_participle(F), transitive(F, storm_).
%-------------------------------------------------------------------
persuades lex  v(F) :: third_singular(F), raising_so(F, persuade_).
persuade  lex  v(F) :: plural(F), raising_so(F, persuade_).
persuade  lex  v(F) :: nonfinite(F), raising_so(F, persuade_).
persuaded lex  v(F) :: past_participle(F), raising_so(F, persuade_).
%-------------------------------------------------------------------
has       lex  v(F) :: third_singular(F), perfective(F).
have      lex  v(F) :: plural(F), perfective(F).
have      lex  v(F) :: nonfinite(F), perfective(F).
%-------------------------------------------------------------------
is        lex  v(F) :: third_singular(F), passive_voice(F).
are       lex  v(F) :: plural(F), passive_voice(F).
be        lex  v(F) :: nonfinite(F), passive_voice(F).
been      lex  v(F) :: past_participle(F), passive_voice(F).
%-------------------------------------------------------------------
to        lex  v(F) :: infinitival(F), infinitive(F).

% templates (macros) ***********************************************

m3s(F) :- F/head/agreement *= G,  G/gender *= masculine,
                                  G/person *= third,
                                  G/number *= singular.

m3p(F) :- F/head/agreement *= G,  G/gender *= masculine,
                                  G/person *= third,
                                  G/number *= plural.

third_singular(F) :-
     finite(F),
     F/syncat/first/head/agreement *= G,
     G/person *= third, G/number *= singular.

plural(F) :-
     finite(F),
     F/syncat/first/head/agreement *= G,
```

```
     G/number *= plural.

finite(F) :-             F/head/form *= finite.
nonfinite(F) :-          F/head/form *= nonfinite.
past_participle(F) :-    F/head/form *= pastparticiple.
infinitival(F) :-        F/head/form *= infinitival.

passive(F) :-            F/head/form *= passive.

%-------------------------------------------------------------------

hfp(Mother, Head) :- Mother/head *= Head/head.  % head feature principle

passive_voice(F) :-
     empty(F, G, H), passive(H),
     F/head/trans/arg1 *= unspecified,
     F/head/trans/arg2 *= G/head/trans.

infinitive(F) :-
     empty(F, G, H), nonfinite(H),
     np_comp1(H, G).

perfective(F) :- auxiliary(F, perfective_), comp2(F, G), past_participle(G).


empty(F, G, H) :-
     np_comp1(F, G), vp_comp2(F, H),
     F/head/trans *= H/head/trans, two_comps(F).

intransitive(F, TR) :- iv(F, TR), one_comp(F).
auxiliary(F, TR) :-    av(F, TR), two_comps(F).
transitive(F, TR) :-   tv(F, TR), two_comps(F).
raising_so(F, TR) :-   ov(F, TR), three_comps(F). % subject-to-object

iv(F, TR) :- predicate(F, TR),  subject(F).
av(F, TR) :- predicate(F, TR),  lowered_subject(F).
tv(F, TR) :- iv(F, TR),         direct_object(F).
ov(F, TR) :- tv(F, TR),         object_control(F).


predicate(F, TR) :- F/head/trans/pred *= TR.

subject(F) :-
     np_comp1(F, G),
     G/head/trans *= F/head/trans/arg1.

lowered_subject(F) :-
     np_comp1(F, G),
     vp_comp2(F, H), comp1(H, G),
     H/head/trans *= F/head/trans/arg1.

direct_object(F) :-
     np_comp2(F, G),
     G/head/trans *= F/head/trans/arg2.

object_control(F) :-
```

```
     vp_comp3(F, G), infinitival(G),
     G/head/trans *= F/head/trans/arg3,
     np_comp2(F, H), np_comp1(G, H).

%----------------------------------------------------------------------

np_comp1(F, G) :- comp1(F, G), np(G).
np_comp2(F, G) :- comp2(F, G), np(G).
vp_comp2(F, G) :- comp2(F, G), vp(G), one_comp(G).
vp_comp3(F, G) :- comp3(F, G), vp(G), one_comp(G).

comp1(F, G) :-  F/syncat/first *= G.
comp2(F, G) :-  F/syncat/rest/first *= G.
comp3(F, G) :-  F/syncat/rest/rest/first *= G.

one_comp(F)    :- F/syncat/rest *= end.
two_comps(F)   :- F/syncat/rest/rest *= end.
three_comps(F) :- F/syncat/rest/rest/rest *= end.

% category labels ****************************************************

category_label(F, C) :- F/cat *= C.

s(F)  :- category_label(F, s ).
np(F) :- category_label(F, np).
vp(F) :- category_label(F, vp).
v(F)  :- category_label(F, v ).
xp(F). % dummy label

% semantic representation ********************************************

semantic_representation(F, LF) :- F/head/trans *= LF.

% example sentences **************************************************

ex1( 1, s, [uther, sleeps]).
ex1( 2, s, [knights, sleep]).
ex1( 3, s, [uther, storms, cornwall]).
ex1( 4, s, [uther, has, stormed, cornwall]).
ex1( 5, s, [knights, have, stormed, cornwall]).
ex1( 6, s, [uther, persuades, knights, to, sleep]).
ex1( 7, s, [uther, persuades, knights, to, storm, cornwall]).
ex1( 8, s, [knights, have, persuaded, uther, to, storm, cornwall]).
ex1( 9, s, [cornwall, is, stormed]).
ex1(10, s, [knights, are, stormed]).
ex1(11, s, [uther, is, persuaded, to, sleep]).
ex1(12, s, [knights, are, persuaded, to, storm, cornwall]).
ex1(13, s, [cornwall, has, been, persuaded, to, be, stormed]).
```