INCREMENTAL PARSING OF LAMBEK CALCULUS USING PROOF-NET INTERFACES

Denis Béchet
IRISA – INRIA Rennes
Campus Universitaire de Beaulieu
F-35042 RENNES
FRANCE
dbechet@irisa.fr

Abstract

The paper describes an incremental parsing algorithm for natural languages that uses *normalized interfaces of modules of proof-nets*. This algorithm produces at each step the different possible partial syntactical analyses of the first words of a sentence. Thus, it can analyze texts *on the fly* leaving partially analyzed sentences.

1 Introduction

Categorial grammars and Lambek calculus are based on the idea that sentences are produced from words using lexical rules. In fact, Lambek associates to each word a set of types in *Lambek calculus* [7]. A correct sentence is a list of words such that the parser can build a proof in this logical system of a list of types that must be associated in the lexicon to the list of words. Usually, a parser for Lambek calculus tries to delimit a sentence in input stream. Then it selects, for each word, one of its types that are found in the lexicon. For each list of formulas, it starts a theorem prover. If a proof is found, the sentence is accepted and the proof is given as result. Otherwise, the parser tries the next possible list of types. If none are left, the words do not form a correct sentence.

In the paper, we want to define a parsing algorithm that processes incrementally "on the fly" on the input stream. In fact, we need to characterize any left part of a syntactical analysis. Thus, we use (non-commutative intuitionistic) *linear logic proof-nets* [4] which are an alternative presentation of Lambek calculus because we need to consider (left) parts of a proof that are named *modules* in proof-net theory [4, 3] (see [8] for their use in computational linguistics). In order to compare different modules, we characterize them using *normalized interfaces*. With interfaces, parsing becomes a simple game of composition of interfaces and *combinators* that can be presented independently of Lambek calculus or linear logic.

In fact, the parser does not need intuitionism: It is also adapted to a classical "word". The difference between a classical and an intuitionistic system only comes from the form of the lexicon that associates some modules to each word of the alphabet. Moreover, it is very suitable when types are complex. For instance, a grammar that describes only tree like structures does not need a module presentation. A simple stack machine would be enough and one would rather prefer to use a CFG or a lexicalized version like HPSG [14]. Systems like LTAG [6] or PPTS [5] are very close to an intuitionistic logical system. More powerful than Lambek calculus with adjunction operation and stretching they usually do not use complex types (functional types of order[1] greater than 2) and the global parse structure is a tree.

Thus, we think that our system is very useful for situation when either the lexicon is not intuitionistic or when it uses types with an order greater than 2. A higher order and classical system is certainly the best word for linear logic modules. For example, a classical system can be used in place of a traditional intuitionistic one to normalize syntactical analyses. In French, adjectives like "petit" (small) must be placed before the noun but some others like "noir" (black)

[1] For Lambek calculus, the order of a formula can be computed by: $o(A) = 0$ if A is atomic, $o(A/B) = o(B \setminus A) = max(o(A), o(B) + 1)$.

for instance must appear after the noun. Now, a noun phrase like "le petit chat noir" (the small black cat) has two analyses: "(le (petit (chat noir)))" or "(le ((petit chat) noir)". Using trees or proof-nets, the two analyses are fundamentally different and lead to two semantically different structures. This is certainly a spurious ambiguity that comes from the fact that a noun has only one conclusion that is used first on the left then on the right or vice-versa. But, with a classical word, a noun can possess two conclusions (a left and a right one): a noun has type $N_l^+ \otimes N_r^+$. An adjective is either $N_l^+ \otimes N_l^-$ or $N_r^- \otimes N_r^+$[2]. With this coding, there is only one analysis of "le petit chat noir" and the syntactical and semantical spurious ambiguity disappears. But, because types become more complex, their order is higher and we really need a parser that can handle graph structures rather than trees: this is a situation where logic is classical, types are complex and parsing structures are graphs (with loops).

The parsing algorithm that we introduce here looks like a stack automata with the traditional shift and reduce operations. However, because the underlying structures behind the stack are modules (in fact interfaces of modules that are essentially graphs), the choice between a reduction or a shift is difficult. The paper proves that this test can be done by a graph rewriting system that normalizes interfaces in a polytime. The parser gets as input a non modified lexicon and the algorithm does not need to compile the grammar that it uses which is certainly an advantage for a dynamic system (for instance a system associated to a learning mechanism that transforms dynamically its grammar).

The paper presents first Lambek calculus, non-commutative intuitionistic multiplicative linear logic modules and proof-nets. The next sections define interfaces and their normalization. Section 6 presents the interface calculus and the parsing algorithm.

2 Lambek Calculus

The reader not familiar with Lambek Calculus may find nice presentations in various articles even the first one written by Lambek[7] or more recently in [9, 11, 1, 10]. The types, or formulas, of Lambek calculus are generated from a set of atomic formulas by three binary connectives "/" (over), "\" (under) and "•" (product). As a logical system, we use a Gentzen-style sequent presentation on Figure 1. A sequent $\Gamma \vdash A$ is composed of a sequence of formulas Γ which is the antecedent configuration and a succedent formula A.

$$\frac{}{A \vdash A}\ \mathbf{Ax} \qquad \frac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\ \mathbf{Cut} \qquad \frac{\Gamma, B \vdash A}{\Gamma \vdash A/B}\ /R \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[B/A, \Gamma] \vdash C}\ /L$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \bullet B}\ \bullet R \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \bullet B, \Delta \vdash C}\ \bullet L \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \setminus B}\ \setminus R \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[\Gamma, A \setminus B] \vdash C}\ \setminus L$$

Figure 1: Gentzen-style presentation of Lambek calculus

3 Lambek proof-nets

Lambek Calculus With Empty Sequence using proof-nets may be defined as the intuitionistic non-commutative multiplicative fragment of linear logic[3]. A *formula* is a couple constituted of a *term* and a *polarity*. *Polarities* are either *input* noted $-$ or *output* noted $+$. *Terms* are inductively constructed from atomic terms $X, X \in \mathcal{V}$ where $\mathcal{V}$ is a set of propositional variables and from the binary connectives $\bullet$, $\setminus$ and $/$. Formulas are noted by capital letters A^+, B^-. *Sequents* are circular lists of formulas. *Intuitionistic sequents* are circular lists of formulas where one and only one formula is output. They are written $A_1, \cdots, A_n \vdash A$ where A^+ is the output formula and $A_1^-, \cdots, A_n^-$ are the other input formulas: The circular list is cut between A^+ and A_1^-. Capital Greek symbols Δ, Γ denote lists of formulas.

[2] $+$ is an output and $-$ an input.

[3] The results presented here are adapted to Lambek Calculus Without Empty Sequence by adding parenthesis to frontiers and a complementary condition to correctness criterion. For interface calculus as it is presented in Section 6, this system corresponds to a restriction on the lexicon and not on the parser.

Definition 1 [Link, Module, Frontier, Proof-structure]

- *A* link *is one of the 8 drawings listed in Figure 2. Formula occurrences above the link are the* premises *and formulas under the link are the* conclusions.

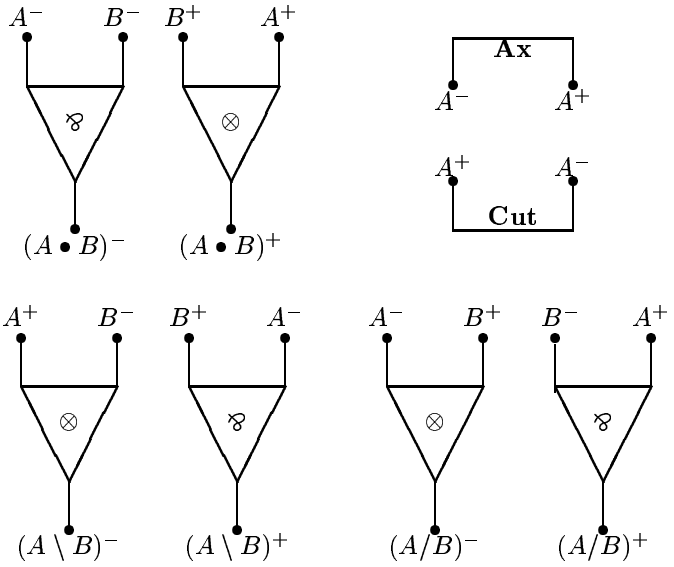

Figure 2: Links

- *A* module *is a planar drawing such that a formula occurrence is premise of at most one link and conclusion of at most one link. The formula occurrences that are not conclusion of a link and the formula occurrences that are not premise of a link must be on the border of the drawing.*
- *The* frontier *of a module is composed of the list (from left to right on a drawing of a module) of formula occurrences that are not conclusion of a link (they are the* conclusions *of the module) and the list (from right to left on a drawing of a module) of formula occurrences that are not premise of a link (they are the* premises *of the module).*
- *A* proof-structure *is a module that has no premise.*
- *An* intuitionistic proof-structure *is a proof-structure such that the conclusions form an intuitionistic sequent (there is exactly one output formula).*

Figure 3 shows a module composed of five $\otimes$-links, one $\parr$-link and two axiom links. Six formula occurrences are premise and conclusion of some links, four are only conclusion, six are only premise and one NP^- is neither premise nor conclusion. Thus, the conclusions of the module are, from left to right, NP^-, $(NP \setminus S)/VP^-$, $(S/VP) \setminus (S/VP)^-$, NP^- and $(NP \setminus VP)^-$. The premises are, from right to left, VP^-, NP^+, NP^-, VP^+, S^-, S^+ and S^-. The formula occurrence NP^- that is not connected to a link appeared both as a conclusion and as a premise of the module.

Definition 2 [Proof-net] *A* proof-net *is an intuitionistic proof-structure that corresponds to a sequential proof in Lambek calculus.*

Remark: In fact, not every intuitionistic proof-structure is a proof-net. But, there exist criteria that tell us if a proof-structure is or is not a proof-net. In this article, we use the Danos-Regnier criterion [3].

Definition 3 [Proof-structure/module switching] *A switching of a proof-structure (or a module) is a selection for every $\parr$-link between* left *or* right *position. This switching induces a planar graph by replacing each link by the edges shown on Figure 4 (right and left positions are completely independent of the kind of $\parr$-link (input, left output or right output $\parr$-links)).*

Definition 4 [Correct proof-structure] *A proof-structure is correct (it verifies the correctness criterion) if and only if for every switching, the graph is acyclic and connected.*

Theorem 5 *Correct intuitionistic proof-structures without cut are proof-nets (they correspond to a sequential proof).*

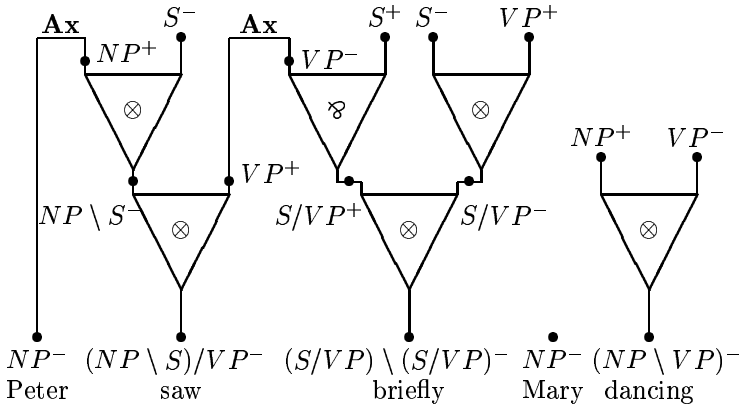

Figure 3: A module (part of the proof-net of Figure 5)

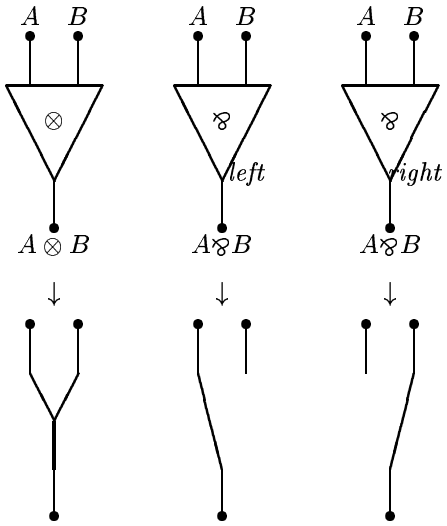

Figure 4: Switching links

This is the usual sequentialization theorem of proof-nets. In [4, 2, 15], the theorem is given in the case of classical linear logic. Here, we must take into account the fact that the proofs are intuitionistic and non-commutative [16].

Figure 5 shows a proof-net corresponding to the sentence "Peter saw briefly Mary dancing". The category of each word is supposed to be given by a lexicon (several formulas are allowed for a word).

4 Module Interfaces

We can split a proof-structure in two modules by splitting the circular list of conclusions in two. Axiom links that connect the two parts are cut: the link is left in the right module, thus this module has new conclusions (one for each cut axiom link), the left module has new premises (one for each cut axiom link). Figure 6 shows the splitting of the proof-net of Figure 5 corresponding to "Peter saw" and "briefly Mary dancing $\vdash S$". In this example, we split the conclusions between succedent S and the first formula NP (remember that the list is circular) and between the second formula $(NP \setminus S)/VP$ and the third formula $(S/VP) \setminus (S/VP)$. The left module has two conclusions NP^- and $(NP \setminus S)/VP^-$ and two premises VP^+ and S^-. The right module is a proof-structure (but neither a proof-net nor an intuitionistic proof-structure) because it does not have a premise. Its conclusions are S^-, VP^+, $(S/VP) \setminus (S/VP)^-$, NP^-, $NP \setminus VP^-$ and S^+.

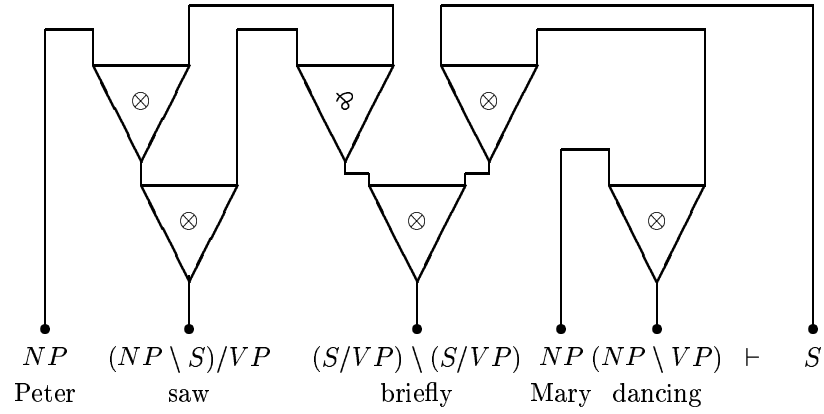

Figure 5: A proof-net corresponding to "Peter saw briefly Mary dancing"

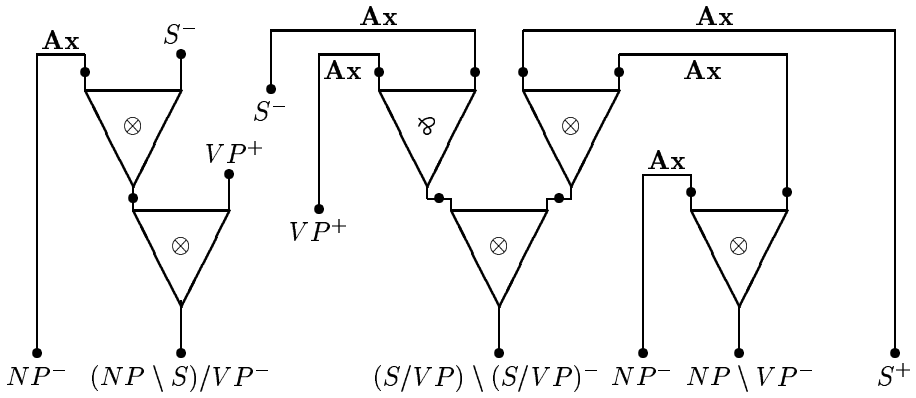

Figure 6: A splitting of a proof-net in two modules

Definition 6 [Orthogonal modules] *A module M with conclusions $A_1, \ldots, A_k$ and premises $B_1, \ldots, B_n$ is* orthogonal *to a proof-structure N with conclusions $B_1, \ldots, B_n, C_1, \ldots, C_m$, (notation $M \perp N$) if and only if the proof-structure obtained by linking the n premises of M to the n first conclusions of N is a proof-net.*

For instance, because the left module and the right proof-structure of Figure 6 are defined by splitting the proof-net of Figure 5, they are orthogonal.

Definition 7 [Equivalent modules] *Two modules M_1 and M_2 with the same premises are equivalent (notation $M_1 \equiv M_2$) if and only if for every proof structure N, $M_1 \perp N \iff M_2 \perp N$*

For instance, the modules which are shown on Figure 7 corresponding to the two first words and the three first words of "Peter saw briefly Mary dancing" on Figure 5 are equivalent. From a linguistic point of view, two equivalent modules correspond to two left parts of sentences that are syntactically equivalent. This equivalence is particularly interesting because it does not need to follow the constituent boundary as the previous example shows.

However, this equivalence is not easy to check directly on modules. In the context of commutative linear logic, [2] computes the set of partitions of the premises of each module induced by the switching positions of the $\parr$-links of the modules. If a module has k $\parr$-links, this is done in 2^k steps even if the number of premises is bound. Thus, we need a characterization of modules that enables us to test this equivalence in polynomial time. For that, we introduce *interfaces* of modules and a normalizing (in polynomial time) and confluent rewriting system for each class of equivalent modules.

Definition 8 [Interface] *An interface is a triple (B, N, m) where:*

- *B, the* border *of the interface, is a list of formula occurrences from the set of atomic terms $\mathcal{V}$.*

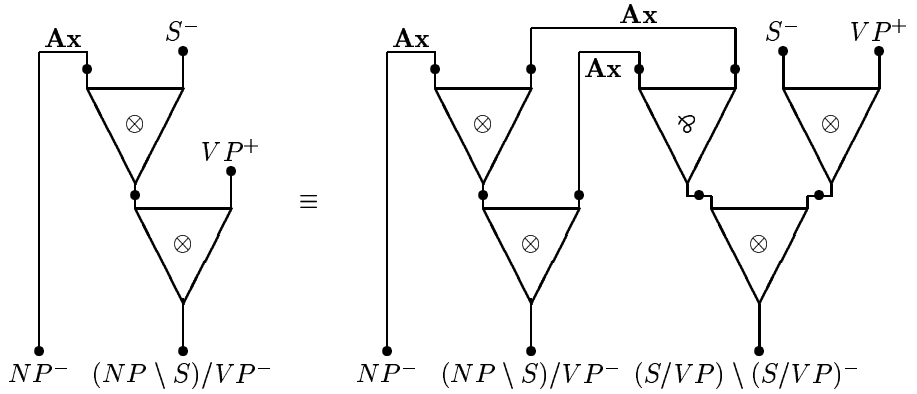

Figure 7: Two equivalent modules

- $N = (V, L)$, *the* net *of the interface, is a (non-planar) drawing made from vertices* V *and* k*-ary* $\parr$*-links* L *shown on Figure 8. Each link is connected to a particular vertex called the* conclusion *and to a multiset of vertices that are its* premises. *The arity is the number of premises and must be greater or equal to 1 and must not be necessarily equal for all the* $\parr$*-links of a module.*

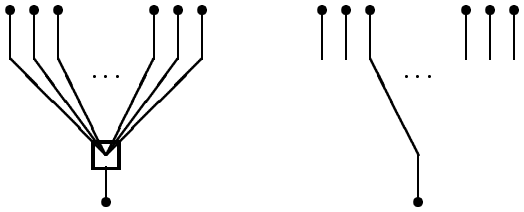

Figure 8: k-ary $\parr$-link and one of its switching positions

- m, *the* mapping *of the interface, is a function* $m : B \to V$ *that maps formula occurrences of the border to (not necessarily different) vertices of the net.*

Definition 9 [Natural (premise) interface]

- *The* natural interface *of a module (or proof-structure)* M *towards a border* B *that must be (contiguous) formula occurrences of the frontier of the module is defined by transforming each link of the module by the corresponding local net defined on Figure 9. The mapping of the interface associates to each formula occurrence of* B *the vertex of the net that corresponds to the translation of the formula occurrence of* M*. This interface is noted* $\Lambda_B(M)$*.*
- *The* natural premise interface *of a module* M *is the natural interface of the module towards the list of premises of* M*. It is noted* $\Lambda(M)$*.*

Figure 10 gives the natural premise interface of the left module of Figure 6. There are six vertices, three unary $\parr$-links, one binary $\parr$-link. The border has two formula occurrences noted VP_1^+ and S_2^- that are mapped to two different vertices.

An interface looks like a module except that it is not necessarily planar and vertices can be connected to more than two links. However, correctness criteria can be applied to it.

Definition 10 [Interface switching, Correct interface]

- *A switching of an interface is a selection for every* k*-ary* $\parr$*-link from one of its* k *positions. This switching induces a graph by replacing each link by an edge following its switching position (see Figure 8).*
- *An interface is* correct *(it verifies the correctness criterion for interfaces) if and only if for every switching, the graph is acyclic and each component has at least a vertex that is mapped to a formula occurrence of the border.*

Theorem 11 *Testing that an interface is correct is polytime.*

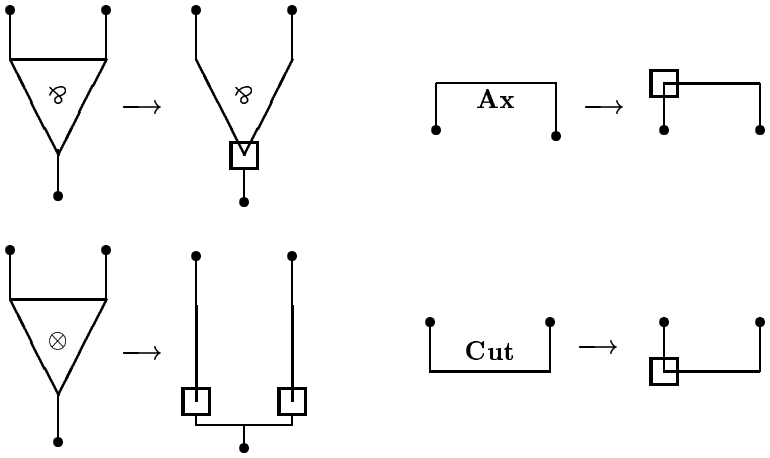

Figure 9: Translation from links (module) to local nets (interface)

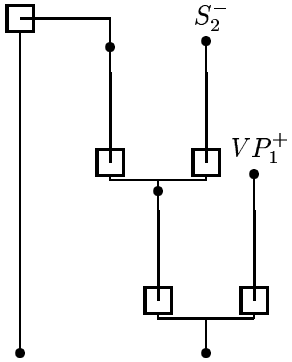

Figure 10: The natural interface of the left module of Figure 6

Proof:
This result is not very original because it is just an adaptation to interfaces of results on correctness criteria of proof-nets. One can use for instance blue-red graphs like in [15]. □

Theorem 12 *The natural premise interface of a module that is orthogonal to some proof-structure is correct.*

Proof:
A switching of the natural interface corresponds exactly to a switching of the module. Because the module M is orthogonal to a proof-structure N, M and N connected together form a proof-net P. A switching of M and a switching of N define a switching of P which is correct. The graph for P for this switching is acyclic and connected. So, the switching of the natural interface induces a graph that is acyclic and where each vertex must be connected to a vertex corresponding to the border (which is connected to N in P). □

Definition 13 [Orthogonal interfaces]

- *Two correct interfaces* $I_1 = (B_1, (V_1, L_1), m_1)$ *and* $I_2 = (B_2, (V_2, L_2), m_2)$ *with borders* $A_1, \ldots, A_n$ *and* $A_n, \ldots, A_1$ *are* orthogonal *(notation* $I_1 \perp I_2$*) if and only if for every switching of the nets of* I_1 *and* I_2*, the graph obtained from the union of the two induced graphs by connecting the n edges* $m_1(A_1)$ *to* $m_2(A_1)$*,* $\ldots$*,* $m_1(A_n)$ *to* $m_2(A_n)$ *is acyclic and connected.*
- *A correct interface* I *with a border* $A_1, \ldots, A_n$ *is* orthogonal *to a proof-structure* N *with conclusions* $A_1, \ldots, A_n, B_1, \ldots, B_m$*, (notation* $I \perp N$*) if and only if* $I \perp \Lambda_{A_n,\ldots,A_1}(N)$.

For instance, because the left module and the right proof-structure of Figure 6 are defined by splitting the proof-net of Figure 5, the interface on Figure 10 and the right proof-structure of Figure 6 are orthogonal.

Definition 14 [Equivalent interfaces] *Two correct interfaces I_1 and I_2 with the same border are equivalent (notation $I_1 \equiv I_2$) if and only if for every proof structure N, $I_1 \perp N \iff I_2 \perp N$*

Theorem 15 *Two modules M_1 and M_2 with the same premises are equivalent if and only if $\Lambda(M_1)$ and $\Lambda(M_2)$ are equivalent.*

Proof:
The proof is straightforward. □

5 Interface normalization

For a particular border B, we want to know if two modules are equivalent. Theorem 15 allows us to use interfaces rather than modules. This section presents the main result of the paper by introducing interface transformations that are compatible with the equivalence of modules. Moreover, the orientation of the transformations leads to a normalizing mechanism that gives a unique representative of each class of equivalence. Thus, two correct interfaces (then two modules) are equivalent if and only if their normalized interfaces are equal.

Definition 16 [Flat interface transformations, Interface normalization transformations]

- *Flat interface transformations are transformations of correct interfaces. Figure 11 shows the four rules: ⅋-erasing, ⅋-transitivity, Edge merging and Vertices merging.*

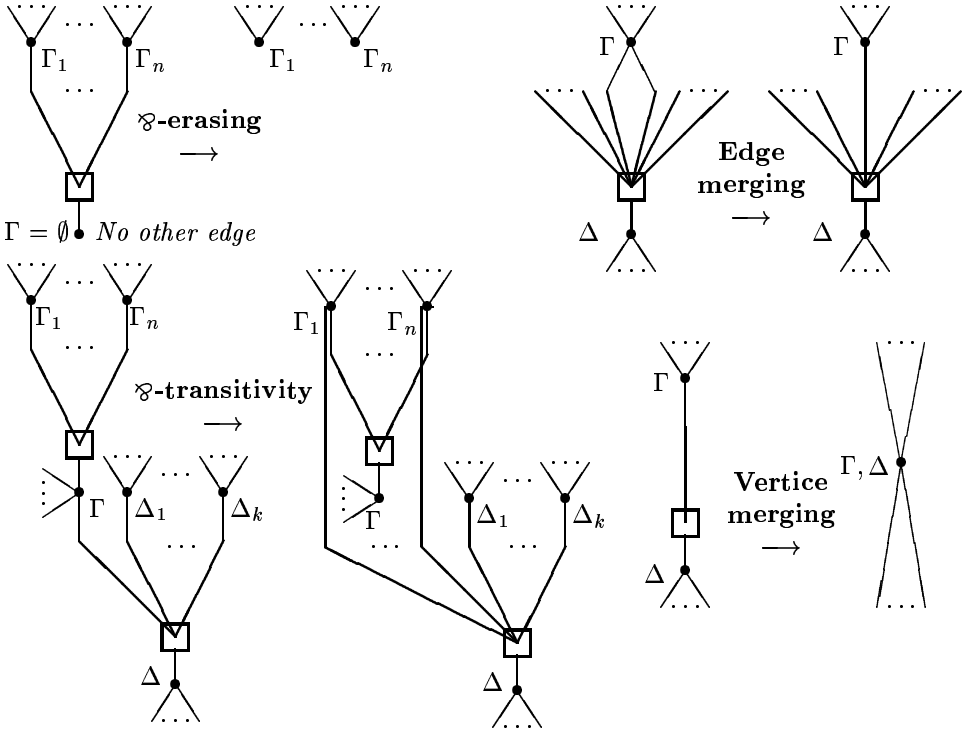

Figure 11: Flat Interface Transformations

- *Interface normalization transformations are transformations of correct interfaces. Figure 12 shows the three rules: Propagation, Merging and Completion.*

The transformations need some explanations. Capital Greek letters like Γ or Δ represent formula occurrences from the border that are mapped to a vertex. For all rules except ⅋-erasing, this mapping is free. In ⅋-erasing, the conclusion of the n-ary ⅋-link must not be mapped to any formula occurrence of the border. Moreover, this vertex must not be premise or conclusion of another ⅋-link. In Merging, a n-ary ⅋-link disappears and two vertices are merged. In fact, this ⅋-link must have as premises all the premises of all the ⅋-links whose conclusion is the single

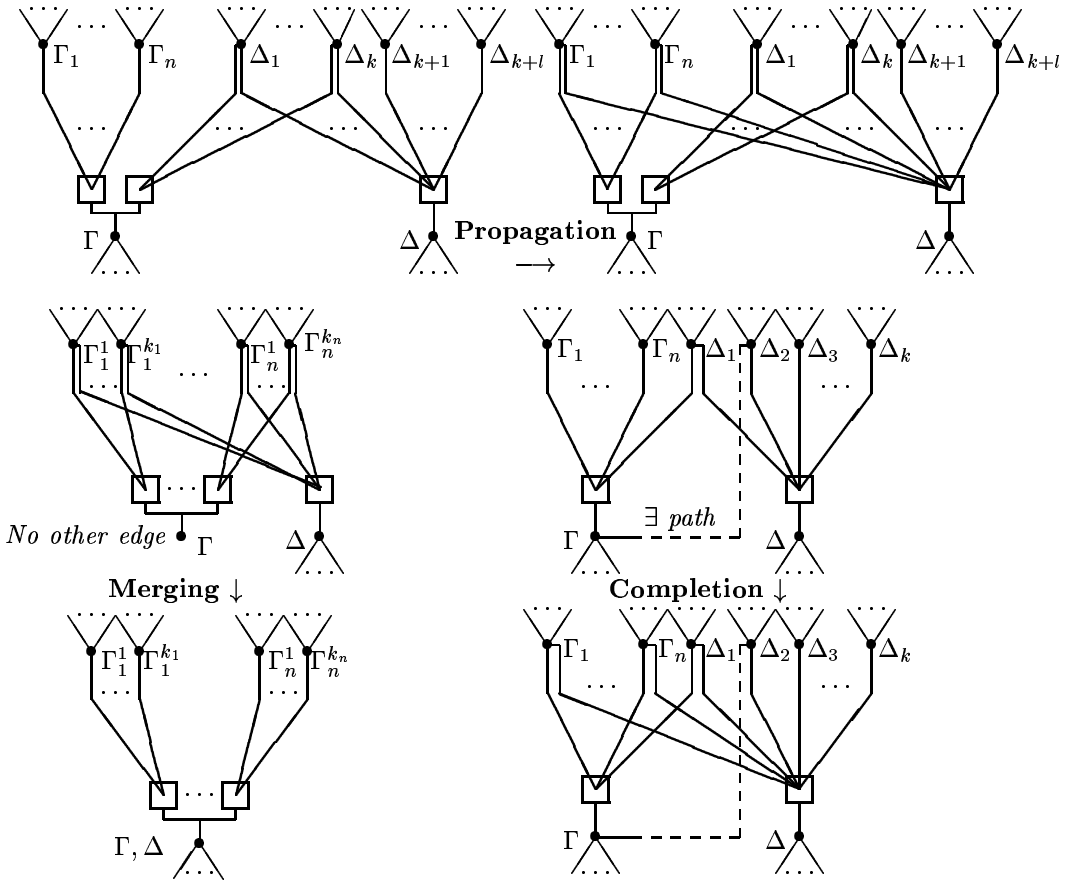

Figure 12: Interface Normalization Transformations

vertex mapped to Γ. This is almost equivalent to the condition shown on Figure 12 for this vertex. For Completion, we must verify that there exists a path between the vertex mapped to Γ and the vertex mapped to Δ_2. **This path must be simple and must not follow two different edges of the same $\parr$-link**. Moreover, it must not take the $\parr$-link with conclusion the vertex mapped to Γ. Finally, for all rules, a new edge of a $\parr$-link is effectively added only if none already exists for this $\parr$-link with the same premise (Edge merging in implicitly used).

Theorem 17 *On the net of a correct interface, flat interface transformations and interface normalization transformations preserve correctness and are compatible with interface equivalence (they do not change the class of the interface).*

Proof:
The different proofs use a common technique that proves that a net and its transformed net have the same orthogonal that is the same set of orthogonal proof-structures. In fact, we use a compositional lemma that says that on correct interfaces, a local transformation that transforms a sub-net by an equivalent one does not change the class of the global interface. A second step need the key transformation shown on Figure 13 that preserves correctness and the equivalence of interfaces. This transformation can be seen as a small brick that can be used extensively for instance for proving correctness and compatibility of $\parr$-transitivity, Propagation and Merging. □

Theorem 18 *In a class of equivalent interfaces that has at least one orthogonal proof-structure, there exists only one interface that can not be reduced by transformations of Theorem 17*

Proof:
This proof is relatively long and can not be presented in detail in the paper due to lack of space. Firstly, an interface that is in a class that has at least one orthogonal proof-structure must be correct. Secondly, a correct interface that is reduced towards flat interface transformations must

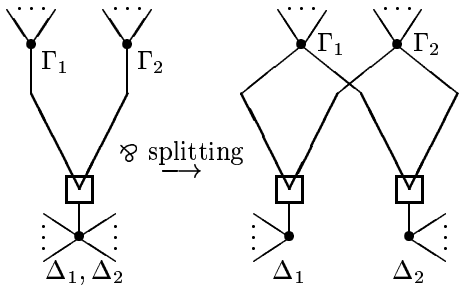

Figure 13: Key transformation

be **flat**. A flat interface is an interface where a vertex can not be, at the same time, premise and conclusion of one or several $\parr$-links. Thus, a vertex can be alone but in this case it must be mapped to one or several formula occurrences of the border. It can be premise of one or several $\parr$-links and must be mapped to one or several formula occurrences of the border. It can be conclusion of a unique $\parr$-link but in this case it must be mapped to one or several formula occurrence of the border. Finally, it can be conclusion of several $\parr$-links and can be mapped to one or several formula occurrence of the border. This property is important because it gives a bound on the number of reduced correct interfaces towards the number of formula occurrences of the border. One can prove that there are at most n^2 different correct interfaces with a fixed border of n formula occurrences. The third part of the proof is a recurrence on the size of interfaces that shows that if we have two correct flat interfaces that are equivalent and are reduced towards interface normalization transformations, these interfaces are equal. □

Theorem 19 *The rewriting system on correct interfaces that performs firstly flat interface transformations and secondly interface normalization transformations terminates in polynomial time towards the size of the interface and is confluent.*

Proof:
In fact, interface normalization transformations transform flat interfaces into flat interfaces. Thus, the result of the rewriting system is reduced (towards flat interface transformations).

For each transformation, the number of vertices and $\parr$-links of the net can not increase. For flat interface transformations, the unique problem can arrive if one can use $\parr$-transitivity in a loop. But this is not possible because the interface is correct and so does not have a cycle following from conclusion to premise a circular list of $\parr$-links. This also prove that the number of reductions using $\parr$-transitivity transformation is bound by the square of the number of $\parr$-links. The number of edges of a $\parr$-link is also bound by this number plus the initial number of edges. So, this first step is polytime. The second step that uses interface normalization transformations terminates because the number of reduction using Merging is bound by the number of $\parr$-links. The two other rules increase the number of edges. Because at this level, the interface is flat, the number of edges is bound by the square of the number of vertices (two vertices can not be linked by more than one edge in flat interfaces) and reduction is polynomialy bound.

With the previous theorem, we know that there exists only one reduced correct interface for each class. So, the system is confluent. □

6 Interface calculus and parsing

In fact, we can use directly interfaces rather than module in the parsing algorithm. This is done by introducing a left-to-right composition of interfaces. What we have to do is to compute for each word the different interfaces corresponding to the formulas associated to the word in the lexicon. Because it is possible to have axiom links on a proof-net that connect two dual atomic types of the same formula (however for Lambek calculus without empty sequence this is very rare), several different interfaces correspond to a formula. Now, for an interface with a border $A_1, \ldots, A_n$, there exists $n+1$ different compositions. For $0 \leq i \leq n$, the i-th combinator gives $A_n, \ldots, A_{i+1}$ to the left and $A_i, \ldots, A_1$ to the right. Thus, for our parsing algorithm, we suppose that the lexicon associates to each word a set of combinators. Each combinator is an interface where the border is

divided in two lists: a *left border* that must match the left part of the sentence and a *right border* that must match the right part of the sentence.

Definition 20 [Lexicon, Combinator] *A* lexicon $\mathcal{L} \subset \mathcal{W} \times \mathcal{C}$ *is a finite binary relation between words* $\mathcal{W}$ *and combinators* $\mathcal{C}$ *that are normalized correct interfaces where the border is divided in two sub-lists* the left and right borders.

Definition 21 [Interface/combinator composition] *The composition of a normalized correct interface* I *with border* $A_1, \ldots, A_n$ *and a combinators* C *with left border* $B_1, \ldots, B_k$ *and right border* $C_1, \ldots, C_l$ *is possible when* $k \leq n$*, when* $\forall i, 1 \leq i \leq k, A_i = B_{k-i+1}^{\perp}$*, when* $n - k + l > 0$ *and when the interface obtained from* I *and* C *by linking (with an axiom link) in the reverse order the first* k *formula occurrences of* I *with the* k *formula occurrences of* C *corresponding to* $B_1, \ldots, B_k$ *(the formulas are erased from the border) is a correct interface. The result of the composition which is noted* $I \circ C$ *is the normalized correct interface corresponding to the previous correct interface. Its border is* $C_1, \ldots, C_l, A_{k+1}, \ldots, A_n$*.*

Definition 22 [Parser] *The* parsing algorithm *parses sentences from left to right adding a word at each step. It computes the different possible normalized correct interfaces that correspond to the already parsed left part of the sentence.* $\mathcal{L}$ *is the lexicon of the parser. It starts with the list of interfaces corresponding to the combinators of the first word of the stream that have an empty left border.*

If $I_1, \ldots, I_n$ *are the possible correct interfaces at last step:*

- *If no more word is on the input, the parser searches in* $I_1, \ldots, I_n$ *the interfaces that have a border with only* **one** *formula. These atomic types are the possible types of the parsed sentence: phrase, noun phrase, question...*
- *Otherwise, the parser gets the next word* w *and searches the combinators associated to it in* $\mathcal{L}$*. If* $C_1, \ldots, C_k$ *are the possible combinators, it tries the* $n \times k$ *possible compositions, computes the possible resulting normalized correct interfaces* $I_i \circ C_j$ *and continues with the next word.*

Figure 14 shows a partial analysis of the first three words of "Peter saw briefly Mary dancing". Small rectangles represent the combinators corresponding to the three words. Wide rectangles are the successive computed interfaces (their net is not shown here). In this simple case, we only put one combinator for each word. But, of course, this is not the case for a complex lexicon.

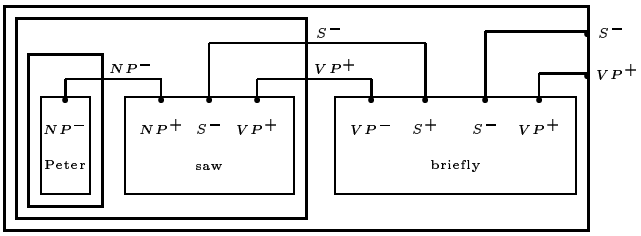

Figure 14: Successive computed interfaces and combinators for "Peter saw briefly"

7 Discussion and conclusion

The parsing algorithm that is introduced in the last section is not polytime because at each step, the number of possible correct interfaces is not bound. This result is not a surprise because the complexity of Lambek calculus is still open. But, this parser can restrict the language that it recognizes with certain linguistically founded arguments. For instance, in [12, 13], proof-nets that correspond to the syntactical analysis of an acceptable sentence have a limited depth. Intuitively, this limit says that we can not understand very complex sentences that are "not flat". For the parser, it means that the number of formulas of the border of interfaces must be bound by a parameter and, of course, our algorithm is polytime but it uses a much more compact representation than usual interface modeling that is based on sets of partitions of the formulas of the interface and a bi-orthogonal calculus.

Moreover, this parser can give partial results and can be adapted to find on a stream of words some constituents like noun phrases. This is particularly useful when the input is not

completely correct. Another useful result of the parser is that it can produce in a compact way all the possible syntactical analyses of the same sentence. Thus, a semantical module may use this compact representation to find the proper one: Syntax is of course not enough to eliminate all ambiguities that are very often in natural languages.

References

[1] Denis Bechet and Philippe de Groote. Constructing different phonological bracketings from a proof net. In *Proceedings of the First Conference on Logical Aspects of Computational Linguistics, Nancy, France, Septembre 1996 (Lecture Note in Artificial Intelligence, vol. 1328)*, pages 119–133, 1997.

[2] Vincent Danos. *La Logique Linéaire Appliquée à l'étude de Divers Processus de Normalisation (Principalement du λ-Calcul)*. PhD thesis, University of Paris VII, June 1990.

[3] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[4] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[5] Aravind Joshi and Seth Kulick. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 20:637–667, 1997.

[6] Aravind Joshi and Seth Kulick. Partial proof trees, resource sensitive logics and syntactic constraints. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *LNCS/LNAI*, pages 21–42. Springer-Verlag, 1997.

[7] Joachim Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170, 1958.

[8] Alain Lecomte and Christian Retoré. Words as modules: a lexicalised grammar in the framework of linear logic proof nets. In Carlos Martin-Vide, editor, *Mathematical and Computational Analysis of Natural Language — selected papers from ICML'96*, volume 45 of *Studies in Functional and Structural Linguistics*, pages 129–144. John Benjamins publishing company, 1998.

[9] Michael Moortgat. *Categorial Investigations: Logical & Linguistic Aspects of the Lambek Calculus (Groningen-Amsterdam Studies in Semantics)*. Foris Publications, 1988.

[10] Michael Moortgat. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam, 1997.

[11] Glyn Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, Dordrecht, 1994.

[12] Glyn Morrill. Incremental processing and acceptability. *Computational Linguistics*, 26(3):319–338, 2000.

[13] Guy Perrier. Interaction grammars. In *Proceedings of the 5th International Conference on Computational Linguistics (CoLing 2000)*, 2000.

[14] Carl J. Pollard and Ivan A. Sag. Hpsg: A new theoretical synopsis. In Byung-Soo Park, editor, *Linguistic Studies on Natural Language*, volume 1 of *Kyunghee Language Institute Monograph*. Hanshin, Seoul, Korea, 1992.

[15] Christian Retoré. *Réseaux and Séquents Ordonnés*. PhD thesis, University of Paris VII, 1993.

[16] Dirk Roorda. *Resource Logics: proof-theoretical investigations*. PhD thesis, University of Amsterdam, 1991.