Permutational Grammar for free word order languages

Mats Eeg-Olofsson and Bengt Sigurd

1 Abstract and introduction

Permutational Grammar, PG, is a grammar inspired by the Free Word Order grammar, FOG, presented in Vladimir Pericliev & Alexander Grigorov 1992. Some languages, notably Latin, are said to have free word order, see e.g. Siewierska 1988. The name Permutational Grammar is derived from the use of permutations in order to generate order variation. The general problem to be solved by FOG and PG is the generation and analysis of a great number of word order variants with (roughly) the same meaning. PG accomplishes this by specifying some basic phrase structure orders with their functional (and semantic) representations, and then permuting the corresponding sequences of constituents to obtain all the other sequences with the same meaning.

Permutational Grammar can be regarded as a generative phrase structure grammar with transformations represented by permutations. It is developed from SWETRA grammar (see Sigurd 1994). The constituent parsing trees are not represented explicitly. PG is written with generative rewrite rules and implemented in Prolog via the Definite Clause Grammar (DCG) formalism. The Prolog implementation used here is LPAProlog. The rules state that permutations of the constituents to the right of the rewrite symbol have the functional representation given as an argument to the left of the rewrite symbol. These rewrite rules can be compiled into rules that generate all possible permutations of the basic word order 'on the fly'.

It is possible to apply constraints to the permutations generated. One may, for example, introduce an order constraint like \texttt{imbefore(C1,C2,M)}, which states that a constituent matching the description C1 must occur immediately before another constituent matching C2 in the list of constituents M. Another example is \texttt{last(C,M)}, which states that a C must occur last in the list M.
Such constraints can easily be expressed in Prolog. The order constraints may be considered as implementations of the linear precedence (LP) rules of Generalized Phrase Structure Grammar, see Gazdar et al. 1985. One may also associate to the Constraint Grammar presented in Karlsson 1990.

In this paper we will only demonstrate the potential of PG for Latin and Swedish. A detailed permutational grammar of Basque is presented in Holmer & Sigurd in this volume.

2 Word order in Latin
The Latin sentences used traditionally to demonstrate that word order is free are typically (cf. Pericliev & Grigorov 1992), reorderings of the following words: Puella bona amat puerum parvum. The word order of the equivalent English sentence, (The) good girl loves (the) poor boy, can hardly be changed without changing at the same time the grammaticality or the meaning of the sentence. In Latin this sentence is supposed to be changeable into e.g. Parvum puella bona amat puerum and Amat bona puella parvum puerum. In the grammar written by Tidner 1944 it is stated (p. 256, in translation) that word order in Latin is generally more free than in Swedish. However, certain orders are especially frequent according to Tidner.

1. The predicate is generally placed last in the sentence (Hannibal Alpes transgressus est ‘Hannibal has passed the Alps’). This is confirmed by other sources, where it is also said that a focused word, often the subject, generally occurs first. This gives Latin a basic SOV word order.

2. An adjective is generally placed after its head (lus civile ‘Civil law’).

3. A focused word may be placed initially and separated from its head as shown by the following sentence where the adjective magna is separated from its head praemia placed finally: Magna proponit iss, qui ilium occiderint praemia ‘Great rewards he proposed for those who killed this (person)’.

4. The preposition is often inserted between a determiner and its head in a prepositional phrase: Hunc in modum ‘in this way’.

Word order in Latin was probably not as free as some have suggested on the basis of occasional orders in poetry, but we will use an extremely free Latin here for the sake of demonstration.

3 A toy grammar of Latin
The following pedagogical grammar constructed with the labels in Pericliev & Grigorov uses DCG rules with a standard Prolog implementation. Given the lexicon below, it can only generate the sentence Puella (bona) puerum (parvum) amat with (or without) adjectives, which follows Tidner’s recommendation in having the verb last and the adjectives after their heads.

The labels should be easy to identify. The arrow rule states that there is a Latin sentence pattern (ls0) where a nominative adjective (A1) occurs after a nominative noun (N1), preceding a noun in the accusative (N2) followed by an adjective in the accusative (A2). A verb (V) occurs last. To the left of the rewrite symbol the corresponding functional roles of the phrases are stated within square brackets.

Note that the terms that make up the functional representation within the brackets are subj, pred, and obj in that standardized order. The value (meaning) of the attribute adjective is inserted before its head in the bracket parenthesis. The order of the attribute and the head is arbitrary in the functional representation but standardized in SWETRA grammar. When there is not any adjective, the lexical rules will assign the value [] (the empty list) to the corresponding adjective variable.

ls0([subj([A1,N1]),pred(V),obj([A2,N2])]) —> noun_nom[N1],adj_nom[A1],noun_acc[N2],adj_acc[A2],verb(V).

The following rules constitute a suitable lexicon, expressed as standard DCG lexical rewrite rules in Prolog. The word-semantic representations are Machinese English in order to facilitate translation between Latin and Swedish (to be presented).

verb(m(love,pres)) —> [amat].
adj_nom(m(good,_)) —> [bona].
adj_acc(m(poor,_)) —> [parvum].
adj_nom([]) —> [].
adj_acc([]) —> [].
noun_nom(m{girl, sg)) —> [puella].
noun_acc(m{boy, sg)) —> [puerum].

When called by the command ls0(F,X,[]), Grammar ls0 can only generate the following four sentences. The functional representation for each sentence is given before the sentence.

[subj([m{good, _60888}), m{girl, sg)}], pred(m{love, pres}), obj([m{poor, _60816}), m{boy, sg})]
[puella, bona, puerum, parvum, amat]
[subj([m{good, _60888}), m{girl, sg)}], pred(m{love, pres}), obj([[]], m{boy, sg})]
[puella, bona, puerum, amat,]
[subj([[]], m{girl, sg)}], pred(m{love, pres}), obj([m{poor, _60234}), m{boy, sg})]
[puella, puerum, parvum, amat ]
4 Permutational grammars for Latin

The first step in constructing a permutational grammar which can generate
more word orders is to place the constituent phrases in a list which can be
permuted by the predicate permute. This is done in the following rule:

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \rightarrow \{M=\text{adj\_nom}(A1), \text{noun\_nom}(N1), \text{verb}(V), \text{adj\_acc}(A2), \text{noun\_acc}(N2), \text{permute}(M,M2), \text{imbefore}(\text{noun\_nom}(N1), \text{adj\_nom}(A1), M2), \text{imbefore}(\text{noun\_acc}(N2), \text{adj\_acc}(A2), M2), \text{surf}(M2) \} \]

What is written after the rewrite arrow \( \rightarrow \) within braces (\{, \}) states the
condition that the variable \( M \) is the list of constituents. This list is then
permuted by the standard predicate permute. The predicate surf analyses the
permuted list \( M2 \) and uses it to find surface realizations of the phrases within the
list.

One may achieve the same result by introducing a new rewrite operator
expressed by the keyword \text{dominates}. This operator has a function similar
to the standard \( \rightarrow \) rewrite arrow, but it also expresses that the order
between the daughter constituents to the right of it is arbitrary. Such
generalized rules, using extended Prolog syntax as host formalism, can then be
compiled into Prolog. The rule below uses the operator \text{dominates}:

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \text{ dominates} \text{noun\_nom}(N1), \text{adj\_nom}(A1), \text{noun\_acc}(N2), \text{adj\_acc}(A2), \text{verb}(V) \]

This rule can then be compiled into a corresponding DCG rule with
standard Prolog implementation and conditions added within curly brackets.

The Prolog code for the compiler, including the surf predicate, is specified
in the Appendix.

5 Constraints

Many languages have restrictions on the order between the subject, the object
and the finite verb, or between head nouns and their modifiers. The following
extended grammar rule states that the Latin \text{noun\_nom} must occur
immediately before \text{adj\_nom}. This restriction has the effect that a reduced
number of permutations is generated. Note that we have used a different order
in the basic list of daughter constituents. This has no effect on the sentences
permitted.

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \rightarrow \{M=\text{adj\_nom}(A1), \text{noun\_nom}(N1), \text{verb}(V), \text{adj\_acc}(A2), \text{noun\_acc}(N2), \text{permute}(M,M2), \text{imbefore}(\text{noun\_nom}(N1), \text{adj\_nom}(A1), M2), \text{imbefore}(\text{noun\_acc}(N2), \text{adj\_acc}(A2), M2), \text{surf}(M2) \} \]

The sequence \text{parvum puerum puella amat bona} is rejected by this
grammar.

In the present framework, this can be accomplished by the introduction of
yet another operator represented by the keyword \text{provided}. This operator is
optional, but it must be followed by a list of restrictions. The following
notational variant of rule \text{lsl} uses both the operators \text{dominates} and
\text{provided}.

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \text{ dominates} \text{noun\_nom}(N1), \text{adj\_nom}(A1), \text{noun\_acc}(N2), \text{adj\_acc}(A2), \text{verb}(V) \]

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \text{ provided} \text{noun\_nom}(N1), \text{adj\_nom}(A1) \]

The following version of syntactic rule \( lsl \) is different as it requires both
the nominative and the accusative noun to occur before their respective
attributes. This is brought about by the added constraint

\[ \text{imbefore}(\text{noun\_acc}(N2), \text{adj\_acc}(A2)) \]

The above grammar can only generate 24 sentences.

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \text{ dominates} \text{noun\_nom}(N1), \text{adj\_nom}(A1), \text{noun\_acc}(N2), \text{adj\_acc}(A2), \text{verb}(V) \]

\[ lsl([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2])]) \text{ provided} \text{noun\_nom}(N1), \text{noun\_acc}(N2), \text{adj\_nom}(A1), \text{adj\_acc}(A2) \]

The following permutational Latin grammar \( lsl3 \) extended with
adverbials can generate 5760 permutations using the additional word and
phrases mentioned below. No extra order is introduced in this extreme
grammar, which is written directly in DCG.

\[ lsl3([\text{subj}([A1,N1]), \text{pred}(V), \text{obj}([A2,N2]), \text{advl}(Av)]) \rightarrow \{M=\text{adj\_nom}(A1), \text{noun\_nom}(N1), \text{verb}(V), \text{adj\_acc}(A2), \text{noun\_acc}(N2), \text{adv\_l}(Av), \text{permute}(M,M2), \text{surf}(M2) \} \]

\[ \text{adv}(m(\text{willingly, _})) \rightarrow \text{libenter}. \] % Adverb
\[ \text{adv}([P, [A, N]]) \rightarrow p(P), \text{noun\_acc}(N), \text{adj\_acc}(A). \] % Prep phrase
\[ p(m(\text{with, _})) \rightarrow \text{con}. \] % in order to match Swedish examples

The following are some (somewhat strange) examples generated by the call
\[ lsl3(F, X, []). \]

\[ F = \{\text{subj}([[]], \text{m(girl, sg)})), \text{pred}(\text{m(love, pres)}), \text{obj}([[]], \text{m(boy, sg)}))\], \text{advl}([\text{m(with, _34764)}, [[], \text{m(boy, sg)]})\],}
6 A Swedish permutational grammar

For comparison, the following Swedish grammar has been developed. It has been constructed with a view to translation between Latin and Swedish, and allows 4032 permutations.

Swedish belongs to the V2 languages, and this characteristic has been implemented as a condition that the second element of the permuted syntactic sequence should be the finite verb. The code \( M_2 = [\_ , \text{svfin}\{\}_1] \) states that the second element of the list \( M_2 \) must be the finite verb. We will not explain the Prolog details any more here. Three different basic syntactic patterns are implemented, the second pattern includes an adverbial.

A characteristic of Swedish is the so called stranded preposition, i.e. the prepositional head of a prepositional phrase which is left alone when its object noun phrase is moved to the front position. An example is: *Flickan leker pojkken med* ['The boy plays with the girl']. The third syntactic pattern below shows how such sentences are handled, namely by requiring that the missing (fronted) noun phrase is the same as the first (focused) noun phrase.

Using the functional representations as an interlingua, the grammars presented allow automatic translation between Latin and Swedish.

Swedish into Latin:

\[
\text{ss}(F, \{\text{good, flicka}, \text{ålskar}, \text{gärna}, \text{fattig}, \text{pojke}\}, \{}\), \text{ls3}(F, X, \{}\)
\]

No.1 : \( F = \{\text{subj} \{\text{m}\{\text{good, 4962}\}, \text{m}\{\text{girl, sg}\}\}), \text{pred}\{\text{m}\{\text{love, pres}\}), \text{obj}\{\text{m}\{\text{poor, 4707}\}, \text{m}\{\text{boy, sg}\}\}), \text{advl}\{\text{m}\{\text{willingly, 4821}\}\}, X = \{\text{bona, puella, amat, parvum, puerum, libenter}\}
\]

Latin into Swedish:

\[
\text{ls3}(F, \{\text{puella, libenter, bona, parvum, puerum, amat}\}, \{}\), \text{ss}(F, Y, \{}\)
\]

\( F = \{\text{subj} \{\text{m}\{\text{good, 85344}\}, \text{m}\{\text{girl, sg}\}\}), \text{pred}\{\text{m}\{\text{love, pres}\}), \text{obj}\{\text{m}\{\text{poor, 85284}\}, \text{m}\{\text{boy, sg}\}\}), \text{advl}\{\text{m}\{\text{willingly, 85044}\}\}, Y = \{\text{god, flicka, ålskar, gärna, fattig, pojke}\}
\]

\( Y = \{\text{gärna, ålskar, god, flicka, fattig, pojke}\}
\]

\( Y = \{\text{gärna, ålskar, fattig, pojke, god, flicka}\}
\]

The last sentence is incorrect if *god flicka* is to be the subject. This can be remedied by requiring that the subject is either the np before the finite verb (SV order) or the first np after the verb (VS order).

Translation of Swedish sentence with stranded preposition into Latin:

\[
\text{ss}(F, \{\text{pojke, alskar, god, flicka, fattig, pojke, med}\}, \{}\), \text{ls3}(F, X, \{}\)
\]

No.1 : \( F = \{\text{subj} \{\text{m}\{\text{good, 31446}\}, \text{m}\{\text{girl, sg}\}\}), \text{pred}\{\text{m}\{\text{love, pres}\}), \text{obj}\{\text{m}\{\text{poor, 31350}\}, \text{m}\{\text{boy, sg}\}\}), \text{advl}\{\text{m}\{\text{with, 31215}\}, \{1\}, \text{m}\{\text{boy, sg}\}\}), X = \{\text{bona, puella, amat, parvum, puerum, con, puerum}\} \% 720 solutions
\]

7 Conclusion

The order between the phrases representing Subject, Predicate and Object (SVO) and between a modifier (A) and its head (N) are typological characteristics of languages. Many languages require additional conditions, e.g. on the place of the focused constituent. It has to be first in the sentence initially before the finite verb in Swedish, immediately before the finite verb in Basque. Such conditions can easily be handled by Permutational Grammar. Permutational Grammar may help in discovering grammatical constraints and typological universals.

\[
\begin{align*}
\text{sadv}(N, \{P, N1\}) & \rightarrow \text{sp}(P), \text{snp}(N1). \% \text{prep phrase} \\
\text{sadv}(N, \{P, \}_1) & \rightarrow \text{sp}(P). \% \text{defective pp} \\
\text{sp}(m(\text{with, } _1) & \rightarrow \{\text{med}\}. \\
\text{svfin}(m(\text{love, pres})) & \rightarrow \{\text{alskar}\}.
\end{align*}
\]
References

Appendix: Prolog compilation of PG rules into Prolog via DCG rules

% Operator declarations
:- op(1200, xfx, dominates).
:- op(1100, xfx, provided).

% Compile PG rules
translate_pg((LHS dominates RHS), Prolog_rule) :- !, % True PG rule
translate_rhs(RHS, RHS_trans),
DCG_rule = (LHS --&gt; RHS_trans),
translate_dcg(DCG_rule, Prolog_rule).
translate_pg(Rule, Trans) :- translate_dcg(Rule, Trans). % Ordinary DCG rule

% Translate right hand side of true PG rule
translate_rhs((Daughters provided Restriction), % Constraints included
( [ M = Daut_list, permute(M,M2), Exprestrict ], surf(M2) )) :- !,
commalist(Daughters, Daut_list),
translate_restriction(Restriction, Exprestrict, M2),
translate_rhs(Daughters_only, ([ M = Daut_list, permute(M,M2) ], surf(M2)) )) :-
commalist(Daughters_only, Daut_list). % Add list argument to constraint predicates
translate_restriction((R,R2), (Expr1,Expr2), M2) :- !,
R1 :-.. R11, append(R11, [M2], R1expl),
Expr1 :-.. R1expl,
translate_restriction(R2, Expr2, M2),
translate_restriction(R, Expr, M2) :-
R :-.. R1, append(R1, [M2], Rexpl),
Expr :-.. Rexpl.

% Parsing a constituent list using difference lists
surf([], L, L).
surf([H|T], Li, Lo) :-
Goal1 :-.. H, append(Goal1, [Li, L], H1),
Goal2 :-.. H1, call(Goal2),
surf(T, Li, Lo).
/* The predicate translate_dcg/2 translates DCG rules into standard Prolog code. In many Prolog implementations this predicate can be defined simply as a built-in predicate called expand_dcg/2 or the like */
% translate_dcg(Rule, Trans) :- expand_dcg(Rule, Trans).
commalist((E1,E2,Rest), [E1 | Rest1]) :- !,
commalist((E2,Rest), Rest1).
commalist((E1,E2), [E1,E2]) :- !.
commalist((E1), []). % Sample constraint predicate
% An element matching X occurs immediately before an element Y in the list L
imbefore(X,Y,[X,Y | _]) :- imbefore(X,Y,T).
imbefore(X,Y,[X,Y | _]) :- imbefore(X,Y,T).
% Standard list permutation predicate
permute([],[]).
permute([F|L],P) :- permute(R,M), insert(F,M,P).
% Standard list insertion predicate
insert(E,L,[E|L]).
insert(E,[F|L], [F|L1]) :- insert(E,L, L1).