Conjoinability and unification in Lambek categorial grammars

Annie Foret
Email : foret@irisa.fr
IRISA and University of Rennes1, FRANCE

December 18, 2001

Abstract

Recently, learning algorithms in Gold's model have been proposed for some particular classes of classical categorial grammars [Kan98]. We are interested here in learning Lambek categorial grammars.

In general grammatical inference uses unification and substitution. In the context of Lambek categorial grammars it seems appropriate to incorporate an operation on types based both on deduction (Lambek derivation) and on substitution instead of standard substitution and standard unification.

After an introduction (in connection with learning), this paper will recall conjoinability results [Lam58, Pen93] - using groups - ; we then consider a characterization of conjoinability - using quasi-groups - for the non-associative version of Lambek calculus. We then relate these characterizations to the modified unification investigated in [For01] for the associative Lambek calculus.

1 Introduction

Categorial grammars have been studied in the domain of natural language processing; these grammars are lexicalized grammars that assign types (or categories) to the lexicon. We focus here on Lambek categorial grammars [Lam58] to which linear logic [Gir95] is closely connected. We also consider the Non-Associative Lambek Calculus introduced in [Lam61] that avoids the associativity of formula concatenation.

Learning (in the sense of Gold [Gol67]) in our context is a symbolic issue that may be described as follows. Let $\mathcal{G}$ be a class of grammars, that we wish to learn from examples. The issue is to define an algorithm, that when applied to a finite set of sentences, yields a grammar in the class that generates the examples; the

algorithm is also required to converge. Formally, let $L(G)$ denote the language associated with grammar G, and let V be a given alphabet, a learning algorithm is a function ϕ from finite sets of words in V^* to $\mathcal{G}$, such that for $G \in \mathcal{G}$ with $L(G) = (e_i)_{i \in N}$ there exists a grammar $G' \in \mathcal{G}$ and there exists $n_0 \in N$ such that : $\forall n > n_0\ \phi(\{e_1, \ldots, e_n\}) = G' \in \mathcal{G}$ with $L(G') = L(G)$.

One good reason to use categorial grammars in a learning perspective is that they are fully lexicalized : the rules are already known, only types assigned to words have to be derived from examples. Essential known results on this subject may be found in [Kan98]. The learning technique avoids to add a new type each time there is a new use of a word in an example, but applies a unification algorithm instead. One important case is when we limit the number of types per word for example to only one type.

Our aim is to explore learning mechanisms for Lambek calculus. Whereas standard unification is used in existing works such as [BP90, Kan98, Nic99, Bon00], in such a context, it seems appropriate to use not only unification but also deduction to combine grammars in the learning process. This issue is discussed in [For01] for the associative version of the Lambek calculus where we define a preorder $\|=$ on types and a $\|=$ -unification that we characterize in terms of groups. This paper is in some sense complementary to it, we show similar results for the non-associative version in terms of quasi-groups. As detailed after, $\|=$ -unification and conjoinability as introduced in [Lam58] are strongly connected.

The paper is organized as follows. Section 2 addresses background definition and known results. Section 3 gives a characterization of conjoinability in non associative Lambek calculus : we follow the main steps of [Pen93] for the associative case ; we here establish a connection with quasi-groups instead of groups. As pointed by M. Mortgaat, such a characterization of conjoinability in the non-associative case was observed in [Ver96] but without mentionning quasi-groups. Section 4 applies these conjoinability results to $\|=$ -unification mixing deduction and substitution ; we establish a characterization of $\|=$ -unifiability in the non-associative version, in terms of unification in quasi-groups ; this equational theory has been treated in the context of term rewriting systems by [Hul80], who provides a unification algorithm.

2 Background

2.1 Categorial grammars

In this section, we introduce basic definitions concerning categorial grammars. The interested reader may also consult [Cas88, Ret00, Bus97, Moo97] for an introduction or for further details.

Let Σ be a fixed alphabet.

Types. *Types* are constructed from Pr (set of *primitive types*) and three binary connectives $/$, $\backslash$ and $\otimes$ for products. In some versions, we drop $\otimes$.

Tp denotes the set of types. Pr contains a *distinguished type*, written S, also called the *principal type* ; we assume that $Pr = Pr_c \cup Var$, where Pr_c is a set of *constants* with $S \in Pr_c$ and Var is a set of *variables* (variables are different from constants).

Classical categorial grammar. A *classical 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 $G : c \mapsto A$.

Derivation $\vdash_{AB}$ on types without $\otimes$. The relation $\vdash_{AB}$ is the smallest relation $\vdash$ between Tp^+ and Tp, such that for all $\Gamma, \Delta \in Tp^+$ and for all $A, B \in Tp$:

$A \vdash A$
if $\Gamma \vdash A$ and $\Delta \vdash A \setminus B$ then $\Gamma, \Delta \vdash B$ (Backward application)
if $\Gamma \vdash B / A$ and $\Delta \vdash A$ then $\Gamma, \Delta \vdash B$ (Forward application)

We give a formulation of Lambek calculus, written L, including products consisting in introduction rules on the left and on the right of a sequent. For Lambek calculus without products, one simply drops the rules for $\otimes$.

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 $A, B \in Tp$:

$A \vdash A$
if $A, \Gamma \vdash B$ then $\Gamma \vdash A \setminus B$ ($\backslash right$)
if $\Gamma, A \vdash B$ then $\Gamma \vdash B / A$ ($/right$)
if $\Gamma \vdash A$ and $\Delta, B, \Delta' \vdash C$ then $\Delta, \Gamma, A \setminus B, \Delta' \vdash C$ ($\backslash left$)
if $\Gamma \vdash A$ and $\Delta, B, \Delta' \vdash C$ then $\Delta, B / A, \Gamma, \Delta' \vdash C$ ($/left$)
if $\Delta, A, B, \Delta' \vdash C$ then $\Delta, (A \otimes B), \Delta' \vdash C$ ($\otimes$-left)
if $\Gamma \vdash A$ and $\Gamma' \vdash B$ then $\Gamma, \Gamma' \vdash (A \otimes B)$ ($\otimes$-right)

When we replace Tp^+ by Tp^* in $\Gamma \in Tp^+$ in the definition above, we get another version of Lambek calculus, without the non-empty left hand-side requirement, which we refer to as $L_\emptyset$ with derivation relation $\vdash_{L_\emptyset}$.

We now give a formulation of Non associative Lambek calculus, written NL, including products. In the Gentzen presentation, the derivability relation holds between a term in $\mathcal{S}$ and a formula in Tp, where the term language is $\mathcal{S} ::=$

$Tp|(\mathcal{S},\mathcal{S})$. Terms in $\mathcal{S}$ are also called *G-terms*. A sequent is a pair $(\Gamma, A) \in \mathcal{S} \times Tp$. The notation $\Gamma[\Delta]$ represents a G-term with a distinguished occurrence of Δ (with the same position in premise and conclusion of a rule).

Non associative Lambek Derivation $\vdash_{NL}$. The relation $\vdash_{NL}$ is the smallest relation $\vdash$ between $\mathcal{S}$ and Tp, such that for all $\Gamma, \Delta \in \mathcal{S}$ and for all $A, B, C \in Tp$:

$A \vdash A$
if $(A, \Gamma) \vdash B$ then $\Gamma \vdash A \setminus B$ ($\setminus right$)
if $(\Gamma, A) \vdash B$ then $\Gamma \vdash B \,/\, A$ ($/right$)
if $\Gamma \vdash A$ and $\Delta[B] \vdash C$ then $\Delta[(\Gamma, A \setminus B)] \vdash C$ ($\setminus left$)
if $\Gamma \vdash A$ and $\Delta[B] \vdash C$ then $\Delta[(B \,/\, A, \Gamma)] \vdash C$ ($/left$)
if $\Delta[(A, B)] \vdash C$ then $\Delta[A \otimes B] \vdash C$ ($\otimes$-left)
if $\Gamma \vdash A$ and $\Delta \vdash B$ then $(\Gamma, \Delta) \vdash (A \otimes B)$ ($\otimes$-right)

Gentzen terms translation. For $\Delta \in \mathcal{S}$, we define its formula translation Δ° as : $(\Delta_1, \Delta_2)^\circ = \Delta_1^\circ \otimes \Delta_2^\circ$ and $A^\circ = A$ for $A \in Tp$.

Note. We recall that the cut rule is satisfied by $\vdash_{AB}$, $\vdash_L$ and $\vdash_{NL}$.

Notation. In some sections, we may write simply $\vdash$ instead of $\vdash_{AB}$, $\vdash_L$ or $\vdash_{NL}$.

Language. Let G be a classical 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 $A_1, \dots, A_n \vdash_{AB} t$
The *language of* G, written $L(G)$, is the set of strings generated by G. We define similarly $L_L(G)$ replacing $\vdash_{AB}$ with $\vdash_L$ in the definition of $L(G)$. We also define similarly $L_{NL}(G)$ replacing $\vdash_{AB}$ by $\vdash_{NL}$ in the sequent where the types are parenthesized in some way.

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 $(John\ likes\ Mary) \in L(G_1)$ since ($N,\ N \setminus (S \,/\, N),\ N \vdash_{AB} S$)
G_1 is a rigid (or 1-valued) grammar.

2.2 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;

$\sqsubseteq$ on grammars as follows : $G_1 \sqsubseteq G_2$ iff $\exists \sigma : \sigma(G_1) \subseteq G_2$ [2] where we write $G_1 \subseteq G_2$ whenever G_2 contains all type assignments of G_1.

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 $\forall \theta \in \; U(\mathcal{A}) \; : \sigma \preceq \theta$.

[1] for a calculus including $\otimes$

[2] this is a simplified version : Kanazawa adds a faithfullness condition such that if two types assigned to the same symbol in G_1 are distinct they are kept distinct by σ which has no impact in the rigid case.

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.

2.3 Conjoinability

We now define the conjoinability relation introduced in [Lam58]. We also view this equivalence as the special case of $\|=$ -unifiability 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 $t_1, t_2, ...t_n$.

2.4 Conjoinability and group models in Lambek Calculus

We now recall some properties of conjoinability in the $\vdash_L$ case. The following result is due to Lambek.

Proposition 1 (Diamond property in L) *Let t_1 and t_2 be two types. The following assertions are equivalent in L :*
(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 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* $[[t]] =_{FG} [[t']]$

3 Conjoinability and quasi-groups in NL

3.1 Conjoinability in NL

We first get the following useful congruence property. Recall that a *pre-congruence* is a binary relation that is *compatible* with the operations here with respect to $/$, $\setminus$, $\otimes$; a pre-congruence is a *congruence* if it is an equivalence.

Proposition 3 *In NL : $\sim^1$ and $\sim$ are pre-congruences and $\sim$ is a congruence.*

Proof. We get easily that $\sim^1$ is a pre-congruence :
- if $A \vdash B$ then $(A, C) \vdash B \otimes C$ then $A \otimes C \vdash B \otimes C$;
- if $A \vdash B$ then $(A \,/\, C, C) \vdash B$ then $A \,/\, C \vdash B \,/\, C$;
- if $A \vdash B$ then $(C \,/\, B, A) \vdash C$ then $C \,/\, B \vdash C \,/\, A$;
the other cases are similar. We then get that $\sim$ is a congruence as a corollary ■

Next property is a central one. It has analogues in the associative case, in [Lam58] or in [Pen93] where the constructed types are different. In the following property we take the construction in [Lam58] and show that it can be extended in the non-associative case (the simplified one in [Pen93] is not suitable[3], this is also observed in [Ver96]).

Proposition 4 (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 [4] 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 [5] 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 ($t_1 \sim t_2$)

[3] it involves different parenthesizing

[4] from $t_2 \vdash t$:
$((t \,/\, t), (t \,/\, t) \setminus t_2) \vdash t$ (by $\setminus$ -left)
$(t \,/\, t) \setminus t_2 \vdash (t \,/\, t) \setminus t$ (by $\setminus$ -right)
$(t_1 \,/\, ((t \,/\, t) \setminus t)) \otimes ((t \,/\, t) \setminus t_2) \vdash t_1$ (by $/$ -left, $\otimes$-left)
hence $t' \vdash t_1$ and secondly $t' \vdash t_2$ from $t_1 \vdash t$ and $t \vdash (t \,/\, t) \setminus t$:
$(t_1 \,/\, ((t \,/\, t) \setminus t), t) \vdash t$ (by $/$ -left)
$(t_1 \,/\, ((t \,/\, t) \setminus t)) \vdash (t \,/\, t)$ (by $/$ -right)
$(t_1 \,/\, ((t \,/\, t) \setminus t)) \otimes ((t \,/\, t) \setminus t_2) \vdash t_2$ (by $\setminus$ -left, $\otimes$-left)

[5] from $t' \vdash t_2$ and $t' \otimes (t' \setminus t') \vdash t'$:
$(t', t_2 \setminus (t' \otimes (t' \setminus t'))) \vdash t'$ (by $\setminus$ -left)
$t_2 \setminus (t' \otimes (t' \setminus t')) \vdash (t' \setminus t')$ (by $\setminus$ -right)
$(t_1, (t_2 \setminus (t' \otimes (t' \setminus t')))) \vdash (t_1 \otimes (t' \setminus t'))$ (by $\otimes$-right)
hence $t_1 \vdash t$ and secondly $t_2 \vdash t$ from $t' \vdash t_1$:
$(t' \otimes (t' \setminus t')) \vdash (t_1 \otimes (t' \setminus t'))$ (by $\otimes$-right, $\otimes$-left)
$(t_2, (t_2 \setminus (t' \otimes (t' \setminus t')))) \vdash (t_1 \otimes (t' \setminus t'))$ (by $\setminus$ -left)

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 $t' \vdash t_2$) . [6]

2. Note also that a finite set of types has a join iff the types are pairwise conjoinable.

3.2 Quasi-groups

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

$x.(x \backslash y) = y$
$(x / y).y = x$
$x \backslash (x.y) = y$
$(x.y) / y = x$

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 :

$(x / y) \backslash x \rightarrow y$
$x / (y \backslash x) \rightarrow y$

Variants. The case of quasi-groups with a neutral element e is similar, see [Hul80] for details. [7]

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].

---

[6] if (i) then $t_1 \sim t_2$ is obvious. If $t_1 \sim t_2$ we proceed by induction on n such that for $1 \leq i < n$: $s_i \vdash s_{i+1}$ *or* $s_{i+1} \vdash s_i$: if $s_n \vdash s_{n+1} = t_2$ then by induction $\exists s_0$ such that $s_0 \vdash s_i (i < n+1)$ from which by cut $s_0 \vdash s_i (i \leq n+1)$; if $s_{n+1} \vdash s_n$ then by induction $\exists s'$ such that $s_i \vdash s' (i < n+1)$ from which by cut $s_i \vdash s' (i \leq n+1)$ ■

[7] one adds the equations :

$e.x = x$
$x.e = x$

that are also oriented from left to right and a canonical rewrite system is obtained by adding the following rules :

$e \backslash x \rightarrow x$
$x / e \rightarrow x$
$x / x \rightarrow e$
$x \backslash x \rightarrow e$

3.3 Quasi-groups as models

We show that quasi-groups are models for $\vdash_{NL}$. Our interest is to use them as (complete) models for conjoinability.

Free quasi-group. Let FQG denote the free quasi-group with generators Pr, operations ., $/$ $\backslash$.

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

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

Proposition 5 (Models) *If* $\Gamma \vdash_{NL} A$ *then* $[[\Gamma]] =_{FQG} [[A]]$

Proof. We may proceed by induction on derivations (see Annex).

Residuated groupoids. We may also consider quasi-groups as a particular case of residuated groupoids that are known as (complete) algebraic models [Bus97] for NL. A *residuated groupoid* is a structure $(M, \circ, \Rightarrow, \Leftarrow, \leq)$ such that $(M, \circ)$ is a groupoid, $\leq$ is a partial ordering on M, and $\Rightarrow, \Leftarrow$ are binary operations on M which fullfill the equivalences (for $a, b, c \in M$) : $a \leq c \Leftarrow b$ iff $a \circ b \leq c$ iff $b \leq a \Rightarrow c$. In the case of quasi-groups, we may take (. , $\backslash$, $/$, $=$) as ($\circ, \Rightarrow, \Leftarrow, \leq$) for which the above equivalences are easy to verify. [8]

From FQG to types. We define Φ from FQG to Tp as the morphism from $(FQG, ., \backslash, /)$ to $(Tp, \otimes, \backslash, /)$ defined by $\Phi(p) = p$ for p atomic; (by this we mean [9] : $\Phi(p) = p$ for p atomic , $\Phi(a.b) = \Phi(a) \otimes \Phi(b)$, $\Phi(a \backslash b) = \Phi(a) \backslash \Phi(b)$, $\Phi(a / b) = \Phi(a) / \Phi(b)$).

Remarks. The following facts are straightforward :
1. $\Phi([[A]]) = A$
2. $[[\Phi(a)]] =_{FQG} a$
3. If $a =_{FQG} b$ then $[[\Phi(a)]] =_{FQG} [[\Phi(b)]]$ [10]

[8] if $a = c / b$ then $a.\, b = (c / b).\, c = c$; if $a.\, b = c$ then $c / b = (a.\, b) / b = a$; the other verification are dual to these ones.

[9] For the variant with neutral element, we add : $\Phi(e) = S \backslash S$

[10] Note however that we may have $a =_{FQG} b$ with $\Phi(a) \neq \Phi(b)$

3.4 Characterizing NL-conjoinability by quasi-groups

We show how 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 3.2.

Proposition 6 (quasi-group equality and $\sim$ in NL)
1. If $a = b$ is an instance of an equation in QG then $\Phi\ (a) \sim^1 \Phi\ (b)$
2. If $a =_{FQG} b$ then $\Phi\ (a) \sim \Phi\ (b)$

Proof. See Annex.

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

Proof. This property is a corollary of previous results as follows :
- if $t \sim t'$ then we can find t_0 such that both $t_0 \vdash t$ and $t_0 \vdash t'$ therefore since quasi-groups are models : $[[t_0]] =_{FQG} [[t]] =_{FQG} [[t']]$;
- if $[[t]] =_{FQG} [[t']]$ then we get $\Phi\ ([[t]]) \sim \Phi\ ([[t']])$ by prop 6 ; but we also have by remark 1 $\Phi\ ([[t]]) = t$ and $\Phi\ ([[t']]) = t'$, hence the result ■
An alternate proof is to consider a quasi-group on classes for $\sim$.

4 An application to unification for learning in NL

$\|=$ *-unification* has been investigated in [For01] in the associative Lambek calculus. It strongly relies on the characterization of $\sim$ by free groups in [Pen93].

In this section, we proceed similarly for NL using the characterization of $\sim$ by free quasi-groups.

In Lambek calculus, from a learning point of view, we are 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 recall standard definitions on unification modulo a set of equations.

Unification in (modulo) an equational theory. For any set E of equations, let $=_E$ denote the equivalence generated by E. We say that a set of terms $\mathcal{A}$ is unifiable modulo E (or E-unifiable) whenever there exists a substitution σ (then called E-unifier), such that for all terms $t, t' \in \mathcal{A} : \sigma(t) =_E \sigma(t')$. Unification has been studied for many equational theories, in particular in connection with rewriting. We recall that free groups (in the abelian case, and also in the non abelian case) admit a finite presentation by equations, that produces a canonical rewriting system; let G denote such a set of equations for non abelian groups.

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.*
These definitions are extended to sets of types in the usual way.

Two remarks :

1. If σ is a (standard) unifier of types A and B (ie $\sigma(A) = \sigma(B)$) then σ also is a $\|=$ -unifier of A and B (taking $\sigma(A)$ as left-hand side).

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 $t \|= A$ and $t \|= B$.

   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 $\|=$ -unification in L to G-unification ([For01]):

Proposition 7 (Characterizing $\|=$ -unifiablity and $\|=$ -unifiers in L) *Let A and B be two types in Tp, L is such that :*

1. *A and B are $\|=$ -unifiable iff their images $[[A]]$ and $[[B]]$ are G-unifiable.*

2. *σ is a $\|=$ -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 $\|=$ -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 8 (Characterizing $\|=$ -unifiablity and $\|=$ -unifiers in NL) *Let A and B be two types in Tp, NL is such that :*

1. *A and B are $\|=$ -unifiable iff their images $[[A]]$ and $[[B]]$ are QG-unifiable.*

2. *σ is a $\|=$ -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 $\|=$ -unifier of types A and B.*

Proof. See Annex.

We may thus benefit from existing results for quasi-groups such as the existence of term rewriting systems and unification algorithms. Let us conclude with examples.

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 $\|=$ -unification succeeds in L and in NL since the types for c_3 are $\|=$ -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 :*

$$[[A \,/\, B]] =_{FQG} [[A]] \,/\, [[B]]$$

$$[[(A \,/\, C) \,/\, (B \,/\, C)]] =_{FQG} ([[A]] \,/\, [[C]]) \,/\, ([[B]] \,/\, [[C]])$$

do not simplify in quasi-groups for atomic A, B, C. Whereas in a group interpretation they get reduced to the same formula $[[A]].[[B]]^{-1}$.

References

[Bon00] Roberto Bonato. *A Study On Learnability for Rigid Lambek Grammars*. Italian laurea degree master thesis and irisa research report, to appear, 2000.

[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.

[Cas88] Claudia Casadio. Semantic categories and the development of categorial grammars. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 95–124. Reidel, Dordrecht, 1988.

[For01] Annie Foret. On mixing deduction and substitution in Lambek categorial grammars. *LACL'01, Le Croisic*, 2001.

[Gir95] Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 1–42. Cambridge University Press, 1995.

[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.

[Kan98] Makoto Kanazawa. *Learnable classes of categorial grammars*. Studies in Logic, Language and Information. FoLLI & CSLI, 1998. distributed by Cambridge University Press.

[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] 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.

[Ret00] Christian Retoré. Systèmes déductifs et traitement des langues:un panorama des grammaires catégorielles. Technical Report RR-3917 2000, INRIA, Rennes, France, 2000. A revised version to appear in Traitement automatique du langage naturel, TSI.

[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.

ANNEX

Proof of proposition 5 by induction on derivations

case (axiom) is obvious.

case ($\backslash right$) if $(A, \Gamma) \vdash B$ then $\Gamma \vdash A \setminus B$

By induction $[[A]].[[\Gamma]] =_{FQG} [[B]]$, using $[[\Gamma]] =_{FQG} [[A]] \setminus ([[A]].[[\Gamma]])$
this entails : $[[\Gamma]] =_{FQG} [[A]] \setminus [[B]] = [[A \setminus B]]$;

case ($/right$) is treated similarly.

case ($\backslash left$) if $\Gamma \vdash A$ and $\Delta[B] \vdash C$ then $\Delta[(\Gamma, A \setminus B)] \vdash C$

By induction $\Gamma =_{FQG} A$ and $[[\Delta[B]]] =_{FQG} [[C]]$
which entails using $[[A]].([[A]] \setminus [[B]]) =_{FQG} B$:
$[[(\Gamma, A \setminus B)]] =_{FQG} [[A]].([[A]] \setminus [[B]]) =_{FQG} B$
then $[[\Delta[(\Gamma, A \setminus B)]]] =_{FQG} [[\Delta[B]]] =_{FQG} [[C]]$

case ($/left$) is similar.

case ($\otimes$-left) is obvious

case ($\otimes$-right) if $\Gamma \vdash A$ and $\Delta \vdash B$ then $(\Gamma, \Delta) \vdash (A \otimes B)$

By induction $[[\Gamma]] =_{FQG} [[A]]$ and $[[\Delta]] =_{FQG} [[B]]$
which entails $[[(\Gamma, \Delta)]] =_{FQG} [[A]].[[B]] = [[A \otimes B]]$ ■

Proof of proposition 6. Part 1 :

$$\begin{array}{lll}
\Phi\ (x.(x \setminus y)) \sim^1 \Phi\ (y) & & \text{from } x \otimes (x \setminus y) \vdash y \\
\Phi\ ((x\ /\ y).y) \sim^1 \Phi\ (x) & & \text{similarly from } (x\ /\ y) \otimes y \vdash x \\
\Phi\ (x \setminus (x.y)) \sim^1 \Phi\ (y) & & \text{from } y \vdash x \setminus (x \otimes y) \\
\Phi\ ((x.y)\ /\ y) \sim^1 \Phi\ (x) & & \text{similarly from } x \vdash (x \otimes y)\ /\ y
\end{array}$$

Part 2. We proceed by induction on an equational deduction for $a =_{FQG} b$. Such a deduction either is an instance of an equation, an instance of the reflexivity rule or follows from the symmetry rule, from the transitivity rule, or from the compatibility rules. The case of an equation is dealt by part 1 of this proposition. The cases of reflexivity, symmetry and transitivity are obvious. The case of compatibility is obtained by the congruence property for $\sim$ ■

Proof of proposition 8. We proceed in several steps.

- The assertion "A and B are $\|=$ -unifiable" is equivalent to "there exists C and σ such that both $C \vdash \sigma(A)$ and $C \vdash \sigma(B)$"; from the conjoinability results, this is still equivalent to $\sigma(A) \sim \sigma(B)$ and also to $[[\sigma(A)]] =_{FQG} [[\sigma(B)]]$ from the characterization of $\sim$ by quasi-groups.

- Let σ_{QG} denote the QG-substitution defined by $\sigma_{QG}(p) = [[\sigma(p)]]$ (for p atomic). We can easily show that $\sigma_{QG}([[t]]) =_{FQG} [[\sigma(t)]]$ for any type t. [11]

- Suppose that $[[\sigma(A)]] =_{FQG} [[\sigma(B)]]$, we also have $\sigma_{QG}([[A]]) =_{FQG} [[\sigma(A)]]$ and $\sigma_{QG}([[B]]) =_{FQG} [[\sigma(B)]]$, therefore σ_{QG} is a QG-unifier of $[[A]]$ and $[[B]]$. Conversely, suppose that σ_{QG} is a QG-unifier, using the same arguments, we get $[[\sigma(A)]] =_{FQG} [[\sigma(B)]]$. Which provides the first assertion in part 2 of the proposition.

- Suppose that θ is any is QG-unifier of $[[A]]$ and $[[B]]$ we consider the substitution θ_{NL} defined by $\theta_{NL}(p) = \Phi(\theta(p))$ (for p atomic) and show that it is a $\|=$ -unifier of A and B. We first get $[[\theta_{NL}(p)]] =_{def} [[\Phi(\theta(p))]] =_{FQG} \theta(p)$ (for p atomic), from previous remark 2 on Φ . We then get $[[\theta_{NL}(t)]] =_{FQG} \theta([[t]])$ for any t, by easy induction on t [12]. This provides the second assertion in part 2 of the proposition, since $\theta([[A]]) =_{FQG} \theta([[B]])$ implies $[[\theta_{NL}(A)]] =_{FQG} [[\theta_{NL}(B)]]$

- Finally we get part 1 from the whole construction above ■

---

[11] case t atomic is clear from the definition of σ_{QG} since $[[p]] = p$; case $t = t_1 \setminus t_2$ is $\sigma_{QG}([[t_1 \setminus t_2]]) = \sigma_{QG}([[t_1]] \setminus [[t_2]]) = \sigma_{QG}([[t_1]]) \setminus \sigma_{QG}([[t_2]]) =_{ind} [[\sigma(t_1)]] \setminus [[\sigma(t_2)]] = [[\sigma(t_1) \setminus \sigma(t_2)]] = [[\sigma(t)]]$; the other cases are similar.
This may also be viewed as two compositions of morphisms : $\sigma_{QG} \circ [[]]$ and $[[]] \circ \sigma$, that are equal on atomic formulas.

[12] case t atomic is clear from the definition of θ_{NL} since $[[p]] = p$; case $t = t_1 \setminus t_2$ is $\theta([[t_1 \setminus t_2]]) = \theta([[t_1]] \setminus [[t_2]]) = \theta([[t_1]]) \setminus \theta([[t_2]]) =_{ind} [[\theta_{NL}(t_1)]] \setminus [[\theta_{NL}(t_2)]] = [[\theta_{NL}(t_1) \setminus \theta_{NL}(t_2)]] = [[\theta_{NL}(t)]]$; the other cases are similar.