Lauri Carlson
University of Helsinki
Research Unit for Computational Linguistics

LP RULES IN UNIFICATION GRAMMAR

The purpose of this paper is to consider extension of unification based context free grammar with (a suitable generalization of) the IDLP formalism of Gazdar et al. (1985).

We define unification based context free grammar (UCFG) as a generalization of CF grammar. The shared property is the CF forr of productions (a single nonterminal on the LHS of a rule). This allows use of appropriately modified CFG parsinc algorithms. The HUG grammar formalism of Lauri Karttunen (papeı presented in this conference) can be regarded as an instance of UCFG.

The structure of the present paper is as follows. In section 1, definitions are given to relevant types of grammar. Section 2 discusses generalization of IDLP grammar to unification. A parsing problem bound up with the generalization is described ir section 2.2, which closes with a definition of unification IDLF grammar. Direct parsing of UIDLP grammar is discussed in sectior 2.3. The last section 2.4 proposes a generalization of the notion of a unification LP rule, aimed to avoid the cost of parsing full UIDLP grammar while retaining enough expressive power for the statement of common types of word ordeı constraints.

1. Definitions

Standard notations of formal grammar theory are used (see e.g Hopcroft and Ullman (1979) for definitions). **in, +, and iff** are used for set theoretic membership, union and definitional equivalence, respectively.

1.1. Definition of standard CF grammar

We recapitulate the definition of standard CF grammar to serve as a point of comparison.

G = **<N,T,P,S>, N, T** disjoint and finite, **P** a finite subset of **NxV*.**

u => w iff u = xAy and w = xUy for some A -> U in P, xy in V*.

w in L(A) iff A =>* w. L(G) = L(S).

Then the definition of UCFG can be approached as follows.

1.2. Definition of unification based CF grammar (UCFG)

G = (N,T,P,S) where **N = L(G')** for the following CFG:

G' = (N',T',P',Cat), T' = F + C + =),(,1,v,fail.

P' =
Cat -> fail
Cat -> Var, Var -> v; Varl
Cat -> c, c in C
Cat -> (f1=Cat f2=Cat ... fn=Cat), where f1,f2,...,fn = F

P a finite subset of **N x V*, S in N.**

The foremost difference here is that the set of nonterminal symbols of a UCFG is infinite (it is the language of another context free grammar).

Although only a finite number of nonterminals can occur in rules, a rule can match (unify with) an infinity of different nonterminals in the course of different derivations. In other words, a UCFG rule can have an infinite number of rule instances.

To define the yield relation, we need to introduce a number of auxiliary concepts.

1.2.1. Substitutions

An **substitution** is a partial function **s** from **L(Var)** to **L(Cat)**. A substitution **s** is extended to V* by the clause **s(xy)** = **s(x)s(y)**.

A substitution which is one-one in **L(Var)** is a called a renaming of variables. We define a notational equivalence relation == among alphabetic variants so that **x==y** if **x** = **s(y)** for a renaming of variables **s**. == is extended to productions in the obvious way: **p** = **A** -> **U** == **B** -> **W** iff **AU** == **BW**.

A substitution **s** with values in **L(Var)** can be iterated. To define eventual values of **s**, we define **s*(x)** so that **s*(x)** = **x** if **s(x)** is undefined or $s^n(x) = x$ for some n>0; else **s*(x)** = **s*(s(x))**.

The smallest substitution is the empty substitution **0**. The inconsistent substitution **1** is such that **1(x)** = **fail** for any **x**.

1.2.2. Unification

A **unifier** for categories **A,B** is a substitution **s** s.t. **s(A)** = **s(B)**. **s** is a **maximally general unifier (mgu)** for **A,B**, or **s** = **mgu(A,B)**, if **u(s(AB))** = **u(AB)** for any unifier **u** for **A,B**. It can be determined up to alphabetic variance using the following schema.

mgu(A,B) = **u(A,B,0)**, where

u(c,c,0) = 0, c in C

u(v,B,0) = <v,B> if **v** in L(Var) does not occur in **B.**

u((f1=c1...fn=cn),(f1=d1...fn=dn),0) = u(c1,d1,u((f2=c2...fn=cn),(f2=d2...fn=dn),0)

u(A,B,s) = s + u(s*(A),s*(B),0)

otherwise **u(A,B,s) = 1.**

A unifies with B iff **mgu(A,B)** is not **1.** The **unification** of **A** and B is **mgu(A,B)'(A) = mgu(A,B)'(B). A subsumes** B iff **mgu(A,B)(A)** == **A.** That is, unification of **B** into **A** does not change **A.**

1.2.3. Derivability

With these basic concepts, we can define the UCFG yield relation as follows.

u => **w** iff **u** = **xAy** and **w** = **s(xWy)** where B -> **W** == **p** for some production **p in P, xy in V*** and substitution **s** = **mgu(A,B).**

The main differences compared to CF are that the matching relation between **A** and the left hand side of **p** is not identity but unification (a rule can match an infinite number of nonterminals consistent with it) and that the expansion of **A** with **p** can instantiate (rewrite) nonterminals in **u** outside **A.**

The == relation in the definition allows renaming of variables between repeated applications of a rule. We have chosen to rename the production **p.** Equivalently, we could rename **u.**

Example UCFG:

(cat=S num=v0) -> (cat=A num=v1) (cat=B num=v1) (cat=C num=v1)

(cat=v1 num=(num=v2)) -> (cat=v1 num=v2) (cat=v1 num=0)

(cat=A num=0) -> a

(cat=B num=0) -> b

(cat=C num=0) -> c

This UCFG generates the non-CF language $a^n b^n c^n$.

1.3. CF IDLP grammar

Standard rewriting rules contain dominance and precedence information connected together. The idea of IDLP grammars is to separate dominance from precedence. There are two types of rules: ID rules of form

A -> B1, ... , Bn

where **A** is a nonterminal and **B1, ... Bn** form a multiset (set with possible repetitions) of symbols in **V**, and LP rules of form

A < B

where **A** and **B** are nonterminals in **N.**

A CFIDLP grammar **H** can be defined as a pair **(G,<)** where **G** is a standard CF grammar and < is a strict partial ordering in **NxN.** For simplicity, we fix a standard ordering of the RHS's of ID rules which contains < as a subset. Henceforth we assume ID rules are normalized into standard order.

To define the yield relation, we proceed as follows. Let **L** be a subset of **V***. We define **Sat(L,<)** as the set of those words **w in L** that are not of form **xByAz** where **A < B.** If **w** is in **Sat(V*,<)** we say that **w** satisfies <. A production **p** satisfies < iff its RHS satisfies <. We define a relation --> in **V*** so that **x --> y** iff **y** is a permutation of **x** and **y** satisfies <.

Then **u => w** in **H = (G,<) iff u** = **xAy, w** = **xUy,** and **A -> W** is in **P** st.t **W --> U.**

The novelty here is that a given production actually defines a set of ordered productions obtained by permuting its right hand side in LP acceptable ways.

Thus any IDLP grammar has an equivalent ordered grammar. Conversely, any ordered G has a trivial equivalent ILDPG. The conversions affect the nonterminal vocabulary and therewith the parse trees generated by the grammars.

In Gazdar et al. (1985:49) the strong generative capacity of IDLP grammars is characterized in terms of the ECPO (Equivalent Constant Partial Ordering) property. In an IDLPG with fixed nonterminal vocabulary, if A < B in the RHS of one production, A < B in the RHS of all productions. If and only if a ordered grammar has this property, it can be rewritten as an IDLP grammar without changing the nonterminal vocabulary.

Since it is easy to move from the extended vocabulary of a covering IDLP grammar to the original vocabulary of the ordered grammar in any given derivation (cf. Aho and Ullman72:275), the ECPO property is of slight practical interest. This is all the more true in UCFG, where the feature composition of the top category provides a more flexible description of sentence structure than derivation history.

1.3.2. Parsing of IDLP grammar

This section assumes standard notions of Earley parsing (see e.g. Aho and Ullman 1972).

Barton (1985) shows parsing of IDLP grammars NP complete in the worst case (when < is empty). The basic fact is that the number n! of permutations of a string of length n grows exponentially with n. The combinatorial explosion arises in cases of lexical ambiguity, where (say) a string of form **a1...an** can be parsed in n! ways by an IDLP grammar **H = (G,<)** with **G = (S -> A1...An, Ai -> aj for all i,j)** and < empty.

Though exponential in the worst case, direct parsing of IDLP grammars can show savings compared to parsing corresponding ordered grammars. The source of the savings is that the number of items in the parse table can be kept small by keeping together items that are represented separately in the ordered grammar. In the worst case, this gets the number of items down from O(/G/!) to $O(2^{/G/})$ (one item per subset of RHS symbols instead of one per permutation).

An Earley parse item **A -> x.y** is <--> to item **A -> z.w** iff **x** <--> **z** and **y** <--> **w.** To simplify the identification of equivalent items, they can be normalized to a fixed alphabetic order.

The test **yB** <--> **By** involves checking LP-acceptability of **By** This can be done by looping through all LP rules and pairs **B,C, C** in **y.** Since the result of the test depends only on **G,** it can be precomputed.

2. UIDLP grammar

2.1. Definition

Analogy with the CF case suggests the following definitions.

A UIDLP grammar is a pair **H = (G,<)** where **G** is a UCFG and < is a strict partial order on **V**.

Let **L** be a subset of **V***. **Sat(L,<) = (w in L: w** is not of form **xCyDz** where **A** < B and **CD** subsumes **BA)**. If **w** is in **Sat(V*,<)** we say that **w** satisfies <. A production **p** satisfies < iff its **RHS** satisfies <. **x --> y** iff **y** is a permutation of **x** and **y** satisfies <.

Then **u** => **w** in **H** = **(G,<)** iff **u** = **xAy** and **w** = **s(xWy)** where **s(W)** <--> U and B -> **U** == **p** for some production **p in P, xy in V*** and substitution **s** = **mgu(A,B)**.

This definition combines the UCFG and IDLP yield relations in the straightforward way. LP rules are used as local tests checking a permutation of the RHS of a rule when the rule is applied. Category identity as the matching relation is generalized to subsumption.

2.2. Nonlocal subsumption problem

It follows from these definitions that the LP-acceptable permutations of an instance of a UID rule can be a proper subset of the permutations of the original rule. The reason is that in general, **Sat(L,<)** is a subset of **Sat(s(L),<)** but not vice versa. Instantiation can make LP rules applicable which do not apply to the uninstantiated rule.

In other words, in UIDLP grammar, word order can be sensitive to context. For example, the order of a V and its complements may depend on the character of the clause they belong to (German, Finnish). This cannot be decided locally by looking at the verb and its complements. This situation is exemplified in the following UIDLP grammar. (CF category symbols indexed with feature equations serve as shorthands for UCFG category symbols.)

-42-

S' -> S(type=main)
S' -> Comp S(type=sub)
S(type=x) -> NP VP(type=x)
VP(type=x) -> V(type=x) NP
Comp -> dass
NP -> Jungen
V -> sind
V(type = main) < NP
NP < V(type=sub)

This grammar left generates the sentences Jungen sind Jungen and dass Jungen Jungen sind (but not the illicit orders).

However, the same grammar parsed bottom up right to left accepts the illicit sentence dass Jungen sind Jungen:

```
S'
Comp           S(type=sub)
               S(type=x)
               NP      VP(type=x)          - passes LP rules
                       V(type=x)  NP
dass          Jungen   sind       Jungen
```

A similar paradox can be constructed between left and right top down derivations of a variant grammar with rules **S' -> Comp(type=x) S (type=x), Comp(type=main) -> e, Comp(type=sub) -> dass.**

This result is undesirable in that it imports an order dependent, procedural feature to an otherwise declarative formalism. The LP-acceptability of a sentence can depend on the order applying the rules of grammar, so that certain parsing strategies can pass sentences which fail LP-rules on the surface.

As far as the definition of derivability is concerned, what we should have said to begin with is fairly clear. We want LP rules to regulate the order of sister constituents at all stages of derivation (in particular, at the end of derivations). This can be stated as a global condition on derivations, or, as is actually done in GPSG (Gazdar et al. 1985:46), by interpreting LP conditions as node admissibility conditions. In this interpretation, an LP-rule **A** < **B** reads:

(LP) A pair of nodes **B'A'** subsuming **BA** cannot appear as sisters in a derivation tree.

Given that LP-acceptability is taken care of by (LP), we can simplify the UIDLPG yield relation as follows:

u => **w** in **H** = **(G,<)** iff **u** = **xAy** and **w** = **s(xWy)** where **s(W)** is a permutation of **U** and **B** -> **U** == **p** for some production **p in P, xy in V*** and substitution **s** = **mgu(A,B)**.

2.3. Parsing of UIDLP grammars

One way to parse UIDLP as revised above is to simply apply LP rules in the completed parse so as to filter out LP-inconsistent parses.

This is conceptually straightforward but inefficient. All permutations would be considered only to be discarded at the final step. We should bring LP constraints to bear as soon as possible, i.e. apply LP rules as global constraints on derivations.

Consider again the illicit parse above. Although **V** and its complement **NP** do not subsume the LP rule **NP < V(type=sub)**, they do unify with the rule. (If they did not unify with the rule, they could not subsume it later either.) Such undecided applications of LP rules should remain active until a decision can be made. (Again, the decision should be done as early as possible to cut off parses.) Let us say that in such cases, the LP rule **properly unifies** with the pair of categories.

Assume **B < C** properly unifies with **B', C'** when item **A -> xC'.yB'z** is formed. We need to save the active constraint with the undecided item in such a way that it will be reapplied to instances of the original undecided pair.

To do so, we associate with each undecided item with a list of constraints of the following kind. A constraint **c** is of form **"A < B to B'A'"**. It is **matched** if **A'B'** subsumes **AB**, **irrelevant** if **A'B'** does not unify with **AB**, and **active** otherwise. If **s** is an substitution, **s(c) = "A < B to s(B'A')"**.

Whenever a new item **i = s(A -> xC'.yz)** is to be combined from items **j = A -> x.yC'w** and **k = C" -> u.** using mgu **s**, we check each constraint on the constraint list **(s(c): c is on the constraint list of i or j or is of form "B < C to C'B'", B' in y and B < C a LP rule.)** If **c** is matched, reject the combination. If **c** is irrelevant, delete the constraint from the constraint list. Finally, if the combination is not rejected, assign the remaining constraints as the constraint list of the new item.

The above procedure is rather cumbersome. What is more, it is difficult to find cases in actual languages where it is really needed. The types of context sensitive word order constraints I have found allow for a simpler fix which is sketched in the following section.

2.4. Generalized LP rules

A UCFG category can be represented by a set of feature equations, specifying values of features instantiated in each category. Conversely, the set constitutes the least solution of its representing equations in the domain of UCFG categories.

A pair of UCFG categories can be similary represented as a higher order category with attributes 0,1 for the members of the pair. Feature equations can then be used to describe category pairs. (The idea is adapted from Karttunen (this volume).)

The standard interpretation of a unification LP rule **A < B** can be restated as follows.

(1) If **A'B'** subsumes **AB**, then **A' < B'**.

The contraposition of this constraint (given **A'** is not **B'**) is

(2) If **A' > B'**, then not: **A'B'** subsumes **AB**.

If the complement **c(AB)** of the specification **AB** can be expressed, the contrapositive constraint can be further rewritten as

(3) If **A' > B'** then **A'B'** unifies with **c(AB)**.

Then (3) could be directly verified by unifying **c(AB)** into A'B' whenever the antecedent of (3) is taken. This suffices to guarantee satisfaction of the consequent of (3), in fact strengthens it to subsumption.

In our example grammar, since **(type=sub) = c(type=main)**, application of **NP < V(type=sub)** to the pair **V(type=x) NP** could simply instantiate **V** into **V(type=main)**.

These observations suggest the following generalization of the notion of a LP rule.

(4) Assume **x > y**. Then if **xy** subsumes **AB** then **xy** subsumes **CD**.

Or equivalently,

(5) **x < y** if **xy** subsumes **AB**, else **xy** subsumes **CD**.

Let us consider some special cases of this general form. The original rule **A < B** is obtained as

(6) **x** < **y** if **xy** subsumes **AB** else **xy** subsumes **fail.**

Our example of nonlocal subsumption is taken care of by the pair of rules

(7) Assume **x** > **y.** Then if **xy** subsumes **((x cat) = V)(y cat) = NP))** then **xy** subsumes **((x type) = sub)**

(8) Assume **x** > **y.** Then if **xy** subsumes **((x cat) = NP)(y cat) = V))** then **xy** subsumes **((x type) = main)**

These rules instantiate the main and subordinate clause features at the time when the VP order is fixed. (An abbreviation of complementary rules such as (7)-(8) into one rule seems appropriate.)

The next example is a rule of functional word order interpretation.

(9) **x** < **y** if **xy** subsumes **(x = (y subject))** else **xy** subsumes **((x function) = rheme)**

This rule says that a subject following the main verb is rhematic.

As suggested before, generalized LP rules of form (4) (respectively, (5)) can be implemented in parsing so that the consequent (else) condition of the rule is actually unified in when the order condition of the rule is applicable (violated).

An implementation of generalized LP rules of this kind into HUG is in progress.

It should be kept in mind that generalized LP rules do not solve the nonlocal subsumption problem. At best, they make it possible to avoid the problem by allowing statement of some context sensitive word order rules without reference to nonlocal conditions.

References

Aho, A. and J. Ullman (1972), **The Theory of Parsing, Translation, and Compiling. Volume 1: Parsing.** Prentice-Hall.

Barton, E. (1985), "The Computational Difficulty of ID/LP Parsing", in **Proceedings of the 23rd Annual Meeting of the ACL,** Chicago.

Gazdar, G, E. Klein, G. Pullum and I. Sag (1985), **Generalized Phrase Structure Grammar.** Harvard University Press.

Hopcroft, J. and J. Ullman (1979), **Introduction to Automata Theory, Languages, and Computation.** Addison-Wesley.

Shieber, S. (1984), "Direct Parsing of ID/LP Grammars", **Linguistics and Philosophy 7,** 135-154.