Mémoire

(Rapport de stage : page 16)

Introduction

présenter les différentes logiques, notamment logique intuitionniste

Table des matières

(mettre le plan du rapport de stage au début du rapport de stage)

1 Introduction aux calculs de séquents à travers LK pour la logique classique et LJ pour la logique intuitionniste

(calculs de séquents : outils / procédé de raisonnement ... dans l'introduction générale du mémoire ?)

Nous introduisons dans cette première partie les définitions dont nous aurons besoin sur les calculs de séquents, à travers l'exemple du calcul de séquents **LK**. Nous expliquons en quoi ce calcul défini par Gentzen correspond à la logique classique. Nous présentons ensuite le calcul **LJ**, que Gentzen a dérivé de **LK** afin de représenter la logique intuitionniste. Enfin, nous expliquons comment certaines propriétés des deux logiques considérées peuvent se comprendre en étudiant leur calcul de séquents respectif.

Les formules considérées dans cette partie sont construites à partir de constantes $\bot$ (*faux*) et $\top$ (*vrai*), de variables propositionnelles, du connecteur unaire $\neg$ (*non*), et des connecteurs binaires $\wedge$ (*et*), $\vee$ (*ou*) et $\to$ (*implique*). On s'intéresse en effet à la partie propositionnelle de chaque logique.

1.1 Des séquents et des règles

Un calcul de séquents se caractérise par sa propre définition d'un objet syntaxique appelé *séquent*, ainsi que par un ensemble de *règles* agissant sur les séquents.

Par exemple, pour le calcul de séquents **LK**, la définition d'un séquent est la suivante, et les règles sont données dans la figure 1.

Définition 1. *Un* **séquent** *de* **LK** *consiste en deux listes de formules* Γ *(les "hypothèses") et* Δ *(les "conclusions") ; on l'écrit* $\Gamma \vdash \Delta$.

Notation. X, Y désigne la liste obtenue en concaténant les listes X et Y ; si X ou Y est une formule, on la considère comme la liste à un élément correspondante.

Identité	**Coupure**
$\dfrac{}{A \vdash A}\,(id)$	$\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,(cut)$
Règles logiques	
$\dfrac{}{\Gamma, \bot \vdash \Delta}\,(\bot L)$	$\dfrac{}{\Gamma \vdash \top, \Delta}\,(\top R)$
$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\,(\neg L)$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\,(\neg L)$
$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge L)$	$\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\,(\wedge R)$
$\dfrac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\,(\vee L)$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\,(\vee R)$
$\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\,(\to L)$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\,(\to R)$
Règles structurelles	
$\dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\,(\textit{weakening } L)$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\,(\textit{weakening } R)$
$\dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\,(\textit{contraction } L)$	$\dfrac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\,(\textit{contraction } R)$
$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}\,(\textit{exchange } L)$	$\dfrac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2}\,(\textit{exchange } L)$

FIGURE 1 – Règles du calcul **LK**

Pour une règle $\dfrac{prem_1 \quad ... \quad prem_p}{concl}\,(\mathcal{R})$, $\mathcal{R}$ est le nom de la règle, $prem_1$, ... , $prem_p$ sont les ***prémisses***, et *concl* la ***conclusion***. Les prémisses et la conclusion sont des séquents où A, B sont des formules quelconques et Γ, Δ des listes de formules quelconques. L'idée est qu'une règle affirme : si toutes les prémisses sont vraies, alors la conclusion est aussi vraie. Cette idée est formalisée en 1.3.

On distingue deux grandes familles de règles. Les ***règles logiques*** remplacent une formule de la conclusion par une ou des formules plus simples. La formule remplacée, appelée *formule principale*, doit avoir une forme donnée en fonction de la règle. Les ***règles structurelles*** manipulent la structure du séquent en enlevant, dupliquant, déplaçant des formules dont on n'a pas besoin de connaître la forme. Elles dépendent du choix de structure du séquent : par exemple, si on représentait Γ et Δ par des *multiensembles*, c'est-à-dire des collections où le nombre d'occurrences est pris en compte mais pas l'ordre des éléments, on n'aurait pas besoin des règles d'échange *exchange L* et *exchange R*.

1.2 Variantes d'écriture de certaines règles

Cette présentation diffère de celle de Gentzen, mais elle en est suffisamment proche pour qu'on puisse quand même appeler ce calcul de séquents **LK**. Gentzen écrit deux règles $\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge L_1)$ et $\dfrac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge L_2)$ à la place de $\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge L)$, et

de même deux autres règles à la place de $\vee R$. On a cependant une équivalence grâce aux règles structurelles. Ci-dessous à gauche, on retrouve en effet la règle $\wedge L_1$ à partir de $\wedge L$ et *weakening* L, et on peut faire de même pour $\wedge L_2$. À droite, on retrouve $\wedge L$ à partir de $\wedge L_1$ et $\wedge L_2$ et *contraction* L. On s'autorise à faire agir $\wedge L_1$ sur une formule qui n'est pas la dernière de la liste : cela est possible en appliquant plusieurs fois la règle *exchange* L, ce qu'on a pas fait explicitement par souci de lisibilité.

$$\dfrac{\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A, B \vdash \Delta}\,(weakeningL)}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge L) \qquad \dfrac{\dfrac{\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B, B \vdash \Delta}\,(\wedge L_1)}{\Gamma, A \wedge B, A \wedge B \vdash \Delta}\,(\wedge L_2)}{\Gamma, A \wedge B \vdash \Delta}\,(contractionL)$$

1.3 Prouvabilité d'un séquent

Une ***instance*** d'une règle $\mathcal{R}$ a la même forme que la règle : $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$, mais ici les σ_i et σ sont des séquents connus explicitement ; bien entendu il faut qu'il s'agisse de séquents qui correspondent à la forme donnée par la définition de la règle. Une ***preuve*** (ou ***arbre de preuve***) est un arbre dont les nœuds sont étiquetés par un séquent et une règle et ont la même arité que le nombre de prémisses de la règle, et tel que : pour tout nœud de séquent σ et de règle $\mathcal{R}$, si σ_1, ... , σ_p sont les séquents associés à chacun de ses fils respectivement, alors $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$ est une instance de $\mathcal{R}$. Les feuilles d'un tel arbre sont les nœuds auxquels est associé un axiome.

Définition 2. *Un séquent σ est* **prouvable** dans un calcul de séquents *s'il existe un arbre de preuve tel que le séquent associé à la racine est σ. De manière équivalente, on peut définir l'ensemble des séquents prouvables comme le plus petit ensemble vérifiant : pour toute instance $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$ d'une règle, si pour tout i, σ_i est prouvable, alors σ est prouvable (en particulier pour toute instance $\dfrac{}{\sigma}\,(\mathcal{A})$ d'un axiome $\mathcal{A}$, σ est prouvable).*

1.4 Lien avec une logique, interprétation des séquents

Un calcul de séquents est généralement associé à une logique, à travers une propriété similaire à la suivante, qui concerne **LK** et la logique classique.

Proposition 3. *Une formule A est valide en logique classique si, et seulement si, le séquent $\vdash A$ est prouvable par le calcul* **LK** *(on écrit $\vdash A$ pour $\emptyset \vdash A$, $\emptyset$ désignant ici la liste vide).*

Les séquents sont des objets syntaxiques pratiques à manipuler à l'aide de règles. Cependant, ils ont souvent une interprétation dans la logique considérée : par exemple pour **LK**, un séquent $\Gamma \vdash \Delta$ s'interprète comme une formule de logique classique grâce à la propriété suivante. Il signifie ainsi : "si on suppose toutes les formules de Γ, on peut montrer au moins une formule de Δ", d'où les appellations "hypothèses" pour les formules de Γ et "conclusions" pour celles de Δ.

Proposition 4. *Un séquent $\Gamma \vdash \Delta$ est prouvable par le calcul* **LK** *si, et seulement si, la formule $\left(\bigwedge_{G \in \Gamma} G\right) \to \left(\bigvee_{D \in \Delta} D\right)$ est valide en logique classique.*

1.5 Calcul des séquents LJ et logique intuitionniste

Gentzen a dérivé le calcul **LJ** de **LK** dans le but de correspondre à la logique intuitionniste. La modification apportée à **LK** pour cela est simple et efficace : on restreint les séquents à ceux qui ont exactement une formule à droite, c'est-à-dire dans Δ avec les notations habituelles ; les règles sont adaptées en conséquence. La principale conséquence est l'impossibilité d'utiliser les règles structurelles à droite du séquent. La nouvelle définition d'un séquent est donc la suivante, et les règles sont données dans la figure 2.

Définition 5. *Un* **séquent** *de* **LJ** *consiste en une liste de formules* Γ *(les "hypothèses") et une formule* D *(la "conclusion") ; on l'écrit* $\Gamma \vdash D$.

Identité	**Coupure**
$\dfrac{}{A \vdash A}\,(id)$	$\dfrac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D}\,(cut)$
$\dfrac{}{\Gamma, \bot \vdash D}\,(\bot L)$	**Règles logiques**
$\dfrac{\Gamma, A, B \vdash D}{\Gamma, A \wedge B \vdash D}\,(\wedge L)$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,(\wedge R)$
$\dfrac{\Gamma, A \vdash D \quad \Gamma, B \vdash D}{\Gamma, A \vee B \vdash D}\,(\vee L)$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,(\vee R_1)$ $\dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,(\vee R_2)$
$\dfrac{\Gamma \vdash A \quad \Gamma, B \vdash D}{\Gamma, A \to B \vdash D}\,(\to L)$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,(\to R)$
Règles structurelles	$\dfrac{\Gamma \vdash D}{\Gamma, A \vdash D}\,(weakening\ L)$
$\dfrac{\Gamma, A, A \vdash D}{\Gamma, A \vdash D}\,(contraction\ L)$	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash D}{\Gamma_1, B, A, \Gamma_2 \vdash D}\,(exchange\ L)$

FIGURE 2 – Règles du calcul **LJ**

Les règles structurelles à droite de **LK**, qui changent le nombre de formules à droite ou en nécessitent au moins deux, disparaissent. La règle $\dfrac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\,(\vee L)$ ne s'adapte pas non plus aux séquents de **LJ** ; on la remplace par deux règles comme en 1.2.

Le connecteur $\neg$ disparaît également. En logique intuitionniste, $\neg A$ a bien un sens, mais on préfère considérer qu'il s'agit d'une simple notation pour $A \to \bot$. Cette décision est justifiée par le fait qu'en logique intuitionniste, $\neg$ n'est pas involutif : la formule $\neg\neg A$ n'est pas équivalente à A.

Comme **LK** avec la logique classique, **LJ** est liée à la logique intuitionniste : **LJ** permet de caractériser la prouvabilité d'une formule en logique intuitionniste. Un séquent de **LJ** peut également s'interpréter en logique intuitionniste.

Proposition 6. *Une formule* A *est prouvable en logique intuitionniste si, et seulement si, le séquent* $\vdash A$ *est prouvable par le calcul* **LJ**.

Proposition 7. *Un séquent* $\Gamma \vdash D$ *est prouvable par le calcul* **LJ** *si, et seulement si, la formule* $\left(\bigwedge_{G \in \Gamma} G\right) \to D$ *est prouvable en logique intuitionniste.*

1.6 Une différence entre logique classique et logique intuitionniste

C'est la possibilité d'avoir plusieurs formules dans la partie droite du séquent qui permet de prouver davantage de séquents dans **LK** que dans **LJ**. On comprend ainsi la différence entre le "ou" classique et le "ou" intuitionniste. En logique classique, prouver $A \vee B$, c'est prouver le séquent $\vdash A, B$: les deux formules sont encore présentes. Un bon exemple est la preuve ci-contre dans **LK** du principe du tiers exclu $A \vee \neg A$, qu'on écrit $A \vee (A \to \bot)$ pour mieux comparer à la logique intuitionniste où le $\neg$ se comporte différemment. Si on peut appliquer l'axiome *id* à la formule A (ce qui nécessite deux occurrences distinctes de la formule, une de chaque côté), c'est bien parce qu'on a conservé les deux parties de la formule initiale. Tandis qu'en logique intuitionniste, pour prouver $A \vee B$ c'est-à-dire $\vdash A \vee B$, les seules règles applicables sont $\vee R_1$ et $\vee R_2$: il faut donc prouver $\vdash A$ ou prouver $\vdash B$; une fois qu'on a choisi lequel on va prouver, on n'a plus accès à l'autre. Ainsi, on ne peut pas prouver $A \vee (A \to \bot)$, car ni le séquent $\vdash A$ ni le séquent $\vdash (A \to \bot)$ n'est prouvable.

Remarque. Même avec la présentation de **LK** de Gentzen où on a deux règles $\vee R_1$ et $\vee R_2$ au lieu de notre unique règle $\vee R$, dans la preuve ci-contre de $A \vee (A \to \bot)$, on utilise une contraction à droite afin de conserver à la fois A et $A \to \bot$. Le principe est le même que dans les équivalences d'écriture des règles en 1.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,(\text{id})}{A \vdash A, \bot}\,(\text{weak. R})}{\vdash A\,,\ A \to \bot}\,(\to R)}{\vdash A\,,\ A \vee (A \to \bot)}\,(\vee R_2)}{\vdash A \vee (A \to \bot)\,,\ A \vee (A \to \bot)}\,(\vee R_1)}{\vdash A \vee (A \to \bot)}\,(\textit{contr. } R)$$

Finalement, on peut considérer que la logique intuitionniste est obtenue à partir de la logique classique en interdisant certaines règles structurelles. Une autre logique couramment étudiée, la logique linéaire, s'obtient à partir de la logique classique en conservant toutes les règles structurelles, mais en restreignant fortement leur utilisation.

2 Logique linéaire (LL) et logique linéaire intuitionniste (ILL)

La logique linéaire (LL) est une extension de la logique classique. Elle présente un grand un intérêt des points de vue logique aussi bien qu'informatique, notamment parce qu'elle contient une notion de "ressources" qui ne peuvent pas être utilisées inconsidérément. On peut considérer qu'elle est obtenue à partir de la logique classique en limitant l'usage des règles structurelles, si bien que de nouvelles nuances apparaissent, pour lesquelles sont introduits de nouveaux connecteurs.

Voici un calcul de séquents pour la logique linéaire. Les séquents sont les mêmes que ceux de **LK** : deux listes de formules Γ et Δ, avec la notation $\Gamma \vdash \Delta$. Les connecteurs, différents, sont donnés dans la figure 3. Les formules sont alors construites à partir de variables propositionnelles, de ces connecteurs, et de constantes qui sont chacune un élément neutre associé à un connecteur. Par exemple $\mathbf{1}$ est l'élément neutre pour $\otimes$, c'est-à-dire que pour toute formule A, $A \otimes \mathbf{1} = \mathbf{1} \otimes A = A$. Les règles sont données dans la figure 4. Des explications sur ces connecteurs et ces règles sont proposées dans la sous-section 2.1, puis une interprétation en terme de "ressources" dans la sous-section 2.2. Enfin, on présente la logique linéaire intuitionniste (ILL).

(sources)

Symbole	Nom	Élément neutre	
		Symbole	Nom
$\otimes$	*tenseur / produit tensoriel*	**1**	*un*
&	*avec*	$\top$	*top*
$\oplus$	*plus*	**0**	*zéro*
$\parr$	*produit parallèle*	$\bot$	*bot*
$\multimap$	*implication linéaire*	pas d'élément neutre	

Connecteurs binaires

Symbole	Nom
$\neg$	*non*
!	*bang*
?	*why not*

Connecteurs unaires

FIGURE 3 – Les connecteurs de la logique linéaire

2.1 Limitation des règles structurelles, nouveaux connecteurs, modalités

L'usage des règles structurelles est restreint à certains types de formules qu'on verra plus loin. Une conséquence importante est qu'on n'a plus l'équivalence entre les deux écritures possibles des règles liées à $\wedge$, vues en 1.2. C'est pourquoi on introduit deux connecteurs distincts $\otimes$ et &, qui pourraient tous deux se lire "et" en première approximation, correspondant chacun à une définition possible de $\wedge$ dans **LK** (cf. les règles liées à ces connecteurs dans la figure 4). De même, le $\vee$ de logique classique est séparé en deux connecteurs $\oplus$ et $\parr$. On souhaite tout de même garder des règles structurelles, dont l'usage est seulement limité. On introduit pour cela des connecteurs unaires *modaux*, ou *modalités*, ! et ?. Les règles structurelles peuvent seulement être appliquées à des formules comportant un de ces connecteurs : ! si la formule est à gauche, ? si elle est à droite. On a aussi des règles d'introduction de ces connecteurs, où par exemple $!\Gamma$ représente une liste dont toutes les formules sont de la forme $!A$. On retrouve des relations entre $\otimes$ et & grâce à ces modalités : par exemple $(!A) \otimes (!B) = !(A\&B)$, qu'on discute dans la prochaine sous-section.

2.2 Interprétation : une idée de "ressources"

La logique linéaire est une logique de "ressources". L'implication linéaire $A \multimap B$ peut en effet se comprendre comme "on peut dépenser un objet de type A pour obtenir un objet de type B". Ici, "type" est simplement un mot du langage courant et non un terme mathématique ou informatique.La formule $A \otimes B$ représente la possession à la fois d'un objet de type A et d'un autre objet de type B. On comprend alors la notion de "dépense" en remarquant que, si on a $A \multimap B$ et $A \multimap C$, on n'obtient pas pour autant $A \multimap B \otimes C$ car un seul objet de type A ne peut pas être dépensé deux fois ; en revanche on obtient bien $A \otimes A \multimap B \otimes C$. On n'obtient pas non plus $A \otimes A \otimes A \multimap B \otimes C$ car un objet de type A ne serait pas dépensé ; cela est lié au fait que la formule $A \multimap \mathbf{1}$ n'est pas universellement valide, où **1** est l'élément neutre pour $\otimes$. Une analogie courante et pertinente compare ce fragment de la logique linéaire aux équations de réaction en chimie, avec la maxime de Lavoisier bien connue "Rien ne se perd, rien ne se crée, tout se transforme". Par exemple, l'équation $CH_4 + 2O_2 \longrightarrow CO_2 + 2H_2O$ peut être fidèlement représentée par la formule $!(CH_4 \otimes O_2 \otimes O_2 \multimap CO_2 \otimes H_2O \otimes H_2O)$, où CH_4 etc. sont considérés comme des constantes ; la modalité !, discutée plus loin, signifie qu'on peut appliquer cette réaction autant de fois qu'on le souhaite.

Intéressons-nous maintenant à &, l'autre connecteur issu du $\wedge$ du logique classique. La formule $A\&B$ représente la possibilité de posséder, au choix, un objet de type A ou un objet de type B. À partir de $A \multimap B$ et $A \multimap C$, on obtient bien $A \multimap B\&C$: on a le choix de

$\dfrac{}{A \vdash A}\ (id)$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut)$
$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\neg L)$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ (\neg R)$
$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}\ (\otimes L)$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma' \vdash B, \Delta'}{\Gamma \vdash A \otimes B, \Delta, \Delta'}\ (\otimes R)$
$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash \Delta}\ (\&L_1) \qquad \dfrac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash \Delta}\ (\&L_2)$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \& B, \Delta}\ (\&R)$
$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta}\ (\oplus L)$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ (\oplus R_1) \qquad \dfrac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ (\oplus R_2)$
$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'}\ (\parr L)$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta}\ (\parr R)$
$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \multimap B \vdash \Delta, \Delta'}\ (\multimap L)$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}\ (\multimap R)$
$\dfrac{\Gamma \vdash \Delta}{\Gamma, \mathbf{1} \vdash \Delta}\ (\mathbf{1}L) \qquad \dfrac{}{\mathbf{0} \vdash}\ (\mathbf{0}L)$	$\dfrac{}{\vdash \mathbf{1}}\ (\mathbf{1}R) \qquad \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \mathbf{0}, \Delta}\ (\mathbf{0}R)$
$\dfrac{}{\Gamma, \bot \vdash \Delta}\ (\bot L)$	$\dfrac{}{\Gamma \vdash \top, \Delta}\ (\top R)$
$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}\ (!L) \qquad \dfrac{!\Gamma, A \vdash ?\Delta}{!\Gamma, ?A \vdash ?\Delta}\ (?L)$	$\dfrac{!\Gamma \vdash A, ?\Delta}{!\Gamma \vdash !A, !\Delta}\ (!R) \qquad \dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash ?A, \Delta}\ (?R)$
$\dfrac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}\ (weakening\ L)$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash ?A, \Delta}\ (weakening\ R)$
$\dfrac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}\ (contraction\ L)$	$\dfrac{\Gamma \vdash ?A, ?A, \Delta}{\Gamma \vdash ?A, \Delta}\ (contraction\ R)$
$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}\ (exchange\ L)$	$\dfrac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2}\ (exchange\ R)$

FIGURE 4 – Les règles d'un calcul de séquents pour LL

la façon dont on dépense l'objet de type A. Le fait qu'on ne possède finalement qu'un seul objet, de type A ou de type B, peut donner l'impression qu'il s'agit d'une disjonction. Le point important est la possibilité de choisir le type parmi A et B. Cela entraîne qu'on peut prouver $A \& B \multimap A$ et $A \& B \multimap B$, mais pas $A \multimap A \& B$, donc il s'agit bien d'une conjonction. C'est $A \oplus B$ qui représente la possession d'un objet dont on sait que le type est A ou B, mais on ne peut pas choisir lequel des deux. Il s'agit cette fois d'une disjonction : on peut prouver $A \multimap A \oplus B$ mais pas $A \oplus B \multimap A$.

La modalité $!A$ signifie qu'on peut posséder un objet de type A autant de fois qu'on le souhaite, y compris zéro. Un exemple significatif est l'égalité $(!A) \otimes (!B) = !(A \& B)$: posséder simultanément autant d'objets qu'on veut de type A et autant d'objets qu'on veut de type B, revient à avoir autant de fois qu'on veut la possibilité de choisir de posséder un objet de type A ou un objet de type B.

$\parr$, ?

2.3 Logique linéaire intuitionniste

La logique linéaire intuitionniste (ILL) peut être associée à un calcul dont les séquents sont les mêmes que ceux de **LJ** : une liste de formules Γ et une formule D, avec la notation $\Gamma \vdash D$; les règles sont données dans la figure 5.

La logique linéaire intuitionniste (ILL) est obtenue à partir de la logique linéaire de la même manière que la logique intuitionniste est obtenue à partir de la logique classique : en se restreignant aux séquents avec une unique formule à droite. On ne garde que les connecteurs binaires $\otimes$, $\multimap$ et & avec leur élément neutre éventuel, et le connecteur modal !. Le $\neg$ disparaît pour la même raison qu'en logique intuitionniste. (?) Les autres connecteurs sont liés à des règles qui ne s'adaptent pas aux séquents avec une unique formule à droite, à l'exception de $\oplus$ qui pourrait être conservé sans problème. Néanmoins, on se conforme à une tradition qui consiste à parler de *logique linéaire intuitionniste* lorsqu'on ne considère pas de connecteur $\oplus$, et de *logique linéaire intuitionniste à produits finis* lorsqu'on le rajoute.

(à faire)

FIGURE 5 – Les règles d'un calcul de séquents pour ILL

3 Le procédé d'élimination de la coupure (p.é.c.) pour ILL

coupure intéressante : transitivité, conclusion prouvée à son tour utilisée comme hypothèse

souvent, coupure non nécessaire : correct et surtout complet sans. mais démonstration souvent difficile

démonstration également intéressante. constructive : transformer n'importe quelle preuve en preuve du même séquent sans coupure. pour cela, énumération de transformations élémentaires : agissent sur petite partie de la preuve.

ces transformations permettent de définir une relation d'équivalence

4

(On suppose qu'on a présenté la logique linéaire et son calcul de séquents, et le procédé d'élimination de la coupure (qu'on abrège pour l'instant parfois en p.é.c., mais cela peut changer). On distingue le p.é.c. "strict" avec seulement les transformations nécessaires pour le théorème d'élimination de la coupure, du p.é.c. "large" où on a rajouté les transformations qui font que alpha, lambda, rho deviennent des isomorphismes et donc permettent d'obtenir vraiment une catégorie monoïdale.)

(source : chapitre 3)

4.1 Invariant modulaire et catégorie

Notation. Soit p une preuve du séquent σ. On écrit $\dfrac{p}{\sigma}$ car souvent, il est intéressant d'expliciter σ pour prolonger p en une preuve plus complexe d'un autre séquent.

À chaque preuve p, on associe une ***dénotation*** $[p]$. On veut que cela constitue un invariant selon l'élimination de la coupure : deux preuves on la même dénotation si, et seulement si,

une peut être obtenue en appliquant à l'autre des transformations autorisée par la procédure d'élimination de la coupure. Ceci est motivé par une analogie avec la théorie des nœuds, où les invariants sont relatifs aux transformations de Reidemeister. (expliquer ? ne pas en parler ?)

On demande également la propriété suivante. Soit A, B, C des formules, et p_1 et p_2 des preuves de $A \vdash B$ et $B \vdash C$ respectivement. La règle de coupure fournit immédiatement la preuve p de $A \vdash C$ ci-contre. On veut que sa dénotation $[p]$ se déduise à partir de $[p_1]$ et $[p_2]$. On introduit pour cela la **loi de composition** $\circ$ telle que $[p] = [p_2] \circ [p_1]$. On dit alors que l'invariant est *modulaire*.

$$\dfrac{\overset{p_1}{\overset{\cdots\cdots}{A \vdash B}} \quad \overset{p_2}{\overset{\cdots\cdots}{B \vdash C}}}{A \vdash C}\,(cut) \qquad p$$

On remarque que la loi de composition $\circ$ est **associative** et présente pour chaque formule une **identité** à gauche et à droite. En effet, soit A, B, C, D des formules, et p_1, p_2, p_3 des preuves respectives de $A \vdash B$, $B \vdash C$, $C \vdash D$. On peut en déduire deux preuves de $A \vdash D$:

$$\dfrac{\dfrac{\overset{p_1}{\overset{\cdots\cdots}{A \vdash B}} \quad \overset{p_2}{\overset{\cdots\cdots}{B \vdash C}}}{A \vdash C}\,(cut) \quad \overset{p_3}{\overset{\cdots\cdots}{C \vdash D}}}{A \vdash D}\,(cut) \qquad \text{et} \qquad \dfrac{\overset{p_1}{\overset{\cdots\cdots}{A \vdash B}} \quad \dfrac{\overset{p_2}{\overset{\cdots\cdots}{B \vdash C}} \quad \overset{p_3}{\overset{\cdots\cdots}{C \vdash D}}}{B \vdash D}\,(cut)}{A \vdash D}\,(cut)$$

qui sont équivalentes d'après le procédé d'élimination de la coupure. Cela signifie précisément que $([p_1] \circ [p_2]) \circ [p_3] = [p_1] \circ ([p_2] \circ [p_3])$. L'identité pour une formule A est la dénotation de la preuve $\dfrac{}{A \vdash A}\,(id)$; on note cette dénotation id_A. Soit p une preuve de $A \vdash B$, les deux preuves $\dfrac{\dfrac{}{A \vdash A}\,(id) \quad \overset{p}{\overset{\cdots\cdots}{A \vdash B}}}{A \vdash B}\,(cut)$ et $\overset{p}{\overset{\cdots\cdots}{A \vdash B}}$ sont équivalentes selon l'élimination de la coupure, ce qui signifie que $[p] \circ id_A = [p]$. De même, pour p une preuve de $B \vdash A$, on a $id_A \circ [p] = [p]$.

Ces propriétés sur les dénotations permettent d'utiliser le formalisme des catégories.

Définition 8. *Une* **catégorie** *consiste en une collection d'*objets *et une collection de* morphismes*, cette dernière munie d'une opérations binaire partielle* $\circ$ *appelée* composition*, avec les propriétés suivantes.*

- *À chaque morphisme f est associé un couple d'objets (A, B) ; on écrit $f : A \to B$. On dit que $A \to B$ est le* type *de f, et que f est un morphisme* de A vers B*, et encore que A est le* domaine *ou la* source *de f, et B le* codomaine *ou la* cible *de f.*
- *Pour tous objets A, B, C et morphismes f, g tels que $f : A \to B$ et $g : B \to C$, le morphisme $g \circ f$ existe et $g \circ f : A \to C$.*
- ***Identité.*** *Pour tout objet A, il existe un morphisme particulier $id_A : A \to A$ appelé* identitié sur A*, tel que pour tout objet B, pour tout $f : B \to A$, $id_A \circ f = f$ et pour tout $g : A \to B$, $g \circ id_A = g$.*
- ***Associativité.*** *Pour tous $f : A \to B$, $g : B \to C$, $h : C \to D$, on a $h \circ (g \circ f) = (h \circ g) \circ f$.*

Notations. *Soit $\mathcal{C}$ une catégorie, on note $Obj(\mathcal{C})$ la classe de ses objets et $Hom(\mathcal{C})$ celle de ses morphismes. Pour des objets A et B, on note $Hom_{A \to B}(\mathcal{C})$ la classe des morphismes de type $A \to B$. On peut omettre l'argument $\mathcal{C}$ s'il n'y a pas d'ambiguïté, par exemple si on travaille sur une seule catégorie.*

Les dénotations des preuves s'organisent sous la forme de la catégorie suivante, qu'on appellera $\mathcal{CP}$. (choix du nom ?) À chaque formule A on associe une dénotation $[A]$. Les

dénotations des formules, deux à deux distinctes, constituent les objets de la catégorie. Les morphismes sont des dénotations de preuves. Les morphismes de $[A]$ vers $[B]$ sont les dénotations des preuves de $A \vdash B$; si ce séquent n'est pas prouvable, il n'y en a pas. Comme on l'a vu, la composition $\circ$ est bien associative, et l'identité sur $[A]$ est le morphisme id_A. On notera aussi A la dénotation de la formule A, sauf lorsqu'on souhaite insister sur le fait qu'il s'agit d'une dénotation. Cela permet d'alléger l'écriture et de retrouver la notation employée dans les définitions sur les catégories.

Cette définition ne tient compte que des dénotations de preuves où le séquent prouvé a une unique formule à gauche et une unique formule à droite ; les autres séquents des preuves peuvent avoir n'importe quelle forme. Cette restriction est conservée dans (quelques sous-sections ? tout le reste de la partie ? toujours ? Est-ce qu'on peut étendre ce formalisme à n'importe quelle preuve de la logique intuitionniste linéaire grâce au théorème de MacLane, cf. mail ?)

4.2 Produit tensoriel et bifoncteur

Le connecteur de logique linéaire $\otimes$, appelé *produit tensoriel*, induit un opérateur sur les dénotations de formules, aussi noté $\otimes$: on pose $[A] \otimes [B] = [A \otimes B]$. On souhaite étendre cet opérateur aux dénotations de preuves. Pour cela, on remarque qu'à partir de preuves p_1 de $A_1 \vdash B_1$ et p_2 de $A_2 \vdash B_2$, on peut déduire la preuve p ci-contre de $A_1 \otimes A_2 \vdash B_1 \otimes B_2$. On définit alors $[p_1] \otimes [p_2] = [p]$.

$$\dfrac{\dfrac{\overset{p_1}{\cdots}\quad\overset{p_2}{\cdots}}{A_1 \vdash B_1 \quad A_2 \vdash B_2}\,(\otimes R)}{A_1, A_2 \vdash B_1 \otimes B_2}\,(\otimes L)}{A_1 \otimes A_2 \vdash B_1 \otimes B_2}$$
$$p$$

L'opérateur est bien défini : si on a deux autres preuves p'_1 et p'_2 telles que $[p'_1] = [p_1]$ et $[p'_2] = [p_2]$, et si p' est la preuve obtenue à partir de p'_1 et p'_2 selon le procédé utilisé pour construire p, alors $[p'] = [p]$. En effet, le procédé d'élimination de la coupure autorise évidemment à remplacer la preuve p_1 apparaissant dans la preuve p par une preuve p'_1 qui lui est équivalente.

Intéressons-nous à la compatibilité de $\otimes$ avec $\circ$. Soit des morphismes de $f_1 : A_1 \to B_1$, $f_2 : A_2 \to B_2$, $g_1 : B_1 \to C_1$, $g_2 : B_2 \to C_2$. Pour chaque morphisme f, soit $p(f)$ une preuve de dénotation f. Les deux preuves suivantes sont équivalentes selon le p.é.c., ce qui signifie que $(g_1 \otimes g_2) \circ (f_1 \otimes f_2) = (g_1 \circ f_1) \otimes (g_2 \circ f_2)$.

$$\dfrac{\dfrac{\dfrac{\overset{p(f_1)}{\cdots}\quad\overset{p(f_2)}{\cdots}}{A_1 \vdash B_1 \quad A_2 \vdash B_2}\,(\otimes R) \quad \dfrac{\overset{p(g_1)}{\cdots}\quad\overset{p(g_2)}{\cdots}}{B_1 \vdash C_1 \quad B_2 \vdash C_2}\,(\otimes R)}{A_1, A_2 \vdash B_1 \otimes B_2}\,(\otimes L) \quad \dfrac{}{B_1, B_2 \vdash C_1 \otimes C_2}\,(\otimes L)}{A_1 \otimes A_2 \vdash B_1 \otimes B_2 \quad B_1 \otimes B_2 \vdash C_1 \otimes C_2}\,(cut)}{A_1 \otimes A_2 \vdash C_1 \otimes C_2}$$

$$\dfrac{\dfrac{\dfrac{\overset{p(f_1)}{\cdots}\quad\overset{p(g_1)}{\cdots}}{A_1 \vdash B_1 \quad B_1 \vdash C_1}\,(cut) \quad \dfrac{\overset{p(f_2)}{\cdots}\quad\overset{p(g_2)}{\cdots}}{A_2 \vdash B_2 \quad B_2 \vdash C_2}\,(cut)}{A_1 \vdash C_1 \quad A_2 \vdash C_2}\,(\otimes R)}{A_1, A_2 \vdash C_1 \otimes C_2}\,(\otimes L)}{A_1 \otimes A_2 \vdash C_1 \otimes C_2}$$

On a également l'égalité $id_{[A]\otimes[B]} = id_{[A]} \otimes id_{[B]}$ en raison de l'équivalence des preuves

$$\dfrac{}{A \otimes B \vdash A \otimes B}\,(id) \quad \text{et} \quad \dfrac{\dfrac{\dfrac{}{A \vdash A}\,(id) \quad \dfrac{}{B \vdash B}\,(id)}{A, B \vdash A \otimes B}\,(\otimes R)}{A \otimes B \vdash A \otimes B}\,(\otimes L) \quad .$$

Ces propriétés signifient que l'opérateur $\otimes$ constitue un **bifoncteur** sur $\mathcal{CP}$, c'est-à-dire un foncteur de $\mathcal{CP} \times \mathcal{CP}$ vers $\mathcal{CP}$, avec les définitions suivantes.

Définition 9. *Soit $\mathcal{C}$ et $\mathcal{D}$ des catégories. Un* **foncteur** *$F : \mathcal{C} \longrightarrow \mathcal{D}$ de $\mathcal{C}$ vers $\mathcal{D}$ consiste en une application des objets de $\mathcal{C}$ vers les objets de $\mathcal{D}$ et une application des morphismes de $\mathcal{C}$ vers les morphismes de $\mathcal{D}$, notées toutes deux F par abus d'écriture, tel que*

- *Pour tous objets A, B et morphisme $f : A \to B$ de $\mathcal{C}$, on a $F(f) : F(A) \to F(B)$.*

- *Pour tous morphismes $f : A \to B$ et $g : B \to C$ de $\mathcal{C}$, on a $F(g \circ f) = F(g) \circ F(f)$.*
- *Pour tout objet A de $\mathcal{C}$, on a $F(id_A) = id_{F(A)}$.*

Définition 10. *Soit $\mathcal{C}$ et $\mathcal{D}$ des catégories. On définit la* **catégorie produit** *$\mathcal{C} \times \mathcal{D}$, où $Obj(\mathcal{C} \times \mathcal{D}) = Obj(\mathcal{C}) \times Obj(\mathcal{D})$ et $Hom_{(A,A')\to(B,B')} = Hom_{A\to B}(\mathcal{C}) \times Hom_{A'\to B'}(\mathcal{D})$.*

4.3 Catégorie monoïdale

La catégorie $\mathcal{CP}$ ayant été munie du bifoncteur $\otimes$, on se demande s'il s'agit d'une catégorie monoïdale, dont la définition est donnée après deux définitions auxiliaires.

Définition 11. *Soit $\mathcal{C}$ et $\mathcal{D}$ des catégories. Soit $F, G : \mathcal{C} \longrightarrow \mathcal{D}$ des foncteurs de $\mathcal{C}$ vers $\mathcal{D}$. Une* **transformation naturelle** *θ de F vers G est une famille $\big(\theta_A : F(A) \to G(A) \big)_{A \in Obj(\mathcal{C})}$ de morphismes de $\mathcal{D}$, indexée par les objets de $\mathcal{C}$, telle que le diagramme suivant commute dans $\mathcal{D}$ pour tout morphisme $f : A \to B$ de $\mathcal{C}$*

$$\begin{array}{ccc} F(A) & \xrightarrow{\theta_A} & G(A) \\ \big\downarrow{\scriptstyle F(f)} & & \big\downarrow{\scriptstyle G(f)} \\ F(B) & \xrightarrow{\theta_B} & G(B) \end{array}$$

On note $\theta : F \Rightarrow G$, voire $\theta : F \Rightarrow G : \mathcal{C} \longrightarrow \mathcal{D}$.

Définition 12. *Soit une catégorie, un morphisme $f : A \to B$ est un* **isomorphisme** *s'il existe un morphisme $f^{-1} : B \to A$ tel que $f^{-1} \circ f = id_A$ et $f \circ f^{-1} = id_B$.*

Définition 13. *Une* **catégorie monoïdale** *est une catégorie $\mathcal{C}$ munie d'un bifoncteur $\otimes : \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ (pour lequel on utilise une notation infixe) avec d'un objet particulier e, telle qu'il existe des* ***isomorphismes naturels***

- *$\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$, permettant de parler d'associativité ;*
- *$\lambda_A : e \otimes A \to A$;*
- *$\rho_A : A \otimes e \to A$, permettant avec le précédent d'appeler e l'objet* ***unité*** *;*

et tel que les diagrammes suivants commutent pour tous objets A, B, C, D.

$$\begin{array}{ccc} & (A \otimes B) \otimes (C \otimes D) & \\ {\scriptstyle \alpha_{A\otimes B,C,D}}\nearrow & & \searrow{\scriptstyle \alpha_{A,B,C\otimes D}} \\ ((A \otimes B) \otimes C) \otimes D & & A \otimes (B \otimes (C \otimes D)) \\ \big\downarrow{\scriptstyle \alpha_{A,B,C}\otimes id_D} & & \big\uparrow{\scriptstyle id_A \otimes \alpha_{B,C,D}} \\ (A \otimes (B \otimes C)) \otimes D & \xrightarrow{\alpha_{A,B\otimes C,D}} & A \otimes ((B \otimes C) \otimes D) \end{array}$$

$$\begin{array}{ccc} (A \otimes e) \otimes B & \xrightarrow{\alpha_{A,e,B}} & A \otimes (e \otimes B) \\ & {\scriptstyle \rho_A \otimes id_B}\searrow \quad \swarrow{\scriptstyle id_A \otimes \lambda_B} & \\ & A \otimes B & \end{array}$$

Ces diagrammes sont appelés diagrammes pentagonal *et* triangulaire*. Leur commutativité est la* propriété de cohérence*.*

Précision. *On parle d'isomorphismes naturels car il y a bien des transformations naturelles associées. En fait la condition sur α est $\alpha : F \Rightarrow G : \mathcal{C} \times \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ avec $F : (x, y, z) \mapsto (x \otimes y) \otimes z$ et $G : (x, y, z) \mapsto x \otimes (y \otimes z)$ où x, y, z sont trois objets ou trois morphismes de $\mathcal{C}$. On écrit $\alpha_{A,B,C}$ pour $\alpha_{(A,B,C)}$. Les $\alpha_{A,B,C}$ sont bien des morphismes de $\mathcal{C}$, qui est la catégorie*

d'arrivée de F et G. On veut aussi $\lambda : (e \otimes \bullet) \Rightarrow Id_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{C}$ où $(e \otimes \bullet) : \begin{cases} A \mapsto e \otimes A \\ f \mapsto id_e \otimes f \end{cases}$ et où $Id_{\mathcal{C}}$ est le foncteur identité sur $\mathcal{C}$. Il y a une condition similaire pour ρ.

On munit la catégorie $\mathcal{CP}$ du bifoncteur $\otimes$ et on choisit comme objet unité $[\mathbf{1}]$, la dénotation de la formule $\mathbf{1}$, élément neutre pour le connecteur $\otimes$. On obtient une **catégorie monoïdale**, ou presque une catégorie monoïdale, selon les choix effectués pour définir le procédé d'élimination de la coupure. En effet, on peut définir des morphismes $\alpha_{A,B,C}$, λ_A, ρ_A comme les dénotations des preuves suivantes

$$\alpha_{A,B,C} : \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}(id) \quad \dfrac{\dfrac{}{B \vdash B}(id) \quad \dfrac{}{C \vdash C}(id)}{B, C \vdash B \otimes C}(\otimes R)}{A, B, C \vdash A \otimes (B \otimes C)}(\otimes R)}{A \otimes B, C \vdash A \otimes (B \otimes C)}(\otimes L)}{(A \otimes B) \otimes C \vdash A \otimes (B \otimes C)}(\otimes L) \qquad \lambda_A : \dfrac{\dfrac{}{A \vdash A}(id)}{\mathbf{1}, A \vdash A}(\mathbf{1}L)}{\mathbf{1} \otimes A \vdash A}(\otimes L) \qquad \rho_A : \dfrac{\dfrac{}{A \vdash A}(id)}{A, \mathbf{1} \vdash A}(\mathbf{1}L)}{A \otimes \mathbf{1} \vdash A}(\otimes L)$$

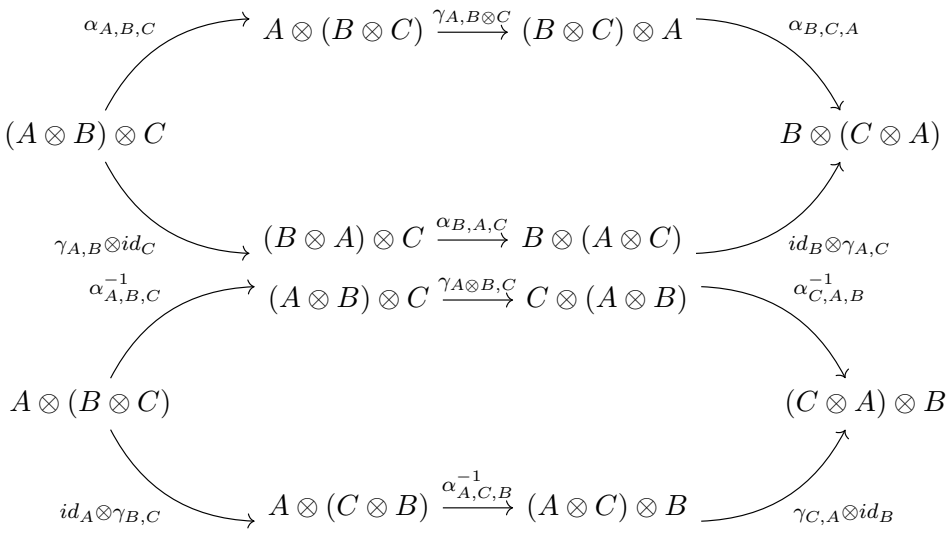
Il s'agit bien de transformations naturelles, et on a bien la propriété de cohérence. (démonstrations, ou au moins explications, exemples de preuves équivalentes ?) La définition d'une *catégorie monoïdale* exige que ces morphismes α, λ, ρ soient des isomorphismes. On n'écrit plus les indices, qui sont toujours les mêmes. On peut définir des morphismes naturels $\bar{\alpha}_{A,B,C} : A \circ (B \circ C) \to (A \circ B) \circ C$, $\bar{\lambda}_A : A \to e \circ A$, $\bar{\rho}_A : A \to A \circ e$, par exemple $\bar{\lambda}_A$ est la dénotation de $\dfrac{\dfrac{}{\vdash \mathbf{1}}(\mathbf{1}R) \quad \dfrac{}{A \vdash A}(id)}{A \vdash \mathbf{1} \otimes A}(\otimes R)$. On pense naturellement à ces morphismes quand on cherche des inverses de α, λ et ρ. On a bien $\lambda \circ \bar{\lambda} = id_A$ et $\rho \circ \bar{\rho} = id_A$. En revanche, si on considère un p.é.c. "strict", on n'a aucune des égalités suivantes : $\bar{\lambda} \circ \lambda = id_{e \circ A}$, $\bar{\rho} \circ \rho = id_{A \circ e}$, $\bar{\alpha} \circ \alpha = id_{(A \circ B) \circ C}$, $\alpha \circ \bar{\alpha} = id_{A \circ (B \circ C)}$. Si on considère au contraire la version plus "large" du p.é.c., on a bien toutes ces égalités. Les transformations autorisées qui ont été ajoutées au p.é.c. strict pour obtenir cette version large sont d'ailleurs expressément choisies pour satisfaire ces égalités. Dans la suite, on suppose toujours qu'on a choisi la version "large" du p.é.c., donc $\mathcal{CP}$ est bien une catégorie monoïdale.

4.4 Échange et catégorie monoïdale symétrique

La logique linéaire est commutative : la règle d'échange ... permet de construire la preuve canonique de $A \otimes B \vdash B \otimes A$ ci-contre. On peut alors ajouter un nouvel isomorphisme naturel à $\mathcal{CP}$ pour en faire une catégorie monoïdale symétrique. Des variantes "non commutatives" ont été étudiées dans la littérature, où la règle d'échange est supprimée ou affaiblie. Dans ce dernier cas, on peut parfois obtenir quand même une catégorie monoïdale tressée.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B}(id) \quad \dfrac{}{A \vdash A}(id)}{B, A \vdash B \otimes A}(\otimes R)}{A, B \vdash B \otimes A}(exchange)}{A \otimes B \vdash B \otimes A}(\otimes L)$$

Définition 14. *Soit $\mathcal{C}$ une catégorie monoïdale, avec les notations précédentes. C'est une* **catégorie monoïdale tressée** *s'il existe un* ***isomorphisme naturel*** $\gamma_{A,B} : A \otimes B \to B \otimes A$ *tel que les* diagrammes hexagonaux *suivants commutent.*



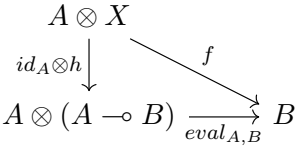
Définition 15. *Soit $\mathcal{C}$ une catégorie monoïdale tressée, avec les notations précédentes. C'est une* **catégorie monoïdale symétrique** *si pour tous objets A, B, on a $\gamma_{B,A} = \gamma^{-1}_{A,B}$. Dans ce cas, on n'a pas besoin de vérifier la commutativité du second diagramme de la définition précédente, qui est entraînée par la commutativité du premier diagramme appliquée à $\gamma_{B,A}$.*

Dans $\mathcal{C}$, on définit $\gamma_{A,B}$ comme la dénotation de la preuve canonique de $A \otimes B \vdash B \otimes A$ donnée précédemment. On obtient alors une **catégorie monoïdale symétrique**. (vrai ? démonstrations, explications, exemples ?)

4.5 Implication linéaire et catégorie monoïdale fermée

Intéressons-nous maintenant à l'implication linéaire $\multimap$. Ce connecteur logique induit encore une fois un opérateur sur les dénotations de formules, qui a un sens en théorie des catégories.

Définition 16. *Soit $\mathcal{C}$ une catégorie monoïdale, toujours avec les mêmes notations. Une* **structure fermée à gauche** *sur $\mathcal{C}$ consiste en un opérateur $\multimap$ sur les objets et pour tous objets A, B, un morphisme $eval_{A,B} : A \otimes (A \multimap B) \to B$, vérifiant la propriété d'universalité suivante : pour tout objet X et tout morphisme $f : A \otimes X \to B$, il existe un unique morphisme $h : X \to A \multimap B$ tel que le diagramme suivant commute.*



On dit alors que $\mathcal{C}$ est une **catégorie monoïdale fermée**.

On munit $\mathcal{CP}$ de l'opérateur $\multimap$ sur les dénotations de formules correspondant à l'implication linéaire. On définit $eval_{A,B}$ comme la dénotation de la preuve ci-contre. On obtient alors une **catégorie monoïdale fermée**. (Est-ce vrai ? Est-ce qu'on peut expliquer facilement comment obtenir h à partir de f ?)

$$\dfrac{\dfrac{}{A \vdash A}\,(id) \quad \dfrac{}{B \vdash B}\,(id)}{\dfrac{A, (A \multimap B) \vdash B}{A \otimes (A \multimap B) \vdash B}\,(\otimes L)}\,(\multimap L)$$

(généralisation à n'importe quelle preuve de logique intuitionniste ?)

Conclusion

Rapport de stage

Introduction

La logique intuitionniste présente un grand intérêt philosophique, logique et informatique. Elle diffère notamment de la logique classique par l'absence du principe du tiers exclu $A \vee \neg A$. La recherche automatisée de preuves en logique intuitionniste propositionnelle (ILP) est un thème de recherche mature. Parmi les techniques fréquemment utilisées se trouvent des adaptations de calculs de séquents bien choisis. Par exemple, l'équipe TYPES a développé un prouveur automatisé pour ILP appelé STRIP [**?**], basé sur une variante du calcul de séquents **LJT** [1]. Cependant, des structures de données sophistiquée sont nécessaires, si bien qu'une certification de ce prouveur semble compromise. Le calcul de séquents **LSJ** [2], relativement récent (2012), paraît prometteur pour la recherche automatisée de preuve dans ILP : l'algorithme basé dessus est facile à implémenter avec des structures de données relativement simples, ce qui est favorable à une possibilité de certification. C'est ce qui a conduit mes encadrants à proposer ce sujet de stage : recherche de preuves compilée et certifiée en logique intuitionniste propositionnelle, avec pour idée de se baser sur **LSJ**.

J'ai effectué ce stage au sein de l'équipe TYPES du LORIA de Nancy, sous la direction de D. Galmiche et D. Larchey-Wendling. J'ai implémenté un premier prouveur d'ILP à partir du calcul de séquents **LSJ**. Je l'ai testé sur les formules de la banque ILTP [**?**], et j'ai découvert que ce calcul (qui apparemment n'avait pas encore été appliqué à la réalisation pratique d'un prouveur) offre une bien meilleure complexité, sur une formule donnée d'ILTP, que tous les prouveurs actuellement répertoriés. Ensuite, j'ai essayé de simplifier le code de ce prouveur, encore trop complexe, dans une perspective de certification. Je n'ai pas eu le temps d'aborder la certification en elle-même, qui serait très longue à effectuer. J'ai cependant réalisé et comparé diverses implémentations afin d'étudier comment on pourrait faciliter une certification éventuelle sans trop perdre en efficacité.

Dans ce rapport, j'explique comment un algorithme de recherche de preuve peut être déduit d'un calcul de séquents, et quelles propriétés du calcul de séquent peuvent rendre cet algorithme effectif et efficace. Je donne pour exemples **LJ** et **LJT**, deux calculs dont des variantes ont été effectivement employée pour réaliser des prouveurs d'ILP, par exemple STRIP. J'introduis ensuite **LSJ**, qui présente des propriétés dont on comprend mieux l'intérêt à la lumière de ce qui précède, puis une légère variante **LSJ**ℓ, qui permet d'obtenir facilement une bonne complexité spatiale, et je prouve que **LSJ**ℓ est bien équivalent à **LSJ**. En ce qui concerne l'implémentation, je donne quelques détails essentiels ou remarquables. Puis, j'explique certaines décisions prises en faveur de la certification. Enfin, je présente les différentes variantes de prouveur d'ILP que j'ai effectivement réalisées, et les compare en termes de facilité de certification et d'efficacité.

Table des matières

On s'intéresse à la partie propositionnelle de la logique intuitionniste : les formules sont construites à partir de la constante $\bot$ (*faux*), de variables propositionnelles et des connecteurs binaires $\wedge$ (*et*), $\vee$ (*ou*), $\to$ (*implique*). La notation $\neg A$ signifie $A \to \bot$.

On utilise les notions et définitions suivantes, introduites dans la section 1 du mémoire : calcul de séquents, séquent, règle, prémisse, conclusion, instance, preuve ou arbre de preuve, séquent prouvable dans un calcul.

On se réfère également au calcul **LJ** défini dans cette même section, mais on travaille ici sur une représentation légèrement différente. On s'intéresse en effet à des *multiensembles*, c'est-à-dire des collections où le nombre d'occurrences est pris en compte, mais non l'ordre des éléments. Un séquent de Γ consiste alors en un multiensemble (au lieu d'une liste) de formules Γ et une formule D, toujours noté $\Gamma \vdash D$. On utilise la notation X, Y pour désigner la réunion de deux multiensembles X et Y, au lieu de la concaténation de deux listes, X ou Y pouvant encore une fois être une formule représentant ici un multiensemble à un élément. Malgré le changement de structure et grâce à la réutilisation de la notation X, Y , les règles de **LJ** sont toujours données par la figure 2 page 5, à l'exception de la règles d'échange $\dfrac{\Gamma_1, A, B, \Gamma_2 \Rightarrow D}{\Gamma_1, B, A, \Gamma_2 \Rightarrow D}\ (permut.\ L)$ qui peut être oubliée puisqu'un multiensemble ne tient pas compte de l'ordre.

1 Propriétés favorisant l'application d'un calcul de séquents à la recherche automatisée de preuve et exemples de calculs

Il existe plusieurs calculs de séquents pour la logique intuitionniste, sans parler des nombreuses autres logiques existantes. Un tel calcul comporte ses propres définition d'un séquent, règles, et construction pour chaque formule d'un séquent qui est prouvable par le calcul si et seulement si la formule est prouvable en logique intuitionniste. En dériver l'algorithme de recherche de preuve ci-après est assez naturel. Sa correction est immédiate par construction. En revanche, la terminaison pose problème. Elle n'est pas toujours assurée, et même quand elle l'est, souvent difficile à prouver. Nous présentons quelques propriétés sur les calculs de séquents qui sont intéressantes pour assurer la terminaison et améliorer la complexité de l'algorithme qui leur est associé. Enfin, nous donnons deux exemples de calculs de séquents existants pour la logique intuitionniste, antérieurs au calcul que nous utilisons pour l'implémentation.

1.1 Algorithme de recherche de preuve

On considère un calcul de séquents. Pour déterminer si un séquent est prouvable, on choisit une règle dont il peut être la conclusion et on applique récursivement la recherche de preuve aux prémisses correspondantes. Si elles sont toutes prouvables (en particulier, s'il n'y en a pas : si le séquent est la conclusion d'un axiome), alors par définition le séquent initial est aussi prouvable ; de plus, si on a calculé un arbre de preuve pour chaque prémisse, on en obtient un pour le séquent initial. Sinon, on essaie une autre règle (sauf dans certains cas où on peut conclure grâce à la notion de règle ou prémisse inversible que nous verrons plus loin). Si on a essayé toutes les règles applicables au séquent sans succès, c'est-à-dire que pour chacune, au moins une prémisse est non prouvable (en particulier, s'il n'y a aucune règle applicable : si le séquent n'est la conclusion d'aucune instance), on conclut que le séquent initial n'est pas prouvable.

Cet algorithme est correct par construction et d'après la définition de la prouvabilité d'un séquent. En revanche, il y a des causes possibles de non terminaison, qui se regroupent en deux catégories : "largeur" infinie, "profondeur" infinie. Pour éviter une "largeur" infinie, il faut que pour un séquent donné, le nombre d'instances dont il est conclusion soit fini. En

ce qui concerne le problème de "profondeur" dû à la récursivité, on peut souvent munir les séquents d'un ordre bien fondé, de sorte que pour toute instance de règle, les prémisses sont toutes strictement inférieures à la conclusion.

Remarque : la règle de coupure. La règle de coupure, par exemple pour **LJ** : $\dfrac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D}$ (cut), rend l'algorithme proposé inutilisable parce qu'il ne termine jamais. En effet, on explore indéfiniment en "largeur", car le nombre d'instances dont un séquent donné est conclusion est infini, A pouvant être n'importe quelle formule. Heureusement, cette règle est souvent non nécessaire. De nombreux calculs la formulent car c'est une bonne chose que l'implication représentée par un séquent soit transitive, mais s'en passent ensuite grâce à un *théorème d'élimination de la coupure*, souvent difficile à établir. Voir la section **??** du mémoire pour plus de détails. Quoi qu'il en soit, on ne s'intéresse désormais qu'à des calculs dans lesquels cette règle ne figure pas.

1.2 Propriétés intéressantes des calculs de séquents

Un calcul de séquents peut présenter certaines des propriétés suivantes, qui contribuent à assurer la terminaison ou à améliorer la complexité de l'algorithme précédent.

Absence de contraction. Certaines règles, comme la contraction à gauche de **LJ** : $\dfrac{A, A, \Gamma \Rightarrow D}{A, \Gamma \Rightarrow D}$ $(contraction\ L)$, sont problématiques pour la terminaison de l'algorithme. En effet, pour essayer de prouver $A, \Gamma \Rightarrow D$, on peut être amené à essayer de prouver $A, A, \Gamma \Rightarrow D$, puis en appliquant encore la même règle à essayer de prouver $A, A, A, \Gamma \Rightarrow D$, et ainsi de suite, sans fin. Il est parfois possible d'adapter l'algorithme à une possibilité de contraction en prenant certaines précautions, comme on le verra pour le calcul **LJ**.

Propriété de la sous-formule. La formule B est une ***sous-formule*** de la formule A si B est égale à A ou si A est de la forme $A_1 c A_2$ où c est un connecteur et [B est une sous-formule de A_1 ou B est une sous-formule de A_2]. Un calcul de séquents vérifie la ***propriété de la sous-formule*** si tout séquent prouvable σ admet une preuve telle que toute formule apparaissant dans (un séquent de) cette preuve est une sous-formule d'une formule de σ. La propriété de la sous-formule est très utile pour un calcul de séquents. Souvent, elle fait partie des arguments qui permettent de montrer la terminaison. Elle fournit en effet un ordre bien fondé sur les formules, qu'il reste à étendre de façon bien choisie aux séquents. Elle est également utile lors de l'implémentation : si on veut appliquer la recherche de preuve à un séquent donné, on peut connaître à l'avance la liste exhaustive de toutes les formules susceptibles d'apparaître. On peut donc effectuer une indexation préliminaire, puis représenter les formules par des objets de taille constante, par exemple des entiers, au lieu d'arbres qui peuvent être coûteux en mémoire. Voir la sous-section ? pour un exemple détaillé d'une telle indexation.

Inversibilité de certaines règles ou prémisses. Dans l'algorithme proposé, il peut être assez long de montrer qu'un séquent n'est pas prouvable, puisqu'on essaie toutes les instances dont il est la conclusion. La notion d'inversibilité permet de terminer beaucoup plus rapidement dans certains cas. Une prémisse $prem_k$ d'une règle $\dfrac{prem_1 \quad \ldots \quad prem_p}{concl}$ $(\mathcal{R})$ (aussi appelée *k-ième prémisse de $\mathcal{R}$*) est ***inversible*** si on a : si $prem_k$ est non prouvable, alors

concl est non prouvable. Une règle est ***inversible*** si toutes ses prémisses sont inversibles. Ainsi, si au cours de la recherche de preuve, on obtient qu'une prémisse inversible est non prouvable, on peut directement conclure que la conclusion ne l'est pas non plus, sans avoir besoin d'essayer d'autre règle.

Localité des règles. L'algorithme nécessite de savoir déterminer, pour un séquent donné, toutes les instances dont il est conclusion, et en particulier calculer les prémisses de ces instances. Pour une instance, la *formule principale* est la formule de la conclusion qui est remplacée dans les prémisses par d'autres formules (règles logiques), ou dupliquée ou supprimée (règles structurelles). Dans le cas des règles logiques, la formule principale doit avoir une forme particulière, par exemple présenter un connecteur donné. Souvent, pour une conclusion et une règle données et un choix de formule principale autorisé par la règle, il existe une unique instance correspondante, dont on peut facilement calculer toutes les prémisses. Parfois, on a aussi le sens inverse : à partir de la k-ième prémisse, si on connaît la règle et le numéro k et la formule principale, on peut construire la conclusion. On dit qu'une règle est ***locale*** si on a cette dernière propriété pour toutes les prémisses de toutes ses instances. Si toutes les règles sont locales (et si on cherche juste à décider si un séquent est prouvable sans demander d'arbre de preuve le cas échéant), alors on peut ne garder qu'un seul séquent en mémoire à tout moment, plus des informations (numéro de prémisse, formule principale) qui sont moins coûteuses. Lorsqu'on s'intéresse à une instance dont le séquent retenu est conclusion, on transforme ce séquent en une prémisse, sur laquelle on relance l'algorithme. Et inversement, on a parfois besoin de revenir à la conclusion à partir d'une prémisse et des informations supplémentaires retenues : par exemple pour ensuite calculer une autre prémisse de l'instance, ou encore pour essayer d'appliquer une autre règle à la conclusion si on a trouvé une prémisse non prouvable et non inversible. Ainsi, si toutes les règles du calcul sont locales, on peut améliorer la complexité spatiale de l'algorithme.

1.3 Deux exemples de calculs de séquents pour la logique intuitionniste

LJ. Pour appliquer le calcul **LJ** présenté dans la première partie à la recherche automatique de preuves, il faut quelques ajustements sur le calcul lui-même et sur l'algorithme proposé. Il faut notamment enlever la règle de coupure , ce qui est possible car le calcul reste évidemment correct, mais même complet (propriété d'élimination de la coupure). Pour la même raison, on peut aussi enlever la règle *weakening L*, à condition de récrire la règle *id* en $\dfrac{}{\Gamma, A \vdash A}\,(id)$. En revanche, on ne peut pas supprimer purement et simplement la règle $\dfrac{A, A, \Gamma \Rightarrow D}{A, \Gamma \Rightarrow D}\,(\mathit{contraction}\ L)$. On ne pourrait par exemple plus prouver la formule $\neg\neg(A \vee \neg A)$. Mais comme on l'a vu, cette règle pose un problème de terminaison de l'algorithme. Une solution consiste à remplacer les deux règles *contraction L* et $\dfrac{\Gamma \vdash A \quad \Gamma, B \vdash D}{\Gamma, A \to B \vdash D}\,(\to L)$ par une seule règle $\dfrac{A \to B, \Gamma \Rightarrow A \quad B, \Gamma \Rightarrow D}{A \to B, \Gamma \Rightarrow D}\,(\to L)$. On n'a alors plus d'appels récursifs sur des séquents strictement croissants $\Gamma, A \Rightarrow D$ puis $\Gamma, A, A \Rightarrow D$ puis $\Gamma, A, A, A \Rightarrow D$ etc. En revanche, on peut avoir un appel récursif sur un séquent déjà rencontré, par exemple si $D = A$, une prémisse est identique à la conclusion dans $\dfrac{A \to B, \Gamma \Rightarrow A \quad B, \Gamma \Rightarrow A}{A \to B, \Gamma \Rightarrow A}\,(\to L)$. On ajoute alors un système de détection de cycles en retenant tous les séquents rencontrés. L'algorithme obtenu est correct et termine. On a la propriété de la sous-formule. Les règles sont toutes inversibles et locales sauf $\to L$, dont seule

la deuxième prémisse est inversible. Mais la détection de cycles est très coûteuse.

LJT. Le calcul **LJT** est introduit par R. Dyckhoff dans [1] pour pallier le problème de cycles de **LJ**. Il n'y a pas de règle de contraction, et la règle $\to L$ est remplacée par quatre règles selon la structure de A dans la formule principale $A \to B$: par exemple $\dfrac{B, A, \Gamma \Rightarrow D}{A \to B, A, \Gamma \Rightarrow D}\,(\to L_1)$ où A doit être réduite à une variable, ou encore $\dfrac{A_1 \to (A_2 \to B)\,,\ \Gamma \Rightarrow D}{(A_1 \wedge A_2) \to B\,,\ \Gamma \Rightarrow D}\,(\to L_2)$. On n'a pas la propriété de la sous-formule, mais on peut quand même déterminer toutes les formules susceptibles d'apparaître lorsqu'on essaie de prouver un séquent donné. Dyckhoff montre que l'algorithme termine en choisissant bien un bon ordre sur les formules puis sur les séquents. Toutes les règles sont inversibles et locales sauf une des règles qui remplacent $\to L$, qui comporte deux prémisses dont seule la deuxième est inversible. Une variante de **LJT** est le fondement du prouveur STRIP [**?**].

2 Le calcul de séquents utilisé pour l'implémentation : LSJ, légèrement modifié en LSJℓ

Le calcul de séquents utilisé pour l'implémentation est **LSJ**ℓ, une variante de **LSJ**. Le calcul **LSJ** est présenté par M. Ferrari, C. Fiorentini et G. Fiorino dans [2]. Il présente des priopriétés très intéressantes pour l'application à la recherche de preuve. **LSJ**ℓ est fondamentalement le même calcul, avec une représentation des séquents un peu plus riche en informations. Il a été proposé par mon maître de stage D. Larchey-Wendling. **LSJ**ℓ hérite de toutes les bonnes propriétés de **LSJ**, en ajoutant la localité des règles.

2.1 Séquents et règles de LSJ

Définition 17. *Un* **séquent** *de* **LSJ** *est la donnée de trois multiensembles* Θ*,* Γ *et* Δ *de formules ; on écrit* $\Theta\,;\,\Gamma \Rightarrow \Delta$*.*

On a vu que dans les calculs **LK** et **LJ**, le séquent $\Gamma \Rightarrow \Delta$ représente la formule $\left(\bigwedge_{G\in\Gamma} G\right) \to \left(\bigvee_{D\in\Delta} D\right)$, respectivement en logique classique et en logique intuitionniste, avec Δ contenant exactement une formule pour **LJ**. C'est une interprétation courante en calcul de séquents. Pour **LSJ**, on ne sait pas représenter un séquent $\Theta\,;\,\Gamma \Rightarrow \Delta$ par une seule formule. On a cependant le résultat suivant : un séquent $\emptyset\,;\,\Gamma \Rightarrow \Delta$ est prouvable dans **LSJ** si et seulement si la formule $\left(\bigwedge_{G\in\Gamma} G\right) \to \left(\bigvee_{D\in\Delta} D\right)$ est prouvable en logique intuitionniste. Γ et Δ ont donc une signification ordinaire. En revanche, Θ est propre à **LSJ**, et difficile à interpréter. On peut dire que Θ contient des formules gardées en réserve, non accessibles directement (une formule de Θ ne peut pas être *formule principale*), mais qui peuvent être transférées dans Γ et ainsi devenir accessibles. On verra que les seules règles qui agissent sur Θ sont celles qui concernent le connecteur $\to$. L'article [2] propose bien une interprétation du séquent $\Theta\ ;\ \Gamma \Rightarrow \Delta$ pour Θ quelconque, en utilisant des modèles de Kripke. Nous ne la détaillons pas, car ce qui nous intéresse surtout est la propriété suivante qui découle du résultat énoncé sur un séquent avec Θ vide.

Proposition 18. *Soit* A *une formule, elle est valide en logique intuitionniste si et seulement si le séquent* $\emptyset\,;\,\emptyset \Rightarrow A$ *est prouvable dans* **LSJ**.

Les ***règles*** du calcul **LSJ** sont données dans la figure 6. Toutes les règles sont des *axiomes* ou des *règles logiques*. Il n'y a pas de *règle structurelle* ni de règle de *coupure*.

$$\frac{}{\Theta\,;\bot,\Gamma\Rightarrow\Delta}\,(\bot L) \qquad \frac{}{\Theta\,;A,\Gamma\Rightarrow A,\Delta}\,(id)$$

$$\frac{\Theta\,;A,B,\Gamma\Rightarrow\Delta}{\Theta\,;A\wedge B,\Gamma\Rightarrow\Delta}\,(\wedge L) \qquad \frac{\Theta\,;\Gamma\Rightarrow A,\Delta \quad \Theta\,;\Gamma\Rightarrow B,\Delta}{\Theta\,;\Gamma\Rightarrow A\wedge B,\Delta}\,(\wedge R)$$

$$\frac{\Theta\,;A,\Gamma\Rightarrow\Delta \quad \Theta\,;B,\Gamma\Rightarrow\Delta}{\Theta\,;A\vee B,\Gamma\Rightarrow\Delta}\,(\vee L) \qquad \frac{\Theta\,;\Gamma\Rightarrow A,B,\Delta}{\Theta\,;\Gamma\Rightarrow A\vee B,\Delta}\,(\vee R)$$

$$\frac{\Theta\,;B,\Gamma\Rightarrow\Delta \quad B,\Theta\,;\Gamma\Rightarrow A,\Delta \quad B\,;\Theta,\Gamma\Rightarrow A}{\Theta\,;A\to B,\Gamma\Rightarrow\Delta}\,(\to L)$$

$$\frac{\Theta\,;A,\Gamma\Rightarrow B,\Delta \quad \emptyset\,;A,\Theta,\Gamma\Rightarrow B}{\Theta\,;\Gamma\Rightarrow A\to B,\Delta}\,(\to R)$$

FIGURE 6 – Les règles du calcul **LSJ**

Propriétés de LSJ. (Pour les démonstrations, voir [2].) Le calcul **LSJ** est sans contraction et vérifie la propriété de la sous-formule. L'algorithme décrit dans la deuxième partie termine pour **LSJ**. Les règles $\wedge L$, $\wedge R$, $\vee L$ et $\vee R$ sont inversibles ; les deux premières prémisses de $\to L$ et la première prémisse de $\to R$ sont inversibles ; la troisième prémisse de $\to L$ et la deuxième prémisse de $\to R$ ne sont pas inversibles.

Non localité de certaines règles. Les règles $\to L$ et $\to R$ ne sont pas locales : pour chacune, les formules représentées par Δ dans la conclusion n'apparaissent nulle part dans la dernière prémisse, il n'est donc pas possible de retrouver la conclusion en connaissant uniquement cette prémisse, la formule principale et le numéro de la prémisse, puisqu'il n'y a aucun moyen d'en déduire ce qui se trouve dans Δ. C'est pour cette raison qu'on introduit le calcul **LSJℓ**, dans lequel toutes les règles sont locales.

2.2 Séquents et règles de LSJℓ

Le calcul **LSJℓ** est très proche du calcul **LSJ** : chaque règle de **LSJℓ** est l'adaptation directe d'une règle de **LSJ** à une autre structure des séquents. Contrairement à **LSJ**, les règles de **LSJℓ** sont toutes locales. Pour cela, les séquents de **LSJℓ** représentent chacun un séquent de **LSJ**, avec un peu plus d'informations : celles qui sont parfois nécessaires pour retrouver la conclusion à partir d'une prémisse. Cette représentation est exhaustive et correcte. On définit en effet une surjection $\mathbf{\Phi}$ de l'ensemble des séquents de **LSJℓ** dans l'ensemble des séquents de **LSJ**, et on montre dans la sous-section suivante qu'un séquent de **LSJℓ** est prouvable dans **LSJℓ** si, et seulement si, son image par $\mathbf{\Phi}$ est prouvable dans **LSJ**.

Définition 19. *Un* **séquent** *de* **LSJℓ** *est la donnée de deux multiensembles* Γ *et* Δ *de couples* " entier : formule "*, et d'un entier naturel* n*, tels que tous les entiers présents dans* Γ *sont* $\leq n+1$ *et tous ceux présents dans* Δ *sont* $\leq n$ *; on écrit* $\Gamma \Rightarrow_n \Delta$*.*

Lien avec les séquents de LSJ : l'application $\mathbf{\Phi}$. Soit M un multiensemble de couples " *entier : formule* ", l'entier d'un couple étant appelé son indice. On note M_k le multiensemble obtenu à partir de M en ne gardant que les couples d'indice k, et $M_{\leq k}$ celui obtenu en ne gardant que les couples d'indice inférieur à k. On note $\mathsf{forget}(M)$ le multiensemble de formules obtenu en oubliant l'indice et ne gardant que la formule de chaque couple de M. On définit l'application $\mathbf{\Phi}$ de l'ensemble des séquents de **LSJℓ** dans l'ensemble des séquents de **LSJ**, qui à $\Gamma' \Rightarrow_n \Delta'$ associe $\Theta\,;\Gamma \Rightarrow \Delta$ où : $\begin{cases} \Theta = \mathsf{forget}(\Gamma'_{n+1}) \\ \Gamma = \mathsf{forget}(\Gamma'_{\leq n}) \\ \Delta = \mathsf{forget}(\Delta'_n) \end{cases}$. C'est une **surjection** : en effet tout séquent $\Theta\,;\Gamma \Rightarrow \Delta$ de **LSJ** a au moins pour antécédent le séquent $\Gamma' \Rightarrow_0 \Delta'$, avec $\Gamma' = 0:\Gamma \cup 1:\Theta$ et $\Delta' = 0:\Delta$, où par exemple $0:\Gamma$ est le multiensemble de couples obtenu

à partir de Γ en remplaçant chaque occurrence d'une formule A par une occurrence du couple $0 : A$.

Les ***règles*** du calcul **LSJℓ** sont données dans la figure 7. Chacune correspond à une règle de **LSJ**.

LSJℓ présente les mêmes propriétés que **LSJ**, auxquelles s'ajoute la localité de toutes les règles.

n et parfois i désignent toujours des entiers naturels, avec $i \leq n$

$$\frac{}{i : \bot, \Gamma \Rightarrow_n \Delta}\,(\bot L) \qquad \frac{}{i : A, \Gamma \Rightarrow_n n : A, \Delta}\,(id)$$

$$\frac{i : A, i : B, \Gamma \Rightarrow_n \Delta}{i : A \wedge B, \Gamma \Rightarrow_n \Delta}\,(\wedge L) \qquad \frac{\Gamma \Rightarrow_n n : A, \Delta \quad \Gamma \Rightarrow_n n : B, \Delta}{\Gamma \Rightarrow_n n : A \wedge B, \Delta}\,(\wedge R)$$

$$\frac{i : A, \Gamma \Rightarrow_n \Delta \quad i : B, \Gamma \Rightarrow_n \Delta}{i : A \vee B, \Gamma \Rightarrow_n \Delta}\,(\vee L) \qquad \frac{\Gamma \Rightarrow_n n : A, n : B, \Delta}{\Gamma \Rightarrow_n n : A \vee B, \Delta}\,(\vee R)$$

$$\frac{i : B, \Gamma \Rightarrow_n \Delta \quad n+1 : B, \Gamma \Rightarrow_n n : A, \Delta \quad n+2 : B, \Gamma \Rightarrow_{n+1} n+1 : A, \Delta}{i : A \to B, \Gamma \Rightarrow_n \Delta}\,(\to L)$$

$$\frac{0 : A, \Gamma \Rightarrow_n n : B, \Delta \quad 0 : A, \Gamma \Rightarrow_{n+1} n+1 : B, \Delta}{\Gamma \Rightarrow_n n : A \to B, \Delta}\,(\to R)$$

FIGURE 7 – Les règles du calcul **LSJℓ**

2.3 Équivalence entre LSJℓ et LSJ

On note $\mathfrak{S}$ l'ensemble des séquents de **LSJ**, et $\mathfrak{S}'$ l'ensemble des séquents de **LSJℓ**. Soit $\sigma \in \mathfrak{S}$, on note $\vdash \sigma$ si σ est prouvable dans **LSJ** ; soit $\sigma' \in \mathfrak{S}'$, on note $\vdash' \sigma'$ si σ' est prouvable dans **LSJℓ**. Montrons que pour tous $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$, on a $\vdash \sigma$ si et seulement si $\vdash' \sigma'$, où $\mathbf{\Phi}$ est la surjection de $\mathfrak{S}'$ sur $\mathfrak{S}$ définie précédemment.

Soit $\mathcal{R}$ une règle de **LSJ**. Pour les distinguer, on notera ici $\mathcal{R}'$ la règle de **LSJℓ** de même nom. On écrit $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$ et $\dfrac{\sigma'_1 \quad ... \quad \sigma'_p}{\sigma'}\,(\mathcal{R}')$ des instances de ces règles.

Lemme 20. *Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$ et soit $\mathcal{R}$ une règle de* **LSJ**.

1) Si $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$ *alors il existe σ'_1, ... , σ'_p tels que pour tout k, $\sigma_k = \mathbf{\Phi}(\sigma'_k)$, et* $\dfrac{\sigma'_1 \quad ... \quad \sigma'_p}{\sigma'}\,(\mathcal{R}')$.

2) Si $\dfrac{\sigma'_1 \quad ... \quad \sigma'_p}{\sigma'}\,(\mathcal{R}')$ *, posons pour tout k, $\sigma_k = \mathbf{\Phi}(\sigma'_k)$, alors* $\dfrac{\sigma_1 \quad ... \quad \sigma_p}{\sigma}\,(\mathcal{R})$. *Pour un axiome $\mathcal{A}$, cela signifie simplement :* $\dfrac{}{\sigma}\,(\mathcal{A})$ *si et seulement si* $\dfrac{}{\sigma'}\,(\mathcal{A}')$.

Démonstration. On le montre pour chaque règle ; c'est une conséquence assez directe de la définition de $\mathbf{\Phi}$. Faisons-le par exemple pour *id* et $\to L$. À chaque fois, on se donne $\sigma = \Theta\,;\Gamma \Rightarrow \Delta$ et $\sigma' = \Gamma' \Rightarrow_n \Delta' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$.

id : On a $\dfrac{}{\sigma}\,(\mathrm{id})$ si et seulement s'il existe une formule A appartenant à la fois à Γ et Δ, ce qui équivaut, par définition de $\mathbf{\Phi}$, à : il existe A et $i \leq n$ tels que $n : A \in \Delta'$ et $i : A \in \Gamma'$, c'est-à-dire $\dfrac{}{\sigma'}\,(\mathrm{id}')$.

$\to L$: 1) Si $\dfrac{\sigma_1 \quad \sigma_2 \quad \sigma_3}{\sigma}\,(\to L)$ alors il existe A, B et $\widetilde{\Gamma}$ tels que $\Gamma = A \to B, \widetilde{\Gamma}$ et $\sigma_1 = \Theta\,; B, \widetilde{\Gamma} \Rightarrow \Delta$ et $\sigma_2 = B, \Theta\,; \widetilde{\Gamma} \Rightarrow A, \Delta$ et $\sigma_3 = B\,; \Theta, \widetilde{\Gamma} \Rightarrow A$; et il existe $i \leq n$ tel que $i : A \to B \in \Gamma'$. On pose $\widetilde{\Gamma}' = \Gamma' - i : A \to B$ (on retire une seule occurrence de $i : A \to B$ de Γ') et $\sigma'_1 = i : B, \widetilde{\Gamma}' \Rightarrow_n \Delta'$ et $\sigma'_2 = n+1 : B, \widetilde{\Gamma}' \Rightarrow_n n : A, \Delta'$ et $\sigma'_3 = n+2 : B, \widetilde{\Gamma}' \Rightarrow_{n+1} n+1 : A, \Delta'$ et on vérifie qu'on a bien $\sigma_1 = \mathbf{\Phi}(\sigma'_1)$ et $\sigma_2 = \mathbf{\Phi}(\sigma'_2)$ et $\sigma_3 = \mathbf{\Phi}(\sigma'_3)$ (en remarquant que $\widetilde{\Gamma} = \mathsf{forget}(\widetilde{\Gamma}'_{\leq n})$), et aussi $\dfrac{\sigma'_1 \quad \sigma'_2 \quad \sigma'_3}{\sigma'}\,(\to L')$).

2) Si $\dfrac{\sigma'_1 \quad \sigma'_2 \quad \sigma'_3}{\sigma'}\,(\to L')$ alors il existe i, A, B et $\widetilde{\Gamma}'$ tels que $\Gamma' = i : A \to B, \widetilde{\Gamma}'$ et σ'_1, σ'_2 et σ'_3 ont la forme donnée ci-dessus ; on pose $\widetilde{\Gamma} = \Gamma - A \to B$ (on retire une seule occurrence de $A \to B$ de Γ), alors les images σ_1, σ_2 et σ_3 par $\mathbf{\Phi}$ de σ'_1, σ'_2 et σ'_3 respectivement s'écrivent comme ci-dessus et donc $\dfrac{\sigma_1 \quad \sigma_2 \quad \sigma_3}{\sigma}\,(\to L)$. □

Théorème 21. *Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$, alors $\vdash \sigma$ si et seulement si $\vdash' \sigma'$.*

Démonstration. Par récurrence sur la *taille* de $\sigma \in \mathfrak{S}$, c'est-à-dire la somme des tailles des formules des trois multiensembles apparaissant dans σ.

On initialise pour tout $\sigma = \Theta\,;\Gamma \Rightarrow \Delta$ tel que toutes les formules dans Γ et dans Δ sont atomiques : soit $\sigma' = \Gamma' \Rightarrow_n \Delta' \in \mathfrak{S}'$ tel que $\sigma = \mathbf{\Phi}(\sigma')$. Alors toutes les formules associées à un $i \leq n$ dans Γ' et toutes les formules associées à n dans Δ' sont aussi atomiques. En étudiant la forme des conclusions des règles non axiomatiques de **LSJ** comme de **LSJℓ**, on remarque que si σ (resp. σ') est la conclusion d'une règle de **LSJ** (resp. **LSJℓ**), alors la règle est un axiome. L'initialisation est donc un cas particulier de ce qui suit avec $p = 0$ (ce qui entraîne qu'on n'utilise en fait pas l'hypothèse de récurrence).

Soit $\sigma = \Theta\,;\Gamma \Rightarrow \Delta \in \mathfrak{S}$. Soit $\sigma' = \Gamma' \Rightarrow_n \Delta' \in \mathfrak{S}'$ tel que $\sigma = \mathbf{\Phi}(\sigma')$.

On suppose $\vdash \sigma$. Alors il existe une règle $\mathcal{R}$ de **LSJ** et σ_1, ... , $\sigma_p \in \mathfrak{S}$ (avec éventuellement p nul) tels que $\vdash \sigma_k$ pour tout k et $\dfrac{\sigma_1 \quad \ldots \quad \sigma_p}{\sigma}\,(\mathcal{R})$. D'après le lemme, il existe σ'_1, ... , $\sigma'_p \in \mathfrak{S}'$ tels que $\sigma_k = \mathbf{\Phi}(\sigma'_k)$ pour tout k et $\dfrac{\sigma'_1 \quad \ldots \quad \sigma'_p}{\sigma'}\,(\mathcal{R}')$. Pour tout k, on applique l'hypothèse de récurrence à σ_k qui a une *taille* strictement inférieure à celle de σ, et on obtient $\vdash' \sigma'_k$. On en déduit $\vdash' \sigma'$.

On suppose $\vdash' \sigma'$. Alors il existe une règle $\mathcal{R}'$ de **LSJℓ** et σ'_1, ... , $\sigma'_p \in \mathfrak{S}'$ tels que $\vdash' \sigma'_k$ pour tout k et $\dfrac{\sigma'_1 \quad \ldots \quad \sigma'_p}{\sigma'}\,(\mathcal{R}')$. On pose $\sigma_k = \mathbf{\Phi}(\sigma'_k)$ pour tout k. D'après le lemme on a $\dfrac{\sigma_1 \quad \ldots \quad \sigma_p}{\sigma}\,(\mathcal{R})$, en particulier on peut appliquer l'hypothèse de récurrence aux σ_k donc $\vdash \sigma_k$ pour tout k, d'où $\vdash \sigma$. □

3 Éléments d'implémentation

On utilise l'algorithme dont le pseudo-code est donné dans [2]. Il s'agit de l'algorithme donné en 1.1, en mettant à profit l'inversibilité de la plupart des règles et prémisses. Adapter l'algorithme de **LSJ** à **LSJℓ** est immédiat ; cependant, on ne garde en mémoire qu'un seul séquent qu'on modifie en place, et on utilise la localité des règles pour retrouver le séquent initial avant d'essayer une nouvelle prémisse ou une nouvelle instance.

3.1 Ordre d'essai des instances : choix de la formule principale

Ce qui est précisé dans [2], mais pas en 1.1 puisque c'est propre au calcul **LSJ**, est l'ordre dans lequel on s'intéresse aux différentes instances dont un séquent donné est conclusion. On privilégie naturellement celles qui offrent une possibilité de terminer rapidement la recherche de preuve, ce qui induit une priorité en fonction de la règle associée : d'abord les axiomes,

puis les règles inversibles à une seule prémisse $\wedge L$ et $\vee R$, puis celles à deux prémisses $\wedge R$ et $\vee L$, et enfin les règles non inversibles $\rightarrow L$ et $\rightarrow R$.

Une formule est dite ***atomique*** si elle est réduite à $\bot$ ou une variable ; sinon, elle est ***composée***, c'est-à-dire de la forme $A\,c\,B$ avec c un connecteur. On remarque que, pour une formule composée donnée d'un séquent, il y a exactement une instance dont ce séquent est conclusion avec cette formule comme formule principale, et la règle associée ne dépend que de la formule et de sa position *à gauche* (dans Γ) ou *à droite* (dans Δ) du séquent . De plus toute instance ne correspondant pas à un axiome a une formule principale, qui est composée. Décider l'instance à considérer revient alors à choisir une formule principale.

Pour cela, on associe à chaque formule de Γ et de Δ une **priorité** selon la figure 8. Plus l'entier est petit, plus la formule est prioritaire. On choisit alors la formule la plus prioritaire parmi les formules "accessibles", c'est-à-dire les formules de Γ auxquelles est associé un indice inférieur à l'indice n du séquent, ou celles de Δ avec un indice égal à n ; ce sont exactement les formules qui auraient figuré dans Γ ou Δ d'un séquent $\Theta\,;\Gamma \Rightarrow \Delta$ de **LSJ**. Ce choix de formule principale n'intervient qu'après avoir testé les axiomes. Si la formule la plus prioritaire est de priorité 6, il n'y a aucune formule principale possible, donc le séquent est non prouvable. Le seul cas où on peut avoir plusieurs instances à tester pour un séquent donné est celui où la formule la plus prioritaire est de priorité 5. Dans ce cas, on teste successivement les formules "accessibles" de forme $A \rightarrow B$ de Γ comme de Δ, jusqu'à trouver une instance dont toutes les prémisses sont prouvables donc aussi la conclusion, ou les avoir toutes testées sans succès et conclure que le séquent initial n'est pas prouvable. On a arbitrairement choisi que $\wedge L$ est prioritaire sur $\vee R$, et $\vee L$ sur $\wedge R$. Cela permet de déduire directement de la priorité la règle concernée, sauf dans le cas particulier des "implique".

Forme	Côté	Priorité
$A \wedge B$	L	1
$A \vee B$	R	2
$A \vee B$	L	3
$A \wedge B$	R	4
$A \rightarrow B$	L, R	5
atomique	L, R	6

FIGURE 8 – Priorité selon la forme de la formule et son appartenance à Γ ("côté L") ou Δ ("côté R")

3.2 Indexation

Une formule peut être considérée comme un **arbre binaire** dont les feuilles sont étiquetées par la constante $\bot$ ou une variable propositionnelle, et les nœuds internes sont étiquetés par un connecteur binaire. La notion de sous-formule correspond alors à celle de sous-arbre. On utilise la propriété de la sous-formule, que **LSJℓ** hérite de **LSJ**, pour obtenir une représentation des formules plus facile à manipuler. En effet, si l'objectif est de déterminer la prouvabilité de la formule A en logique intuitionniste, on applique l'algorithme de preuve au séquent $\Rightarrow A$, donc toutes les formules susceptibles d'apparaître sont des sous-formules de la formule A. On peut ainsi réaliser une phase préliminaire d'**indexation** où on associe un entier à chaque sous-formule de A, c'est-à-dire chaque nœud (nœud interne ou feuille) de l'arbre correspondant. On retient en plus un tableau qui à l'entier représentant la formule B, associe : la constante $\bot$ ou la variable locale correspondant à B si B est atomique, ou "$i\ c\ j$" si $B = C\,c\,D$ avec c un connecteur binaire et i, j les entiers respectivement associés à C, D. Le nombre de cases de ce tableau est la *taille* de la formule A dont on veut décider la prouvabilité, c'est-à-dire le nombre de nœuds de l'arbre correspondant. On peut alors accéder facilement aux fils de n'importe quelle formule considérée, et comme on a récursivement cette information sur les fils, on peut retrouver, à partir du numéro d'une formule, toutes les informations de l'arbre associé.

On peut aussi déterminer, au cours de l'indexation, l'unique côté possible pour un numéro de formule donné, le *côté* d'une formule dans un séquent étant L si elle est dans Γ, R si elle est dans Δ. En effet, si $H = A \wedge B$ ou $H = A \vee B$, alors A et B ont toutes deux le même côté que H ; si $H = A \to B$ alors B a le même côté que H et A le côté opposé. On distingue ici les différentes occurrences d'une même formule, qui ont des numéros associés distincts. Comme on sait que dans la recherche de preuve pour une formule A, le côté de cette formule est R dans le séquent initial $\Rightarrow A$, on peut associer un côté à chaque numéro représentant une sous-formule de A.

```
((a & b) -> (non (c) | (a & b)))
  sf   classe   description   côté
   1        1          Var a     L
   2        2          Var b     L
   3        3          1 & 2     L
   4        4          Var c     L
   5        0           Faux     R
   6        5        4 -> 5      R
   7        1          Var a     R
   8        2          Var b     R
   9        3          7 & 8     R
  10        6          6 | 9     R
  11        7       3 -> 10      R
```

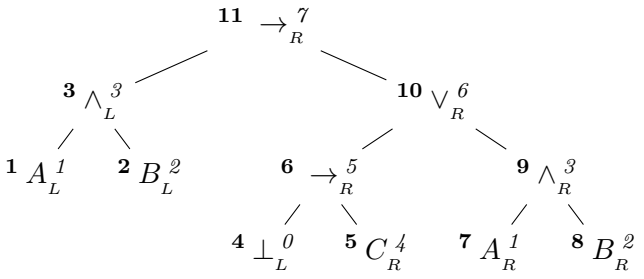


FIGURE 9 – Indexation de la formule $(a \wedge b) \to (\neg c \vee (a \wedge b))$. "sf" comme "sous-formule" est le numéro associé à une formule (en gras dans l'arbre), "description" ce qui est stocké dans le tableau. La classe d'une formule (en italique) est expliquée dans la prochaine sous-section.

3.3 Classes d'égalité structurelle pour l'axiome *id*

L'axiome $\dfrac{}{i : A, \Gamma \Rightarrow_n n : A, \Delta}\,(id)$ $(i \leq n)$ affirme qu'un séquent est prouvable si une même formule A apparaît, de façon "accessible" (condition sur les indices), de part et d'autre du séquent. "même formule" est à comprendre au sens de l'égalité structurelle, c'est-à-dire l'égalité des arbres associés. On teste très souvent cette égalité au cours de la recherche de preuve. Le faire sur des arbres est long (de l'ordre du nombre de nœuds du plus petit). On préfère, au cours de l'indexation, construire un deuxième tableau, qui à chaque numéro de formule associe un numéro de *classe*, à comprendre comme les classes d'équivalence de l'égalité structurelle ; voir a figure 9. Ainsi, on détermine ensuite en temps constant si les formules associées à deux entiers donnés sont égales structurellement.

3.4 Structure de données pour les multiensembles Γ et Δ du séquent

Soit $\sigma = \Gamma \Rightarrow \Delta$ le séquent qu'on garde en mémoire au cours de l'algorithme. Le choix de la représentation de Γ et Δ, multiensembles de couples " *indice : formule* ", est motivé par ce qui suit. On doit souvent, et donc on voudrait pouvoir rapidement

- choisir un couple principal $i : H$ le plus prioritaire ;
- retirer le couple principal choisi $i : H$ du séquent donc de Γ ou Δ, et si $H = A \, c \, B$, ajouter $j : A$ ou $k : B$ où j et k sont des indices valant n, l'indice du séquent, ou $n+1$;
- inversement, retirer $j : A$ ou $k : B$ et rajouter $i : H$.

Nous représentons Γ, comme Δ, par un tableau à 6 cases, une par priorité. Dans la case p se trouve une liste de couples (*un indice* i , *la liste des formules* H *de priorité* p *telles que* $i : H$ *est dans* Γ *resp.* Δ) ; ces couples sont triés par indices décroissants ; un tel couple n'est présent que si sa liste de formules est non vide.

Le tableau indexé par les indices permet d'une part un choix rapide du couple principal, et d'autre part, d'ajouter rapidement n'importe quel couple $j : A$ quelle que soit sa priorité. Utiliser par exemple une liste de tous les couples triés par priorité assurerait le premier point, mais pas le second. Le tri des indices par ordre décroissant est pour l'ajout de $j : A$ ou $k : B$, car dans toutes les règles, j et k valent n ou $n+1$, où n est l'indice du séquent. Avoir un seul couple avec une liste de formules par indice permet l'ajout en temps constant d'un couple d'indice n même dans Γ, où il peut y avoir des couples d'indice $n+1$. Le couple principal se trouve au début de la liste car il est choisi comme le premier convenable rencontré, son retrait est donc en temps constant. Enfin, pour assurer que retirer $j : A$ ou $k : B$ ou rajouter $i : H$ se fait aussi en temps constant, on vérifie qu'entre le moment où on passe d'une conclusion d'une instance à une prémisse, et celui où on repasse de la prémisse à la conclusion, la représentation des multiensembles de la prémisse est exactement la même, avec toutes les listes dans le même ordre. On obtient bien toutes les opérations voulues en temps constant, sauf dans un cas : lorsque les formules les plus prioritaires sont des "implique", il faut parfois en essayer plusieurs. On effectue des permutations circulaires pour faire varier le couple principal, choisi comme celui en tête de liste. On n'oublie pas, selon le nombre de couples principaux qui ont été essayés, de finir de permuter les formules afin de bien revenir à une représentation du séquent identique à la représentation initiale.

3.5 Informations supplémentaires pour un test rapide des axiomes

Les axiomes sont testés à chaque appel récursif sur un nouveau séquent. L'axiome *id* notamment est long à tester naïvement : il faudrait tester, pour tout couple (A, B) de formules "accessibles" avec A dans Γ et B dans Δ, si $A = B$ (égalité structurelle expliquée en 3.3). La complexité associée est le produit des cardinaux de Γ et Δ. On préfère retenir les multiensembles obtenus à partir de Γ et Δ en remplaçant la formule de chaque couple par sa classe, appelés respectivement $\widetilde{\Gamma}$ et $\widetilde{\Delta}$. On retient également deux variables booléennes $fauxL$ et id. Lorsqu'on ajoute une formule à Δ, on met à jour $\widetilde{\Delta}$, et on vérifie si la classe de la formule ajoutée est présente et "accessible" dans $\widetilde{\Gamma}$, auquel cas on assigne $vrai$ à id. De même pour un ajout à Γ en échangeant les rôles de $\widetilde{\Delta}$ et $\widetilde{\Gamma}$, mais en plus, si la formule ajoutée est $\bot$ avec un indice inférieur à celui du séquent, on assigne $vrai$ à $fauxL$. Lorsqu'on transforme une prémisse d'une instance en la conclusion, on réassigne $faux$ aux deux variables : si l'instance a été testée, c'est parce qu'aucun axiome n'était applicable à la conclusion. Ainsi, lorsqu'au début de chaque appel récursif sur le séquent actuellement en mémoire, on commence par tester les axiomes, il suffit de regarder si id ou $fauxL$ contient $vrai$.

L'intérêt d'avoir remplacé les formules par leur classe est qu'on peut représenter $\widetilde{\Gamma}$ et $\widetilde{\Delta}$ avec une structure de données similaire à celle pour Γ et Δ : un tableau indexé par les classes et dans la case cl, une liste de couples (*un indice* i , *le nombre de couples* $i : cl$ *dans* $\widetilde{\Gamma}$ *resp.* $\widetilde{\Delta}$). Vérifier si une classe donnée est présente et "accessible" se fait alors en temps constant.

4 Perspective de certification

Un des objectifs du stage était de s'intéresser à la certification d'un prouveur de logique intuitionniste s'appuyant sur **LSJ**. Écrire la certification d'un tel programme est très long et n'a pas été abordé. En revanche, l'implémentation a été modifiée de façon à faciliter la certification, souvent au détriment de l'efficacité. Plusieurs variantes d'implémentation ont

ainsi été étudiées, afin de comparer certaines méthodes en termes de facilité de certification et d'efficacité.

4.1 Un langage simple pour faciliter la certification

Un premier pas vers la certification est l'utilisation d'un langage relativement simple, afin de limiter le nombre de propriétés à démontrer. Le langage que nous avons employé pour cette raison, qu'on notera $\boldsymbol{T}$, comporte un seul type de données : des arbres binaires, qui consistent en l'arbre vide ou une feuille étiquetée par un entier naturel ou un couple d'arbres. Les expressions sont données dans la figure 10, *var* désignant une variable et *nomf* un nom de fonction, des chevrons $<.,.>$ étant utilisés pour les couples d'arbres. La condition du **if** doit être une feuille d'un entier n, correspondant à *faux* si $n = 0$ et *vrai* sinon ; **isnull**, **isint** et la comparaison renvoient de même une feuille contenant 0 ou 1. La première expression du **match** doit s'évaluer à un couple, sinon il y a une erreur à l'exécution. De même, dans la comparaison ou dans **succ** (successeur) ou **pred** (prédécesseur), les expressions doivent être des feuilles. Une déclaration de fonction est un triplet de la forme (*nomf*, *var*, *expr*) comportant le nom de la fonction, celui de son argument, et enfin son corps dans lequel l'argument peut apparaître comme une variable libre. Un programme consiste en une liste de déclarations de fonctions considérées comme mutuellement récursives, puis une expression à évaluer.

$$
\begin{aligned}
expr = \quad & var \mid vide \mid n \in \mathbb{N} \mid <expr, expr> \mid \textbf{match } expr \textbf{ with } <var, var> \Rightarrow expr \\
\mid \quad & \textbf{let } var = expr \textbf{ in } expr \mid \textbf{call } nomf\ expr \mid \textbf{isnull } expr \mid \textbf{isint } expr \\
\mid \quad & expr \leq expr \mid \textbf{if } expr \textbf{ then } expr \textbf{ else } expr \mid \textbf{succ } expr \mid \textbf{pred } expr
\end{aligned}
$$

FIGURE 10 – Les expressions du langage $\boldsymbol{T}$

L'intérêt d'utiliser ce langage est que mon encadrant D. Larchey-Wendling a réalisé pour celui-ci un compilateur certifié vers un langage exécutable par une machine abstraite assez simple, ainsi qu'un programme certifié simulant l'exécution de cette machine abstraite. On notera $\boldsymbol{M}$ le langage associé à cette machine abstraite. L'objectif de ces travaux sur $\boldsymbol{T}$ est justement de pouvoir écrire, grâce à ce langage, des prouveurs certifiés.

4.2 Compilation d'un programme spécifique à une formule donnée

On a vu en 3.1 que pour une formule composée donnée d'un séquent, il y a exactement une instance dont le séquent est conclusion avec cette formule principale, et la règle associée ne dépendent que du connecteur de la formule et de sa position à gauche ou à droite du séquent. Ces informations sont connues à l'issue de l'indexation. On peut alors calculer, pour chaque formule composée, de numéro f, et chaque prémisse, de numéro k, de l'unique règle associée, des fonctions de transformation de séquent $prem_{f,k}$ et $rev_{f,k}$, qui permettent de passer de la conclusion à la k-ième prémisse et réciproquement. Ces fonctions prennent en argument l'indice i de la formule principale, qui n'est connu qu'à l'exécution.

On peut ainsi, pour une formule donnée, compiler un code dans lequel des fonctions pour chaque sous-formule sont écrites en dur. L'exécution de ce code doit ensuite renvoyer un booléen indiquant si cette formule est prouvable. Calculer ces fonctions n'est pas très long par rapport à la recherche de preuve elle-même, effectuée à l'exécution, au cours de laquelle chacune de ces fonctions peut être appelée à plusieurs reprises.

L'intérêt principal de cette approche est qu'elle permet de se fixer un premier objectif : certifier le programme qui a été compilé en fonction de la formule, c'est-à-dire montrer que son exécution renvoie le bon résultat. Bien sûr, il faudrait ensuite aussi certifier la compilation vers ce programme afin d'obtenir un prouveur certifié. Mais on s'autorise dans un premier temps à effectuer en OCaml l'indexation, puis la compilation vers $\boldsymbol{T}$ des fonctions associées aux sous-formules. On ajoute ensuite ces fonctions à du code en $\boldsymbol{T}$ ne dépendant pas de la formule, pour obtenir un code intégralement en $\boldsymbol{T}$ dont l'exécution doit renvoyer la prouvabilité de la formule initiale.

5 Différentes variantes de prouveur réalisées, tests effectués, comparaison des résultats

Le langage de programmation utilisé est OCaml, à la fois pour l'algorithme de recherche de preuve à proprement parler, et pour les fonctionnalités annexes : analyseurs, interpréteurs, et même gestion des appels système pour le chronométrage, le parcours d'un répertoire ou l'écriture dans un fichier.

On génère parfois des fichiers auxiliaires contenant du code en langage $\boldsymbol{T}$ (langage défini, comme $\boldsymbol{M}$, en **??**). On peut se demander quel est l'intérêt de générer ces fichiers qui sont ensuite analysés, plutôt que de construire directement un arbre de syntaxe abstraite. La réponse est qu'une partie importante du code de ces fichiers, qui ne dépend pas de la formule qu'on essaie de prouver, a été écrite à la main directement en $\boldsymbol{T}$, donc on a de toute façon besoin d'un analyseur pour $\boldsymbol{T}$; de plus il est intéressant d'avoir un aperçu visuel du code généré pour une formule donnée.

On a finalement implémenté cinq programmes distincts, partageant des morceaux de code plus ou moins importants (par exemple, tous partagent ce qui concerne l'indexation), qui sont tous des prouveurs de logique intuitionniste : ils renvoient si une formule de logique intuitionniste donnée en argument est prouvable. Une description rapide de chacun est donnée en **??**.

Afin de tester ces prouveurs, on a utilisé la bibliothèque ILTP [**?**], qui contient des formules de référence pour évaluer les prouveurs de logique intuitionniste propositionnelle. Les fichiers d'ILTP sont en Prolog mais n'utilisent qu'un nombre très limité de constructions syntaxiques ; on a donc pu écrire un analyseur naïf ne reconnaissant que ces constructions, qui extrait d'un fichier ILTP une formule à donner en argument au prouveur, ainsi que le booléen, indiquant si cette formule est prouvable, que le prouveur est censé renvoyer, afin de pouvoir tester rapidement la correction du prouveur.

En **??**, on compare notre prouveur le plus efficace ("prouveur simple" dans **??**) aux prouveurs répertoriés par ILTP. En **??**, on compare entre eux nos différents prouveurs.

5.1 Les différents prouveurs implémentés

- "**Prouveur simple**". Ce prouveur, le premier implémenté, est le plus simple et le plus efficace. Il effectue directement la recherche de preuve telle qu'elle est décrite dans l'article présentant **LSJ** [2], en utilisant tout ce qui est décrit dans la section **??**, notamment les structures de données élaborées de **??**. En revanche, aucun effort n'est fait pour faciliter la certification.

- "**Prouveur simple avec listes**".
- "**Prouveur *T***". On compile un programme en langage ***T*** adapté à la formule donnée en argument. Pour cela, on réalise l'indexation, puis, comme expliqué en **??** on compile des fonctions correspondant à chaque sous-formule de la formule initiale. On ajoute ensuite ces fonctions à du code en ***T*** écrit à l'avance, car indépendant de la formule. Le programme intégral en ***T*** obtenu est écrit dans un fichier temporaire. Dans un deuxième temps, on exécute ce programme à l'aide d'un analyseur pour ***T*** puis d'un interpréteur relativement naïf. Cela renvoie un booléen qui doit indiquer si la formule initiale est prouvable en logique intuitionniste.
- "**Prouveur *T M***". On génère le même programme en ***T*** que précédemment. Mais cette fois, pour l'exécuter, on utilise un compilateur (fourni par D. Larchey-Wendling) de ***T*** vers ***M***. On simule ensuite assez facilement l'exécution du code en ***M*** par la machine abstraite simple à laquelle ce langage correspond. (Le code en ***M*** n'est pas écrit dans un fichier, mais directement passé au simulateur sous la forme d'un arbre de syntaxe abstraite.)
- "**Prouveur compilé caml**". On compile, selon la méthode employée par le "prouveur ***T***", un programme adapté à la formule donnée en argument, mais ce programme est écrit en OCaml. On lance ensuite le compilateur ocamlc sur le fichier temporaire contenant ce programme, puis on lance le fichier exécutable généré. Ce prouveur a été implémenté afin de tester de l'efficacité de la compilation de fonctions adaptées à la formule, sans l'influence de l'utilisation du langage ***T*** moins élaboré.

5.2 Efficacité de LSJ

Le "prouveur simple", qui est une implémentation assez directe de l'algorithme proposé dans [2], avec de légers changements comme la transformation du calcul en **LSJℓ**, se compare honorablement aux prouveurs répertoriés par ILTP.

5.3

- implémentation effective
- tests (ILTP)
- résultats
- commentaires

Conclusion

algorithme implémenté naïvement en une demi-journée. mais retravaillé, sans cesse amélioré (on ne veut pas seulement un programme qui fonctionne)

écriture d'analyseurs, automatisation de tests...

factorisation de code, partage entre les différents prouveurs (génie logiciel)

essai de différentes implémentations

l'algorithme vient de [2], **LSJℓ** de ... l'efficacité de l'algorithme vient entièrement de **LSJ** le langage ***T***

j'ai remarqué que
j'ai choisi les implémentations réalisées parmi diverses suggestions

rajouter : **LSJ** permet aussi de récupérer un contre-modèle de Kripke pour une formule non prouvable

Références

[1] R. Dyckhoff, "Contraction-free sequent calculi for intuitionistic logic.," *J. Symb. Log.*, vol. 57, no. 3, pp. 795–807, 1992.

[2] M. Ferrari, C. Fiorentini, and G. Fiorino, "Contraction-Free Linear Depth Sequent Calculi for Intuitionistic Propositional Logic with the Subformula Property and Minimal Depth Counter-Models," *Journal of Automated Reasoning*, vol. 51, no. 2, pp. 129–149, 2013.

(ajouter les références du mémoire)