Burenhult, Niclas. 1999. 'A bibliographical guide to Aslian linguistics'. *Mon-Khmer Studies* 29, 133-41.
Burenhult, Niclas. 2000. 'Jahai phonology: a preliminary survey'. To appear in *Mon-Khmer Studies* 31.
Diffloth, Gérard. 1976a. 'Minor-syllable vocalism in Senoic languages'. In P. N. Jenner et al. (eds), *Austroasiatic studies* I, 229-47. Honolulu: The University Press of Hawaii.
Diffloth, Gérard. 1976b. 'Jah-Hut: an Austroasiatic language of Malaysia'. In Nguyen Dang Liem (ed.), *South-East Asian linguistic studies* 2, 73-118. Canberra: Pacific Linguistics.
Kruspe, Nicole. 1999. *Semelai*. Ph. D. dissertation, University of Melbourne.
Matisoff, James A. 1973. 'Tonogenesis in Southeast Asia'. In Larry M. Hyman (ed.), *Consonant types and tone* (*Southern California Occasional Papers in Linguistics* 1), 71-95. Los Angeles: University of Southern California.
Matisoff, James A. to appear. 'Aslian: Mon-Khmer of the Malay Peninsula'. From Chapter III of *Languages of Mainland Southeast Asia*. Cambridge: Cambridge University Press.

Lund University, Dept. of Linguistics
Working Papers 48 (2001), 15–23

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 `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 `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 *(Ius 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 illum 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 ]
```

```
[subj([[], m(girl, sg)]), pred(m(love, pres)), obj([[], m(boy,
  sg)])]
[puella, puerum, 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:

```
ls1([subj([A1,N1]),pred(V),obj([A2,N2])]) -->
{M=[noun_nom(N1),adj_nom(A1),noun_acc(N2),adj_acc(A2),verb(V)]
  , permute(M,M2)}, surf(M2).
```

What is written after the rewrite arrow `-->` 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 relizations of the phrases within the list.

One may achieve the same result by introducing a new rewrite operator expressed by the keyword **`dominates`**. This operator has a function similar to the standard `-->` 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 **`dominates`**:

```
ls1([subj([A1,N1]),pred(V),obj([A2,N2])]) dominates
  noun_nom(N1), adj_nom(A1), noun_acc(N2), adj_acc(A2),
  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 `noun_nom` must occur immediately before `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.


```
ls2([subj([A1,N1]),pred(V),obj([A2,N2])]) -->
{M=[adj_nom(A1),noun_nom(N1),verb(V),adj_acc(A2),
  noun_acc(N2)], permute(M,M2),
imbefore(noun_nom(N1),adj_nom(A1),M2),
  imbefore(noun_acc(N2),adj_acc(A2),M2)},surf(M2).
```

The sequence *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 **`provided`**. This operator is optional, but it must be followed by a list of restrictions. The following notational variant of rule ls2 uses both the operators **`dominates`** and **`provided`**.

```
ls2([subj([A1,N1]),pred(V),obj([A2,N2])]) dominates
  adj_nom(A1),noun_nom(N1),verb(V),adj_acc(A2),noun_acc(N2)
provided imbefore(noun_nom(N1),adj_nom(A1)).
```

The following version of syntactic rule (`ls2`) 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

```
imbefore(noun_acc(N2),adj_acc(A2))
```

The above grammar can only generate 24 sentences.

```
ls2([subj([A1,N1]),pred(V),obj([A2,N2])]) dominates
adj_nom(A1),noun_nom(N1),verb(V),adj_acc(A2),noun_acc(N2)
provided imbefore(noun_nom(N1),adj_nom(A1),
  imbefore(noun_acc(N2),adj_acc(A2)).
```

The following permutational Latin grammar (`ls3`) 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.

```
ls3([subj([A1,N1]),pred(V),obj([A2,N2]),advl(Av)]) -->
{M=[adj_nom(A1),noun_nom(N1),verb(V),adj_acc(A2),noun_acc(N2),
adv(Av)], permute(M,M2)}, surf(M2).

adv(m(willingly,_)) --> [libenter]. % Adverb
adv([P,[A,N]]) --> p(P),noun_acc(N),adj_acc(A). % Prep phrase
p(m(with,_)) --> [con]. % in order to match Swedish examples
```

The following are some (somewhat strange) examples generated by the call `ls3(F,X,[])`.

```
F = [subj([[], m(girl, sg)]), pred(m(love, pres)), obj([[],
  m(boy, sg)]), advl([m(with, _34764), [],m(boy, sg)])],
```

```
X = [con, puerum, puerum, amat, puella]
X = [bona, puella, amat, parvum, puerum, libenter]
X = [bona, puella, amat, parvum, puerum, con, puerum]
```

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 `M2=[_, svfin(_) |_]` states that the second element of the list `M2` 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 pojken med* [the girl plays the boy with] '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.

```
ss([subj(N1),pred(V),obj(N2)]) -->
   {M=[snp(N1),svfin(V),snp(N2)],
permute(M,M2), M2=[_,svfin(_)|_]}, surf(M2).
% verb second no adverbial

ss([subj(N1),pred(V),obj(N2),advl(A)]) -->
% verb second with adverbial
{M=[snp(N1),svfin(V),snp(N2),sadv(N3,A)], permute(M,M2),
   M2=[_,svfin(_)|_]}, surf(M2), {A≠[P,[]]}.
% no defect pp

ss([subj(N1),pred(V),obj(N2),advl(Av)]) -->
% defective pp, prep stranded
snp(N3),svfin(V),snp(N1),snp(N2),sadv(N3,A),{A=[P,[]],Av=[P,N3
   ]}.
% no permutations

% noun phrase rule
snp([A,N]) --> sadj(A),snoun(N).
snp([[],N]) --> snoun(N).

% lexicon
sadj(m(good,_)) --> [god].
snoun(m(girl,sg)) --> [flicka].
sadj(m(poor,_)) --> [fattig].
snoun(m(boy,sg)) --> [pojke].
sadv(_,m(willingly,_)) --> [gärna].
```



```
sadv(N,[P,N1]) --> sp(P),snp(N1). % prep phrase
sadv(N,[P,[]]) --> sp(P). % defective pp
sp(m(with,_)) --> [med].
svfin(m(love,pres)) --> [älskar].
```

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

Swedish into Latin:

```
ss(F,[god, flicka, älskar, gärna,fattig, pojke],
   []),ls3(F,X,[])
No.1 : F = [subj([m(good, _4962), m(girl, sg)]), pred(m(love,
   pres)), obj([m(poor, _4707), m(boy, sg)]),
   advl(m(willingly, _4821))], X = [bona, puella, amat,
   parvum, puerum, libenter]
```

Latin into Swedish:

```
ls3(F, [puella,libenter,bona,parvum,puerum,amat], []), ss(F,
   Y, [])
F = [subj([m(good, _85344), m(girl, sg)]), pred(m(love,
   pres)), obj([m(poor, _85284), m(boy, sg)]),
   advl(m(willingly, _85404))],
Y = [god, flicka, älskar, gärna, fattig, pojke]
Y = [gärna, älskar, god, flicka, fattig, pojke]
Y = [gärna, älskar, 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:

```
ss(F, [pojke, älskar, god, flicka, fattig, pojke, med], []),
   ls3(F, X, [])
No.1 : F = [subj([m(good, _31446), m(girl, sg)]), pred(m(love,
   pres)), obj([m(poor, _31350), m(boy, sg)]), advl([m(with,
   _31215), [[],m(boy, sg)]])],
X = [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.

References

Gazdar G., E. Klein, G. K. Pullum & I. A. Sag. 1985. *Generalized phrase structure grammar.* Oxford: Basil Blackwell.

Karlsson, F. 1990. 'Constraint Grammar as a framework for parsing unrestricted text'. In H. Karlgren (ed.), *Proceedings of the 13th International Conference of Computational Linguistics* 3, 168-73. Helsinki.

Pericliev, V. and A Grigorov. 1992. 'Extending Definite Clause Grammar to handle flexible word order'. In B. du Boulay & V. Sgurev (eds), *Artificial Intelligence V: Methodology, Systems, Applications,* 161-70. Amsterdam: North-Holland.

Siewierska, A. 1988. *Word order rules.* London: Croom Helm.

Sigurd, B. 1994. *Computerized grammars for analysis and translation.* Lund: Lund University Press.

Tidner, E. 1944. *Latinsk språklära.* Uppsala: Almqvist & Wiksell.

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 --> 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((R1,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, L1], H1),
   Goal2 =.. H1, call(Goal2),
   surf(T, L1, 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,[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,T).

% Standard list permutation predicate
permute([],[]).
permute([F|R],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).
```