GENERIC RULES AND NON-CONSTITUENT COORDINATION

Julio Gonzalo
UNED
julio@ieec.uned.es

Teresa Solías
Universidad de Valladolid
solias@cpd.uva.es

Abstract

We present a metagrammatical formalism, *generic rules*, to give a default interpretation to grammar rules. Our formalism introduces a process of *dynamic binding* interfacing the level of pure grammatical knowledge representation and the parsing level. We present an approach to non-constituent coordination within categorial grammars, and reformulate it as a generic rule. This reformulation is context-free parsable and reduces drastically the search space associated to the parsing task for such phenomena.

1 Introduction

Grammatical theories always make a distinction between unmarked and marked phenomena. The general case is to use certain mechanisms which are likely to deal with regular behaviour. More complex mechanisms should be available when exceptional behaviour is present. However, this difference is not formally expressed in most grammar accounts of complex phenomena. The need to restrict licensing of additional grammar resources is commonly asserted outside of the formal framework used. As exceptional rules involve a higher computational cost, parsing processes associated to such formalisms become computationally intractable.

Non-constituent coordination treatments show this pattern. In general, a conjunction combines constituents. But there are cases where the conjunction coordinates linguistic elements which do not form a constituent. For instance, consider '*John went to Alabama in summer and to Spain in fall*'. The conjunction '*and*' is coordinating ' *to Alabama in summer*' with ' *to Spain in fall*', which do not form constituents. The solutions displayed in the literature incorporate additional grammatical resources to deal with such data. We would expect a high restriction on the use of the rules associated to this phenomena, but all we have is informal comments about when to apply them. That makes parsing time of the grammar exponential in the length of the string.

Though there is a general concern on the need to make explicit some process of rule licensing, no formal framework has been proposed, to our knowledge, to regulate the interaction between regular and exceptional grammatical resources. In this paper we propose a metagrammatical formalism, *generic rules*, to give a different degree of specificity to each grammatical rule. We present an approach to parse non-constituent coordination within categorial grammars that can be parsed in polynomial time when reformulated as a generic rule.

The essential idea is to introduce a process of *dynamic binding* interfacing the level of pure grammatical representation and the parsing processes. When the parsing process calls the grammar to combine linguistic objects, the dynamic binding process inspects the applicable grammatical resources, considers their different priority and returns an effective rule, or set of rules, to be applied on such linguistic objects. This approach is inspired in the object-oriented programming paradigm, where polymorphism is exploited in order to get default and specific behaviour.

In order to make dynamic binding a meaningful process, we will view generic and specific behaviour as object-oriented functions that map daughters information into mother features. A set of such functions will be called a *generic rule.* Precedence between them will be established according to the specificity of the type constraints on the daughters.

This paper is structured as follows: first, we outline the main ideas behind our approach. Next, we make a more formal presentation of *generic rules.* Then, we turn to the linguistic problem of non-constituent coordination, and give a linguistic explanation within categorial grammar that uses the *tuple* operation introduced in [Solías, 1993, Morrill and Solias, 1993]. Recognition with that grammar is NP-hard, as it is with most extensions of Lambek Calculus. We present, then, a reformulation of that grammar as a generic rule. The representational facilities of generic rules allow the specification of each grammatical resource at an appropriate level of specificity, drastically constraining the search space for the parsing task.

2 Generic Rules

2.1 Underlying Ideas

Consider a context-free grammar - possibly augmented with functional restrictions attached to each rule - that includes the hierarchy $\mathtt{VP} \rightarrow \mathtt{VP}_1 \mid \mathtt{VP}_2 \mid \mathtt{VP}_3 \mid \mathtt{VP}_4 \mid \mathtt{VP}_i$ and the rule $\mathtt{S} \rightarrow \mathtt{NP\ VP}$.

If we decide to write down a new rule that specifies a different, more specific behaviour for $\mathtt{VP}_i$ when combining with a NP, we could think of adding a new rule as: $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$. If our aim is to state this rule as *the* rule to be used when the VP belongs to the more specific subclass $\mathtt{VP}_i$, we want the production $\mathtt{S} \Rightarrow \mathtt{NP\ VP}_i$ to be valid, but not the production $\mathtt{S} \Rightarrow \mathtt{NP\ VP} \Rightarrow \mathtt{NP\ VP}_i$. Just adding $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ to the CFG does not capture the exceptional sense of the rule; in a CFG, both derivations are equally possible.

We have two options to get the desired behaviour from the CFG. The first one is to keep $\mathtt{S} \rightarrow \mathtt{NP\ VP}$ and rewrite the specific rule $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ into the following set of rules:

$$\mathtt{S} \rightarrow \mathtt{NP\ VP}_1 \mid \mathtt{NP\ VP}_2 \mid \mathtt{NP\ VP}_3 \mid \mathtt{NP\ VP}_4 \tag{1}$$

The second one is to rewrite the hierarchy, in order to keep the number of rules invariant:

$$\begin{array}{ll} \mathtt{VP} \rightarrow \mathtt{VP}_i \mid \mathtt{VP}_t & \mathtt{S} \rightarrow \mathtt{NP\ VP}_t \\ \mathtt{VP}_t \rightarrow \mathtt{VP}_1 \mid \mathtt{VP}_2 \mid \mathtt{VP}_3 \mid \mathtt{VP}_4 & \mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i \end{array} \tag{2}$$

None of these options is an incremental way of capturing exceptions in a grammar. The introduction of an exception forces the revision of previously described lexical types and grammar rules.

This naive example exemplifies a problem that becomes significant when an operation or rule carries on most of the information-combining task. This is a standard situation in lexicalist approaches to grammar. For instance, [Dowty, 1988] proposes a distinction between a basic categorial grammar, which only includes *forward* and *backward application*, and an extended grammar that also includes *type raising* and *functional composition.* The basic grammar is used for most of the analysis, whereas the extended one should only be used in specific situations such as non-constituent coordination. It is suggested that this rules should be licensed only when functional application fails to parse a string. However, no formalization of the different status of such rules is offered.

We propose a formalism to express and handle default and exceptional rules within a phrase-structure approach that permits this kind of representations. Our formalism describes rules as object-oriented functions that map daughters information into mother features. As in object-oriented programming, precedence between rules is not given statically in the linguistic signature, but it is dynamically established upon the types of the input structures of the compositional process. When combining two constituents, a metagrammatical mechanism establishes a partial order for the candidate rules and selects the most appropriate rule (or set of rules) to be applied.

Before defining such framework, let us introduce the proposal with a generic rule representation for the naive example above. The generic rule having the desired behaviour for rule `S` $\rightarrow$ `NP VP` and rule `S`$_i$ $\rightarrow$ `NP VP`$_i$ has this aspect:

$$\text{SYN} = \begin{cases} \text{SYN}_{NP \otimes VP}(x, y) = S \\ \text{SYN}_{NP \otimes VP_i}(x, y) = S_i \end{cases} \tag{3}$$

Equation 3 describes a generic rule, SYN, that returns a type for the composition of two linguistic objects. $\text{SYN}_{NP \otimes VP}(x, y) = S$ is a partial rule applicable when the arguments x, y belong to the types NP, VP, in that order. $\text{SYN}_{NP \otimes VP_i}(x, y) = S_i$ is applicable when the arguments belong to the types NP, VP_i.

A bottom-up parsing process may call SYN when it tries to combine two objects with types x, y. The applicable partial rules will be $\{\text{SYN}_{a \otimes b} \mid a \le x, b \le y\}$. The natural way to establish precedence between rules is attending to the specificity of the type constraints of their arguments: $\text{SYN}_{a \otimes b}$ will be more specific than $\text{SYN}_{c \otimes d}$ if $a \le c, b \le d$.

Some examples of the application of SYN would be:

$$\begin{array}{ll} \text{SYN}(NP, VP) = S & \text{SYN}(NP, VP_2) = S \\ \text{SYN}(VP_2, NP) = \perp & \text{SYN}(NP, VP_i) = S_i \end{array} \tag{4}$$

In each case, the following partial rules have been applied:

$$\begin{array}{ll} \text{SYN}(NP, VP) \Longrightarrow \text{SYN}_{NP \otimes VP} & \text{SYN}(NP, VP_2) \Longleftrightarrow \text{SYN}_{NP \otimes VP} \\ \text{SYN}(VP_2, NP) \Longleftrightarrow \perp \text{ (no applicable rule)} & \text{SYN}(NP, VP_i) \Longleftrightarrow \text{SYN}_{NP \otimes VP} \end{array} \tag{5}$$

In that way, we have the ability to express default rules, and rules with different degrees of specificity in an incremental way, without a need to give them different formal status. Also, only minor changes have to be made to a CFG parsing algorithm in order to work with generic rules.

2.2 Definition of a Generic Rule

Consider a partially ordered set (*poset*) $\langle \le, \mathcal{T} \rangle$ and a domain of linguistic objects $\mathcal{O}$ typed with $\langle \le, \mathcal{T} \rangle$ (i.e., there is a function type:$\mathcal{O} \rightarrow \mathcal{T}$). Consider also an informational domain Φ associated to $\mathcal{O}$ by means of a function ϕ:$\mathcal{O} \rightarrow \Phi$ (perhaps multivaluated) that associates at least one element in Φ to every object of $\mathcal{O}$.

A *generic rule* G over the tuple $\langle \langle \le, \mathcal{T} \rangle, \mathcal{O}, \Phi \rangle$ is another tuple $\langle \langle \preceq, \mathcal{T}^* \rangle, \mathcal{F}, \mathcal{G} \rangle$, where:

- $\mathcal{F}$ is a set of functions, that will be called *partial rules*, of the form:

  $$\text{f}_{t_1 \otimes t_2} : \Phi \times \Phi \rightarrow \Phi$$

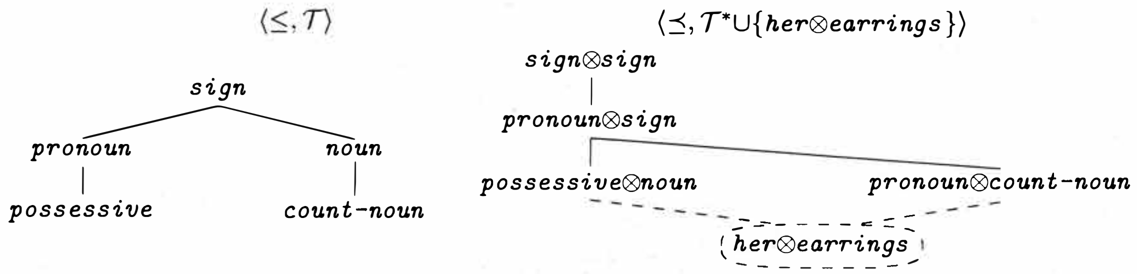
  where $t_1, t_2 \in \mathcal{T}$, and the following condition holds on the set of partial rules:
  $\forall \text{f}_{t_1 \otimes t_2}, \text{f}_{q_1 \otimes q_2} \in \mathcal{F}, t_1 = q_1 \wedge t_2 = q_2 \Longleftrightarrow \text{f}_{t_1 \otimes t_2} = \text{f}_{q_1 \otimes q_2}$.

- $\langle \preceq, \mathcal{T}^* \rangle$ is a partially ordered poset defined over $\mathcal{F}$ as follows:

  $$\mathcal{T}^* \equiv \{t_1 \otimes t_2 \mid \text{f}_{t_1 \otimes t_2} \in \mathcal{F}\}$$
  $$\forall x_1 \otimes x_2, y_1 \otimes y_2 \in \mathcal{T}^* \quad x_1 \otimes x_2 \preceq y_1 \otimes y_2 \Longleftrightarrow x_1 \le y_1 \textit{ and } x_2 \le y_2$$

  We call *cartesian poset* $\langle \preceq, \mathcal{T}^* \rangle$ over a poset $\langle \le, \mathcal{T} \rangle$ to any subset of $\mathcal{T} \times \mathcal{T}$ equipped with the partial ordering relation $\preceq$.

  We call *cartesian type* of a partial rule $\text{f}_{t_1 \otimes t_2}$ to the type specification $t_1 \otimes t_2$.



$\langle \leq, \mathcal{T} \rangle$
sign
pronoun
noun
possessive
count-noun
$\langle \preceq, \mathcal{T}^* \cup \{her \otimes earrings\} \rangle$
sign⊗sign
pronoun⊗sign
possessive⊗noun
pronoun⊗count-noun
her⊗earrings

Figure 1: When the generic rule associated to $\mathcal{T}^*$ is invoked to compose *her* and *earrings*, a cartesian type *her⊗earrings* is dynamically considered, but it has more than one direct supertype; therefore, there is not a single most specific rule to be applied on that constituents.

- $\forall x_1, x_2 \in \mathcal{T}, \mathcal{G}(x_1, x_2) \equiv f_{t_1 \otimes t_2}$ such that $t_1 \otimes t_2$ is the lowest upper bound of $x_1 \otimes x_2$ in $\mathcal{T}^* \cup \{x_1 \otimes x_2\}$.

  $\mathcal{G}$ is called the *dynamic binding function* for G. It interfaces parsing processes with the grammar. selecting the most specific partial rule in $\mathcal{F}$ given two arguments of types x_1, x_2.

Note that $\mathcal{T}^*$ is a subset of the cartesian product $\mathcal{T} \times \mathcal{T}$, and that the partial ordering relation $\preceq$ is deduced from the relation $\leq$ that holds between the elements of $\mathcal{T}$. By means of the definition of *cartesian poset*, we have managed to axiomatize precedence issues between partial rules. turning the dynamic binding process into a simple order checking over the elements of a poset.

The function $\mathcal{G}$ provides the interface of the generic rule, as a system of object-oriented functions, with the parsing process that calls it. In particular, the dynamic binding function provides the default interpretation for partial rules. Given a pair of arguments of types x_1, x_2, a dynamic extension $\mathcal{T}^* \cup \{x_1 \otimes x_2\}$ of the original cartesian poset $\langle \preceq, \mathcal{T}^* \rangle$ that includes the cartesian type $x_1 \otimes x_2$ is considered. In virtue of the definition of the partial ordering relation $\preceq$, the new type $x_1 \otimes x_2$ takes its place in the hierarchy. Its supertypes denote all the applicable rules to compose the pair of arguments, and the partial ordering between this supertypes denotes precedence between them. The action of the generic rule G over a couple of linguistic objects with types x_1, x_2 and associated expressions ϕ_1, ϕ_2 is given by $\mathcal{G}(x_1, x_2)(\phi_1, \phi_2)$.

In our definition, precedence is used to keep the most specific rule and override the rest. However, more sophisticated binding processes could be considered. For instance, if the partial rules were unification constraints, some default version of unification could be performed on the applicable rules, taking their relative precedence into account.

We have adopted the restriction of binary branching (binary rules) to take full advantage of the concept of dynamic binding with the minimum effort. However, this is not a serious limitation: most linguistically significant rules are binary, and those which are not can be easily converted in binary rules.

2.3 Well-formedness

Given the definition of the preceding section, there can be situations where precedence conflicts cannot be solved. When the cartesian type that represents the arguments of a call to the generic rule has more than one direct supertype, there is not a single "most specific rule" (see the example in Figure 1).

If this indeterminacy is possible for a given generic rule, we say that the Cartesian Poset $\langle \preceq, \mathcal{T}^* \rangle$ is not well-formed:

A Cartesian Poset $\langle \preceq, \mathcal{T}^* \rangle$ *is* **well-formed** *iff*
$\forall\ x_1 \otimes x_2, y_1 \otimes y_2 \in \langle \preceq, \mathcal{T}^* \rangle$ *not ordered, such that* $m_1 = x_1 \wedge y_1 \in \langle \leq, \mathcal{T} \rangle$ *and* $m_2 = x_2 \wedge y_2 \in \langle \leq, \mathcal{T} \rangle$*, it holds that* $m_1 \otimes m_2 \in \langle \preceq, \mathcal{T}^* \rangle$ [1]

[1]This definition assumes that the original poset is bounded complete, i.e., each pair of types have at most

This kind of indeterminacy appears also when dealing with multiple default inheritance [Daelemans et al., 1992] or default unification over feature structures with structure-sharing [Bouma, 1992, Carpenter, 1993], and can only be solved adding extra ordering information or forbidding such situations. Our well-formedness condition adopts the first solution. The main reason is that the cartesian hierarchies are not defined by the user, but arranged by the system. Such conflicts are potentially dangerous, as the grammar writer is not necessarily aware of them. An advantage of the definition of well-formedness is that it can detect inconsistencies at compile-time and signal them for correction.

2.4 Parsing with Generic Rules

A generic rule may interact with a phrase-structure grammar to perform the composition of some informational field. In [Gonzalo, 1995], for instance, we propose generic rules as a well-suited mechanism to perform categorial semantic interpretation as a modular process that interacts with a phrase-structure grammar. On the other hand, a grammar could be made up exclusively of generic rules, adequately combined to perform parsing. We will present a simple account of each of these possibilities.

We will adopt here the deductive parsing approach described in [Shieber et al., 1994]; it provides a neat account of parsing systems, simplifying our presentation. We will start from a bottom-up shift reduce algorithm as presented in [Shieber et al., 1994]. We will gradually adopt this algorithm to capture generic rule parsing.

Let α be a string of terminals w_i. Let $[\alpha \bullet, j]$ stand for $\alpha w_{j+1} \cdots w_n \overset{*}{\Rightarrow} w_1 \cdots w_n$. In [Shieber et al., 1994] a shift-reduce bottom-up deductive parsing system is expressed as the following calculus:

Inference Rules:

Axiom: $[\bullet, 0]$
Goal: $[S \bullet, n]$

Shift $\dfrac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$

Reduce $\dfrac{[\alpha\gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \rightarrow \gamma$

This parser can be augmented to do semantic interpretation. If each rule has an associated function that related the meanings of the elements in the rhs with the meaning of the lhs element, the basic tuples are $[\alpha \bullet, j, \phi]$. The reduce rule carries on semantic composition:[2]

$$\textbf{Reduce} \quad \frac{[\alpha\gamma \bullet, j, \Phi < \phi_1 \cdots \phi_{|\gamma|} >]}{[\alpha B \bullet, j, \phi < f(\phi_1 \cdots \phi_{|\gamma|}) >]} \quad B \rightarrow \gamma, \mathrm{f}$$

where f is the semantic composition function associated to the rule $B \rightarrow \gamma$.

We can augment the shift-reduce parser in a similar way to represent the interaction between a generic rule and a phrase-structure grammar. We have to take account of two differences between generic rule parsing and the syntactic-semantic parser above:

1. The action of the generic rule has to be specified through the dynamic binding function. There is not a partial rule associated to each context-free rule.

2. The interpretation of the context-free rules has to be slightly modified to take into account the hierarchization of non-terminals. A hierarchy is equivalent, in context-free terms, to a set of unary rules in which every inmediate-ordering relation is expressed writing a type as a left-hand side of a rule, and its subtype as the right-hand side. Being these rules "compiled" into the hierarchy, a context-free rule as $T \rightarrow T_1 T_2$, has to be interpreted as $T \rightarrow X_1 X_2$ such that $X_1 \leq T_1, X_2 \leq T_2$.

one common subtype.

[2]The other rules are modified trivially.

The basic element in the logic that represents the parsing algorithm is the same as in the preceding case: $[\alpha \bullet, j, \Phi]$. Now, Φ is an expression that belongs to the informational domain associated to a generic rule G.

The reduce rule that takes account of the interaction of a generic rule with a context-free grammar is

$$\textbf{Reduce} \quad \frac{[\alpha x_1 x_2 \bullet, j, \Phi < \phi_1 \phi_2 >]}{[\alpha A \bullet, j, \Phi < \mathcal{G}(x_1, x_2)(\phi_1, \phi_2) >]} \quad \begin{array}{l} A \rightarrow BC, x_1 \leq B, x_2 \leq C \\ \mathcal{G}(x_1, x_2)(\phi_1, \phi_2) \neq \perp \end{array} \tag{6}$$

The interaction with the generic rule is expressed in the term $\mathcal{G}(x_1, x_2)(\phi_1, \phi_2)$. The left part, $\mathcal{G}(x_1, x_2)$, performs dynamic binding, returning the effective rule that will be applied on x_1, x_2.

The side conditions on this rule are: one, that exists an applicable syntactic rule $A \rightarrow BC, x_1 \leq B, x_2 \leq C$. And two, that the generic function, applied over ϕ_1, ϕ_2, returns a positive result. Note that it could be the case that there is an appropriate syntactic rule, but the additional restrictions imposed by the generic rule do not hold [3].

It is interesting to remark that the dynamic binding process depends on the particular objects in the Reduce rule, not only on the conditions of the context-free rule. That makes impossible to pre-attach an "effective partial rule" to each context-free rule at compilation time, such that the binding process could be performed off-line. This fact is reflected in the term $\mathcal{G}(x_1, x_2)$. If it were $\mathcal{G}(B, C)$, binding could be done at compile time. Though this behaviour introduces and additional complexity factor in generic rule parsing, it allows the specification of semantic and syntactic processes as independent mechanisms that interact modularly.

A particular case of generic rule is that in which the combination functions simply return a type, as in our first example. In such a rule, the role of every partial rule is similar to that of a context-free rule, and the generic rule works as a context-free grammar in which rules has a default interpretation. For this kind of generic rules, that combine syntactic information, it is senseless to consider their interaction with a context-free grammar, as they are parsable by themselves. Such parsing does not differ substantially from context-free parsing. The main restriction is that, due to the functional interpretation of partial rules, bottom-up algorithms are best suited that top-down mechanisms, that would require the definition of inverse functions.

The reduce rule that takes account of syntactic parsing with a generic rule can be described as follows:

$$\textbf{Reduce} \quad \frac{[\alpha x_1 x_2 \bullet, j]}{[\alpha \mathcal{SYN}(x_1, x_2)(x_1, x_2) \bullet, j]} \quad \mathcal{SYN}(x_1, x_2)(x_1, x_2) \neq \perp \tag{7}$$

The expression $\mathcal{SYN}(x_1, x_2)(x_1, x_2)$, with both arguments repeated, may seem confusing. Note, however, that each couple of arguments plays very different roles. The first couple are the arguments of the dynamic binding function, from which an effective partial rule is obtained. Such partial rule is then applied to the second set of arguments. They are related to the informational domain associated to the generic rule, which is, in this case, the type information associated to the linguistic objects.

The side condition, in this case, is precisely the one that licenses the syntactic combining operation. If $\mathcal{SYN}(x_1, x_2)(x_1, x_2)$ does not return a positive result, then we cannot build a phrase out of the arguments x_1, x_2.

The computational cost of parsing processes associated to this kind of generic rules carrying on syntactic information is obviously the same as the cost of parsing a context-free grammar. The only difference between both processes is the way to select the appropriate rule when the Reduce step is called. In context-free parsing, this step involves looking up the available rules and matching the objects involved with the right-hand side of the rules. In the worst case, this process introduces a factor G in the overall complexity of parsing, being G the size of the

[3]This would be the case of semantic disambiguation.

grammar. For a generic rule, the reduce rule involves introducing a new type in the hierarchy of rules. Again, this implies a factor G in the overall complexity, being G the number of partial rules associated to the generic rule. The dependence with the length of the string is obviously the same, as the surface behaviour of a context-free grammar and a generic rule is exactly the same for bottom-up parsing.

Another interesting case is when we have a generic rule to specify syntactic restrictions, and another one to perform semantic interpretation. It is easy to specify an algorithm to parse such grammars from the preceding cases. Now the elements of the logic have the form $[\alpha \bullet, j, \Phi]$, and the parser includes the following reduce rule:

Reduce

$$\frac{[\alpha x_1 x_2 \bullet, j, \Phi\phi_1\phi_2]}{[\alpha\mathcal{SYN}(x_1,x_2)(x_1,x_2) \bullet, j, \mathcal{SEM}(x_1,x_2)(\phi_1,\phi_2)]} \quad \begin{array}{l} \mathcal{SYN}(x_1,x_2)(x_1,x_2) \neq \perp \\ \mathcal{SEM}(x_1,x_2)(\phi_1,\phi_2) \neq \perp \end{array} \tag{8}$$

Let us see a (linguistically weird) example. Consider the grammar made up of the following generic rules:

$$\text{SYN} = \begin{cases} \text{SYN}_{NP\otimes VP}(X,Y) = S \\ \text{SYN}_{NP\otimes VP_i}(X,Y) = S_i \end{cases} \tag{9}$$

$$\text{SEM} = \begin{cases} \text{SEM}_{NP\otimes VP}(X,Y) = sem(X)(sem(Y)) \\ \text{SEM}_{Proper\text{-}Noun\otimes VP}(X,Y) = sem(Y)(sem(X)) \wedge (\lambda P.P(\text{Pete}))(sem(Y)) \end{cases} \tag{10}$$

The syntactic rule has already been considered in section 2.1. We assume the same hierarchy of that example, augmented with a new type ***proper-noun*** $\leq$ ***NP***. The semantic generic rule takes account of a very special semantic issue, namely that Pete has a weak personality and imitates his friends in everything they do. If we consider the phrases

- ***Betty*** : $\lambda P.P(\text{Betty})$; ***Betty*** $\leq$ ***proper-noun***
- ***got+angry*** : $\lambda x.\text{ANGRY}(x)$; ***got+angry*** $\leq VP_i$

the application of the deductive system above to the string '*Betty got angry*' would be as follows:

1. $[\bullet, 0,]$
2. $[\textit{Betty} \bullet, 1, \lambda P.P(\text{Betty})]$ **(Shift)**
3. $[\textit{Betty}\ \ \textit{got+angry} \bullet, 2, \lambda P.P(\text{Betty}), \lambda x.\text{ANGRY}(x)]$ **(Shift)**
4. $[S_i \bullet, 2, \text{ANGRY(Betty)} \wedge \text{ANGRY(Pete)}]$ **(Reduce)**

In the only application of the reduce rule, the following terms were used:

$\mathcal{SYN}(\textit{Betty},\textit{got+angry})(\textit{Betty},\textit{got+angry}) = \text{SYN}_{NP\otimes VP_i}(\textit{Betty},\textit{got+angry}) = S_i$

$\mathcal{SEM}(\textit{Betty},\textit{got+angry})(\lambda P.P(\text{Betty}, \lambda x.\text{ANGRY}(x)) =$

$\text{SEM}_{Proper\text{-}Noun\otimes VP}(\lambda P.P(\text{Betty}, \lambda x.\text{ANGRY}(x)) = \text{ANGRY(Betty)} \wedge \text{ANGRY(Pete)}$

This example illustrates the point, stated before, that it is not possible to attach a semantic rule to each syntactic rule at compile time. It is, essentially, a dynamic binding process.

A phrase-structure grammar with a one-to-one correspondence between syntactic and semantic rules would need 12 pairs of rules to get the same behaviour as the two generic rules above. The incrementality and modularity of the generic rule approach is evident in this case.

3 Non-Constituent Coordination and Categorial Grammar

The general scheme for coordination corresponds to the conjunction of constituents belonging to the same type: '*Nothing is certain, except death and taxes*', '*Take the money and run*', '*the long and winding road*'. Within the categorial grammar framework, such cases are solved using the basic function application rules (corresponding to the non-associative Lambek calculus in a sequent calculus presentation). Nevertheless, there are cases of the so-called *non-constituent coordination*, where the conjoined expressions are not constituents in the classical sense: '*John met Jane yesterday and Chris today*', '*John read a book about linguistics on Monday and a journal about computers on Tuesday*' (Left-node raising) , '*John made and Peter painted a wooden chair* (Right-node raising). The most common solution is to postulate extra-grammatical levels of representation and/or special purpose parsing algorithms.

There have been a number of proposals within the categorial framework, however, that deal with such phenomena at a grammatical level, extending the number of rules or the set of basic operators. The combinatory rules of type raising and functional composition [Steedman, 1985], for instance, introduce associativity in the structural resources of the grammar. [Dowty, 1988] uses these rules to assign a category to the coordinated conjuncts. With the exception of [Wittenburg and Wall, 1990], that proposes a (less intuitive) normal form for such rules that avoid spurious ambiguity, all the proposals suffer from intractability of the parsing task. Significantly, none of them includes, to our knowledge, a formal differentiation between default and exceptional rules.

We will consider here a particularly simple account for non-constituent coordination phenomena based on the *sequence product* operator introduced in [Solías, 1993]. Its reformulation as a generic rule will show the advantages of expressing default and exceptional grammar rules.

In the non-constituent coordination examples above, we may consider '*Jane yesterday*', '*a book about linguistics on Tuesday*', etc, as being tuples of expressions belonging to the tuple product. [4]. In this case the conjunction scheme type $(x \setminus x)/x$ would substitute a sequence product by the variable x.

In order to introduce this operator we need to extend the basic string algebra of types by adding a tuple operation. Thus the algebra would be $(L^*, +, \langle ., . \rangle)$, being $+$ the concatenative and associative operation and $\langle ., . \rangle$ the operation of tuple formation.

The model-theoretic definition for the sequence product is as follows:

$$D(\langle A, B \rangle) = \{< x_1, x_2 >: x_1 \in D(A), x_2 \in D(B)\} \tag{11}$$

The sequent rules corresponding to 11 are:

$$\frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow \langle A, B \rangle} R\langle\rangle \qquad \frac{\Gamma, A, B, \Delta \Rightarrow C}{\Gamma, \langle A, B \rangle, \Delta \Rightarrow C} L\langle\rangle \tag{12}$$

Examples with more than two elements in the sequence product will need a generalization of the tuple operation. This generalization is straightforward using the standard definition of n-tuple: $< x_1, ..., x_n >=<< x_1, ..., x_{n-1} > x_n >$. Now we are able to account for a sentence like '*John read a book about linguistics on Monday and a journal about computers on Tuesday*' by using a 3-tuple $\langle np, pp, (np \setminus s) \setminus (np \setminus s) \rangle$. Right-node raising examples proceed in a similar way.

This approach avoids using type-raising (a rule that can be applied on any category at any time), but still suffers from intractability, as it is available for every combination of types. Again, the problem relies on the exceptional status that should be given to the rules that deal with non-canonical phenomena.

[4]For these simplest cases of Non-constituent coordination we may also use Lambek's associative product. However the introduction of the sequence product brings advantages in more intricate examples of coordination like the ones considered in [Solías, 1993]. Therefore we will use the sequence product in consideration of further extensions of this work.

4 A Generic Rule to Parse Non-Constituent Coordination

The framework of generic rules offers a natural way to express the grammar to deal with non-constituent coordination as an arrangement of default rules, where each combination of types is performed according to the most specific rule available.

The essential rules we want to express are:

- *By default, two types are combined using functional application and, if functional application is not applicable, they cannot be combined.*
- *When we try to combine two or more verbal complements and there is a conjunction inmediately preceding them, a sequence product can be formed with them.*
- *When we try to combine a noun phrase and a verb followed by a conjunction, a sequence product can be formed with them.*

"Verbal complement", for our present purposes, stands for a $\boldsymbol{np}$, a $\boldsymbol{vm}$ or, in turn, a sequence product [5].

Such rules would guarantee that a sequence product is formed only in the relevant cases. We only need an additional rule to scan optimally the elements that match the sequence product on the left-side of the coordination.

The formalism of generic rules allows for a direct specification of such set of - still very informal - rules. Once turned into a generic rule, they can be parsed with any bottom-up context-free recognition algorithm (reformulated as the shift-reduce parsing in section 2.4).

The first step is to express the operations related to the grammar as binary rules:

$$\begin{array}{ll} \mathbf{fa} : X/Y \quad Y \to \quad X & \mathbf{I}_{\langle\ldots\rangle} : conj \quad X \quad Y \to \quad conj \quad \langle X, Y\rangle \\ \mathbf{ba} : Y \quad Y\backslash X \quad \to \quad X & \mathbf{D}_{\langle\ldots\rangle} : \langle X_1 \cdots X_n\rangle \to X_1 \cdots X_n \end{array} \tag{13}$$

$$\mathbf{scan} : X_n \quad \langle\langle X_1 \cdots X_{n-1}\rangle\, X_n\rangle\backslash\langle X_1 \cdots X_m\rangle \to \langle X_1 \cdots X_{n-1}\rangle\backslash\langle X_1 \cdots X_m\rangle$$

fa and **ba** are the usual forward and backward application rules. $\mathbf{I}_{\langle\ldots\rangle}$ is the rule to introduce a sequence product. As stated, it seems a context-sensitive rule: The formation of a sequence product is only possible when a conjunction is present to the left of the elements that form a sequence product [6]. However, this context-sensitivity does not overcome context-free grammar parsing, as the licensing element is a lexical item, a terminal, and its presence can be checked in constant time. $\mathbf{I}_{\langle\ldots\rangle}$ rule implements the stepped application of rules $L/$ and $R\langle\rangle$ in the cancellation of type $(\langle A,\ B\rangle \setminus \langle A,\ B\rangle)/\langle A,\ B\rangle \quad A,\ B$. **scan** matches elements to the left of the conjunction with the items in the tuple built to the right. This reformulation is intended to a) keep the binary rules arrangement to allow easy formulation of the generic rule, and b) take into account the asymmetry between the formation process for the right and the left coordinated conjuncts [7] . The necessity of bulding a tuple is given by the right conjunct, and the left conjunct is built only to match the right one. **scan** is implementing the consecutive application of $L\backslash$ and $R\langle\rangle$. $\mathbf{D}_{\langle\ldots\rangle}$ is the rule to eliminate the tuple operator and it implements the $L\langle\rangle$ rule.

The crucial point to write a generic rule based upon the rules above is to determine the types of the arguments associated to each operation. By default, any combination of categories is driven by functional application rules: forward application (**fa**) and backward application (**ba**). Therefore, the first partial rule would be licensed on arguments of the most general type $\boldsymbol{T}$. This rule will try to apply **fa** or **ba** , and will return $\perp$ if both fail, forbidding that combination.

[5]We use $\boldsymbol{vm}$ as an abbreviation for $(\boldsymbol{np}\backslash \boldsymbol{s})\backslash(\boldsymbol{np}\backslash \boldsymbol{s})$. In this application we are only taken under consideration the possibility of having types $\boldsymbol{np}$ and $\boldsymbol{vm}$ as verbal complements. This set should be extended if some other kind of complementation were considered.

[6]As [Solías, 1992] pointed out, the conjunction licenses type sequences. Therefore, sequence product can appear in coordination environments, whereas other categories do not.

[7]The stepped cancellation of the right and the left conjuncts has well-known linguistic motivation, first stated in [Ross, 1967].

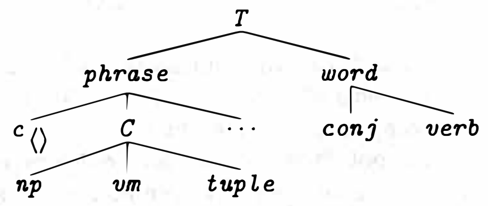
The second partial rule is the rule of tuple formation. The type conditions over the arguments of such a rule arise in a natural way from the informal specification made at the beginning of this section: both tupled elements have to be verb complements (in the case of left-node raising) or an np followed by a verb (in the case of right-node raising). We need two partial rules to establish both type specifications. We will consider a type C that has $\boldsymbol{np}$ and $\boldsymbol{vm}$ as direct subclasses. Another possibility is that one of the tupled members is, in turn, a tuple. Therefore, the tuple has to be introduced as a direct subclass of C as well. That dependency provides the possibility of building n-tuples.

Finally, we need a scan rule to match objects at the left of the conjunction once the right tuple has been combined with the conjunction (by means of simple forward application). Therefore, the type constraint on right elements is type C again. The second one has to be a coordination of tuples missing some elements to its right. We will denote this type as $\mathbf{c}_{\langle\rangle}$. This rule has to act in coordination with $\mathbf{D}_{\langle\ldots\rangle}$, which will be applied only after the last conjoined element in the right coordinated conjunct has been cancelled.

To establish the interaction between the rules of 13 and the partial rules, we will use the following operations of composition, disjunction and optionality:

- $r \circ p(x, y) \equiv r(p(x, y))$
- $r \vee p(x, y) \equiv \begin{cases} r(x,y) \text{ if } r(x,y) \neq\perp, p(x,y) =\perp \\ p(x,y) \text{ if } p(x,y) \neq\perp, r(x,y) =\perp \\ \perp \text{ if } p(x,y) =\perp, r(x,y) =\perp \end{cases}$
- $[r](x) \equiv \begin{cases} r(x) \text{ if } r(x) \neq\perp \\ x \text{ if } r(x) =\perp \end{cases}$

That is the fragment of hierarchy that we need for our present purposes:



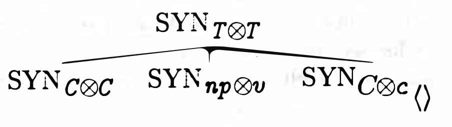
The type C represents verb complements, as introduced above. The type $c_{\langle\rangle}$ represents the combination of a conjunction with a tuple as its right coordinated conjunct. It is needed to specify the scan rule over the appropriate kind of objects. The type ***tuple*** is included as a verb complement, to allow formation and coordination of n-tuples.

Given that hierarchy, we propose the following generic rule to parse sentences including non-constituent coordination:

$$\mathrm{SYN} = \begin{cases} \mathrm{SYN}_{T\otimes T}(X, Y) = (\mathbf{fa} \vee \mathbf{ba})(X, Y) \\ \mathrm{SYN}_{C\otimes C}(X, Y) = \mathbf{I}_{\langle\ldots\rangle}(X, Y) \\ \mathrm{SYN}_{\boldsymbol{np}\otimes \boldsymbol{v}}(X, Y) = \mathbf{I}_{\langle\ldots\rangle}(X, Y) \\ \mathrm{SYN}_{C\otimes \mathbf{c}_{\langle\rangle}}(X, Y) = (\mathbf{D}_{\langle\ldots\rangle} \circ \mathbf{scan})(X, Y) \end{cases} \tag{14}$$

Note, again, that no precedence has to be defined by the grammar writer to control the interaction of the rules. An easy, natural analysis of the suitable arguments for each rule has implicitly defined a partial ordering between them.

The Hasse diagram of the cartesian poset associated to that generic rule is:



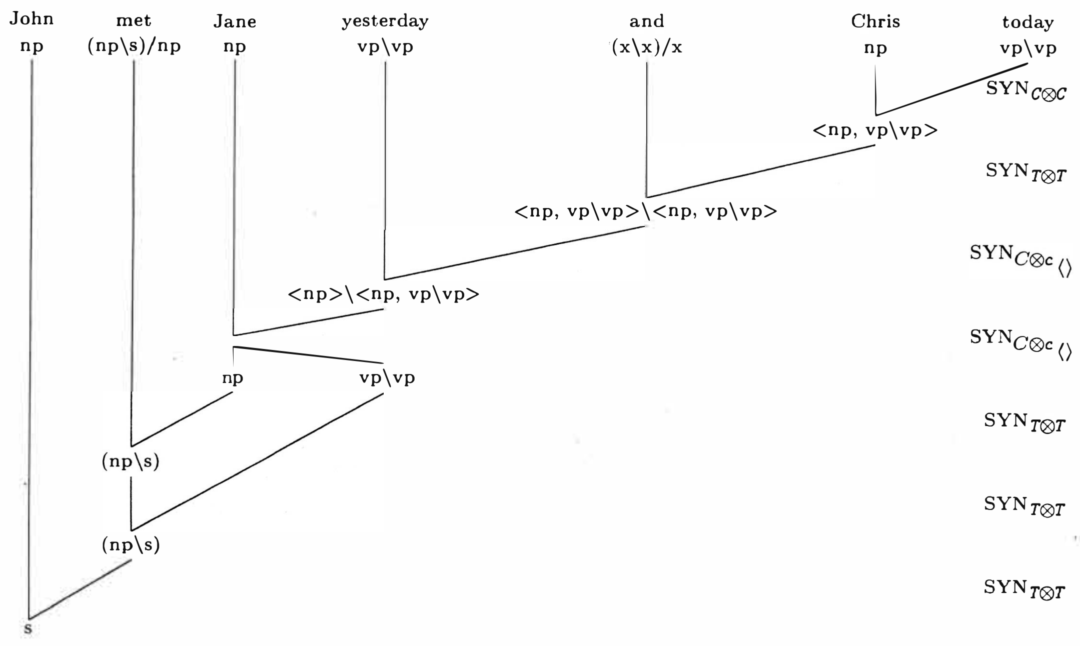


Figure 2: Analysis of "John met Jane yesterday and Chris today" according to SYN.

This rule can be parsed with the algorithm in 7. The only novelty is that some word indexing has to be kept so that the presence of a conjunction inmediately to the right of the arguments can be checked in order to apply SYN$_{C\otimes C}$ and SYN$_{np\otimes v}$.

The Figure 2 shows how dynamic binding works to get an analysis of '*John met Jane yesterday and Chris today*'. We have annotated each step of the analysis with the partial rule effectively applied on the constituents being combined. That partial rule is signalled by the dynamic binding function as the most specific to combine the types involved in the combination process.

A context-free parser with the modifications shown in section 2.4 for the **reduce** step can produce this analysis at a context-free cost in time, both in the length of the string and the size of the grammar. Compared with the calculus presented in section 3, the generic rule does not suffer from intractability and can be parsed with well-known, general parsing techniques.

A parser with heuristics or daemons to control coordination processes could achieve a similar efficiency. The clear advantage of a generic rule is that the knowledge that reduces the search space is declaratively introduced at the grammar level, and controlled at an intermediate level between the grammar and the parser (by means of dynamic binding). This enhances linguistic motivation, modularity and incrementality (both to extend the grammar and to control the parsing processes).

5 Conclusions

The possibility of formally stating grammar rules with a default interpretation has some advantages from a parsing perspective and from the point of view of knowledge representation. On one side, it provides a declarative and modular way to reduce the search space of parsing processes without altering parsing algorithms with heuristic recipes. On the other side, it provides a linguistically motivated account of exceptional behaviour that is particularly appealing for lexicalized grammar formalisms where the lexicon is the repository of most of the linguistic information, and there are only a few, very general rules that govern linguistic phenomena.

Our account of non-constituent coordination illustrates the advantages of such default arrangements for grammar rules. We have presented a categorial account of non-constituent coordination in which incombinable constituents on both sides of conjunction are treated as tuples of elements by the introduction of a sequence type in the conjunction type. Once we have formed a single tuple constituent, we can combine it with the remaining elements. This approach of non-constituent coordination within Categorial Grammar needs some rules of introduction and elimination of the sequence operator which are not commonly needed for other simpler linguistic phenomena, and that makes the parsing process intractable in its original Lambek-style formulation. When reformulated as a generic rule, the type conditions on the arguments for each rule are used by the dynamic binding process to fire the most appropriate grammar rule at every parsing step. The process turns to be context-free parsable, and exhibits a highly restricted search space.

References

[Bouma, 1992] Bouma, G. (1992). Feature structures and nonmonotonicity. *Computational Linguistics, 18-2.*

[Carpenter, 1993] Carpenter, R. (1993). Skeptical and credulous default unification with application to templates and inheritance. In *Inheritance, Defaults and the Lexicon.* Cambridge University Press.

[Daelemans et al., 1992] Daelemans, W., De Smedt, K., and Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics, vol 18-2.*

[Dowty, 1988] Dowty, D. (1988). Type raising, functional composition, and non-constituent conjunction. In *Categorial Grammars and Natural Language Structures.* Reidel Publishing Company.

[Gonzalo, 1995] Gonzalo, J. (1995). An object-oriented approach to treat exceptions in grammar. In *Grammar formalisms for NLP.* University of Tuebingen Seminar für Sprachwissenschaft technical report series, to appear.

[Morrill and Solias, 1993] Morrill, G. and Solías, T. (1993). Tuples, discontinuity and gapping in categorial grammar. In *EACL-93.*

[Ross, 1967] Ross, J. (1967). *Constraints on variables in Syntax.* PhD thesis, MIT.

[Shieber et al., 1994] Shieber, S., Schabes, Y., and Pereira, F. (1994). Principles and implementation of deductive parsing. Technical Report TR-11-94, Center for Research in Computing Technology, Harvard University.

[Solías, 1992] Solías, M. (1992). *Gramáticas Categoriales, Coordinación Generalizada y Elisión.* PhD thesis, Universidad Autónoma de Madrid.

[Solías, 1993] Solías, T. (1993). Sequence product, gapping and multiple wrapping. In *Proceedings of the workshop on Linear Logic and Lambek Calculus.*

[Steedman, 1985] Steedman, M. (1985). Dependency an coordination in the grammar of dutch and english. *Language,* 61(3).

[Wittenburg and Wall, 1990] Wittenburg, K. and Wall, R. (1990). Parsing with categorial grammar in predictive normal form. In *Current Issues in Parsing Technologies.* Kluwer Academic Publishers.