APPRENTISSAGE DE GRAMMAIRES CATEGORIELLES DE MODULES

Rapport de stage de:

Camilo THORNE

redigé sous la direction de:

M. **D. BECHET**

D.E.A. d'Intelligence Artificielle et
Optimisation Combinatoire,
Universités de Paris 8 – Paris 13.

Villetaneuse, le 20 septembre 2004.

Table des matières

Introduction

Le but de ce travail est d'apprendre, via la méthode de l' inférence grammatical issue du modèle de Gold, à partir d'informations syntaxiques, des grammaires catégorielles. On le fera en suivant de près les méthodes développées par Kanazawa ainsi que des résultats généraux d'apprenabilité démontrés par lui dans [6].

Le modèle de Gold est autrement connu comme celui de l'identification à la limite à partir d'exemples positifs. Ce modèle requiert la donnée d'un cadre conceptuel $\langle \mathcal{E}, \mathcal{H}, M \rangle$; $\mathcal{E}$ dénotant un espace d'échantillons (ou d'instances positives), $\mathcal{H}$ un espace d'hypothèses ou concepts et M une fonction telle que pour chaque $h \in H$, $M(h) \subseteq \mathcal{E}$. On peut ainsi dire que, *modulo* la fonction M, l'hypothèse h génère la partie $M(h)$ de $\mathcal{E}$ ou que les hypothèses partitionnent l'espace $\mathcal{E}$, etc. L'apprentissage revient ensuite à faire comme une sorte d'induction : à retrouver l'hypothèse ou concept sous-jacent à une partie arbitraire de $\mathcal{E}$. Ce qui s'exprime formellement par la définition d'une certaine procédure effective : un algorithme ϕ, dit *algorithme d'apprentissage*, vérifiant certaines propriétés, notamment que ϕ converge, dans un sens à préciser, vers quelque $h \in \mathcal{H}$ sur l'énumération (infinie) des éléments de $\mathcal{E}$ et qui soit au moins semi-calculable. Autrement dit, un algorithme nous permettant de retrouver, étant donnée une certaine partie finie de $\mathcal{E}$ en entrée, l'hypothèse qui la génère.

Pour les langages, et plus particulièrement pour les langages catégoriels, on prend pour cadres conceptuels des systèmes grammaticaux (en fait, une version moins générale de ceux-ci) dont l'espace d'hypothèses $\mathcal{H}$ est un ensemble ou classe GK de grammaires (dans un sens très général et à préciser – vid. *infra*) ; l'ensemble $\mathcal{E}$ d'échantillons un ensemble récursif $\subseteq \Sigma^*$ de phrases ou chaînes (de suites finies de mots, un mot étant vu sous la forme d'un symbole) sur un alphabet Σ ; et la fonction M devient la fonction L telle que pour chaque $G \in GK$, $L(G) \subseteq \Sigma^*$ – qui à toute grammaire associe le langage

qu'elle engendre. Par la suite, on travaillera avec des cadres ayant comme classe de grammaires GK, la classe des grammaires catégorielles.

Les grammaires catégorielles de GK sont des triplets $G = \langle \Sigma, \delta, S \rangle$ où Σ est un ensemble de mots ou lexique, δ une fonction d'assignation des mots (des éléments du lexique) aux types et S un type distingué, celui des phrases ou expressions bien formées du langage que l'on modélise. Comme système de types on aura $MINCLL$, la logique linéaire intuitionniste multiplicative sans constantes non conmutative[1]. Car, dans une phrase, l'ordre des syntagmes, des ces parties syntaxiques, est autrement très important. Nos types seront donc des formules linéaires et pour faire le *parsing* de la phrase il suffira d'en donner l'arbre de preuve, i.e. celui du séquent $A_1, ..., A_n \vdash S$[2], où $A_1, ..., A_n$ sont les types de ses n mots. L'un des avantages étant que la logique linéaire est une logique sensible aux ressources : on peut interpréter intuitivement l'implication linéaire $A \multimap B$ comme "l'action (tâche ou processus) denoté par B consomme la ressource denoté par A".

Dans le cadre usuel des grammaires catégorielles il y a, en outre, deux types principaux d'apprentissage : (**a**) à partir de suites (de mots ou caractères) (**b**) à partir de structures (où l'on a "enrichi" les suites avec des parenthèses et d'autres marquers syntaxiques). Or, il a été demontré dans [6] que les classes GCC_k, et GCC_{rigid}, celles des grammaires catégorielles classiques, respectivement, k-valuées (au plus k types assignés à chaque mot ou lexème) et rigides (au plus un type associé à chaque mot ou lexème) sont apprenables à partir des structures. Béchet et Foret dans [2] sont parvenus au meme résultat pour $GCNA_k$ et $CGNA_{rigid}$, les classes des grammaires catégorielles non-associatives k-valuées et rigides. D'où l'intrêt d'assortir chaque mot de l'alphabet de base avec des informations – des modules, car cela a pour conséquence des résultats d'apprenabilité non-triviaux – l'apprentissage à partir des suites étant souvent impossible (cf. [6] et [2]).

L'interêt des grammaires catégorielles vient du fait qu'elles ont le même pouvoir expressif que les grammaires indepédantes du contexte, c'est-à-dire des grammaires de la forme $\langle V, \Sigma, P, S \rangle$ ne contenant pas de production vide $A \to \epsilon$ (avec $A \in V$), et telles que toutes les productions de P sont de la forme $A \to \alpha$ avec $A \in V$ et $\alpha \in (V \cup \Sigma)^+$. Autrement dit, pour chaque

[1] i.e. le sous-système de MLL dont on a enlevé la règle d'échange à gauche et à droit et où l'on n'admet qu'a plus une seule formule à droite.

[2] Ou, de ce qui revient au même, de $A_1 \otimes ... \otimes A_n \vdash S$.

G catégorielle d'alphabet Σ il y a une grammaire hors contexte G' de même alphabet telle que pour chaque $s \in \Sigma^*$, $S \Rightarrow_G^* s$ exactement si $s \in L(G')$ – et réciproquement. Ce résultat a été prouvé par Pentus (cf. [6]) en servant de LC (le calcul de Lambek), système logique équivalent à $MINCLL$. Ce qui à son tour nous apprend les limitations des grammaires ayant $MINCLL$ comme système de catégories, car pour modéliser formellement la syntaxe des langues naturelles on a besoin de monter jusqu'aux grammaires de type 1 (dépendantes du contexte) et 0 même dans la hiérarchie de Chomsky (alors que les grammaires hors contexte sont de type 2).

Les informations seront (dans le cadre de ce travail) des modules, des graphes issus de la théorie des réseaux et des structures de preuve pour la logique linéaire. Le réseau ou structure devenant le *parse* de la phrase. La fonction ϕ aura ainsi comme entrées des mots annotés par des modules et pour sortie une grammaire de modules sur laquelle elle convergera au sens de Gold.

Ce rapport se structure de la façon suivante. Le **Chapitre 1** s'occuppe des défintions de base : on y voit ce que c'est, dans la cas général, une grammaire ainsi qu'un système grammatical ; ensuite, on expose ce que c'est que être apprenable dans ce cadre. Le **Chapitre 2** s'attarde sur les systèmes de types nécessaires pour définir les grammaires catégorielles. Le **Chapitre 3** est consacrée aux grammaires catégorielles usuelles. Le **Chapitre 3** porte sur l'application de la théorie des réseaux de preuve à la analyse syntaxique ; on y dit, par ailleurs, ce que c'est qu'un module. Enfin, dans le **Chapitre 5**, qui est la partie à proprement parler originale de ce travail, on définit les grammaires de modules ainsi que les grammaires de modules avec étiquettage syntaxique d'alpahabet fini et infini de même que leur système grammatical. On les applique ensuite les résultats du **Chapitre 1**. On remarque que l'on a veillé pour chaque partie soit aussi indépendante que possible des autres. Suivent les **Conclusions**. Quant à l'**Annexe A**, on y implémente dans LISP le formalisme de grammaires catégorielles classiques rigides.

Je tiens à remercier mon directeur, M. D. Béchet, pour tous ses conseils, sa patience et ses si précieuses remarques. Je remercie aussi à tous les autres membres du LIPN (Laboratoire d'Informatique de Paris Nord de l'université de Paris 13) au sein duquel ce rapport vit le jour et sans lesquels il ne l'aurait peut-être jamais fait.

Chapitre 1

Apprentissage grammatical

On commence avec un état de l'art de la théorie de l'apprentissage formel – en fait, du modèle de Gold restreint à des systèmes grammaticaux dont la classe de grammaires soit déjà semi-récursive (ce qui c'est le cas lorsqu'on a affaire à des grammaires catégorielles) et pour laquelle tous les langages de la classe de langages associée soient décidables par rapport au problème de l'appartenance. Ensuite, on parle de la propriété de densité finie bornée, définie par Shinohara dans [14] qui entraîne des résultats d'apprenabilité imposant des contraintes encore plus fortes sur les systèmes. Au demeurant, on se bornera aux conditions positives d'apprenabilité (en l'occurrence, à des conditions suffissantes), quoiqu'on citera néanmoins les conditions négatives les plus importantes (l'existence d'un point limite).

1.1 Grammaires et systèmes grammaticaux

On commence par quelques notions élmentaires sur les langages formels :

Définition 1.1.1 *On pose :*

1. Un alphabet*, noté* Σ*, est un ensemble fini non vide. Ses éléments sont appellés* mots *ou* symboles*.*

2. Σ^* *(resp.* Σ^+*) dénote l'ensemble des suites ou chaînes finies sur un alphabet* Σ *(resp. l'ensemble des suites finies non vides).* ϵ *dénote la chaîne vide.*

3. Un langage L *(quelconque) sur un alphabet* Σ *est une partie de son ensemble de suites finies – i.e.* $L \subseteq \Sigma^*$.

5. Une classe de langages*, notée* $\mathcal{L}$*, sur un alphabet* Σ *est un sous-ensemble de l'ensemble des parties de* Σ^* *– i.e.* $\mathcal{L} \subseteq \mathcal{P}(\Sigma^*)$.

4. $\forall s = m_1...m_k \in \Sigma^*$*, on appelle* m_i*, pour* $i \in [1, k]$, facteur *ou* lettre *de* s.

Et on passe ensuite aux grammaires et systèmes grammaticaux (en toute généralité) :

Définition 1.1.2 *Un* système grammatical *est un triplet* $GS = \langle\, S^*,\, GK,\, L \,\rangle$ *avec :*

1. $S^* \subseteq \Sigma^*$ *un ensemble récursif de suites finies de symboles (de chaînes de caractères) sur un alphabet* Σ*;*

2. GK *une classe d'objets finis sur* Σ*, dite* classe de grammaires *; on appelle* grammaire *tout* $\in GK$.

3. $L : GK \to \mathcal{P}(S^*)$.

Remarque 1.1.1

• *Intuitivement, une grammaire est un objet finitaire sur lequel ou avec lequel on peut faire des calculs (cf. [6]). Ainsi peut-on dire (informellement) qu'une machine de Turing est une grammaire, ou sinon qu'un automate fini l'est, ou encore qu'il s'agit d'une grammaire dans la hiérarchie de Chomsky. Les systèmes grammaticaux ne sont ainsi que des versions un tant soit peu plus faibles des cadres conceptuels à la Gold (cf. [6]).*

• *Dans la suite de ce travail, pour chaque classe quelconque* GK *de grammaires sur un alphabet* Σ *on notera* $\mathcal{L}(GK)$ *sa classe de langages associées – i.e.* $\mathcal{L}(GK) = \{L(G) \mid G \in GK\} \subseteq \mathcal{P}(\Sigma^*)$.

• *Pou faire de l'apprentissage il faut associer un certain* $n \in \mathbb{N}$*, que l'on note* $t(G)$ *et que l'on appelle* taille*. Cette notion variera fortement selon le*

système grammatical que l'on ait choisi comme cadre de base (vid. infra*). On la notera, néanmoins, toujours de la même manieère.*

• *Il faut munir chaque classe GK de grammaires d'un certain système GS d'un ordre que l'on notera toujours (et abusivement)* $\sqsubseteq$*. La défintion de cet ordre dépendra et de la classe et du système. Par exemple, pour les grammaires à la Chomsky (vid.* infra*) on le fera par le biais de l'inclusion de leurs alphabets et de leurs productions. Alors que pour les grammaires catégorielles (vid.***Chapitre 3***) on raisonnera plutôt sur leurs fonctions d'assignation.*

Définition 1.1.3 *Soient GK une classe de grammaires,* $G, G' \in GK$*. On dit que G est* équivalente *à G′ ou encore qu'elles ont le* même pouvoir expressif*, ce que l'on note* $G \equiv_L G'$*, ssi* $L(G) = L(G')$*.*

1.2 La théorie de l'apprentissage

Définition 1.2.1 *Soit* $GS = \langle S^*, GK, L\rangle$ *un système grammatical On appelle* algorithme d'apprentissage *(pour une classe GK de gramaires), noté* ϕ_{GK}*, tout algorithme tel que :*

1. D'abord :

$$\phi_{GK} : \bigcup_{i \in I\!N} (\mathcal{S}^i) \rightarrow GK.$$

2. $\forall L \in \mathcal{L}(GK)$ *avec* $L =< s_i >_{i \in \mathbb{N}}$ $\exists n_0 \in \mathbb{N}$ *et* $G_0 \in GK$ *tels que d'une part* $L = L(G_0)$ *et d'autre part* $\forall n > n_0$ *:*

• $\phi_{GK}(< s_1, ..., s_{n_0} >) = G_0$ *et*

• $\phi_{GK}(< s_1, ..., s_n >) = \phi_{GK}(< s_1, ..., s_{n_0} >)$.

Auquel cas, on dit, en plus, que ϕ_{GK} converge *vers* G_0 *sur* $< s_i >_{i \in \mathbb{N}}$.

Définition 1.2.2 *Soit GK une classe de grammaires. On dit que GK est* apprenable par identification à la limite à partir d'exemples positifs *ou* apprenable *tout court ssi* $\exists\ \phi_{GK}$ *un algorithme d'apprentisage pour GK.*

Le concept d'apprentissage ainsi défini répose sur celui de l'apprentissage de classes de langages par identification à la limite à partir d'exemples positifs dans le modèle de Gold. Dans ce cadre il s'agit de définir (de trouver) un algorithme (pas forcément total) $\phi_{\mathcal{L}}$ pour une classe de langages $\mathcal{L}$ r.e. qui permette de générer à terme chaque $L \in \mathcal{L}$, en prenant comme entrée un échantillon positif (une suite ou énumération finie d'expressions bien formées du langage à apprendre) $\mathcal{E} \subseteq L$, d'une taille bornée par un certain $n_0 \in \mathbb{N}$ – i.e. $\mathcal{E} =< s_0, ..., s_{n_0} >$. Ce qui implique que $\forall k \in \mathbb{N}$:

$$\phi_{\mathcal{L}}(< s_0, ..., s_{n_0} >) = \phi_{\mathcal{L}}(< s_0, ..., s_{n+k} >)$$

Intuitivement, ce que l'on modélise formellement est la capacité d'un agent humain de générer un nombre arbitrairement grand ou infini de phrases syntaxiquement correctes d'un langage donné après un nombre fini d'erreurs, comme les enfants lorsqu'ils apprennent leur langue maternelle – i.e. par éssai et erreur. Et cela par le biais d'un ensemble, lui aussi fini, de règles syntaxiques (une grammaire) que l'on "infère" ou "induit" en ce faisant.

Définition 1.2.3 *Une classe $\mathcal{L}$ une classe de langages sur un alphabet Σ a une* élasticité infinie *(*finie *sinon) ssi $\exists < s_i >_{i \in \mathbb{N}}$ et $\exists < L_i >_{i \in \mathbb{N}}$ avec $\forall i \in \mathbb{N}$, $L_i \in \mathcal{L}$, $s_i \in \Sigma^*$ et :*

1. $s_i \notin L_i$.

2. $\{s_0, ..., s_i\} \subseteq L_{i+1}$.

Théorème 1.2.1 *Soit $GS = \langle S^*, GK, L \rangle$ un système grammatical pour lequel le problème de l'appartenance est décidable* [1] *avec GK une classe de grammaires r.e. (i.e. récursivement énumerable ou semi-récursive). Alors, si $\mathcal{L}(GK)$ a une élasticité finie, GK est apprenable.*

Ce résultat, demontré dans [6], est très important, car il nous fournit une condition suffissante d'apprenabilité. C'est en fait sur ce théorème que l'on s'appuye pour prouver l'apprenabilité des grammaires de modules que l'on

[1] Pour lequel il existe, $\forall GK' \subseteq GK$ un algorithme ou procédure effective décidant le problème :
1. Donnée : $s \in \Sigma^*$, $G \in GK'$.
2. Question : $s \in L(G)$?

verra plus tard ; en fait, on démontrera une propriété plus forte : à savoir, la densité finie bornée du système grammatical associé (cf. Shinohara, [14]). Le résultat qui suit est tiré, lui aussi, de [6] :

Théorème 1.2.2 *Soient $\mathcal{L}$ et $\mathcal{L}'$ deux classes de langages sur des alphabets Σ et Σ', respectivement. Si $\mathcal{L}'$ a une élasticité finie et si $R \subseteq \Sigma^+ \times (\Sigma')^+$ est finiment valuée, i.e. si l'ensemble $\{s' \in (\Sigma')^+ \mid s'Rs\}$ est un ensemble fini, alors la classe $\{R^{-1}[L'] \mid L' \in \mathcal{L}'\}$ a, elle aussi, une élasticité finie.*

Définition 1.2.4 *Soit $G \in GK$. G est* réduite par rapport à $\mathcal{X} \subseteq S^*$ *ssi*

1. $\mathcal{X} \subseteq L(G)$.

2. $\forall G' \in GK$, $G' \sqsubset G \Rightarrow \mathcal{X} \not\subseteq L(G')$.

Définition 1.2.5 *Soit $GS = \langle S^*, GK, L \rangle$ un système grammatical. On dit que GS a une* densité finie bornée *ssi*

1. $\forall G, G' \in GK$, $G \sqsubseteq G' \Rightarrow L(G) \subseteq L(G')$. (i.e. L est une fonction monotone).

2. $\forall \mathcal{X} \subseteq S^*$, $\forall n \in \mathbb{N}$, $card\{L(G) \mid t(G) \leq n$, $G \in GK$ et G est réduite par rapport à $\mathcal{X}\}$ est fini.

Théorème 1.2.3 *Soit $GS = \langle S^*, GK, L \rangle$ un système grammatical. Si GS a une densité finie bornée, alors la classe $\{L(G) \mid G \in GK$ et $t(G) \leq n\}$ a une élasticité finie.*

Ce théorème s'ensuit de celui démontré par Shinohara dans [14]. Shinohara y souligne que son résultat est valable pour tout cadre conceptuel $\langle \mathcal{E}, \mathcal{H}, M \rangle$ – en anglais : *conceptual framework*, dont les systèmes grammaticaux (qui n'en sont qu'un cas particulier). Il en fait usage pour montrer que, par exemple, le système $GS_{dc} = \langle \Sigma^*, GDC, L \rangle$ des grammaires dépendantes du contexte [2] possède une épaisseur finie.

[2] vid. *infra* pour la définition.

Schématiquement, on a ainsi que :

épaisseur finie bornée ⇒ élasticité finie ⇒ apprenabilité

On finit cette section avec une condition négative d'apprenabilité, que l'on énonce même si l'on ne s'en servira pas par la suite : l'existence d'un point limite – i.e. d'un langage de la classe qui soit la réunion (arbitraire, donc infinie) d'une suite infinie des langages de la classe.

Définition 1.2.6 *Soit $\mathcal{L}$ une classe (arbitraire) de langages sur un alphabet Σ. On dit que $\mathcal{L}$ admet un* point limite *ssi $\exists L \in \mathcal{L}$, $\exists < L_i >_{i \in \mathbb{N}}$ tels que :*

1. $\forall i \in I\!N$, $L_i \in \mathcal{L}$;

2. $\forall i, j \in I\!N$, $i < j \Rightarrow L_i \subset L_j$;

3. et finalement :

$$L = \bigcup_{i \in \mathbb{N}} L_i.$$

Proposition 1.2.1 *Soit $\mathcal{L}$ une classe de langages (arbitraire et sur un alphabet Σ). Si $\mathcal{L}$ admet un point limite, alors elle n'est pas apprenable.*

La preuve de la proposition précédente se trouve (aussi) dans [6] et elle nous donne, enfin, le schéma que voici :

existence d'un point limite ⇒ non-apprenabilité ⇒ élasticité infinie

1.3 Quelques exemples

Pour illustrer quelques unes de ces notions, on donne en exemple les cadre usuel grammaires hors-contexte et dépendantes du contexte, auxquelles on ferra, en outre, allusion par la suite.

Définition 1.3.1 *Une* grammaire (à la Chomsky et sans production vide) *est un quadruplet $G = \langle \Sigma, V, S, P \rangle$ avec :*

1. Σ *un alphabet, dit des* terminaux*, et* V *un ensemble non vide aussi, dit ensemble des* variables*, tels que* $\Sigma \cap V = \emptyset$*.*

2. $S \in V$ *un symbole dit* axiome *ou* symbole initial*.*

3. $P \subseteq (\Sigma \cup V)^+ \times (\Sigma \cup V)^+$ *un ensemble fini de* productions *de la forme* $\alpha \to \beta$ *avec* $\alpha \in (\Sigma \cup V)^+$*, dit la* tête *de la production, et* $\beta \in (\Sigma \cup V)^+$*, dit le* corps *de la production. Par ailleurs, si* $\beta \in \Sigma$ *on parle de* production terminale*.*

Définition 1.3.2 *On dit que* G dérive en une étape *la chaîne* $\beta \in (\Sigma \cup V)^+$ *en partant de* $\alpha \in (\Sigma \cup V)^+$*, ce que l'on écrit* $\alpha \Rightarrow_G \beta$ *ssi* $\alpha \to \beta \in P$*. On note* $\Rightarrow_G^*$ *la clôture réflexive et transitive de cette relation. On a ainsi, en particulier, que* $L(G) = \{s \in \Sigma^+ \mid S \Rightarrow_G^* s\}$*.*

Définition 1.3.3 *L'ensemble* P *des productions d'une grammaire* G *étant fini, on définit la* taille $t(G)$ *de* G *comme* $t(G) = card(P)$*.*

Définition 1.3.4 *Soient* G, G' *deux grammaires, d'alphabets* Σ *et* Σ' *et de productions* P *et P', respectivement. Alors* $G \sqsubseteq G'$ *ssi* $\Sigma \subseteq \Sigma'$ *et* $P \subseteq P'$*. On note* $\sqsubset$ *l'ordre strict associé.*

Remarque 1.3.1

• *En imposant des contraintes à l'ensemble* P *de productions on peut définir ce que l'on appelle les* types *des grammaires : le type 0 (grammaires r.e. ou récursives), le type 1 (grammaires sensibles au contexte), le type 2 (grammaires hors contexte) et, le type 3 (grammaires régulières) – la hiérarchie de Chomsky. Les grammaires catégorielles sont pour leur part, une variante spéciale de ce formalisme comme on le verra par la suite.*

On définit maintenant les grammaires hors-contexte et les grammaires dépendantes du contexte :

Définition 1.3.5 *On pose :*

1. On dit qu'un grammaire G est hors-contexte *ssi G est une grammaire dont toutes les productions sont de la forme $A \rightarrow \alpha$, avec $A \in V$ et $\alpha \in (V \cup \Sigma)^+$. On note GHC leur classe.*

2.On dit qu'un grammaire G est dépendante du contexte *ssi G est une grammaire dont toutes les productions sont de la forme $\alpha \rightarrow \beta$ avec $\alpha, \beta \in (V \cup \Sigma)^+$. On note GDC leur classe.*

Exemple 1.3.1

(α) Soit $G_{reg} = \langle \{a\}, \{S\}, S, \{S \rightarrow a\} \rangle$. $L(G) = \{a\}$. La taille de G_{reg} est $t(G_{reg}) = 1$.

(β) Le système grammatical associé aux grammaires indépendantes du contexte est le triplet $G_{hc} = \langle \Sigma^*, GHC, L \rangle$.

(γ) Le système grammatical associé aux grammaires dépendantes du contexte est le triplet $GC_{dc} = \langle \Sigma^*, GDC, L \rangle$. Il possède une densité finie bornée – ce qui implique que la classe $\{L(G) \mid G \in GDC \text{ et } t(G) \leq n\}$ possède une élasiticité finie $\forall n \in \mathbb{N}$ (cf.[14]).

Chapitre 2

Les systèmes de types

Comme on l'a remarqué dans l'introduction, les grammaires catégorielles se servent d'un système de dérivation pour effectuer l'analyse syntaxique. En fait, ce sont des grammaire lexicaliseés comportant un lexique assorti d'une fonction qui associe à chacun de ses éléments un ensemble fini (éventuellement vide) de types. Ensuite, pour faire l'analyse syntaxique d'une phrase, on raisonne sur les types associés aux lexèmes les composant – en l'occurence, on cherche une preuve. Avant de passer à leur définition il faut que l'on s'attarde sur les systèmes de types, relevant tous de la logique linéaire, et que l'on énonce leurs propriétés les plus importantes.

2.1 Les systèmes : LC et $MINCLL$

Définition 2.1.1 *Soit $\mathcal{A} = \{X_i \mid i \in \mathbb{N}\}$ un ensemble dénombrable (dit des* atomes*). L'ensemble des* types*, noté $\mathcal{T}$, est défini par la grammaire :*

$$\mathcal{T} = \mathcal{A} \mid \mathcal{T} \multimap \mathcal{T} \mid \mathcal{T} \circ\!\!- \mathcal{T} \mid \mathcal{T} \otimes \mathcal{T}.$$

Définition 2.1.2 *Un* séquent linéaire intuitionniste *est une expression formelle $\Gamma \vdash A$, où Γ est une suite ou séquence finie (de formules ou types) et A une formule (ou type), et $\vdash$ un symbole se lisant "these". La partie à droite de $\vdash$ est dite le* contexte droite *du séquent et la partie à gauche, son* contexte gauche.

Définition 2.1.3 *On définit,* $\forall A \in \mathcal{T}$ *le* rang *de* A*, noté* $r(A)$*, par induction comme suit :*

1. $r(X) = 0,\ \forall X \in \mathcal{A}$.
2. $r(A \otimes B) = max\{r(A), r(B)\} + 1$.
3. $r(A \multimap B) = max\{r(A), r(B)\} + 1$.
4. $r(A \circ\!\!- B) = max\{r(A), r(B)\} + 1$.

Définition 2.1.4 *MINCLL est le système défini par les règles suivantes*[1] *:*

1. *Axiome :*

$$\frac{}{A \vdash A}\, ax$$

2. *Cut :*

$$\frac{\Gamma, A, \Gamma' \vdash B \qquad \Delta \vdash A}{\Gamma, \Delta, \Gamma' \vdash B}\, cut$$

3. $\otimes^g$ *:*

$$\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, A \otimes B, \Gamma' \vdash C}$$

4. $\otimes^d$ *:*

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\, \otimes^d$$

5. $\multimap^g$ *:*

$$\frac{\Gamma, B, \Gamma' \vdash C \qquad \Delta \vdash A}{\Gamma, \Delta, A \multimap B, \Gamma' \vdash C}\, \multimap^g$$

6. $\circ\!\!-^g$ *:*

$$\frac{\Gamma, A, \Gamma' \vdash C \qquad \Delta \vdash B}{\Gamma, A \circ\!\!- B, \Delta, \Gamma' \vdash C}\, \circ\!\!-^g$$

[1] cf. [7] et [9].

7. $\circ\!\!-^d$ *:*

$$\frac{\Gamma, B \vdash A}{\Gamma \vdash A \circ\!\!- B} \circ\!\!-^d$$

8. $-\!\!\circ^d$ *:*

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A -\!\!\circ B} -\!\!\circ^d$$

9. On appelle 1. et 2. règles structurelles *(puisqu'elles ne portent sur aucun connecteur) et les autres,* règles logiques*.*

10. Pour chaque règle r*, on appelle les séquents au-dessus de la barre les* prémisses *de la règle et le séquent d'au-dessous, sa* conclusion*.*

11. Les notions de preuve*, de* prouvabilité *et de* théorème *sont définies de la manière usuelle à partir des règles ci-dessus.*

Exemple 2.1.1

Voici un exemple de preuve formelle ou de dérivation $\pi \in MINCLL$ (où A, B et C sont des méta-variables qui répresentent des formules quelconques, comme dans l'énoncé des règles) :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\mathrm{ax} \quad \dfrac{\dfrac{}{B \vdash B}\,\mathrm{ax} \quad \dfrac{}{C \vdash C}\,\mathrm{ax}}{B, B -\!\!\circ C \vdash C} -\!\!\circ^g}{A -\!\!\circ B, B -\!\!\circ C, A \vdash C} -\!\!\circ^g}{A -\!\!\circ B, B -\!\!\circ C \vdash A -\!\!\circ C} -\!\!\circ^d}{(A -\!\!\circ B) \otimes (B -\!\!\circ C) \vdash A -\!\!\circ C} \otimes^g$$

Dans la literature, on se sert plutôt du calcul de Lambek, noté LC, comme système de types. Mais ce calcul-ci s'avère équivalent à $MINCLL$ (ils sont le même pouvoir expressif) *modulo* une certaine traduction *, lorsqu'on ajoute la contrainte que le contexte gauche des séquents ne soit pas vide. Ce qui veut dire que la logique linéaire constitue un tout aussi bon départ pour définir une grammaire catégorielle que LC.

Définition 2.1.5 *L'ensemble de types associé au calcul de lambek LC (i.e. son langage) est l'ensemble $\mathcal{T}'$ défini par la grammaire :*

$$\mathcal{T}' = \mathcal{A} \mid \mathcal{T}'\backslash\mathcal{T}' \mid \mathcal{T}'/\mathcal{T}' \mid \mathcal{T}' \bullet \mathcal{T}'.$$

$\mathcal{A}$ étant le même ensemble d'atomes que pour $\mathcal{T}$. Bien évidémment, la définition du rang d'un type s'adapte facilement à ce système.

Définition 2.1.6 *LC est défini comme suit :*

1. *Axiome :*

$$\frac{}{A \vdash A}\, ax$$

2. *Cut :*

$$\frac{\Gamma, A, \Gamma' \vdash B \qquad \Delta \vdash A}{\Gamma, \Delta, \Gamma' \vdash B}\, cut$$

3. $\bullet^g$ *:*

$$\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, A \bullet B, \Gamma' \vdash C}\, \bullet^g$$

4. $\bullet^d$ *:*

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \bullet B}\, \bullet^d$$

5. $\backslash^g$ *:*

$$\frac{\Gamma, B, \Gamma' \vdash C \qquad \Delta \vdash A}{\Gamma, \Delta, A\backslash B, \Gamma' \vdash C}\, \backslash^g$$

6. $/^g$ *:*

$$\frac{\Gamma, A, \Gamma' \vdash C \qquad \Delta \vdash B}{\Gamma, A/B, \Delta, \Gamma' \vdash C}\, /^g$$

7. $\backslash^d$ *:*

$$\frac{\Gamma, B \vdash A}{\Gamma \vdash A\backslash B}\, \backslash^d$$

8. $/^d$ *:*

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A/B}\, /^d$$

Définition 2.1.7 *On définit la* taille *d'une preuve* π*, notée* $\tau(\pi)$*, comme suit :*

$$\tau(\pi) = \begin{cases} 0 & \text{si } \pi \text{ est un axiome ;} \\ max\{\tau(\pi') \mid \pi' \text{ est une sous-preuve de } \pi\} & \text{sinon.} \end{cases}$$

Définition, par ailleurs, valable pour les deux systèmes.

Définition 2.1.8 *On définit* $^\star : \mathcal{T} \to \mathcal{T}'$ *par induction sur les types (ou formules) comme suit :*

1. $X^\star = X, \forall X \in \mathcal{A}$.

2. $(A \multimap B)^\star = (A^\star \backslash B^\star)$.

3. $(A \circ\!\!- B)^\star = (A^\star / B^\star)$.

4. $(A \otimes B)^\star = (A^\star \bullet B^\star)$.

6. On l'étend aux suites de types, aux séquents et aux preuves.

6. * *est une bijection.*

Théorème 2.1.1 *Le séquent* $\Gamma \vdash A$*, avec* $\Gamma \neq \emptyset$*, est démontrable dans* $MINCLL$ *ssi le séquent* $\Gamma^\star \vdash A^\star$ *l'est dans* LC*.*

(**Preuve**) Par récurrence sur la taille $\tau(\pi)$ des dérivations.

($\Rightarrow$)

(**i**) $\tau(\pi) = 0$. π est un axiome. On prend alors $\pi^\star =$

$$\frac{}{A^\star \vdash A^\star}\,\mathrm{ax}$$

(**ii**) $\tau(\pi) = k+1$. (On se limite à un seul cas, les autres étant analogues). Soit $\pi \in MINCLL$ s'achevant par $-\circ^d$:

$$\frac{\begin{array}{c}\pi' \\ \vdots \\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A -\circ B}\,-\circ^d$$

Ce qui donne, par hypothèse de récurrence sur π' :

$$\frac{\begin{array}{c}\pi'^\star \\ \vdots \\ \Gamma^\star, A^\star \vdash B^\star\end{array}}{\Gamma^\star \vdash A^\star \backslash B^\star}\,\backslash^d$$

($\Leftarrow$)

(**i**) $\tau(\pi) = 0$. π est un axiome. On prend alors $\pi^{\star -1} =$

$$\frac{\dfrac{}{A^{\star -1} \vdash A^{\star -1}}\,\mathrm{ax}}{A \vdash A}\,df$$

Car * est une bijection et l'image inverse existe donc toujours.

(**ii**) $\tau(\pi) = k+1$. (On se limite à un seul cas, les autres étant analogues). Soit $\pi \in KL$ s'achevant par $\backslash^g$:

$$\frac{\begin{array}{cc}\pi_1 & \pi_2 \\ \vdots & \vdots \\ \Gamma, B, \Gamma' \vdash C & \Delta \vdash A\end{array}}{\Gamma, \Delta, A\backslash B, \Gamma' \vdash C}\,\backslash^g$$

Ce qui donne, par hypothèse de récurrence sur π_1 et π_2 :

$$\dfrac{\dfrac{\begin{array}{c}\pi_1^{\star-1}\\ \vdots\\ \Gamma^{\star-1}, B^{\star-1}, \Gamma'^{\star-1} \vdash C^{\star-1}\end{array}}{\Gamma, B, \Gamma' \vdash C}\, df \qquad \dfrac{\begin{array}{c}\pi_2^{\star-1}\\ \vdots\\ \Delta^{\star-1} \vdash A^{\star-1}\end{array}}{\Delta \vdash A}\, df}{\Gamma, \Delta, A \multimap B, \Gamma' \vdash C}\, \multimap^g$$

Ce qui suffit. **C.Q.F.D.**

2.2 Leurs propriétés

Lemme 2.2.1 *MINCLL admet l'élimination des coupures.*

Lambek, dans [8], l'ayant montré pour LC, on voit que, de par le **Théorème 2.1.1**, il en va de même pour $MINCLL$. Il s'agit d'une propriété assez importante et utile. Car elle nous apprend que pour chaque $\pi \in MINCLL$ d'un séquent $\Gamma \vdash A$, il existe $\pi_{norm} \in MINCLL$ du même séquent, π_{norm} étant la preuve sans coupures (i.e. où la règle *cut* a été enlevée) réduite obtenue à partir de π au bout d'un nombre fini étapes de normalisation. Une forme normale qui avec la propriété de la sous-formule énoncée ci-dessous nous permet, par inspection de tous les cas possibles, de montrer qu'il y des séquents $\Gamma \vdash A$ n'ayant pas de preuve possible.

Proposition 2.2.1 *Toute formule ayant une occurrence dans une preuve $\pi \in MINCLL$ sans coupures d'un séquent $\Gamma \vdash A$, est soit une sous-formule de A, soit une sous-formule de Γ.*

On appelle la propriété énoncée ci-dessus la *propriété de la sous-formule.* Et elle nous aide à vérifier que ce système est non-commutatif, ce qui découle, en gros, de l'abscence de la règle structurelle d'échange. Car toute preuve

admettant une forme normale (i.e. sans coupures), satisfait la propriété de la sous-formule.

Théorème 2.2.1 *Dans MINCLL les connecteurs ne sont pas commutatifs.*

(**Ebauche de preuve**) Pour s'en convaincre il suffit de regarder ce qui se passe pour $\otimes$ ($\multimap$ et $\circ\!\!-$ étant, en soi, non-commutatifs) [2]. On prend deux atomes X et Y et l'on montre qu'il n'y a pas de preuve formelle de $X \otimes Y \vdash Y \otimes X$. Sinon, il existe une preuve sans coupures (**Lemme 2.2.1**) π_{norm}, satisfaisant en outre la propriété de la sous-formule (**Proposition 2.2.1**), ainsi que toutes ses sous-preuves. Cette preuve π_{norm} s'achêve soit par $\otimes^g$:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ X, Y \vdash Y\end{array}}{X \otimes Y \vdash Y}\otimes^g \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash X\end{array}}{X \otimes Y \vdash Y \otimes X}\otimes^d$$

Soit par $\otimes^g$:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ X \vdash Y\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ Y \vdash X\end{array}}{X, Y \vdash Y \otimes X}\otimes^d}{X \otimes Y \vdash Y \otimes X}\otimes^g$$

π_{norm} ne peut donc avoir, par hypothèse de normalité, que l'une des deux formes ci-dessus, à la symétrie près[3]. Mais, hélas, les séquents hypothèses auxquels on arrive, et *a fortiori* les sous-preuves π_i, pour $i \in \{1, 2\}$, dont ils sont les feuilles, sont, dans tous les cas, impossibles, ce qui conclut. **C.Q.F.D.**

[2] Ou, si l'on veut, pour $\bullet$, $\backslash$ et $/$ respectivement.

[3] On omet les deux cas symétriques puisque le raisonnment est le même.

L'aspect non-commutatif de $MINCLL$ est une propriété importante (et, du même coup de LC, du calcul de Lambek) de la logique étudiée, car l'ordre des types dans le contexte Γ d'un séquent $\Gamma \vdash A$, doit réfléter celui des syntagmes de la phrase. Un ordre essentiel pour, par exemple, les langues néolatines (et même pour toutes les langues encore vivantes de la famille indoeuropéenne) – c'est donc une contrainte essentielle.

Remarque 2.2.1

- *Dans LC on aurait écrit le séquent ci-dessus* $X \bullet Y \vdash Y \bullet X$.

Chapitre 3

Les grammaires catégorielles

Le formalisme des grammaires catégorielles constitue dans son ensemble une autre façon de définir le formalisme des grammaires tel qu'il a été esquissé au **Chapitre 1** (les grammaires à la Chomsky). En particulier le formalisme des grammaires catégorielles classiques est équivalent à celui les grammaires hors-contexte (cf. [6]). L'avantage étant qu'elles nous permettent de sortir du cadre habituel des systèmes de réecriture en faisant appel à la théorie de la preuve. L'ajout d'informations, par exemple par le biais d'un langage enrichi avec des structures (cf. [6]), des types ou des modules (voir *infra*), peut rendre compte des extensions (beaucoup plus complexes) au modèle des grammaires formelles tout court (grammaires de traits, etc.) et *a fortiori* jeter une lumière sur leur apprenabilité.

Les grammaires catégorielles sont des grammaires lexicalisées, où à chaque symbole $m \in \Sigma$ (de leur alphabet) on associe un ou plusieurs symboles tirés d'un autre ensemble $\mathcal{T}$ dit des *types* censées nous informer sur la catégorie syntaxique (la partie de la phrase) à laquelle m appartient. Ensuite, pour vérifier si une suite $m_1...m_k \in \Sigma^+$ est bien formée on se sert d'un calcul logique (traditionellement LC, le calcul de Lambek, ou la logique linéaire) admettant l'élimination des coupures pour déterminer s'il existe un arbre de preuve π ayant le type distingué S étiquettant sa racine et comme étiquettes de ses feuilles les types $A_1, ...A_n$ associés à $m_1, ..., m_n$. Ainsi les appelle-t-on dans les pays anglophones des *proof-theoretical grammars*. Certains auteurs (cf. [6]) les définissent comme une relation finie $\subseteq \Sigma \times \mathcal{P}_f(\mathcal{T})$. Quoi qu'il en soit, l'une des propriétés les plus remarquables est de constituer un formalisme équivalent aux grammaires hors contexte (ou de type 1) $\epsilon - free$, comme on l'affirmé dans l'introduction. En effet, *modulo* cette équivalence

de pouvoir expressif (vid. *infra*), on peut songer à adapter les définitions du **Chapitre 1** afin d'appliquer les résultats que l'on y énonce – telle, par exemple, la notion de taille.

Définition 3.0.1 *Une* grammaire catégorielle *est un triplet* $G = \langle \Sigma, \delta, S \rangle$ *avec :*

1. Σ *un alphabet non vide dit des* mots *ou* lexèmes.

2. $\delta : \Sigma \to \mathcal{P}_f(\mathcal{T})$ *une fonction finie des lexèmes vers les types.*

3. $S \in \mathcal{A}$ *(le type des expressions bien formées).*

4. *Si* $A \in \delta(m)$ *pour* $m \in \Sigma$, $A \in \mathcal{T}$, *on écrit* $G : m \mapsto A$.

Définition 3.0.2 *On définit la* taille *t(G) d'une grammaire catégorielle G comme suit :*

$$t(G) = \sum_{m \in \Sigma} \sum_{G : m \mapsto A} r(A) + 1$$

Définition 3.0.3 *Soient* $G, G' \in GC$ *d'alphabets* Σ, Σ' *et fonctions* δ, δ', *respectivement*[1]. $G \sqsubseteq G'$ *ssi* $\Sigma \subseteq \Sigma'$ *et* $\forall m \in \Sigma$, $\delta(m) \subseteq \delta'(m)$. *On note* $\sqsubset$ *l'ordre strict associé.*

Remarque 3.0.2

• *Le système grammatical des grammaires catégorielles (tout court) est le système* $GS_{GC} = \langle \Sigma^*, GC, L \rangle$.

Définition 3.0.4 *On note GC la classe de grammaires définie sur un alphabet* Σ *donné et* GC_{rigid} *et* GC_k *les classes de grammaires où,* $\forall m \in \Sigma$, *on a, respectivement,* $card(\delta(m)) \leq 1$ *et* $card(\delta(m)) \leq k$.

[1] Le type distingué S reste le même pour toute la classe.

Définition 3.0.5 *Soit* $G = \langle\, \Sigma,\, R,\, S \,\rangle \in GC$. *Le* langage généré par G, *noté L(G), est l'ensemble des suites* $s = m_1 \dots m_n \in \Sigma^*$ *telles que :*

1. $\exists\, A_1, ..., A_n \in \mathcal{T}$ *tels que,* $\forall i \in [1, n]$, $A_i \in \delta(m_i)$ *; et*

2. *Le séquent* $A_1, ..., A_n \vdash S$ *est démontrable dans MINCLL.*

3. *On appelle* langage catégoriel *un tel langage.*

Définition 3.0.6 *On note* $\mathcal{L}(GC)$, $\mathcal{L}(GC_{rigid})$, $\mathcal{L}(GC_k)$ *les classes de langages associées.*

Exemple 3.0.1

On va commencer par un exemple élémentaire, celui d'un langage hors contexte tout simple qui peut être aussi generé par une grammaire catégorielle. Voici donc la grammaire $G_0 \in GC_2$ telle que $L(G_0) = \{a^n b^n \mid n \in \mathbb{N}^*\}$. Soit $X \in \mathcal{A}$. $G_0 = \langle\, \{a, b\},\, \delta_0,\, S \,\rangle$; δ_0 est determinée par les assignations :

$$a \mapsto X$$

$$a \mapsto X \circ\!\!- S$$

$$b \mapsto X \multimap\!\!\circ S$$

$t(G_0) = (1 + 2) + 1 = 5$. On peut montrer que la suite $aabb \in L(G)$. En voici la preuve $\pi \in MINCLL$ associée :

$$\dfrac{\dfrac{}{X \vdash X}\,\mathrm{ax} \qquad \dfrac{\dfrac{}{S \vdash S}\,\mathrm{ax} \qquad \dfrac{\dfrac{}{X \vdash X}\,\mathrm{ax} \qquad \dfrac{}{S \vdash S}\,\mathrm{ax}}{X, X \multimap\!\!\circ S \vdash S}\,\multimap\!\!\circ^g}{X \circ\!\!- S, S, X \multimap\!\!\circ S \vdash S}\,\circ\!\!-^g}{X \circ\!\!- S, X, X \multimap\!\!\circ S, X \multimap\!\!\circ S \vdash S}\,\multimap\!\!\circ^g$$

Exemple 3.0.2

Voici un exemple plus poussé, qui montre que les grammaires catégorielles, autrement connues comme *proof-theoretical grammars* réussissent bel et bien à capturer quelques traits (et *a fortiori* à modéliser) de la syntaxe du français. Soit la grammaire rigide $G_1 = \langle\, \Sigma_h,\, \delta_2,\, S \,\rangle$, avec δ_2 définie par :

$$chat \mapsto N$$

$$escalier \mapsto N$$

$$court \mapsto SN \multimap S$$

$$monte \mapsto (SN \multimap S) \circ\!\!- N$$

$$le \mapsto SN \circ\!\!- N$$

$$Jean \mapsto SN$$

Où N représente la catégorie syntaxique des substantifs, SN celle des syntagmes nominaux, $N \multimap SN$ celle des verbes intransitifs, $(SN \multimap S) \circ\!\!- SN$ celles des verbes transitifs et $SN \circ\!\!- N$ celle des articles. Des types qui, de par la non-commutativité de leurs connecteurs nous renseignent sur la positition de chaque partie du discours au sein de la phrase. Quant à sa taille, on a que $t(G_1) = 1 + 1 + 2 + 3 + 2 + 1 = 10$.

(α) Voici la preuve, *modulo* le typage, que la phrase *Le chat monte l'escalier* $\in L(G_1)$:

$$\dfrac{\dfrac{}{N \vdash N}\,\text{ax} \qquad \dfrac{\dfrac{}{SN \vdash SN}\,\text{ax} \qquad \dfrac{\dfrac{}{N \vdash N}\,\text{ax} \qquad \dfrac{\dfrac{}{SN \vdash SN}\,\text{ax} \qquad \dfrac{}{S \vdash S}\,\text{ax}}{SN, SN \multimap S \vdash S}\,\multimap^g}{SN \circ\!\!- N, N, SN \multimap S \vdash S}\,\multimap^g}{SN \circ\!\!- N, N, (SN \multimap S) \circ\!\!- SN, SN \vdash S}\,\circ\!\!-^g}{SN \circ\!\!- SN, N, (SN \multimap S) \circ\!\!- SN, SN \circ\!\!- N \vdash S}\,\circ\!\!-^g$$

(β) Ou que *Le chat noir monte l'escalier* $\in L(G_1)$ [2] :

$$\dfrac{\dfrac{\dfrac{}{N \vdash N}\,\text{ax} \quad \dfrac{}{N \vdash N}\,\text{ax}}{N, N \multimap N \vdash N}\,\multimap^g \qquad \dfrac{\dfrac{}{SN \circ\!\!- SN, N, (SN \multimap S) \circ\!\!- SN, SN \circ\!\!- N \vdash S}\,\text{Thm}}{SN \circ\!\!- SN, N, N \multimap N, (SN \multimap S) \circ\!\!- SN, SN \circ\!\!- N \vdash S}\,\text{cut}$$

(γ) Ou que *Jean court* $\in L(G_1)$:

$$\dfrac{\dfrac{}{SN \vdash SN}\,\text{ax} \quad \dfrac{}{S \vdash S}\,\text{ax}}{SN, SN \multimap S \vdash S}\,\multimap^g$$

Pour finir, on énonce l'équivalence entre les grammaires catégorielles classiques et les grammaires hors contexte. On rappelle que le grammaires catégorielles classiques sont les grammaires catégorielles dont le systme de types se réduit au fragment implicatif gauche de $MINCLL$ (i.e. au système $MINCLL_{\{\multimap^g, \circ\!\!-^g\}}$). On note GCC cette classe. La preuve se trouve dans [6] :

Proposition 3.0.2 *$\forall L \subseteq \Sigma^*$, $\exists G \in GCC$ telle que $L(G) = L$ ssi L est un langage hors-contexte $\epsilon - free$.*

[2] La ligne double indique que l'on a appliqué plusieurs règles d'un seul coup.

Chapitre 4

Les réseaux de preuve et les modules

Dans cette section on va regarder un peu les applications de la théorie des structures et réseaux de preuve aux grammaires catégorielles. En effet, comme le système de types n'est qu'un sous-système de la logique linéaire (LL) ou encore de la logique linéaire multiplicative (MLL), cette application est envisageable. Qui plus est, elle ne change rien à la définition des grammaires catégorielles. Bien au contraire, il facilite le processus de recherche de preuve. Car cette procédure réduit le nombre de preuves par séquent, alors que l'on peut envisager, dans le cadre, par exemple de MLL, de MLL^{-}, le sous-système de MLL sans constante (i.e. sans 1 ni $\perp$), ou encore de $MELL$ (la logique linéaire multiplicative avec exponentiels et constantes), une infinité de preuves, même normales (car l'élimination de coupures ou normalisation n'est pas confluente). C'est donc comme un sorte de syntaxe graphique des preuves. Un réseau est un graphe (une strucure de preuve construite petit à petit en suivant le *parse* des types du séquent dont on cherche la preuve) dont on peut en extraire des arbres de démonstration (en gros : séquentialiser). Et il va sans dire que l'élimination de coupures s'y applique aussi (et que la séquentialisation demeure stable par cette opération sur les structures). Aussi un réseau nous fournit-il, par rapport à une grammaire $G \in GC$ l'analyse syntaxique des phrases engendrées par G. Dans ce chapitre on suit de près Rétoré (cf. [12], [11], [9] et [10]).

4.1 Les structures et réseaux de preuve

Définition 4.1.1 *L'ensemble des types (ou formules) associé à MLL^-, la logique linéaire multiplicative sans constante, est donné par la grammaire que voici ($\mathcal{A}$ étant un ensemble dénombrable d'atomes) :*

$$\mathcal{T}'' = \mathcal{A} \mid \mathcal{A}^\perp \mid \mathcal{T}''\wp\mathcal{T}'' \mid \mathcal{T}'' \otimes \mathcal{T}''$$

Remarque 4.1.1

• *Avec ces nouveaux types, les implications peuvent être introduites moyennant des définitions éliminatives :*

$$A \multimap B = A^\perp \wp B,\ A \circ\!\!- B = A \wp B^\perp.$$

Définition 4.1.2 *Un séquent linéaire monolatère (i.e. pour MLL^-, la logique linéaire multiplicative monolatère sans constante) est une expression de la forme $\vdash A_1, ..., A_n$, les A_i, pour $i \in [1, n]$ étant des types appartennant à $\mathcal{T}''$.*

Définition 4.1.3 *MLL^- La logique linéaire multiplicative monolatère sans constantes est définie par les règles que voici :*

1. *Axiome :*

$$\frac{}{\vdash A^\perp, A}\, ax$$

2. *Cut :*

$$\frac{\vdash \Gamma, A^\perp \qquad \vdash A, \Delta}{\vdash \Gamma, \Delta}\, cut$$

3. *Echange :*

$$\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}\, ech$$

4. *$\otimes$:*

$$\frac{\vdash \Gamma, A, \Delta \qquad \vdash \Gamma', B, \Delta'}{\vdash \Gamma, \Gamma', A \otimes B, \Delta, \Delta'}\, \otimes$$

5. $\wp$ *:*

$$\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, A \wp B, \Delta}\ \wp$$

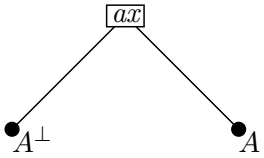
Remarque 4.1.2

• *La négation est involutive et définie* a priori *sur les atomes, mais on peut l'étendre aux formules (ou types) en posant* $A^{\perp\perp} = A$, $(A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp}$ *et* $(A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp}$. *Il en découlent les lois de De Morgan. Du reste, le calcul ci-dessus est,* modulo *la négation (ou orthogonalité) équivalent à sa version bilatère.*

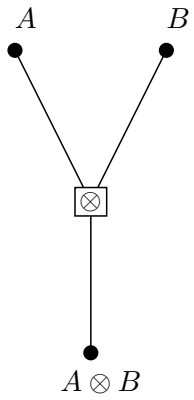
Définition 4.1.4 *L'ensemble* $\mathcal{SP}$ *des* structures de preuve *est l'ensemble des graphes construits à partir des graphes (étiquettés) suivants appellés* liens *:*

1. Lien hypothèse :

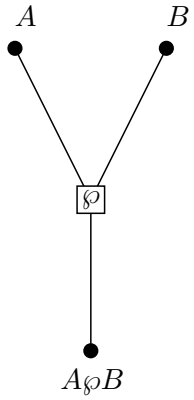
2. Lien axiome :



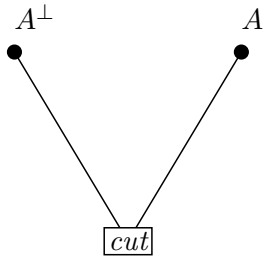
3. Lien tenseur :



4. Lien par :



5. Lien cut :



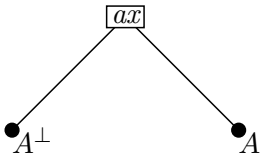
6. On appelle les sommets au dessus du sommet étiquetté ax, $\otimes$, $\wp$ *ou cut, les* prémisses *ou* entrées *du lien et celui d'au dessous, ses* conclusions *ou* sorties. *– un lien hypothèse étant hypothèse et conclusion de lui même. De plus, on dit qu'une hypothèse est* libre *si jamais elle n'est pas la conclusion d'un autre lien –* liée *sinon.*

7. Une structure de preuve *est un graphe constitué des liens 1 à 5 ne comportant pas (pour aucun de ses liens) d'hypothèses libres.*

Les structures de preuve sont construites petit à petit en suivant l'arbre d'analyse des types que l'on retrouve dans un séquent $\vdash A_1, ..., A_n$. L'intérêt étant que si ce séquent admet une preuve (dans MLL^-), alors on aboutira à une structure qui sera un réseau : une structure séquentialisable, d'où, en la déconstruisant, ressort un arbre de démonstration.

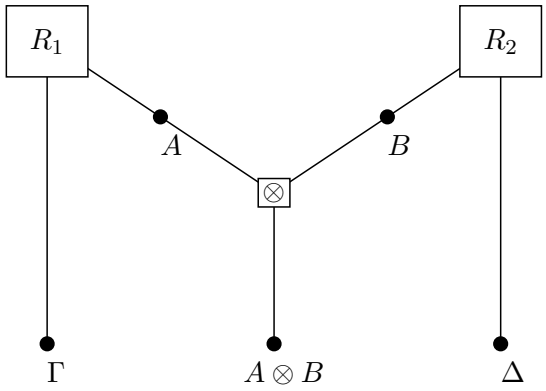
Définition 4.1.5 *L'ensemble des* réseaux de preuve*, noté $\mathcal{RP}$, est défini par induction structurelle comme suit :*

1. Base :

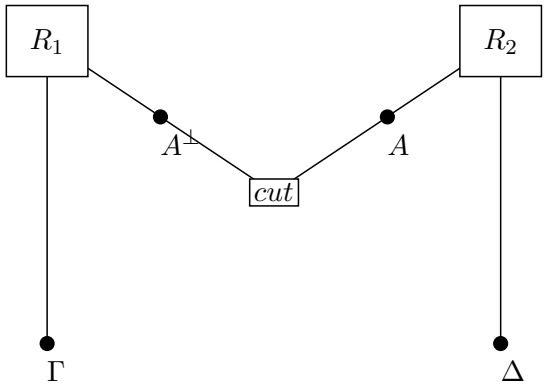


Est le réseau de preuve associé au séquent $\vdash A^\perp, A$*, démontrable* via *un axiome.*

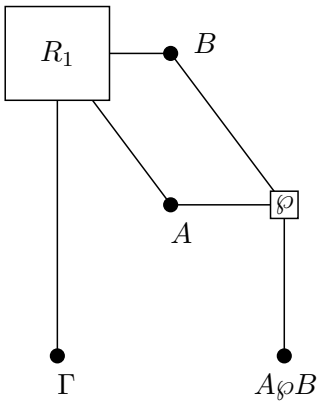
2. Si R_1 et R_2 sont des réseaux de preuves pour les séquents $\vdash \Gamma, A$ et $\vdash B, \Delta$ respectivement, il en est de même pour :



3. Si R_1 et R_2 sont des réseaux de preuves pour les séquents $\vdash \Gamma, A^\perp$ et $\vdash A, \Delta$ respectivement, il en est de même pour :



4. Si R_1 est un réseau de preuve pour le séquent $\vdash \Gamma, A, B$, alors il en est de même pour :



4. On appelle les sommets au dessus du sommet étiquetté ax, $\otimes$ *ou* $\wp$, *les* prémisses *ou* entrées *du sommet et celui d'au dessous, ses* conclusions *ou* sorties.

Un réseau est donc l'image d'une preuve π du calcul par une certaine traduction. Ce qui implique que, étant donnée une structure de preuve S s'averant être un réseau, on peut en extraire une arbre de preuve – séquentialiser :

Définition 4.1.6 *Une structure de preuve est* séquentialisable *ssi elle est l'image par une certaine traduction d'une preuve π de MLL^-.*

Les réseaux sont trivialement séquentialisables, et sont *a fortiori* des structures de preuve séquentialisables, puisqu'ils ont été définis en suivant les règles du calcul MLL^-. De même, une structure de preuve séquentialisable est un réseau. On a, en effet, l'équivalence :

Proposition 4.1.1 *Soit S une structure de preuve. S est un réseau de preuve ssi S est séquentialisable.*

Définition 4.1.7 *Un* graphe de correction *d'une struture de preuve S est un graphe non orienté obtenu en enlevant les étiquettes de S et en ne gardant qu'une des arêtes de chaque lien $\wp$ (en choisissant un* switch*).*

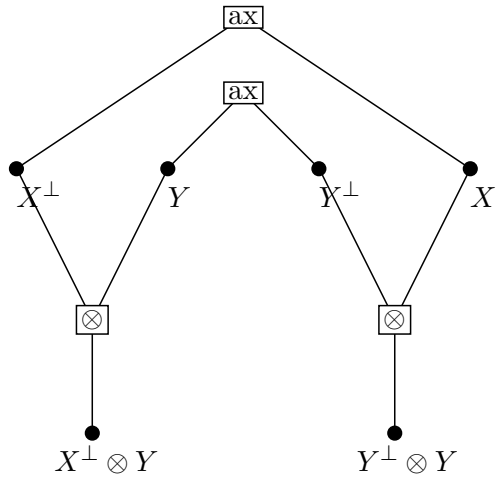
La propriété d'être séquentialisable est équivalente (pour les structures de preuve) à celui du critère de correction de Danos-Régnier : que tous ses graphes de correction soient acycliques et connexes. Une structure S de preuve possède 2^m graphes de correction, où l'entier m dénote le nombre de ses sommets $\wp$. On les obtient *grosso modo* de la façon suivante : (**a**) On enlève toutes leurs étiquettes. (**b**) Si jamais on a affaire à des sommets étiquettés par un $\wp$, on construit deux graphes de correction distincts en opérant un choix (le *switch*), pour chacun (i.e. entre ses deux prémisses). Ainsi sait-on si S est ou n'est pas séquentialisable après un nombre potentiellement exponentiel d'étapes. Toutefois il existe d'autres critères qui permettent de réduire le nombre d'étapes à un polynôme du nombre de sommets de la structure S [1].

Proposition 4.1.2 *Soit S une structure de preuve. S est séquentialisable ou bien S est un réseau ssi tous ses graphes de correction sont acycliques et connexes.*

Proposition 4.1.3 $\mathcal{RP} \subset \mathcal{SP}$.

(**Preuve**) Le graphe suivant est une structure de preuve mais n'est pas pour autant un réseau, puisque son (seul) graphe de correction contient un cycle :

[1] Le critère de Girard, etudié dans [18].



C.Q.F.D

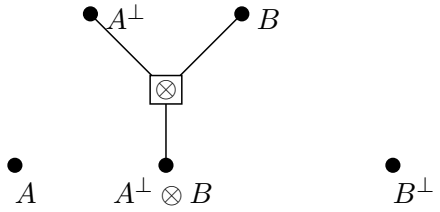
On résume donc les pas à suivre. Soit un séquent $\vdash A_1, ..., A_n$. (**a**) D'abord, on contruit les liens correspondant à chacun des types A_i en suivant leur arbre de parse – on obtient une forêt syntaxique. (**b**) Ensuite, on relie ces liens avec des liens axiome et des liens cut – si jamais on arrive à un graphe sans hypothèses libres, on a une structure de preuve. (**c**) Enfin, on regarde si la structure de preuve est séquentialisable en examinant ses graphes de correction – et cela pour toutes les structures de preuve possibles auxquelles on soit arrivé prélablement. Si l'on trouve un réseau, alors $\vdash A_1, ..., A_1$ possède une preuve.

Par ailleurs, il faut dire que le formalisme des réseaux admet l'élimination des coupures et que la séquentialisation demeure stable par normalisation.

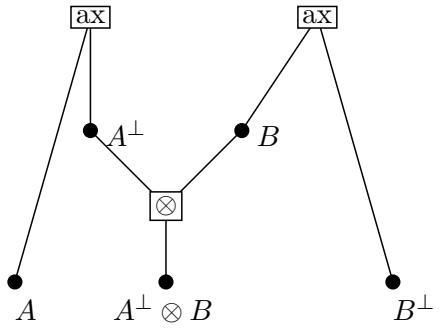
Exemple 4.1.1

Le séquent $\vdash A, A^\perp \otimes B, B^\perp$ est-il démontrable dans MLL^- ?

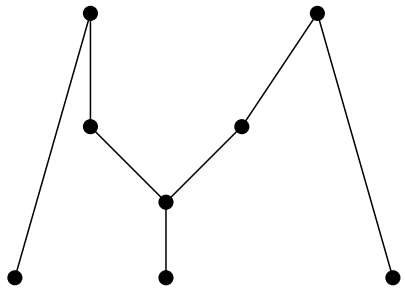
(α) On commence par se donner la forêt syntaxique (les arbres d' analyse de ses types ou formules) :



(β) Ensuite, on regarde si l'on peut construire une structure de preuve, en ajoutant des liens axiome. La voici :



(γ) Dont voici le (seul) graphe de correction :



Qui est acyclique et connexe. La structure est bien un réseau, Elle est séquentialisable. En effet, il lui correspond (aux échanges près) la preuve que voici :

$$\dfrac{\dfrac{}{\vdash A, A^{\perp}}\,\mathrm{ax} \qquad \dfrac{}{\vdash B, B^{\perp}}\,\mathrm{ax}}{\vdash A, A^{\perp} \otimes B, B^{\perp}}\,\otimes$$

Pour faire le lien avec $MINCLL$, le formalisme des réseaux doit être, bien entendu, adapté et étendu, du fait qu'il a été developpé pour le système MLL^{-} monolatère. On a d'autres connecteurs, $\wp$ (le dual de $\otimes$) et la négation linéaire ($^{\perp}$). Pour cela on se sert d'emblée d'une variante s'appuyant sur les formules (types) positives (les *outputs*) et négatives (les *inputs*) et où la notion de formule négative (resp. contexte négatif) remplace celle de formule orthogonale (resp. contexte orthogonal) – ce qui sert à simuler la négation linéaire. L'idée étant que un si un séquent $\Gamma \vdash A$ est prouvable dans $MINCLL$, alors il en sera de même pour $\vdash \Gamma^{-}, A^{+}$ dans MLL^{-} monolatère, ou encore, dans le système $IMLL^{-}$, (la logique linéaire intuitionniste multiplicative sans constantes, qui contient $MINCLL$), système équivalent à MLL^{-} restreint aux séquents $\vdash A_1, ..., A_n, B$ avec A_i, pour $i \in [1, n]$, négatif et B positif (cf.[9]). Ce qui impliquera l'existence d'un réseau de preuve. Enfin, pour obtenir la réciproque et capturer la non-commutatitivité de $MINCLL$ on introduira une contrainte pour les réseaux : leur planarité (pas de croisements entre leurs arêtes).

Définition 4.1.8 *Soit $\mathcal{A}$ un ensemble infini dénombrable d'atomes. On définit l'ensemble des* formules *ou* types négatifs *et* positifs*, (i.e.* inputs *et* outputs*) notés $\mathcal{N}$ et $\mathcal{P}$ simultanément par la grammaire :*

$$\mathcal{P} = \mathcal{A} \mid \mathcal{N} \wp \mathcal{P} \mid \mathcal{P} \wp \mathcal{N} \mid \mathcal{P} \otimes \mathcal{P}.$$

$$\mathcal{N} = \mathcal{A}^{\perp} \mid \mathcal{P} \otimes \mathcal{N} \mid \mathcal{N} \otimes \mathcal{P} \mid \mathcal{N} \wp \mathcal{N}.$$

Définition 4.1.9 *On définit les traductions $^{+} : \mathcal{T} \to \mathcal{P} \cup \mathcal{N}$ et $^{-} : \mathcal{T} \to \mathcal{P} \cup \mathcal{N}$ par induction simultanée comme suit :*

1. $X^{+} = X,\ X^{-} = X^{\perp}, \forall X \in \mathcal{A}$.

2. $(A \multimap B)^{-} = A^{-} \otimes B^{+}$.

3. $(A \multimap B)^- = A^+ \otimes B^-$.

4. $(A \circ\!\!- B)^+ = B^- \wp A^+$.

5. $(A \multimap B)^+ = B^+ \wp A^-$.

6. $(A \otimes B)^- = A^- \wp B^-$.

7. $(A \otimes B)^+ = B^+ \otimes A^+$.

8. On les étend aux suites de types (i.e. aux contextes).

Remarque 4.1.3

• *Les traductions* $^+$ *et* $^-$ *sont des bijections de* $\mathcal{T}$ *dans* $\mathcal{P}$ *et dans* $\mathcal{N}$*, respectivement (cf. [9]).*

• $\mathcal{N} \cup \mathcal{P} \subset \mathcal{T}''$ *– car,* $X \otimes Y \in \mathcal{T}''$ *mais* $X \otimes Y \notin \mathcal{P}$ *et* $X \otimes Y \notin \mathcal{N}$.

Définition 4.1.10 *On pose :*

1. Le système noté $MINCLL^+$ *est le système composé des règles de* $MINCLL$ *où l'on se permet des contextes vides à gauche et qui comprend la règle d'échange intuitionniste que voici :*

$$\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}\, ech$$

2. $IMLL^-$ *dénote la logique linéaire intuitionniste multiplicative sans constante.*

Voici donc les résultats justifiant l'usage des réseaux de preuve pour $MINCLL$.

Théorème 4.1.1 *Sont équivalents :*

(1) Le séquent $\Gamma \vdash A$ *est démontrable dans* $MINCLL^+$.

(2) $\Gamma \vdash A$ *est démontrable dans* $IMLL^-$.

(3) $\vdash \Gamma^-, A^+$ *est démontrable dans* MLL^-.

(4) Il existe un réseau de preuve R *pour* $\vdash \Gamma^-, A^+$.

(**Ebauche de preuve**)

((1) $\Longleftrightarrow$ (2)) cf. Rétoré, [9].

((2) $\Longleftrightarrow$ (3)) On remarque qu'un séquent $A_1, ..., A_n \vdash B$ avec $A_i \in \mathcal{N}$, pour $i \in [1, n]$, et $B \in \mathcal{N}$ est démontrable dans MLL^- exactement si $A_1, ..., A_1 \vdash B$ admet une preuve dans $IMLL^-$.

((3) $\Longleftrightarrow$ (4)) On sait qu'un séquent $\vdash \Gamma, \Delta$ (du langage de MLL^-) admet une preuve dans MLL^- exactement s'il possède un réseau. Il suffit à ce moment-là de remarquer que le séquent $\vdash \Gamma^-, A^+$ est ègal à un certain séquent $\vdash \Delta, B$ de MLL^- avec $\Delta = \Gamma^-$ et $B = A^+$. **C.Q.F.D.**

Le réseaux auxquels on a eu jusqu'à présent affaire ne réussissent pas encore à capturer la non-commutativité. Pour ce faire il faut demander en plus qu'ils soient planaires – i.e. qu'il n'y ait pas de croisement entre leurs arêtes. L'ensemble de ces réseaux de preuve (les réseaux planaires) coïncide exactement avec celui des preuves de $MINCLL$ (cf. [9], [12]) :

Théorème 4.1.2 $\Gamma \vdash A$ *est démontrable dans* $MINCLL$ *ssi il existe un réseau de preuve planaire* R *pour* $\vdash \Gamma^-, A^+$.

Exemple 4.1.2

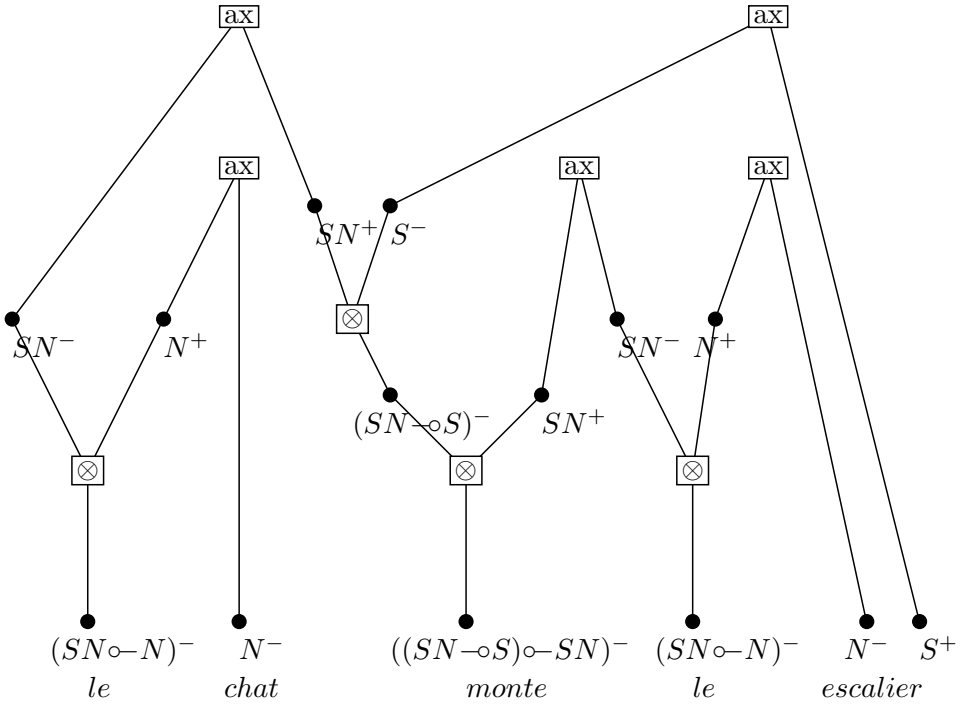
• Exemple (α) :

Voici un exemple de réseau de preuve (une structure de preuve S séquentialisable) montrant que *Le chat monte l'escalier* $\in L(G_1)$ – cf. l'**Exemple 3.0.1**. En effet, le séquent :

$$SN\circ\!\!-N, N, (SN\multimap S)\circ\!\!-SN, SN\circ\!\!-N, N \vdash S$$

Admet une preuve (*modulo* l'équivalence énoncée auparavant) dans le système $MINCLL$, car la structure suivante[2] est un réseau planaire pour :

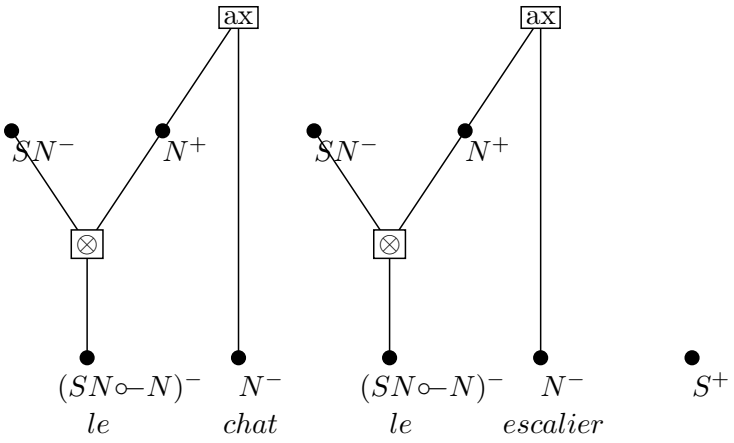
$$\vdash (SN\circ\!\!-N)^-, N^-, ((SN\multimap S)\circ\!\!-SN)^-, (SN\circ\!\!-N)^-, N^-, S^+.$$



• Exemple (β) :

Par contre, *Le chat l'escalier* $\notin L(G_1)$, car il n'y pas de réseau possible pour $\vdash (SN\circ\!\!-N)^-, N^-, (SN\circ\!\!-N)^-, N^-, S^+$:

[2]Son seul graphe de correction au sens du critère de Danos-Régnier étant acyclique et connexe.



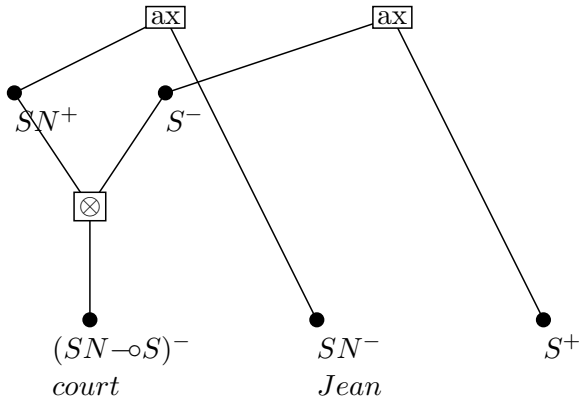
En effet, on ne peut joindre à la forêt syntaxique du séquent que deux liens axiomes, et cela de sorte à empecher des croisements entre ces liens. On ne peut donc même pas construire une structure de preuve, puisque cela entraînerait l'ajout de nouveaux types au séquent de départ (par exemple, S^-). On conclut donc qu'il n'y a pas de réseau possible pour ce séquent.

• Exemple(γ)

On voit aussi que *court Jean* $\notin L(G_1)$ – i.e. qu'il n'y a pas de preuve π de la logique linéaire intuitionniste non commutative pour le séquent

$$\vdash (SN\multimap S)^-, SN^-, S^+.$$

Intuitivement, cette phrase combine ses syntagmes dans le mauvais ordre. Ce qui se traduit formellement par la non-planarité du graphe qui n'est donc, malgré son acyclicité et sa connexité (qui entraîne celle de ses graphes de correction), un réseau pour $MINCLL$:



4.2 Les modules

Définition 4.2.1 *L'ensemble $\mathcal{M}$ des modules, est l'ensemble des liens M telles que :*

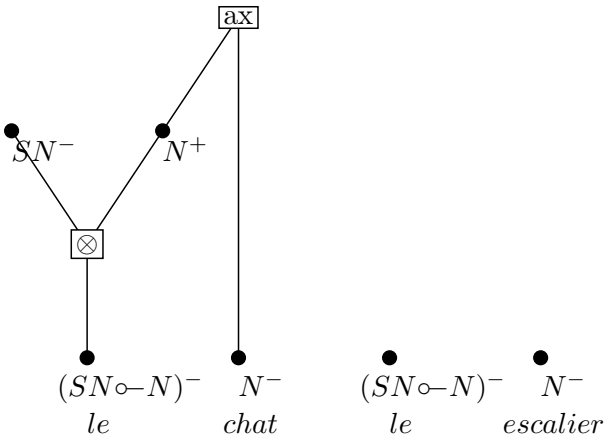
1. M a été obtenu à partir d'une structure de preuve $S \in \mathcal{ST}$ en enlevant au moins un noeud axiome – ce qui a pour effet de couper le graphe.

2. M vérifie le critère de correction suivante : ses graphes de correction sont acycliques et ne comportent pas de composante interne.

Ce sont donc des réseaux de preuve partiels. De plus, si l'on a un réseau R ayant k liens axiome on obtiendra en le coupant au plus $2k$ modules (car, en enlevant un lien axiome, on obtient au plus deux modules).

Exemple 4.2.1

Voici trois exemples de modules – des sous graphes de réseaux et donc des structures de preuve. Ils sont tirés du réseau de l'exemple 5.2 (β). On voit bien qu'ils sont des modules puisque leurs graphes de correction sont acycliques et connexes (l'absence de sommets $\wp$ les rendant identiques à ces derniers graphes) :


ax
SN^-
N^+
$(SN\circ\!\!-N)^-$
N^-
$(SN\circ\!\!-N)^-$
N^-
le
chat
le
escalier

Chapitre 5

Les grammaires de modules

Voici donc la partie originale de ce travail et à laquelle, comme on l'a indiqué dans l'introduction, mène l'état de l'art décrit au cours des chapitres précédents. On définit ici la classe des grammaires de modules avec étiquettage syntaxique ou grammaires de modules étiquettées, ainsi que quelques unes des ses sous-classes qui s'avèrent apprenables.

Une grammaire de modules est, en essence, une grammaire catégorielle où l'ensemble $\mathcal{T}$ des types a été substitué par $\mathcal{M}$, l'ensemble des modules de la logique linéaire. Pour arriver aux grammaires étiquettées il faut étendre leur définition jusqu'à envelopper les langages des modules $\subseteq (\Sigma \times \mathcal{M})^*$, d'une façon analogue à celle qu'on entreprend lorsqu'on travaille dans le cadre des grammaires catégorielles classiques et des langages de structures (cf. [6]).

Aussi les grammaires de modules étiquettées sont-elles des grammaires où l'alphabet est constitué d'un ensemble de symboles "typés" ou "étiquettés" par l'un de ces graphes (i.e. par un $M \in \mathcal{M}$). Autrement dit, l'alphabet est un sous-ensemble de $\Sigma \times \mathcal{M}$. Il y a ainsi deux cas à étudier : *primo*, la classe GC_{modf} des grammaires définies sur des alphabets finis et *secundo*, la classe GC_{mod} des grammaires définies sur $\Sigma \times \mathcal{M}$ tout entier (qui est, lui, infini dénombrable). Comme on le verra par la suite, GC_{modf} est apprenable tout court (de par son élasticité finie) et $\forall n \in \mathbb{N}$ la classe des $G \in GC_{mod}$ réduites par rapport à un sous-ensemble fini fixe de $(\Sigma \times \mathcal{M})^+$ de taille bornée par n est, elle aussi apprenable (de par son densité ou épaisseur finie). Enfin, on remarque que dans $\mathcal{M}$ on se restreint par la suite aux modules dont les types ou formules les étiquettant sont des images, par les traductions $^-$ et

$^+$, des types de $\mathcal{T}$.

5.1 Les grammaires de modules étiquettées

Définition 5.1.1 *On appelle* alphabet de modules *toute partie de* $\Sigma \times \mathcal{M}$, Σ *étant un alphabet fini et* $\mathcal{M}$ *l'ensemble des modules.*

Définition 5.1.2 *Une* grammaire catégorielle de modules avec étiquettage *ou* grammaire étiquettée *est un triplet* $G = \langle\, \Sigma_{mod}, \delta_{mod}, S \,\rangle$ *avec :*

1. $\Sigma_{mod} \subseteq \Sigma \times \mathcal{M}$.

2. $\delta_{mod} : \Sigma_{mod} \to \mathcal{P}_f(\mathcal{M})$ *une fonction finie telle que :*

- $\forall m^M \in \Sigma_{mod}$, *soit* $\delta_{mod}(m^M) = \{M\}$ *soit* $\delta_{mod}(m^M) = \emptyset$.
- $supp(G) = \{m^M \in \Sigma_{mod} \mid \delta_{mod}(m^M) \neq \emptyset\}$ *est fini.*

3. $S \in \mathcal{A}$ *(le type des expressions bien formées).*

4. Si $M \in \delta_{mod}(m^M)$, *on écrit* $G : m^M \mapsto M$.

Définition 5.1.3 $L(G)$, *le* langage engendré *par* G, *est l'ensemble des suites* $s \in \Sigma^+_{mod}$ *telles que :*

1. $s = m_1^{M_1} ... m_k^{M_k}; M_1, ..., M_k$ *avec des conclusions étiquetées* $A_1, ..., A_m$;

2. $G : m_i^{M_i} \to M_i$, *pour* $i \in [1, k]$ *; et*

3. Le graphe obtenu à partir des modules $M_1, ..., M_k$ *en ajoutant des liens axiome est un réseau de preuve pour* $\vdash A_1^-, ..., A_m^-, S^+$.

Définition 5.1.4 *Une* grammaire étiquettée d'alphabet fini *est une grammaire étiquettée* $G = \langle\, \Sigma_{modf}, \delta_{modf}, S \,\rangle$ *telle que* $\Sigma_{modf} \subset \Sigma \times \mathcal{M}$ *est un ensemble fini et* δ_{modf} *est définie sur* Σ_{modf} *(et à valeurs dans* $\mathcal{M}$*), toutes choses égales par ailleurs – y comprise la définition de langage engendré.*

Définition 5.1.5 *On note GC_{mod} la classe des grammaires étiquettées d'alphabet Σ_{mod} et $\mathcal{L}(GC_{mod})$ leur classe de langages associée. De même, GC_{modf}, $\mathcal{L}(GC_{modf})$ notent, respectivement, les grammaires étiquettées d'alphabet fini (d'alphabet fini $\Sigma_{modf} \subset \Sigma_{mod}$) et leur classe de langages associée.*

Remarque 5.1.1

• *Le système grammatical concernée des grammaires étiquettées tout court est $GS_{mod} = \langle\, \Sigma^+_{mod},\, GC_{mod},\, L \,\rangle$.*

• *Le système des grammaires étiquettées d'alphabet fini (ou ayant un alphabet fini) est le triplet $GS_{modf} = \langle\, \Sigma^+_{modf},\, GC_{modf},\, L \,\rangle$.*

Définition 5.1.6 *Soient $G, G' \in GC_{mod}$. $G \sqsubseteq G'$ ssi $supp(G) \subseteq supp(G')$. On note $\sqsubset$ l'ordre strict associé.*

Fait 5.1.1 *$\sqsubseteq$ est un ordre partiel.*

Définition 5.1.7 *Soit $G, \in GC_{mod}$. La taille de G, notée $t(G)$, est égale au cardinal de $supp(G)$ – i.e. $t(G) = card(supp(G))$.*

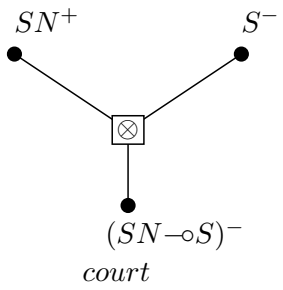
Exemple 5.1.1

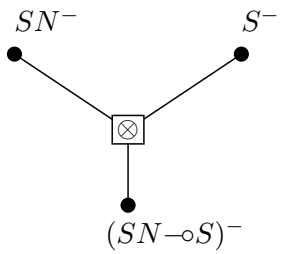
Pour illustrer ces notions, voici deux grammaires G_2 et $G_3 \in GC_{mod}$:

(α) On se donne d'abord G_2 :

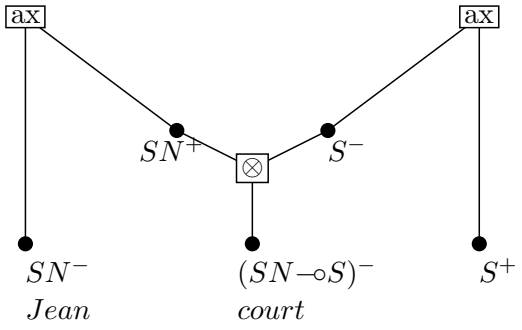
$G_2 = \langle\, \Sigma_{mod},\, \delta^2_{mod},\, S \,\rangle$ avec δ^2_{mod} définie par :

Et :





Notons M_1 et M_2 les deux modules ci-dessus (i.e. les "images" de δ^2_{mod}). M_1 et M_2 sont bien des modules puisque ses graphes de correction (au sens Danos-Régnier) sont acycliques, connexes et ne comportent pas de composante interne. Le réseau qui en résulte est, en outre, planaire, ce qui nous permet de dire que $L(G_2)$ contient la phrase $Jean^{M_1} court^{M_2}$:

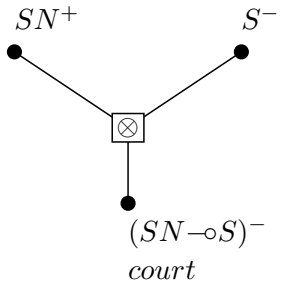


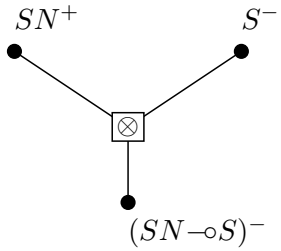
On a par ailleurs que :

- La taille de G_2 est $t(G_2) = card(supp(G_2)) = 2$; et
- $supp(G_2) = \{Jean^{M_1}, court^{M_2}\}$.

(β) Et ensuite G_3 :

$G_3 = \langle\, \Sigma_{mod},\, \delta^3_{mod},\, S \,\rangle$, où δ^3_{mod} est définie par :





Où :

- $supp(G_3) = \{court^{M_2}\}$;
- $t(G_3) = card(supp(G_3)) = 1$;
- $G_3 \sqsubseteq G_2$ et
- $L(G_3) \subseteq L(G_2)$ – car $L(G_3) = \emptyset$.

5.2 Apprenabilité des grammaires étiquettées

Voici donc les résulats promis. Dans cette section on adopte la convention de réprésenter les modules pour ses types sortants (pour ses types conclusion). Ainsi, un module M avec conclusion A sera noté ou écrit A. Par exemple, on écrira la chaîne $a^{\bullet X^-}$ de Σ^+_{mod}, a^{X^-} (on conservera les symboles et les types sortants et on en ommétra le reste, dont le graphe).

Lemme 5.2.1 *$\mathcal{L}(GC_{modf})$ possède une élasticité finie.*

(**Preuve**) Supposons le contraire. Ils existent alors deux suites infinies $< L_i >_{i\in\mathbb{N}}$ et $< s_i >_{i\in\mathbb{N}}$ avec, $\forall i \in \mathbb{N}$, $L_i \in \mathcal{L}(GC_{modf})$, $s_i \in \Sigma^+_{modf}$ et :

(**1**) $s_i \notin L_i$;

(**2**) $\{s_0, ..., s_i\} \subseteq L_{i+1}$.

Or, Σ_{modf} étant fini, las classe GC_{modf} (des grammaires étiquettées d'alphabet Σ_{modf} sera, elle aussi finie et *a fortiori*, $\mathcal{L}(GC_{modf})$ aussi. En effet soit $\Sigma_{modf} = \{m_1^{M_1}, ..., m_k^{M_k}\}$ un alphabet de modules fini fixe. On sait que, en général, toute grammaire catégorielle est determinée par sa fonction d'assignation. Ainsi, $\forall\, i \in [1, k]$, $\forall G \in GC\prime_{modf}$, on a de deux choses l'une :

(**a**) Soit $\delta_{modf}(m_i^{M_i}) = \{M_i\}$.

(**b**) Soit $\delta_{modf}(m_i^{M_i}) = \emptyset$.

Autrement dit, les grammaires de la classe diffèrent selon que leur fonction d'assignation soit (lorsque (**a**) est verifiée) ou ne soit pas (si au contraire, (**b**) est verifiée) définie sur le mot $m_i^{M_i}$, pour $i \in [1, k]$. Ce qui nous donne autant de grammaires que de parties de Σ_{modf}, en l'occurrence, 2^k, qui est un nombre fini. Et si la classe de grammaires est finie, il en va de même pour leur classe de langages engendrés.

Ceci implique que $\exists i_0, j_0 \in \mathbb{N}$, $i_0 \neq j_0$, tels que $L_{i_0} = L_{j_0}$. Supposons, s.p.g. que $i_0 < j_0$; $i_0 + 1 \leq j_0$. Donc par (**1**), $s_{i_0} \in L_{j_0} = L_{i_0}$; et au même temps, par (**2**), $s_{i_0} \notin L_{i_0}$, ce qui est absurde. **C.Q.F.D.**

Théorème 5.2.1 *GC_{modf} est apprenable.*

(**Preuve**) C'est un corrollaire, moyennant le **Théorème 2.1.1**, du **Lemme 5.2.1**. **C.Q.F.D.**

Lemme 5.2.2 *Le système GS_{mod} a une densité finie bornée.*

(**Preuve**) On procède par étapes.

(**1**) $\forall G, G' \in GC_{mod}$, $G \sqsubseteq G'$ implique que $L(G) \subseteq L(G')$ (fait trivial).

(**2**) Soit $\mathcal{X} \subseteq \Sigma^+_{mod}$, $\mathcal{X}$ fini. Posons $R_{\mathcal{X}} = \{L(G) \mid G \in GC_{mod},\ t(G) \leq n$ et G est réduite par rapport à $\mathcal{X}\}$. Alors $\forall n \in \mathbb{N},\ R_{\mathcal{X}}$ est une classe finie. On examine tous les cas possibles.

(**2.1**) Tout d'abord, commençons par définir quelques ensembles nécessaires par la suite. Aussi pose-t-on, $\forall S \subseteq \Sigma^+_{mod}$:

(**a**) $lettres(S) = \{m^M \in \Sigma_{mod} \mid \exists s \in S$ tel que m^M est un facteur de $s\}$.

En particulier $lettres(\mathcal{X}) = \{m^M \in \Sigma_{mod} \mid \exists s \in \mathcal{X}$ tel que m^M est un facteur de $s\}$. Un ensemble qui est, lui aussi, fini. On a besoin de raisonner par la suite sur les grammaires engendrant les langages de $\mathcal{X}$. Pour ce faire, on pose :

(**b**) $C_1 = \{G \in GC_{mod} \mid \mathcal{X} \subseteq L(G)\}$.

Qui est la classe des grammaires étiquettées $G \in GC_{mod}$ de la forme $G = \langle\, \Sigma_{mod},\, \delta_{mod},\, S \,\rangle$ telles que :

(**b.i**) $\forall s \in \mathcal{X}$ avec $s = m_1^{M_1} ... m_p^{M_k}$, $m_i^{M_i} \in supp(G)$, pour $i \in [1, k]$.

(**b.ii**) $\forall s \in L(G) - \mathcal{X},$ avec $s = m_1^{M_1} ... m_r^{M_r}, m_j^{M_j} \in supp(G)$ aussi, pour $j \in [1, r]$.

Cette classe est celle des grammaires engendrent $\mathcal{X}$ et pour lesquelles on vérifie que $lettres(\mathcal{X}) \subseteq supp(G)$. En effet, soit $m^M \in lettres(\mathcal{X})$. Alors,

par définition de cet ensemble, m^M est facteur d'une chaîne s de $\mathcal{X}$, et si G est une grammaire de modules vérifiant les conditions énoncées dans (**b**) alors, comme G génère s (i.e. comme $s \in L(G)$) on a que $m^M \in supp(G)$.

(**c**) $C_2 = \{G \in C_1 \mid G \text{ est réduite par rapport à } \mathcal{X}\}$.

(**d**) $C_3^n = \{G \in C_1 \mid t(G) \leq n\}$.

Qui est la classe des $G \in C_1$ dont la taille est bornée par $n \in \mathbb{N}$.

(**2.2**) On ainsi que $R_{\mathcal{X}} = \{L(G) \mid G \in C_3^n \cap C_2\}$. Ce qui nous permet désormais de raisonner sur $C_3^n \cap C_2$, autrement dit, sur la classe des grammaires étiquettées générant les $L \in R_{\mathcal{X}}$. Et ceci, selon que $\mathcal{X}$ soit ou ne soit pas vide.

(**a**) $\mathcal{X} = \emptyset$. Alors on montre que C_1 et C_2, ne sont pas vides. Que C_2 ne contient qu'une et une seule instance, la grammaire notée $G_{\mathcal{X}}$ (en l'ocurrence, la grammaire vide). Que $\forall n \in \mathbb{N}$, la classe C_3^n n'est pas non plus vide. Et enfin que $\forall n \in \mathbb{N},\ C_2 \cap C_3^n = \{G_{\mathcal{X}}\}$ et *a fortiori* que $R_{\mathcal{X}}$ est une classe finie.

(**a.i − existence**) $\mathcal{X} = \emptyset$ implique que $lettres(\mathcal{X}) = \emptyset$. On prend donc une grammaire au support vide i.e. on prend $G_{\mathcal{X}} = \langle\, \Sigma_{mod}, \delta_{mod}^{\mathcal{X}}, S \,\rangle$ avec $\delta_{mod}^{\mathcal{X}}$ telle que $supp(G_{\mathcal{X}}) = \emptyset$. $L(G_{\mathcal{X}}) = \emptyset$; d'où : $\mathcal{X} \subseteq L(G_{\mathcal{X}})$; donc $G_{\mathcal{X}} \in C_1$. En outre, $G_{\mathcal{X}} \in C_2$, c'est-à-dire que $G_{\mathcal{X}}$ est une grammaire réduite par rapport à $\mathcal{X}$. Sinon ce que $\exists G_0 \in GC_{mod}$ telle que $G_0 \sqsubset G_{\mathcal{X}}$ et telle que $\mathcal{X} \subseteq L(G_{\mathcal{X}})$ et dont il est par ailleurs clair que $G_0 \in C_1$. Mais, comme $supp(G_{\mathcal{X}}) = \emptyset$, alors, par définition de $\sqsubseteq$, $G_{\mathcal{X}} \sqsubseteq G_0$. Contradiction.

(**a.ii − unicité**) Soit $G \in C_2$. Or, comme $supp(G_{\mathcal{X}}) = \emptyset$, $supp(G_{\mathcal{X}}) \subseteq supp(G)$. Donc $G_{\mathcal{X}} \sqsubseteq G$ et de deux choses l'une :

- $G_{\mathcal{X}} = G$.

- $G_{\mathcal{X}} \sqsubset G$. Mais G est réduite par rapport à $\mathcal{X}$, ce qui implique que $\mathcal{X} \not\subseteq L(G_{\mathcal{X}})$. Contradiction.

(**a.iii**) $G_{\mathcal{X}} \in C_3^0$, car $t(G_{\mathcal{X}}) = 0$. Donc, $\forall n \in \mathbb{N}, C_3^n \neq \emptyset$. Et, puisque $C_2 = \{G_{\mathcal{X}}\}$, il s'ensuit que, $\forall n \in \mathbb{N},\ C_3^n \cap C_2 = \{G_{\mathcal{X}}\}$ et $R_{\mathcal{X}} = \{L(G_{\mathcal{X}})\}$, qui est finie.

(**b**) $\mathcal{X} \neq \emptyset$. $\mathcal{X} = \{s_1, ..., s_k\}$ avec $s_i = m_{i_1}^{M_{i_1}} ... m_{i_p}^{M_{i_p}}$ tel que $m_{i_j} \in \Sigma$ et $M_{i_j} \in \mathcal{M}$, pour $i \in [1,k]$ et $j \in [1,p]$. De même $lettres(\mathcal{X}) = \{m_{i_j}^{M_{i_j}} \mid i \in [1,k], j \in [1,p]\} \neq \emptyset$ aussi. Le raisonnement à suivre cette fois est similaire à celui du cas précédent. On arrive aussi à une unique grammaire réduite par rapport à $\mathcal{X}$, notée à nouveau $G_{\mathcal{X}}$, si celle-ci existe. $R_{\mathcal{X}}$ reste finie dans tous les cas de figure.

(**b.i**) La classe C_1 peut être vide. En effet, il se peut que parmi les chaînes s_i, pour $i \in [1,k]$, de $\mathcal{X}$ il en existe au moins une dont il n'y ait pas de réseau de preuve possible. Auquel cas tous ses sous-classes le seraient aussi. Ainsi aurait-on que $\forall n \in \mathbb{N},\ C_1 = C_2 = C_3^n \cap C_2 = \emptyset$ et par suite que $R_{\mathcal{X}} = \emptyset$, classe qui est finie.

(**b.ii**) Sinon ce que C_1 n'est pas vide. L'idée dans ce cas est de tirer profit du fait que $C_1 \neq \emptyset$ et que $\mathcal{X} \subseteq L(G)$ implique $supp(G) \neq \emptyset$; ce qui nous permet d'affirmer que $\forall G \in C_1$, $supp(G) \neq \emptyset$. Ceci nous autorise à raisonner sur l'intersection (arbitraire) de leurs supports :

(**b.ii.1 – existence**) On prend la grammaire de modules suivante, $G_{\mathcal{X}} = \langle\, \Sigma_{mod}, \delta_{mod}^{\mathcal{X}}, S \,\rangle$ avec $\delta_{mod}^{\mathcal{X}}$ telle que :

$$(*)\ supp(G_{\mathcal{X}}) = lettres(\mathcal{X}) \subseteq \bigcap_{G \in C_1} supp(G)$$

Cette défintion a sens puisque l'intersection (arbitraire) des supports des grammaires de C_1 est définie et que $\forall G \in C_1$, $lettres(\mathcal{X}) \subseteq supp(G)$ (cf. **2.1.b**). On a ainsi que $\mathcal{X} \subseteq L(G_{\mathcal{X}})$, car, en effet, $\forall i \in [1,k],\ \forall i \in [1,p],\ G_{\mathcal{X}} : m_{i_j}^{M_{i_j}} \mapsto M_{i_j}$. D'où : $G_{\mathcal{X}} \in C_1$. De plus, elle est réduite par rapport à $\mathcal{X}$, ce qui revient à dire que $G_{\mathcal{X}} \in C_2$. Sinon ce que, *a contrario*, $\exists G_0 \in GC_{mod}$ telle que $G_0 \sqsubset G_{\mathcal{X}}$ et telle que $\mathcal{X} \subseteq L(G_0)$. Il est clair que $G_0 \in C_1$. Mais ceci implique aussi, par la condition $(*)$, que $supp(G_{\mathcal{X}}) \subseteq supp(G_0)$ et *a fortiori* que $G_{\mathcal{X}} \sqsubseteq G_0$, ce qui contredit la supposition.

(**b.ii.2 – unicité**) Soit $G \in C_2$. Alors $G \in C_1$ aussi. D'où, par définition de $G_{\mathcal{X}}$, $supp(G_{\mathcal{X}}) \subseteq supp(G)$ i.e. $G_{\mathcal{X}} \sqsubseteq G$ et on effectue le même raisonnement qu'au (**a.ii**).

(**b.iii**) Comme $lettres(\mathcal{X}) \neq \emptyset$, et que ce dernier ensemble est fini, on a que $card(lettres(\mathcal{X})) = m_0$, pour quelque $m_0 \in \mathbb{N}^*$ (m_0 est non nul). La

taille de $G_{\mathcal{X}}$ (de par sa définition, i.e. son support) est donc $t(G_{\mathcal{X}}) = m_0$. On va montrer, que :

(**b.iii.1**) $\forall n < m_0$, $C_2 \cap C_3^n = \emptyset$. Ce qui implique que $R_{\mathcal{X}} = \emptyset$, qui est finie.

• $n = 0$. La classe $C_3^0 \cap C_2$ est vide puisque $G_{\mathcal{X}}$, la grammaire réduite par rapport à $\mathcal{X}$, est de taille non-nulle.

• $n = k + 1, k \in [0, m_0 - 1]$. Par hypothèse de récurrence finie sur k, $C_2 \cap C_3^k = \emptyset$. Or, $C_2 \cap C_3^{k+1} = \emptyset$ aussi. Sinon soit $G_0 \in C_2 \cap C_3^{k+1}$. Alors G_0 est une grammaire réduite par rapport à $\mathcal{X}$ et de taille $t(G_0) \leq k + 1$. D'autre part, $G_0 \in C_2$ implique que $G_0 = G_{\mathcal{X}}$ (car cette classe se réduit à une seule instance). Mais ceci est impossible, puisque $t(G_{\mathcal{X}}) = m_0 > k + 1$.

(**b.iii.2**) $\forall n \geq k_0$, $C_2 \cap C_3^n = \{G_{\mathcal{X}}\}$. Ce qui implique que $R_{\mathcal{X}} = \{L(G_{\mathcal{X}})\}$, qui est finie.

• $n = m_0$. On prend la grammaire $G_{\mathcal{X}}$. Donc $C^{m_0} \cap C_2 = \{G_{\mathcal{X}}\}$.

• $n = k+1, k \geq m_0$. Par hypothèse de récurrence sur k, $C_3^k \cap C_2 = \{G_{\mathcal{X}}\}$; ce qui suffit puisque $C_3^{k+1} \cap C_3^k \cap C_2 = C_3^k \cap C_2$.

Ce qui établit le résultat. **C.Q.F.D.**

Corollaire 5.2.1 $\forall n \in \mathbb{N}$, *la classe* $\{L(G) \mid G \in GC_{mod}$ *et* $t(G) \leq n\}$ *a une élasticité finie.*

(**Preuve**) C'est une conséquence immédiate du **Lemme 5.2.2** moyennant le **Théorème 1.2.3**. **C.Q.F.D.**

Théorème 5.2.2 $\forall n \in \mathbb{N}$, *la classe* $\{G \in GC_{mod} \mid t(G) \leq n\}$ *est apprenable.*

(**Preuve**) C'est une conséquence immédiate du **Corollaire 5.2.1** moyennant le **Théorème 1.2.1**. **C.Q.F.D.**

Par contre, la classe $\mathcal{L}(GC_{mod})$ n'a pas, elle, d'élasticité finie :

Théorème 5.2.3 *$\mathcal{L}(GC_{mod})$ a une élasticité infinie*[1].

(**Preuve**) On démontre le théorème par étapes :

(**1**) On définit d'abord une suite $< S^i >_{i\in \mathbb{N}}$ de modules :

$$S^0 = S^-;$$

$$S^1 = (S \multimap S)^-;$$

$$S^i = (\underbrace{S \multimap (...(S}_{i-fois} \multimap S)))^-.$$

(**2**) On définit deux suites (ou séquences) infinies $< s_i >_{i\in \mathbb{N}}$, $< L_i >_{i\in \mathbb{N}}$ avec, pour $i \in \mathbb{N}$, $L_i \in \mathcal{L}(GC_{mod})$ et $s_i \in \Sigma^+_{mod}$ comme suit :

(**a**) D'emblée $< s_i >_{i\in \mathbb{N}}$:

$$s_0 = a^{S^0};$$

$$s_1 = a^{S^0} a^{S^1};$$

$$s_i = \underbrace{a^{S^0} ... a^{S^0}}_{i-fois} a^{S^i}.$$

Où le module S^i est le i-ème terme de la suite ou séquence $< S^i >_{i\in \mathbb{N}}$. On remarque, en outre, que $\forall i, j \in \mathbb{N}$, si $i \neq j$, alors $a^{S^i} \neq a^{S^j}$ (et *a fortiori* $s_i \neq s_j$).

(**b**) Ensuite $< L_i >_{i\in \mathbb{N}}$, que l'on définit par le biais de la suite $< G_i >_{i\in \mathbb{N}}$ des grammaires de modules les engendrant, nommément celle où :

[1] C'est ici où l'on se sert le plus de la convention déjà citée sur les modules.

G_0 est la grammaire telle que :

$$supp(G_0) = \emptyset.$$

G_1 est la grammaire définie par :

$$G_1 : a^{S^0} \mapsto S^0;^2$$

G_i est la grammaire définie par

$$G_i : a^{S^0} \mapsto S^0;$$

$$G_i : a^{S^1} \mapsto S^1;^3$$

$$\vdots$$

$$G_i : a^{S^i} \mapsto S^i.^4$$

On pose ainsi : $L_i = L(G_i)$, pour $i \in \mathbb{N}$. On a en plus que $\forall i, j \in \mathbb{N}$, $i \leq j$ implique que $L(G_i) \subseteq L(G_j)$ et donc que la suite $< L_i >_{i \in \mathbb{N}}$ est croissante.

(**3**) Enfin, d'après (**1**) et (**2**), on est en mesure d'affirmer que, $\forall n \in \mathbb{N}$:

$$(\mathbf{a})\ s_n = \underbrace{a^{S^0}...a^{S^0}}_{n-fois} a^{S^n} \notin L_n = L(G_n);$$

$$(\mathbf{b})\ \{s_0, ..., s_n\} \subseteq L_{n+1} = L(G_{n+1}).$$

En effet, raisonnons par récurrence sur $n \in \mathbb{N}$:

• $n = 0$. Le cas de base est trivial : $s_0 = a^{S-} \notin L(G_0)$ (car $a^{S-} \notin supp(G_0)$), mais $\{s_0\} \subseteq L(G_1)$.

• $n = k+1$. La propriété est vraie pour k. $s_{k+1} = \underbrace{a^{S^0}...a^{S^0}}_{(k+1)-fois} a^{S^{k+1}}$. Alors :

[2] Pour rappel : $S^0 = S^-$.
[3] Pour rappel : $S^1 = (S \multimap S)^-$.
[4] Pour rappel : $S^i = (\underbrace{S \multimap (...(S}_{i\ fois} \multimap S)))^-$.

D'une part, $a^{S^{k+1}} \notin supp(G_{k+1})$. Donc $s_{k+1} \notin L(G_{k+1})$, puisque le séquent

$$\vdash \overbrace{S^-, ..., S^-}^{(k+1)-fois}, S^+$$

N'admet pas de preuve.

D'autre part, on a que $a^{S^{k+1}} \in supp(G_{k+2})$ et que le séquent

$$\vdash \overbrace{S^-, ..., S^-}^{(k+1)-fois}, (\underbrace{S \multimap (...(S}_{(k+1)-fois} \multimap S)))^-, S^+$$

Admet une preuve. Donc $s_{k+1} \in L(G_{k+2})$. Par ailleurs, par hypothèse de récurrence sur k, $\{s_0, ..., s_k\} \subseteq L(G_{k+1}) \subseteq L(G_{k+2})$ et par suite on a enfin que $\{s_0, ..., s_k, s_{k+1}\} \subseteq L(G_{k+2})$.

Ce qui conclut. **C.Q.F.D.**

Conclusions

Voici les principales conclusions et perpectives du présent travail.

1. Au vu de l'équivalence de pouvoir expressif établie entre $MINCLL$ (respectivement LC) et les réseaux de preuve planaires, on est mesure de dire que les réseaux sont capables de modéliser les mêmes phénomènes syntaxiques et de remplacer les arbres de analyse syntaxique (*bottom up* ou d'analyse ascendant) des grammaires catégorielles par des graphes qui simplifient le processus d'analyse syntaxique d'une phrase. Un réseaux de preuve (un $R \in \mathcal{RP}$) est un type spécial de structure de preuve (une $S \in \mathcal{SP}$) ; une structure de preuve séquentialisable dont on peut extraire un arbre de démonstration (en calcul de séquents), car image *modulo* une certaine traduction, de celui-ci – et cela dès lors que cette structure vérifie un certain critère, le critère de correction.

2. Les structures et réseaux mènent à leur tour à définir un nouveau ensemble : l'ensemble $\mathcal{M}$ des modules. Tout comme les types dans une grammaire catégorielle usuelle, ils sont censés capturer formellement la notion de syntagme (de partie de la phrase, comme le nom commun, le syntagme nominal, l'adverbe et ainsi de suite). Qui plus est, leur ajout (moyennant des structures de preuve – des liens – axiome) doit nous fournir un réseau.

3. L'équivalence, déjà notée, entre preuves et réseaux de preuve (i.e. des structures de preuve séquentialisables) permet ensuite de définir une nouvelle classe des grammaires catégorielles : les grammaires de modules. Celles-ci sont des grammaires où des modules (des $M \in \mathcal{M}$) jouent le rôle des types (des $A \in \mathcal{T}$). Classe à partir de laquelle on peut (de façon analogue que pour les grammaires catégorielles classiques et les langages de structures[5]) définir deux nouvelles classes : *Primo* la classe de grammaires

[5] cf. [6]

catégorielles avec étiquettage syntaxique ou grammaires étiquettées, d'alphabet $\Sigma_{mod} = \Sigma \times \mathcal{M}$ et dont la fonction δ_{mod} est définie sur Σ_{mod} et à valeurs sur $\mathcal{M}$ – i.e. des grammaires de modules où, intuitivement, on a rajouté des informations (en l'ocurrence, relatives à leur catégorie syntaxique) aux éléments de leur lexiques. Et *secundo* les grammaires étiquettées d'alphabet $\Sigma_{modf} \subset \Sigma \times \mathcal{M}$ fini, où l'alphabet revient à une partie finie de $\Sigma \times \mathcal{M}$.

4. Les grammaires étiquettées d'alphabet fini (i.e. les $G \in GC_{modf}$) sont apprenables dans le modèle de Gold (i.e. par identification à la limite à partir d'exemples positifs). Ceci est une conséquence de l'élasticité finie de la classe de ses langages engendrés.

5. On peut définir un nombre dénombrale de sous-classes apprenables de la classe GC_{mod}, la classe de grammaires étiquettées d'alphabet infini, les classes des grammaires de taille $\leq n$, pour $n \in \mathbb{N}$.

6. Ce dernier résultat n'implique pas, pour autant, que la classe $\mathcal{L}(GC_{mod})$ ait une élasticité finie et que GC_{mod} soit ainsi immédiatement apprenable – l'élasticité finie n'étant qu'une condition suffissante d'apprenabilité. En fait, c'est le contraire.

7. Parmi les innombrables perspectives (tant pratiques que théoriques) on peut envisager notamment les suivantes :

i. Trouver l'algorithme de apprentissage $\phi_{GC_{mod}}$ pour la classe GC_{mod} – tout en montrant qu'il en est bel et bien un.

ii. Implémenter ces grammaires ainsi que leur algorithme d'apprentissage, et cela et en restant dans le cadre de la logique linéaire et en profittant, autant que possible, des résultats issus de la théorie de la preuve.

iii. Voir quels sont les rapports qu'elle a, du côté du pouvoir expressif, avec les grammaires hors-contexte.

iv. Etudier leur capacité à modéliser et la syntaxe (par exemple, l'anaphore ou les règles transformationelles) et la sémantique (moyennant les interprétations de la logique linéaire, par exemple) des langues naturelles.

v. Etudier les systèmes grammaticaux et conceptuels dans toute leur généralité – d'un point de vue algébrique, par exemple.

Annexe. Les grammaires catégorielles classiques

On impléménte ici les grammaires catégorielles AB. C'est-à-dire celles où le système de types (i.e. de dérivation) est le fragment du calcul de Lambek (LC ou $MINCLL$) réduit aux règles d'élimination pour $-\circ$ et $\circ-$ (aux règles gauches pour un calcul de séquents). Mais il n'en est rien, puisque, de par le théorème de Pentus, leur classe a le même pouvoir expressif que la classe des grammaires hors-contexte. Le langage de programmation choisi pour ce faire est SCHEME, un dialecte interprété (et faiblement typé) du LISP. La définition formelle des grammaires catégorielles fut énoncé au **Chapitre 3**. Quant aux définitions des types (i.e. de $\mathcal{T}$ et de $\mathcal{T}''$), des séquents, des traductions $^+$ et $^-$, ainsi que des systèmes LC et $MINCLL$ on renvoie le lecteur au **Chapitre 2**.

Prolegomènes

Voici les procédures auxiliaires dont on s'est servi par la suite :

a. **forall-x** (procédure booléene) nous permet de quantifier une propriété P sur les éléments d'une liste L :

```
(define (forall-x L P)(if (and(= 1 (long(comp(map P L))))
                              (equal? #t (car
                                           (comp
                                              (map P L)))))
                          #t
                          #f))
```

b. La procédure (récursive) que voici est très importante, car elle nous permet d'inverser la liste de types de **mot-inf** (voir *infra* dans le bon ordre (de façon à simuler les applications de $-\circ^g$ et $\circ-^g$) :

```
(define (inverser L) (cond ((null? L)
                            L)
                           ((atom (car L))
                            (append (inverser (cdr L))
                                    (list (car L))))
                           ((and(atom (caar L))
                                (equal? '- (der(car L))))
                            (append (inverser (cdr L))
                                    (list (car L))))
                           ((append (inverser (cdr L))
                                    (list
                                       (inverser (car L)))))))
```

a. On pose ce qu c'est qu'un atome a :

```
(define (atom? a) (if (equal? a '() )
                      #f
                     (not(pair? a))))
```

b. **der** retourne le dernier élément d'une liste L :

```
(define (der L) (if (null? (cdr L))
                    (car L)
                    (der (cdr L))))
```

c. **anticdr** retourne la liste L sans son dernier élément :

```
(define (anticdr L) (reverse (cdr (reverse L))))
```

d. **elt** teste si l'objet x appartient à la liste L :

```
(define (elt? x L) (cond ((null? L)
                          #f)
                         ((equal? x (car L))
                          #t)
                         (else (elt? x (cdr L)))))
```

e. **prof** retourne le max des profondeurs des éléments de L :

```
(define (prof L) (cond ((null? L)
                        0)
                       ((atom (car L))
                        (prof (cdr L)))
                       (else (max (+ 1 (prof (car L)))
                                  (prof (cdr L))))))
```

f. **mir-do** retourne la liste miroir de L :

```
(define (mir-do L) (do ((liste L (cdr liste))
                        (acc () (cons(car liste) acc)))
                        ((null? liste) acc)))
```

g. **comp** compacte la liste L en éliminant les répétitions :

```
(define (comp L) (cond ((null? L)
                         L)
                       ((null? (cdr L))
                         L)
                       ((elt(car L)(cdr L))
                        (comp(cdr L)))
                       (else(cons(car L)
                            (comp(cdr L))))))
```

h. **long** retourne la longueur de la liste L :

```
(define (long L) (cond ((null? L) 0)
       ((null? (cdr L)) 1)
       (else (long_iter L 0))))
(define (long_iter L acc) (if (null? L)
       acc
      (long_iter (cdr L) (+ 1 acc))))
```

Les types

Les types (ou formules) $\mathcal{T}$ sont impléméntés dans les types (primitifs) ATOM et LIST – par exemple, le type N sera réprésenté par l'atome `n` et le type $S \multimap S$ par la liste `(s -o s)`.

```
(define (type? T)(cond((equal? T (void))
                       #f)
                      ((atom? T)
                       #t)
                      ((and(pair? T)
                           (= (long T) 3)
                           (type? (car T))
                           (type? (der T))
                           (or (eq? '-o (cadr T))
                               (eq? 'o- (cadr T))
                               (eq? 'o (cadr T))))
                       #t)
                      (else #f)))
```

Le calcul des séquents

§1. On étend l'ensemble des types $\mathcal{T}$ à $\mathcal{T}''$:

```
(define (type2? T)(cond((equal? T (void))
                        #f)
                       ((atom? T)
                        #t)
                       ((and (pair? T)
                             (= 2 (long T))
                             (atom (car T))
                             (equal? '- (cadr T)))
                        #t)
                       ((and(pair? T)
                            (= (long T) 3)
                            (type2? (car T))
                            (type2? (der T))
                            (or (equal? '-o (cadr T))
```

```
                        (equal? 'o- (cadr T))
                        (equal? 'o (cadr T)))
                        (equal? '8 (cadr T)))
                  #t)
                 (else #f)))
```

§2. La négation est involutive et définie sur les atomes.

```
(define (neg T)(cond((atom? T)
                    (list T '-))
                   ((and(= 2 (long T))
                        (atom? (car T))
                        (equal? '- (cadr T)))
                    (car T))
                   (else (void))))
```

§3. Définition d'un séquent quelconque.

Un séquent est une expression formelle $A_1, ..., A_n \vdash B_1, ..., B_m$, où les suites à gauche et à droite du symbole thèse $\vdash$, appellés contextes gauche et droite sont des suites de types (i.e. les A_i et les B_j sont des types). Il sera réprésenté par la liste `(A1 ... An => B1 ... Bm)` les `Ai` et les `Bi` étant soit des atomes soit des types (des listes les réprésentant) quelconques.

§3.1.Définition des contextes droite et gauche d'un séquent.

§3.1.1. Le contexte gauche.

```
(define (ga S)(if (and(> 3 (long S))
                      (equal? '=> (car S)))
                  '()
                  (ga_iter S '())))
(define (ga_iter S acc)(if(equal? (car S) '=>)
                          (mir_do acc)
                          (ga_iter (cdr S)
                                   (cons (car S) acc))))
```

§3.1.2. Le contexte droite.

```
(define (dr S)(if (and(> 3 (long S))
                       (equal? '=> (der S)))
                  '()
                  (dr_iter S '())))
(define (dr_iter S acc)(if(equal? (der S) '=>)
                          acc
                          (dr_iter (anticdr S)
                                   (cons (der S) acc))))
```

§3.2. Les séquents.

```
(define(sequent? S)(or(and(= 1 (long S))
                          (equal? '=> (car S)))
                      (and(elt? '=> S)
                          (or (null? (ga S))
                              (null? (dr S))
                              (forall-x (ga S) type2?)
                              (forall-x (dr S) type2?)))))
```

§4. On définit ce qu'est un séquent intuitionniste et un séquent de Lambek.

Dans un séquent intuitionniste, le contexte droite contient au plus un type. Dans un séquent de Lambek on empêche en plus que le contexte gauche soit vide.

```
(define (sequent_i? S)(and(sequent? S)
                          (>= 1 (long (dr S)))))

(define (sequent_lam? S) (and(sequent_i? S)
                             (not(null? (ga S)))))
```

§5. Voici les traductions $^+$ et $^-$ (définies par récurrence simultanée) :

```
(define (trad-neg T)(cond ((atom? T)
                           (neg T))
                          ((and(type? T)
                               (equal? (cadr T) '-o))
```

```
                          (list (trad-pos(car T))
                                '0
                                (trad-neg(caddr T))))
                          ((and(type? T)
                               (equal? (cadr T) '0-))
                          (list (trad-neg(car T))
                                '0
                                (trad-pos(caddr T))))
                          ((and(type? T)
                               (equal? (cadr T) '0))
                          (list (trad-neg(car T))
                                '8
                                (trad-neg(caddr T))))
                         (else (void))))

(define (trad-pos T) (cond((atom? T)
                           T)
                          ((and(type? T)
                               (equal? (cadr T) '-0))
                           (list (trad-pos(caddr T))
                                 '0
                                 (trad-neg(car T))))
                          ((and(type? T)
                               (equal? (cadr T) '0-))
                           (list (trad-neg(caddr T))
                                 '0
                                 (trad-pos(car T))))
                          ((and(type? T)
                                (equal? (cadr T) '0))
                           (list (trad-pos(car T))
                                 '8
                                 (trad-pos(caddr T))))
                          (else (void))))
```

§6. On étend $^{+}$ et $^{-}$ aux séquents.

```
(define (trad S)(append(map trad-neg (ga S))
                       (map trad-pos (dr S))))
```

Le moteur d'inférence

On s'intéresse ici à savoir si un séquent de Lambek du langage restreint à $\{-\circ, \circ-\}$ est prouvable ou démontrable dans le fragment implicatif gauche de la logique linéaire intuitionniste multiplicative non commutative et sans constantes ($MINCLL_{\{-\circ^g, \circ-^g\}}$), – autrement dit, dans le fragment implicatif gauche du calcul de Lambek. C'est -à-dire :

$$\frac{\Gamma, B, \Gamma' \vdash C \qquad \Delta \vdash A}{\Gamma, \Delta, A-\circ B, \Gamma' \vdash C} -\circ^g$$

Avec :

$$\frac{\Gamma, A, \Gamma' \vdash C \qquad \Delta \vdash B}{\Gamma, A\circ- B, \Delta, \Gamma' \vdash C} \circ-^g$$

Or, *modulo* les traductions $^+$ et $^-$, ce système est équivalent au système $MALL^-$ (logique linéaire multiplicative sans constantes) restreint à des séquents ayant une pré-image dans le langage de $MINCLL_{\{-\circ^g, \circ-^g\}}$ –cf. **Théorème 4.1.1** et **Théorème 4.1.2**. Ce qui veut dire que l'on n'a besoin que d'une seule règle, $\otimes$:

$$\frac{\vdash \Gamma, A, \Delta \qquad \vdash \Gamma', B, \Delta'}{\vdash \Gamma, \Gamma', A \otimes B, \Delta, \Delta'} \otimes$$

Implémentée dans **inf** ci-dessous. Et dont l'ordre ou la position des types par rapport à l'application de de la règle dépendra de leur polarité (négative ou positive) et donc de leurs pré-images – ce qui permet de simuler (sur ce fragment de langage) les règles $-\circ^g$ et $\circ-^g$.

mot-inf nous permettra ensuite déterminer que, par exemple, le séquent répresenté par la liste `(x (x -o y) => y)` est prouvable, et que `((x -o y) x => y)` ne l'est pas !

Le fait de se restreindre à ce fragment nous donne de façon directe, du fait que l'on se situe dans le cadre des grammaires catégorielles classiques (ou AB) quoiqu'en format calcul de séquents, l'équivalence de pouvoir expressif avec le formalisme des grammaires hors-contexte (cf. la **Proposition**

3.1 prouvée dans [6]).

(α) Schématiquement :

procedure **demon**(Γ, A)
variables Γ : *Sequence*, $A : \mathcal{T}$
begin
    if mot-inf($\vdash \Gamma^-, A^+$) $= \vdash$ then TRUE
                                  else FALSE
    eif
end

(β) Et en SCHEME :

```
(define (demon? S)(cond((not(sequent_lam? S))
                        #f)
                       ((equal? '() (mot_inf (trad S)))
                        #t)
                       (else #f)))
```

En effet, un séquent de Lambek $A_1, ..., A_n \vdash B$ est démontrable exactement si la procédure **mot-inf** appliquée au séquent $\vdash (A_1, ..., A_n)^-, B^+$ et donc à la liste `(trad(A1...An => B))` rétourne comme valeur la liste vide (i.e. le séquent vide).

L'algorithme

L'algorithme travaille comme suit. Il essaie d'imiter la contruction d'une structure de preuve (séquentialisable) pour un séquent de Lambek. Là où l'on relie par un lien axiome un atome et sa négation (contigüs dans la liste qu'il parcourt), on efface les deux types (et le connecteur !). Il la parcourt d'abord de gauche à droite (pour s'occuper des types ayant comme connecteur principal $-\circ$), puis de droite à gauche (pour en faire autant avec ceux qui ont $\circ-$). Le nombre d'étapes ou d'itérations de ce calcul est bornée par la profondeur du séquent σ donnée en entrée, i.e. par :

$$prof(\sigma) = max\{r(A) \mid A \text{ type de } \sigma\}.$$

Il y aura succès si on parvient à la liste vide ; échec sinon (i.e. si **mot-inf** retourne une liste qui n'est pas vide).

i. **mot-inf** revient à une itération bornée par la profondeur du séquent (donc, de la liste qui le réprésente) de la procédure de base **inf**.

(α) Voici son spécification (en pseudo-code) :

```
procedure mot-inf(Γ)
variables Γ : SEQ
begin
      Δ := Γ
      for 0 ≤ i ≤ max{r(A) | A ∈ Γ}
          do
            i := i + 1
            if even(i) = TRUE thenΔ :=inf(Δ)
                                else Δ :=inf(miroir(Δ))
            eif
          edo
      ef
end
```

(β) Et voici son implémentation :

```
(define (mot-inf S)(if (null? S)
                       S
                       (inf(cons (der S)
                           (iter_inf S (anticdr S) 0)))))
(define (iter_inf S acc com)(if (= com (prof S))
                                acc
                                (iter_inf S
                                          (inverser (inf acc))
                                          (+ 1 com))))
```

ii. Procédure de base **inf**.

Elle réprésente la règle $\otimes$.

(α) En voici la spécification :

procedure **inf**($\vdash A_1, ..., A_n$)
variables $\vdash A_1, ..., A_n : Sequent$
begin
if $A_1 = neg(A_2)$ then inf($\{A_3, ..., A_n\}$)
else if$A_2 = B_1 \otimes ... \otimes B_m$ et
$A_1 = neg(B_1)$
then inf($\vdash B_2 \otimes ... \otimes B_m, ..., A_n$)
else inf($\vdash A_3, ..., A_n$)
eif
eif
end

(β) Et voici l'implémentation :

```
(define (inf S)(cond((null? S)
                     S)
                    ((null? (cdr S))
                     S)
                    ((and(> 2 (prof S))
                         (equal? (neg(car S))
                                 (cadr S)))
                         (inf (cddr S)))
                    ((and(<= 2 (prof S))
                         (equal? (neg(car S))

                                 (car(cadr S))))
                     (inf(append(cddr(cadr S))
                                (cddr S))))
                    (else (cons (car S)(inf(cdr S))))))
```

Les grammaires catégorielles classiques rigides

Une grammaire catégorielle classique rigide est une grammaire catégorielle rigide $G = \langle \Sigma, \delta, S \rangle$ dont le système de types est $MINCLL_{-\circ,\circ-}$ – toutes

choses égales par ailleurs. Aussi est-elle determinée par sa fonction $\delta : \Sigma \to \mathcal{P}_f(\mathcal{T})$, avec la convention que si $T \in I(m)$ (pour $m \in \Sigma$), on écrit $G : m \mapsto T$. G revient ainsi à écrire ces assignations de types que l'on réprésente par une liste `(m T)` – d'où : `G = ((m1 T1) ... (mk Tp))` avec possiblement des $i, j \in [1, k]$ tels que $i = j$. Ensuite on fera tout par rapport à cette grammaire. On se borne, au demeurant, aux grammaires rigides où à chaque mot ou lexème on fait correspondre au plus un type.

§1. Les grammaires.

```
(define (assig? A)(if (and (pair? A)
                           (= (long A) 2)
                           (atom (car A))
                           (type? (der A)))
                      #t
                      #f))

(define(gram_cat? G)(cond((not(pair? G)) #f)
                         ((null? G) #t)
                         (else (gram_cat_iter G #t))))
(define(gram_cat_iter G acc)(cond((null? G) acc)
                                 ((assig? (car G))
                                  (gram_cat_iter(cdr G)
                                                (and #t acc)))
                                 (else (gram_cat_iter
                                             (cdr G)
                                             (and #f acc)))))
```

§2. Le langage engendré par G, $L(G)$.

§2.1. L'alphabet de G.

```
(define (Sigma G)(if (not(gram_cat? G))
                     (void)
                     (Sigma_iter G '())))
(define (Sigma_iter G acc)(if (null? G)
                              (comp acc)
                              (Sigma_iter (cdr G)
```

```
                                        (cons
                                           (caar G)
                                            acc))))
```

§2.2. Les types $\mathcal{T}(G) \subset \mathcal{T}$ associés à G.

```
(define (Types_g G)(if (not(gram_cat? G)); donne Types(G).
                       (void)
                       (Types_g_iter G '())))
(define (Types_g_iter G acc)(if (null? G)
                                (comp acc)
                                (Types_g_iter (cdr G)
                                              (cons (der(car G))
                                                    acc))))
```

§2.3. Le type $T(m) \in \mathcal{T}$ associé à un lexème $m \in \Sigma$.

```
(define (Types_m a G)(cond((not(gram_cat? G))
                           '(void))
                          ((not(elt a (Sigma G)))
                          '(void))
                          (else(Types_m_iter a G '()))))

(define(Types_m_iter m G acc)(cond((null? G)
                                   acc)
                                  ((equal? m (caar G))
                                   (Types_m_iter m
                                                 (cdr G)
                                                 (cons
                                                    (der(car G))
                                                     acc)))
                                  (else (Types_m_iter m
                                                      (cdr G)
                                                       acc))))
```

§2.4. Les types $T(s) \subset \mathcal{T}$ associés à une chaîne $s \in \Sigma^*$.

Il reste à dire que les chaînes seront représentés, elles aussi, comm'il d'ailleurs habituel dans les dialectes du LISP, par des listes. $s = m_1...m_k$

deviendra la liste `(m1 ... mk)` (de longueur k). De même, la séquence $T(m_1), ..., T(m_k)$, deviendra la liste

```
((Types_m m1 G) ... (Types_m mk G)).
```

Donc :

```
(define (Types_s s G)(cond((not(gram_cat? G))
                           (void))
                          ((null? G)
                           '())
                          (else (Type_s_iter s G '()))))
(define (Type_s_iter s G acc)(if(null? s)
                                (mir-do acc)
                                (Type_s_iter (cdr s)
                                             G
                                             (append
                                              (Types_m
                                                 (car s) G)
                                               acc))))
```

L'analyse syntaxique

Détérminer si une chaîne $s = m_1...m_k$ appartenant à Σ^* se trouve dans $L(G)$ revient à savoir si le séquent $T(m_1), ..., T(m_k) \vdash S$ est démontrable (au sens de **mot-inf** et donc des listes qui vont avec).

```
(define (engendre? s G)(demon? (append (Types_s s G)
                               (list '=> 's))))
```

Quelques exemples

§1. Exemples concernant les séquents.

```
(sequent? '(a (a -o b) => c d)) ==> #t
(sequent_i? '(a (a -o b) => c d)) ==> #f (plus d'un type!)
(sequent_lam? '(=> (a -o a)) ==> #f (contexte vide!)
(sequent_i? '(=> (a -o a)) ==> #t
```

§2. Exemples concernant le fragment $MINCLL_{\{-\circ^g, \circ-^g\}}$.

```
(demon? '(x (x -o y) (y -o z) => z)) ==> #t
(demon? '(x (x -o y) ((y -o z) o- w) w => z)) ==> #t
(demon? '(y ((y -o z) o- w) w => z)) ==> #t
(demon? '(x (x -o (y o- z)) (z o- w) w => y)) ==> #f
--mauvais séquent!
(demon? '(sn ((sn -o s) o- sn) sn => s) ==> #t
(demon? '(sn sn ((sn -o s) o- sn))) ==> #f
--mauvais ordre des types!
```

§3. Un exemple de grammaire.

Une grammaire catégorielle classique rigide est determinée par sa fonction δ. Aussi G_1 (cf. **Exemple 3.0.2**), determinée par :

$$jean \mapsto SN$$

$$court \mapsto (SN \multimap S)$$

$$monte \mapsto ((SN \multimap S) \circ\!- SN)$$

$$le \mapsto (SN \circ\!- N)$$

$$escalier \mapsto N$$

Sera-t-elle représenté par la liste de couples :

```
(define G_1 '((jean sn)
              (court (sn -o s))
              (monte ((sn -o s) o- sn))
              (le (sn o- n))
              (escalier n)))
```

Qui en est bel et bien une :

```
(gram_cat? G_1) ==> #t
```

Et dont le langage engendré n'est pas vide :

```
(engendre? '(jean court) G_1) ==> #t
(engendre? '(court jean) G_1) ==> #f
--mauvais ordre des syntagmes!
(engendre? '(le escalier) G_1) ==> #f
--ce n'est pas une phrase!
(engendre? '(jean monte le escalier) ==> #t
```

Un petit effet de bord

Voici pour terminer un tout petit effet de bord. Il permet de créer l'environnement **PARSEUR** dans l'environnement DRSCHEME dans lequel on a testé et défini les algorithmes précédents. Il prend en entrée et des grammaires (classiques rigides) et des phrases et nous dit si cette dernière appartient au langage engendré correspondant.

```
(define(PARSEUR)
(begin
   (display "Savez-vous ecrire une
             grammaire categorielle classique k-rigide?")
   (newline)
   (let ((G (read)))
        (if (eq? #t (gram_cat? G))
            (begin
               (display "OK. Trouvez maintenant une
                         phrase engendree par elle.")
               (newline)
               (if (engendre? (read) G)
                   (display "Tres bien.")
                   (begin
                      (display "Non. Recommencez une
                                fois de plus!")
                      (newline)
                      (PARSEUR))))
            (begin
               (display "Non! Recommencez!")
               (newline)
               (PARSEUR))))))
```

Bibliographie

[1] Denis Bechet. Incremental parsing of lambek calculus using proof-net interphases. http ://www-lipn.univ-paris13.fr/∼bechet, 2002.

[2] Denis Bechet et Annie Foret. k-valued non-associative lambek grammars are learnable from function-argument structures. *Electronic Notes in Theoretical Computer Science*, (84), 2003.

[3] Jacques Chazarain. *Programmer avec SCHEME. De la pratique à la théorie*. International Thomson Publishing, 1996.

[4] Antoine Cornuejols et Yves Kodratoff. *Apprentissage artificiel. Concepts et algorithmes*. Eyrolles, 2002.

[5] John E. Hopcroft et Jeffrey D. Ullman. *An Introduction to Automata Theory, Languages and Computation*. Addison Welesley, 1979.

[6] Makoto Kanazawa. *Learnable Classes of Categorial Grammars*. CSLI, 1998.

[7] Makoto Kanazawa. Lambek calculus : Recognizing power and complexity. http ://www.illa.uva.nl/j50/contribs/kanazawa/∼1999, 1999.

[8] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65 :154–170, 1958.

[9] Christian Retore. Calcul de lambek et logique linéaire. *T.A.L.*, 37(2) :39–70.

[10] Christian Retore. Logique linéaire et syntaxe des langues. Technical report, Université de Nantes et INRIA, 2002. (Habilitation à diriger des recherches).

[11] Christian Retore et André Lecomte. Words as modules and modules as partial proof-nets in a lexicalised grammar. In Carlos Martin-Vide, editor, *Mathematical and Computational Analysis of Natural Language*, volume 45 of *Functional and Structural Linguistics*. John Benjamins, 1998.

[12] Christian Retore et François Lemarche. Proof nets for the lambek calculus – an overview. In *Proceedings of the 1996 Roma Workshop 'Proofs and Linguistic Categories – Applications of Logic to the Analysis and Implemenation of Natural Language'*, 1996.

[13] Takeshi Shinohara. Inductive inference from positive data is powerful. In *Annual Workshop on Computational Learning Theory*, volume 3, 1990.

[14] Takeshi Shinohara. Inductive inference of monotonic formal systems from positive data. In *International Workshop on Algorithmic Learning Theory*, number 1, 1990.

[15] Jacques Stern. *Fondements mathématiques de l'informatique*. Ediscience International, 1994.

[16] Hans-Joerg Tiede. *Deductive Systems and Grammars : Proofs as Grammatical Structures*. PhD thesis, Indiana University (Bloomington), 1999.

[17] Hans-Joerg Tiede. Proof theory and formal grammars. applications of normalization. In Benedikt Loewe Thoralf Rarsch, Wolfgang Malzkorn, editor, *Foundations of The Formal Sciences II : Applications of Mathematical Logic in Philosophy and Linguistics*. Kluwer, 2000.

[18] A. S. Troelstra et H. Schwichtenberg. *Basic Proof Theory*. Cambridge U. Press, 2000.