On the computation of joins for non associative Lambek categorial grammars

Annie Foret
Email : foret@irisa.fr
http://www.irisa.fr/prive/foret

IRISA and University of Rennes1
Campus de Beaulieu
35042 Rennes Cedex, France.

Abstract. This paper deals with an application of unification and rewriting to Lambek categorial grammars used in the field of computational linguistics. Unification plays a crucial role in the acquisition of categorial grammar acquisition, as in [Kan98] ; a modified unification has been proposed [For01a] in this context for Lambek categorial grammars, to give an account of their logical part. This modified unification ($\|=$ -unification) relies both on deduction (Lambek derivation) and on substitution ; it is strongly related to the conjoinability relation [Lam58,Pen93] that is characterized by a free group equivalence and by a quasi-group one in the non-associative version. In view of grammatical inference, we also need to compute joins of the conjoinability relation, when they exists. This paper deals with this issue for the non-associative version of Lambek grammars, and provides an algorithm based on quasi-group rewriting.

1 Introduction

Categorial grammars, introduced in [BH53] and extended to Lambek grammars in [Lam58], have been studied in the field of natural language processing. Since they are completely lexicalized, they are well adapted to learning perspectives and an actual way of research is to determine the sub-classes of such grammars that remain learnable in the sense of Gold [Gol67].

Recent works from [Kan98] and [Nic99] following [BP90] have answered the problem for different sub-classes of classical categorial grammars (we recall that the whole class of classical categorial grammars is equivalent to context free grammars; the same holds for the class of Lambek grammars [Pen93] that is

substitution instead of standard substitution and standard unification. Such an operation has been proposed and characterized in [For01a,For01b].

This modified unification is strongly related to the conjoinability relation [Lam58,Pen93] that is characterized by a free group equivalence and by a quasi-group one in the non-associative version.

In order to apply this operation, we also need to compute joins when they exists. This paper deals with this issue for the non-associative version of Lambek grammars, and provides an algorithm based on quasi-group rewriting.

The paper is organized as follows.

Section 2 gives background definition on categorial grammars. Section 3 addresses unification in the context of categorial grammar acquisition. Section 4 explains the conjoinability relation and its properties in L. Section 5 gives details on the conjoinability relation in NL, (the non-associative version) including its connection with quasi-groups. Section 6 gives some recalls from [For01a,For01b] about $\|=$- unification in L and NL and their connections with conjoinability. Section 7 describes our new contribution related to conjoinability in NL : an algorithm based on quasi-group rewriting for computing joins. Section 8 concludes.

2 Categorial Grammars

The reader not familiar with Lambek Calculus and its non-associative version will find nice presentation in the first one written by Lambek [Lam58,Lam61] or more recently in [Kan88,AT95,dG99].

The *types* Tp, or formulas, are generated from a set of *primitive types* Pr, or atomic formulas by three binary connectives " / " (over), " \ " (under) and "$\bullet$" (product): $Tp ::= Pr \mid Tp \setminus Tp \mid Tp \,/\, Tp \mid Tp \bullet Tp$. As a logical system, we use a Gentzen-style sequent presentation. A sequent $\Gamma \vdash A$ is composed of a sequence of formulas Γ which is the antecedent configuration and a succedent formula A.

Let Σ be a fixed alphabet. A *categorial grammar* over Σ is a finite relation G between Σ and Tp. If $< c, A > \in G$, we say that G *assigns* A to c, and we write $G \ : c \mapsto A$.

Lambek Derivation $\vdash_L$. The relation $\vdash_L$ is the smallest relation $\vdash$ between Tp^+ and Tp, such that for all $\Gamma, \Gamma' \in Tp^+, \Delta, \Delta' \in Tp^*$ and for all $A, B \in Tp$: see figure 1.

Non Associative Lambek Derivation $\vdash_{NL}$ In the Gentzen presentation,

$$\frac{\Gamma, A, \Gamma' \vdash C \quad \Delta \vdash A}{\Gamma, \Delta, \Gamma' \vdash C}\,\mathbf{Cut} \qquad A \vdash A$$

$$\frac{\Gamma \vdash A \quad \Delta, B, \Delta' \vdash C}{\Delta, B \,/\, A, \Gamma, \Delta' \vdash C}\,/L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash B \,/\, A}\,/R$$

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

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

Fig. 1. L System

$$\frac{\Gamma[A] \vdash C \quad \Delta \vdash A}{\Gamma[\Delta] \vdash C}\,\mathbf{Cut} \qquad A \vdash A$$

$$\frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(B \,/\, A, \Gamma)] \vdash C}\,/L \qquad \frac{(\Gamma, A) \vdash B}{\Gamma \vdash B \,/\, A}\,/R$$

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

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

Fig. 2. NL System

$NL_\emptyset$ for the Non associative Lambek calculus with empty antecedents (left part of the sequent). We also refer to [Bus97,Moo97] for more details on NL.

Note [Cut Elimination]. We recall that the cut rule is admissible in $\vdash_L$ and $\vdash_{NL}$: every derivable sequent has a cut -free derivation.

Language. Let G be a categorial grammar over Σ. G *generates* a string $c_1 \dots c_n \in \Sigma^+$ iff there are types $A_1, \dots, A_n \in Tp$ such that : $G : c_i \mapsto A_i$ $(1 \leq i \leq n)$ and $A_1, \dots, A_n \vdash_L S$. The *language of* G, written $\mathcal{L}_L(G)$ is the set

Rigid and *k*-valued Grammars. Categorial grammars that assign at most k types to each symbol in the alphabet are called *k-valued grammars*; 1-valued grammars are also called *rigid* grammars.

Example 1. Let $\Sigma_1 = \{John, Mary, likes\}$ and let $Pr = \{S, N\}$ for sentences and nouns respectively. Let $G_1 = \{John \mapsto N,\ Mary \mapsto N,\ likes \mapsto N \setminus (S\ /\ N)\}$. We get $(John\ likes\ Mary) \in \mathcal{L}_{NL}(G_1)$ since $((N,\ N \setminus (S\ /\ N)),\ N)\ \vdash_{NL} S$. G_1 is a rigid (or 1-valued) grammar.

Note. Classical (or AB) categorial grammars, correspond to derivation rules ($\vdash_{AB}$) that are also valid in Lambek grammars; they are not detailed here.

2.1 Unification and grammatical inference

Substitution and unification play an important role in grammatical inference. In this section we recall some related definitions.

Substitutions. A substitution σ is a function from variables in Var to types in Tp which is extended from types to types by :
$\sigma(p) = p$ for $p \in Pr_c$ (with $\sigma(S) = S$)
$\sigma(A \setminus B) = \sigma(A) \setminus \sigma(B)$ $\sigma(B\ /\ A) = \sigma(B)\ /\ \sigma(A)$
$\sigma(B \otimes A) = \sigma(B) \otimes \sigma(A)$ [1]
We extend this definition to G-terms and sequences by : $\sigma(\Gamma, \Delta) = (\sigma(\Gamma), \sigma(\Delta))$ and respectively by : $\sigma(A_1, \ldots, A_n) = \sigma(A_1), \ldots, \sigma(A_n)$

Note. The following principle (hereafter called the *replacement principle*) holds for $\vdash_{AB}$, $\vdash_L$ and $\vdash_{NL}$: if $\Gamma \vdash B$ then $\sigma(\Gamma) \vdash \sigma(B)$.

Substitutions extended to grammars. Given a substitution σ, and a grammar G, $\sigma(G)$ denotes the grammar obtained by applying σ in the type assignments, that is : $\sigma(G) = \{< c, \sigma(A) >; < c, A > \in G\}$.

Preorders based on substitutions. Substitution allows to define several preorders as follows :

$\preceq$ on types defined by $A \preceq B$ iff $\exists \sigma\ : \sigma(A) = B$ (B is said an *instance* of A, or also A is said more general than B) ;

$\preceq$ on substitutions defined by $\sigma \preceq \theta$ iff $\forall A \in Tp : \sigma(A) \preceq \theta(A)$; this yields $\sigma \preceq \theta$ iff $\exists \rho : \rho\sigma = \theta$ (where we write $\rho\sigma$ the composition of substitutions σ with ρ as : $\forall x : \rho\sigma(x) = \rho(\sigma(x))$) ; σ is said to be *more general* than θ;

Note. It is easy to see (using the replacement principle) that if $G_1 \sqsubseteq G_2$ then $L(G_1) \subseteq L(G_2)$ (also for L and NL). This fact is useful in the learning approach.

Unification. A set of types $\mathcal{A}$ is said *unifiable* whenever there exists a substitution σ such that $\sigma(A) = \sigma(B)$ for all $A, B \in \mathcal{A}$; σ is then said a unifier of the types $\mathcal{A}$. Let $U(\mathcal{A})$) denote the set of unifiers of $\mathcal{A}$. A *principal unifier* of $\mathcal{A}$ is a unifier σ, such that $\forall \theta \in\ U(\mathcal{A})\ : \sigma \preceq \theta$.

The usual unification problem is as follows : given two types to indicate whether these types are unifiable and if so to give a principal unifier. One important property is that when two types are unifiable, they admit a principal unifier that is unique modulo variable renaming.

Unification extended to rigid categorial AB-grammars and least upper bounds. A substitution σ is said to unify a family $\mathcal{F}$ of sets of types, if σ unifies each set in the family. A *principal unifier* σ of $\mathcal{F}$ is a unifier of $\mathcal{F}$ such that forall unifier θ of $\mathcal{F} : \sigma \preceq \theta$.

Let us fix mgu a function that computes a principal unifier (undefined if there is none) for each set of types (or family of sets of types). Let G_1 and G_2 be rigid grammars with no common variables. We consider the family $\mathcal{F}$ of the sets $\mathcal{A}_c$ for each c in the alphabet of G_1 or $G_2 : \mathcal{A}_c = \{A; < c, A > \in G_1 \ or\ < c, A > \in G_2\}$. We let $G_1 \sqcup G_2 = mgu(\mathcal{F})(G_1 \cup G_2)$. This operation computes the least upper bound of rigid grammars (with respect to $\sqsubseteq$); it has properties of particular interest for the convergence of the learning algorithm.

3 Conjoinability

In this section we first recall useful definitions and properties on joins.

3.1 Conjoinability

We now define the conjoinability relation introduced in [Lam58]. This equivalence can be viewed as the special case of $\|=$ -unifiability (defined later) without variables.

Equivalence on types.
1. The *join-equivalence*, written $\sim$, is defined by :
$t \sim t'$ iff $\exists t_1, ..., t_n : \forall i < n\ (t_i \vdash t_{i+1}$ or $t_{i+1} \vdash t_i)$ with $t = t_1, t' = t_n$
2. We also define $\sim^1$ by : $t \sim^1 t'$ iff $t \vdash t'$ or $t' \vdash t$
The equivalence $\sim$ is thus the transitive closure of $\sim^1$.
3. Types $t_1, t_2, ...t_n$ are said *conjoinable* whenever there exists t such that $t_i \vdash t$ (for all $i \leq n$). In this case t is called a *join* for $t_1, t_2, ...t_n$.
We now recall some properties of conjoinability in the $\vdash_L$ case. The following

Proof. We follow the version of [Pen93]
- if $t_i \vdash t$, for i=1,2, we verify that $t' \vdash t_i$, for i=1,2, where :

$$t' = (t_1 \,/\, t).t.(t \setminus t_2)$$

- if $t' \vdash t_i$, for i=1,2, we verify that $t_i \vdash t$, for i=1,2, where :

$$t = (t' \,/\, t_1) \setminus t' \,/\, (t_2 \setminus t')$$

Proposition 2 *Let t_1 and t_2 be two types. The assertion $t_1 \sim t_2$ in L is also equivalent to (i) and (ii) of the diamond property above.*

Free group interpretation.

Let FG denote the free group with generators Pr, operation . and with neutral element Λ.

We associate with each formula A an element in FG written $[[A]]$ as follows :
$[[p]] = p$ for p atomic
$[[A_1 \setminus A_2]] = [[A_1]]^{-1}.[[A_2]]$
$[[A_1 \,/\, A_2]] = [[A_1]].[[A_2]]^{-1}$
$[[A_1 \otimes A_2]] = [[A_1]].[[A_2]]$
We extend the notation to non-empty sequents by :
$[[A_1, A_2, \ldots, A_n]] = [[A_1]].[[A_2]]. \ \ldots \ .[[A_n]]$

The following known property states that such groups are models :

$$\text{if } \Gamma \vdash_L A \text{ then } [[\Gamma]] =_{FG} [[A]]$$

Proposition 2 together with the following completeness result of Pentus is of particular interest for the investigations of $\|=$ -unification.

Theorem 1 (characterization of $\sim$ in L by groups).
For any types t and t' :
$t \sim t'$ in L iff $[[t]] =_{FG} [[t']]$

4 Conjoinability and quasi-groups in NL

The join-equivalence in the non associative version of Lambek grammars enjoys similar properties ; a similar characterization holds using quasi-groups instead of groups. We give the main properties (details may be found in [For01a]).

4.1 Conjoinability in NL

Next property is a central one. It has analogues in the associative case, in

Proposition 3 (Diamond property in NL) *Let t_1 and t_2 be two types. The following assertions are equivalent in NL :*
(i) t_1 and t_2 are conjoinable ($\exists t : t_1 \vdash t$ and $t_2 \vdash t$)
(ii) ($\exists t' : t' \vdash t_1$ and $t' \vdash t_2$)

Proof. We extend the version of Lambek to NL
- if $t_i \vdash t$, for i=1,2, we verify that $t' \vdash t_i$, for i=1,2, where :

$$t' = (t_1 \,/\, ((t \,/\, t) \setminus t)) \otimes ((t \,/\, t) \setminus t_2)$$

- if $t' \vdash t_i$, for i=1,2, we verify that $t_i \vdash t$, for i=1,2, where :

$$t = (t_1 \otimes (t' \setminus t')) \,/\, (t_2 \setminus (t' \otimes (t' \setminus t')))$$ ■

Corollaries.
1. As a corollary in NL, we get that two types t_1 and t_2 are join-equivalent ($t_1 \sim t_2$) iff they have a join, that is (i) ($\exists t : t_1 \vdash t$ and $t_2 \vdash t$) or equivalently iff (ii) ($\exists t' : t' \vdash t_1$ and $t' \vdash t_2$) .
2. Note also that a finite set of types has a join iff the types are pairwise conjoinable.

4.2 Quasi-groups

Quasi-groups are sets equiped with three binary operations $.$, $/$, $\setminus$ that satisfy the following equations :

$x.(x \setminus y) = y$
$(x \,/\, y).y = x$
$x \setminus (x.y) = y$
$(x.y) \,/\, y = x$

The equational theory, here written QG, of quasi-groups admits a canonical rewriting system as found by Knuth and Bendix (70) and further studied in [Hul80].

The above four equations oriented from left to right are completed with two new rules to produce such a canonical rewrite system :

$(x \,/\, y) \setminus x \rightarrow y$
$x \,/\, (y \setminus x) \rightarrow y$

QG-Unification. Solving equations in quasi-groups, or QG-unification admits a complete finite algorithm as shown by [Hul80], whose procedure is based on rewriting. A similar result holds for quasi-groups with identity [Hul80].

From types to FQG. We associate with each type formula A an element in FQG written $[[A]]$ as follows :

$[[p]] = p$ for p atomic
$[[A_1 \setminus A_2]] = [[A_1]] \setminus [[A_2]]$
$[[A_1 / A_2]] = [[A_1]] / [[A_2]]$
$[[A_1 \otimes A_2]] = [[A_1]].[[A_2]]$
We extend the notation to G-terms by :
$[[(A_1, A_2)]] = [[A_1]].[[A_2]]$

Proposition 4 (Models) *If $\Gamma \vdash_{NL} A$ then $[[\Gamma]] =_{FQG} [[A]]$*

4.4 Characterizing NL-conjoinability by quasi-groups

NL-conjoinability and quasi-groups are related, in a way similar to [Pen93] for L-conjoinability and groups.

We recall that QG also denotes the set of four equations defining its equational theory as in section 4.2.

Theorem 2 (characterization of $\sim$ in NL by quasi-groups).
For any types t and t' : $t \sim t'$ in NL iff $[[t]] =_{FQG} [[t']]$

5 $\|=$ - unification in L and NL

$\|=$ -*unification* has been investigated in [For01b] for Lambek calculus. It strongly relies on the characterization of $\sim$ by free groups in [Pen93] in the L case ; in the NL case it involves similarly [For01a] the characterization of $\sim$ by free quasi-groups.

We now give an overview on this operation.

We are first interested in the following relation on types :

Definition 1 *The relation $\|=$ on types is defined by*

$$t_1 \|= t_2 \ \ \textit{iff} \ \ \exists \sigma \ : \ t_1 \vdash \sigma(t_2)$$

where t_1, t_2 are types and σ is a substitution.

We now adapt the standard definitions of unification to deal with deduction.

Definition 2 *Two types A, B are said* $\|=$-unifiable *whenever (i) there exists a type t and a substitution σ such that $t \vdash \sigma(A)$ and $t \vdash \sigma(B)$; σ is then said a* $\|=$-unifier *of A and B and t is a* $\|=$-unificand *of A and B*

2. When A and B have no common variables, this definition (i) is equivalent to (ii) the existence of a type t such that both $t \models A$ and $t \models B$.
   The case without shared variable is precisely the interesting one for the unification step in a learning process such as the *RG*-algorithm in [Kan98].

The following property relates $\models$-unification in L to G-unification ([For01a]):

Proposition 5 (Characterizing $\models$-unifiablity and $\models$-unifiers in L) *Let A and B be two types in Tp, L is such that :*

1. *A and B are $\models$-unifiable iff their images $[[A]]$ and $[[B]]$ are G-unifiable.*
2. *σ is a $\models$-unifier of types A and B iff its translation σ_G is a G-unifier of $[[A]]$ and $[[B]]$, (where σ_G is defined by : $\sigma_G(p) = [[\sigma(p)]]$ for p atomic) ; conversely any G-unifier of $[[A]]$ and $[[B]]$ is the translation of a $\models$-unifier of types A and B.*

Note. A similar result holds for $L_\emptyset$. We now formulate a similar result for NL, relying on quasi-groups instead of groups.

Proposition 6 (Characterizing $\models$-unifiablity and $\models$-unifiers in NL) *Let A and B be two types in Tp, NL is such that :*

1. *A and B are $\models$-unifiable iff their images $[[A]]$ and $[[B]]$ are QG-unifiable.*
2. *σ is a $\models$-unifier of types A and B iff its translation σ_{QG} is a QG-unifier of $[[A]]$ and $[[B]]$, (where σ_{QG} is defined by : $\sigma_{QG}(p) = [[\sigma(p)]]$ for p atomic) ; conversely any QG-unifier of $[[A]]$ and $[[B]]$ is the translation of a $\models$-unifier of types A and B.*

Example 2. Consider the lifting rule that holds in the non-associative NL. Suppose that after some type computation, we get the following grammars (where N denotes a constant, and $x_1, \ldots$ are variables) :
$G_1 = \{c_1 \mapsto x_1,\ c_2 \mapsto x_2,\ c_3 \mapsto (x_1 \,/\, N) \setminus x_2\}$
$G_2 = \{c_1 \mapsto x_3,\ c_2 \mapsto x_4,\ c_3 \mapsto N\}$
$G_3 = \{c_1 \mapsto x_5,\ c_2 \mapsto x_6,\ c_3 \mapsto x_5 \,/\, (N \setminus x_6)\}$
Standard unification fails to produce a rigid grammar since the types for c_3 are not unifiable. However $\models$-unification succeeds in L and in NL since the types for c_3 are $\models$-unifiable :

$$[[(x_1 \,/\, N) \setminus x_1]] =_{FQG} [[N]] =_{FQG} [[x_1 \,/\, (N \setminus x_1)]]$$

Example 3. Consider the Geach rule that holds in the associative calculus but not in the non-associative NL :

6 Constructing Joins via polarized rewriting

In this section we propose a method for constructing joins in NL when such joins exist. Our method is based on the rewriting system and an observation on the rewriting steps from the derivability point of view.

6.1 Rewriting rules and $\vdash$ derivability

Definition 3 *Let us call a rewrite rule* $g \rightarrow d$ increasing *whenever* $g \vdash d$ *and* decreasing *whenever* $d \vdash g$.

We observe that all rules in QG (free quasi-groups) are either increasing $(r1, r2)$ or decreasing (from $r3$ to $r6$) :

$x.(x \setminus y) \vdash y$
$(x \,/\, y).y \vdash x$
$y \vdash x \setminus (x.y)$
$x \vdash (x.y) \,/\, y$
$y \vdash (x \,/\, y) \setminus x$
$y \vdash x \,/\, (y \setminus x)$

6.2 Polarities

We now define the positive and negative occurrences in a formula, as usual.

Definition 4 (Polarities in formulas and sequents) *Each occurrence in a formula has a polarity either positive or negative; we define the positive ones inductively as follows, (the others beeing negative) : an atomic formula has itself as positive occurrence, no negative occurrence. In a formula* $A = A_1 \setminus A_2$, *or* $A = A_2 \,/\, A_1$, *the positive occurrences are the formula* A *itself, the positive occurrences in* A_2 *and the negative occurrences in* A_1 . *In a formula* $A = A_1 \otimes A_2$, *the positive occurrences are the formula* A *itself, the positive occurrences in* A_1 *and* A_2 .

In a sequent $A_1, \ldots A_n \vdash C$ the polarity of an occurrence of B is :
- if this occurrence is in C then its polarity in C,
- else when this occurrence is in A_i, then the opposite of its polarity in A_i.

We extend the notion of polarities to rewriting steps.

Definition 5 *A step of rewriting on a term* A *is said* positive *when it applies at a positive occurrence of* A, *it is said* negative *otherwise.*

6.3 Computing joins on a rewriting chain

$right(x,y,z) = (x \otimes (z \setminus z)) \,/\, (y \setminus (z \otimes (z \setminus z)))$
$left(x,y,t) = (x \,/\, (t^t)) \otimes ((t \,/\, t) \setminus y) \qquad where\ (u^w) = (w \,/\, u) \setminus w$

Let $A = A_1 \rightarrow A_2 \ldots \rightarrow A_n = A^{\downarrow}$ denote the successive terms in the normalization of A by the rewrite system.

Proposition 7 (Construction on a rewriting chain) *For each step i of $A = A_1 \rightarrow A_2 \ldots \rightarrow A_n = A^{\downarrow}$, we define a* left-join *written $lj(A, A_i)$ and a* right-join *$rj(A, A_i)$ for A and A_i, as follows :*

(1) $rj(A_1, A_1) := A_1$

(2) $lj(A_1, A_1) := A_1$

(3) if $A_i \rightarrow A_{i+1}$ is positive with an increasing rule, or negative with a decreasing rule, then $A_i \vdash A_{i+1}$ therefore we put $lj(A, A_{i+1}) = lj(A, A_i)$ and we compute $rj(A, A_{i+1})$ applying Lambek construct to $A, A_{i+1}, lj(A, A_{i+1})$, that is :

(3.1) $lj(A, A_{i+1}) := lj(A, A_i)$

(3.2) $rj(A, A_{i+1}) := right(A, A_{i+1}, lj(A, A_{i+1}))$

(4) if $A_i \rightarrow A_{i+1}$ is positive with a decreasing rule, or negative with an increasing rule, then $A_{i+1} \vdash A_i$ therefore we put $rj(A, A_{i+1}) = rj(A, A_i)$ and we compute $lj(A, A_{i+1})$ applying Lambek diamond construct to $A, A_{i+1}, rj(A, A_{i+1})$, that is :

(4.1) $rj(A, A_{i+1}) := rj(A, A_i)$

(4.2) $lj(A, A_{i+1}) := left(A, A_{i+1}, rj(A, A_{i+1}))$

We get indeed from the above construction and details that :

$lj(A, A_i) \vdash A \vdash rj(A, A_i)$

$lj(A, A_i) \vdash A_i \vdash rj(A, A_i)$

in other words $lj(A, A_i)$ is a left join of A and A_i and $rj(A, A_i)$ is a right join of A and A_i

6.4 Computing joins of two terms along their rewriting chains

Suppose A and B are conjoinable, by the characterization property this is equivalent to $A^{\downarrow} = B^{\downarrow}$ by the rewriting system for quasi-groups.

Let $A = A_1 \rightarrow A_2 \ldots \rightarrow A_n = A^{\downarrow}$ and $B = B_1 \rightarrow B_2 \ldots \rightarrow B_m = B^{\downarrow}$ denote respective chains of normalization by the rewrite system.

We then get :

(1.1) $lj(A, A_n) \vdash A$ and $A \vdash rj(A, A_n)$
(1.2) $lj(A, A_n) \vdash A_n = B_m$ and $A_n = B_m \vdash rj(A, A_n)$
(2.1) $lj(B, B_m) \vdash B$ and $B \vdash rj(B, B_m)$
(2.2) $lj(B, B_m) \vdash A_n = B_m$ and $A_n = B_m \vdash rj(B, B_m)$

$D := left(lj(A, A_n), lj(B, B_m), A_n)$
$C := right(rj(A, A_n), rj(B, B_m), B_m)$
These terms C and D are such that :

(5.1) $D \vdash lj(A, A_n) \vdash A$
(5.2) $D \vdash lj(B, B_m) \vdash B$
(6.1) $A \vdash rj(A, A_n) \vdash C$
(6.2) $B \vdash rj(B, B_m) \vdash C$

that is C and D are respectively a right-join and a left-join of A and B as desired. We may summarize this as follows :

Proposition 8 *Let* $A = A_1 \rightarrow A_2 \ldots \rightarrow A_n = A^{\downarrow}$ *and* $B = B_1 \rightarrow B_2 \ldots \rightarrow B_m = B^{\downarrow}$ *denote respective chains of normalization by the rewrite system. The following construct* $D := left(lj(A, A_n), lj(B, B_m), A_n)$ *and* $C := right(rj(A, A_n), rj(B, B_m), B_m)$ *define respectively a left-join and a right-join of* A *and* B

6.5 Example (sketch)

We consider A B as follows :

(a) $A = A_1 = z.(z \setminus (x \,/\, y) \setminus x)$
and $B = B_1 = y$

A yields rewriting steps, according to two deductions

an increasing one :
(b1) $A_1 = z.(z \setminus (x \,/\, y) \setminus x) \vdash_{inc} (x \,/\, y) \setminus x = A_2$
followed by a decreasing one :
(b2) $A_3 = y \vdash (x \,/\, y) \setminus x = A_2$

We now compute the joins for A :

(c1) $rj(A_1, A_1) = lj(A_1, A_1) = A_1$
(c2) $lj(A_1, A_2) = lj(A_1, A_1) = A_1$
(c2') $rj(A_1, A_2) = right(A_1, A_2, A_1) = ...$
(c3) $rj(A_1, A_3) = rj(A_1, A_2)$
(c3') $lj(A_1, A_3) = left(A_1, A_3, rj(A_1, A_3))$
(c3'=) $lj(A_1, A_3) = left(A_1, A_3, right(A_1, A_2, A_1))$

as concerns B, there is no rewriting steps

(d3) $rj(B_1, B_1) = lj(B_1, B_1) = B_1 = y$

we conclude with

(e1) $lj(A_1, A_3)$ and $lj(B_1, B_1)$ have $A_3 = B_1 = y$ as right-join

7 Conclusion

Unification plays a crucial role in the acquisition of categorial grammar, used in the field of computational linguistics. After an overview of a modified unification proposed in [For01a] in this context for Lambek categorial grammars and of an overview the conjoinability relation [Lam58,Pen93] we have focused on the quasi-groups that characterize the non-associative version. We have shown how to compute joins of the conjoinability relation, when they exists, this method is based on quasi-group rewriting while using "Lambek diamond construct".

For futur algorithmic developments in the context of Lambek categorial grammars, in automatic acquisition applications or compacting grammar issues, we may thus hope to benefit from results (such as on groups or quasi-groups), in the field of term rewriting systems and unification.

References

[AT95] E. Aarts and K. Trautwein. Non-associative Lambek categorial grammar in polynomial time. *Mathematical Logic Quaterly*, 41:476–484, 1995.

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

[BP90] Wojciech Buszkowski and Gerald Penn. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454, 1990.

[Bus97] W. Buszkowski. Mathematical linguistics and proof theory. In van Benthem and ter Meulen [vBtM97], chapter 12, pages 683–736.

[dG99] Philippe de Groote. Non-associative Lambek calculus in polynomial time. In *8^th Workshop on theorem proving with analytic tableaux and related methods*, number 1617 in Lecture Notes in Artificial Intelligence. Springer-Verlag, March 1999.

[FL02] Annie Foret and Yannick Le Nir. Lambek rigid grammars are not learnable from strings. In *COLING'2002, 19th International Conference on Computational Linguistics*, Taipei, Taiwan, 2002.

[For01a] Annie Foret. Conjoinability and unification in Lambek categorial grammars. In *New Perspectives in Logic and Formal Linguisitics, Proceedings Vth ROMA Workshop*, Roma, 2001. Bulzoni Editore.

[For01b] Annie Foret. Mixing deduction and substitution in Lambek categorial grammars, some investigations. In *LACL'01, 4th International Conference on Logical Aspects of Computational Linguistics*, number 2099 in Lecture Notes in Artificial Intelligence, Le Croisic, France, 2001. Springer-Verlag.

[Gol67] E.M. Gold. Language identification in the limit. *Information and control*, 10:447–474, 1967.

[Hul80] J. M. Hullot. *Compilation de formes canoniques dans des théories équationelles*. PhD thesis, University de Paris Sud, 1980.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American mathematical monthly*, 65:154–169, 1958.

[Lam61] Joachim Lambek. On the calculus of syntactic types. In Roman Jakobson, editor, *Structure of language and its mathematical aspects*, pages 166–178. American Mathematical Society, 1961.

[Moo97] Michael Moortgat. Categorial type logic. In van Benthem and ter Meulen [vBtM97], chapter 2, pages 93–177.

[Nic99] Jacques Nicolas. Grammatical inference as unification. Rapport de Recherche RR-3632, INRIA, 1999. http://www.inria.fr/RRRT/publications-eng.html.

[Pen93] Mati Pentus. Lambek grammars are context-free. In *Logic in Computer Science*. IEEE Computer Society Press, 1993.

[Pen93] M. Pentus. The conjoinability relation in Lambek calculus and linear logic. ILLC Prepublication Series ML–93–03, Institute for Logic, Language and Computation, University of Amsterdam, 1993.

[RB01] Christian Retoré Roberto Bonato. Learning rigid lambek grammars and minimalist grammars from struc tured sentences. *Third workshop on Learning Language in Logic, Strasbourg*, september 2001.

[vBtM97] J. van Benthem and A. ter Meulen, editors. *Handbook of Logic and Language*. North-Holland Elsevier, Amsterdam, 1997.

[Ver96] Koen Versmissen. *Grammatical Composition: Modes, Models, Modalities.* PhD thesis, Utrecht University, 1996.