— Notes —

The `movewh` *Grammar*

1. This grammar allows two kinds of movement: Wh movement of the standard sort (subjects and objects can be extracted to form questions), and local V-movement, implementing subject-auxiliary inversion. Whether this latter construction is really a matter of movement (i.e, whether it shares any important properties with Wh-movement) is open to doubt, but it illustrates the possibility of passing around non-local information via more than one "channel". Actually, it would have been more accurate to call them "registers", since they can only contain a single item at a time.

```
rec( Sentence ):-
        s_matrix( Sentence, [] ).


dump_lang( [] ):- nl.
dump_lang( [S|Ss] ):-
        print( S ),
        nl,
        dump_lang( Ss ).

%%% s_matrix gives us a single start symbol,
%%% instead of three.
s_matrix  --> s( decl, nogap ).
s_matrix --> s( whQ, nogap ).
s_matrix --> s( ynQ, nogap ).
```

Figure 1: User interface predicates. The start symbol.

2. "Paradigmatic coordinates" are given in different orders for verbs and pronouns. In particular, the order for verbs is (Tense,Number,Person) but for pronouns it is (Person,Number,Case). The orders follow the organization of paradigms in traditional grammars. (In particular, E.V.Gordon's Introduction to Old Norse...)

3. Major Categories

- s(ClauseType, WhGapChannel)
- predicate(Agreement, VGapChannel, WhGapChannel)
- peri(VerbForm, WhGapChannel)
- vp(VerbForm, WhGapChannel)

```
s( decl, WhGap ) -->
        np( $(Agr,nom), -wh ),
        predicate( Agr, nogap, WhGap ).
s( decl, gap($(Agr,nom)) ) -->
        predicate( Agr, nogap, nogap ).

s( subord, WhGap ) -->
        (   [that] ; [] ),
        s( decl, WhGap ).

s( ynQ, nogap ) -->
        (
            infl( Agr, VForm )
        ;
            do( $(fin,Agr) ),
            {VForm = inf}
        ),
        np( $(Agr,nom), -wh ),
        predicate( _, gap(VForm), nogap ).

s( whQ, nogap ) -->
%%% This structure is very flat!
        np( $(Agr0,Case0), +wh ),
        (
            infl( Agr, VForm )
        ;
            do( $(fin,Agr) ),
            {VForm = inf}
        ),
        np( $(Agr,nom), -wh ),
        predicate( _, gap(VForm), gap($(Agr0,Case0)) ).
s( whQ, nogap ) -->
        np( $(Agr,nom), +wh ),
        predicate( Agr, nogap, nogap ).
```

Figure 2: The S rules.

```
predicate( Agr, nogap, WhGap ) -->
        (
            infl( Agr, VForm ),
            peri( VForm, WhGap )
        ;
            vp( $(fin,Agr), WhGap )
        ).
predicate( Agr, nogap, WhGap ) -->
        (
            infl( Agr, VForm ),
            [not],
            peri( VForm, WhGap )
        ;
            do( $(fin,Agr) ),
            [not],
            vp( inf, WhGap )
        ).
predicate( _, gap(VForm), WhGap ) -->
        (   [not]
        ;   []
        ),
        peri( VForm, WhGap ).



peri( VForm, WhGap ) -->
        (
            aux( VForm, Selects ),
            peri( Selects, WhGap )
        ;
            vp( VForm, WhGap )
        ).
```

Figure 3: Rules for Predicate and Periphrasis.

```
vp( VForm, WhGap ) -->
        (
            {VForm = pass} ->
            %% No (Wh) gaps in passives:
            {WhGap = nogap},
            %% Passive only applies to transitive V:
            verb( pass, np )
        ;
            verb( VForm, SubCat ),
            complement( SubCat, WhGap )
        ).



complement( np, gap(_,acc) ) --> [].
complement( np, nogap ) --> np( $(_,acc), -wh ).

complement( sBar, WhGap ) --> s( subord, WhGap ).
```

Figure 4: Rules for VP.

- np(NPInflection, WhFeature)

4. Feature Structures. I mainly tried to separate inflectional features, features particular to a given word form, from "lemma" features, features shared by a whole family of word forms (example: all of the different forms of the verb "think" subcategorize for sBar). Within the inflectional features, I separated Person and Number from Case (for NPs), since Person and Number travel together in the grammar to places where Case doesn't go (e.g., to verbs), and vice versa (e.g., object gap-fillers must be compatible with accusative case, but nobody cares about their agreement features). Feature (sub-)structures are all just represented as tuples under the functor '$'. The top level of the structure is generally unpacked into distinct argument positions of an actual Prolog predicate, however.

5. Type Hierarchies, PSG-style. We don't think of infl/2 as a grammatical category but as a sort of meta-category, a category of categories, consisting of modals and auxiliaries. Put another way, it is the supertype of both modal and aux. Aux itself can be seen as the supertype of both "have" and "be". We should probably add "do" to the list, and perhaps "get" as well (for use in "get"-passives like "The villain will get killed in the last scene.")

```
%%%---( Noun Phrases )--------------------------

np( $($(3,Num),_Case), Wh ) -->
        det( Num, Wh ),
        noun( Num ).
np( $($(P,N),Case), -wh ) --> pers_pron( P, N, Case ).
np( $($(P,N),Case), +wh ) --> wh_pron( P, N, Case ).

det( A, -wh ) --> [the], {number_feature(A)}.
det( A, +wh ) --> [which], {number_feature(A)}.
det( sg, -wh ) --> [every].

noun( sg ) --> [hero].
noun( sg ) --> [villain].
noun( pl ) --> [heros].
noun( pl ) --> [villains].



%%%---( Pronouns )------------------------------

pers_pron( 1, sg, nom ) --> [i].
pers_pron( 1, sg, acc ) --> [me].
pers_pron( 1, pl, nom ) --> [we].
pers_pron( 1, pl, acc ) --> [us].
pers_pron( 3, sg, nom ) --> ( [he] ; [she] ; [it] ).
pers_pron( 3, sg, acc ) --> ( [him] ; [her] ; [it] ).
pers_pron( 3, pl, nom ) --> [they].
pers_pron( 3, pl, acc ) --> [them].
pers_pron( 2, _Num, _Case ) -->
        [you].


wh_pron( 3, _, _ ) -->
        [who].
wh_pron( 3, _, acc ) -->
        [whom].
```

Figure 5: Rules for NP.

```
%%% The following isn't very practical for such a
%%% small database, but it illustrates a technique for
%%% taking more advantage of Prolog's built in
%%% indexing.
%%% Exercise: Redo this so that Case is the primary
%%% key. Think about why this might be a better
%%% choice than using Person.
/*
pers_pron( 1, Num, Case ) -->
        p_pn_1( Num, Case ).
pers_pron( 3, Num, Case ) -->
        p_pn_3( Num, Case ).
pers_pron( 2, _, _ ) --> [you].

p_pn_1( sg, Case ) -->
        p_pn_1_sg( Case ).
p_pn_1( pl, Case ) -->
        p_pn_1_pl( Case ).
p_pn_3( sg, Case ) -->
        p_pn_3_sg( Case ).
p_pn_3( pl, Case ) -->
        p_pn_3_pl( Case ).

p_pn_1_sg( nom ) --> [i].
p_pn_1_sg( acc ) --> [me].

p_pn_1_pl( nom ) --> [we].
p_pn_1_pl( acc ) --> [us].

p_pn_3_sg( nom ) --> ( [he] ; [she] ; [it] ).
p_pn_3_sg( acc ) --> ( [him] ; [her] ; [it] ).

p_pn_3_pl( nom ) --> [they].
p_pn_3_pl( acc ) --> [them].
*/
```

Figure 6: Example: Indexing a lexical database.

```
infl( _, inf ) -->
        modal.
infl( Agr, Selects ) -->
        aux( $(fin,Agr), Selects ).


aux( VFeats, Selects ) -->
        (
            have( VFeats, Selects )
        ;
            be( VFeats, Selects )
        ).


%%%---( Modals )-------------------------------

modal --> [will].
modal --> [must].



%%%---( Do )-----------------------------------

do( $(fin,$(P,N)) ) -->
        [W],
        {word_form( W, v(fin,_,P,N), do )}.


%%%---( Auxiliary ``have'' )--------------------

have( VFeats, perf ) --> have_form( VFeats ).

have_form( $(fin,$(3,sg)) ) --> ( [has] ; [had] ).
have_form( inf ) --> [have].



%%%---( Auxiliary ``be'' )----------------------

be( VFeats, prog ) --> {\+ VFeats = prog}, be_form( VFeats ).
be( VFeats, pass ) --> be_form( VFeats ).

be_form( $(fin,$(P,N)) ) -->
        [W],
        {word_form( W, be(fin,_T,P,N), be )}.
be_form( perf ) --> [been].
be_form( inf ) --> [be].
be_form( prog ) --> [being].
```


Figure 7: Auxiliaries and their cousins.

```
%%%---( Main Verbs )---------------------------
%%%
%%% Verbal paradigms are given in unit clauses
%%% of the form
%%%
%%%    verb(Inf,Pres,Past,PresPart,PastPart).
%%%
%%% Note that these are *morphological*
%%% categories. So, for example, both perfective
%%% aspect and passive voice make use of the same
%%% morphological form of the verb.

verb( $(fin,$(Prs,Num)), SubCat ) -->
        [W],
        {word_form( W, v(fin,_Tns,Num,Prs), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( pass, SubCat ) -->
        [W],
        {word_form( W, v(ppa), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( prog, SubCat ) -->
        [W],
        {word_form( W, v(ppr), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( perf, SubCat ) -->
        [W],
        {word_form( W, v(ppa), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( pres, SubCat ) -->
        [W],
        {word_form( W, v(_,pres,_,_), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( past, SubCat ) -->
        [W],
        {word_form( W, v(_,past,_,_), HeadWord ),
         subcat( HeadWord, SubCat )}.
verb( inf, SubCat ) -->
        [W],
        {word_form( W, v(inf), HeadWord ),
         subcat( HeadWord, SubCat )}.
```

Figure 8: Main verb interface to the lexicon.

```
%subcat( believe, sBar ).
subcat( capture, np ).
%subcat( claim, sBar ).
subcat( entertain, np ).
subcat( pursue, np ).
subcat( think, sBar ).
subcat( watch, np ).


aux( have ).
aux( be ).
aux( do ).
```

Figure 9: Lemma features, indexed by headword.

```
%%% At the moment these are all verbal forms.

word_form( am, be(fin,pres,sg,1), be ).
word_form( are, Feats, be ):-
        (   Feats = be(fin,pres,sg,2)
        ;   Feats = be(fin,pres,pl,A),
            person_feature( A )
        ).
word_form( be, be(inf), be ).
word_form( been, be(ppa), be ).
word_form( being, be(ppr), be ).
word_form( capture, Feats, capture ):-
        regular_verb_baseform_features( Feats ).
word_form( captured, Feats, capture ):-
        regular_verb_pastform_features( Feats ).
word_form( captures, v(fin,pres,sg,3), capture ).
word_form( capturing, v(ppr), capture ).
word_form( did, v(fin,past,A,B), do ):-
        number_feature( A ),
        person_feature( B ).
word_form( do, Feats, do ):-
        regular_verb_baseform_features( Feats ).
word_form( does, v(fin,pres,sg,3), do ).
word_form( done, v(ppa), do ).
word_form( entertain, Feats, entertain ):-
        regular_verb_baseform_features( Feats ).
word_form( entertained, Feats, entertain ):-
        regular_verb_pastform_features( Feats ).
word_form( entertaining, v(ppr), entertain ).
```

Figure 10: The word-form list (1).

```
word_form( entertains, v(fin,pres,sg,3), entertain ).
word_form( had, Feats, have ):-
        regular_verb_pastform_features( Feats ).
word_form( has, v(fin,pres,sg,3), have ).
word_form( have, Feats, have ):-
        regular_verb_baseform_features( Feats ).
word_form( is, be(fin,pres,sg,3), be ).
word_form( must ).
word_form( pursue, Feats, pursue ):-
        regular_verb_baseform_features( Feats ).
word_form( pursues, v(fin,pres,sg,3), pursue ).
word_form( pursued, Feats, pursue ):-
        regular_verb_pastform_features( Feats ).
word_form( pursuing, v(ppr), pursue ).
word_form( think, Feats, think ):-
        regular_verb_baseform_features( Feats ).
word_form( thinking, v(ppr), think ).
word_form( thinks, v(fin,pres,sg,3), think ).
word_form( thought, Feats, think ):-
        regular_verb_pastform_features( Feats ).
word_form( was, be(fin,past,sg,A), be ):-
        non_second_person( A ).
word_form( watch, Feats, watch ):-
        regular_verb_baseform_features( Feats ).
word_form( watched, Feats, watch ):-
        regular_verb_pastform_features( Feats ).
word_form( watches, v(fin,pres,sg,3), watch ).
word_form( watching, v(ppr), watch ).
word_form( were, be(fin,past,A,B), be ):-
        (   A = sg, B = 2
        ;   A = pl, person_feature( B )
        ).
word_form( will ).
```

Figure 11: The word-form list (2).

```
regular_verb_baseform_features( v(inf) ).
regular_verb_baseform_features( v(fin,pres,sg,A) ):-
        non_third_person( A ).
regular_verb_baseform_features( v(fin,pres,pl,B) ):-
        person_feature( B ).

regular_verb_pastform_features( v(ppa) ).
regular_verb_pastform_features( v(fin,past,A,B) ):-
        number_feature( A ),
        person_feature( B ).


number_feature( sg ).
number_feature( pl ).

person_feature( 1 ).
person_feature( 2 ).
person_feature( 3 ).

non_third_person( 1 ).
non_third_person( 2 ).

non_second_person( 1 ).
non_second_person( 3 ).


case_feature( nom ).
case_feature( acc ).
```

Figure 12: Auxiliary predicates of the wordform list.