From Intuitionistic Proof Nets to Interaction Grammars

Guy Perrier

From Intuitionistic Proof Nets to Interaction Grammars

GUY PERRIER
LORIA - Université Nancy 2
Campus Scientifique - B.P. 239 -
54506 Vandœuvre-les-Nancy Cedex - France
E-mail: perrier@loria.fr

September 24, 1999

Abstract

From intuitionistic proof nets of linear logic, we abstract an order between axiom links and we use it for representing proof nets in the more compact form of precedence trees. This provides us with new formal tools for revisiting grammatical formalisms and leads us to introduce Interaction Grammars, which synthesize Categorial Grammars and Phrase Structure Grammars.

Introduction

Even if Categorial Grammars [Adj35, BH53, BH64, Lam58, Lam61] precede linear logic [Gir87] historically, they illustrate the power of this approach to logic for expressing the resource sensitivity of the syntax of natural languages. This does not mean that all syntactic aspects of natural languages can be encoded in linear logic. It would even be dogmatic to pretend this. On the other hand, the study of linear logic can help us with the difficult task of designing grammatical formalisms which are sufficiently subtle for capturing whole the complexity of natural language. This is the point of view from which we consider the relationship between grammar and linear logic.
Resource sensitivity of linear logic entails a specific form of proof: *proof nets* [Gir87]. In the general framework of classical linear logic, these proof nets are not directed so that each extremity of a proof net can be viewed as either an input or an output; in other words, each formula that is attached to an extremity of a proof net can be considered either as an assumption (input) or as a conclusion (output) of the proof.
If we restrict ourselves to *intuitionistic linear logic* (ILL), things freeze in a configuration where all formulas become *polarised*, one as the output and the others as the inputs. In particular, the assumptions of the proof are fixed once and for all and there is a unique conclusion which is also fixed once and for all, so that every proof net becomes directed from its assumptions to the conclusion. Lamarche has devised a correctness criterion for these proof nets which takes their specificity into account [Lam94]. This criterion is based on particular paths inside intuitionistic proof nets. In [Lam94], these paths go from the conclusion of the proof net to all assumptions, whether they are actual or cancelled. We use again these paths but in the opposite direction and we call them *precedence paths* because they induce an order between the atomic formulas of each proof net.
From the precedence paths of a proof net, Lamarche has sketched a more compact representation of proof nets which only takes the induced order between atomic formulas into account[1]. We propose to perfect this representation and to make it more suitable for theorem proving in linear logic. Since we have chosen the framework of proof nets, theorem proving amounts to building proof nets. We propose to associate what we call a *precedence graph* to every proof net in construction. This graph expresses what is essential

[1] This representation was introduced in a private communication and did not give rise to publication.

in the proof net, the order between atomic formulas induced by precedence paths so that the problem of proof net construction is reduced to that of transforming a specific graph into a specific tree by merging some of its vertices.
We could take advantage of this approach for theorem proving in ILL but this is not our purpose in this paper where we are interested in linguistic issues. The notion of precedence tree reveals the deep relationship between two classes of formalism that are often used for representing the syntax of natural languages:

- *Phrase Structure Grammars* which manipulate trees representing the recursive decomposition of phrases into their immediate constituents; Tree Adjoining Grammars (TAGs) [JLT75] are a well known instance of these grammars;
- *Categorial Grammars* which are characterised as calculi of syntactic types; originally, these calculi were purely algebraic but then they were viewed as proofs in particular logic systems; the original reference of these systems is the Lambek Calculus [Lam58].

Recent research has revealed the links between these two approaches: Abrusci, Fouqueré and Vauzeilles present TAG in the form of a deductive system in a fragment of non-commutative intuitionistic linear logic [AFV97] while Joshi and Kulick [JK97] have a similar proposal that uses partial proof trees in natural deduction.
From the notion of precedence tree associated with a proof net, we also aim to find a synthesis between Phrase Structure Grammars and Categorial Grammars. The grammatical formalism that we propose and which we call *Interaction Grammars* (IGs) can be viewed as a variant of Phrase Structure Grammars because it associates a tree to every syntactic constituent in order to represent its structure. Its originality lies in the fact that each syntactic tree, which is inert in classical phrase structure grammars, is made active by polarising its nodes:

- a negative node represents a constituent which is expected; it expresses a valence of its mother node which has to be filled;
- a positive node represents an effective constituent which is able to fill a valence of another constituent.

In this way, a syntactic tree becomes able to interact with others in order to build more complex trees, from this stems the name of Interaction Grammars. The elementary operation through which interaction happens consists in merging two dual nodes into a unique neutral node.
IGs are connected to Categorial Grammars because building the syntax of phrases can be viewed as a logical calculus. Syntactic trees with their polarised nodes represent precedence trees associated to incomplete proof nets. The interaction of these trees which ends in a unique tree representing the syntax of a whole sentence, is then interpreted as the combination of partial proof nets in a unique and complete proof net. This process results from the iteration of the elementary operation of merging two dual nodes of the syntactic trees, which amounts to adding an axiom link to the corresponding incomplete proof net. This briefly describes the skeleton of IG. Since this skeleton is expressed in the commutative logic framework of IILL, it only aims to express the linearity of dependencies between syntactic constituents of a sentence, however long their distance is and independently of their relative order. This last point is a major divergence with respect to the usual approach of Categorial Grammars and Phrase Structure Grammars. Instead of linking word order and dependency between constituents, we dissociate these two aspects and we refer word order to the same level as morphological information. The observation that the same linguistic properties are expressed in some languages by word order and in others by morphology leads us to this uniform treatment.
For representing extra-logical linguistic information, we label the nodes of syntactic trees with feature matrices, so that the basic objects of the formalism become *polarised syntactic trees labelled with feature matrices*. Then, the elementary parsing operation of merging dual nodes is replaced with unification of feature matrices labelling nodes with opposite polarities. We meet again the basic mechanism of Unification Grammars as it appears in LFG [Bre82] or HPSG [PS94] but with major differences. In IG, the dependency graph between grammatical constituents is external to the feature system and its nodes are

polarised whereas in LFG and in HPSG the graph is integrated in the feature system and it is devoid of polarisation.

The paper is divided into two sections:

- In Section 1, we study proof nets from the purely proof-theoretic viewpoint, independently of their linguistic applications from which we draw the notion of precedence tree.
- In Section 2, we apply this notion to designing a new grammatical formalism, IG, which we compare with other existing systems. In particular, we will see how IGs are very close to D-Tree Grammars [RVSW95]. This is all the more striking since the two formalisms have completely independent origins: IGs come from linear logic and its application to linguistics while D-Tree Grammars derive from TAGs.

1 Proof nets and precedence trees

As Lamarche has shown [Lam94, Lam96], the correctness of ILL proof nets can be characterised in a way that is specific to the intuitionistic framework. From this criterion, he has outlined a more compact representation of proof nets which only takes into account the order that is inferred by a proof net on its axiom links. We propose to develop this representation.
Since, in linguistic applications, we only use the implicative fragment of propositional intuitionistic linear logic (IILL), we restrict ourselves to this fragment but, at the end of the section, we show how the results can be easily extended to all of the multiplicative fragment (IMLL).

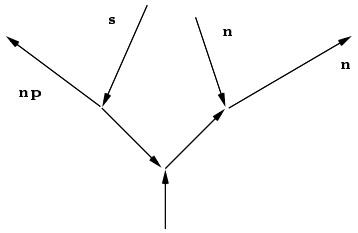
1.1 IILL proof nets

In this subsection, we recall Lamarche's criterion for IILL proof nets. Let $\mathcal{P}$ be a set of propositions. The set of IILL formulas built from $\mathcal{P}$ is defined by the grammar $\mathcal{F} ::= \mathcal{P} \mid \mathcal{F} \multimap \mathcal{F}$.
The IILL sequent calculus handles sequents that are constituted of IILL formulas and that have exactly one formula in their succedent. Cut free proofs have the sub-formula property: in a bottom up proof construction of a sequent $F_1, \cdots, F_n \vdash G$, a formula F that occurs following a decomposition, is a sub-formula of one of the formulas $F_1, \cdots, F_n, G$. Let us call this formula F'. The position of F on the left or the right hand side of the sequent where it occurs only depends on its syntactic relation with F' and on the position of F' in the conclusion sequent.
We take this property into account with the notion of *polarised formula.* A polarised formula is an IILL formula that is provided with a polarity – or + to indicate whether it belongs to the succedent or to the antecedent of a sequent. In other words, a positive formula is a resource to be consumed and a negative formula is a resource to be produced in a proof. Rather than the opposite convention, we have chosen this one because positive expresses something which is actual whereas negative expresses something which is virtual and which expects its contrary to be neutralised, even if this convention contradicts the fact that negative formulas are translated into negations in the one-sided sequent calculus.
From the polarity of a formula, we compute the polarity of its sub-formulas by means of two rules, one for the negative implication and one for the positive implication. If we represent a positive formula by an up arrow and a negative formula by a down arrow, the rules can be represented by the following schemes.

By iteration, we define the *syntactic tree of a polarised formula.* It is constituted of two kinds of links: *positive links* which correspond to the decomposition of positive implications and *negative links* which

correspond to the decomposition of negative implications.

Example 1.1 *Here is the syntactic tree of the positive formula* $((np \multimap s) \multimap n \multimap s)^+$.



From an IILL sequent $F_1, \cdots, F_n \vdash G$, we build the forest of the syntactic trees of its polarised formulas $F_1^+, \cdots, F_n^+, G^-$. This constitutes an initial *proof frame*. Then, to prove the sequent, we have to connect each negative leaf with a positive leaf of the same type by means of a link which is called an *axiom link* in proof net theory. When all leaves are connected, the proof frame becomes a *proof structure*. The inputs of the proof structure represent the positive formulas $F_1^+, \cdots, F_n^+$ and its unique output represents G^-. Let us define the notions of proof frame and proof structure more precisely.

Definition 1.1 *An IILL proof frame is a forest of syntactic trees of polarised IILL formulas with exactly one positive formula (its output) and possibly axiom links between dual atomic formulas.*
An IILL proof structure is an IILL proof frame all of whose all atomic formulas are connected in pairs by axiom links.

Of course, not all proof structures are proof nets. Among the variety of correctness criteria for proof nets, we have chosen that of Lamarche because it uses the specific properties of intuitionistic proof nets and it is well suited to linguistic applications. This criterion uses particular paths in proof nets, which we call *precedence paths*.

Definition 1.2 *In a proof frame, a precedence path is a path in the usual sense for directed graphs that include no positive premises of negative links except possibly as its beginning.*

According to this definition and the syntax of proof structures, there is exactly one maximal precedence path that starts from a given formula. Either this path is infinite and includes one cycle or it ends at the output of the proof structure.

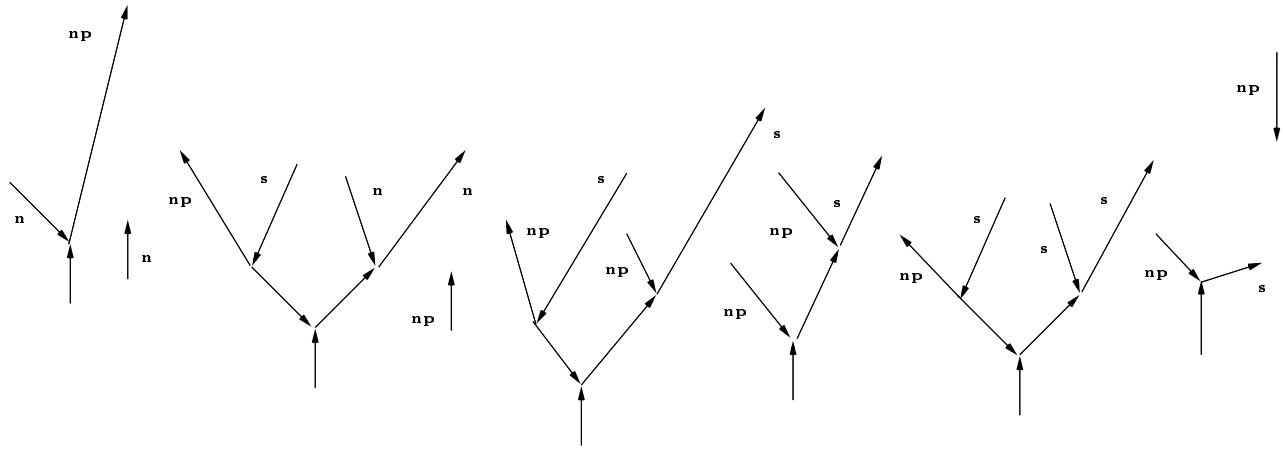
Definition 1.3 *A proof structure is a* proof net *if there is a precedence path from every formula occurrence to the output and if the path from the positive premise of any negative link to the output includes the negative premise.*

In this definition, the first condition can be restricted to positive links and reformulated as follows: in a proof net, the maximal path that starts from a positive premise of a positive link does not lead to the negative premise of the same link. The second condition, which applies to negative links, can be formulated as the contrary of the first condition: in a proof net, the maximal path that starts from a positive premise of a negative link necessarily leads to the negative premise of the same link.

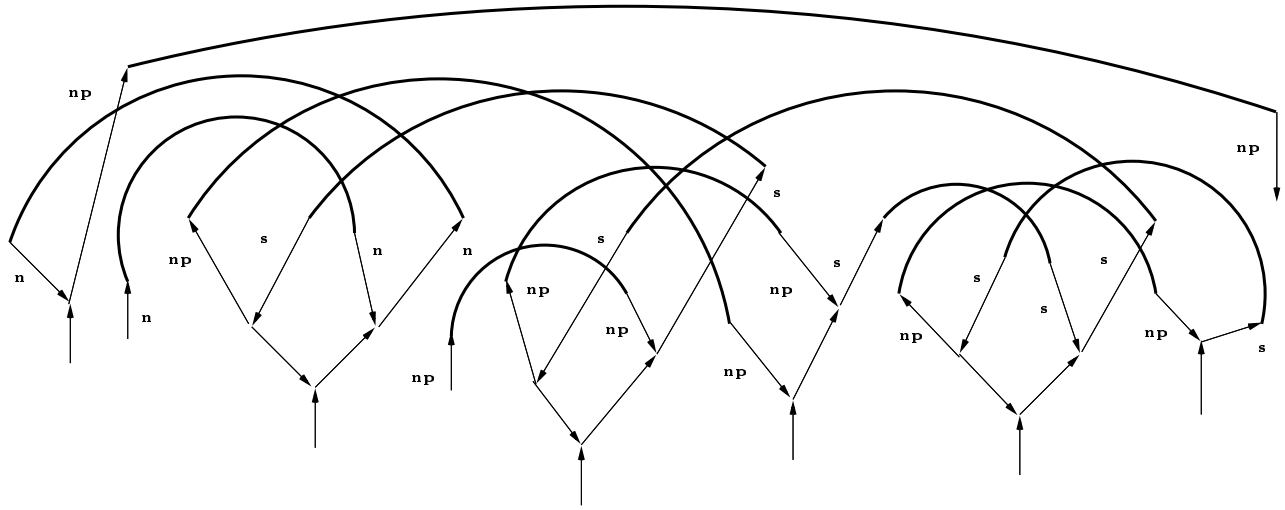
Example 1.2 *We consider the following sequent :*

$n \multimap np,\ n,\ (np \multimap s) \multimap n \multimap n,\ np,\ (np \multimap s) \multimap np \multimap s,\ np \multimap np \multimap s,\ (np \multimap s) \multimap s \multimap s,\ np \multimap s \vdash np$

The syntactic trees of these formulas constitute the initial proof frame which has the following form.



By adding axiom links, we obtain the proof structure below which is a proof net because it verifies Lamarche's criterion.



Our representation of IILL proof nets corresponds to the two-sided IILL sequent calculus: in a proof structure, the inputs correspond to the formulas on the left hand side of the associated sequent and the unique output correspond to the unique formula on the right hand side of the sequent. There is another representation of IILL using the one-sided sequent calculus with polarities of linear logic [Lam95]: in this representation, the negative and positive implications are respectively replaced with negative $\parr$-formulas and positive $\otimes$-formulas but there is no essential difference between the two representations.

1.2 Precedence tree of an IILL proof

From the notion of precedence paths in IILL proof nets, we abstract a more compact representation of IILL proofs which only takes atomic formulas into account. In a proof net, precedence paths infer an order between its atomic formulas which is used to build what we call the precedence tree of the corresponding IILL proof. Even if this order is not sufficient to re-build the corresponding proof net, it is sufficient to verify its correctness. Lamarche has outlined such a representation which we aim to make more precise and to develop here.

As we are interested in proof net construction, we have to start with a more general notion than that of precedence tree, the notion of *precedence graph* which describes the order between atomic formulas inferred by precedence paths in a proof frame. This notion is defined first syntactically, independently of its relationship with proof nets.

Definition 1.4 *A* precedence graph *is a directed graph with the following properties:*

- *the vertices are polarised or neutral atomic formulas;*
- *there are two kinds of edges:*
  - *actual links (drawn with continuous lines) the source of which is negative or neutral and the target of which is positive or neutral;*
  - *virtual links (drawn with dotted lines) the source of which is positive or neutral and the target of which is negative or neutral;*

A precedence tree is a precedence graph which is a tree.

The following proposition aims to establish a relationship between the notions of precedence graph and proof frame.

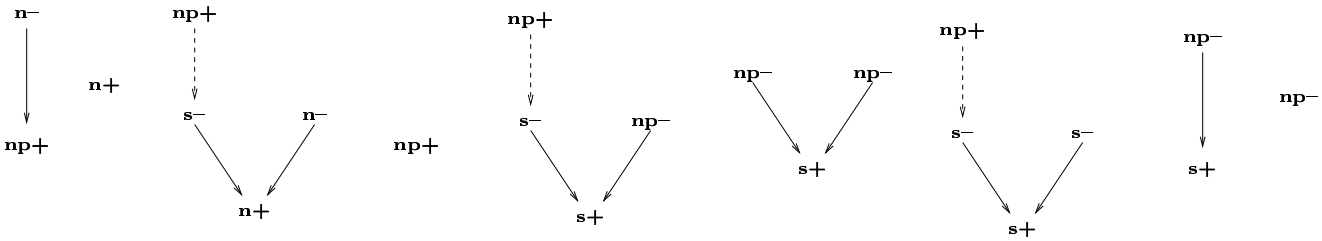
Proposition 1.1 *Let Φ be a proof frame and $P(\Phi)$ be the graph that is defined from Φ as follows:*

- *Its vertices are the occurrences of polarised atomic formulas in Φ and if two formulas A^+ and A^- are connected by an axiom link in Φ, their corresponding vertices P(A^+) and P(A^-) in P(Φ) are merged in a neutral vertex A.*
- *For every precedence path from a negative atomic formula A_1^- to a positive atomic formula A_2^+ in Φ which crosses no axiom links, an actual link from $P(A_1^-)$ to $P(A_2^+)$ is added to P(Φ).*
- *If there is no precedence path from the positive premise F^+ to the negative premise G^- of a negative link in Φ, let A_1^+ be the first atomic formula on a precedence path that starts from F^+ and let A_2^- be the last atomic formula on a precedence path that ends with G^- [2]. A virtual link from $P(A_1^+)$ to $P(A_2^-)$ is added to $P(\Phi)$.*

$P(\Phi)$ *is a precedence graph which is called* the precedence graph of Φ.

In P(Φ), the paths that only use actual edges, are a compact translation of the precedence paths that exist in Φ and the virtual edges represent the precedence paths that are necessary to create by adding new axiom links to transform Φ into a proof net.

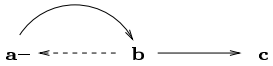
Example 1.3 *Here is the precedence graph of the proof frame given in Example 1.2.*



Proposition 1.1 shows how to map a proof frame into a precedence graph. The converse problem can be formulated as follows: for any precedence graph Θ, is there a proof frame Φ such that Θ is equal to $P(\Phi)$. In the framework of IILL, the answer is no.

Example 1.4 *A precedence graph with two negative vertices without outgoing edges is not the image of a proof frame because it contradicts the unicity of the output in a proof frame.*
A precedence graph with two actual edges outgoing from the same vertex is not the image of a proof frame because it contradicts the fact that a precedence path that stems from any formula occurrence is completely determined.
If we extend the logical framework to IMLL, the previous counter-examples fail but not the following one which is linked to the tree-like character of logical formulas.

[2] The unicity of these two precedence paths is due to the syntax of proof frames in IILL but this property no longer holds in IMLL.



Building a proof frame that corresponds to the precedence graph above fails because of an alternating cycle in this graph, that is, a cycle where actual edges and virtual edges alternate.

The following proposition expresses necessary and sufficient conditions for a precedence graph to be the image of a proof frame.

Proposition 1.2 *For a precedence graph Θ, there exists a proof frame Φ such that Θ is equal to $P(\Phi)$ iff it verifies the following conditions:*

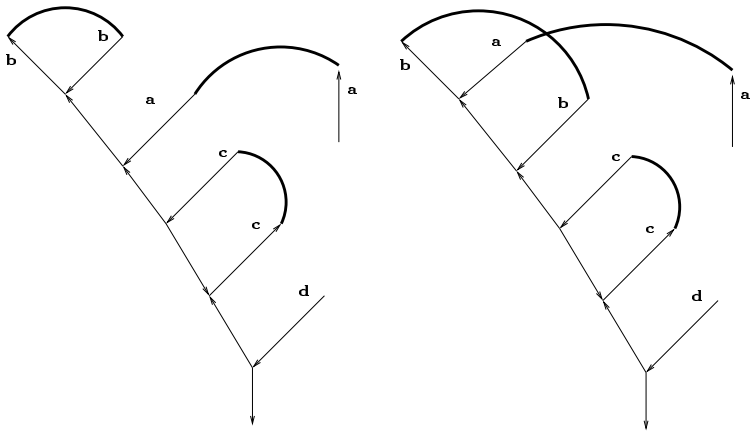
1. *it has no alternating cycle;*
2. *every virtual edge constitutes the unique path from its source to its target;*
3. *there is one link of each kind at most that starts from a vertex;*
4. *there is exactly one negative or neutral vertex without outgoing actual edges; it is called the* root *of the graph and furthermore it does not have an outgoing virtual edge.*

Proof 1.1 *The four conditions are easy to verify for the image of a proof frame. Now, we consider a precedence graph Θ that verifies the four conditions. Let us prove that Θ is the image of a proof frame by induction on the number n of neutral vertices in Θ.*
If n=0, because of Conditions 1 and 3, Θ is a forest of trees and because of Condition 4, exactly one tree has a negative root. We deduce that we can build a forest of syntactic trees of IILL formulas which has Θ as precedence graph.
If n>0, we dissociate a neutral vertex A into a negative vertex A^- and a positive vertex A^+ and we re-distribute the edges that have A as the source or the target to A^- and A^+ according to the constraints in Definition 1.4 of a precedence graph. The four conditions are still true for this precedence graph Θ'. By induction hypothesis, we associate a proof frame Φ' to Θ' and by adding an axiom link we obtain a proof frame Φ. Because of Condition 2, adding an axiom link does not entail deleting a virtual edge in the corresponding precedence graph, so that Φ has Θ as its precedence graph. □

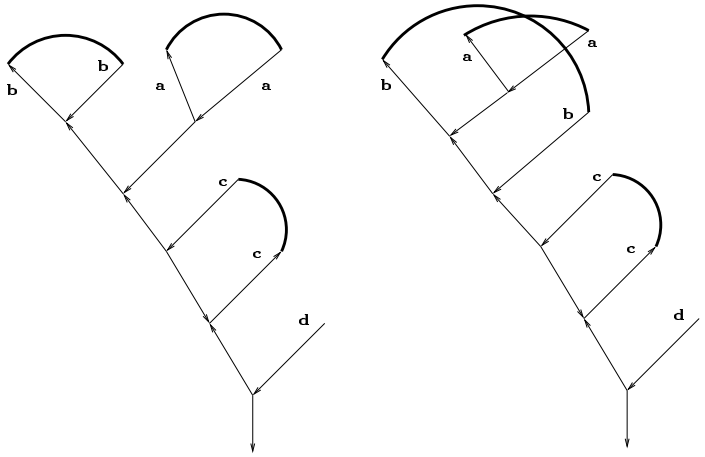
Now, if we know under what conditions a precedence graph is the image of a proof frame, we do not know if this proof frame is unique. In other words, when the inverse image $H^{-1}(\Theta)$ of a precedence graph Θ is not empty, is it reduced to a singleton ? The following example shows that this not always the case.

Example 1.5 *Let Θ be the following precedence graph.*

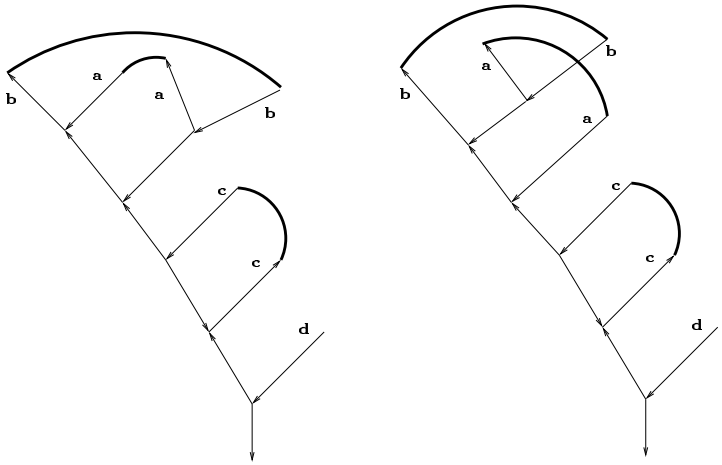
By using the method presented in the proof of Proposition 1.2, we obtain the two canonical elements of $H^{-1}(\Theta)$ below. They differ in the order between the arguments a and b of a linear implication.



These proof frames are canonical in the sense that they include a minimum number of negative links which correspond to the virtual edges of Θ *but there are four other elements of* $H^{-1}(\Theta)$ *which result from the addition of negative links to the canonical proof frames. We can first add a negative link between* $a+$ *and* $a-$*.*



We can alternatively add a negative link between $a+$ *and* $b-$*.*



From this example, it is easy to generalise the properties of $H^{-1}(\Theta)$. There is a canonical element Φ of $H^{-1}(\Theta)$. The other elements of $H^{-1}(\Theta)$ are obtained from Φ in two ways:

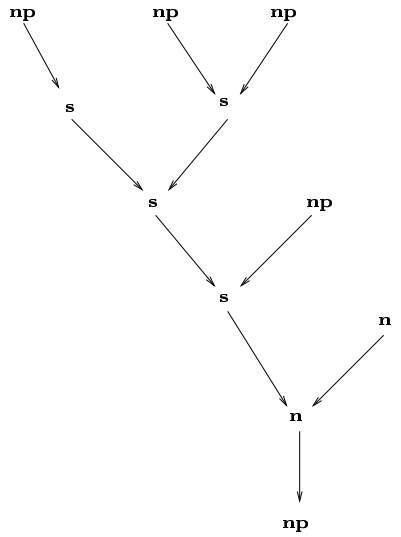
- adding negative links with inputs and negative formulas of Φ as premises which create no new virtual edge in Θ;
- permuting negative (positive) premises of two consecutive positive (negative) links.

If Θ includes no neutral vertices, $H^{-1}(\Theta)$ is a singleton. As the previous example shows, mapping a proof structure into a precedence graph implies a loss of information but the information that is essential to check its correctness is preserved. A proof net is characterised by the specific form of its precedence graph, that of a *terminal precedence tree.*

Definition 1.5 *A terminal precedence tree is a precedence tree without virtual links all nodes of which are neutral.*

Proposition 1.3 *A proof structure is a proof net iff its precedence graph is a terminal precedence tree. This is equivalent to the property that it includes no cycles and no virtual edges.*

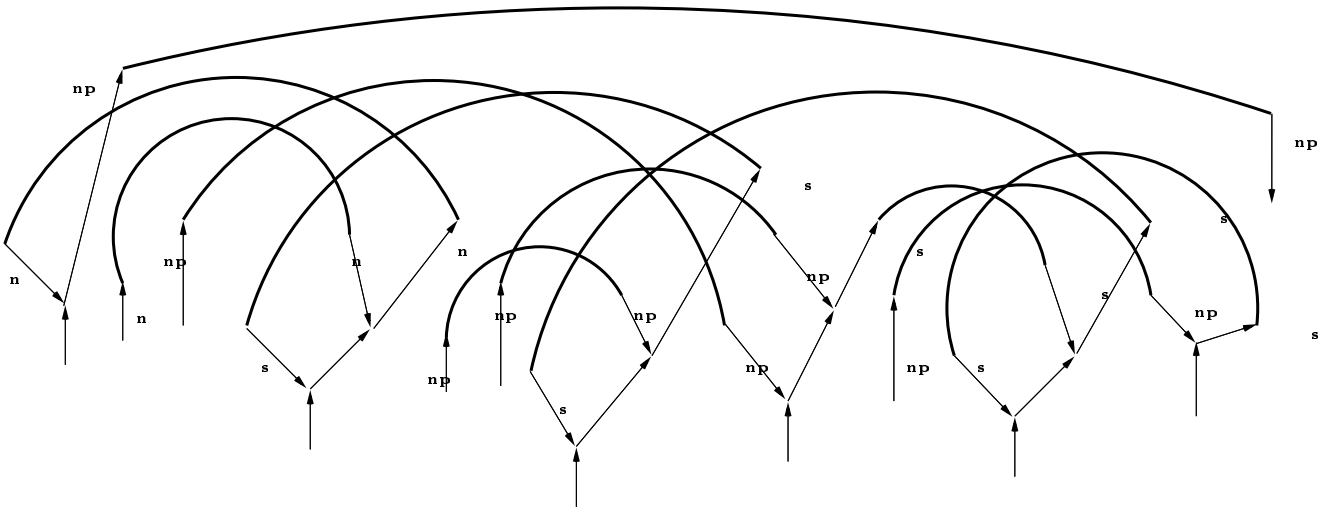
Example 1.6 *The precedence graph of the proof structure given in Example 1.2 has the following form.*



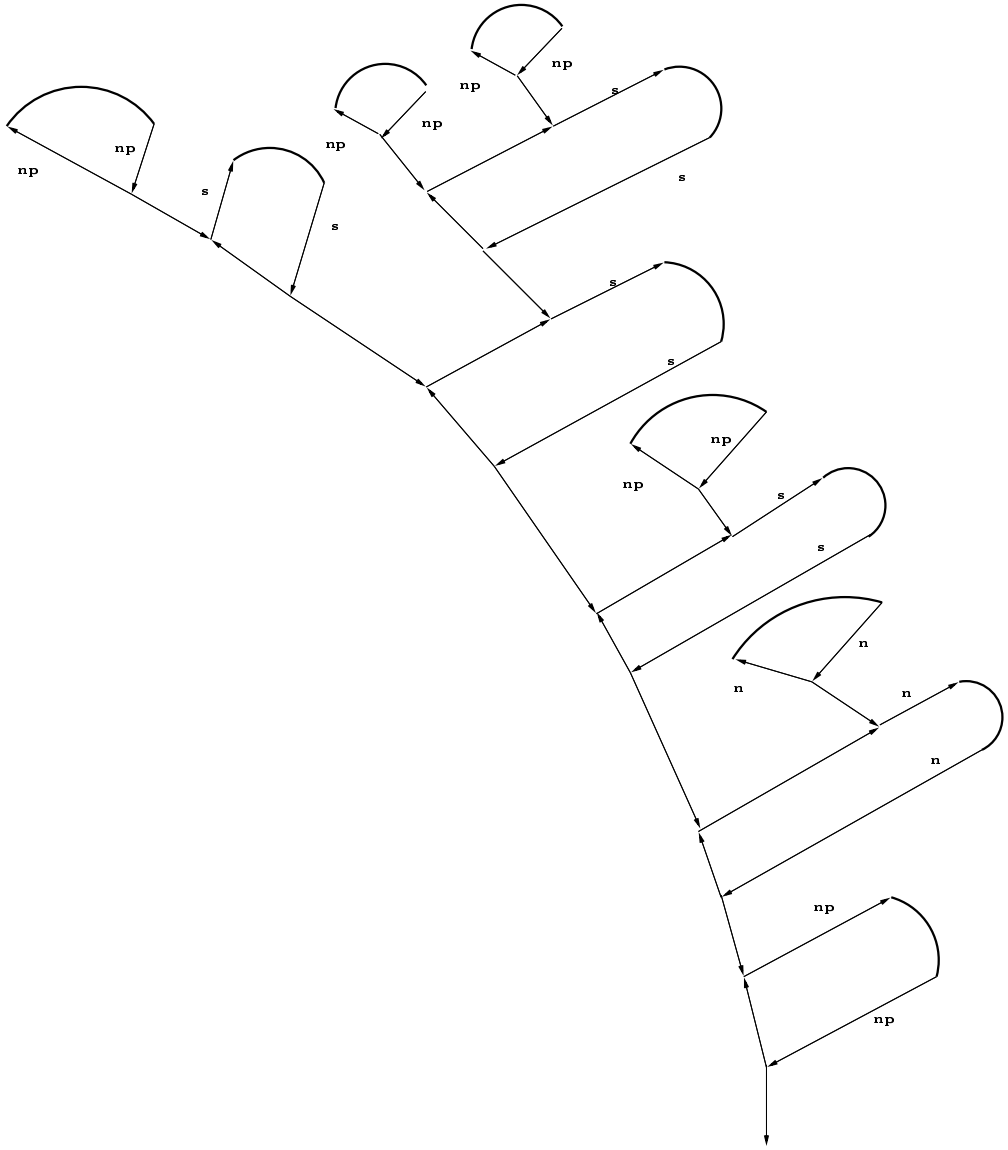
Since this graph is a terminal precedence tree, the corresponding proof structure is a proof net.

As we have already mentioned it, the proof of Proposition 1.2 provides us with an algorithm that generates a proof frame from a precedence graph with a minimal number of negative links. If the precedence graph is a terminal precedence tree , the resulting proof frame is a proof net without negative links.
There is also an algorithm that generates a proof frame from a precedence graph with a maximal number of negative links. Its principle is to break up every neutral vertex into two dual vertices and to link them together with a virtual edge. In this way, we obtain a proof net constituted of exactly one negative formula.
The first algorithm produces an initial proof frame which includes a minimum of order between atomic formulas. In this way, the constraints on the setting of axiom links are also minimum. Then the progressive addition of axiom links contributes to ordering all atomic formulas. The first algorithm produces an initial proof frame that includes a maximum of order between atomic formulas so that there is only one possibility of adding axiom links to build a proof net.

Example 1.7 *Let us consider the precedence tree of Example 1.6. With the algorithm for producing a minimal number of negative links, we obtain the corresponding proof net below.*



We do not recover the proof net from which we have started in Example 1.2 but we obtain a proof net without negative links which is a proof of the sequent:
$n \multimap np,\ n,\ np,\ s \multimap n \multimap n,\ np,\ np,\ s \multimap np \multimap s,\ np \multimap np \multimap s,\ np,\ s \multimap s \multimap s,\ np \multimap s \vdash np$
With the algorithm producing a maximal number of negative links, we obtain the corresponding proof net.



This is the unique proof of the sequent
$\vdash ((((((((((np \multimap np) \multimap s) \multimap s) \multimap ((np \multimap np) \multimap (np \multimap np) \multimap s) \multimap s) \multimap s) \multimap s) \multimap (np \multimap np) \multimap s) \multimap s) \multimap (n \multimap n) \multimap n) \multimap n) \multimap np) \multimap np$

1.3 Proving theorems viewed as building terminal precedence trees

After the static translation of proof nets into precedence trees, it remains to transpose the progressive construction of a proof net in the framework of precedence graphs. The elementary operation of this process is the addition of an axiom link to a proof frame. In the framework of precedence graphs, this amounts to merging a negative formula with a positive formula of the same type. In this way, the construction of a proof net is transformed into the derivation of a terminal precedence tree from a precedence graph. Let us define the notion of derivation for precedence graphs intrinsically.

Definition 1.6 *A precedence graph Θ_1 is derived from another precedence graph Θ_2 if Θ_1 is obtained from Θ_2 by iteration of the following operation: merging a negative formula with a positive formula of the same type and deleting the virtual links that become doubling paths after the merge.*

From this definition, the following proposition holds immediately.

Proposition 1.4 *A sequent $F_1, \cdots, F_n \vdash G$ is provable in IILL iff a terminal precedence tree can be derived from the precedence graph that is constituted of the syntactic trees of $F_1^+, \cdots, F_n^+, G^-$.*

In order to guarantee the correctness of the proof net in construction, we have to verify that no cycle is generated during a derivation. In this way, we get a partial correctness criterion : at any moment, the current proof frame is correct in the sense that it remains a potential proof net. This is essential for the efficiency of the construction. For example, all proof frames of Example 1.5 cannot become proof nets because their precedence graphs include a cycle.
In the course of a derivation we can even forecast the future states of the current precedence graph in order to avoid unsuccessful merges. In particular, if we try to merge the root A^- of the graph with a positive formula A^+, there cannot be a virtual link starting from A^+ to a formula other than A^-. Otherwise, this link will not be deleted without creating cycle.
Even with such refinements, the derivation of precedence trees leaves space for non-determinism. To reduce it, we can order the merges of dual formulas in a clever manner. In particular, we can use two strategies which are based on the knowledge of the final goal: we aim to build a precedence tree. Thus, we can build it from the root to the leaves by using an *top-down strategy*, by analogy with parsing strategies in language theory, or we can build it from the leaves to the root by using a *bottom-up strategy*. These strategies are defined precisely as follows:

- In the bottom-up strategy, we merge a negative formula A^- which has only neutral formulas as successors with a positive formula A^+ the only possible successors of which are A^- or some of its own successors.
- In the top-down strategy, we merge a positive formula A^+ which has only neutral formulas as predecessors with a negative formula A^- the only possible predecessors of which are A^+ or some of its own predecessors.

The top-down strategy is used by Morrill and Merenciano [MM97] in a non commutative framework, which is a non-essential difference here.
Both previous strategies repeat the same basic operation, merging dual formulas, but in a different order. At each step, when we perform this operation, we make a choice which may turn out to be wrong and lead to backtracking. Now, we can relax this choice: instead of merging a positive formula and a dual one, we can add a virtual link from the first to the second. Then, when we get more information, we can either merge the formulas or link them through a module which realizes the virtual link. In performing this operation, we must take care that the precedence graph still remains the image of an IILL proof frame. For this, we must verify that the conditions of Proposition 1.2 still hold. There is no problem for the operation of merging dual formulas but adding a virtual link can create alternating cycles, from which the following definition.

Definition 1.7 *A precedence graph Θ_1 is* derived in a broad sense *from another precedence graph Θ_2 if Θ_1 is obtained from Θ_2 by a sequence of operations that have one of the following form:*

- *merging a positive formula with a negative formula of the same type,*
- *connecting a positive formula with a negative formula of the same type with a virtual link provided this does not generate alternating cycle.*

In both cases, the virtual links that become redundant after the operation are deleted.

In order to distinguish the notion of derivation given in Definition 1.6 from this one, we call the first *strict derivation* as opposed to *derivation in a broad sense*. As in the case of strict derivation, the following proposition guarantees the equivalence between provability in IILL and derivability of precedence trees.

Proposition 1.5 *A sequent $F_1, \cdots, F_n \vdash G$ is provable in IILL iff a terminal precedence tree can be derived in a broad sense from the precedence graph which is constituted from the syntactic trees of $F_1^+, \cdots, F_n^+, G^-$.*

In a pure proof-theoretic framework, the interest of derivability in a broad sense is not obvious but it becomes clear in linguistic applications where it is closely akin to the adjoining operation of TAG.

As a conclusion of this section, let us come back to the original proposal of Lamarche: his idea was to represent the order between the atomic formulas of the initial proof frame in the form of a justification tree (precedence tree according to our vocabulary), following the idea of [HO93]. Then, he added links representing axiom links between nodes of this tree to transform the proof frame into a proof structure and he transposed its criterion to verify that the structure is a proof net.
We have perfected this model by identifying the nodes that are the extremities of an axiom link and by exhibiting the specific roles of the two kinds of edges, actual and virtual edges.
The results can be easily extended to all of the multiplicative framework (IMLL). The function that maps every proof frame into a precedence graph and the correctness criterion are left unchanged but there is a difference in the final result: the precedence graph of a proof net is not necessary a tree but a directed acyclic graph. Finally, Conditions 3 and 4 are not necessarily in Proposition 1.2 to prove the existence a proof frame that has a given precedence graph as its image.

2 Interaction Grammars (IG)

The presentation of proof nets under the form of terminal precedence trees finds a natural application in the representation of natural language syntax. According to the current view of Categorial Grammar [Moo96], resource sensitive logics are used to represent the syntax of natural languages in the form of deductive systems. Within this framework, formalisms differ in their logical systems but, usually, all are built around the Lambek Calculus because they are based on the principle that grammatical constituents are essentially characterised by their ability to be concatened with other constituents in order to build bigger structures. The Lambek Calculus which is both resource and order sensitive is well suited to expressing this principle. For instance, in French, in the canonical setting, a transitive verb expects a noun phrase on its left (its subject) and a noun phrase on its right (its object) in order to build a sentence. In the Lambek Calculus, its grammatical category is represented by the formula $np\backslash s/np$ ($\backslash$ and / respectively represent the left and the right implications of the Lambek Calculus).
In the design of IG, we start from another principle: what is fundamental in natural language syntax is the linear nature of dependencies between grammatical constituents: in most cases, they have valences which must be filled and which must be filled one time precisely. In this view, the difference between local dependencies and long distance dependencies, the difference between left dependencies and right dependencies are secondary. For instance, a transitive verb is characterised by the fact that it expects a subject and an object. These two valences must be filled one time each one, whatever the way of performing it is: in a canonical sentence, they are filled by the noun phrases that are immediately on the left and on the right of the verb but in a relative clause they can be filled by a long distance dependency. IILL is well suited for expressing all syntactic dependencies in a uniform way. Within this logic the category "transitive verb" is represented by the formula $np \multimap np \multimap s$. [Oeh94] and [Mor95] already chose this logical system to represent natural language syntax and we continue in the same way. We will study the differences between IG and the systems of Oehrle and Morrill. For the moment, what we have to note, is that Oehrle and Morrill give preference to the sequent calculus as proof framework, while we choose that of proof nets in order to exploit the results presented in the previous section. More precisely, if we model the structure of a grammatical constituent with a partial proof net under the form of its precedence tree, the final syntactic structure of the parsed phrase results from interaction between precedence trees for constituting a unique terminal tree which is the image of a complete proof net. The interaction capacity of each constituent is expressed by the polarised nodes of the associated precedence tree and the basic operation in the process of interaction consists in merging two dual nodes. Such is the basic mechanism of IG which we present in Subsection 2.1 .
Of course, this mechanism must be controlled and enriched by linguistic information to produce grammatical and meaningful sentences. In IG, contrary to the current view of Categorial Grammars, the

role of logic is minimal: logic is only used to manage the dependency structure between grammatical constituents. Word order is dealt with in the same manner as morphological and semantic features by extra-logical devices. Such a view fits in with the observation that the respective roles of word order and morphology are variable from a language to another. For instance, in French, the subject of a verb is usually determined by word order while, in German, it is determined by the morphological form of the nominative case. This leads us to deal with word order by means of the same extra-logical machinery as for morphological and semantic information. We present these machinery which is based on *feature matrices* and *unification* in Subsections 2.2 and 2.3.
Then, in Subsections 2.4 and 2.5, we define IG formally and we study its generative power in comparison with other formalisms.

2.1 Proof nets as syntactic structures

As any categorial grammar, an interaction grammar is defined from a set Cat_0 of atomic categories. An atomic category is an IILL proposition that represents a complete grammatical category. An incomplete category has valences which have to be filled by other categories while a complete category expects nothing: it can be only used to fill a valence of an incomplete category. For instance, an interaction grammar can be defined from the set Cat_0 composed of the three atomic categories: s (sentence), np (noun phrase) and n (common noun).
Then, incomplete categories are IILL formulas constructed from Cat_0 with linear implication.
A lexicon gives all the possible grammatical categories of each word of a given language in the form of a set of IILL formulas. Then, parsing a phrase consists in selecting IILL formulas from the lexicon which represent the grammatical functions that the words fill in the sentence. These formulas constitute a multi-set of assumptions from which we have to deduce the category representing the whole phrase. In this way, parsing a sentence amounts to proving a theorem.

Example 2.1 *Let us consider the French phrase* Les élèves que Paul pense voir en entrant *(the students whom Paul believes he sees entering). A lexicon provides a grammatical category for each word.*

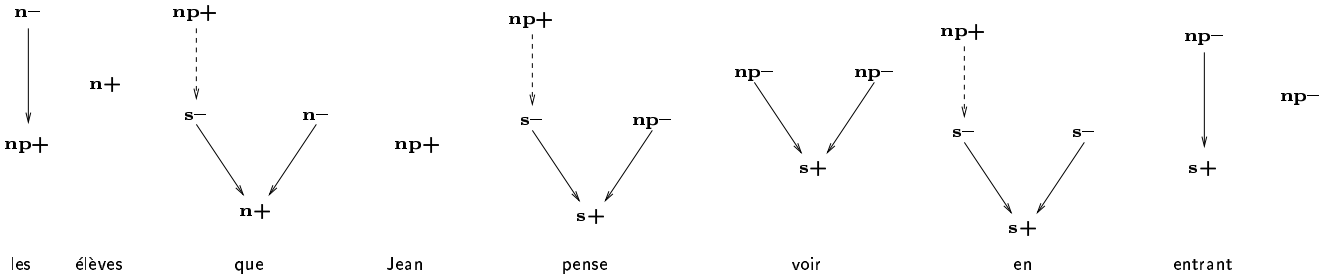
- *The determiner* les *takes a common noun as an argument to return a noun phrase, from this its category* $n \multimap np$.
- students *is a common noun (category* n*).*
- *One possible category for the relative pronoun* que *is* $(np \multimap s) \multimap n \multimap n$*: it takes a sentence where the object of a transitive verb is missing, that is, a phrase of the category* $np \multimap s$ *to give a relative clause which acts here as a modifier of a common noun, that is with type* $n \multimap n$.
- Paul *is a noun phrase (category* np*).*
- *The verb* pense *expects a verb phrase in the infinitive form as a complement and a noun phrase as a subject to return a sentence, from this its category:* $(np \multimap s) \multimap np \multimap s$.
- *The transitive verb* voir *has the category* $np \multimap np \multimap s$*, which expresses its double valence: it expects a noun phrase as a subject and a noun phrase as an object.*
- *The preposition* en *takes a verb phrase as an argument to return a modifier of sentence; thus its category is* $(np \multimap s) \multimap s \multimap s$.
- *Finally, the verb phrase* entrant *has the category* $np \multimap s$.

Parsing the noun phrase Les élèves que Paul pense voir en entrant *amounts to proving the following sequent which has been already studied in Example 1.2:*

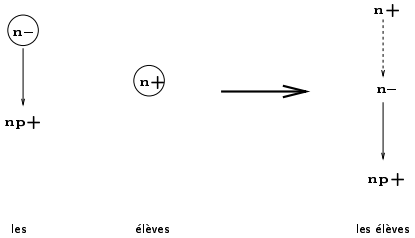
$$n \multimap np,\ n,\ (np \multimap s) \multimap n \multimap n,\ np,\ (np \multimap s) \multimap np \multimap s,\ np \multimap np \multimap s,\ (np \multimap s) \multimap s \multimap s,\ np \multimap s \vdash np$$

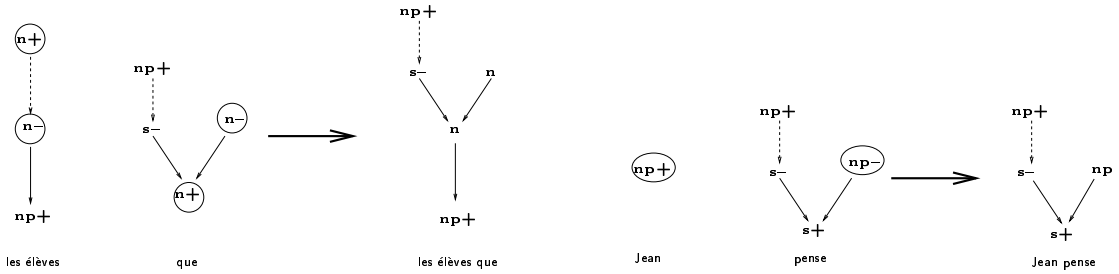
If parsing natural languages amounts to proving IILL sequents, Section 1 shows that it is preferable to do this in the framework of precedence graphs. We merely have to add a preliminary step: from the phrase $\omega_1.\cdots.\omega_n$ to be parsed, for each word ω_i, we must first select a corresponding grammatical category F_i from a lexicon and, if the phrase $\omega_1.\ldots.\omega_n$ is not a sentence, we must also foresee its grammatical category G. Then, we have to prove the sequent $F_1, \ldots, F_n \vdash G$, that is, derive a precedence tree from the precedence graph constituted of the syntactic trees of the formulas $F_1^+, \ldots, F_n^+, G^-$.
Of course, parsing strategies must take linguistic factors into account. Even if the bottom-up and top-down strategies which are described in Subsection 1.3 are more efficient in a purely proof-theoretic framework, this is not the case when applied to parsing natural languages. The goal of this paper is not to study the most efficient parsing strategies. However, it is interesting to see to what extent the formalism lends itself to implementing such strategies.
In this spirit, we present the *incremental strategy* which comes close to the mental process of parsing utterances. In a left-to-right parsing process, each constituent which is already formed, attempts to interact with the word that is immediately on its right to build a bigger constituent. If this is not possible, it is stored until a suitable constituent appears on its right. Such a strategy is not possible by using strict derivation of precedence graphs and requires derivation in a broad sense: instead of merging two dual vertices, we need sometimes to relax this operation by connecting the vertices together with a virtual link. Such an operation then allows this link either to be replaced by a whole module or to be reduced by identification of its extremities, which takes us back to the first elementary operation of merger. In the first case, the linguistic interpretation of the phenomenon is the modification of a constituent by adjoining another in an analogous manner to the TAG adjoining operation. Relaxing the elementary operation offers the advantage that it reconciles incrementality with monotonicity in the parsing process: starting from an under-specified precedence graph, we make it more specific in each left-to-right step. The following example illustrates to what extent this relaxation is necessary in the incremental strategy.

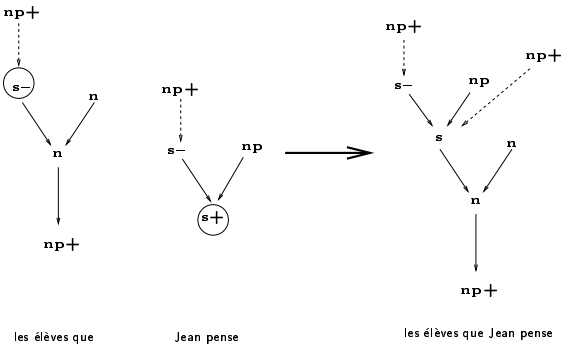
Example 2.2 *We propose to parse the phrase* Les élèves que Paul pense voir en entrant. *The initial precedence graph, which follows from the selection of the grammatical categories given previously for the words of the phrase, is that of Example 1.3.*

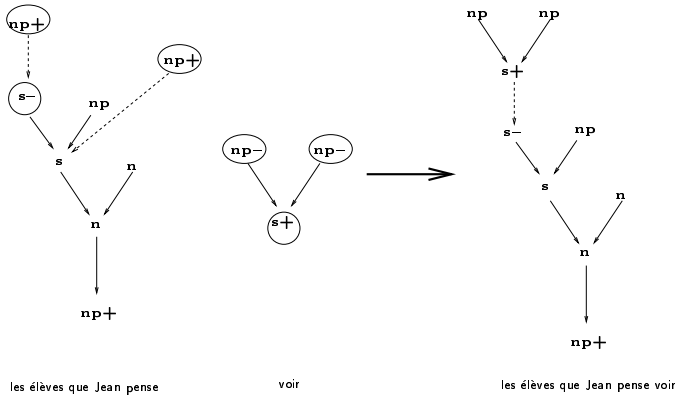


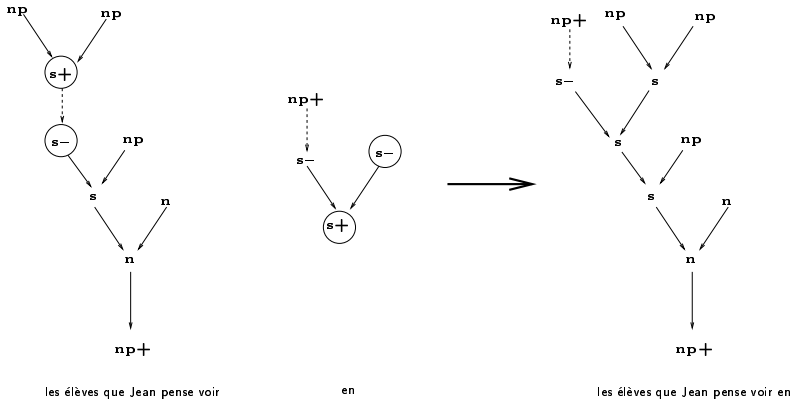
Here is a sequence of parsing steps within the incremental strategy. On the left of each arrow, there are the two constituents that interact in the parsing step and on the right there is the outcome of the interaction. The nodes which are active at each step are surrounded.

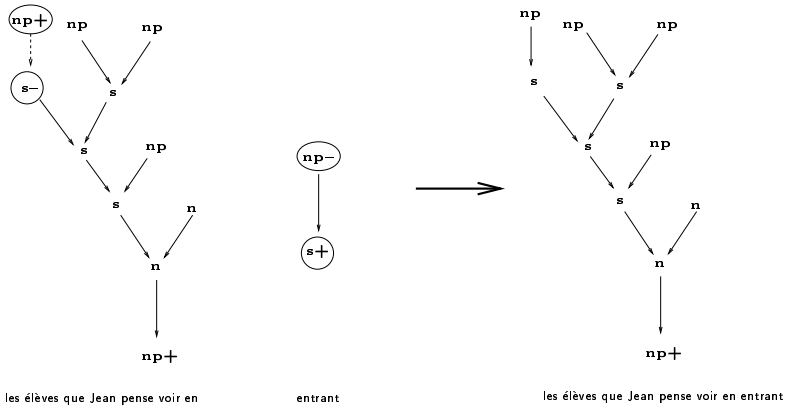



les élèves
que
les élèves que
Jean
pense
Jean pense


les élèves que
Jean pense
les élèves que Jean pense


les élèves que Jean pense
voir
les élèves que Jean pense voir


les élèves que Jean pense voir
en
les élèves que Jean pense voir en



By merging the root of this tree with the isolated node np^-, we find again the precedence tree given by Example 1.3 as the parsing outcome.

From this example, we can induce some general remarks on the incremental strategy. Each step consists in an interaction between two constituents which takes only one of the following forms.

Simple interaction One constituent tries to fill a valence of another constituent. If this is possible, it can perform this immediately by merging its root with a dual node of the second tree. This is the case, for example, when *Jean* and *pense* are merged to build *Jean pense*: the verb *pense* expects a noun phrase which is expressed by the negative node $np-$ in its syntactic tree and the noun phrase *Jean* comes up to this expectation because its syntactic tree contains a positive node $np+$. The two constituents are merged by identification of the nodes $np-$ and $np+$. Simple interaction corresponds to substitution in TAG and to application in Categorial Grammar.
In a simple interaction, instead of merging two dual nodes, we can introduce a virtual link between them. The interest of such an operation is that it preserves the possibility for a constituent to be modified later. In this way, as we will see later, we integrate the adjoining mechanism in our system. For instance, in the example above, the first parsing step consists in building the syntactic structure of *les élèves* from those of *les* and *élèves*. Rather than merge the node $n-$ of the first syntactic tree and the node $n+$ of the second syntactic tree, we add a virtual link between the two nodes. In this way, we preserve the possibility of modifying the common noun *élèves* by replacing this virtual link with an actual module. Here, a module will really take the place of the virtual link: it corresponds to the relative clause *que Paul pense voir en entrant*. Adjoining is already achieved in the second parsing step. The phrase *les élèves* is merged with the relative pronoun *que* which is viewed as the beginning of a relative clause. This is performed by merging two pairs of dual nodes simultaneously. In this way, we forbid any new adjoining, which is sensible linguistically. In a general way, the choice between merging two dual nodes and connecting them by a virtual link depends on the probability of having adjoining to do later in the parsing process. In any case, if useless virtual links are added in the parsing process, they can be deleted at the end as the example above shows.

Complex interaction One constituent try to fill a valence of another constituent whereas some of their sub-constituents try to do the same. In other words, a simple interaction includes only one elementary operation while a complex operation includes two or more elementary operations. Each of them consists either in merging two dual nodes or connecting them with a virtual link.
For example, the incomplete constituent *les élèves que Jean pense* interact with the verb *voir* to build the noun phrase *les élèves que Jean pense voir*. The interaction is composed of three elementary operations:

- in two of them, which consist in merging dual nodes *np*, phrase *les élèves que Jean pense* provides the verb *voir* with its subject and its object;
- in the last operation, the verb *voir* provides the verb *pense* with its complement. Here, the dual nodes *s+* and *s-* are only connected by a virtual link in order to preserve the possibility of adjoining a future complement to *voir*. This complement will be the gerundive *en entrant*.

Hypothetical reasoning in Lambek Grammar and the adjoining operation in TAG correspond to two specific and different forms of complex interaction in IG but this mechanism is more general than the first ones.

Some interactions appear as complex only because of some degree of arbitrariness in the definition of the basic grammatical categories. For example, the last step where *les élèves que Jean pense voir en* interact with *entrant* to build *les élèves que Jean pense voir en entrant* belongs strictly to the class of complex interactions, but, if we take *verb phrase* as a basic category, the interaction becomes simple.
Now, the model that we have outlined is not sufficient to capture the grammaticality of sentences by means of only provability in linear logic. Provability in linear logic only expresses that all words of a phrase fill their valence in terms of categories. Other syntactic aspects such as word order, case and agreement are completely disregarded in this representation. If we refer to our last example, we are unable to detect the ungrammaticality of the phrases

$$* \textit{ les que élèves Paul pense voir en entrant}$$
$$* \textit{ les élèves que Paul pensent voir en entrant} \quad .$$

As a conclusion, it is necessary to enrich our model to capture syntactic properties of natural languages that go beyond simple grammatical valences.

2.2 From atomic categories to feature matrices

As the last examples show it, propositional linear logic is not sufficient for taking into account all aspects of natural language syntax. We must enrich this basic framework. Until now, we express the structure of each grammatical constituent with a partial proof net represented under the form of a precedence tree. This tree is labelled with polarised atomic categories and we propose to replace these atomic labels with *feature matrices.*
To describe the syntactic and semantic properties of a language, we assume a set of attributes and for each attribute A a domain D_A of values. In most cases, D_A is a finite set of atoms without any structure but it can also be a more or less complex algebra.
Two attributes have such a structured domain and they play a particular role among all attributes. The first, *phon*, gives the phonological form of the associated constituent. Its domain is a non-commutative free monoid the generators of which are the words of the language. Information on word order is given by this feature. The second attribute, *sem*, gives a semantic representation of the constituent under the form of a higher order typed lambda-term *à la* Montague. The basic semantic types represent boolean values and individuals. Compound types are built from the basic types with the operator $\rightarrow$.
A feature pairs an attribute A and a value $v \in D_A$ to express a syntactic or semantic piece of information.

Example 2.3 *The value of the feature* sem *for the noun phrase* les élèves que Jean pense voir en entrant *is* plu(λx.(élève x) $\wedge$ (penser(simult (voir x j) (entrer j)) j)). *This is the set of individuals* x *that verify the property* λx.(élève x) $\wedge$ (penser(simult (voir x j) (entrer j)) j). *This property is satisfied when both propositions* (élève x) *and* penser(simult (voir x j) entrer j)) j *are satisfied at the same time. The constant* j *represents the individual* Jean. *The predicate* simult *takes two propositions* (voir x j) *and* (entrer j) *as arguments and it is true when both propositions are true simultaneously in the course of time. We could imagine a more sophisticated representation of time but our aim here is not to build a complete semantic model but rather to give a flavour of what it is possible to do within our formalism.*

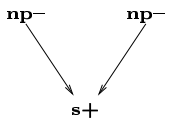
In order to express under-specified features, the values of features can contain variables, denoted by capital letters (B, P, T, X...). For this, we associate a set X_A of variables to each attribute A. From D_A and X_A, we build a set of algebraic terms $D_A[X_A]$. Then, a feature is built by pairing an attribute A and an element of $D_A[X_A]$.

Example 2.4 *If a relative clause governed by the pronoun* que *plays the role of a noun modifier, a possible value of the feature* sem *for the modified common noun is* λx.(Px) $\wedge$ B. B *is a boolean variable that represents the semantics of the relative clause whereas* P *is a property that represents the semantics of the common noun which precedes the relative clause and is modified by this. The semantics of the*

modified common noun is a property characterised for any generic individual x^3 *by the conjunction of* Px *and* B.

We enrich the precedence trees that express the structure of syntactic constituents by replacing the atomic categories as labels with feature matrices. The variables that are used to under-specify features in these enriched syntactic trees behave as universally quantified first order variables so that the syntactic tree can be viewed as representing partial proof nets in the logical fragment IILL extended to first order by universal quantification.
A syntactic tree can be presented in a simplified manner as a hierarchy of successive sub-matrices that are embedded one into another. All actual and virtual links are implied and can be recovered without problems by means of the rules articulating the nature of links and the polarities of their extremities for precedence graphs (see Definition 1.4). There is only one ambiguous case: when a neutral node constitutes an immediate constituent of another neutral node, the link between them may be virtual or actual. If the link is virtual, there is no way to delete it and obtain a correct proof net. Thus, we assume that, in this case, the link is actual.

Example 2.5 *We consider the syntactic tree which is associated to the transitive verb* voir *by a lexicon in Example 2.2.*



If we replace the atomic categories with feature matrices as the labels of the syntactic trees, we obtain the following syntactic tree:

$$
\textit{voir}\left[\begin{array}{l}
+\\
\textbf{cat} = s\\
\textbf{phon} = X_1.\textit{voir}.X_2\\
\textbf{sem} = \textit{voir}\ I_1\ I_2\\
\textbf{mod} = \textit{inf}\\
\left[\begin{array}{l} -\\ \textbf{cat} = \textit{np}\\ \textbf{phon} = X_1\\ \textbf{sem} = I_2\\ \textbf{funct} = \textit{subj}\end{array}\right.\\
\\
\left[\begin{array}{l} -\\ \textbf{cat} = \textit{np}\\ \textbf{phon} = X_2\\ \textbf{sem} = I_1\\ \textbf{funct} = \textit{obj}\end{array}\right.
\end{array}\right.
$$

In this tree, the variables X_1, X_2, I_1 *and* I_2 *are local and they can be viewed as universally quantified variables so that the syntactic tree is equivalent to the first order linear logic formula:*
$\forall\ X_1\ X_2\ I_1\ I_2(\textit{np}(\textbf{phon} = X_1,\ \textbf{sem} = I_2,\ \textbf{funct} = \textit{subj}) \multimap \textit{np}(\textbf{phon} = X_2,\ \textbf{sem} = I_1,\ \textbf{funct} = \textit{obj}) \multimap s(\textbf{phon} = X_1.\textit{voir}.X_2,\ \textbf{sem} = \textit{voir}\ I_1\ I_2,\ \textbf{mod} = \textit{inf}))$.
A syntactic tree cannot be translated into a first order linear logic formula in every case: if it includes neutral nodes, it corresponds to a partial proof net.

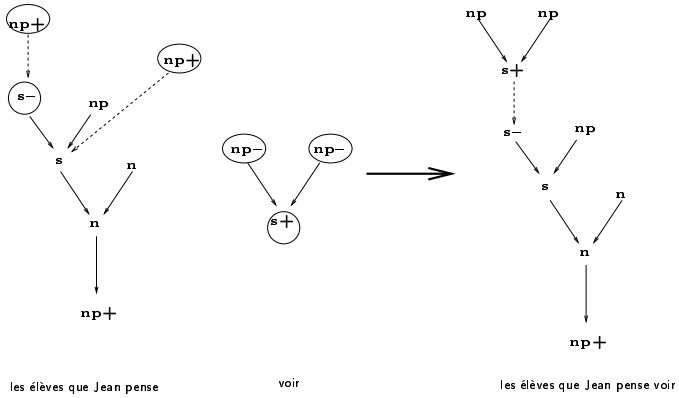
2.3 Syntactic composition as unification of feature matrices

The corollary of the enrichment of syntactic trees with feature matrices is an enrichment of the two elementary operations of syntactic composition with *unification*. These operations consist in either merging two dual nodes or connecting them through a virtual link. When we restricted linguistic information to grammatical categories, dual nodes were nodes with opposite polarities labelled with the same category.

[3]The variables that are associated to the λ-terms representing the values of the feature *sem* are denoted by lower case letters to be distinguished from the variables used to express under-specified values of features.

Now, dual nodes still have opposite polarities but they are labelled with unifiable feature matrices. The notion of unification that we use here is the same as that used in Unification Grammars [Abe93]. Unifying two matrices M_1 and M_2 consists in building their union with the following condition: when the matrices respectively include features $A = v_1$ and $A = v_2$ the equation $v_1 = v_2$ must have a solution in the domain D_A. In this case, the result of unifying M_1 and M_2 includes a feature $A = v$ where v proceeds from instantiating the variables of v_1 or v_2 with any of the most general solutions of the equation $v_1 = v_2$ [4]. Apart from particular features as phonological and semantic features, attribute domains are usually reduced to finite sets of atomic values without additional structure so that equations with the form $v_1 = v_2$ become trivial.

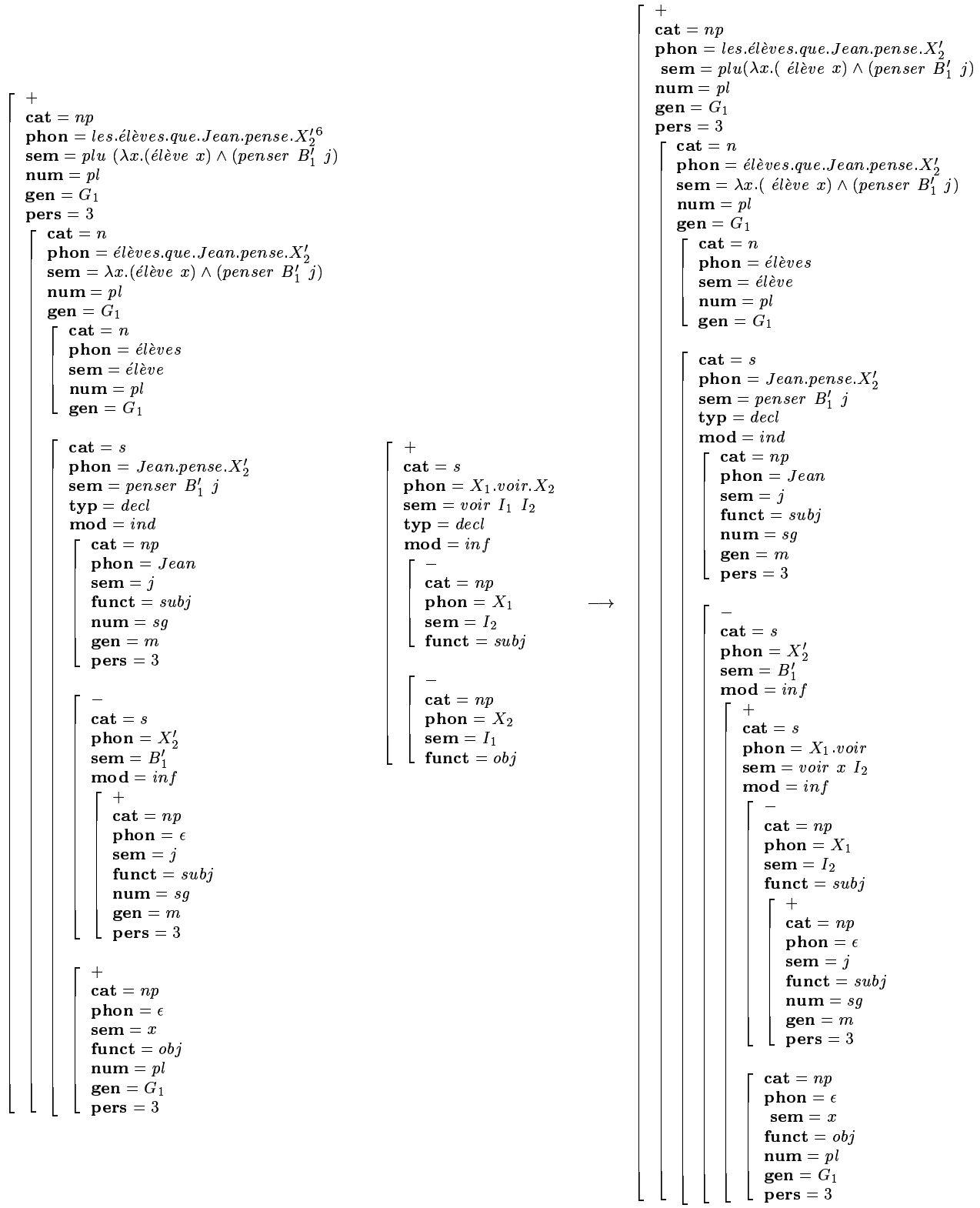
Example 2.6 *Consider the following step in the parsing process described in Example 2.2.*



By labelling the bare syntactic trees with feature matrices, we obtain the step below. It is constituted of three elementary operations on feature matrices which are richer than the corresponding operations on the bare trees:

- *the verb* voir *finds its subject by unification of two feature matrices and, as a consequence,* j, *which represents the individual* Jean, *becomes the semantic reference of this subject, whereas its phonological feature gets an empty form;*
- *the verb* voir *finds its object by unification of two other feature matrices. This unification entails the instantiation of both phonological and semantic features reference: the object of the verb* voir *takes an empty phonological form and an undetermined element* x *of the set* élève *as its semantic reference;*
- *a virtual link is put between the infinitive clause which is expected as a complement of the verb* pense *and the infinitive clause which is provided by the verb* voir, *the corresponding matrices are unifiable but they are not unified immediately to allow a future modification of the second infinitive clause.*

[4] In the most cases, the equation $v_1 = v_2$ has one solution at most but in some cases (higher order unification for the semantic feature for instance) it can have several solutions which are not equivalent.



[6] Variables which are common to both syntactic trees are renamed to avoid illicit capture.

2.4 Formal definition of Interaction Grammars

Now, we are in position to define *Interaction Grammars* formally. These grammars use the notions of *feature matrix* and *syntactic tree* which have to be defined first.

Definition 2.1 *A universe of attributes $\mathcal{A}$ is a set of attributes, each attribute A being a name associated with an algebra D_A, which is its domain, and with a set X_A of variables.*
A feature on $\mathcal{A}$ is a pair $A = v$ of an attribute A from $\mathcal{A}$ and a value v from $D_A[X_A]$.
A feature matrix on $\mathcal{A}$ is a set of features on $\mathcal{A}$ and a syntactic tree on $\mathcal{A}$ is a precedence tree labelled with feature matrices on $\mathcal{A}$.

Definition 2.2 *An Interaction Grammar* G *is given by:*

- *a finite vocabulary $\mathcal{V}_G$ composed of the words of the language;*
- *a universe of attributes $\mathcal{A}_G$; $\mathcal{A}_G$ contains a special attribute* phon *which has the free non-commutative monoid generated by $\mathcal{V}_G$ as its domain,*
- *a lexicon $\mathcal{L}ex_G$ which associates a finite set of syntactic trees on $\mathcal{A}_G$ to each word of $\mathcal{V}_G$ with the constraint that every syntactic tree associated to any word w by the lexicon contains w in its phonological feature* [5]*;*
- *a feature f_G on $\mathcal{A}_G$ which characterises the terminal phrases of the language* [6].

Let us give an idea of a lexicon which defines an Interaction Grammar for the French language.

Example 2.7 *Here are the possible entries of a lexicon that are used to parse the phrase* les élèves que Jean pense voir en entrant.

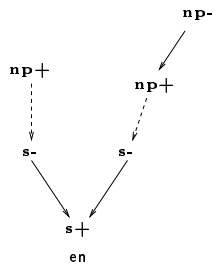
$$les \left[\begin{array}{l} + \\ \textbf{cat} = np \\ \textbf{phon} = les.X_1 \\ \textbf{sem} = plu\ P_1 \\ \textbf{num} = pl \\ \textbf{gen} = G_1 \\ \textbf{pers} = 3 \\ \left[\begin{array}{l} - \\ \textbf{cat} = n \\ \textbf{phon} = X_1 \\ \textbf{sem} = P_1 \\ \textbf{gen} = G_1 \\ \textbf{num} = pl \end{array}\right] \end{array}\right] \quad élèves \left[\begin{array}{l} + \\ \textbf{cat} = n \\ \textbf{phon} = élèves \\ \textbf{sem} = élève \\ \textbf{num} = pl \end{array}\right] \quad que \left[\begin{array}{l} + \\ \textbf{cat} = n \\ \textbf{phon} = X_1.que.X_2 \\ \textbf{sem} = \lambda x.(P_1\ x) \wedge B_1 \\ \textbf{num} = N_1 \\ \textbf{gen} = G_1 \\ \left[\begin{array}{l} - \\ \textbf{cat} = n \\ \textbf{phon} = X_1 \\ \textbf{sem} = P_1 \\ \textbf{num} = N_1 \\ \textbf{gen} = G_1 \end{array}\right] \\ \left[\begin{array}{l} - \\ \textbf{cat} = s \\ \textbf{phon} = X_2 \\ \textbf{sem} = B_1 \\ \textbf{typ} = decl \\ \left[\begin{array}{l} + \\ \textbf{cat} = np \\ \textbf{phon} = \epsilon \\ \textbf{sem} = x \\ \textbf{funct} = obj \\ \textbf{num} = N_1 \\ \textbf{gen} = G_1 \\ \textbf{pers} = 3 \end{array}\right] \end{array}\right] \end{array}\right] \quad Jean \left[\begin{array}{l} + \\ \textbf{cat} = np \\ \textbf{phon} = Jean \\ \textbf{sem} = j \\ \textbf{num} = sg \\ \textbf{gen} = m \\ \textbf{pers} = 3 \end{array}\right]$$

[5] We can extend lexical entries to sequences of words for taking into account discontinuous constituents or idioms.
[6] Usually, this feature is $cat = s$ and it is present in the label of the root of the syntactic tree.

$$
\textit{pense}\ \left[\begin{array}{l}
+\\
\textbf{cat} = s\\
\textbf{phon} = X_1.\textit{pense}.X_2\\
\textbf{sem} = \textit{penser}\ B_1\ I_1\\
\textbf{typ} = \textit{decl}\\
\textbf{mod} = \textit{ind}\\
\left[\begin{array}{l}
-\\
\textbf{cat} = np\\
\textbf{phon} = X_1\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}\\
\textbf{num} = \textit{sg}\\
\textbf{gen} = G_1\\
\textbf{pers} = 3
\end{array}\right.\\
\left[\begin{array}{l}
-\\
\textbf{cat} = s\\
\textbf{phon} = X_2\\
\textbf{sem} = B_1\\
\textbf{mod} = \textit{inf}\\
\left[\begin{array}{l}
+\\
\textbf{cat} = np\\
\textbf{phon} = \epsilon\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}\\
\textbf{num} = \textit{sg}\\
\textbf{gen} = G_1\\
\textbf{pers} = 3
\end{array}\right.
\end{array}\right.
\end{array}\right.
\qquad
\textit{voir}\ \left[\begin{array}{l}
+\\
\textbf{cat} = s\\
\textbf{phon} = X_1.\textit{voir}.X_2\\
\textbf{sem} = \textit{voir}\ I_1\ I_2\\
\textbf{mod} = \textit{inf}\\
\left[\begin{array}{l}
-\\
\textbf{cat} = np\\
\textbf{phon} = X_1\\
\textbf{sem} = I_2\\
\textbf{funct} = \textit{subj}
\end{array}\right.\\
\left[\begin{array}{l}
-\\
\textbf{cat} = np\\
\textbf{phon} = X_2\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{obj}
\end{array}\right.
\end{array}\right.
$$

$$
\textit{en}\ \left[\begin{array}{l}
+\\
\textbf{cat} = s\\
\textbf{phon} = X_1.\textit{en}.X_2\\
\textbf{sem} = \textit{simult}\ B_1\ B_2\\
\left[\begin{array}{l}
-\\
\textbf{cat} = s\\
\textbf{phon} = X_1\\
\textbf{sem} = B_1\\
\left[\begin{array}{l}
+\\
\textbf{cat} = np\\
\textbf{phon} = X_3\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}\\
\textbf{num} = N_1\\
\textbf{gen} = G_1\\
\textbf{pers} = P_1\\
\left[\begin{array}{l}
-\\
\textbf{cat} = np\\
\textbf{phon} = X_3\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}\\
\textbf{num} = N_1\\
\textbf{gen} = G_1\\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right.\\
\left[\begin{array}{l}
-\\
\textbf{cat} = s\\
\textbf{phon} = X_2\\
\textbf{sem} = B_2\\
\textbf{mod} = \textit{pres.part}\\
\left[\begin{array}{l}
+\\
\textbf{cat} = np\\
\textbf{phon} = \epsilon\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}\\
\textbf{num} = N_1\\
\textbf{gen} = G_1\\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right.
\qquad
\textit{entrant}\ \left[\begin{array}{l}
+\\
\textbf{cat} = s\\
\textbf{phon} = X_1.\textit{entrant}\\
\textbf{sem} = \textit{entrer}\ I_1\\
\textbf{mod} = \textit{pres.part}\\
\left[\begin{array}{l}
-\\
\textbf{cat} = np\\
\textbf{phon} = X_1\\
\textbf{sem} = I_1\\
\textbf{funct} = \textit{subj}
\end{array}\right.
\end{array}\right.
$$

To take into account the fact that the preposition en *identifies the subjects of two clauses, a definite and a gerundive clause, we have slightly modified the structure of the precedence tree that represents its grammatical function with respect to Example 2.2. Here is the precedence tree in the previous form which underlies the syntactic tree associated to the preposition* en.



The structure above says that the preposition en *expects two sentences: the first which is the main sentence, represented by the category* $s-$ *on the right, and the second in the gerundive form, represented by the category* $s-$ *on the left, which acts as a modifier of the main sentence. We have added two nodes of type* np *over the node* $s-$ *on the right in order to distinguish the subject of the main sentence. Then, some of its features will be transmitted to the subject of the gerundive sentence which is provided by the preposition* en. *The variables* I_1, N_1, G_1 *and* P_1 *which are present in the matrices attached to both subjects express that they refer to the same semantic entity.*

The language that is generated by an Interaction Grammar results from all terminal syntactic trees which are derived from the lexicon. The notion of derivation for syntactic trees comes directly from Definition 1.7 where merger of dual nodes is replaced with unification of dual nodes.

Definition 2.3 *A syntactic tree T_1 is derived from another syntactic tree T_2 if T_1 is obtained from T_2 by a sequence of operations that have one of the following forms:*

- *unifying a positive node with a negative node,*
- *connecting a positive node with a negative node with a virtual link provided the nodes are unifiable and the connection does not generate an alternating cycle.*

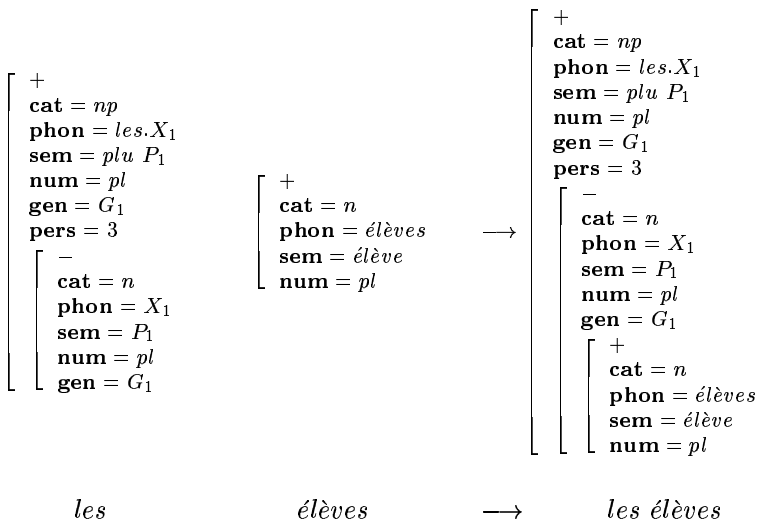
In both cases, the virtual links which become redundant after the operation are deleted.

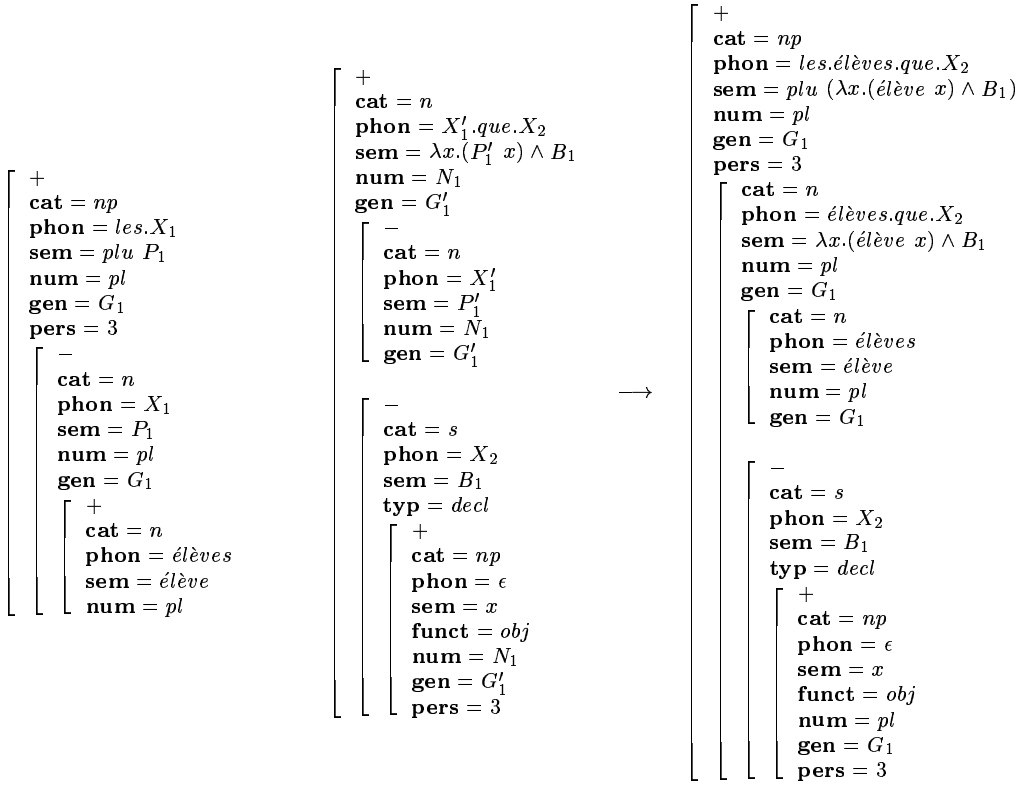
Definition 2.4 *The language $\mathcal{L}_G$ generated by the Interaction Grammar G is the set of strings $w_1 \ldots . w_n \in \mathcal{V}_G^*$ such that there exists a set $\{T_1, \ldots, T_n, T\}$ of syntactic trees with the following property:*

- *for every* k, *T_k is associated to w_k by the lexicon $\mathcal{L}ex_G$;*
- *T is derived from $T_1, \ldots, T_n$ and T is a terminal precedence tree in the sense given by Definition 1.5;*
- *the feature f_G is present in the matrix labelling the root of T.*

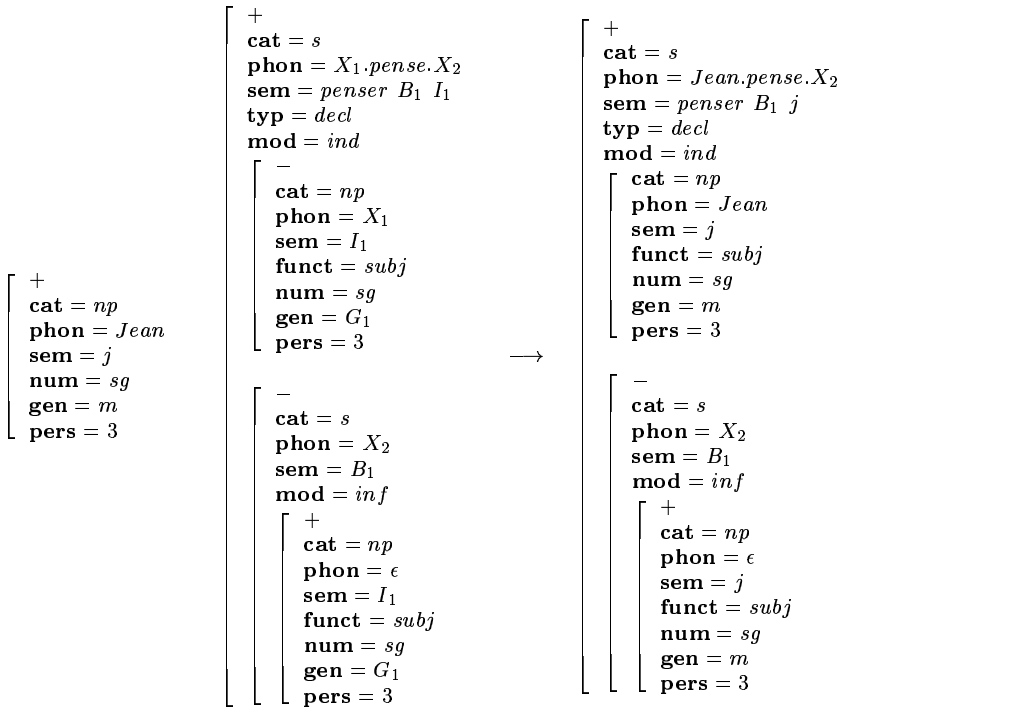
To parse phrases that are built from the vocabulary, we can use different strategies but it is not the purpose here to discuss which is the most efficient parsing strategy. By taking Example 2.2 again, we merely want to show how the parsing process works.

Example 2.8 *Here are the successive matrices that result from the step by step process of parsing the phrase* les élèves que Jean pense voir en entrant *by applying the incremental strategy.*

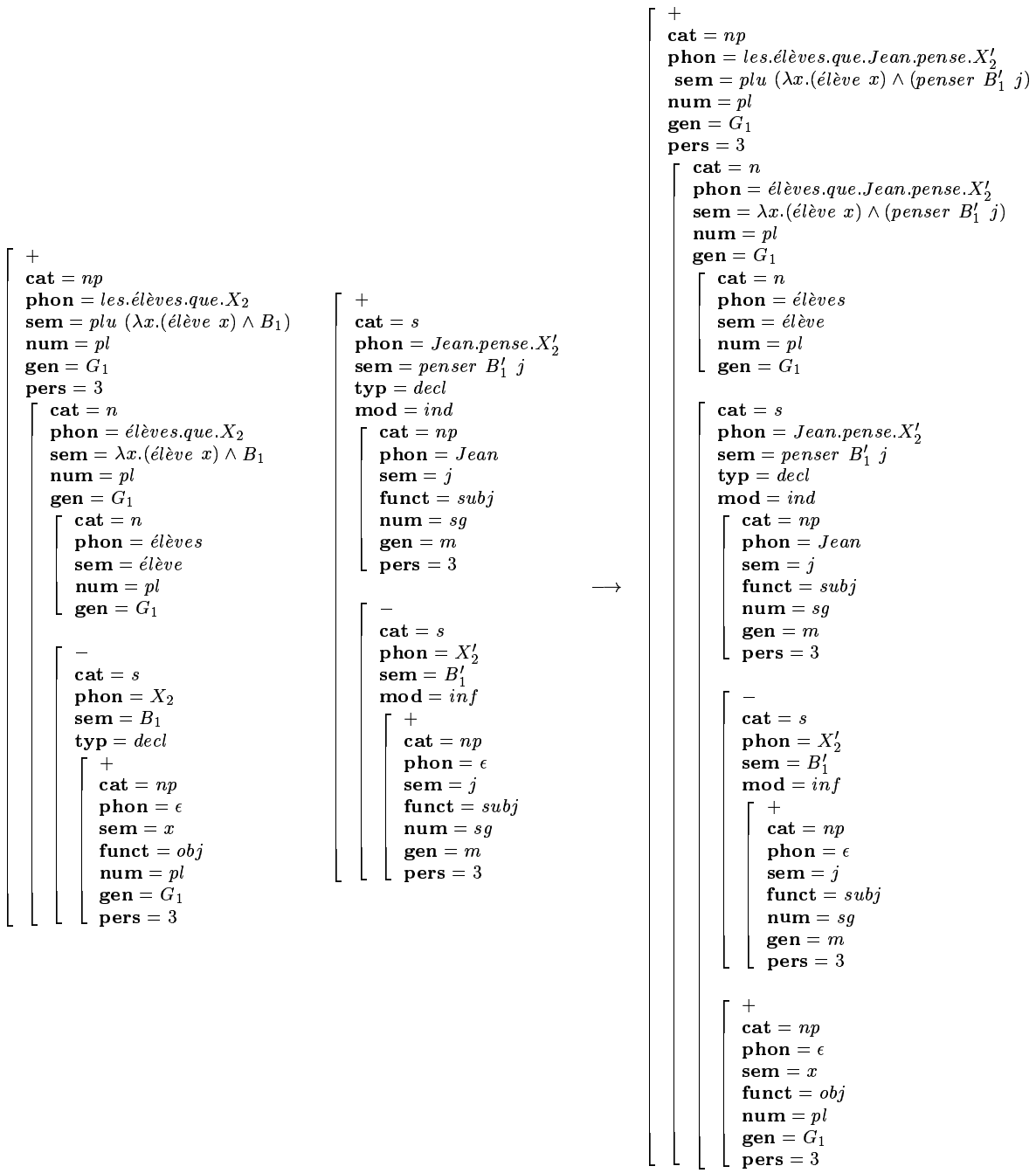




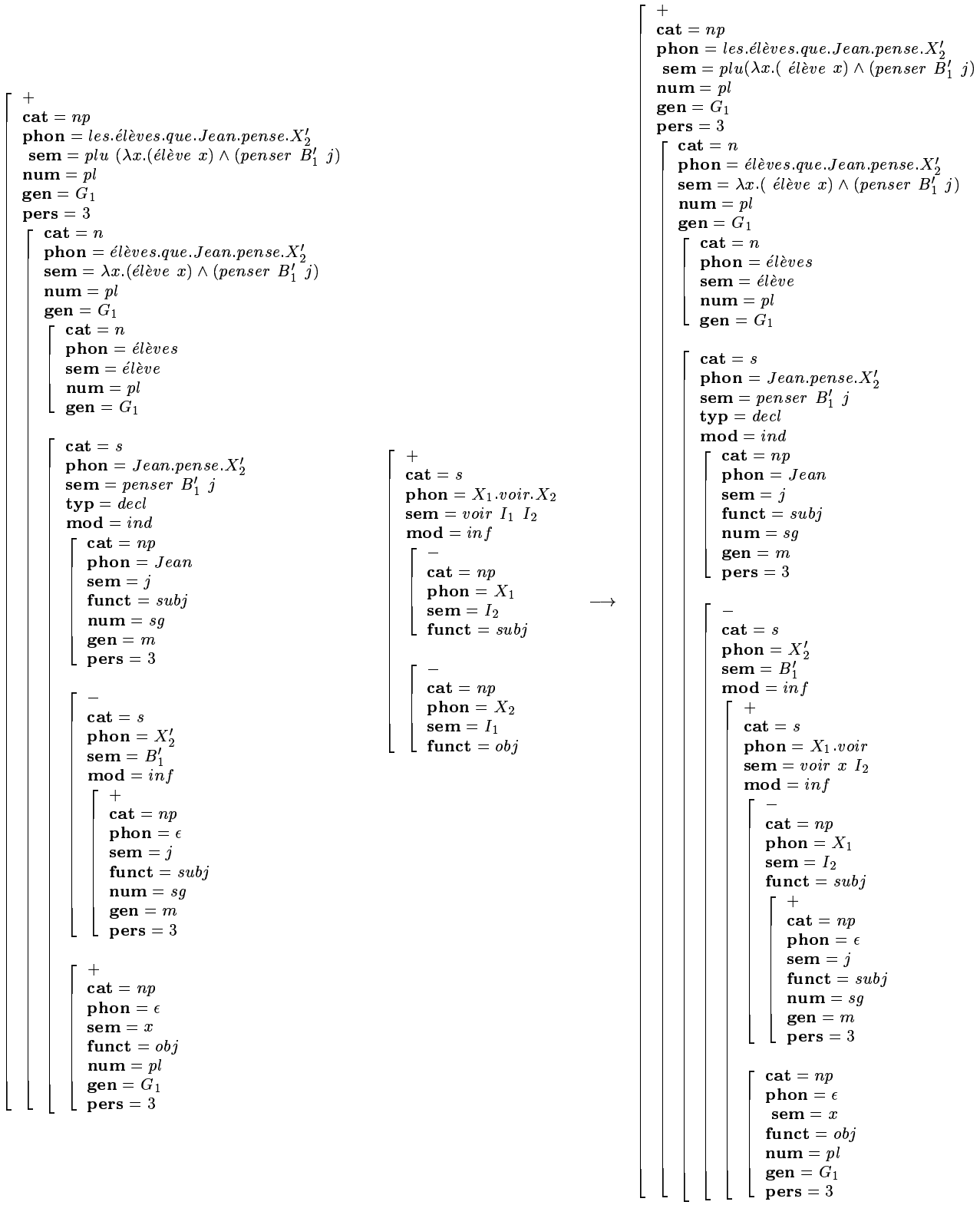
les élèves *que* ⟶ *les élèves que*



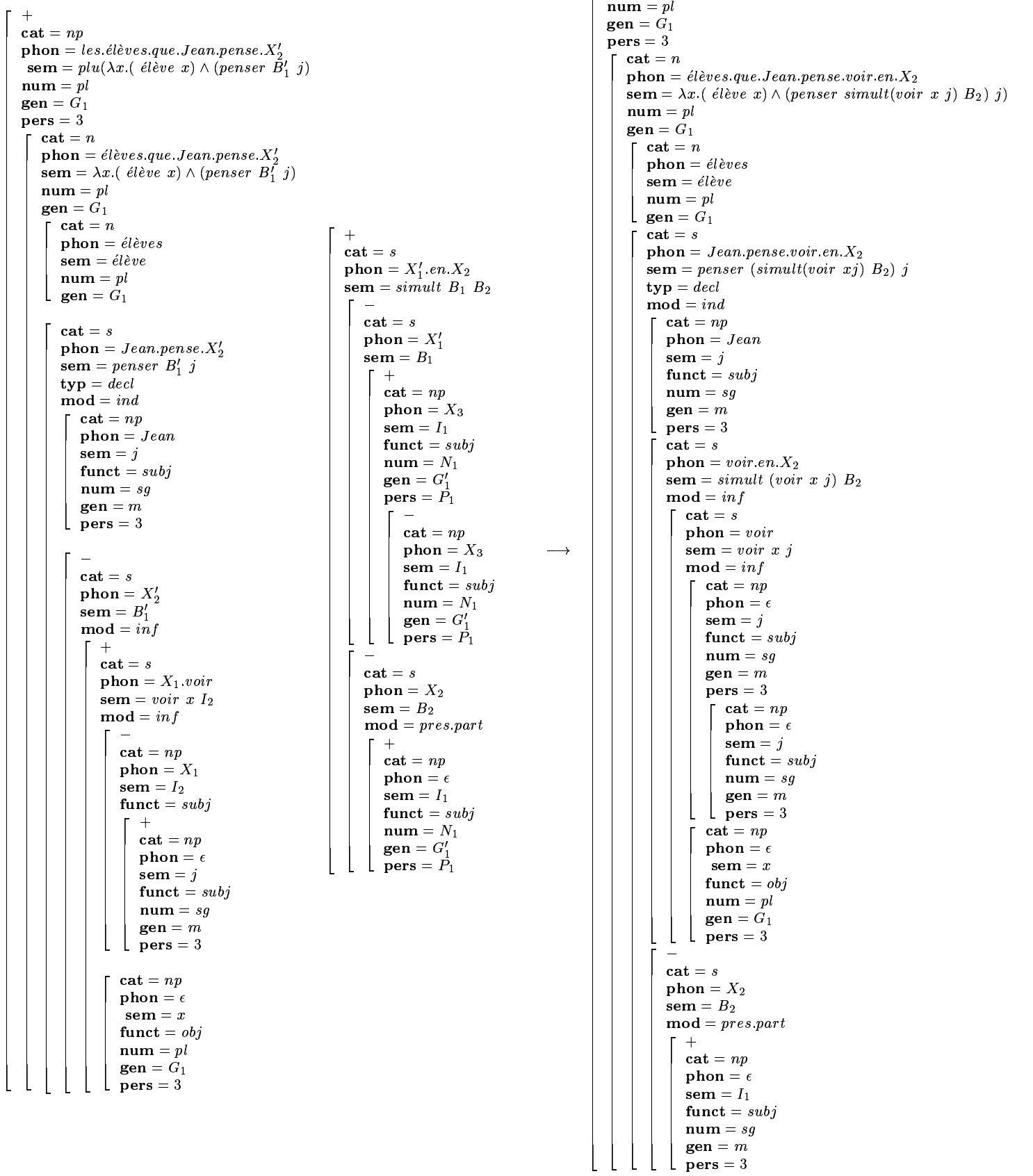
Jean *pense* ⟶ *Jean pense*



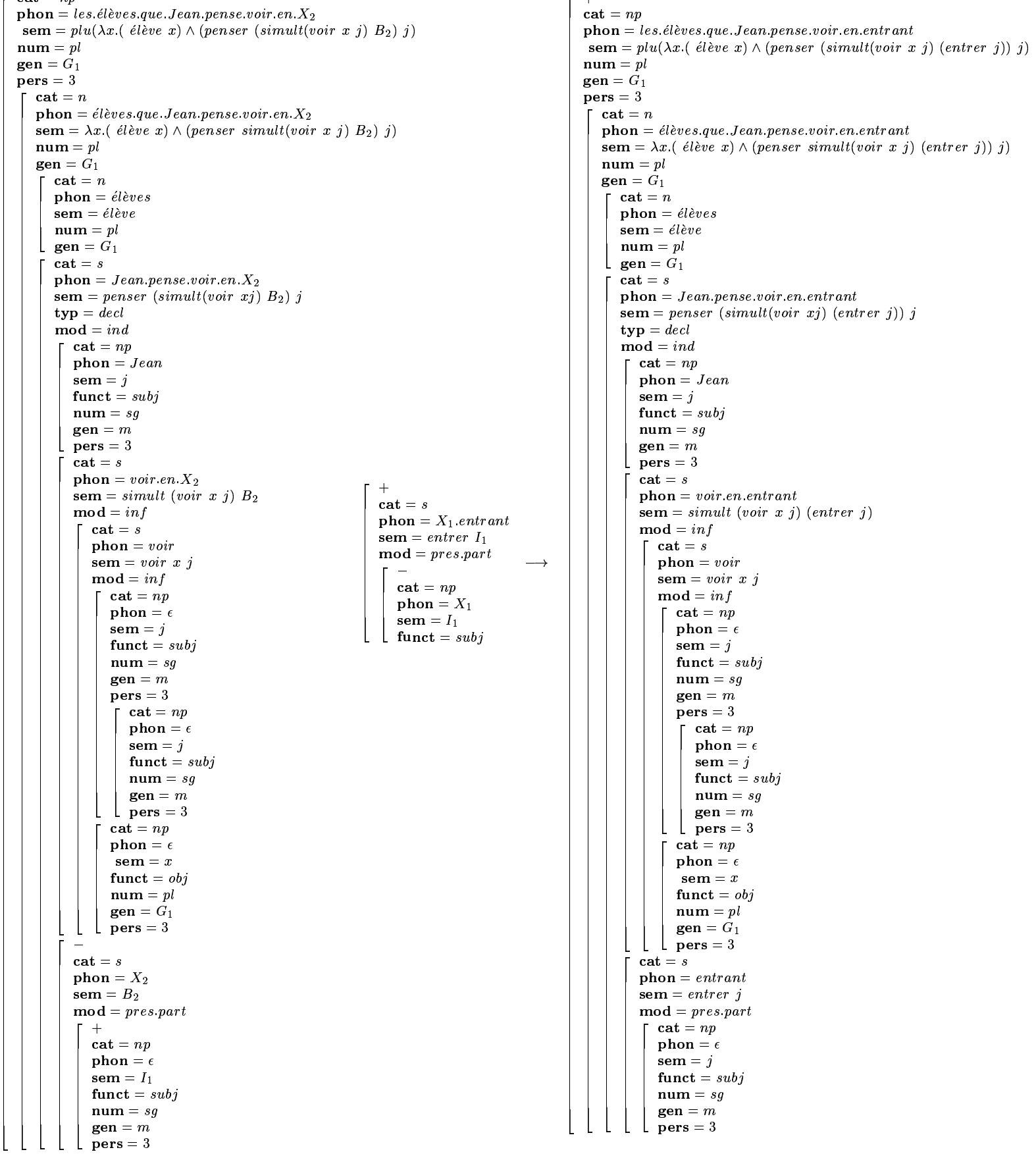
les élèves que *Jean pense* ⟶ *les élèves que Jean pense*



les élèves que Jean pense *voir* ⟶ *les élèves que Jean pense voir*



les élèves que Jean pense voir *en* ⟶ *les élèves que Jean pense voir en*



les élèves que Jean pense voir en *entrant* ⟶ *les élèves que Jean pense voir en entrant*

2.5 Generative power and limits of Interaction Grammars

2.5.1 Which function for logic in syntax ?

As we have mentioned in the introduction, a major divergence between Lambek Grammar (LG) and IG lies in the division of labour between logical and extra-logical machinery. Whereas in the first system logic is used for both resource and order management, in the second, logic is only devoted to resource management while constituent order is managed by the feature system. As a consequence, IG is more flexible than LG and it has a greater generative power. If it is obvious that every language generated by a Lambek grammar can be generated by an interaction grammar, the converse is not true.
Amongst all the dependencies between the syntactic components of a sentence, the only ones which are representable in the Lambek calculus are adjacent dependencies. In particular, the Lambek Calculus can represent only peripheral extraction from relative clauses and not median extraction. This is the case with our example *les élèves que Jean pense voir en entrant.*
A way of going beyond these limitations is to extend the logic. The initial kernel of the Lambek Calculus is left unchanged but it is extended with new connectives which are used to express linguistic phenomena that defy the power of the pure Lambek Calculus. This approach has been fruitfully explored by Moortgat [Moo88, Moo96], and Morrill [Mor90, Mor94], who have designed new connectives, such as unary modalities and binary operators for extraction and wrapping, and new logical systems such as multi-modal systems to extend the power of the Lambek Calculus.
The philosophy of IG takes an opposite way by relaxing the logical framework drastically: instead of the Lambek Calculus, we have chosen its commutative counterpart, more precisely its implicative fragment IILL. In doing this, we aim to represent all adjacent and long distance dependencies between syntactic constituents of a sentence in a logically uniform way and we defer the other syntactic phenomena such as word order to a non-logical level.
Such an approach was already explored by Oehrle. In [Oeh94], he proposes a labelled deductive system which takes IMLL as kernel. Each grammatical constituent is characterised by its syntactic type, which is an IMLL formula, labelled with a phonological term and a semantic term. The phonological term is a typed higher order lambda term which indicates how the phonological form of the constituent will combine with phonological forms of other constituents to compose the phonological form of a more complex constituent. The semantic term plays a similar role for semantic composition. In this way, there is a division of the grammatical labour – as Oehrle says – between the logical system and the term system which is flexible enough to go beyond the limitations of the Lambek Calculus. Oehrle illustrates this increased expressivity with the problem of quantifier scope but he could also have taken the example of median extraction which is representable in his system.
Nevertheless, Oehrle's system has some limitations with respect to IG, which come from its functional nature: every non-atomic constituent is viewed as a functor which expects other constituents as arguments and these arguments are considered as atoms so that there is no available information about their internal structure. One can see this in the phonological and semantic terms associated with the constituent. These terms are lambda-abstractions whose variables represent the phonological and semantic forms of the argument constituents: a characteristic of a variable is its atomicity. On the contrary, the syntactic trees of IG enable this formalism to express information about the structure of the arguments, which is sometimes necessary as the following example shows.

Example 2.9 *With the phrase* les élèves que Jean pense voir en entrant*, which was used as an illustration of our philosophy in this paper, let us try to represent the preposition* en *in Oehrle's system. It is a functor which takes two arguments, a gerundive clause and a definite clause to return a modified definite clause. Thus, its syntactic type has the form* $(np \multimap s) \multimap s \multimap s$ *as indicated in Example 2.1. Since the phonological form of the resulting clause is obtained by concatenation of the first definite clause, the preposition* en *and the gerundive clause, the phonological term associated with* en *has the form* $\lambda\ S\ T\ .\ T + en + S$ *where "+" is used for the concatenation operation to avoid confusion with the abstraction operation of the* λ*-calculus. The variables* S *and* T *represent the phonological forms of, respectively, the gerundive and the definite clauses, which are the arguments of the preposition.*
Now, the semantic term which is associated with the preposition en *should have the form* $\lambda\ P\ Q\ .Q \wedge P$ *but such a term would not take into account the fact that the two argument clauses the meanings of which*

are represented by the variables P *and* Q *have the same subject and there is no way of expressing this because* P *and* Q *are atomic variables. On the contrary, this is possible in IG as shown by the following syntactic tree which represents a possible lexical entry for the preposition* en.

$$
\begin{array}{l}
\left[\begin{array}{l}
+ \\
\textbf{cat} = s \\
\textbf{phon} = X_1.en.X_2 \\
\textbf{sem} = simult\ B_1\ B_2 \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_1 \\
\textbf{sem} = B_1 \\
\left[\begin{array}{l}
+ \\
\textbf{cat} = np \\
\textbf{phon} = X_3 \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1 \\
\left[\begin{array}{l}
- \\
\textbf{cat} = np \\
\textbf{phon} = X_3 \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right. \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_2 \\
\textbf{sem} = B_2 \\
\textbf{mod} = pres.part \\
\left[\begin{array}{l}
+ \\
\textbf{cat} = np \\
\textbf{phon} = \epsilon \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right.
\end{array}
$$

All explanations about this tree can be found in Example 2.7. What is interesting to note now, is that, in this tree, variable I_1 *is used as the common semantic reference for the subject of the gerundive clause and the subject of the definite clause.*

In the same direction as Oehrle, we can mention the work of Morrill [Mor95] who arrived at a similar system but starting from another motivation: his goal was to find a common framework for implementing variants and extensions of the Lambek Calculus.

2.5.2 How going from TAG to IG leads us to D-Tree Grammars

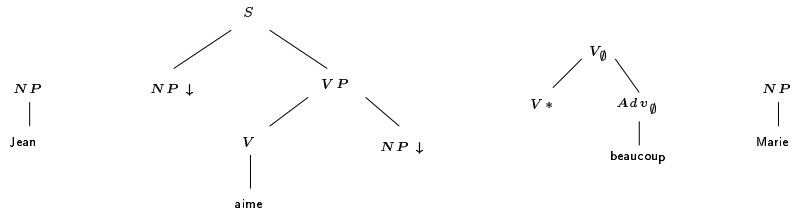
As we have mentioned in the introduction, recent work [AFV97, JK97] highlights the links between TAG and Categorial Grammar. Thus, it is not surprising to find some relationship between TAG and IG. As we have remarked in Subsection 2.1, the adjoining mechanism which is the cornerstone of TAG, is easily representable in IG. Hence, it seems to be natural to go further and to try to translate TAG into IG. The principle of such a translation is that the TAG elementary and derived trees are translated into IG syntactic trees. Then, the operations of substitution and adjoining are expressed by means of the two elementary operations of IG: merger of two dual nodes and introduction of a virtual link between two dual nodes. Substitution is represented by a simple interaction and adjoining by a specific complex interaction. A critical point lies in the representation of the adjoining constraints which are associated to each node of a tree. For the sake of simplicity, we assume that these constraints are boolean: either

adjoining is allowed at a node or not.
Moreover, whereas the order between daughters of a node in a TAG tree matters, this does not matter in an IG syntactic tree. This order, which expresses word order, will be represented by means of a phonological feature in IG in the same way as explained previously. For simplification, at first we disregard word order as well as all other grammatical features from our presentation: in this stage, the only grammatical property that we consider is membership of a syntactic category. Let us see in detail how to translate a TAG tree into an IG tree node by node:

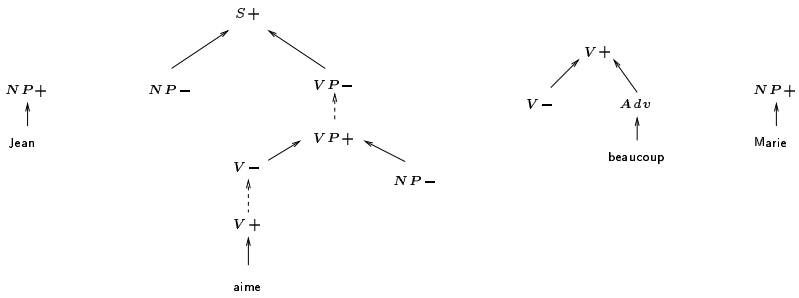
- each internal node with a null adjoining constraint is translated into a neutral node with the same label;
- each internal node where adjoining is allowed is translated into a dipole of dual nodes with the same label connected together by a virtual link;
- each terminal node is translated into a neutral node with the same label;
- each non-terminal leaf is translated into a negative node with the same label.

Finally, each edge of the TAG tree becomes an actual link of the resulting IG tree. Let us see how this translation works on a simple example.

Example 2.10 *We consider the sentence* Jean aime beaucoup Marie *and we assume that a TAG provides the following elementary trees as lexical entries used for parsing this sentence.*



The null adjoining constraint is present at all leaves and, when it appears at other nodes, it is represented by the symbol ∅ indexing the label of the node.
According to the rules described above, these trees are translated into the following IG syntactic trees.

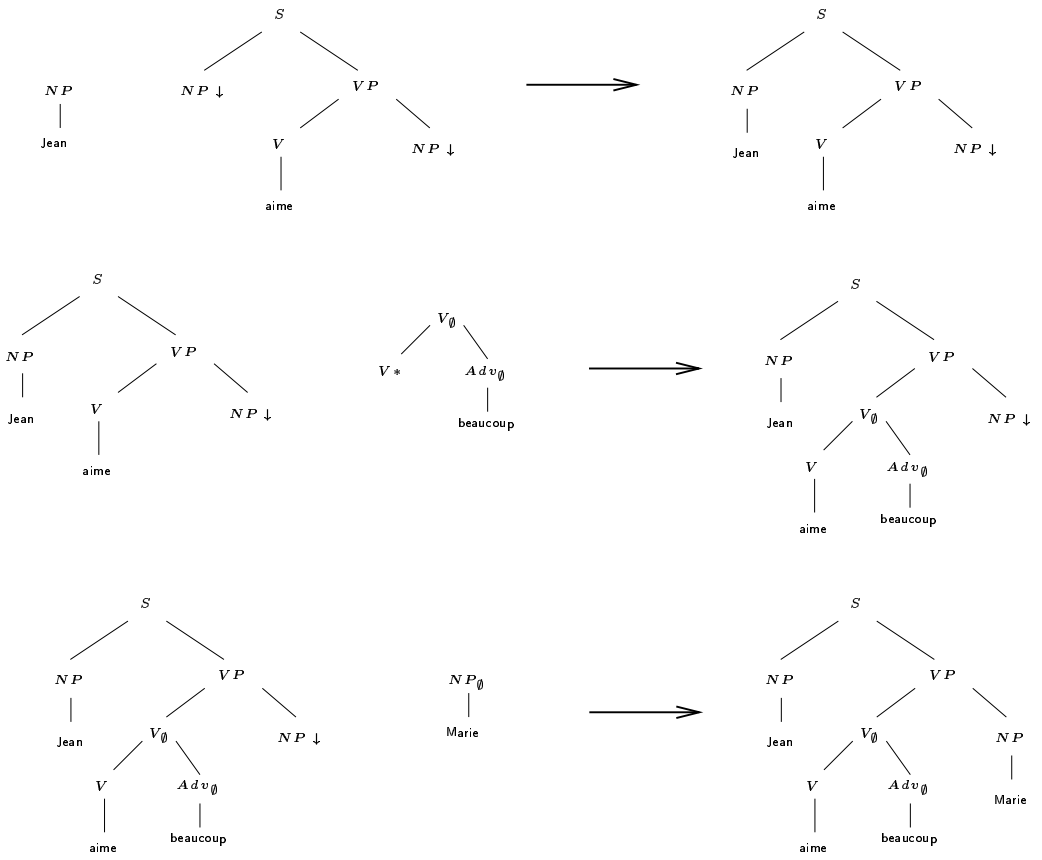


With respect to the presentation of IG trees which we have given, the above are reversed to conform to the usual presentation of TAG trees.

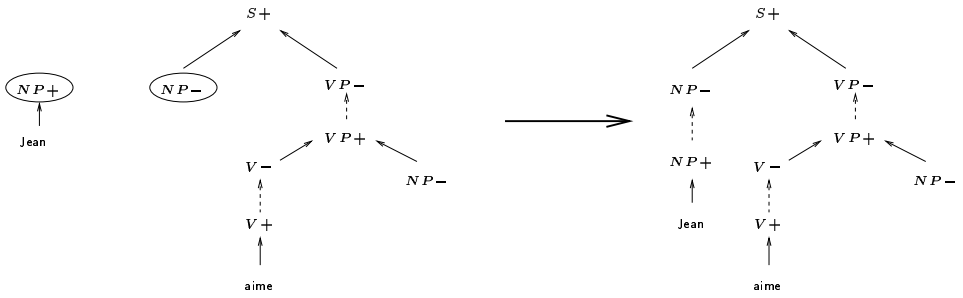
The IG trees that result from such a translation are reminiscent of the quasi-trees which were designed by Vijay-Shanker [VS92] for re-interpreting TAG in terms of constraint descriptions. In this approach, TAG trees are viewed as sets of constraints and the adjoining operation becomes an addition of constraints on such sets. In this way, parsing is viewed as a monotonic process starting from an under-specified set of constraints given by a lexicon and leading by successive addition of new constraints to a more constrained set. The solution of this final set then provides the syntactic structure of the parsed phrase.
The main feature of quasi-trees is that they can be obtained from TAG trees by dissociating the nodes where the adjoining operation is allowed and by introducing a relation of domination between the two

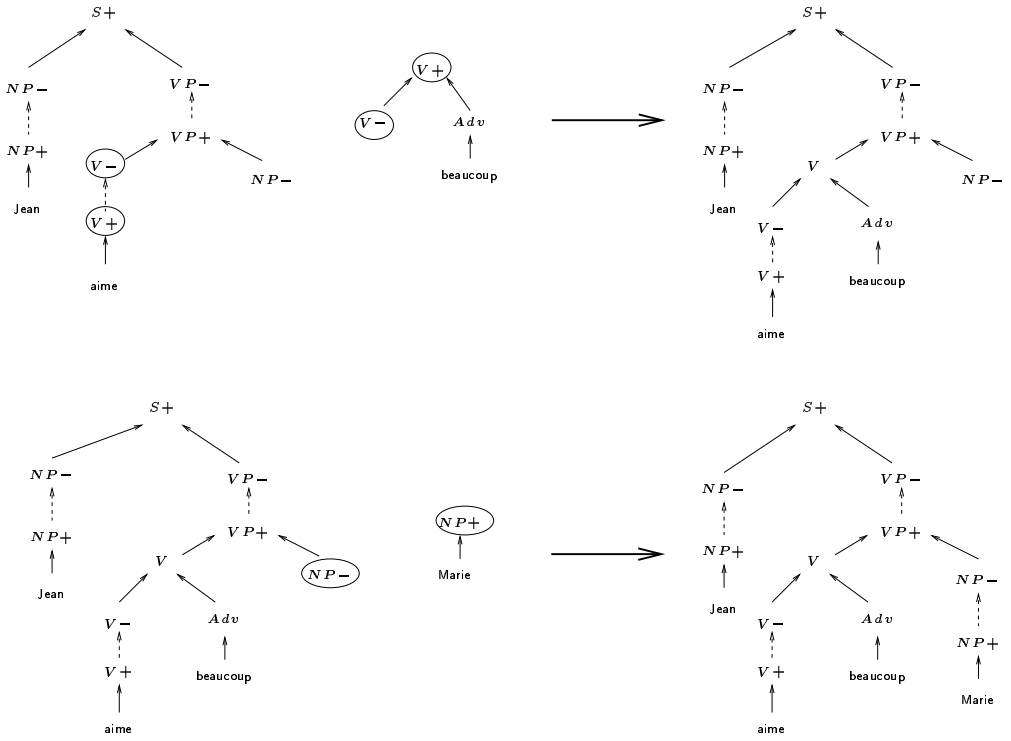
new nodes in addition to the relation of immediate domination specifying the original TAG trees. The translation of TAG trees into IG trees gives a similar outcome. We find again the dissociation of nodes and the two kinds of relation between nodes: dominance and immediate dominance, respectively, correspond to virtual and actual links. Nevertheless, there are two important differences: in IG, nodes are polarised and adjoining constraints on the roots are not expressed in the representation of IG trees; as we will see now, these constraints can be taken into account in the course of the parsing process. Let us continue with our example to see how the substitution and adjoining operation are expressed with the basic operations of IG.

Example 2.11 *Let us parse the sentence* Jean aime beaucoup Marie *with TAG from the lexical entries given by Example 2.10. The parsing process is composed of two substitutions and one adjoining operation which occur in the following order if we choose an incremental strategy:*



This TAG derivation is translated into an IG derivation as follows (the active nodes of the two interacting trees are surrounded):

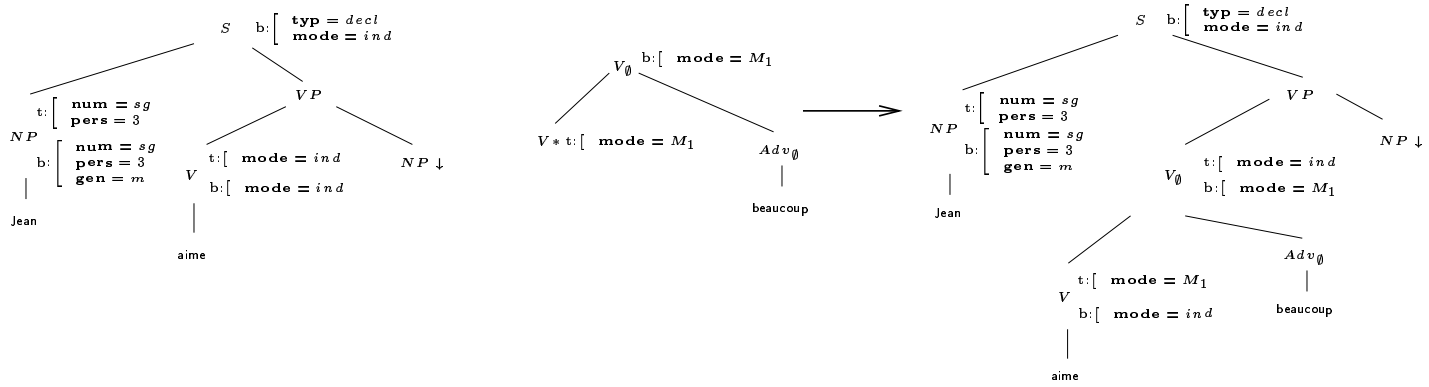




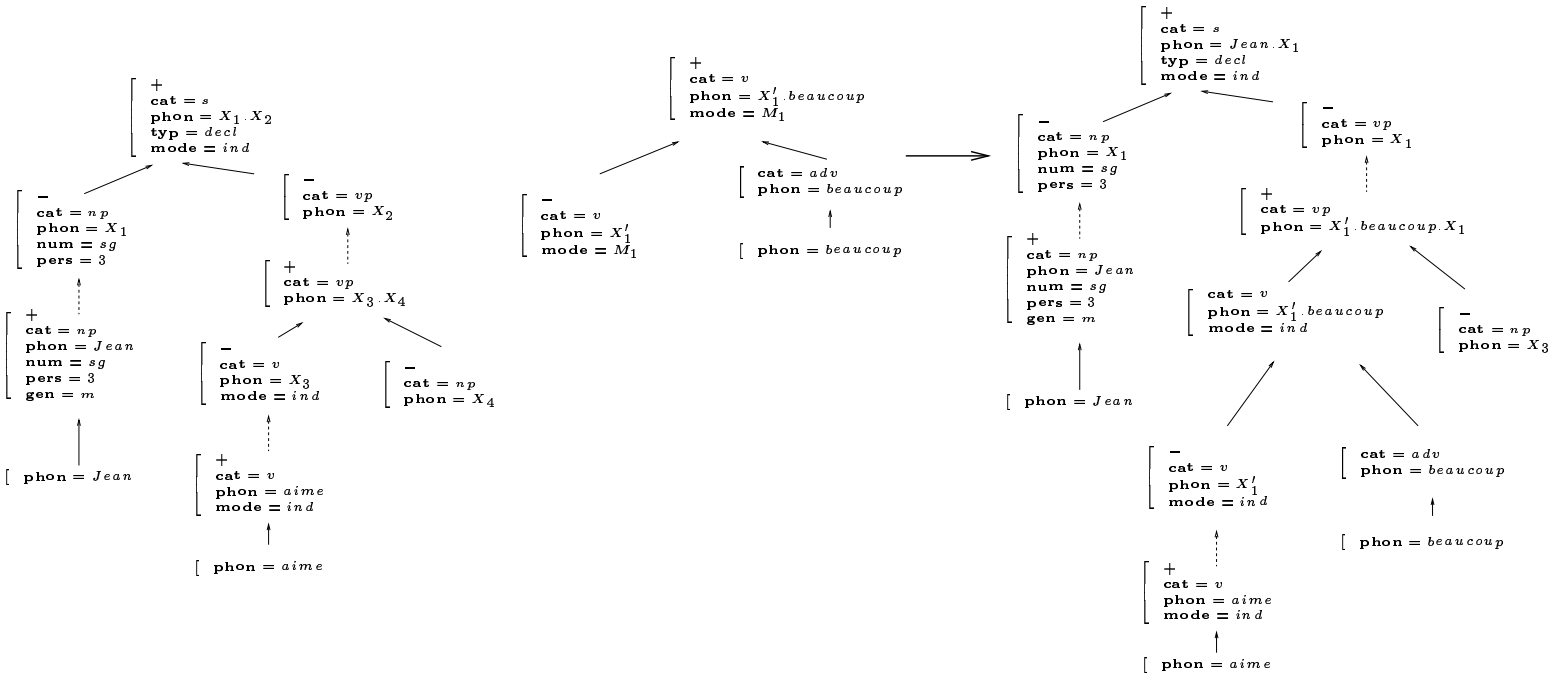
In the first and the last steps, substitution is expressed by a simple interaction which consists in adding a virtual link between two dual nodes to keep future adjunctions possible. In the second step, adjoining an auxiliary tree is expressed by a complex interaction in the form of two elementary operations: two dual nodes are merged while two others are connected by a virtual link. In the first case, no adjunction will be allowed at the resulting node contrary to the second case.

By replacing atomic categories with feature matrices, we can express both order between syntactic constituent which is inherent to the structure of TAG trees and feature structures which are added to nodes in FTAG. In FTAG, each node is equipped with two sets of features: the top set which communicates with the bottom set of the father node and the bottom set which communicates with the top set of the daughter nodes.
A reason for the introduction of quasi-trees by Vijay-Shanker was to justify this duality in the feature structure of FTAG. Such a duality appears also naturally in IG trees. In these trees, a dipole of dual nodes represents an internal node where adjoining is allowed. What is the top set of features in TAG becomes the feature matrix associated with the positive extremity of the dipole in IG whereas the bottom set of features becomes the feature matrix associated with the negative extremity of the dipole. For the root of a TAG tree, the top set is empty whereas the bottom set becomes the feature matrix associated with the positive node which is the translation of the root. For any non-terminal leaf, this is the contrary: the bottom set is empty whereas the top set becomes the feature matrix associated with the negative node which is the translation of the leaf.
In this way, it is possible to give a formal translation of FTAG into IG. We will merely give the flavour of such a translation with the help of an example.

Example 2.12 *Let us take again a derivation step from Example 2.11. TAG trees are replaced with FTAG trees where each node is labelled with a top and bottom set of features. These sets are not presented in an exhaustive way; only some features are mentioned to keep the schema readable.*



The translation of this derivation step into IG is the following [7].



What is remarkable is that Vijay-Shanker's tree descriptions gave rise to the development of a new grammatical formalism, D-Tree Grammars, which is still closer to IG [RVSW95]. The motivation was to overcome some linguistic limitations of TAG operations. The basic objects of D-Tree Grammars are Vijay-Shanker's tree descriptions but the adjoining operation and substitution are replaced with more general operations: subsertion and sister-adjunction. The first operation, when translated in the framework of IG appears as a particular form of complex interaction in the sense given previously.
Muskens and Krahmer use this model to underspecify semantics and to deal with the difficult problem of quantifier scope [MK]. It is interesting to note that they have felt the need to introduce polarisation in tree descriptions so that their model moves closer to IG though their research was conducted independently from ours.

[7] Contrary to the usual presentation of IG syntactic tree under the form of embedded matrices, we present these trees in a form similar to FTAG trees.

2.5.3 Some limitations of Interaction Grammars

Syntactic composition as tree plugging or tree superposition By confining themselves to the multiplicative fragment of linear logic, IGs are limited in their way of performing syntactic composition. The only elementary operation that is expressible in this framework consists in filling a valence of a constituent by another atomic constituent, which corresponds to inserting an axiom link in a proof net. However, from a linguistic viewpoint, it is sometimes desirable to be able to superpose syntactic trees partially in order to realize linguistic constraints without having to fill any valence.

Example 2.13 *The lexical entry of the preposition* en *given in Example 2.7 is a bit awkward in its manner of expressing the requirement that both argument clauses must have the same subject. Let us recall the syntactic tree representing this entry.*

$$
\left[\begin{array}{l}
+ \\
\textbf{cat} = s \\
\textbf{phon} = X_1.en.X_2 \\
\textbf{sem} = simult\ B_1\ B_2 \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_1 \\
\textbf{sem} = B_1 \\
\left[\begin{array}{l}
+ \\
\textbf{cat} = np \\
\textbf{phon} = X_3 \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1 \\
\left[\begin{array}{l}
- \\
\textbf{cat} = np \\
\textbf{phon} = X_3 \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right. \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_2 \\
\textbf{sem} = B_2 \\
\textbf{mod} = pres.part \\
\left[\begin{array}{l}
+ \\
\textbf{cat} = np \\
\textbf{phon} = \epsilon \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right.
\end{array}\right.
\end{array}\right.
$$

To force the subject of the definite clause, which is the first argument of the preposition en*, to also be the subject of the gerundive clause which is its second argument, we must isolate this subject by using a trick. In the syntactic tree above, we have inserted a dipole of dual np nodes: the negative node will merge with the potential subject of the definite clause and then, the positive node will return it to this clause so that it will realized as its subject. We could imagine a simpler entry in the form of the following syntactic tree:*

$$
\left[\begin{array}{l}
+ \\
\textbf{cat} = s \\
\textbf{phon} = X_1.en.X_2 \\
\textbf{sem} = simult\ B_1\ B_2 \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_1 \\
\textbf{sem} = B_1 \\
\left[\begin{array}{l}
\textbf{cat} = np \\
\textbf{phon} = X_3 \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right]
\end{array}\right] \\
\left[\begin{array}{l}
- \\
\textbf{cat} = s \\
\textbf{phon} = X_2 \\
\textbf{sem} = B_2 \\
\textbf{mod} = pres.part \\
\left[\begin{array}{l}
+ \\
\textbf{cat} = np \\
\textbf{phon} = \epsilon \\
\textbf{sem} = I_1 \\
\textbf{funct} = subj \\
\textbf{num} = N_1 \\
\textbf{gen} = G_1 \\
\textbf{pers} = P_1
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Such a syntactic tree cannot work in the usual framework of IG where neutral nodes are totally inert. Therefore, the neutral np node cannot be used to isolate the subject of a definite clause which would come as an argument of the preposition en. *Such a node would be operational only if the syntactic tree above could be partially superposed with the syntactic tree of the definite clause argument so that the neutral np node would act as a filter for the subject of the definite clause.*

We could find a lot of similar examples which suggest the use of tree superposition as another pertinent way of performing syntactic composition in addition to tree plugging. This poses the question of how tree superposition can be embedded in the IG formalism: is it preferable to extend the logical framework to additive linear logic, where superposition of proof nets becomes licit, or to abandon a strictly logical framework ?

Static features versus dynamic features Another limitation of IG lies in the rigidity and the static character of features: in the process of syntactic and semantic composition, all features are present from the beginning to the end. The only transformation that can affect them is an instantiation of their values by unification. As in a lot of other formalisms, this entails first an undesirable inflation of the entries of lexicons in order to represent all syntactic contexts where a constituent can be present. The formalism is then unable to express some linguistic phenomena occurring in reality, except by using serious contortions.

Example 2.14 *The lexical entries presented in Example 2.7 cannot be used to parse the correct phrase* les élèves que voit Jean *because the entry of the verb* voit *corresponds to the canonical setting of a sentence where the subject precedes the verb. The only way of taking this subject inversion into account is to add a new entry for the relative pronoun* que *which transforms the order verb-subject in the relative clause. It is possible to design a syntactic tree which performs this transformation but this will be very complicated and it will not reflect the linguistic phenomenon which operates here: since the relative pronoun provides the object of the verb* voit*, word order is not necessary for determining the subject so that this order can be relaxed.*

Such situations where features are context-sensitive are very frequent in natural languages and they must be taken into account by a linguistic model. A way of introducing dynamism into IG consists in lowering

polarisation down to the level of features and adding inference rules for producing new features from old ones. This is the topic of future work.

Control on the depth of dependencies A last limit of the IG formalism lies in the too great imprecision of virtual links: a virtual link from a positive constituent C_1 to a negative constituent C_2 only means that C_1 has to fill a gap inside C_2 but this can be done at any depth in C_2. Now, from a linguistic viewpoint, there are usually constraints on this depth. A first constraint can be that C_1 be an immediate constituent of C_2 as the following example illustrates.

Example 2.15 *The preposition* en, *as it is used in the key example of this paper, expects a gerundive clause which it provides with the subject. In the syntactic tree that represents the lexical entry of the preposition* en *given in Example 2.7, the relationship between the gerundive clause and its subject is expressed by a virtual link: the gerundive clause plays the role of C_2 and the subject the role of C_1 but nothing in the syntactic tree indicates the depth at which C_1 will fill its function inside C_2. In this way, it is possible to parse the following ungrammatical sentence successfully:* $*$ Jean entre en Pierre voyant que court. *In this case, C_2 is the phrase* Pierre voyant que court *and C_1 is a noun phrase with an empty phonological form and* Jean *as semantic reference. Parsing succeeds because C_1 fills the gap that corresponds to the missing subject of the verb* court *and thus both* voyant *and* court *have their valences saturated.*

Other constraints on the depth at which a constituent can fill a gap in another constituent are usually classed under the heading "barriers to extraction". For example, a relative pronoun provides the relative clause which it introduces with a noun phrase but this noun phrase cannot fill a gap in any constituent of the relative clause: some constituents are islands which cannot be entered. As the following example shows it, virtual links in IG not have the power of expressing such constraints.

Example 2.16 *The phrase* $*$ Pierre que Marie court en voyant *is ungrammatical but a parser which uses the lexical entries given in Example 2.7 will find it correct. In the syntactic tree which represents the lexical entry of the relative pronoun* que, *there is a virtual link between the relative clause and the noun phrase which is provided as object to this clause. They respectively represent the constituents C_2 and C_1. Since the virtual link gives no constraint on the depth at which C_1 will fill the gap in C_2, it will do it as the object of* voyant. *Unfortunately,* en voyant *is an island which forbids any syntactic dependence between one of its sub-constituent and an external element.*

What these examples show, is the necessity to add constraints to virtual links of IG syntactic trees to avoid overgeneration. Such a task is not technically difficult and must be taken into account in future versions of IG.

Conclusion

In this paper, we have shown how linear logic can be fruitfully used to model the syntax of natural languages. A major interest of Interaction Grammars is that, by making a synthesis between Categorial Grammars and Phrase Structure Grammars, they take advantage of both classes of formalism. Hence, they stand in comparison with many existing systems with regard to both the expressiveness and intrinsic quality of the formalism.
The limitations of these grammars which are described at the end of the paper pose the problem of relaxing the linear logic framework to fit in with linguistics. A new version of Interaction Grammars which takes this orientation is in preparation.

Acknowledgements

Thanks to Adam Cichon and François Lamarche who reread this report

References

[Abe93] A. Abeillé. *Les nouvelles syntaxes.* Linguistique. Armand Colin, 1993.

[Adj35] K. Adjukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1:1–27, 1935.

[AFV97] V. M. Abrusci, C. Fouqueré, and J. Vauzeilles. Tree adjoining grammars in noncommutative linear logic. In C. Retoré, editor, *First International Conference on Logical Aspects of Computational Linguistics, LACL'96. Nancy, France, September 1996*, volume 1328 of *LNCS*, pages 96–117. Springer Verlag, 1997.

[BH53] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[BH64] Y. Bar-Hillel. *Language and Information.* Addison-Wesley Publishing Company, 1964.

[Bre82] Joan Bresnan. *The Mental Representation of Grammatical Relations.* MIT Press, 1982.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[HO93] J. M. E. Hyland and L. Ong. Dialogue games and innocent strategies: an approach to intensional full abstraction for PCF. Manuscript, July 1993.

[JK97] A. K. Joshi and S. Kulick. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 20(6):637–667, 1997.

[JLT75] A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[Lam58] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–169, 1958.

[Lam61] J. Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects, 12th Symposium in Applied Mathematics, Providence, Rhode Island, april 1960*, pages 166–178. American Mathematical Society, 1961.

[Lam94] F. Lamarche. Proof Nets for Intuitionistic Linear Logic I: Essential Nets. Preliminary report, Imperial College, April 1994.

[Lam95] François Lamarche. Games semantics for full propositional linear logic. In D. Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 464–473, San Diego, California, June 1995.

[Lam96] F. Lamarche. From proof nets to games. *Electronic Notes in Theoretical Computer Science*, 3, 1996. Special Issue of Linear Logic'96, Tokyo Meeting, march 1996.

[MK] R. Muskens and E. Krahmer. Talking about trees and truth-conditions. In *Logical Aspects of Computational Linguistics, LACL'98. Grenoble, France,December 1998.*

[MM97] J. Merenciano and G. Morrill. Generation as deduction. In C. Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96, Nancy, september 1996*, volume 1328 of *LNCS*. Springer Verlag, 1997.

[Moo88] M. Moortgat. *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht, 1988.

[Moo96] M. Moortgart. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1996.

[Mor90] G. Morrill. Grammar and logical types. In *7th Amsterdam Colloquium*, 1990. Published as *Grammar and Logic* in Theoria, Volume LXII, Part 3, 260–293.

[Mor94] G. Morrill. *Type Logical Grammar.* Kluwer Academic Publishers, Dordrecht and Hingham, 1994.

[Mor95] G. Morrill. Higher order linear logic programming of categorial deduction,. In *EACL, Dublin*, 1995.

[Oeh94] R. Oehrle. Term-labeled categorial type systems. *Linguistics and Philosophy*, 17:633–678, 1994.

[PS94] Carl J. Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

[RVSW95] O. Rambow, K. Vijay-Shanker, and D. Weir. D-tree grammars. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 151–158, 1995.

[VS92] K. Vijay-Shanker. Using description of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517, 1992.