Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars

Pierre Boullier
INRIA-Rocquencourt
Domaine de Voluceau
B.P. 105
78153 Le Chesnay Cedex, FRANCE
Pierre.Boullier@inria.fr

Abstract

The notion of mild context-sensitivity was formulated in an attempt to express the formal power which is both necessary and sufficient to define the syntax of natural languages. However, some linguistic phenomena such as Chinese numbers and German word scrambling lie beyond the realm of mildly context-sensitive formalisms. On the other hand, the class of range concatenation grammars provides added power w.r.t. mildly context-sensitive grammars while keeping a polynomial parse time behavior. In this report, we show that this increased power can be used to define the above-mentioned linguistic phenomena with a polynomial parse time of a very low degree.

1 Motivation

The notion of mild context-sensitivity originates in an attempt by [Joshi 85] to express the formal power needed to define the syntax of natural languages (NLs). We know that context-free grammars (CFGs) are not adequate to define NLs since some phenomena are beyond their power (see [Shieber 85]). Popular incarnations of mildly context-sensitive (MCS) formalisms are tree adjoining grammars (TAGs) [Vijay-Shanker 87] and linear context-free rewriting (LCFR) systems [Vijay-Shanker, Weir, and Joshi 87]. However, there are some linguistic phenomena which are known to lie beyond MCS formalisms. Chinese numbers have been studied in [Radzinski 91] where it is shown that the set of these numbers is not a LCFR language and that it appears also not to be MCS since it violates the constant growth property. Scrambling is a word-order phenomenon which also lies beyond LCFR systems (see [Becker, Rambow, and Niv 92]).

On the other hand, range concatenation grammar (RCG), presented in [Boullier 98a], is a syntactic formalism which is a variant of simple literal movement grammar (LMG), described in [Groenink 97], and which is also related to the framework of LFP developed by [Rounds 88]. In fact it may be considered to lie halfway between their respective *string* and *integer* versions; RCGs retain from the string version of LMGs or LFPs the notion of concatenation, applying it to ranges (couples of integers which denote occurrences of substrings in a source text) rather than strings, and from their integer version the ability to handle only (part of) the source text (this later feature being the key to tractability). RCGs can also be seen as definite clause grammars acting on a flat domain: its variables are bound to ranges. This formalism, which extends CFGs, aims at being a convincing challenger as a syntactic base for various tasks, especially in natural language processing. We have shown that the positive version of RCGs, as simple LMGs or integer indexing LFPs, exactly covers the class *PTIME* of languages recognizable in deterministic polynomial time. Since the composition operations of RCGs are not restricted to be linear and non-erasing, its languages (RCLs) are not semi-linear. Therefore, RCGs are *not* MCS and are more powerful than LCFR systems, while staying computationally tractable: its sentences can be parsed in polynomial time. However, this formalism shares with LCFR systems the fact that its derivations are CF (i.e. the choice of the operation performed at each step only depends on the object to be derived from). As in the CF case, its derived trees can be packed into polynomial sized parse forests. For a CFG, the components of a parse forest are nodes labeled by couples (A, ρ) where A is a nonterminal symbol and ρ is a range, while for an RCG, the labels have the form $(A, \vec{\rho})$ where $\vec{\rho}$ is a vector (list) of ranges. Besides its power and efficiency, this formalism possesses many other attractive proper-

ties. Let us emphasize in this introduction the fact that RCLs are closed under intersection and complementation[1], and, like CFGs, RCGs can act as syntactic backbones upon which decorations from other domains (probabilities, logical terms, feature structures) can be grafted.

The purpose of this paper is to study whether the extra power of RCGs (over LCFR systems) is sufficient to deal with Chinese numbers and German scrambling phenomena.

2 Range Concatenation Grammars

This section introduces the notion of RCG and presents some of its properties, more details appear in [Boullier 98a].

Definition 1 *A positive range concatenation grammar (PRCG)* $G = (N, T, V, P, S)$ *is a 5-tuple where* N *is a finite set of predicate names,* T *and* V *are finite, disjoint sets of terminal symbols and variable symbols respectively,* $S \in N$ *is the start predicate name, and* P *is a finite set of clauses*

$$\psi_0 \rightarrow \psi_1 \dots \psi_m$$

where $m \geq 0$ *and each of* $\psi_0, \psi_1, \dots, \psi_m$ *is a predicate of the form*

$$A(\alpha_1, \dots, \alpha_p)$$

where $p \geq 1$ *is its arity,* $A \in N$ *and each of* $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, *is an argument.*

Each occurrence of a predicate in the RHS of a clause is a predicate *call*, it is a predicate *definition* if it occurs in its LHS. Clauses which define predicate A are called A-clauses. This definition assigns a fixed arity to each predicate name. The arity of S, the start predicate name, is one. The *arity* k of a grammar (we have a k-PRCG), is the maximum arity of its predicates.

Lower case letters such as $a, b, c, \dots$ will denote terminal symbols, while late occurring upper case letters such as T, W, X, Y, Z will denote elements of V.

The language defined by a PRCG is based on the notion of *range*. For a given input string $w = a_1 \dots a_n$ a range is a couple (i, j), $0 \leq i \leq j \leq n$ of integers which denotes the occurrence of some substring $a_{i+1} \dots a_j$ in w. The number i is its *lower bound*, j is its *upper bound* and $j - i$ is its *size*. If $i = j$, we have an *empty* range. We will use several equivalent denotations for ranges: an explicit dotted notation like $w_1 \bullet w_2 \bullet w_3$ or, if w_2 extends from positions $i + 1$ through j, a tuple notation $\langle i..j \rangle_w$, or $\langle i..j \rangle$ when w is understood or of no importance. Of course, only consecutive ranges can be concatenated into new ranges. In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. An *instantiated clause* is a clause in which variables and arguments are consistently (w.r.t. the concatenation operation) replaced by ranges; its components are *instantiated predicates.*

For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \rightarrow B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is an instantiation of the clause $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.[2]

For a grammar G and a source text w, a *derive* relation, denoted by $\underset{G,w}{\Rightarrow}$, is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause.

Definition 2 *The language of a PRCG* $G = (N, T, V, P, S)$ *is the set*

$$\mathcal{L}(G) = \{w \mid S(\bullet w \bullet) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$$

An input string $w = a_1 \dots a_n$ is a sentence if and only if the empty string (of instantiated predicates) can be derived from $S(\langle 0..n \rangle)$, the instantiation of the start predicate on the whole source text.

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments, whose ranges can be arbitrarily scattered over the source text. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

[1] Since this closure properties can be reached without changing the structure (grammar) of the constituents (i.e. we can get the intersection of two grammars G_1 and G_2 without changing neither G_1 nor G_2), this allows for a form of modularity which may lead to the design of libraries of reusable grammatical components.

[2] Often, for a variable X, instead of saying *the range which is bound to* X *or denoted by* X, we will say, the range X, or even instead of *the string whose occurrence is denoted by the range which is bound to* X, we will say the string X.

Note that the order of RHS predicates in a clause is of no importance.

As an example of a PRCG, the following set of clauses describes the three-copy language $\{www \mid w \in \{a,b\}^*\}$ which is not a CFL and even lies beyond the formal power of TAGs.

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X,Y,Z) \\ A(aX,aY,aZ) & \rightarrow & A(X,Y,Z) \\ A(bX,bY,bZ) & \rightarrow & A(X,Y,Z) \\ A(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

Definition 3 *A negative range concatenation grammar (NRCG) $G = (N,T,V,P,S)$ is a 5-tuple, like a PRCG, except that some predicates occurring in RHS, have the form $\overline{A(\alpha_1,\ldots,\alpha_p)}$.*

A predicate call of the form $\overline{A(\alpha_1,\ldots,\alpha_p)}$ is said to be a *negative predicate call.* The intuitive meaning is that an instantiated negative predicate succeeds if and only if its positive counterpart (always) fails. The idea is that the language defined by $\overline{A(\alpha_1,\ldots,\alpha_p)}$ is the complementary w.r.t T^* of the language defined by $A(\alpha_1,\ldots,\alpha_p)$. More formally, the couple $\overline{A(\vec{\rho})} \underset{G,w}{\Rightarrow} \varepsilon$ is in the derive relation if and only if $\not\exists A(\vec{\rho}) \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon$. Therefore this definition is based on a "negation by failure" rule. However, in order to avoid inconsistencies occurring when an instantiated predicate is defined in terms of its negative counterpart, we prohibit derivations exhibiting this possibility.[3] Thus we only define sentences by so called *consistent* derivations. We say that a grammar is consistent if all its derivations are consistent.

Definition 4 *A range concatenation grammar (RCG) is a PRCG or a NRCG.*

The PRCG (resp. NRCG) term will be used to underline the absence (resp. presence) of negative predicate calls.

[3]As an example, consider the NRCG G with two clauses $S(X) \rightarrow \overline{S(X)}$ and $S(\varepsilon) \rightarrow \varepsilon$ and the source text $w = a$. Let us consider the sequence $S(\bullet a \bullet) \underset{G,w}{\Rightarrow} \overline{S(\bullet a \bullet)} \underset{G,w}{\Rightarrow} \varepsilon$. If, on the one hand, we consider this sequence as a (valid) derivation, this shows, by definition, that a is a sentence, and thus $(\overline{S(\bullet a \bullet)}, \varepsilon) \notin \underset{G,w}{\Rightarrow}$. This last result is in contradiction with our hypothesis. On the other hand, if this sequence is not a (valid) derivation, and since the second clause cannot produce a (valid) derivation for $S(\bullet a \bullet)$ either, we can conclude that we have $\overline{S(\bullet a \bullet)} \underset{G,w}{\Rightarrow} \varepsilon$. Since, by the first clause, for any binding ρ of X we have $S(\rho) \underset{G,w}{\Rightarrow} \overline{S(\rho)}$, we conclude that, in contradiction with our hypothesis, the initial sequence is a derivation.

In [Boullier 98a], we presented a parsing algorithm which, for an RCG G and an input string of length n, produces a parse forest in time polynomial with n and linear with $|G|$. The degree of this polynomial is at most the maximum number of free (independent) bounds in a clause. Intuitively, if we consider an instantiation of a clause, all its terminal symbols, variable, arguments are bound to ranges. This means that each position (bound) in its arguments is mapped onto a *source index*, a position in the source text. However, at some times, the knowledge of a basic subset of couples (bound, source index) is sufficient to deduce the full mapping.[4] We call *number of free bounds*, the minimum cardinality of such a basic subset.

In the sequel we will assume that the predicate names *len*, and *eq* are defined:[5]

- $len(l,X)$ checks that the size of the range denoted by the variable X is the integer l, and
- $eq(X,Y)$ checks that the substrings selected by the ranges X and Y are equal.

3 Chinese Numbers & RCGs

The number-name system of Chinese, specifically the Mandarin dialect, allows large number names to be constructed in the following way. The name for 10^{12} is *zhao* and the word for five is *wu*. The sequence *wu zhao zhao wu zhao* is a well-formed Chinese number name (i.e. $5\ 10^{24} + 5\ 10^{12}$) although *wu zhao wu zhao zhao* is not: the number

[4]If XaY is some argument, if $X \bullet aY$ denotes a position in this argument, and if $(X \bullet aY, i)$ is an element of the mapping, we know that $(Xa \bullet Y, i+1)$ must be another element. Moreover, if we know that the size of the range X is 3 and that the sizes of the ranges X and Y are (always) equal (see for example the subsequent predicates *len* and *eq*), we can conclude that $(\bullet XaY, i-3)$ and $(XaY \bullet, i+4)$ are also elements of the mapping.

[5]The current implementation of our prototype system predefines several predicate names including *len*, and *eq*. It must be noted that these predefined predicates do not increase the formal power of RCGs since each of them can be defined by a pure RCG. For example, $len(1,X)$ can be defined by $len_1(t) \rightarrow \varepsilon$ which is a clause schema over all terminals $t \in T$. Their introduction is not only justified by the fact that they are more efficiently implemented than their RCG defined counterpart but mainly because they convey some static information about the length of their arguments which can be used, as already noted, to decrease the number of free bounds and thus lead to an improved parse time. In particular, the parse times for Chinese numbers, MIX, and German scrambling which are given in the next sections rely upon this statement.

of consecutive *zhao*'s must strictly decrease from left to right. All the well-formed number names composed only of instances of *wu* and *zhao* form the set

$$\{wu\ zhao^{k_1}\ wu\ zhao^{k_2} \ldots wu\ zhao^{k_p} \mid k_1 > k_2 > \ldots > k_p > 0\}$$

which can be abstracted as

$$\mathrm{CN} = \{ab^{k_1}ab^{k_2}\ldots ab^{k_p} \mid k_1 > k_2 > \ldots > k_p > 0\}$$

These numbers have been studied in [Radzinski 91], where it is shown that CN is not a LCFR language but an Indexed Language (IL) [Aho 68]. Radzinski also argued that CN also appears not to be MCS and moreover he says that he fails "*to find a well-studied and attractive formalism that would seem to generate Numeric Chinese without generating the entire class of ILs (or some non-ILs)*".

We will show that CN is defined by the RCG in Figure 1.

$$\begin{array}{lrcl}
1: & S(aX) & \rightarrow & A(X, aX, X) \\
2: & A(W, TX, bY) & \rightarrow & len(1, T)\ A(W, X, Y) \\
3: & A(WaY, X, aY) & \rightarrow & \overline{len(0, X)}\ A(Y, W, Y) \\
4: & A(W, X, \varepsilon) & \rightarrow & \overline{len(0, X)}\ \overline{len(0, W)}
\end{array}$$

Figure 1: RCG of Chinese numbers.

Let's call b^{k_i} the i^{th} slice. The core of this RCG is the predicate A of arity three. The string denoted by its third argument has always the form $b^{k_i-l}ab^{k_{i+1}}\ldots$, it is a suffix of the source text, its prefix $ab^{k_1}\ldots ab^{k_{i-1}}ab^{l}$ has already been examined. The property of the second argument is to have a size which is strictly greater than $k_i - l$, the number of leading b's in the current slice still to be processed. The leading b's of the third argument and the leading terminal symbols of the second argument are simultaneously scanned (and skipped) by the second clause, until either the next slice is introduced (by an a) in the third clause, or the whole source text is exhausted in the fourth clause. When the processing of a slice is completed, we must check that the size of the second argument is not null (i.e. that $k_{i-1} > k_i$). This is performed by the negative calls $\overline{len(0, X)}$ in the third and fourth clause. However, doing that, the i^{th} slice has been skipped, but, in order for the process to continue, this slice must be "rebuilt" since it will be used as second argument to process the next slice. This reconstruction process is performed with the help of the first argument. At the beginning of the processing of a new slice, say the i^{th}, both the first and third argument denote the same string $b^{k_i}ab^{k_{i+1}}\ldots$. The first argument will stay unchanged while the leading b's of the third argument are processed (see the second clause). When the processing of the i^{th} slice is completed, and if it is not the last one (case of the third clause), the first and third argument respectively denote the strings $b^{k_i}ab^{k_{i+1}}\ldots$ and $ab^{k_{i+1}}\ldots$. Thus, the i^{th} slice b^{k_i} can be extracted "by difference", it is the string W if the first and third argument are respectively WaY and aY (see the third clause). Last, the whole process is initialized by the first clause. The first and third argument of A are equal, since we start a new slice, the size of the second argument is forced to be strictly greater than the third, doing that, we are sure that it is strictly greater than k_1, the size of the first slice. Remark that the test $\overline{len(0, W)}$ in the fourth clause checks that the size k_p of the rightmost slice is not null, as stipulated in the language formal definition. The derivation for the sentence *abbbab* is shown in Figure 2 where $\stackrel{p}{\Rightarrow}$ means that clause $\#p$ has been applied.

$$\begin{array}{l}
S(\bullet abbbab\bullet) \\
\quad \stackrel{1}{\Rightarrow}\ A(a \bullet bbbab\bullet, \bullet abbbab\bullet, a \bullet bbbab\bullet) \\
\quad \stackrel{2}{\Rightarrow}\ A(a \bullet bbbab\bullet, a \bullet bbbab\bullet, ab \bullet bbab\bullet) \\
\quad \stackrel{2}{\Rightarrow}\ A(a \bullet bbbab\bullet, ab \bullet bbab\bullet, abb \bullet bab\bullet) \\
\quad \stackrel{2}{\Rightarrow}\ A(a \bullet bbbab\bullet, abb \bullet bab\bullet, abbb \bullet ab\bullet) \\
\quad \stackrel{3}{\Rightarrow}\ A(abbba \bullet b\bullet, a \bullet bbb \bullet ab, abbba \bullet b\bullet) \\
\quad \stackrel{2}{\Rightarrow}\ A(abbba \bullet b\bullet, ab \bullet bb \bullet ab, abbbab \bullet \bullet) \\
\quad \stackrel{4}{\Rightarrow}\ \varepsilon
\end{array}$$

Figure 2: Derivation for the CN string *abbbab*.

If we look at this grammar, for any input string of length n, we can see that the maximum number of steps in any derivation is $n+1$ (this number is an upper limit which is only reached for sentences). Since, at each step the choice of the A-clause to apply is performed in constant time (three clauses to try), the overall parse time behavior is linear.

Therefore, we have shown that Chinese numbers can be parsed in linear time by an RCG.

4 MIX & RCGs

Originally described by Emmon Bach, the MIX language consists of strings in $\{a, b, c\}^*$ such that each string contains the same number of occurrences of each letter. MIX is interesting because it has a very simple and intuitive characterization. However, Gazdar reported[6] that MIX may well be outside the class of ILs (as conjectured by Bill Marsh in an unpublished 1985 ASL paper). It has turned out to be a very difficult problem. In [Joshi, Vijay-Shanker, and Weir 91] the authors have shown that MIX can be defined by a variant of TAGs with local dominance and linear precedence (TAG(LD/LP)), but very little is known about this class of grammars, except that, as TAGs, they continue to satisfy the constant growth property. Below, we will show that MIX is an RCL which can be recognized in linear time.

$$
\begin{array}{lllll}
1: & S(X) & \rightarrow & M(X,X,X) \\
2: & M(aX,bY,cZ) & \rightarrow & M(X,Y,Z) \\
3: & M(TX,Y,Z) & \rightarrow & len(1,T)\ \overline{a(T)} \\
 & & & \quad M(X,Y,Z) \\
4: & M(X,TY,Z) & \rightarrow & len(1,T)\ \overline{b(T)} \\
 & & & \quad M(X,Y,Z) \\
5: & M(X,Y,TZ) & \rightarrow & len(1,T)\ \overline{c(T)} \\
 & & & \quad M(X,Y,Z) \\
6: & M(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \\
7: & a(a) & \rightarrow & \varepsilon \\
8: & b(b) & \rightarrow & \varepsilon \\
9: & c(c) & \rightarrow & \varepsilon
\end{array}
$$

Figure 3: RCG of MIX.

Consider the RCG in Figure 3. The source text is concurrently scanned three times by the three arguments of the predicate M (see the predicate call $M(X,X,X)$ in the first clause). The first, second and third argument of M respectively only deal with the letters a, b and c. If the leading letter of any argument (which at any time is a suffix of the source text) is not the right letter, this letter is skipped. The third clause only process the first argument of M (the two others are passed unchanged), and skips any letter which is not an a. The analogous holds for the fourth and fifth clauses which respectively only consider the second and third argument of M, looking for a leading b or c. Note that the knowledge that a letter is not the right one is acquired via a negative predicate call because this allows for an easy generalization to any number of letters. In the case where the three leading letters are respectively a, b and c, they are simultaneously skipped (see clause #2) and the clause #6 is eventually instantiated if and only if the input string contains the same number of occurrences of each letter.

The leading steps in the derivation for the sentence *baccba* are shown in Figure 4 where $\stackrel{p}{\Rightarrow}$ means that clause #p is applied and $\stackrel{\overline{q}}{\Rightarrow}$ means that clause #q cannot be applied, and thus implies the validation of the corresponding negative predicate call.

$$
\begin{array}{ll}
 & S(\bullet baccba\bullet) \\
\stackrel{1}{\Rightarrow} & M(\bullet baccba\bullet, \bullet baccba\bullet, \bullet baccba\bullet) \\
\stackrel{3}{\Rightarrow} & \overline{a(\bullet b \bullet accba)} \\
 & \quad M(b \bullet accba\bullet, \bullet baccba\bullet, \bullet baccba\bullet) \\
\stackrel{\overline{7}}{\Rightarrow} & M(b \bullet accba\bullet, \bullet baccba\bullet, \bullet baccba\bullet) \\
\stackrel{5}{\Rightarrow} & \overline{c(\bullet b \bullet accba)} \\
 & \quad M(b \bullet accba\bullet, \bullet baccba\bullet, b \bullet accba\bullet) \\
\stackrel{\overline{9}}{\Rightarrow} & M(b \bullet accba\bullet, \bullet baccba\bullet, b \bullet accba\bullet) \\
\stackrel{5}{\Rightarrow} & \overline{c(b \bullet a \bullet accba)} \\
 & \quad M(b \bullet accba\bullet, \bullet baccba\bullet, ba \bullet ccba\bullet) \\
\stackrel{\overline{9}}{\Rightarrow} & M(b \bullet accba\bullet, \bullet baccba\bullet, ba \bullet ccba\bullet) \\
\stackrel{2}{\Rightarrow} & M(ba \bullet ccba\bullet, b \bullet accba\bullet, bac \bullet cba\bullet) \\
\stackrel{+}{\Rightarrow} & \varepsilon
\end{array}
$$

Figure 4: Derivation for the MIX string *baccba*.

It is not difficult to see that the length of any derivation is linear in the length of the corresponding input string, and that the choice of any step in this derivation takes a constant time. Therefore, the parse time complexity of this grammar is linear.

Of course, we can think of several generalizations of MIX. We let the reader devise an RCG in which the relation between the number of occurrences of each letter is not the equality, instead, we will study here the case where, on the one hand, the number of letters in T is not limited to three, and, on the other hand, all the letters in T do not necessarily appear in a sentence. If $T = \{b_1, \ldots, b_q\}$ is its terminal vocabulary, and if π is a permutation, the *permutation* language $\Pi = \{w \mid w = \pi(a_1^k \ldots a_i^k \ldots a_p^k)\}$, with $a_i \in T$, $0 \leq p \leq q$ and $i \neq j \Longrightarrow a_i \neq a_j$, can be defined by the set of clauses in Figure 5.

[6] See http://www.ccl.kuleuven.ac.be/LKR/dtr/mix1.dtr.

$$
\begin{array}{lcl}
S(\varepsilon) & \rightarrow & \varepsilon \\
S(TX) & \rightarrow & \mathit{len}(1,T) \\
 & & \quad A(T,TX,TX) \\
A(T,W,T_1X) & \rightarrow & \mathit{len}(1,T_1) \\
 & & \quad M_4(T,W,T_1,W) \\
 & & \quad A(T,W,X) \\
A(T,W,\varepsilon) & \rightarrow & \varepsilon \\
M_4(T,T'X,T_1,T_1'Y) & \rightarrow & \mathit{eq}(T,T')\ \mathit{eq}(T_1,T_1') \\
 & & \quad M_4(T,X,T_1,Y) \\
M_4(T,T'X,T_1,Y) & \rightarrow & \mathit{len}(1,T')\ \overline{\mathit{eq}(T,T')} \\
 & & \quad M_4(T,X,T_1,Y) \\
M_4(T,X,T_1,T_1'Y) & \rightarrow & \mathit{len}(1,T_1')\ \overline{\mathit{eq}(T_1,T_1')} \\
 & & \quad M_4(T,X,T_1,Y) \\
M_4(T,\varepsilon,T_1,\varepsilon) & \rightarrow & \varepsilon
\end{array}
$$

Figure 5: RCG of the permutation language Π.

The basic idea of this grammar is the following. In a source text $w = t_1 \ldots t_m \ldots t_n$, we choose a *reference position* $r, 1 \leq r \leq n$ (for example, if $r = 1$, we choose the first position which corresponds to the leading letter t_1), and a *current position* $c, 1 \leq c \leq n$, and we check that the number of occurrences of the *current terminal* t_c, and the number of occurrences of the *reference terminal* t_r are equal. Of course, if this check succeeds for all the current positions c and for one reference position r, the string w is in Π. This check is performed by the predicate $M_4(T_1, X, T_2, Y)$ of arity four. Its first and third arguments respectively denote the reference position and the current position (T_1 and T_2 are bound to ranges of size one which refer to t_r and t_c respectively) while the second and fourth arguments denote the strings in which the searches are performed: the occurrences of the reference terminal t_r are searched in X and the occurrences of the current terminal t_c are searched in Y. A call to M_4 succeeds if and only if the number of occurrences of t_r in X is equal to the number of occurrences of t_c in Y. The S-clauses select the reference position ($r = 1$, if w is not empty). The purpose of the A-clauses is to select all the current positions c and to call M_4 for each such c's. Note that the variable W is always bound to the whole source text. We can easily see that the complexity of any predicate call $M_4(T_1, X, T_2, Y)$ is linear in $|X| + |Y|$, and since the number of such calls from the third clause is n, we have a quadratic time RCG.

5 Scrambling & RCGs

Scrambling is a word-order phenomenon which occurs in several languages such as German, Japanese, Hindi, ... and which is known to be beyond the formal power of TAGs (see [Becker, Joshi, and Rambow 91]). In [Becker, Rambow, and Niv 92], the authors even show that LCFR systems cannot derive scrambling. This is of course also true for multi-components TAGs (see [Rambow 94]). In [Groenink 97], p. 171, the author said that "*simple LMG formalism does not seem to provide any method that can be immediately recognized as solving such problems*". We will show below that scrambling can be expressed within the RCG framework.

Scrambling can be seen as a leftward movement of arguments (nominal, prepositional or clausal). Groenink notices that similar phenomena also occur in Dutch verb clusters, where the order of verbs (as opposed to objects) can in some case be reversed.

In [Becker, Rambow, and Niv 92], from the following German example

... daß [dem Kunden]$_i$ [den Kuehlschrank]$_j$
... that the client (DAT) the refrigerator (ACC)

bisher noch niemand
so far yet no-one (NOM)

t_i [[t_j zu reparieren] zu versuchen]
to repair to try

versprochen hat.
promised has.

... that so far no-one has promised the client to try to repair the refrigerator.

the authors argued that scrambling may be "doubly unbounded" in the sense that:

- there is no bound on the distance over which each element can scramble;
- there is no bound on the number of unbounded dependencies that can occur in one sentence.

They used the language $\{\pi(n_1 \ldots n_m)\, v_1 \ldots v_m\}$ where π is a permutation, as a formal representation for a subset of scrambled German sentences, where it is assumed that each verb v_i has exactly one overt nominal argument n_i.

However, in [Becker, Joshi, and Rambow 91], we can find the following example

... daß [des Verbrechens]$_k$ [der Detektiv]$_i$
... that the crime (GEN) the detective (NOM)

[den Verdächtigen]$_j$ dem Klienten

the suspect (ACC) the client (DAT)

[PRO$_i$ t_j t_k zu überführen] versprochen hat.
to indict promised has.

... that the detective has promised the client to indict the suspect of the crime.

where the verb of the embedded clause subcategorizes for three NPs, one of which is an empty subject (PRO). Thus, the scrambling phenomenon can be abstracted by the language SCR = $\{\pi(n_1 \dots n_p)\ v_1 \dots v_q\}$. We assume that the set T of terminal symbols is partitioned into the noun part $\mathcal{N} = \{n_1, \dots, n_l\}$ and the verb part $\mathcal{V} = \{v_1, \dots, v_m\}$, and that there is a mapping h from $\mathcal{N}$ onto $\mathcal{V}$ which indicates, when $v = h(n)$, that the noun n is an argument for the verb v. If h is an injective mapping, we describe the case where each verb has exactly one overt nominal argument, if h is not injective, we describe the case where several nominal arguments can be attached to a single verb. To be a sentence of SCR, the string $\pi(n_1 \dots n_i \dots n_p)v_1 \dots v_j \dots v_q$ must be such that $0 \leq p \leq l$, $0 \leq q \leq m$, $n_i \in \mathcal{N}$, $v_j \in \mathcal{V}$, $i \neq i' \Longrightarrow n_i \neq n_{i'}$, $j \neq j' \Longrightarrow v_j \neq v_{j'}$, $\forall n_i \exists v_j$ and $\forall v_j \exists n_i$ s.t. $v_j = h(n_i)$, and π is a permutation. The RCG in Figure 6 defines SCR.

Of course, the predicate names $\mathcal{N}$, $\mathcal{V}$ and h respectively define the set of nouns $\mathcal{N}$, the set of verbs $\mathcal{V}$ and the mapping h between $\mathcal{N}$ and $\mathcal{V}$. The purpose of the predicate name $\mathcal{N}^+\mathcal{V}^+$ is to split any source text w in a prefix part which only contains nouns and a suffix part which only contains verbs. This is performed by a left-to-right scan of w during which nouns are skipped (see the first $\mathcal{N}^+\mathcal{V}^+$-clause). When the first verb is found, we check, by the call $\mathcal{V}^*(Y)$, that the remaining suffix Y only contains verbs. Then, the predicates $\mathcal{N}s$ and $\mathcal{V}s$ are both called with two identical arguments, the first one is the prefix part and the second is the suffix part. Note how the prefix part X can be extracted by the predicate definition $\mathcal{N}^+\mathcal{V}^+(XTY, TY)$ from the first argument (which denotes the whole source text) in using the second argument TY. The predicate name $\mathcal{N}s$ (resp. $\mathcal{V}s$) is in charge to check that each noun n_i of the prefix part (resp. each verb v_j of the suffix part) has both a single occurrence in its own part, and that there is a verb v_j in the suffix part (resp. a noun n_i in the prefix part) such that $h(n_i, v_j)$ is true. The prefix part is examined from left-to-right until completion by the $\mathcal{N}s$-clauses. For each noun T in this prefix part, the single occurrence test is performed by a negative calls to $TinT^*(T, X)$, and the existence of a verb v_j in the suffix part s.t. $h(T, v_j)$, is performed by the $\mathcal{N}in\mathcal{V}^+(T, Y)$ call. A call $TinT^*(T, X)$ is true if and only if the terminal symbol T occurs in X. The $\mathcal{N}in\mathcal{V}^+$-clauses spell from left-to-right the suffix part. If the noun T is not an argument of the verb T' (note the negative predicate call), this verb is skipped, until an h relation between T and T' is eventually found. Of course, an analogous processing is performed for each verb in the suffix part. We can easily see that, the cutting of each source text w in a prefix part and a suffix part, and the checking that the suffix part only contains verbs, takes a time linear in $|w|$. For each noun in the prefix part, the unique occurrence check takes a linear time and the check that there is a corresponding verb in the suffix part also takes a linear time. Of course, the same results hold for each verb in the suffix part. Thus, we can conclude that the scrambling phenomenon can be parsed in quadratic time.

$$
\begin{array}{ll}
S(W) & \rightarrow \mathcal{N}^+\mathcal{V}^+(W, W) \\
\mathcal{N}^+\mathcal{V}^+(W, TY) & \rightarrow len(1, T)\ \mathcal{N}(T) \\
 & \quad \mathcal{N}^+\mathcal{V}^+(W, Y) \\
\mathcal{N}^+\mathcal{V}^+(XTY, TY) & \rightarrow len(1, T)\ \mathcal{V}(T)\ \mathcal{V}^*(Y) \\
 & \quad \mathcal{N}s(X, TY)\ \mathcal{V}s(X, TY) \\
\mathcal{N}s(TX, Y) & \rightarrow len(1, T)\ \overline{TinT^*(T, X)} \\
 & \quad \mathcal{N}in\mathcal{V}^+(T, Y)\ \mathcal{N}s(X, Y) \\
\mathcal{N}s(\varepsilon, Y) & \rightarrow \varepsilon \\
\mathcal{N}in\mathcal{V}^+(T, T'Y) & \rightarrow len(1, T')\ \overline{h(T, T')} \\
 & \quad \mathcal{N}in\mathcal{V}^+(T, Y) \\
\mathcal{N}in\mathcal{V}^+(T, T'Y) & \rightarrow len(1, T')\ h(T, T') \\
\mathcal{V}s(X, TY) & \rightarrow len(1, T)\ \overline{TinT^*(T, Y)} \\
 & \quad \mathcal{V}in\mathcal{N}^+(T, X)\ \mathcal{V}s(X, Y) \\
\mathcal{V}s(X, \varepsilon) & \rightarrow \varepsilon \\
\mathcal{V}in\mathcal{N}^+(T, T'Y) & \rightarrow len(1, T')\ \overline{h(T', T)} \\
 & \quad \mathcal{V}in\mathcal{N}^+(T, Y) \\
\mathcal{V}in\mathcal{N}^+(T, T'Y) & \rightarrow len(1, T')\ h(T', T) \\
TinT^*(T, T'Y) & \rightarrow len(1, T)\ \overline{eq(T, T')} \\
 & \quad TinT^*(T, Y) \\
TinT^*(T, T'Y) & \rightarrow len(1, T)\ eq(T, T') \\
\mathcal{V}^*(TX) & \rightarrow len(1, T)\ \mathcal{V}(T)\ \mathcal{V}^*(X) \\
\mathcal{V}^*(\varepsilon) & \rightarrow \varepsilon \\
\mathcal{N}(n_1) & \rightarrow \varepsilon \\
\dots & \\
\mathcal{N}(n_l) & \rightarrow \varepsilon \\
\mathcal{V}(v_1) & \rightarrow \varepsilon \\
\dots & \\
\mathcal{V}(v_m) & \rightarrow \varepsilon \\
h(n_1, v_1) & \rightarrow \varepsilon \\
\dots & \\
h(n_l, v_m) & \rightarrow \varepsilon
\end{array}
$$

Figure 6: RCG of scrambling.

6 Conclusion

The class of RCGs is a syntactic formalism which seems very promising since it has many interesting properties among which we can quote its power, above that of LCFR systems; its efficiency, with polynomial time parsing; its modularity; and the fact that the output of its parsers can be viewed as shared parse forests. It can thus be used as is to define languages or it can be used as an intermediate (high-level) representation. This last possibility comes from the fact that many popular formalisms can be translated into equivalent RCGs, without loosing any efficiency. For example, TAGs can be translated into equivalent RCGs which can be parsed in $O(n^6)$ time (see [Boullier 98b]).

In this paper, we have shown that this extra formal power can be used in NL processing. We turn our attention to the two phenomena of Chinese numbers and German scrambling which are both beyond the formal power of MCS formalisms. To our knowledge, Chinese numbers were only known to be an IL and it was not even known whether scrambling can be described by an IG. We have seen that these phenomena can both be defined by RCGs. Moreover, the corresponding parse time is polynomial with a very low degree. During this work we have also classified the famous MIX language, as a linear parse time RCL.

References

[Aho 68] Alfred Aho. 1968. Indexed grammars – an extension of context-free grammars. In *Journal of the ACM*, Vol. 15, pages 647–671.

[Becker, Joshi, and Rambow 91] Tilman Becker, Aravind Joshi, and Owen Rambow. 1991. Long distance scrambling and tree adjoining grammars. In *Proceedings of the fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26.

[Becker, Rambow, and Niv 92] Tilman Becker, Owen Rambow, and Michael Niv. 1992. The Derivational Generative Power of Formal Systems or Scrambling is Beyond LCFRS. In *Technical Report IRCS-92-38*, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA.

[Boullier 98a] Pierre Boullier. 1998. Proposal for a Natural Language Processing Syntactic Backbone. In *Research Report No 3342* at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA-Rocquencourt, France, Jan. 1998, 41 pages.

[Boullier 98b] Pierre Boullier. 1998. A Generalization of Mildly Context-Sensitive Formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, University of Pennsylvania, Philadelphia, PA, pages 17–20.

[Groenink 97] Annius Groenink. 1997. SURFACE WITHOUT STRUCTURE Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University, The Netherlands, Nov. 1977, 250 pages.

[Joshi 85] Aravind Joshi. 1985. How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In *Natural Language Processing — Theoretical, Computational and Psychological Perspective*, D. Dowty, L. Karttunen, and A. Zwicky, editors, Cambridge University Press, New-York, NY.

[Joshi, Vijay-Shanker, and Weir 91] Aravind Joshi, K. Vijay-Shanker, and David Weir. 1991. The convergence of mildly context-sensitive grammatical formalisms. In *Foundational Issues in Natural Language Processing*, P. Sells, S. Shieber, and T. Wasow editors, MIT Press, Cambridge, Mass.

[Radzinski 91] Daniel Radzinski. 1991. Chinese Number-Names, Tree Adjoining Languages, and Mild Context-Sensitivity. In *Computational Linguistics*, 17(3), pages 277–299.

[Rambow 94] Owen Rambow. 1994. Formal and Computational Aspects of Natural Language Syntax. In *PhD Thesis*, University of Pennsylvania, Philadelphia, PA.

[Rounds 88] William Rounds. 1988. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. In *ACL Computational Linguistics*, Vol. 14(4), pages 1–9.

[Shieber 85] Stuart Shieber. 1985. Evidence against the context-freeness of natural language. In *Linguistics and Philosophy*, Vol. 8, pages 333–343.

[Vijay-Shanker 87] K. Vijay-Shanker. 1987. A study of tree adjoining grammars. *PhD thesis*, University of Pennsylvania, Philadelphia, PA.

[Vijay-Shanker, Weir, and Joshi 87] K. Vijay-Shanker, David Weir, and Aravind Joshi. 1987. Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL'87)*, Stanford University, CA, pages 104–111.