Linguistic facts as predicates over ranges of the sentence

Benoît Sagot

INRIA-Rocquencourt, Projet Atoll, Domaine de Voluceau, Rocquencourt B.P. 105
78 153 Le Chesnay Cedex, France

Abstract. This paper introduces a novel approach to language processing, in which linguistic facts are represented as predicates over ranges of the intput text, usually, but not limited to, ranges of the current sentence. Such an approch allows to build non-linear analyses with a polynomial parsing complexity that take into account simultaneously and with the same technical status morphological, syntactical and semantical properties, this list being non limitative. Classical analyses, such as constituency trees, dependency graphs, topological boxes and predicate-arguments semantics are then obtained as partial projection of a complete analysis. The formalism presented here is based upon Range Concatenation Grammars (hereafter RCG), and has been successfully implemented, thanks to a previously existing RCG parser and a syntactico-semantical grammar for French.

1 Introduction

The definition of an adequate formalism for natural language processing consists in the search of an optimal balance between linguistic validity and computational efficiency. Moreover, a newly defined formalism can be considered interesting only if it is really implemented, with a complete parser and a large-coverage grammar for an example language, and if it shows interesting properties that are not present in other formalisms, or at a more expensive computational cost.

In this paper, we introduce a new formalism that, we think, has all the above-mentioned properties. This formalism relies on the hypothesis that linguistic properties are best described as predicates over continuous ranges of the input sentence[1] or of a bounded amount of extra tokens (e.g. contextual syntagms[2]). Range Concatenation Grammars (hereafter RCGs), introduced by Boullier, provide a appropriate basis to develop such a formalism. In the remainder, we will present RCGs and their most important properties. Then we will introduce our formalism, called *Meta-RCG*, whose grammars can be converted into RCGs. Finally we will present a fragment of our grammar for French through

[1] In itself this idea is not new, and has been used for example in Datalog grammars [1] or in the constraint logic programming implementations of Property Grammars [2]. However, as made clear later, our formalism relies also on three fundamental properties of RCGs: range concatenation, non-linearity and parsing time polynomiality.

[2] This possibility, already implemented, will not be treated in this paper.

two examples, in order to show how morphological, syntactical and semantical properties interact continuously with each other, leading to global analyses from which classical analyses (constituency tree, dependency graph, topological boxes, predicate-arguments semantics) can be extracted.

It is worth saying that one of the most important motivations for this work is to develop a formalism in which the two following major constraints are satisfied:

– Morphology, syntax, and, more originally, lexical semantics (and if possible more general semantics) are dealt with *simultaneously* during parsing; in particular, there must be no artificial boundary between a syntactic backbone, syntactic features (or decorations) and lexical semantics.
– Parsing must be computationally tractable, and we make the hypothesis that we can assume a polynomial parsing time.

We discuss these hypotheses more deeply at the beginning of section 3.

2 Range Concatenation Grammars

Range Concatenation Grammars (RCGs) have been introduced by Boullier (see for example [3], and applications thereof in [4]). They defined a class of languages, Range Concatenation Langages (RCLs), that covers exactly *PTIME*, the class of all languages recognizable in deterministic polynomial time. Therefore, RCGs are more powerful than all Mildly Context-Sensitive formalisms, such as Linear Context-Free Rewriting Systems (LCFRS) or Multi-Component Tree-Adjoining Grammars (MC-TAGs), while remaining computationally tractable. We shall now define formally RCGs, following [3].

2.1 Positive RCGs

A *positive RCG* is a 5-tuple $G = (N, T, V, P, S)$ in which:

– N is a finite set of *predicate names*,
– T is a finite set of *terminal symbols*,
– V is a finite set of *variable symbols* such as $T \cap V = \emptyset$,
– $\mathtt{S} \in N$ is the *axiom*,
– P is a finite set of *clauses*, which are defined hereafter.

A clause C has the form $\psi_0 \rightarrow \psi_1 \ldots \psi_j \ldots \psi_m$, where $m \geq 0$ and each ψ_j is a *predicate* of the form $\mathtt{A(}\alpha_1, \ldots, \alpha_i, \ldots, \alpha_p\mathtt{)}$, where $p \geq 1$ is its *arity*, $\mathtt{A} \in N$, and each α_i is an *argument* of $\mathtt{A}$. Each argument α_i of $\mathtt{A}$ has the form $\mathtt{X}_1 \ldots \mathtt{X}_l \ldots \mathtt{X}_q$, where each $\mathtt{X}_l$ is in $V \cup T$. The *left-hand part* of C is ψ_0, its *right-hand part* is $\psi_1 \ldots \psi_j \ldots \psi_m$. The predicates of the right-hand side form a set of predicates, which means that order does not matter and that duplicating a predicate is useless. In the following, we will abusively denote a predicate by its name, thus speaking of the *predicate A*. The arity of the axiom has to be 1, and the arity of a predicate $\mathtt{A}$ is fixed. We call $\mathtt{A}$-clause a clause whose left-hand side predicate is $\mathtt{A}$.

We define the arity of an `A`-clause by the arity of `A`, and the arity of a grammar by the maximal arity of its predicates. An RCG of arity k is a k-RCG.

As said before, the definition of a language by an RCG relies on the notion of range of the input string. Given a string $w = a_1 \dots a_n$ of terminal symbols ($w \in T^*$), each pair of integers (i, j) such as $0 \leq i \leq j \leq n$ is called a *range* of w and is denoted $\langle i..j \rangle_w$ or $i..j$ if w can be omitted, i and j being respectively the *lower* and *upper bound* of the range, and $j - i$ its size. If $i = j$, the range is *empty*. Two ranges are equal if and only if their lower and upper bounds are respectively equal[3]. A range $\langle i..j \rangle_w$ corresponds to a substring of w, namely $a_{i+1} \dots a_j$. The concatenation of two ranges $i..j$ and $k..l$ is defined if and only if $j = k$ and is in this case the range $i..l$.

Variable symbols and terminal symbols denote ranges. A terminal symbol `t` denotes a range of length 1 corresponding to a substring of w that is the symbol t. The *concatenation* `XY` of two variable or terminal symbols `X` and `Y` (`X`,`Y`$\in V \cup T$) denotes the range resulting from the concatenation of the ranges denoted respectively by `X` and `Y`, and is therefore defined if and only if these ranges can be concatenated. We often (abusively) denote by *the range `X`*, where `X` is a variable symbol, the range that is denoted by `X`.

Given $w \in T^*$, we call *w-instantiated predicate* or simply *instantiated predicate* a predicate in which all variable and terminal symbols have been replaced by ranges of w, and all ranges of the same argument of the same predicate have then been replaced by the range resulting from their concatenation. If this is possible, the predicate is said to be *instantiable* by w. For example, if the length of w is 3 ($w = a_1 a_2 a_3$), a possible instantiation for the predicate $\mathtt{A(XY,a_3,a_2Y)}$ is $\mathtt{A}(0..3, 2..3, 1..3)$. We define in the same way *instantiated clauses.* The arguments of a predicate (but not different variables of the same argument) can of course be replaced by discontinuous, or even overlapping ranges, since the same variable can occur in several arguments of several predicates of a clause. This is denoted as the *non-linerarity* of RCGs, and allows to express different points of view or properties on a given range that interact with each other. As we will see below, this is a major advantage of RCGs for linguistic use. More generally, it is because of this non-linearity that RCGs have the necessary expressive power to cover all *PTIME*.

For a given positive RCG G and an input string w, a binary *derive* relation, denoted by $\underset{G,w}{\Longrightarrow}$, and having as operands sets of instantiated predicates, by the following. Let $\Gamma_1 \ \gamma \ \Gamma_2$ be an instantiated right-hand side of some clause (and thus a set of predicates, as said before), where γ is also the left-hand side of the instantiated clause $\gamma \rightarrow \Gamma$. We have then $\Gamma_1 \ \gamma \ \Gamma_2 \underset{G,w}{\Longrightarrow} \Gamma_1 \ \Gamma \ \Gamma_2$.

A string $w \in T^*$ of length n is recognized by the grammar G if and only an empty list of predicates can be derived from the instantiated predicate $\mathtt{S}(0..n)$ (`S` being the axiom), i.e. if and only if $\mathtt{S}(0..n) \overset{+}{\underset{G,w}{\Longrightarrow}} \varepsilon$, the binary relation $\overset{+}{\underset{G,w}{\Longrightarrow}}$ being the transitive closure of $\underset{G,w}{\Longrightarrow}$.

[3] Therefore, if $i_1 \neq i_2$, the ranges $i_1..i_1$ and $i_2..i_2$, while both empty, are not equal.

For example, let us consider the non semi-linear language $L = \{a^{2^p} \mid p \geq 0\}$. The following positive RCG recognizes this language:

```
S(XY)       → S(X) EQ(X,Y)
S(a)        → ε
EQ(aX,aY)   → EQ(X,Y)
EQ(ε,ε)     → ε
```

Indeed, this grammar recognizes the input string a thanks to the second clause. And an input string consisting of 2^p times a ($p \geq 1$) is decomposed by the first clause in two ranges that must have the same length (predicate EQ[4]) and denote a (same) substring that has to be in L as well (of length 2^{p-1}).

2.2 Negative RCGs

Positive RCGs cover *PTIME.* Therefore, the set of languages that can be recognized by a positive RCG is closed under complementation. For this reason, it is only for practical reasons, and not to increase the expressing power of the formalism, that negative predicate can be introduced.

Indeed, we call *negative predicate* a predicate marked as such, either by a bar over the predicate ($\overline{\texttt{A(...)}}$), or by the symbol `!` in front of it (`!A(...)`), the intended meaning being based on "negation by failure": the empty list of instantiated predicates can be derived from an instantiated negative predicate if and only if it can not be derived from its positive counterpart.

We call negative RCG an RCG that has at least one clause containing in its right-hand part at least one negative predicate. A negative RCG is consistent if, for any $w \in T^*$, there is no w-instantiated predicate `A(...)` such that the empty list of predicates can be derived from both `A(...)` and `!A(...)`.

2.3 Closure properties

It can be shown easily [3] that RCLs are closed under union, concatenation, Kleene iteration, and, more interestingly, intersection and complementation. The grammars recognizing the operand languages need not be modified. One or two more suffices. In a rather informal way, and with $\texttt{S}_1$ and $\texttt{S}_2$ being the axioms of RCGs recognizing respectively the languages L_1 and L_2, we can get the closures by adding the following clauses, S being the axiom of the resulting language:

Union	$\texttt{S(X)} \rightarrow \texttt{S}_1\texttt{(X)}$ $\texttt{S(X)} \rightarrow \texttt{S}_2\texttt{(X)}$
Concatenation	$\texttt{S(XY)} \rightarrow \texttt{S}_1\texttt{(X) S}_2\texttt{(Y)}$
Intersection	$\texttt{S(X)} \rightarrow \texttt{S}_1\texttt{(X) S}_2\texttt{(X)}$
Kleene iteration	$\texttt{S(}\varepsilon\texttt{)} \rightarrow \varepsilon$ $\texttt{S(XY)} \rightarrow \texttt{S}_1\texttt{(X) S(Y)}$
Complementation	$\texttt{S(X)} \rightarrow \texttt{!S}_1\texttt{(X)}$

[4] The notation ε groups in a rather confusing way two different things, as visible in the last clause of the grammar given as example: it can denote either an empty string, i.e. an element of T^*, or an empty list (or set) of predicates.

2.4 Parsing

Range Concatenation Languages can be recognized and analysed in polynomial time. More precisely, let $|G|$ be the *size* of the k-RCG G, defined as the sum of the number of right-hand side predicates over all its clauses, and l the maximum number of right-hand side predicates in the longest clause. In [3], Boullier gives an algorithm that is $O(|G|n^{2k(1+l)})$ in time.

Moreover, Boullier has developed an efficient RCG-parser that has been already used to build TAG and MC-TAG parsers after an appropriate conversion step, with excellent efficiency results [7].

3 Introducing Meta-RCGs

Range Concatenation Grammars are a powerful though efficient formalism that can be seen as logic programming on ranges of the input string. For this reason, and for others (see [5]), it is a suitable basis to develop a linguistic formalism, but is not satisfying as such. Indeed, a linguistic formalism is almost always twofold, since on the one hand it builds the structure of the sentence, and on the other hand it computes features on this structure. In most formalisms, such as TAGs, LFGs, HPSGs, Dependency Grammars or Categorial Grammars, these two aspects are processed with different operators or even different mechanisms, not necessary simultaneous. In some cases, this leads to imprecise or excessive expressive power for the resulting formalism, and, if the separation between these two aspects is too strict, this also leads to problems concerning, for example, combinatorial explosion, error recovery algorithms, or automatic learning (dealing with unknown words).

Moreover, the limit between the structural backbone (often purely syntactic) and the features computed over it (often referred to as *decorations*), is hardly linguistically justified. Its position is not precisely and uniquely defined: a given formalism or implementation of a formalism can implement a linguistic property in the backbone, an other one treating it as a feature. On the other side, the linearity of most formalisms, i.e. their non-ability to reuse several times the same range of the sentence, makes it impossible to include additional information inside the grammar, such as lexical semantic constraints, in a satisfying way (i.e. not only as a limited set of "semantic" features that are not avoidable for linguistic reasons, but really as a complete set of semantic predicates).

For all these reasons, it seems appropriate to implement, if it is possible, all linguistically-motivated "decorations" inside the grammatical formalism that describes the structural backbone. Furthermore, we make the hypothesis that parsing natural language is possible in a polynomial time[5]. Thus, our linguistic

[5] Even the parsing of the *syntax only* of natural language probably needs the expressive power of RCGs, i.e. all *PTIME*. Indeed, it can be shown (see for example [5] and references therein) that some specific phenomena in some languages are beyond the expressive power of Mildly Context-Sensitive languages, and thus by formalisms as powerful as LCFRS (e.g. Chinese numbers, genitives in old Georgian, scrambling in German, or multiple verbs coordinates in Dutch).

formalism can be seen as decorated RCGs that can be compiled into pure RCGs. In the remaining of this section, we shall define our formalism, called Meta-RCGs (or MRCG), which defines over RCGs these decorations and the way they can be converted into RCG predicates and/or arguments. The borderline between the structure and the decorations is defined precisely by the following: a decoration is a property of (a portion of) an analysis, and the structure retains all properties of ranges.

The main idea underlying this approach is that the non-linearity of RCGs allows to treat as structural predicates (and not as decorations) several different linguistic facts over the same ranges. For example, syntactic and lexical semantic facts are used simultaneously. However, in a linguistic grammar, the analysis of a given range of the input string can be ambiguous. Therefore, the use we make of non-linearity has to be able to guarantee that all facts expressed in a global analysis about a given range are compatible: we have to prevent the apparition of analyses where such a range is analysed in a first way in one part of the global analysis, and in an other incompatible way in an other part of the analysis. However, it is not possible in a polynomial way to label the complete analysis of a range for later identification and re-use. Therefore, it is necessary to define a polynomial amount of information that will be exported by ranges to other parts of the analysis. This is what is done in our formalism, where this amount of information regroups the *heads* of a syntagm and *features* (decorations) associated to it, which will be made accessible at different parts of the analysis thanks to *contexts*. The remainder of the analysis of this syntagm will be inaccessible from the "outside".

Let $G_{RCG} = (N, T, V, P, S)$ be a classic RCG, as defined above. We will define an associated Meta-RCG (hereafter MRCG) G_{MRCG} extending G_{RCG} (the extension concerns grammars, not necessarily associated languages). For this, we will first go through preliminary remarks and new definitions.

3.1 Heads

While controversial by many aspects, the notion of *head of a syntagm* is widespread in linguistic literature, and has been intensively used by many grammatical frameworks, such as HPSG [6], but also LFG, Dependency Grammars, and others. We introduce heads (and coordinating items separating them) by extending the notion of argument in the following way: we call *MRCG-argument*, or more simply *argument* one of the following items:

– an (RCG-)argument, i.e. as said before the concatenation of elements of $V \cup T$,
– a *head-coordination argument* or *hc-argument*, i.e. an element of V followed by the operator $^\wedge$ or the operator `!`,
– a *single-head argument*, or *sc-argument*, i.e. an element of V followed by the operator $^+$,
– a *head-adding argument* or *ha-argument*, i.e. an argument of the form $\mathtt{V}_1{}^+\mathtt{V}_2{}^+\mathtt{V}_3{}^\wedge$.

An argument that is not a classical RCG-argument is called a *syntagmatic argument.* The intended meaning is the following. If `Syntagm` is a range denoting a syntagm, `Syntagm`$^\wedge$ denotes a pair of lists, the first one being the list of its heads and the second one the list of the coordinating items that are between them. Hence, if the substring corresponding to `Syntagm` is "an apple or a pear", a reasonable grammar will give an analysis such as `Syntagm`$^\wedge$ includes a list of heads covering "apple" and "pear" and a list of coordination items covering "or". The argument `Syntagm!` is the same thing, but with a number of heads (respectively coordination items) that is exactly 1 (respectively 0). A single-head argument $\mathtt{V}^+$ creates a syntagmatic argument made out of a list of heads containing only one element, `V` (thus, it has to be of length 1), and an empty list of coordinating items. Finally, a ha-argument $\mathtt{V}_1{}^+\mathtt{V}_2{}^+\mathtt{V}_3{}^\wedge$ is a syntagmatic argument whose heads list is the concatenation of V_1 and the heads list of $\mathtt{V}_3^\wedge$, and whose coordination items list is the concatenation of $\mathtt{V}_2$ and the coordination items list of $\mathtt{V}_3^\wedge$.

For example, and for illustration purposes only, a clause analysing (recursively) as a nominal group[6] a simple coordination of basic nominal groups could look like the following:

$$\begin{array}{c}\mathtt{NP(Det\ Head\ Coord\ Np2,Head^{+}Coord^{+}Np2^{\wedge})}\\ \to \mathtt{NOUN(Head)\ DET(Det,Head)\ COORD(Coord)\ NP(Np2,Np2^{\wedge}).}\end{array}$$

3.2 Features and homonym numbers

A *feature* (or *attribute*) can be defined as a (finite) vector $\mathbf{F} = (f_1, \ldots, f_n)$. A *constant value* of the feature $\mathbf{F}$ is an element of $\mathbf{F}$. A *variable value* of the feature $\mathbf{F}$ is the concatenation of the operator \$ and an element of a set of variable feature-values symbol, V_f (for example, if $\mathtt{g} \in V_f$, `$g` is a valid variable value for $\mathbf{F}$). We call *features list* a list of feature names. An *attribute-value pair*, or *av-pair*, is of the form `F =` v, where `F` is the name of a feature and v is a constant or variable value of $\mathbf{F}$. Finally, an *attribute-value pairs list*, or *av-list*, is a sequence of av-pairs in which an attribute can appear at most once in an av-pair.

We call homonym number a vector similar to a feature, defined as $\mathbf{HN} = (0, \ldots, h_{max})$. First, we define a *terminal argument* as an argument that has at most one variable, and that denotes a range of maximal length 1. The role of a homonym number is then to associate some terminal arguments of some predicates a special number that allows to distinguish between two homonymous terminals (i.e. words). For each predicate, a special function *homonymous-args* gives the list of the positions of its arguments that have such an homonym number. But these numbers are not apparent in MRCG-clauses.

3.3 Contexts

We define a *contextual item* as an element `Ctxt` of the set V of variable symbols possibly followed by the operator `/` or by the operator $^\wedge$, and possibly followed

[6] We do not use here the phrase *noun phrase* for a reason that will be explained later.

by the operator `:` and a features list. The role of contextual items is to modelize long-distance dependencies. For this reason, and although declarative as our whole formalism, the intended meaning of contexts is more easily described with an operational point of view. Long distance dependencies will be treated in 2 steps that share common points with the SLASH feature of HPSG: at one point of the analysis, a contextual item is built out of the concerned syntagm and "pushed" into the predicate-dependant *context* of the concerned predicate, `A`. All predicates related to `A` that can accept this context and for which no new value is explicitly given will inherit this value, thus transporting the context to other parts of the analysis. At some (arbitrarily distant) other point of the analysis, this contextual item, its heads and/or its features can be put into arguments (or "dropped", or "popped") and used.

Hence both operators, with the following meanings: the `/` operator means that the features list associated to the range `Ctxt` (this features list is in this case mandatory) is pushed into the context, but not `Ctxt` itself. The operator ^ means that the range `Ctxt` is a syntagm, and that its heads and coordination items, i.e. `Ctxt`^, have also to be pushed along with the range `Ctxt` in the context.

The percolation of contextual items from one point of the analysis to an other is allowed by a special function that needs to be defined, called *context-of*. This function is defined over the set N of predicate names and, for a given predicate, returns a list of contextual items. The meaning of this function is the following: given a predicate name A, the contextual items in *context-of(*`A)` are associated to all occurrences of `A` predicates. Suppose that, in a given clause, the same contextual item `Ctxt` (or its heads, or some of its associated features) is associated to more than one predicate. All such predicates that are on the right-hand side and that explicitly re-define the value of `Ctxt` will get this value. And all such predicates that are on the right-hand side and that do not re-define the value of `Ctxt` will share its value with the one of the left-hand side predicate. On the left-hand side, the value of `Ctxt` (or of `Ctxt`^ or of some of its features) can be equated to a real range (or to a feature value, if appropriate). For example, if a contextual item `Ctxt` is associated to both predicate names `A` and `B`, and if no specific equation redefines a new value of `Ctxt` for predicate `B`, then a clause such as `A(X)` $\rightarrow$ `B(X)` will ensure that the contextual item denoted by `Ctxt` is passed to the predicate `B` in the same way as the range denoted by `X`. We define only one operation on contexts, denoted by the operator `=`. When applied to a contextual range or contextual heads/coordinations, it is a range-equality operator. When applied to contextual features, it is a value-equality operator. The feature **F** of a contextual item `Ctxt` is accessible through a special dot operator with the following syntax: `Ctxt.F`. The distinction between a "push" and a "drop" (or "pop") lies only in the fact that the `=` operation is done on the left-hand side predicate ("drop") or on a right-hand side predicate ("push"), a direct use of `Ctxt` as a standard range in the right-hand side corresponding also to a "drop".

Finally, we call *context-value pair*, *cv-pair* or *contextual equation*, an expression of the form `Ctxt=`*Range* or `Ctxt.F=`v. We will see examples when we have defined the complete MRCG syntax.

3.4 MRCG

MRCG clauses are an extension of RCG clauses, since predicates are replaced by meta-RCG predicates of the following form:

$$\texttt{A(}\alpha_1,\ldots,\alpha_i,\ldots,\alpha_p\texttt{)}\,\texttt{[}\phi_1\ldots\phi_j\ldots\phi_q\texttt{]}\,\{\kappa_1\ldots\kappa_k\ldots\kappa_r\},$$

where α_i are MRCG-arguments, ϕ_j are av-pairs (equations on features), and κ_k are cv-pairs (contextual equations). The features part and the contextual part are facultative. For example, using `LDDep` as a shortcut for "long-distance dependency syntagm", here is a valid MRCG-clause, assuming that *context-of(*`NP`*)*= (`LDDep`$^\wedge$`:number,case`):

$$\texttt{VP(Verb,Verb}^{+}\texttt{)}\{\texttt{LDDep.case=accusative}\} \rightarrow$$
$$\texttt{VERB(Verb)}$$
$$\texttt{OBJ(LDDep, LDDep}^{\wedge}\texttt{,Verb)[number=LDDep.number]}\{\texttt{LDDep=0}\},$$

where `LDDep=0` is syntactic sugar for "`LDDep` is a 0-length range". The meaning of such a clause is the following: we can build with the range `Verb` a `VP` in a context where we have a long distance dependency `LDDep` with an accusative case if `Verb` is a verb, and if we can make of `LDDep` the direct object of this verb (this object having the same number than the one of `LDDep`), in a context where no long distance dependency is available any more for use.

3.5 Conversion from MRCG to RCG

As said before, the MRCG formalism can be converted into a strongly-equivalent RCG. This allows to use Boullier's efficient RCG parser, but also to avoid problems explained in the first paragraphs of this section that arise when the analysing process is decomposed in more than one step. However, the details of this conversion are not very interesting, and will not be presented here. We have designed and realized such a converter, that we call *MRCG-compiler*. As a side effect, this compiler produces an information file about the conversion that can be used to rebuild the MRCG-analysis of a sentence from the RCG-analysis given by the RCG parser.

3.6 Grammar and lexicon

An important part of the design of a grammatical formalism for natural language is the design of the interface between the grammar and the lexicon. Although it is possible, there is no reason to limit an RCG to be lexicalized, i.e. to include in every clause at least one left-hand side argument that has at least one terminal symbol. However, a lot of information is given by the terminals, here words[7], that has to be both represented and used in a way which, as for all parts of the formalism, has to be both computationally efficient and linguistically acceptable.

Several options could be thought of, taking advantage of the properties of closure of RCGs:

1. Compile a huge grammar with as many terminals as there are different inflected forms in the language, leading to an enormous grammar,
2. Compile separately the non-lexicalized part of the grammar and one or more lexical modules with only lexicalized clauses, either with the RCG parser generator or with a specific module that profits from the hierarchical structure of lexical information, uses appropriate algorithms to deal with the enormous amount of inflected forms and that is able to append the RCG parse forest with consistent sub-forests.
3. Compile the non-lexicalized part of the grammar, and, for each sentence, generate and compile dynamically the set of lexicalized clauses involving terminals present in the input sentence.

Independently of this choice, lexicalized clauses can be represented either as such, or be computable as the result of an inheritance process inside an ontology.

4 Parsing French with Meta-RCGs: two examples

As said before, the aim of the Meta-RCG formalism is to allow the design of grammars that take into account at the same time and with the same technical status morphological, syntactical and (lexical) semantics information. This is achieved by different predicates, thanks to the non-linearity of RCGs and hence Meta-RCGs.

We have developed, and still do, a large-coverage grammar for French language in the Meta-RCG formalism. To show how this formalism can be used for what it has been designed for, we will show how two sentences are analysed. The first sentence, *Un avocat mange un avocat* ("A lawyer eats an avocado"), shows how homonym numbers and lexical semantics work together to fully disambiguate ambiguous inflected forms. The second sentence, *Pierre veut une bière et dormir* ("Pierre wants a beer and [wants to] go to bed"), is an example of heterogeneous coordination (between a noun phrase and an infinitive verb phrase) and of subject control verb. Of course, both for place and complexity reasons, the whole analyses will not be shown, but only simplified parts of them to present the involved mechanisms.

In the following, instantiated MRCG-clauses will respect the following convention: a range $\langle i..j \rangle_w$ covering the substring s will be represented as $s_{i..j}$. If

[7] Since we want to include lexical semantic properties inside the grammar, the traditional approach of a lot of formalisms, which associate to all words a *category*, whose value is used as terminal symbol in the grammar, is not satisfactory. It would lose too much information or would require such a big amount of categories that there would not be any advantage over the direct use of words as terminals.

this range has an homonym number h, it will be represented as $s^h_{i..j}$. If it has heads, the corresponding terminals will be underlined, and its homonym number will be indicated as an exponent.

4.1 Lexical ambiguity and semantics: *Un avocat mange un avocat*

Let us consider the following sentence:

(1) Un avocat mange un avocat
A lawyer eats an avocado

This sentence is syntactically extremely simple. We use it as an example only to give an insight into the top-level clauses of our grammar and to show the role of homonym numbers. Moreover, grammar rules will be simplified. In particular, predicates and clauses dealing with non-existent topological places (e.g. sentence modifiers, adverbs, clitics, long-distance dependencies and so on) will be ignored. Finally, for space reasons, features will not be displayed, except when absolutely necessary, and terminals are abbreviated by their first characters followed by a dot when necessary. Some parts of the analysis, that are not very important for its global understanding, are replaced by textual descriptions printed in italics. This being said, here is how our grammar analyses this sentence (`VK` stands for *verbal kernel*, `VC` for *verbal complex*, and `dth=act` resp. `pass` for `diathesis=active` resp. `passive`):

$$\begin{array}{lll}
\texttt{PHRASE(un av. mange un av.}_{0..5}\texttt{)} & \rightarrow & \texttt{VSEM\_IND(mange}^0_{2..3}\texttt{)} \\
 & & \texttt{PHRASE2(un av. }\underline{\texttt{mange}}^0\texttt{ un av.}_{0..5}\texttt{) .} \\
\texttt{VSEM\_IND(mange}^0_{2..3}\texttt{)} & \rightarrow & \texttt{VERBE(mange}^0_{2..3}\texttt{)} \\
 & & \texttt{LEX(mange}^0_{2..3}\texttt{)[mode=Ind] .} \\
\texttt{VERBE(mange}^0_{2..3}\texttt{)} & \rightarrow & \texttt{.} \\
\texttt{LEX(mange}^0_{2..3}\texttt{)[mode=Ind]} & \rightarrow & \texttt{.} \\
\texttt{PHRASE2(un av. }\underline{\texttt{mange}}^0\texttt{ un av.}_{0..5}\texttt{)} & \rightarrow & \texttt{SUBJECT(un }\underline{\texttt{avocat}}^0_{1..2}\texttt{,mange}^0_{2..3}\texttt{)} \\
 & & \texttt{VP(}\underline{\texttt{mange}}^0\texttt{ un avocat}_{2..5}\texttt{)\{Subj=un }\underline{\texttt{av.}}^0_{1..2}\texttt{\} .} \\
\texttt{VP(}\underline{\texttt{mange}}^0\texttt{ un avocat}_{2..5}\texttt{)} & \rightarrow & \texttt{VK(mange}^0_{2..3}\texttt{,un avocat}_{3..5}\texttt{) .} \\
\texttt{VK(mange}^0_{2..3}\texttt{,un avocat}_{3..5}\texttt{)} & \rightarrow & \texttt{VC(}\underline{\texttt{mange}}^0\texttt{,}_{3..3}\texttt{,}_{3..3}\texttt{)} \\
 & & \texttt{ROLES(}_{2..2}\texttt{,}_{2..2}\texttt{,}_{2..2}\texttt{,un av.}_{3..5}\texttt{,mange}^0_{2..3}\texttt{) .} \\
\texttt{ROLES(}_{2..2}\texttt{,}_{2..2}\texttt{,}_{2..2}\texttt{,un av.}_{3..5}\texttt{,m.}^0_{2..3}\texttt{)} & \rightarrow & \texttt{OBJECT(un }\underline{\texttt{avocat}}^1_{3..5}\texttt{,m.}^0_{2..3}\texttt{)} \\
 & & \texttt{ROLES(un }\underline{\texttt{avocat}}^1_{3..5}\texttt{,}_{2..2}\texttt{,}_{2..2}\texttt{,}_{5..5}\texttt{,m.}^0_{2..3}\texttt{) .}
\end{array}$$

Before going on with the remainder of the analysis, it is necessary to clarify the meaning of the arguments of the predicate `ROLES`. In fact, the 3 first arguments (they are 5 in the real grammar) denotes the syntactico-semantic roles associated with the verb. The first role is the accusative one, the second role is the dative one, and the last one is the genitive one. The recursive predicate `ROLES` "eats" at each call a complement in the list of not-yet-parsed complements (4th argument), analyses it, and, if it fills a not-yet-filled role, fills the corresponding argument of the right-hand side call of `ROLES` with it. When the range of not-yet-parsed arguments is empty, roles filling is completed, and verifications can be done about mandatory or impossible roles. This being said, here is how the analysis goes on:

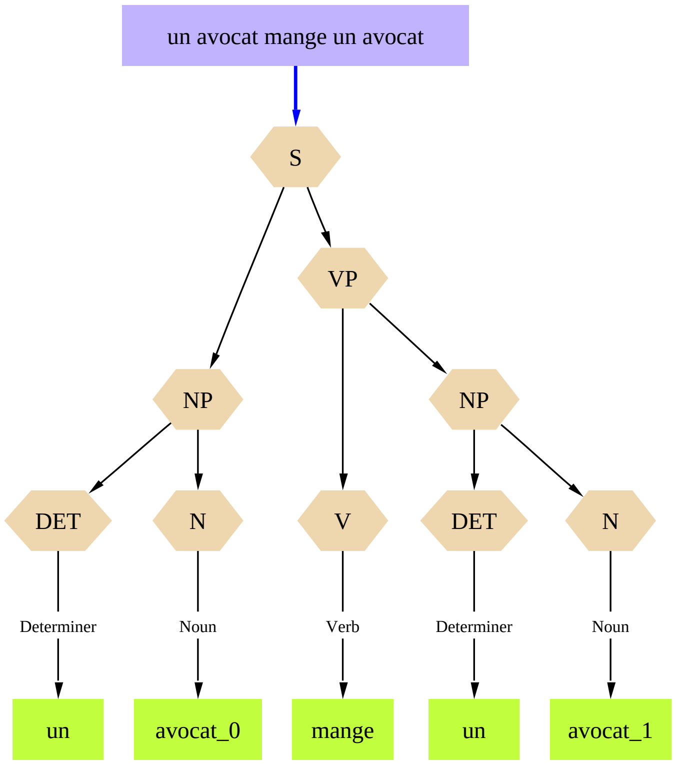

Fig. 1. Constituency view of the analysis of sentence (1).

$$
\begin{array}{lll}
\texttt{ROLES(un }\underline{\texttt{avocat}}^{1}_{3..5}\texttt{,}{}_{2..2}\texttt{,}{}_{2..2}\texttt{,}{}_{5..5}\texttt{,m.}{}^{0}_{2..3}\texttt{)} & \longrightarrow & \textit{True because } \texttt{mange}^{0} \textit{ is a transitive} \\
 & & \textit{verb and the first argument of} \\
 & & \texttt{ROLES}\textit{, i.e. the accusative} \\
 & & \textit{role, is filled by } \texttt{un }\underline{\texttt{avocat}}^{1}_{3..5} \\
\texttt{SUBJECT(un }\underline{\texttt{avocat}}^{0}_{0..2}\texttt{,mange}^{0}_{2..3}\texttt{)} & \longrightarrow & \texttt{SUBJ\_SEM(avocat}^{0}_{1..2}\texttt{,mange}^{0}_{2..3}\texttt{)} \\
 & & \texttt{NP(un }\underline{\texttt{avocat}}^{0}_{0..2}\texttt{) .} \\
\texttt{SUBJ\_SEM(avocat}^{0}_{1..2}\texttt{,mange}^{0}_{2..3}\texttt{)[d.=act]} & \longrightarrow & \texttt{AGENT(avocat}^{0}_{1..2}\texttt{,mange}^{0}_{2..3}\texttt{)} \\
\texttt{AGENT(avocat}^{0}_{1..2}\texttt{,mange}^{0}_{2..3}\texttt{)} & \longrightarrow & \texttt{ANIMATED(avocat}^{0}_{1..2}\texttt{) .} \\
\texttt{OBJECT(un }\underline{\texttt{avocat}}^{1}_{3..5}\texttt{,mange}^{0}_{2..3}\texttt{)} & \longrightarrow & \texttt{OBJ\_SEM(avocat}^{1}_{4..5}\texttt{,mange}^{1}_{2..3}\texttt{)[dth=pass]} \\
 & & \texttt{NP(un }\underline{\texttt{avocat}}^{1}_{3..5}\texttt{) .} \\
\texttt{OBJ\_SEM(avocat}^{1}_{4..5}\texttt{,mange}^{0}_{2..3}\texttt{)[d.=pas]} & \longrightarrow & \texttt{PATIENT(avocat}^{1}_{4..5}\texttt{,mange}^{0}_{2..3}\texttt{)} \\
\texttt{PATIENT(avocat}^{0}_{1..2}\texttt{,mange}^{0}_{2..3}\texttt{)} & \longrightarrow & \texttt{EDIBLE(avocat}^{1}_{4..5}\texttt{) .} \\
\texttt{NP(un }\underline{\texttt{avocat}}^{0}_{0..2}\texttt{)} & \longrightarrow & \textit{True because } \texttt{un }\underline{\texttt{avocat}}^{0}_{1..2} \\
 & & \textit{is a valid noun phrase} \\
\texttt{NP(un }\underline{\texttt{avocat}}^{1}_{3..5}\texttt{)} & \longrightarrow & \texttt{idem}
\end{array}
$$

As can be seen, the process can be summed up as follows:

– Identify the semantic part of the main verb (the past participle if there is one or more auxiliarie(s)), and make of it the head of the sentence,
– Identify the whole verbal kernel (verbal components, clitics, and adverbs that are inbetween),
– Analyse the subject, which is a noun phrase (`NP` clause ; noun phrases can be infinitives or propositions) and a semantic argument of the verb (`SUBJ_SEM` clause), and put it in the context (not used in this sentence),
– Identify one after the other all post-verbal complements (here only one), and analyse it both as a noun phrase (`NP` clause) and as a semantic argument of the verb (`*_SEM` clauses).

Of course, this simplified presentation doesn't explain how are treated attributes of the subject or of a complement, clitics, long-distance dependencies, modifiers, relatives, and so on. It is only the basic skeleton of the grammar.

For illustration purposes, we give in Figure 1 the constituency tree extracted automatically from the global analysis.

4.2 Heterogeneous coordination and control verb: *Pierre veut une bière et dormir*

Let us consider the following sentence:

(2) Pierre veut une bière et dormir
Pierre wants a beer and [wants to] sleep

As said before, this sentence exemplifies two phenomena: heterogeneous coordination between a nominal syntagm (*une bière*) and an infinitive (*dormir*), and subject control verb (*veut*, which controls the subject *Pierre* for the infinitive *dormir*). We will not give the whole analysis of this sentence, but only the most interesting parts.

First, we will see how the heterogeneous coordination is treated. In fact, we make the distinction between a noun phrase and a nominal group. For us, a *noun phrase* is a phrase that plays the role which is canonically fulfilled by a phrase built around a noun. Such a phrase is a *nominal group*. Thus, a nominal group, an infinitive or a proposition are noun phrases. This distinction makes it possible to explain several facts, including the fact that an infinitive and a nominal group can indeed be coordinated, as in (2). It can also deal with the fact that, for example, several verbs accept an infinitive as a direct object. Predicates such as `OBJ_SEM` can nevertheless constraint the object of a verb to be an infinitive, or on the contrary prevent it from being an infinitive. But in our grammar, this is a matter of constraints over the object, and not a fundamental difference between different kinds of syntagms, at least at the level of the *object* relation. Hence the following analysis for this relation, in which contexts are not shown, since they will be useful and thus shown later (`NG` stands for *nominal group*):

OBJECT(une $\underline{\text{bière}}$ et $\underline{\text{dormir}}_{2..6}$)	$\rightarrow$ OBJ_SEM(bière$^{0}_{3..4}$,veut$^{0}_{1..2}$)
	OBJ_SEM(dormir$^{0}_{5..6}$,veut$^{0}_{1..2}$)
	NP(une $\underline{\text{bière}}^{0}$ et $\underline{\text{dormir}}^{0}{}_{2..6}$) .
OBJ_SEM(bière$^{0}_{3..4}$,veut$^{0}_{1..2}$)	$\rightarrow$ NOUN(bière$^{0}_{3..4}$) .
OBJ_SEM(dormir$^{0}_{5..6}$,veut$^{0}_{1..2}$)	$\rightarrow$ *[See below]*
NP(une $\underline{\text{bière}}^{0}$ et $\underline{\text{dormir}}^{0}{}_{2..6}$)	$\rightarrow$ NP(une $\underline{\text{bière}}^{0}{}_{2..4}$)
	NP($\underline{\text{dormir}}^{0}{}_{5..6}$)
	COORD(et$^{0}_{4..5}$)
	[other predicates
	for features processing] .
NP(une $\underline{\text{bière}}^{0}{}_{2..4}$)	$\rightarrow$ NG(une $\underline{\text{bière}}^{0}{}_{2..4}$) .
NG(une $\underline{\text{bière}}^{0}{}_{2..4}$)	$\rightarrow$ *[Standard analysis for a nominal group]*
NP($\underline{\text{dormir}}^{0}{}_{5..6}$)	$\rightarrow$ INFINITIVE($\underline{\text{dormir}}^{0}{}_{5..6}$) .
INFINITIVE($\underline{\text{dormir}}^{0}{}_{5..6}$)	$\rightarrow$ *[See below]*

In our grammar, any noun phrase can potentially be an infinitive. The fact that *veut* is a subject control verb has only one impact: it allows the predicate `OBJ_SEM`(dormir$^{0}_{5..6}$,veut$^{0}_{1..2}$) to be true. Thus, the noun phrase that is the object of *veut*^{0} can really be an infinitive. The analysis of this infinitive, here *dormir*, uses the contextual item `Subject` in the following way. As for the sentence studied in the previous paragraph, the analysis of the subject, here $\underline{\text{Pierre}}^{0}_{0..1}$ (see the `PHRASE2`-clause in the previous example), "pushes" this syntagm in a contextual item named `Subj`. Then, the predicate `INFINITIVE`, which

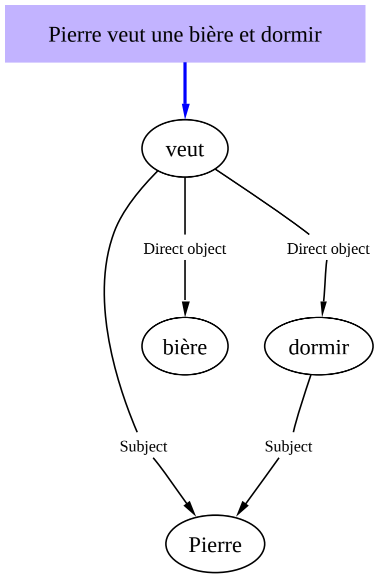

Fig. 2. Dependency view of the analysis of sentence (2).

inherits this context through respectively VP, VK, ROLES, OBJECT and NP, uses it to build the syntactico-semantical dependency thanks to the following clause (VK_INF stands for *verbal kernel of an infinitive*, and the context of INFINITIVE is now shown):

$$\texttt{INFINITIVE}(\underline{\texttt{dormir}}^{0}{}_{5..6})\{\texttt{Subj}=\underline{\texttt{Pierre}}^{0}_{0..1}\ \texttt{Subj.gender=masc}\ \texttt{Subj.number=sing}\}$$
$$\rightarrow \texttt{VK\_INF}(\underline{\texttt{dormir}}^{0}{}_{5..6},\texttt{Pierre}^{0}_{0..1})[\texttt{gender=masc}\ \texttt{number=sing}]\ .$$

For illustration purposes, we give in Figure 2 the dependency graph extracted automatically from the global analysis.

5 Conclusion

We have presented a new formalism, called *Meta-RCG*, and based on Range Concatenation Grammars. This formalism, thanks to the non-linearity of RCGs, allows the development of grammars that implement linguistic facts as predicates over ranges of the input sentence. These facts can deal for example with morphology, syntax, lexical semantics, combinaisons thereof. The non-linearity makes it unnecessary to put in costly and numerous features all the linguistic information that doesn't fit an insufficient backbone. Moreover, the very concept of predicates over ranges of the input string seems very intuitive from a linguistic point of view.

We have shown thanks to two small examples the way linguistic Meta-RCGs can be developed. As said before, we are currently developing such a grammar for French. The development is already quite advanced, and we can deal with phenomena like subject, object or indirect-object control verbs, raising verbs, infinitives, completives (either modifiers or arguments), light verbs, homogeneous and heterogeneous coordination, participial modifiers, attributes of the subject or of the object, auxiliaries, arguments of nouns ("Peter's departure") or adjectives, negation (which can be discontinuous in French), relatives (including relatives in "dont" that can modify the subject or the object of an arbitrarily deep verb inside the relative) and many other less complicated phenomena. Parsing times are very satisfying[8]. We are currently using the EUROTRA corpus of French sentences [8], which gives a good set of simple to very complicated sentences: we proceed sentence by sentence in an exhaustive manner, and modify the grammar so as to get only the appropriate parse (or the appropriate parses if the sentence is really ambiguous). We have currently reached the second half of the file, but we have already implemented several phenomena that appear later on in this corpus. Moreover, we have developed filters that allow to project our analysis

[8] We use a "benchmark-sentence" which has simultaneously a relative in "dont" that modifies the coordinated object of a verb of the relative, with a coordinated subject, and which is an object-controlled verb inside a completive. Parsing time, depending on the state of the grammar, has oscillated in the last months between 0.7 and 2.5 seconds. This sentence is the following: *Paul aime la Normandie dont je sais que Pierre regarde Marie et Paul manger une pomme et une poire verte*

into different views, including (for now) a constituency tree, a dependency graph, topological boxes, and predicate-arguments semantics.

For all these reasons, we believe that we have designed a formalism that virtually satisfies the constraints given in the introduction of this paper to characterize an interesting formalism. Further work includes the continuation of the extension of our grammar for French thanks to this EUROTRA corpus, a more precise definition of the abstract foundations of our formalism, an in-depth linguistic analysis of its properties and of the grammars it allows to write, and a more precise definition of the interface between lexicon and grammar. It also includes the extension of our grammar to other components of the linguistic analysis of a sentence, such as discourse analysis or Montague-like semantics.

References

1. Dahl, V., Tarau, P. and Huang Y.-N.: Datalog Grammars. In: GULP-PRODE 2 (1994) 268–282
2. Blache, P.: Parsing with Constraint Graphs: a Flexible Representation for Robust Parsing. In Di Sciullo, A.M., ed.: Grammars and NLP LNCS, Springer-Verlag (2001)
3. Boullier, P: Range Concatenation Grammars. In Bunt, H., Carroll, J., Satta, G., ed.: New developments in parsing technology. Kluwer Academic Publishers (2004) 269–289
4. Boullier, P.: Counting with range concatenation grammars. Theoretical Computer Science **293** (2003) 391–416
5. Sagot, B., Boullier, P.: Les RCG comme formalisme grammatical pour la linguistique. In: Proceedings of TALN '04, Fez, Marocco (2004) 403–412
6. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar, University of Chicago Press and CSLI Publications (1994)
7. Barthélémy, F., Boullier, P., Deschamp, P., Villemonte de La Clergerie, É.: Guided Parsing of Range Concatenation Languages. In: Proceedings of ACL '01, Toulouse, France (2001) 42–49
8. Danlos, L., Laurens, O.: Présentation du Projet Eurotra et des grammaires d'Eurotra-France. Technical Report n 1, Université Paris 7 - Talana/LISH (1991)