Lexicalized Proof-Nets in Pomset Logic & TAG

Sylvain Pogodalla

Lexicalized Proof-Nets and TAGs

Sylvain Pogodalla

Xerox Research Center Europe
6 chemin de Maupertuis
F-38240 Meylan, France
Sylvain.Pogodalla@xrce.xerox.com

Introduction

First introduced by [12], pomset linear logic can deal with linguistic aspects by inducing a partial order on words. [9] uses this property: it defines modules (or partial proof-nets) which consist in entries for words, describing both the category of the word and its behavior when interacting with other words. Then the natural question of comparing the generative power of such grammars with Tree Adjoining Grammars [7], as [6] pointed some links out, arises.

To answer this question, we propose a logical formalization of TAGs in the framework of linear logic proof-nets. We aim to model trees and operations on these trees with a restricted part of proof-nets (included in the intuitionistic ones), and we show how this kind of proof-nets expresses equivalently TAG-trees.

The first section presents all the definitions. Then, in the second section, we propose a fragment of proof-nets allowing the tree encoding and the third section defines the way we model operations on proof-nets. As replying to the second section, the fourth one allows us to come back from proof-nets to trees. Finally, section 5 shows examples of how the definitions and properties work.

1 Definitions

1.1 TAG

First, extending the original definition of TAG [7] with the substitution operation as in [15,3], we get:

Definition 1. *A* TAG *is a 5-uple* (V_N, V_T, S, I, A) *where:*

1. V_N *is a finite set of non-terminal symbols,*
2. V_T *is a finite set of terminal symbols,*
3. S *is a distinguished non-terminal symbol, the* start *symbol,*
4. I *is a set of* initial *trees,*
5. A *is a set of* auxiliary *trees.*

M. Moortgat (Ed.): LACL'98, LNAI 2014, pp. 230–250, 2001.

Initial trees represent basic sentential structures or basic categories. They have non-terminal nodes to be substituted for and serve as arguments to themselves or to auxiliary trees. A leaf (marked with a $*$) with the same label as the root node characterizes the auxiliary trees. An elementary tree is either an initial tree or an auxiliary tree.

The TAGs we are considering here will always be such that every elementary tree has at least a (terminal) node labeled by a terminal symbol, so that the TAGs are *lexicalized*.

Second, for refering trees and nodes in these trees, we use the notations [7] defined for trees on the finite aphabet V ($V = V_N \cup V_T$):

Definition 2. *γ is a* tree over V *iff it is a function from D_γ into V where the domain D_γ is a finite subset of $J^\star$ such that:*

1. *if $q \in D_\gamma, p < q$, then $p \in D_\gamma$;*
2. *if $p \cdot j \in D_\gamma, j \in J$, then $p \cdot 1, p \cdot 2, \ldots, p \cdot (j-1) \in D_\gamma$*

where $J^\star$ is the free monoid generated by J the set of all natural numbers, $\cdot$ is the binary operation, 0 is the identity and for $q \in J^\star, p \leq q$ iff there is a $r \in J^\star$ such that $q = p \cdot r$, and $p < q$ iff $b \leq q$ and $p \neq q$.

We call elements in D_γ adresses of γ. If $(p, X) \in \gamma$, then we say that X is the label of the node at the adress p in γ. We write it $\gamma(p) = X$.

Third, we require another property:

Property 1 (ϖ). A tree γ satisfies the ϖ property iff $\forall p \in D_\gamma$ such that $\gamma(p) \in V_T$ then $p = q \cdot 1$ and $q \cdot 2 \notin D_\gamma$.

It means that for a tree, if a node is terminal, labeled by a terminal symbol, then it is the unique daughter of its mother-node. Performing the two operations (substitution and adjunction) preserves this property.

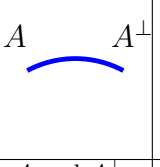
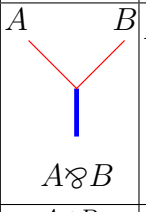
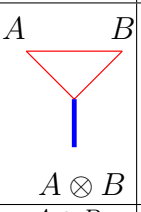
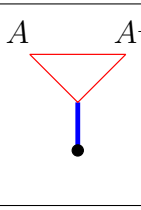
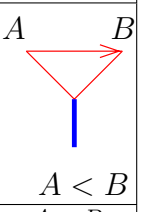
But considering TAGs whose elementary trees have the ϖ property does not restrict the generated language. Indeed, if G is a TAG whose elementary trees do not have the ϖ property, $T(G)$ is the set of all the trees that the two operations produce in the TAG G, $L(G)$ is the language that G generates (the set of strings as sequences of terminal symbol-labeled leaves of trees in $T(G)$) and if G_1 is the TAG made from G in order to get the elementary trees have the ϖ property, then we have no special relation between $T(G)$ and $T(G_1)$. Nevertheless, we have $L(G) \subset L(G_1)$.

Fourth, another restriction, similar to the restriction from [3], is to avoid the use of trees γ such that:

$$\exists p \in D_\gamma, \gamma(p) = \gamma(p \cdot 1) \text{ and } p \cdot 2 \notin D_\gamma$$

It means there is no tree that have an X-labeled node whose unique leaf is also an X-labeled node.

Table 1. Definitions of the links

Name	axiom	$\parr$	$\otimes$	`Cut`	$<$
Premises	none	A and B	A and B	A and $A^\perp$	A and B
R&B-graph					
Conclusions	A and $A^\perp$	$A \parr B$	$A \otimes B$	none	$A < B$

1.2 Lexicalized Proof-Nets

Proof-nets in linear logic have become familiar [4,12,2]. In this paper, we refer to [13]'s notations of proof-nets, extended to the ordered calculus [14]. It defines proof-nets as bicolored (Red and Blue, or Regular and Bold) graphs with the five links corresponding to the axiom, the tensor ($\otimes$), the before ($<$), the par ($\parr$) and the cut (`Cut`). This calculus enjoys cut-elimination [12], a crucial property for our modeling.

Let us remind the main definitions:

Definition 3 (RB-graphs). *A RB-graph is a graph with couloured edges (blue and red, or bold and regular). B-edges are undirected. The R-edges may be undirected or directed, in which case we call them R-arcs.*

Definition 4 (Links). *There are five sorts of links, defined as RB-graphs (see table 1).*

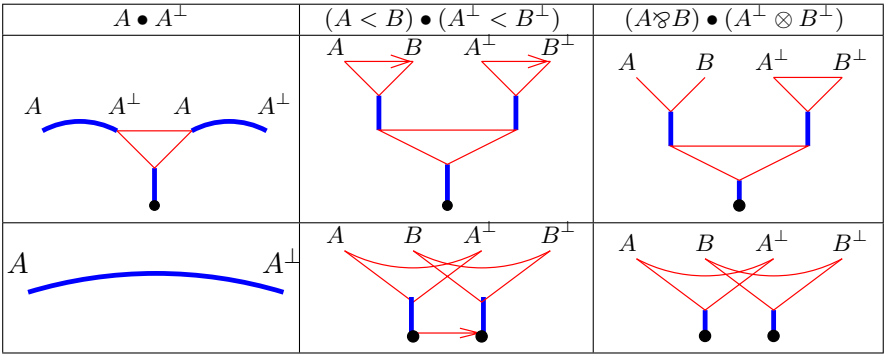
Definition 5 (Proof-structure). *A proof structure is a RB-graph such that any B-edge is the conclusion of exactly one link and the premise of at most one link (the B-edges which are not a premise of any link are called conclusions of the proof-structure, they contain all the cuts), provided with a set of R-arcs between conclusions which defines a strict partial order.*

Definition 6 (Proof-net). *An ordered proof-net is a proof-structure which contains no alternate elementary circuit*[1].

We speak about correctness criterion, or correctness checking to speak about the absence of any alternate elementary circuit in a proof-structure, so that we know wether a proof-structure is a proof-net or not.

[1] a path of even length, starting and ending on the same vertex, using only once every other vertex and with alternating blue and red edges.

Table 2. Rewriting rules on proof-nets for cut-elimination



Proposition 1 (Cut-elimination). *Cuts can be eliminated. More precisely: let Π be a proof-net whose conclusions are $F_1, F_2, \ldots, F_k, \bullet_1, \ldots, \bullet_p$ ordered by $\mathfrak{R}$ (where $F_1, \ldots, F_k$ are formulas and all the $\bullet_i$ are cuts) it is possible to rewrite Π as Π' with conclusions $F_1, \ldots, F_k$ ordered by the restriction of $\mathfrak{R}$ to these formulas. Moreover, this rewriting enjoys strong normalisation and confluence [12].*

Table 2 shows the rewriting rules on proof-nets.

We do not consider all proof-nets, but only those taking their formulas in the $\mathcal{C}$ language defined as follows: $\mathcal{A}$ is an alphabet of atomic formulas (we shall take $\mathcal{A} = V_N$) and

$$\mathcal{B}_1 ::= \mathcal{A}^\perp | \mathcal{A}^\perp \parr \mathcal{B}_1 \qquad \mathcal{B}_2 ::= \mathcal{A} | \mathcal{B}_2 < \mathcal{A}$$

$$\mathcal{C} ::= \mathcal{A} | \mathcal{A}^\perp | \mathcal{A} \otimes \mathcal{A}^\perp | \mathcal{A} \parr \mathcal{A}^\perp | \mathcal{B}_1 \parr (\mathcal{B}_2 \otimes \mathcal{A}^\perp)$$

Moreover, we always set the $\mathfrak{R}$ partial order relation to $\emptyset$.

In addition to logical formulas, we also decorate proof-nets with *labels* from a finite set of terminal symbols. Then, as a restriction of the lexicalized intuitionistic labeled proof-nets defined in [10], we define:

Definition 7. *1.* **Output:** *An* output *is either a B-edge that is labeled by a positive atom or the conclusion of a par-link between two atoms dual one from another (we call such a conclusion a* par-gate*).*

2. **Intuitionistic proof-net:** *An* intuitionistic proof-net (IPN) *is a proof-net which contains one and only one output.*

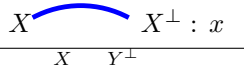
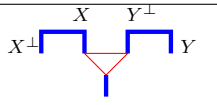
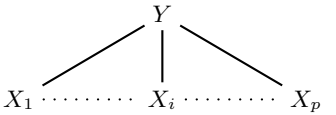
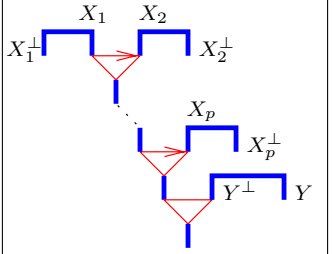
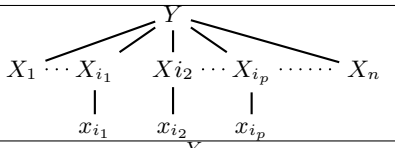
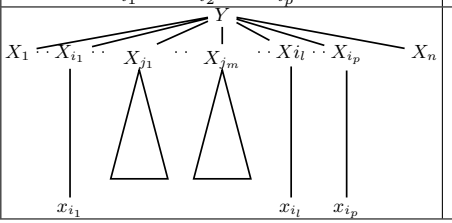
3. **Simply lexicalized proof-net:** *A* simply lexicalized (SLIPN) *is a lexicalized IPN the conclusions of which are of the form:*
   a) atoms or dual of atoms,
   b) output,
   c) $A^\perp_{j_1} \parr \cdots \parr A^\perp_{j_n} \parr ((A_1 < \cdots < A_m) \otimes Y^\perp)$ with $j_i \in [1, m] (j_i \leq j_k$ iff $i \leq k)$ and every $A^\perp_{j_i}$ is labeled by a string w_{j_i} ($A_i \in V_N$ and $w_j \in V_T$),
   d) $X \otimes X^\perp$ with X atomic (we call such a conclusion a tensor-gate*).*

Note that neither the lexicalization nor the intuitionnistic property contribute to the proof-structure correctness checking. The correctness criterion does not change (since it's (almost) the only one to handle the before-link, we would rather keep it). On the other hand, the intuitionistic feature allow the use of (a variant of) intuitionistic paths [8]. It is stable under the operations we are considering and paths enable the decoding from proof nets to trees (see section 4.2).

2 From Trees to Proof-Nets

This section defines for each elementary tree of a TAG a corresponding SLIPN with an induction on the height of the trees. The set of atoms for logical formulas comes from V_N, and labels come from V_T.

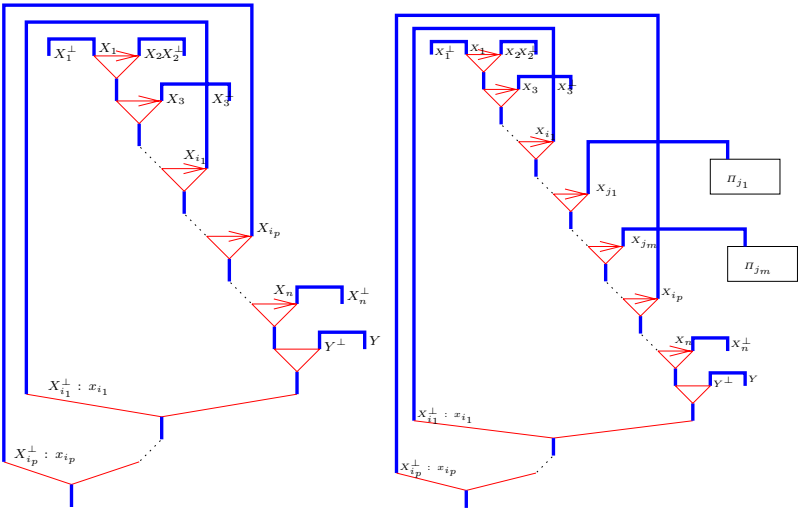
Table 3. Initial trees maping

	Trees	Proof-nets
$h = 1$		
		
		
$h = 2$		(see figure 1(a))
general case		(see figure 1(b))

2.1 Initial Trees

We first give the general idea of this encoding on the tree T_2 of table 5. Forgetting the lexicalized part, we can read this tree as a terminal node (N) preceding another terminal node (V) to produce their mother-node (P). We can express this idea with a formula of pomset logic: $(N < V) \multimap P$. But do not forget that this is a brick from which we want to derive S. Moreover, proof-nets correspond to one-sided sequent, so that, actually, we are more intersted in the dual of such formula, namely: $(N < V) \otimes P^{\perp}$. Thus, we shall have a SLIPN with this latter sub-formula, other connectives dealing with the lexicalization.

Table 3 sums up the translation. Note that for $h = 1$, the two latter cases do not belong to the considered TAG. Nevertheless, we require the definition of their corresponding SLIPNs for the next steps of the induction. The case $h = 2$ only presents the case where lexicalized subtrees' height (at least one exists) is 1 and other subtrees' height is 0, for we deal with the cases where other subtrees' height can be 1 in the next general case. For this latter, we possibly have $\{i_1, \ldots, i_p\} = \emptyset$ and in figure 1(b) the $\Pi_{j_1}, \ldots, \Pi_{j_m}$ are the inductivily built SLIPNs corresponding to the subtrees of γ at $X_{j_1}, \ldots, X_{j_m}$.



(a) SLIPNs corresponding to trees of height 2

(b) SLIPNs corresponding to trees in the general cases

Fig. 1. SLIPNs for higher trees

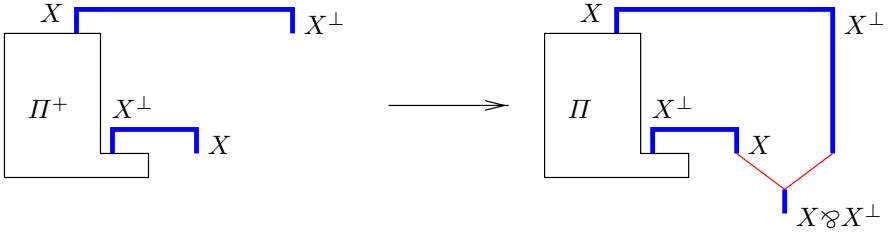


Fig. 2. Auxiliary trees

Definition 8. *For every initial tree T, we call* corresponding initial SLIPN*, or* T transformation*, the SLIPN defined as in table 3 and figure 1.*

Remark 1.
1. The unique output of every SLIPN corresponds to the root node of the tree, and every terminal node, labeled by a non-terminal symbol, of a tree corresponds to a conclusion of the corresponding SLIPN.
2. There is a one-to-one maping between the non-terminal symbol labeled nodes of a tree and the axiom links of the corresponding SLIPN.

2.2 Auxiliary Trees

Let γ be an auxiliary tree and let us define γ^+ as the same tree as γ except for its X^* node replaced with an X node. We call r the adress of N^* in γ so that $\gamma(r) = X^*$ and $\gamma^+(r) = X$. Then, following the definition in the previous section, we can define Π^+ the SLIPN corresponding to γ^+. And, as γ is an auxiliary tree, $\gamma^+(r) = \gamma^+(0)$ and Π^+ has a conclusion X (corresponding to $\gamma^+(0)$) and a conclusion $X^\perp$ (corresponding to $\gamma^+(r)$).

Thus we define Π the corresponding SLPIN to γ as the proof-net built from Π^+ in binding with a $\parr$-link its X and its $X^\perp$ conclusions (see figure 2). Π is a (correct) SLIPN.

Definition 9. *For every auxiliary tree γ, we call* auxiliary corresponding SLIPN*, or* γ transformation*, the SLIPN defined as above. Then, for every elementary tree, we call* corresponding SLIPN *the initial or auxiliary SLIPN corresponding to that tree.*

3 Elementary Operations

This section deals with a particular case of the next section but focuses on the core operations which we can refer to during the generalisation.

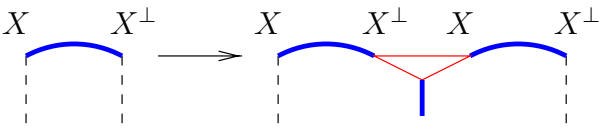


Fig. 3. Adding a tensor-gate

3.1 The Substitution Operation

Let γ_1 and γ_2 be two trees such that we can substitute γ_2 to a terminal node X (whose adress is r) of γ_1, and let Π_1 and Π_2 be their corresponding SLIPNs. Then $\gamma_1(r) = \gamma_2(0)$, and (cf. remark 1) Π_1 has a conclusion $X^\perp$, and the output of Π_2 is X.

Thus we can bind these conclusions with a cut-link and yield a new SLIPN from which we eliminate the cut and obtain a new SLIPN Π.

Definition 10. *For every tree γ resulting from the substitution of γ_2 to a node of γ_1, we define its* corresponding SLIPN *Π as above.*

3.2 The Adjunction Operation

Preparing the Target Tree. In order to allow the adjunction of an auxiliary tree on a target tree, we need to modify a little bit this latter.

Let γ be the tree on which we want to perform an adjunction, r the adress of the node where to perform the adjunction, and Π the corresponding SLIPN. As noted in remark 1, Π contains an axiom link $\gamma(r)$ ⌜⌝ $\gamma(r)^\perp$ corresponding to the node $\gamma(r)$. So we can split this link into two axiom-links linked with a tensor-link and we obtain (with $X = \gamma(r)$) a new SLIPN Π' as shown in figure 3.

Proposition 2. *Adding a tensor-gate preserves the correctness of the proof-net.*

Actually, such a conclusion is an instance of cuts (as a cut is equivalent to a conclusion $(\exists X)(X \otimes X)$). Then if this tensor-gate remains unused, cut-elimination amounts to delete this tensor-gate and come back to a simple axiom-link. The second example of section 5 uses this feature at the very end of the derivation.

Performing the Operation. In this section, we define the SLIPN corresponding to the result of adjoining the tree γ_2 to γ_1. In this preliminary case, both γ_1 and γ_2 are elementary trees (γ_2 is an auxiliary tree and γ_1 is the target tree). And Π_1 and Π_2 correspond to them.

We assume Π_1 already has a tensor-gare added on the axiom corresponding to the node where we want to adjoin γ_2. So that if X labels the stared-node of γ_2, then X also labels the node of γ_1 receiving the adjunction and we have a

tensor-gate conclusion $X \otimes X^{\perp}$ for Π_1 and a par-gate $X \parr X^{\perp}$ (the output) for Π_2 (merely by construction).

Thus, we can bind Π_1 and Π_2 with a cut-link and eliminate this cut to obtain a new SLIPN (because there is no modification on the lexicalized parts, still no alternate elementary circuit and still only one output: γ_1's one).

Definition 11. *For every tree γ resulting from the adjunction of the auxiliary tree γ_2 on γ_1, we call the* corresponding SLIPN *the SLIPN built as above.*

3.3 Operations on Derived Trees

Up to now, we defined a way of modeling elementary trees in the framework of SLIPNs, and a way of combining these SLIPNs to model the adjunction and substitution operations on elementary trees. We now are about to extend this modeling on derived trees, so that a SLIPN will correspond to every tree (elementary or derived) of a TAG.

Definition 12. *For a tree γ and an elementary tree E_0, we call* derivation *of γ the pair $< E_0, ((\diamond_1, E_1, \gamma_1), \ldots, (\diamond_n, E_n, \gamma_n)) >$ such that $\gamma_n = \gamma$ and for every $i \in [1, n]$, γ_i results from the operation $\diamond_i$ (adjunction or substitution) between γ_{i-1} and the elementary tree E_i.*

n is the length *of the derivation.*

Remark 2. For a derived tree, the derivation is not necessarily unique.

Definition 13. *Let γ be a derived tree from the derivation d. Then we can define the* d-SLIPN corresponding *to γ, built only with cut and cut-elimination (between unlabeled conclusions) from the SLIPNs corresponding to the elementary trees of the derivation.*

Actually, proving the existence of this SLIPN interests us more than the simple definition, as it also gives its construction's steps.

Proof. We prove the existence of $\Pi^{(d)}$ by induction. We also prove the property that if γ has a terminal node (except for the stared node of an auxiliary tree), labeled by a non terminal symbol X, then $\Pi^{(d)}$ has a pendant conclusion $X^{\perp}$ (corresponding to this node, hence not labeled neither).

1. if $l = 0 : \gamma$ is an elementary tree, and we already defined its transformation $\Pi^{(0)}$. And its construction also proves the property of the pendant conclusion.
2. if $l > 0$: Let $d_{l-1} =< \gamma_0, ((\diamond_1, E_1, \gamma_1), \ldots, (\diamond l-1, E_{l-1}, \gamma_{l-1})) >$ and $\Pi^{(d_l-1)}$ be the SLIPN corresponding to γ_{l-1} in the d-derivation.

a) If $\diamond_l$ is the substitution of a leaf X of γ_{l-1} by E_l (whose transformation is π_l) then $\Pi^{(d_{l-1})}$ has an axiom-link $X \sqcap X^\perp$ in which $X^\perp$ is a pendant conclusion (induction hypothesis), and π_l has an axiom-link $X \sqcap X^\perp$ in which X is a pendant conclusion ($E_l(0) = X$). Then we can link these two pendant conclusions with a cut-link, and eliminate it. This yields a new SLIPN. It also proves the property of pendant conclusion, as every terminal node of the new tree, labeled with a non-terminal symbol, already was terminal in one or another of the two trees so that (by induction hypothesis) they already had the property.

b) if $\diamond_l$ is the adjunction on the leave X of γ_{l-1} of the auxiliary tree E_l (whose transformation is π_l) then γ_{l-1} has an axiom-link $F \sqcap F^\perp$ we can replace (with respect to the SLIPNs class belonging) with two axiom-links linked together with a tensor-link (i.e. we add a tensor gate $X \otimes X^\perp$ as for adjunctions on elementary trees). π_l has a par-gate $X \parr X^\perp$ so that we can bind the two gates with a cut-link, and then eliminate this latter. We obtain a new SLIPN, and as above, the induction proves the property of the pendant conclusion.

□

Remark 3. This shows that cuts are only between atomic formulas or tensor and par-gate. Actually, the grammar given for the conclusions of SLIPNs indicates that no other cut can occur.

During this section, we made the assumption of allowing adjunctions at every time on every node. Of course, sometimes we do not want such possibilities. Allowing the tensor-gate addition only in the lexicon, and not during the derivation, brings a solution to this option.

Section 5 shows examples for both cases. In particular, the mildly-context sensitivity of TAGs, generating $\{a^n b^n c^n d^n\}$, illustrates the second case.

4 From SLIPNs to Trees

So far, we explained how, given a TAG and a derived tree in this TAG, we could obtain a SLIPN that we qualify as corresponding. But we now have to see how this SLIPN actually corresponds so that we shall henceforth be able to handle only proof-nets and translate the results on trees.

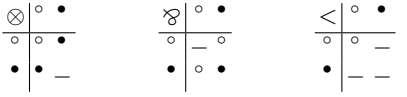
4.1 Polarities

Let us define a positive polarity ($^\circ$) and a negative one ($^\bullet$). Every formula is inductiveley polarized as follows: if α is an atom then α° and $\alpha^{\perp\bullet}$. Then, for each link we define the polarity of the conclusion from premises' ones as in table 4.

We call *input* the negative conclusions, and *output* the positive one (which is coherent with the previous definition of the output).

The grammar on SLIPNs' conclusions shows that SLIPNs are polarized.

Table 4. Polarities of the conclusions



4.2 Reading of SLIPNs

We give an algorithm for the reading of any cut-free SLIPN, based on formulas' polarities. It uses a very simple principle: following from a starting point (namely the output) the positive polarities, we define a path across the proof-net. And every time the path crosses an axiom-link, we add a node to the tree under construction. Actually, we build both the function γ and D_γ. Of course, the path can not cross twice the same axiom-link (such a possibility would occur only with par-gate).

As we shall see later, the path never cross a par-link (except par-gates, from the positive conclusion), always enter a tensor-link through a negative premise and always enter a before-link throuh the positive conclusion. So, because of the before-link, the path is not linear (both premises, positive ones, are likely to be the next on the path), and we define the *first branch* as the path from the positive premise of the before-link at the beginning of the arrow, and the *second branch* as the path from the other premise.

Then, when adding a new node on the tree, its mother-node is the the last node met on the same branch of the path (in a before-link, both premises are on the same branch as the conclusion, but they are themselves on different branchs; other connectives do not create branchs). So that if its mother-node's adress is p, then the new node's adress is $p \cdot j$ with $j \in \mathbb{N}^*$ and for all i such that $0 < i < j$, $p \cdot i \in D_\gamma$

Then, we state the algorithm as follows:

1. Enter the net through the only output (so, if X is the output, $0 \in D_\gamma$, and $(0, X) \in \gamma$).
2. Follow the path defined by the positive polarities until reaching an atom (when a before-link is crossed, first choose the premise at the beginning of the red arc) and cross it. If its conclusions are X and $X^\perp$, we define its adress p as precised above (wrt the branchs) and $p \in D_\gamma$ and $(p, X) \in \gamma$.
3. a) if the input is lexicalized, then lexicalize the last written node of the tree under construction (if the lexicalization is x, we then add $p \cdot 1 \in D_\gamma$ and $(p \cdot 1, x) \in \gamma$). Either there is no more link after, or this input is premise of a par-link whose other polarities are negative. In both cases, come back to the last before-link the path did not go through the two branchs and make as in 2.
   b) else juste follow as in 2
4. Stop when the path joined all the atoms.

The next section will show that every SLIPN built from elementary SLIPNs can be read such a way and that the path cross every axiom-link.

Remark 4. 1. This reading provides a unique correspondance between any axiom link of the SLIPN and a node (labeled by a non terminal symbol). Moreover, if the negative conclusion of an axiom-link is pendant and not lexicalized, then it corresponds to a terminal node of the tree.
2. Two different SLIPNs can have the same reading. It underlines the importance of making precise a base of elementary SLIPNs (corresponding to the elementary trees of a given TAG).

4.3 From SLPINs, Back to Trees

We now have both a maping from trees to SLIPNs, and a maping from SLIPNs to trees. It remains us to see if the composition of these mapings gives the identity.

This consists in three steps:

1. check that the reading of a SLIPN corresponding to an elementary tree is the same as the elementary tree;
2. check that the reading of a SLIPN corresponding to the substitution between two trees is the resulting tree;
3. check that the reading of a SLIPN corresponding to the adjunction between two trees is the resulting tree.

Moreover, given a TAG, the basic bricks we consider are SLIPNs corresponding to elementary trees of this TAG. Then the only way to build new SLIPNs is binding them with cut between unlabeled conclusions.

Let us remind the definition of subtrees and supertrees as in [7]

Definition 14. *Let γ be a tree and $p \in D_\gamma$. Then*

$$\gamma/p = \{(q, X) | (p \cdot q, X) \in \gamma, q \in J^\star\}$$

$$\gamma \backslash p = \{(q, X) | (q, X) \in \gamma, p \not< q\}$$

γ/p is called the subtree of γ *at p and $\gamma\backslash p$ is called the* supertree of γ *at p. Further, for $p \in J^\star$*

$$p \cdot \gamma = \{(p \cdot q, X) | (q, X) \in \gamma\}$$

Property 2. $\gamma = \gamma\backslash p \cup p \cdot (\gamma/p)$ for every tree γ and $p \in D_\gamma$.

Remark 5. If the reading of a SLIPN Π gives γ, and if we make the path begin at any positive conclusion of an axiom link that corresponds to the node at adress p in γ, then the reading algorithm returns γ/p. And of course, the reading of Π, with a pruning at the same axiom link returns $\gamma\backslash p$.

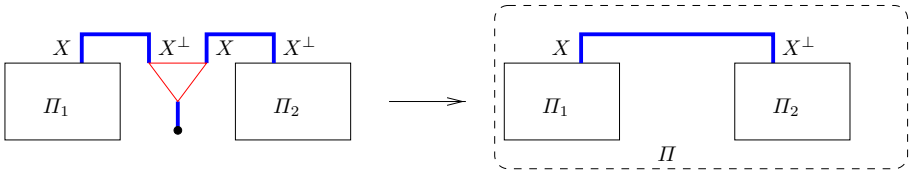


Fig. 4. Substitution

Elementary Reading. Let γ be an elementary tree. To prove the reading of the SLIPN corresponding to γ being γ itself, we simply proceed by induction, with the same steps as for the building of elementary SLIPNs.

Substitution. Let us consider two SLIPNs Π_1 and Π_2, whose readings are respectively γ_1 and γ_2. We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

Proposition 3. *If a negative (not labeled) atomic conclusion of Π_1 and a positive atomic conclusion of Π_2 support a cut-linking, then the corresponding terminal node of γ_1 accepts a substitution by the root of γ_2. And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Proof. Let p be the adress of X in the reading γ_1 of Π_1, where X corresponds to the axiom link of figure 4 (on the left). The adress of X in the reading γ_2 of Π_2 is 0 (the root node) and the reading of γ_2 starts at this X axiom-link. On the other hand, the reading of γ_1 stops at X for its branch. After the cut and the cut-elimination, the reading of the new SLIPN (on the right of figure 4) starts at the output, which also was the output of Π_1, and continues like for γ_1 until the new X axiom-link is reached. Its adress in the new tree γ is also p. There, the reading of γ_2 takes place. So that, as defined in the algorithm, if γ is the reading of the new SLIPN,

$$\forall q \in D_{\gamma_2}, \gamma(p \cdot q) = \gamma_2(q)$$

and nothing changes for the remaining reading: it is the same as for γ_1 (because $\gamma_1 = \gamma \backslash p$). Then the reading γ of Π is such that

$$\gamma = \gamma_1 \cup p \cdot \gamma_2$$

which corresponds to the definition of the substitution of the $\gamma_1(p)$ node with γ_2. □

Adjunction. As above, let us consider two SLIPNs Π_1 and Π_2, whose readings are respectively γ_1 and γ_2. We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

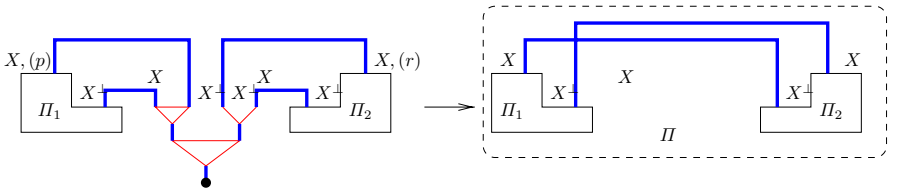


Fig. 5. Adjunction

Proposition 4. *If a tensor-gate of Π_1 and the unique positive conclusion of Π_2 support a cut-linking, then the node corresponding to the tensor-gate on γ_1 accepts an adjunction of γ_2. And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Proof. In the following, when dealing with γ_1, we speak about the reading of Π_1 *without* the tensor-gate.

Let us consider Π on figure 5 (the SLIPN on the right). The input of Π is the same as Π_1. As the positive conclusion of an axiom-link always occurs before the negative conclusion in the path, then new tree γ from Π, after reaching X in Π_1 cross the axiom-link to Π_2, so that $\gamma/p \neq \gamma_1/p$ but $\gamma \backslash p = \gamma_1 \backslash p$ (see remark 5). Yet, The $X^\perp$ on Π_2 is the other conclusion of the starting axiom-link for γ_2. So that at the p adress, for γ, we read γ_2. Then for every $q \in D_{\gamma_2}$, $\gamma(p \cdot q) = \gamma_2(q)$.

Moreover, reaching the X of Π_2, the path does not stop but continue with the remaining part of γ_1, namely γ_1/p. So that at the new adress of X of Π_2 in γ we add γ_1/p. And the new adress of X in γ is $p \cdot r$ (with r the adress of X^* in γ_2).

Then

$$\gamma = \gamma_1 \backslash p \cup p \cdot \gamma_2 \cup p \cdot r \cdot \gamma_1/p$$

which is the definition of the adjunction of γ_2 on γ_1 at X. □

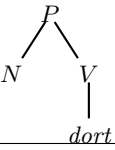
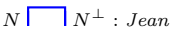
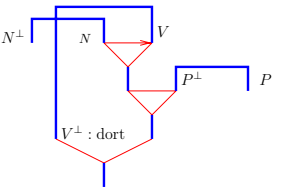
Eventually, we can state the next propositions:

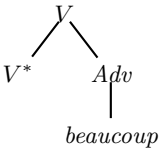
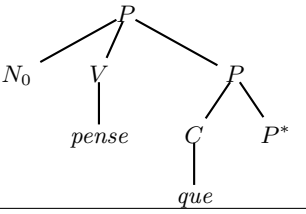
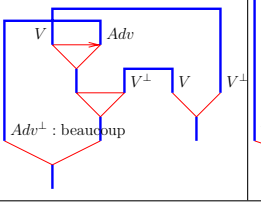
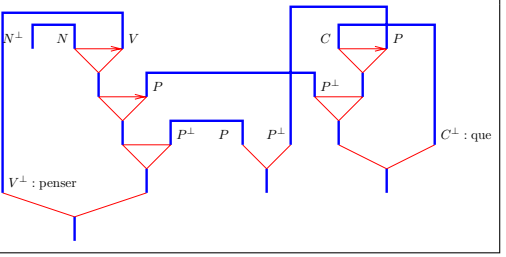
Proposition 5. *Every derived tree (from an elementary tree lexicon) corresponds to the reading of a SLIPN, the latter resulting from* `Cut` *operations between SLIPNs corresponding to the elementary trees of the lexicon.*

Reciprocally, with a lexicon of elementary SLIPNs corresponding to trees, with the restriction of `Cut` operations on formulas that are not lexicalized, the reading of the resulting SLIPNs are the derived trees.

Proof. This is immediate after the previous propositions. □

Table 5. Lexicon

	Jean(T_1)	Dort(T_2)
Trees		
SLIPNs		

	Beaucoup(T_3)	Pense que(T_4)
Trees		
SLIPNs		

5 Examples

5.1 Substitution and Adjunction

First let us define from the lexicon of the TAG the corresponding elementary SLIPNs. We assume the lexicon of table 5. This lexicon can yields the trees of figure 6 (for the first tree: substituting N in T_2 with T_1, then adjoining T_3 on the result. For the second tree: continue with the adjunction of T_4). But we can also make this derivation on the SLIPNs as shown in the figures 7 and 8.

Let us see how to read the SLIPN of figure 7(e), and obtain the derived tree of figure 6(a): first the path enters the net through the atom P (the unique

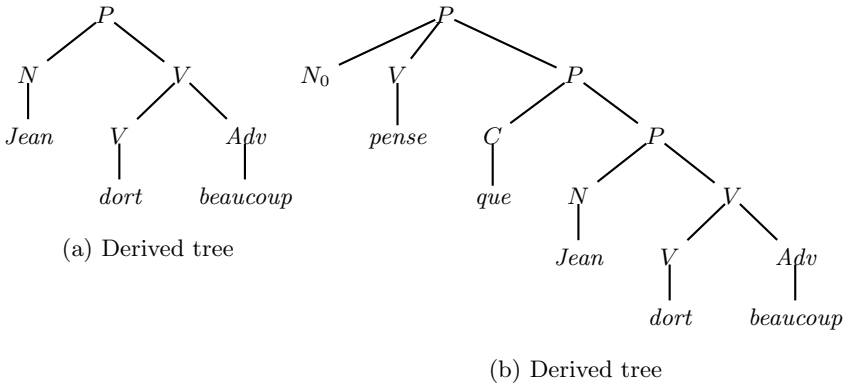


Fig. 6. Resulting trees

output) and marks P as a node. Then it follows the positive polarities and reaches a before-link with two branches. It first chooses the positive formula at the beginning of the red (regular) arc and crosses an axiom-link. There is a negative lexicalized atom. So on the tree we add a lexicalized (*Jean*) node N. Then doing the same with the other premise of the before-link we get a new atom V and the negative conclusion of the axiom-link is not lexicalized. So we have a junction which will produce new branches under the V node. They are two simple branches: one is a (lexicalized by *dort*) V, and the other is a (lexicalized by *beaucoup*) *Adv*.

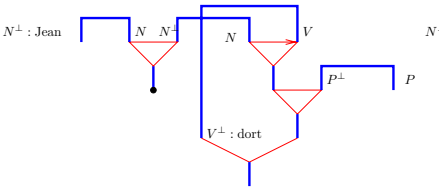
To have a deeper adjunction, let us continue with the adjunction of T_4. Figure 8 shows the different steps of the operation. But we leave the reader check that polarizing the resulting SLIPN of figure **??** and reading it leads to the tree of figure 6(b).

5.2 A Formal Language

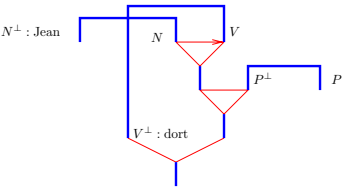
As in the previous section, we first define the lexicon of table 6. Note that in this lexicon, the tensor gate belongs to the lexical item, so that we shall never use a tensor-gate addition during a derivation.

We only initiate the use of this lexicon with an adjunction of T_2 on another instance of T_2 (figure 9(a)), then an adjunction on T_1, resulting in the SLPIN of figure 9(c). At every adjunction on T_2, a new tensor-gate appear (provided by T_2), as the former (on the derived SLIPN) disappears with the adjunction operation (the cut between the par-gate and the tensor gate of T_2). As the reader can check, the reading actually corresponds to our expectations and generates the word *aabbccdd*.

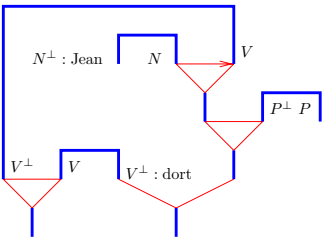
Thus, without splitting any axiom-link and adding any tensor-gate during the derivation, we can generate the language $\{a^n b^n c^n d^n\}$.



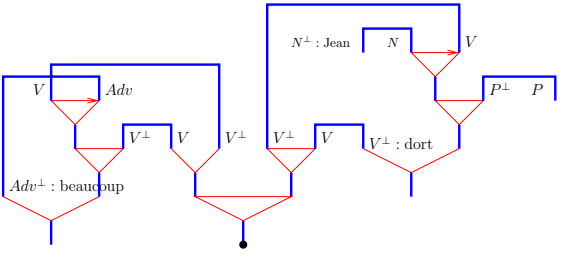
(a) Substitution of the N node of *dort* by *Jean*



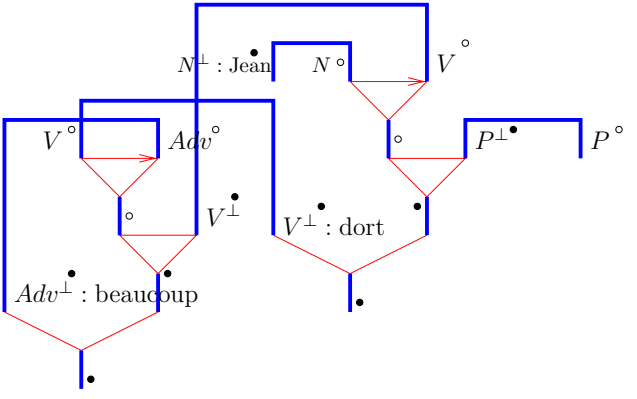
(b) Substitution (continued): cut-elimination



(c) Addition of a tensor-gate on *Jean dort*

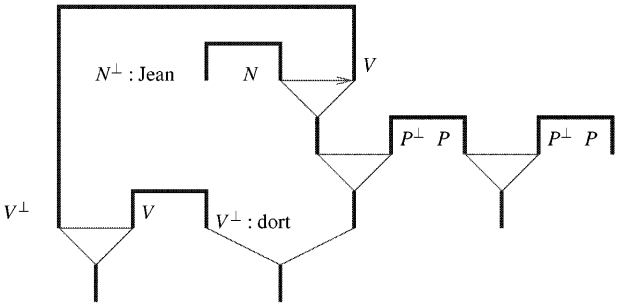


(d) Adjunction of *beaucoup* on *Jean dort*

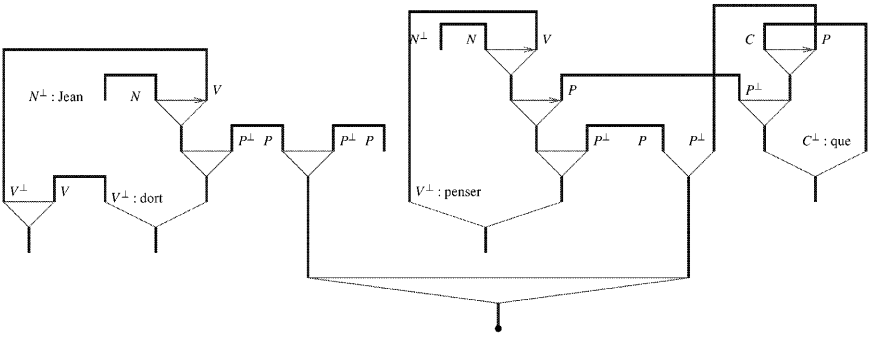


(e) Adjunction (continued): cut-elimination and polarization

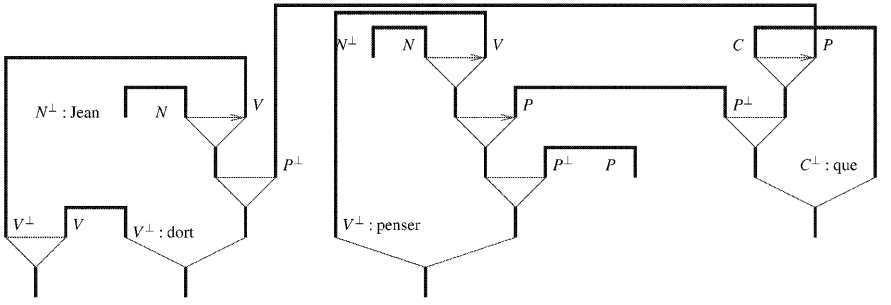
Fig. 7. Operating on SLIPNs



(a) Addition of a tensor-gate

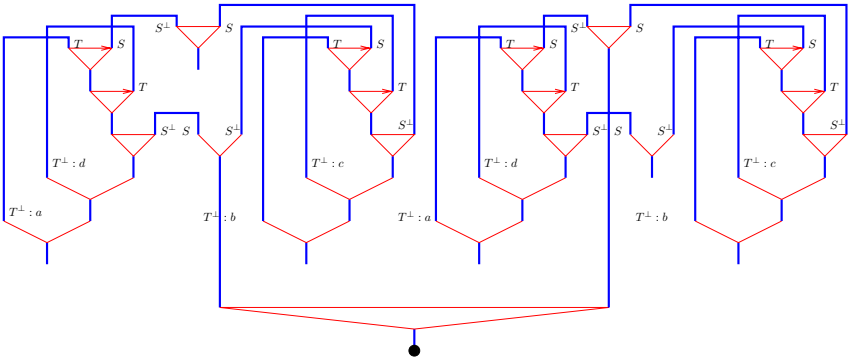


(b) Adjunction of *pense que* on *Jean dort beaucoup*

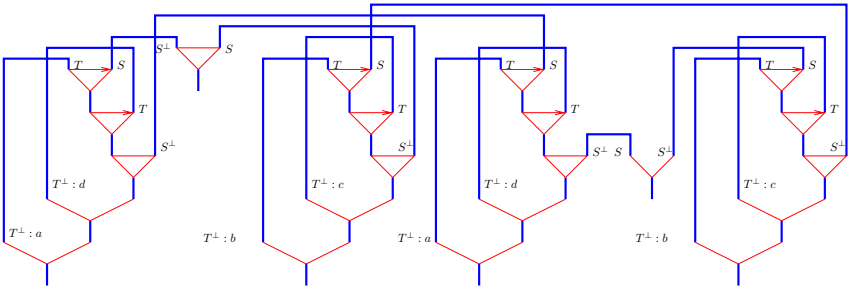


(c) Adjunction (continued): cut-elimination

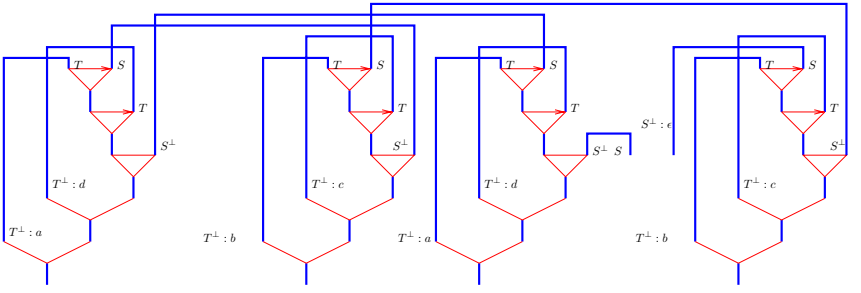
Fig. 8. Operating on SLIPNs (continued)



(a) Adjunction of T_2 on another instance of T_2



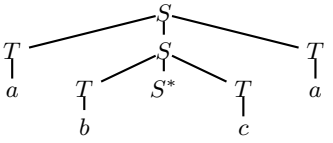
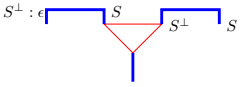
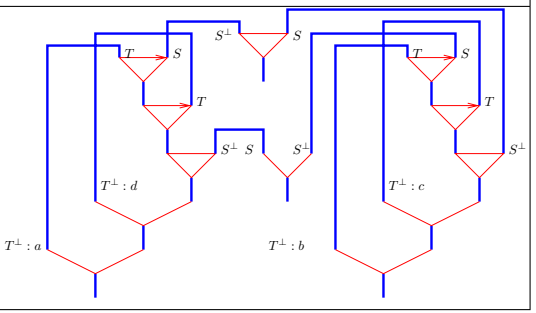
(b) Cut-elimination



(c) Adjunction onT_1 and cut-elimination

Fig. 9. Generating *aabbccdd*

Table 6. Lexicon

	Empty (T_1)	Main tree (T_2)
Trees		
SLIPNs		

Conclusion

Using a restricted fragment of pomset intuitionistic proof-nets, we showed how to generate the same language as a TAG. This indicates how a more generic formalization (namely [11]'s one) allows both keeping generative power and dealing with some linguistic phenomena not by lexical rewriting rules on trees, but by lexical definitions. For instance, we can compare the modeling of clitics in [1] or in [11].

We also want to underline that we do not really use the partial order capabilities of pomset proof-nets: the before-links arrange totally the atoms in order. Of course, this results straightforwardly from the fact that the order in the trees is total, so that the same occurs in the SLIPNs with respect to the before-links.

Moreover, we use both commutative and non-commutative connectors, and the building of the path defines the order of the lexical items. The path performs the splitting of the sequent required in the rules (especially the adjunction rule) of [3].

Finally, to know how to express the semantics, at least two possibilities arise: to see it as for intuitionistic proof-nets [5], or as having an alternative expression like with the derivation trees (trees that track the operations performed during a derivation).

References

1. Abeillé, A. *Les nouvelles syntaxes*. Armand Colin Éditeur, Paris, Armand Colin Éditeur, 103, boulevard St-Michel - 75240 Paris, 1993.
2. Abrusci, V. M. Non-commutative proof nets. In Girard, J.-Y., Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 271–296. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.
3. Abrusci, V. M., C. Fouqueré, and J. Vauzeilles. Tree adjoining grammars in non-commutative linear logic. In Retoré, C., editor, *LACL'96*, volume 1328 of *LNCS/LNAI*, pages 96–117. Springer-Verlag Inc., New-York, USA, September23–25 1996.
4. Girard, J.-Y. Linear logic. In *Theoretical Computer Science*, 50:1–102, 1987.
5. de Groote, P. and C. Retoré. On the semantic readings of proof-nets. In Geert-Jan Kruijff, G. M. and D. Oehrle, editors, *Formal Grammar*, pages 57–70. FoLLI, Prague, August 1996.
6. Joshi, A. K. and S. Kulick. Partial proof trees, resource sensitive logics and syntactic constraints. In Retoré, C., editor, *LACL'96*, volume 1328 of *LNCS/LNAI*, pages 21–42. Springer-Verlag Inc., New-York, USA, September23–25 1996.
7. Joshi, A. K., L. S. Levy, and M. Takahashi. Tree adjunct grammars. In *Journal of Computer and System Sciences*, 10(1):136–163, 1975.
8. Lamarche, F. Proof nets for intuitionistic linear logic: Essantial nets. Technical report, Imperial College, 1994.
9. Lecomte, A. and C. Retoré. Pomset logic as an alternative categorial grammar. In *Formal Grammar*. Barcelona, 1995.
10. Lecomte, A. and C. Retoré. Words as modules: a lexicalised grammar in the framework of linear logic proof nets. In *International Conference on Mathematical Linguistics*. John Benjamins, Tarragona, Spain, 1996.
11. Lecomte, A. and C. Retoré. Logique de ressources et réseaux syntaxiques. In Genthial, D., editor, *TALN'97*, pages 70–83. Pôle langage naturel, Grenoble, 12-13June 1997.
12. Retoré, C. *Réseaux et séquents ordonnés*. Ph.D. thesis, University of Paris VII, 1993.
13. Retoré, C. Perfect matchings and series-parallel graphs: multiplicatives proof nets as r&b-graphs. In Girard, J.-Y., M. Okada, and A. Scedrov, editors, *Linear Logic 96 Tokyo Meeting*, volume 3 of *Electronic Notes in Theoretical Computer Science*. april 1996.
14. Retoré, C. Pomset logic: a non-commutative extension of classical linear logic. In *TLCA'97*, volume 1210 of *LNCS*, pages 300–318. 1997.
15. Schabes, Y., A. Abeille, and A. K. Joshi. Parsing strategies with 'lexicalized' grammars. In *12th International Conference on Computational Linguistics (COLING'88)*, volume 2, pages 579–583. 1988.