# A Generalized Greibach Normal Form for Definite Clause Grammars

Marc Dymetman
CCRIT, Communications Canada
1575 boul. Chomedey
Laval (Québec) H7V 2X2, Canada
dymetman@ccrit.doc.ca

## Abstract

An arbitrary definite clause grammar can be transformed into a so-called Generalized Greibach Normal Form (GGNF), a generalization of the classical Greibach Normal Form (GNF) for context-free grammars.

The normalized definite clause grammar is declaratively equivalent to the original definite clause grammar, that is, it assigns the same analyses to the same strings. Offline-parsability of the original grammar is reflected in an elementary textual property of the transformed grammar. When this property holds, a direct (top-down) Prolog implementation of the normalized grammar solves the parsing problem: all solutions are enumerated on backtracking and execution terminates.

When specialized to the simpler case of context-free grammars, the GGNF provides a variant to the GNF, where the transformed context-free grammar not only generates the same strings as the original grammar, but also preserves their degrees of ambiguity (this last property does not hold for the GNF).

The GGNF seems to be the first normal form result for DCGs. It provides an explicit factorization of the potential sources of undecidability for the parsing problem, and offers valuable insights on the computational structure of unification grammars in general.

## 1 Introduction

From the point of view of decidability, the parsing problem with a definite clause grammar is much more complex than the corresponding problem with a context-free grammar. The parsing problem with a context-free grammar is *decidable*, i.e. there exists an algorithm which determines whether a given string is accepted or not by the grammar;[1] by contrast, the parsing problem with a definite clause grammar is *undecidable*, even if the auxiliary Prolog predicates are restricted to be unifications.[2]

In other words, there does not exist, in general, an algorithm for deciding whether a definite clause grammar $DCG_1$ accepts a string $String$, i.e. whether there exists some linguistic structure $S$ such that $DCG_1$ "analyses $String$ into $S$". On the other hand, under the condition that $DCG_1$ is *offline-parsable*, that is, under the condition that the context-free skeleton of $DCG_1$ is not infinitely ambiguous, then the parsing problem is indeed decidable [7].

The fact that the parsing problem with $DCG_1$ is decidable in theory should be carefully distinguished from the fact that a *given* algorithm for parsing is able to exploit this potentiality. A parsing algorithm for which this is the case, that is, an algorithm which is able, for any offline-parsable $DCG_1$, to decide whether a given string is accepted or not by $DCG_1$ is said to be *strongly stable* [7]. Most parsing algorithms are not strongly stable[3], a notable exception being the "Earley deduction" algorithm [7], once modified according to the proposals in [10].[4]

Top-down implementations—and in particular the standard Prolog implementation of a DCG—are especially fragile in this respect, for they fail to terminate as soon as the grammar contains a left-recursive rule such as:

$$vp(VP_2) \rightarrow vp(VP_1), pp(PP), \{combine(VP_1, PP, VP_2)\}.$$

In [2], automatic local transformations were performed on a DCG in order to eliminate some limited, but frequent in practice, types of left-recursion in such a way that the resulting grammar could be directly implemented in Prolog. This initial work led us naturally to the more general question:

**Question:** Is it possible to automatically transform an

[1] For instance the standard top-down parsing algorithm using a GNF of the grammar.

[2] This assumption, or a similar one, is always made when the decidability properties of logical (or unification) grammars are studied. The reason is simple: if the auxiliary predicates are defined through an unrestricted Prolog program, there is no means of guaranteeing that calling some auxiliary predicate will not result in nontermination, for reasons quite independent from the structure of the grammar under consideration. The assumption that auxiliary predicates are unifications takes care of this interfering problem.

[3] An important class of offline-parsable grammars—which we will call here the class of *explicitly offline-parsable* grammars—is the class of DCGs whose context free skeleton is a *proper* context-free grammar, that is, a grammar without rules of the type $A \rightarrow [\ ]$ (empty productions) or of the type $A \rightarrow B$ (chain rules) [1]. This subclass is much less problematic to parse than the full class of offline-parsable DCGs (for instance a left-corner parsing algorithm will work). However, it is an easy consequence of the GGNF result that, for any offline-parsable DCG, there exists an explicitly offline-parsable DCG equivalent to it.

[4] Stuart Shieber, personal communication.

arbitrary offline-parsable $DCG1$ into an equivalent $DCG2$ in such a way that the standard top-down execution of $DCG2$ terminates (in success or failure) for any given input string?

To answer this question (in the positive), we have taken an indirect route, and, adapting to the case of definite clause grammars some of the standard techniques used to transform context-free grammars into GNF, [5] have established the following results:

1. *Generalized Greibach Normal Form for definite clause grammars* It is possible to transform any definite clause grammar $DCG1$ (offline-parsable or not) into a definite clause grammar $DCG2$ equivalent to $DCG1$—that is, assigning the same analyses to the same strings—and which is in a certain form, called the *Generalized Greibach Normal Form*, or *GGNF*, of $DCG1$.
2. *Explicit representation of offline-parsability in the GGNF* The offline-parsability of $DCG1$ is equivalent to an elementary textual property of $DCG2$: the so-called "unit subgrammar" contains no cycle, i.e. no nonterminal calling itself directly or indirectly.[6]
3. *Termination condition for top-down parsing with the GGNF* If $DCG1$ is offline-parsable, and such that its auxiliary predicates it are unifications, then, for any input $String$, the standard top-down (Prolog) execution of $DCG2$ enumerates all solutions to the parsing problem for $String$ and terminates.

## 2 The Generalized Greibach Normal Form for Definite Clause Grammars

### 2.1 Definite Grammar Schemes

As usually defined (see [6]), a definite clause grammar consists in two separate sets of clauses:

1. *Nonterminal clauses*, written as:

$$a(T_1, \ldots, T_n) \rightarrow \alpha \quad ,$$

where $a$ is a nonterminal, $T_1, \ldots, T_n$ are terms (variables, constants, or complex terms), and $\alpha$ is a sequence of "terminal goals" $[term]$, of "nonterminal goals" $b(T'_1, \ldots, T'_m)$, and of "auxiliary predicate goals" $\{p(T''_1, \ldots, T''_k)\}$.

2. *Auxiliary clauses*, constituting an autonomous definite program, defining the auxiliary predicates appearing in the right-hand sides of nonterminal rules. These clauses are written:

$$p(T_1, \ldots, T_l) :\text{-}\ \beta \quad ,$$

where $\beta$ is some sequence of predicate goals.

A *definite grammar scheme* $DGS$ is syntactically identical to a definite clause grammar, except for the fact that:

1. The $T_i$ arguments appearing in the nonterminal and auxiliary predicate goals are restricted to being variables: no constants or complex terms are allowed;
2. Only nonterminal clauses appear in the definite grammar scheme, but no auxiliary clause; the auxiliary predicates which may appear in the right-hand sides of clauses do not receive a definition.

A definite grammar scheme can be seen as an uncompletely specified definite clause grammar, that is, a definite clause grammar "lacking" a definition for the auxiliary predicates $\{p(X_1, \ldots, X_n)\}$ that it "uses". The auxiliary predicates are "free": the interpretation of $p$ is not fixed *a priori*, but can be an arbitrarily chosen $n$-ary relation on a certain Herbrand universe of terms.[7]

**Example 1** The following clauses define a definite grammar scheme $DGS_1$:[8]

$$\begin{aligned}
&a1(X) \rightarrow a3(E),\ a1(A),\ a2(B),\ \{q(E, A, B, X)\} \\
&a1(X) \rightarrow [oh],\ \{p1(X)\} \\
&a2(X) \rightarrow [oui],\ \{p2(X)\} \\
&a3(X) \rightarrow [\,],\ \{p3(X)\}
\end{aligned}$$

In this definite grammar scheme, only variables appear as arguments; the auxiliary predicates $p1$, $p2$, $p3$ and $q$ do not receive a definition.

If a definite program defining these auxiliary predicates is added to the definite grammar scheme, one obtains a full-fledged definite clause grammar, which can be interpreted in the usual manner.

Conversely, every definite clause grammar can be seen as the conjunction of a definite grammar scheme and of a definite clause program. In order to do so, a minor transformation must be performed: each complex term $T$ appearing as an argument in the head or body of a nonterminal clause must be replaced by a variable $X$, this variable being implicitly constrained to unify with $T$ through the addition of an ad-hoc unification goal in the body of the clause (see [3]).

[5] See the Appendix for some indications on the methods used.

[6] Thus, a side-effect of the GGNF is to provide a decision procedure for the problem of knowing whether a DCG is offline-parsable or not. This is equivalent to deciding whether a context-free grammar is infinitely ambiguous or not, a problem the decidability of which seems to be "quasi-folk-knowledge", although I was innocent of this until the fact was brought to my attention by Yves Schabes, among others: the proof is more or less implicit in the usual technique to make a CFG "cycle-free", [1, p. 150] . See also [4] for a special case of this problem. (*Caveat.* The notion of "cycle" in "cycle-free" is technically different from the notion used here, which simply means: cycle in the graph associated with the relation "callable from". See note 10.)

[7] The domain of interpretation can in fact be any set. Taking it to be the Herbrand universe over a certain vocabulary of functional symbols permits to "simulate" a DCG, by fixing the interpretation of the free auxiliary predicates in this domain. Another linguistically relevant domain of interpretation is the set of directed acyclic graphs built over a certain vocabulary of labels and atomic symbols, which permits the simulation of unification grammars of the PATR-II type.

[8] The usual symbol for the initial nonterminal is $s$; we prefer to use $a1$ for reasons of notational coherence.

**Example 2** Consider the following definite clause grammar:

$$a1(cons(B,A)) \rightarrow a3(E),\ a1(A),\ a2(B).$$
$$a1(X) \rightarrow [oh],\ \{p1(X)\}.$$
$$a2(f) \rightarrow [oui].$$
$$a3(X) \rightarrow [\,],\ \{p3(X)\}.$$

$$p1(nil).$$
$$p3(r).$$

Let us define two new predicates $p2$ and $q$ by the following clauses:

$$p2(f).$$
$$q(E,A,B,cons(B,A)).$$

then the definite clause grammar above can be rewritten as:

$$a1(X) \rightarrow a3(E),\ a1(A),\ a2(B),\ \{q(E,A,B,X)\}.$$
$$a1(X) \rightarrow [oh],\ \{p1(X)\}.$$
$$a2(X) \rightarrow [oui],\ \{p2(X)\}.$$
$$a3(X) \rightarrow [\,],\ \{p3(X)\}.$$

$$p1(nil).$$
$$p2(f).$$
$$p3(r).$$
$$q(E,A,B,cons(B,A)).$$

that is, in the form of a definite grammar scheme to which has been added a set of auxiliary clauses defining its auxiliary predicates. This definite grammar scheme is in fact identical with $DGS_1$ (see previous example).

In the sequel of this paper, we will be interested, not directly in transformations of definite clause grammars, but in *transformations of definite grammar schemes*. The transformation of a definite grammar scheme $DGS$ into $DGS'$ will respect the following conditions:

- The auxiliary predicates of $DGS$ and of $DGS'$ are the same;
- For any definite clause program $P$ which defines the auxiliary predicates in $DGS$ (and therefore also those in $DGS'$), the definite clause grammar $DCG$ obtained through the adjunction of $P$ to $DGS$ has the same denotational semantics as the definite clause grammar $DCG'$ obtained through the adjunction of $P$ to $DGS'$.

Under the preceding conditions, $DGS$ and $DGS'$ are said to be *equivalent definite grammar schemes*.

The grammar transformations thus defined are, in a certain sense, *universal transformations*: they are valid independently from the interpretation given to the auxiliary predicates.

## 2.2 GGNF for definite clause grammars

**Structure of the GGNF** The definite grammar scheme $DGS$, on the terminal vocabulary $V$, having $Q$ as its set of auxiliary predicates, is said to be in *Generalized Greibach Normal Form* if:

1. The nonterminals of $DGS$ are partitioned in three distinct subsets: $A = \{a1\}$, called the initial set; $U$, called the set of unit nonterminals; $N$, called the set of co-unit nonterminals.

2. The rules of $DGS$ are partitioned into three groups of rules, called the factorization group (defining the elements of $A$), the unit group (defining the elements of $U$), and the co-unit group (defining the elements of $N$), graphically presented in the following manner:

   *factorization rules: definition of the elements of A*

   ——

   *unit rules: definition of the elements of U*

   ——

   *co-unit rules: definition of the elements of N*

3. Factorization rules are taken among the two following rules:

   $$a1(X_1,\ldots,X_n) \rightarrow n1(X_1,\ldots,X_n)\ ,$$
   and/or:
   $$a1(X_1,\ldots,X_n) \rightarrow u1(X_1,\ldots,X_n)\ ,$$

   where $n1 \in N$ and $u1 \in U$, and where $n \in \mathbf{N}$ is the arity of the initial nonterminal $a1$.

4. Unit rules are of the form:

   $$u(X_1,\ldots,X_m) \rightarrow \mathcal{U}\ ,$$

   where $u \in U$ is a unit nonterminal of arity $m$, $m \in \mathbf{N}$, and where $\mathcal{U}$ is a finite sequence of nonterminal unit goals of $U$, of auxiliary predicates of $Q$, or is the empty string $[\,]$. The group of unit rules forms a subscheme of the GGNF definite grammar scheme (see below).

5. Co-unit rules are of the form:

   $$n(X_1,\ldots,X_k) \rightarrow [term]\ \mathcal{N}\ ,$$

   where $n \in N$ is a co-unit nonterminal of arity $k$, $k \in \mathbf{N}$, where $[term] \in V$ and where $\mathcal{N}$ is a finite sequence of terminal goals of $V$, of nonterminal unit goals of $U$, of auxiliary predicates of $Q$, or of nonterminal co-unit goals of $N$.

6. The context-free skeleton of $DGS$, considered as a context-free grammar, is *reduced*.[9]

[9]A context-free grammar is said to be reduced iff it all its nonterminals are accessible from $a_1$ and are productive (see [5, pp. 73–78])

The nonterminals of $A$, $U$, and $N$ are defined in function of one another, as well as in function of [ ], of the terminals of $V$, and of the auxiliary predicates of $Q$, according to the definitional hierarchy illustrated below:

$$\begin{tikzcd}
 & A \arrow[dl] \arrow[dr] & \\
N \arrow[loop left] \arrow[rr] \arrow[d] \arrow[dr, dashed] & & U \arrow[loop right] \arrow[d] \arrow[dl, dashed] \\
V & Q & {[\,]}
\end{tikzcd}$$

## 2.3 Structure of the unit subscheme and offline-parsability

One can remark that:

- The group of unit rules is *closed*: the definition of unit nonterminals involves only unit nonterminals (but no co-unit nonterminal). For this reason, the group of unit rules is called the ***unit subscheme*** (or, loosely, the ***unit subgrammar***) of the GGNF definite grammar scheme.

- The unit subscheme can only generate the empty string [ ].

The unit subscheme of the GGNF is said to ***contain a cycle*** iff there exists a unit nonterminal $u(X_1, \ldots, X_m)$ which "calls itself recursively", directly or indirectly, inside this group.[10] One can show that this property is equivalent to the fact that the context-free skeleton of $DGS$ is infinitely ambiguous, or, in other words, to the fact that $DGS$ is *not* offline-parsable [3].

## 2.4 Top-down parsing with the GGNF

Let $DGS$ be a definite grammar scheme in GGNF, having $Q$ for its set of auxiliary predicates. Assume that every element $p$ of $Q$, of arity $n$, is defined through a head clause[11], of the form:

$$p(T_1, \ldots, T_n).$$

where $T_1, \ldots, T_n$ can be any terms; In other words, the auxiliary predicates are constrained to be simply unifications. Let $DCG$ be the definite clause grammar obtained through adjunction of these clauses to $DGS$. The grammar $DCG$ has the following properties:

[10]For example, the scheme:

$$\begin{array}{lcl} u_1(X) & \rightarrow & u_2(Y),\ u_3(Z), \ldots \\ u_2(X) & \rightarrow & u_4(Y),\ u_1(Z), \ldots \\ \ldots & & \end{array}$$

contains a cycle in $u_1$.

[11]We use the terminology 'head clause' for a clause without body. A more standard terminology would be 'unit clause', but this would conflict with our technical notion of 'unit' (a nonterminal generating the empty string [ ]).

1. If the unit subscheme does not contain a cycle, then, for any input string $String$, the standard top-down parsing algorithm terminates, after enumerating all the analyses for $String$;

2. If the unit subscheme contains a cycle, the top-down parsing algorithm can terminate or not, depending on the definition given to the auxiliary predicates.

We give below three examples of definite grammar schemes, and of the equivalent definite grammar schemes in GGNF.

## 2.5 Examples

**Example 3** Consider the definite grammar scheme $DGS_1$ given in Example 1, repeated below:

$$\begin{array}{l} a1(X) \rightarrow a3(E),\ a1(A),\ a2(B),\ \{q(E,A,B,X)\} \\ a1(X) \rightarrow [oh],\ \{p1(X)\} \\ a2(X) \rightarrow [oui],\ \{p2(X)\} \\ a3(X) \rightarrow [\,],\ \{p3(X)\} \end{array}$$

The following definite grammar scheme $DGS_2$ is in GGNF and is equivalent to $DGS_1$:

$$\begin{array}{l} a1(X) \rightarrow n1(X) \\ \overline{\phantom{xx}} \\ \emptyset \\ \overline{\phantom{xx}} \\ n1(X) \rightarrow [oh],\ \{p1(X)\} \\ n1(X) \rightarrow [oh],\ \{p1(Y)\},\ h(Y,X) \\ h(Y,X) \rightarrow [oui],\ \{p2(B)\},\ \{p3(E)\}, \\ \qquad\qquad \{q(E,Y,B,X)\} \\ h(Y,X) \rightarrow [oui],\ \{p2(B)\},\ \{p3(E)\}, \\ \qquad\qquad \{q(E,Y,B,Z)\},\ h(Z,X) \end{array}$$

Suppose $P$ is any auxiliary definite program which defines the auxiliary predicates $p1, p2, p3, q$. Then the definite clause grammars $DCG_1$ and $DCG_2$, obtained by adjunction of this program to $DGS_1$ and $DGS_2$, respectively, are equivalent.

The unit subscheme of $DGS_2$ does not contain a cycle (it is empty[12]). One can conclude that $DCG_1$, as well as $DCG_2$, are offline-parsable. If, moreover, it is assumed that $P$ defines the auxiliary predicates as being unifications, then it can be concluded that top-down parsing with $DCG_2$ will enumerate all possible analyses for a given string and terminate.

For instance, assume that the auxiliary program consists in the following four clauses (see Example 2):

$$\begin{array}{l} p1(nil). \\ p2(f). \\ p3(r). \\ q(E,A,B,cons(B,A)). \end{array}$$

[12]See note 14.

Then, $DCG_1$ becomes:

$a1(X) \rightarrow a3(E),\ a1(A),\ a2(B),\ \{q(E,A,B,X)\}.$
$a1(X) \rightarrow [oh],\ \{p1(X)\}.$
$a2(X) \rightarrow [oui],\ \{p2(X)\}.$
$a3(X) \rightarrow [\,],\ \{p3(X)\}.$

$p1(nil).$
$p2(f).$
$p3(r).$
$q(E,A,B,cons(B,A)).$

and $DCG_2$:

$a1(X) \rightarrow n1(X)$
$n1(X) \rightarrow [oh],\ \{p1(X)\}$
$n1(X) \rightarrow [oh],\ \{p1(Y)\},\ h(Y,X)$
$h(Y,X) \rightarrow [oui],\ \{p2(B)\},\ \{p3(E)\},$
$\qquad \{q(E,Y,B,X)\}$
$h(Y,X) \rightarrow [oui],\ \{p2(B)\},\ \{p3(E)\},$
$\qquad \{q(E,Y,B,Z)\},\ h(Z,X)$

$p1(nil).$
$p2(f).$
$p3(r).$
$q(E,A,B,cons(B,A)).$

These two definite clause grammars are declaratively equivalent. They both accept strings of the form:

*oh oui ... oui*

where *oui* is repeated $k$ times, $k \in \mathbf{N}$, and assign to each of these strings the (single) analysis represented by the term:

if $k = 0$ : $nil$ ,
if $k > 0$ : $cons(f, \ldots cons(f, nil) \ldots)$
($cons$ repeated $k$ times.)

On the other hand, from the operational point of view, if a top-down parsing algorithm is used, $DCG_1$ loops on any input string,[13] while $DCG_2$ enumerates all solutions on backtracking—here, zero or one solution, depending on whether the string is in the language generated by the grammar—and terminates.

**Example 4** Consider the following definite grammar scheme $DGS_3$:

$a1(X) \rightarrow [oui],\ a1(Y),\ \{p(X,Y)\}$
$a1(X) \rightarrow a2(Y),\ \{r(X,Y)\}$
$a2(X) \rightarrow [\,],\ \{q(X)\}$

[13]Remark that $DCG_1$ is left recursive in a "vicious" (covert) way: nonterminal $a1$ calls itself, not immediately, but after calling $a3$, which does not consume anything in the input string.

The GGNF of $DGS_3$ is $DGS_4$ below:

$a1(X) \rightarrow n1(X)$
$a1(X) \rightarrow u1(X)$

———

$u1(X) \rightarrow u2(Y),\ \{r(X,Y)\}$
$u2(X) \rightarrow [\,],\ \{q(X)\}$

———

$n1(X) \rightarrow [oui],\ n1(Y),\ \{p(X,Y)\}$
$n1(X) \rightarrow [oui],\ u1(Y),\ \{p(X,Y)\}$

From an inspection of $DGS_4$ it can be concluded that:

- The unit subscheme does not contain a cycle.[14] Therefore $DGS_4$, and consequently $DGS_3$, is offline-parsable.
- If $DCG_3$ (resp. $DCG_4$) is the definite clause grammar obtained through the adjunction to $SDG_3$ (resp. $SDG_4$) of clauses defining the auxiliary predicates $p, q, r$, then $DCG_3$ and $DCG_4$ are equivalent; Furthermore, if these definitions make $p, q, r$ unification predicates, then top-down parsing with $DCG_4$ terminates, after enumerating all solutions.

**Example 5** Consider the following definite grammar scheme $DGS_5$:

$a1(X) \rightarrow [oh],\ \{p(X)\}$
$a1(X) \rightarrow a1(Y),\ \{q(X,Y)\}$

The GGNF of $DGS_5$ is $DGS_6$ given below:

$a1(X) \rightarrow n1(X)$

———

$u(X,X) \rightarrow [\,]$
$u(X,Y) \rightarrow \{q(X,Z)\},\ u(Z,Y)$

———

$n1(X) \rightarrow [oh],\ \{p(Y)\},\ u(X,Y)$

From an inspection of $DGS_6$ it can be concluded that:

- The unit subscheme contains a cycle. Therefore neither $DGS_6$ nor $DGS_5$ are offline-parsable.
- If $DCG_5$ (resp. $DCG_6$) is the definite clause grammar obtained through the adjunction to $DGS_5$ (resp. $DGS_6$) of clauses defining the auxiliary predicates $p, q$, then $DCG_5$ and $DCG_6$ are equivalent;
- Even if $p, q$ are defined as unifications, top-down parsing with $DCG_6$ may not terminate.

Regarding the last point, let us show that different definitions for $p$ and $q$ result in different computational behaviors:

[14] It can easily be shown that, iff this is the case, then the unit nonterminals can be completely eliminated, as in the case of example 3 above.

**Situation 1** Assume that $p, q$ are defined through the following clauses:

$$p(nil).$$
$$q(f(X), X).$$

In such a situation, top-down parsing with $DCG_6$ of the input string $oh$ does not terminate: an infinite number of solutions $(X = nil, X = f(nil), \ldots)$ are enumerated on backtracking and the program loops.[15]

**Situation 2** Assume that $p, q$ are defined by the following clauses:

$$p(X) \text{ :- } fail.$$
$$q(nil, nil).$$

The first clause defines $p$ as being the 'false' (omitting giving a clause for $p$ would have the same result). In such a situation, top-down parsing with $DCG_6$ terminates.

## 3 Conclusions:

• Few, if any, normal form results for DCGs (and for their close relatives, unification grammars) were previously known. The GGNF transformation can be applied to any DCG, whether offline-parsable or not.

• In the GGNF, the potential sources of undecidability of the parsing problem are factorized in the unit subgrammar, a grammar "over" the empty string [ ]. The GGNF as a whole is offline-parsable exactly when its unit subgrammar is. This is the case iff the unit subgrammar does not contain a nonterminal calling itself recursively.

• The GGNF seems to provide the closest analogue to the GNF that one can hope to find for DCGs.[16]

• If the DCG (or equivalently its GGNF) is offline-parsable then top-down parsing with the GGNF finds all solutions to the parsing problem and terminates.

• The transformation under GGNF can be specialized to the simpler case of a context-free grammar. In this case, the GGNF provides a variant of the standard GNF preserving degrees of ambiguity.[17]

[15]This is not the worst possible case: here, at least, *all* solutions end up being enumerated on backtracking. This would not be the case with more complex definitions for $p$ and $q$.

[16]Justification for this claim is given in [3].

[17]For lack of space, this point was not discussed in the paper. Consider the context-free grammar $CFG$ (which is the skeleton of example 5):

$$a1 \rightarrow [oh]$$
$$a1 \rightarrow a1$$

The GNF for this grammar is the grammar:

$$a1 \rightarrow [oh]$$

The original grammar assigns an infinite degree of ambiguity to $[oh]$, while its GNF does not (in fact it is easy to show that a GNF can never be infinitely ambiguous). On the other hand, the GGNF of $CFG$ is:

$$a1 \rightarrow n1$$
$$\overline{\phantom{xxxx}}$$
$$u \rightarrow [\,]$$
$$u \rightarrow u$$
$$\overline{\phantom{xxxx}}$$
$$n1 \rightarrow [oh]\ u$$

and it can be verified that it preserves degrees of ambiguity. This difference, which may be considered minor in the case of CFGS, plays an important role in the transformation of DCGs.

## Appendix: Some indications on the transformation method

We can only give here some brief indications in the hope that the interested reader will be motivated to look into the full description given in [3]. We start with some comments on the GGNF in the CFG case and then move on to the case of definite grammar schemes.

**CFGs, Algebraic Systems, and the GGNF.** The most powerful transformation methods existing for context-free grammars are algebraic ("matrix based" [8]) ones relying on the concepts of *formal power series* and *algebraic systems* (see [5, 9]). Using such concepts, a context-free grammar such as:

$$\begin{array}{lcl} a_1 & \rightarrow & a_1 a_2 \mid a_2 \mid [\,] \\ a_2 & \rightarrow & [v] \end{array}$$

is reformulated into the algebraic system:

$$\begin{array}{lcl} a_1 & = & a_1 a_2 + a_2 + 1 \\ a_2 & = & [v] \end{array}$$

which represents a fixpoint equation in the variables (or "nonterminals") $a_1, a_2$ on a certain algebraic structure (a non-commutative semiring) of formal power series $\mathrm{N}_\infty \langle\langle V^* \rangle\rangle$, where $\mathrm{N}_\infty$ is the set of non-negative integers, extended to infinity. Informally, an element of $\mathrm{N}_\infty \langle\langle V^* \rangle\rangle$ represents a *language* on the vocabulary $V$ (such that $[v] \in V$), where each string in the language is associated with a number, finite or infinite, which can be interpreted as the degree of ambiguity of this string relative to the system (or, equivalently the corresponding CFG).

In the example at hand, it can be easily verified that the following assigments of formal power series to $a_1, a_2$:

$$\begin{array}{lcl} a_1 & = & 1 + 2[v] + 2[v]^2 + 2[v]^3 + \cdots \\ a_2 & = & [v] \end{array}$$

satisfy the system.[18] In terms of the corresponding CFG, this fact implies that (1) the empty string [ ] is recognized exactly once by the grammar, and that each of the strings $[v]$, $[v][v]$, $[v][v][v]$, ..., is recognized exactly twice by the grammar.

From the point of view of transformations, algebraic systems have certain important advantages over context-free grammars: (1) they make ambiguity degrees explicit, (2) they involve equations (rather than rewriting rules),

[18]Furthermore, in the case at hand, they represent the *unique* solution to this system.

where "equals can be replaced by equals", and (3) they possess a rich algebraic structure (addition, multiplication) which endows them with mathematical perspicuity and power.

There are some substantial differences between the transformation steps used to obtain the GGNF of an algebraic system and the standard ones used to obtain its GNF, the principal one lying in the necessity to preserve degrees of ambiguity at each step. In the GNF case, the initial step consists in first transforming the initial system into a *proper* system (a notion analoguous to that of proper CFG)—an operation which *does not preserve* degrees of ambiguity—and then performing the main transformation. For this reason, the transformation steps in the GGNF case must be formulated in a more global way, which, among other complications, involves the use of certain identities on regular languages.[19] However, there are also important similarities between the GNF and the GGNF transformations, among them the observation that the elementary algebraic system in the variable $a$ on the vocabulary $V = \{[v], [t]\}$:

$$a = a\,[v] + [t]$$

has the unique solution $a = [t]\ [v]^*$, an observation which can be much generalized, and which plays a central role in both cases.

**DCGs, Mixed Systems, and the GGNF.** In order to define the GGNF in the case of Definite Grammar Schemes (or, equivalently, DCGs), we have introduced so-called *mixed systems*, a generalization of algebraic systems capable of representing association of structures to strings. Without going into details, let's consider the following definite grammar scheme:

$$\begin{array}{rcl} a_1(x) & \rightarrow & a_1(y)\, a_1(z)\, \{p(x,y,z)\} \ \mid \\ & & \quad a_2(y)\, \{q(x,y)\} \ \mid \ [\ ]\, \{r(x)\} \\ a_2(x) & \rightarrow & [v]\, \{s(x)\} \end{array}$$

This scheme is reformulated as the mixed system:

$$\begin{array}{rcl} a_{1x} & = & a_{1y}\, a_{2z}\, p_x^{yz} \ + \ a_{2y}\, q_x^y \ + \ r_x \\ a_{2x} & = & [v]\, s_x \end{array}$$

In this system, the variables (or nonterminals) $a_1, a_2$ are seen as functions: $E \rightarrow \mathbf{B}\langle\langle V^* \rangle\rangle$ (where $\mathbf{B}$ is the set of booleans $\{0,1\}$) , that is, as functions mapping elements of a set $E$ (often taken to be a Herbrand universe), representing linguistic structures, into formal series of $\mathbf{B}\langle\langle V^* \rangle\rangle$, that is, into languages over $V$. This can be seen to correspond to the intuitive notion that a nonterminal "associates" structures to strings. As for $p, q, r, s$, they are seen respectively as fonctions from $E^3, E^2, E, E$ into $\mathbf{B} \subset \mathbf{B}\langle\langle V^* \rangle\rangle$, that is, as predicates of different arities over E. The system represents a fixpoint equation on the variables $a_1, a_2$, given the constants $[v], p, q, r, s$.[20]

Although mixed systems are defined on more complex structures than are algebraic systems, the transformation methods for algebraic systems generalize without difficulty to their case, and these methods form the mathematical basis of the results reported in this paper.

[19] Such as the identity $(e+f)^* = e^*(fe^*)^*$.

[20] For the interested reader, the given system expresses (using "conventions of summation" familiar in tensor algebra) the relations:

$$\begin{array}{rcl} a_1 & = & \lambda\, x\, (\sum_{y,z \in E} a_1(y) a_2(z) p(x,y,z) \\ & & \quad + \sum_{y \in E} a_2(y) q(x,y) + r(x)) \\ a_2 & = & \lambda\, x\, ([v] s(x)) \end{array}$$

## Acknowledgments

Thanks to C. Boitet, M. Boyer, F. Guenthner, F. Perrault, Y. Schabes, M. Simard, and G. van Noord for comments and suggestions.

## References

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1: Parsing. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[2] M. Dymetman and P. Isabelle. Reversible logic grammars for machine translation. In *Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, Pittsburgh, PA, June 1988. Carnegie Mellon University.

[3] M. Dymetman. *Transformations de grammaires logiques. Applications au problème de la réversibilité en Traduction Automatique.* Thèse d'Etat. Université de Grenoble, Grenoble, France, July 1992.

[4] A. Haas. A generalization of the offline-parsable grammars. In *Proceedings of the 27th Annual Meeting of the ACL*, 237–42, Vancouver, BC, Canada, June 1989.

[5] Harrison, M.A. 1978. *Introduction to Formal Language Theory.* Reading, MA: Addison-Wesley.

[6] Pereira, F.C.N. and D.H.D. Warren. 1980. Definite Clause Grammars for Language Analysis. *Artificial Intelligence:*13, 231-78.

[7] Pereira, F.C.N. and D.H.D. Warren. 1983. Parsing as Deduction. In *Proceedings of the 21th Annual Meeting of the ACL*, 137-44. Cambridge, MA, June.

[8] Rosencrantz D. J. 1967. Matrix equations and normal forms for context-free grammars. *JACM* 14:3, 501-07.

[9] Salomaa, A. and M. Soittola. 1978. *Automata Theoretic Aspects of Formal Power Series.* New York: Springer Verlag.

[10] S.M. Shieber. 1985. Using restriction to extend algorithms for complex feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the ACL*, 145-152. Chicago, IL, June.