

Calcul des séquents et transfert vers une catégorie

Mémoire encadré par Paul-André Melliès (PPS)

Recherche de preuves compilée et certifiée

Stage effectué au LORIA de Nancy
au sein de l'équipe TYPES
sous la direction de Didier Galmiche et Dominique Larchey-Wendling

Diane Gallois-Wong

L3 — 2014

Mémoire : Calcul des séquents et transfert vers une catégorie

Introduction

Une “preuve” évoque souvent une succession d’arguments dont l’objectif est de convaincre un auditoire ou un lecteur de la validité d’une assertion. La *théorie de la preuve* formalise, à l’aide d’un langage mathématique, cet événement consistant à convaincre quelqu’un : une *preuve* devient alors un objet mathématique. Il s’agit d’une branche de la logique qui s’intègre à la fois en mathématiques, en informatique et en philosophie. Le calcul des séquents, introduit par Gentzen, est un formalisme de théorie de la preuve qui fait l’objet de nombreuses études. Il présente un grand intérêt en informatique, puisqu’on peut en déduire assez naturellement un algorithme de recherche de preuves, bien qu’il faille prendre certaines précautions pour que cet algorithme termine. Ceci est mis en application dans mon stage. J’ai choisi pour ce mémoire de m’intéresser à un aspect complètement différent du calcul des séquents : la construction, à partir d’un calcul des séquents pour la logique linéaire intuitionniste, d’une catégorie mathématique présentant de nombreuses propriétés. Cette construction s’appuie sur une preuve constructive d’un théorème fondamental en calcul des séquents : le théorème d’élimination de la coupure. On retrouve ce théorème en recherche automatisée de preuves, où il présente un intérêt décisif qui consiste à l’oublier ! En effet, il permet de se débarrasser d’une règle de coupure dont la présence est catastrophique pour la terminaison d’un algorithme basé sur un calcul des séquents.

Dans une première partie, on présente le calcul des séquents, en s’appuyant sur l’exemple du calcul **LK** correspondant à la logique classique ; on explique aussi comment Gentzen en dérive un autre calcul **LJ** correspondant à la logique intuitionniste, une logique constructive qu’on peut obtenir à partir de la logique classique en supprimant l’axiome du tiers exclu $A \vee \neg A$. On introduit ensuite la logique linéaire puis la logique linéaire intuitionniste (ILL), présentée à travers des calculs de séquents qui leur correspondent. On expose alors le principe d’une démonstration constructive du théorème d’élimination de la coupure pour ILL, ce qui définit un *procédé d’élimination de la coupure*. Enfin, on s’appuie sur ce dernier afin de construire une catégorie correspondant au calcul des séquents pour ILL, sur laquelle on énonce quelques propriétés mathématiques.

Table des matières

1	Introduction aux calculs des séquents à travers LK pour la logique classique et LJ pour la logique intuitionniste	2
2	Logique linéaire (LL) et logique linéaire intuitionniste (ILL)	6
3	Le procédé d’élimination de la coupure (p.é.c.) pour ILL	8
4	Construction d’une catégorie et propriétés de cette catégorie	10

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

Nous introduisons dans cette première partie les définitions dont nous aurons besoin sur les calculs des séquents, à travers l'exemple du calcul des 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 des séquents respectif.

Les formules considérées dans cette partie sont construites à partir de constantes \perp (*faux*) et \top (*vrai*), de variables propositionnelles, du connecteur unaire \neg (*non*), et des connecteurs binaires \wedge (*et*), \vee (*ou*) et \rightarrow (*implique*). On s'intéresse en effet aux parties propositionnelles de ces logiques.

1.1 Des séquents et des règles

Un calcul des 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 des 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 le note $\Gamma \vdash \Delta$. On appellera séquent classique un tel séquent.

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 $\frac{}{A \vdash A} (id)$	Coupure $\frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'} (cut)$
Règles logiques	
$\frac{}{\Gamma, \perp \vdash \Delta} (\perp L)$	$\frac{}{\Gamma \vdash \top, \Delta} (\top R)$
$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta} (\neg L)$	$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta} (\neg R)$
$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L)$	$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta} (\wedge R)$
$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} (\vee L)$	$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta} (\vee R)$
$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta} (\rightarrow L)$	$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \rightarrow B, \Delta} (\rightarrow R)$
Règles structurelles	
$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} (weakening L)$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta} (weakening R)$
$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} (contraction L)$	$\frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta} (contraction R)$
$\frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta} (exchange L)$	$\frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2} (exchange R)$

FIGURE 1 – Règles du calcul **LK**

Pour une règle $\frac{prem_1 \dots prem_p}{concl}(\mathcal{R})$, \mathcal{R} est le nom de la règle, $prem_1, \dots, prem_p$ 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*.

Ce qui distingue notamment le calcul des séquents d'un système de déduction naturelle est que toutes les règles logiques sont des règles d'*introduction à gauche* ou à *droite*, c'est-à-dire que la conclusion contient une constante ou un connecteur supplémentaire par rapport aux prémisses. En déduction naturelle, on aurait des règles d'*élimination* comme $\frac{\vdash A \wedge B}{\vdash A}$.

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 des séquents **LK**. Gentzen écrit deux règles $\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L_1)$ et $\frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L_2)$ à la place de $\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L)$, 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 n'a pas fait explicitement par souci de lisibilité.

$$\frac{\frac{\Gamma, A \vdash \Delta}{\Gamma, A, B \vdash \Delta} (\text{weakening } L)}{\Gamma, A \wedge B \vdash \Delta} (\wedge L) \qquad \frac{\frac{\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B, B \vdash \Delta} (\wedge L_1)}{\Gamma, A \wedge B, A \wedge B \vdash \Delta} (\wedge L_2)}{\Gamma, A \wedge B \vdash \Delta} (\text{contraction } L)$$

1.3 Prouvabilité d'un séquent

Une **instance** d'une règle \mathcal{R} a la même forme que la règle : $\frac{\sigma_1 \dots \sigma_p}{\sigma}(\mathcal{R})$, mais ici les σ_i et σ sont des séquents connus explicitement ; bien entendu il faut qu'il s'agisse des 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 $\sigma_1, \dots, \sigma_p$ sont les séquents associés à chacun de ses fils respectivement, alors $\frac{\sigma_1 \dots \sigma_p}{\sigma}(\mathcal{R})$ 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 des 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

$\frac{\sigma_1 \quad \dots \quad \sigma_p}{\sigma}(\mathcal{R})$ d'une règle, si pour tout i , σ_i est prouvable, alors σ est prouvable (en particulier pour toute instance $\frac{}{\sigma}(\mathcal{A})$ d'un axiome \mathcal{A} , σ est prouvable).

1.4 Lien avec une logique, interprétation des séquents

Un calcul des 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 $(\bigwedge_{G \in \Gamma} G) \rightarrow (\bigvee_{D \in \Delta} D)$ 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 élégante : on restreint les séquents à ceux qui ont exactement une formule à droite ; les règles sont adaptées en conséquence. La principale conséquence de cette restriction 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 le note $\Gamma \vdash D$. On appellera séquent intuitionniste un tel séquent.*

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

FIGURE 2 – Règles du calcul **LJ**

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

Le connecteur \neg disparaît également. En effet, les règles qui lui sont associées dans **LK** modifient le nombre de formules à droite du séquent. On donne bien un sens à $\neg A$ en logique intuitionniste, mais on considère qu'il s'agit d'une simple notation pour $A \rightarrow \perp$. De plus, 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 $(\bigwedge_{G \in \Gamma} G) \rightarrow D$ 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 des 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 $A \vee (A \rightarrow \perp)$ 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 $\vee R_2$ (et la règle de coupure, mais on verra plus loin qu'elle n'est jamais nécessaire) : 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 $A \vee (A \rightarrow \perp)$, car ni le séquent $\vdash A$ ni le séquent $\vdash (A \rightarrow \perp)$ 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 $A \vee (A \rightarrow \perp)$, on utilise une contraction à droite afin de conserver à la fois A et $A \rightarrow \perp$. Le principe est le même que dans les équivalences d'écriture des règles en 1.2.

$$\frac{\frac{\frac{}{A \vdash A} (\text{id})}{A \vdash A, \perp} (\text{weak. R})}{\vdash A, A \rightarrow \perp} (\rightarrow R) \quad \frac{}{\vdash A \vee (A \rightarrow \perp)} (\vee R)$$

$$\frac{\frac{\frac{\frac{}{A \vdash A} (\text{id})}{A \vdash A, \perp} (\text{weak. R})}{\vdash A, A \rightarrow \perp} (\rightarrow R)}{\vdash A, A \vee (A \rightarrow \perp)} (\vee R_2) \quad \frac{}{\vdash A \vee (A \rightarrow \perp), A \vee (A \rightarrow \perp)} (\vee R_1) \quad \frac{}{\vdash A \vee (A \rightarrow \perp)} (\text{contr. R})$$

Finalement, on peut considérer que la logique intuitionniste est obtenue à partir de la logique classique en interdisant certaines règles structurales. Une autre logique couramment étudiée, la logique linéaire, s'obtient à partir de la logique classique en conservant toutes les règles structurales, 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 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 structurales, si bien que de nouvelles nuances apparaissent, pour lesquelles sont introduits de nouveaux connecteurs.

Voici un calcul des 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, inspirée de ([1], p.40). 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 **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, empruntées à ([2], Table 1), 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” [3] dans la sous-section 2.2. Enfin, on présente la logique linéaire intuitionniste (ILL).

Symbole	Nom
\otimes	<i>tenseur / produit tensoriel</i>
$\&$	<i>avec</i>
\oplus	<i>plus</i>
\wp	<i>produit parallèle</i>
\multimap	<i>implication linéaire</i>

Connecteurs binaires

Symbole	Nom
\neg	<i>non</i>
$!$	<i>bang</i>
$?$	<i>why not</i>

Connecteurs unaires

Symbole	Nom	Élément neutre de
1	<i>un</i>	\otimes
\top	<i>top</i>	$\&$
0	<i>zéro</i>	\oplus
\perp	<i>bot</i>	\wp

Constantes

FIGURE 3 – Les connecteurs de la logique linéaire

Remarque. Une dualité importante liée au connecteur \neg permet de définir un calcul des séquents pour la logique linéaire dans lequel les séquents ne comportent aucune formule à gauche. On a choisi de conserver des séquents avec des formules des deux côtés par similarité avec les autres calculs de séquents étudiés.

2.1 Limitation des règles structurales, nouveaux connecteurs, modalités

L’usage des règles structurales 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 \wp . On souhaite tout de même garder des règles structurales, dont l’usage est seulement limité. On introduit pour cela des connecteurs unaires *modaux*, ou *modalités*, $!$ et $?$. Les règles structurales 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.

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

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

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 $\mathbf{1}$ 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 de la 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 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 .

Les connecteurs \wp et $?$ sont plus difficiles à expliquer naturellement.

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, empruntées pour la plupart à ([4], p.75), 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 enlève les connecteurs et la constante qui sont liés à des règles qui ne s'adaptent pas aux séquents avec une unique formule à droite, à savoir \neg , \wp , $\mathbf{0}$ et $?$.

$\frac{}{A \vdash A} (id)$	$\frac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D} (cut)$
$\frac{\Gamma, A, B \vdash D}{\Gamma, A \otimes B \vdash D} (\otimes L)$	$\frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma \vdash A \otimes B} (\otimes R)$
$\frac{\Gamma, A \vdash D}{\Gamma, A \& B \vdash D} (\& L_1) \quad \frac{\Gamma, B \vdash D}{\Gamma, A \& B \vdash D} (\& L_2)$	$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\& R)$
$\frac{\Gamma, A \vdash D \quad \Gamma, B \vdash D}{\Gamma, A \oplus B \vdash D} (\oplus L)$	$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} (\oplus R_1) \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B} (\oplus R_2)$
$\frac{\Gamma \vdash A \quad \Gamma', B \vdash D}{\Gamma, \Gamma', A \multimap B \vdash D} (\multimap L)$	$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} (\multimap R)$
$\frac{\Gamma \vdash D}{\Gamma, \mathbf{1} \vdash D} (\mathbf{1} L)$	$\frac{}{\vdash \mathbf{1}} (\mathbf{1} R)$
$\frac{}{\Gamma, \perp \vdash D} (\perp L)$	$\frac{}{\Gamma \vdash \top, D} (\top R)$
$\frac{\Gamma, A \vdash D}{\Gamma, !A \vdash D} (!L)$	$\frac{! \Gamma \vdash A}{! \Gamma \vdash !A} (!R)$
$\frac{\Gamma \vdash D}{\Gamma, !A \vdash D} (weakening L)$	$\frac{\Gamma, !A, !A \vdash D}{\Gamma, !A \vdash D} (contraction L) \quad \frac{\Gamma_1, A, B, \Gamma_2 \vdash D}{\Gamma_1, B, A, \Gamma_2 \vdash D} (exchange L)$

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

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

3.1 Le théorème d'élimination de la coupure

La règle de coupure : $\frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'} (cut)$ pour des séquents classiques, $\frac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D} (cut)$ pour des séquents intuitionnistes, est, tout comme $\frac{}{A \vdash A} (id)$, une règle qu'on retrouve dans la plupart des calculs de séquents. Elle est naturelle lorsqu'on

considère l'interprétation d'un séquent classique $\Gamma \vdash \Delta$: “à partir des formules de Γ , on peut montrer une formule de Δ ”. La règle de coupure signifie alors que si on a $\Gamma \vdash A, \Delta$ et $\Gamma', A \vdash \Delta'$ alors, ou bien à partir de Γ , on peut montrer une formule de Δ , auquel cas on a aussi $\Gamma, \Gamma' \vdash \Delta, \Delta'$, ou bien à partir de Γ on peut montrer A , or à partir de A et Γ' on peut montrer une formule de Δ' , donc à partir de Γ, Γ' on peut montrer une formule de Δ' , si bien qu'on a encore $\Gamma, \Gamma' \vdash \Delta, \Delta'$. Cette interprétation s'adapte aussi aux séquents intuitionnistes. L'idée principale est qu'une conclusion établie peut être immédiatement utilisée comme hypothèse (en rappelant que dans $\Gamma \vdash \Delta$ les formules de Γ sont appelées les hypothèses et celles de Δ les conclusions). La règle de coupure représente une transitivité : de $A \vdash B$ et $B \vdash C$ on déduit $A \vdash C$.

Le *théorème d'élimination de la coupure*, pour un calcul donné présentant une règle de coupure, affirme que si on supprime la règle de coupure du calcul, on obtient un calcul équivalent, c'est-à-dire que les séquents prouvables restent les mêmes. Une formulation équivalente est : tout séquent prouvable dans le calcul considéré admet un arbre de preuve dans lequel la règle de coupure ne figure pas. C'est cette formulation qui est utilisée dans la démonstration que nous allons étudier. Le théorème d'élimination de la coupure est vérifié par de nombreux calculs de séquents, notamment tous ceux que nous avons introduits dans ce mémoire. Il est intéressant d'un point de vue sémantique : la transitivité représentée par la règle de coupure est vérifiée par le calcul, même si on ne l'impose pas par un énoncé explicite de cette règle de coupure. Souvent valide, ce théorème est néanmoins fréquemment difficile à établir.

3.2 Éléments d'une démonstration de ce théorème pour ILL

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

Nous présentons un bref aperçu d'une démonstration constructive du théorème d'élimination de la coupure dans le cadre du calcul des séquents lié à la logique linéaire intuitionniste que nous avons présenté. Elle figure dans le chapitre 3 de [1]. En donner les détails serait bien trop long : dans le livre cité, elle occupe une vingtaine de pages. Elle fournit une liste précise de transformations élémentaires, qui à une preuve d'un séquent associent une autre preuve du même séquent. L'idée est qu'en les appliquant successivement dans un ordre bien choisi à une preuve d'un séquent, on peut obtenir une preuve du même séquent n'utilisant jamais la règle de coupure. Un premier point à noter est qu'une transformation élémentaire peut être appliquée à un sous-arbre d'une preuve donnée : par exemple, une preuve $\frac{p}{\sigma}(\mathcal{R})$ peut

être transformée en $\frac{p'}{\sigma'}(\mathcal{R})$ si la preuve p peut être transformée en p' . Les transformations élémentaires sont locales : le nombre de nœuds explicités lorsqu'on en énonce une ne dépasse pas une dizaine. Bien que la coupure ne soit pas une règle logique, on appelle *formule principale* de $\frac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D}$ (*cut*) la formule A qui a un rôle particulier. L'idée associée aux applications successives de transformations élémentaires est de “faire remonter” les utilisations de la règle de coupure à une profondeur de plus en plus grande dans l'arbre. Pour cela, on a parfois besoin de remplacer des utilisations de cette règle par d'autres dont la formule principale est plus petite, quitte à augmenter la taille de l'arbre. Il faut bien entendu que certaines transformations permettent de réduire strictement le nombre d'utilisations de la règle de coupure : par exemple,

$$\frac{A \vdash A \quad (id) \quad \frac{\dots \frac{p}{\Gamma_1, A, \Gamma_2 \vdash B} \dots}{\Gamma_1, A, \Gamma_2 \vdash B} (cut)}{\Gamma_1, A, \Gamma_2 \vdash B} \quad \text{se transforme en} \quad \frac{\dots \frac{p}{\Gamma_1, A, \Gamma_2 \vdash B} \dots}{\Gamma_1, A, \Gamma_2 \vdash B}.$$

Un exemple de transformation permettant de diminuer la taille de la formule principale est celle, associée au connecteur \otimes , qui transforme

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

Le choix de la transformation à appliquer selon la situation est très compliqué : il dépend notamment de la nature des règles qui se situent au-dessus de la règle de coupure qu'on souhaite “faire remonter” (éventuellement en plusieurs transformations), mais ce choix n'est parfois pas le même selon que la formule principale d'une de ces règles est ou non égale à la formule principale de la coupure considérée...

On admettra que le procédé qui consiste à appliquer ces transformations dans un ordre bien choisi fonctionne. On renvoie au chapitre 3 de [1] pour le détail des transformations et du choix de la transformation à appliquer selon la situation.

3.3 Procédé d'élimination de la coupure et relation d'équivalence sur les preuves

On appelle *procédé d'élimination de la coupure*, abrégé en p.é.c., le procédé décrit dans la preuve précédente. Celui-ci est caractérisé par l'ensemble de transformations dont il était question dans la preuve, mais aussi par un choix déterministe de transformation à appliquer selon la situation. On peut construire à partir de ce p.é.c. une relation d'équivalence sur les preuves de logique linéaire intuitionniste. Pour cela, on pose $p \triangleright p'$ si on peut passer de la preuve p à la preuve p' en appliquant le p.é.c., puis on considère la clôture symétrique et transitive de \triangleright . On appelle *équivalence selon le p.é.c.* la relation d'équivalence ainsi obtenue. On remarque que si deux preuves sont équivalentes selon le p.é.c., alors il s'agit de preuves du même séquent.

Dans le chapitre 3 de [1], le p.é.c. est défini très soigneusement, en ne laissant pas plus de liberté que celle qui est nécessaire pour éliminer effectivement toutes les utilisations de la règle de coupure. On qualifiera ce p.é.c. de “strict”. Il existe en effet un p.é.c. “large”, obtenu à partir du précédent en autorisant, dans certaines situations précises, l'application d'une transformation différente de celle indiquée par le p.é.c. “strict”. On verra que considérer la relation d'équivalence associée peut permettre d'obtenir des propriétés intéressantes sur la catégorie qu'on construira dans la partie suivante à partir des preuves de la logique linéaire intuitionniste.

4 Construction d'une catégorie et propriétés de cette catégorie

Cette section est fortement inspirée du chapitre 2 de [1]. On renvoie à celui-ci pour les démonstrations, qui s'appuient souvent sur certaines transformations du procédé d'élimination de la coupure figurant dans le chapitre 3 du même ouvrage et qu'on ne peut pas envisager de recopier ici intégralement.

4.1 Invariant modulaire et catégorie

À chaque preuve p , on associe une **dénotation** $[p]$. On veut que cela constitue un invariant selon l'élimination de la coupure : deux preuves ont la même dénotation si, et seulement si, elles sont équivalentes selon le procédé d'élimination de la coupure (cf. section précédente). Ceci est motivé par une analogie avec la théorie des nœuds, où les invariants sont relatifs aux transformations de Reidemeister.

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 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 $A \vdash D$:

$$\frac{\frac{\frac{\dots p_1 \dots}{A \vdash B} \quad \frac{\dots p_2 \dots}{B \vdash C}}{A \vdash C} (cut) \quad \frac{\dots p_3 \dots}{C \vdash D}}{A \vdash D} (cut) \quad \text{et} \quad \frac{\frac{\dots p_1 \dots}{A \vdash B} \quad \frac{\frac{\dots p_2 \dots}{B \vdash C} \quad \frac{\dots p_3 \dots}{C \vdash D}}{B \vdash D} (cut)}{A \vdash D} (cut)$$

qui sont équivalentes selon le p.é.c.. Cela signifie précisément que $[p_3] \circ ([p_2] \circ [p_1]) = ([p_3] \circ [p_2]) \circ [p_1]$. L'identité pour une formule A est la dénotation de la preuve $\frac{\dots}{A \vdash A} (id)$; on note cette dénotation id_A . Soit p une preuve de $A \vdash B$, les deux preuves $\frac{\dots}{A \vdash A} (id) \quad \frac{\dots p \dots}{A \vdash B} (cut)$

et $\frac{\dots p \dots}{A \vdash B}$ 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 opération binaire partielle \circ appelée composition, avec les propriétés suivantes.

- À chaque morphisme f est associé un couple d'objets (A, B) ; on écrit $f : A \rightarrow B$. On dit que $A \rightarrow B$ 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 $f : A \rightarrow B$ et $g : B \rightarrow C$, le morphisme $g \circ f$ existe et $g \circ f : A \rightarrow C$.
- **Identité.** Pour tout objet A , il existe un morphisme particulier $id_A : A \rightarrow A$ appelé identité sur A , tel que pour tout objet B , pour tout $f : B \rightarrow A$, $id_A \circ f = f$ et pour tout $g : A \rightarrow B$, $g \circ id_A = g$.
- **Associativité.** Pour tous $f : A \rightarrow B$, $g : B \rightarrow C$, $h : C \rightarrow D$, 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 $Hom_{A \rightarrow B}(\mathcal{C})$ la classe des morphismes de type $A \rightarrow B$. On peut omettre l'argument \mathcal{C} s'il n'y a pas d'ambiguïté, par exemple si on travaille sur une seule catégorie.

On construit la catégorie suivante, qu'on appellera \mathcal{CP} . À chaque formule A on associe une dénotation $[A]$. Les dénotations des formules 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. Ceci 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. On donne rapidement, à la fin de cette section, quelques éléments qui permettraient d'étendre ce formalisme à des preuves de n'importe quel séquent intuitionniste.

4.2 Produit tensoriel et bifoncteur

Le connecteur de logique linéaire \otimes , appelé *produit tensoriel*, induit un opérateur sur les dénотations de formules, aussi noté \otimes : on pose $[A] \otimes [B] = [A \otimes B]$. On souhaite étendre cet opérateur aux dénотations 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 $[p_1] \otimes [p_2] = [p]$.

$$\frac{\frac{\frac{p_1}{A_1 \vdash B_1} \dots \frac{p_2}{A_2 \vdash B_2}}{A_1, A_2 \vdash B_1 \otimes B_2} (\otimes R)}{A_1 \otimes A_2 \vdash B_1 \otimes B_2} (\otimes L)$$

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 $f_1 : A_1 \rightarrow B_1$, $f_2 : A_2 \rightarrow B_2$, $g_1 : B_1 \rightarrow C_1$, $g_2 : B_2 \rightarrow C_2$. Pour chaque morphisme f , soit $p(f)$ une preuve de dénотation f . Les deux preuves suivantes sont équivalentes selon le p.é.c., ce qui signifie que $(g_1 \otimes g_2) \circ (f_1 \otimes f_2) = (g_1 \circ f_1) \otimes (g_2 \circ f_2)$.

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

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

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

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** $F : \mathcal{C} \rightarrow \mathcal{D}$ 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 $f : A \rightarrow B$ de \mathcal{C} , on a $F(f) : F(A) \rightarrow F(B)$.
- Pour tous morphismes $f : A \rightarrow B$ et $g : B \rightarrow C$ 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 $Hom_{(\mathcal{C} \times \mathcal{D})}(A, A') \rightarrow (B, B') = Hom_{\mathcal{C}}(A \rightarrow B) \times Hom_{\mathcal{D}}(A' \rightarrow B')$.

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

Définition 12. Soit \mathcal{C} et \mathcal{D} des catégories. Soit $F, G : \mathcal{C} \rightarrow \mathcal{D}$ des foncteurs de \mathcal{C} vers \mathcal{D} . Une **transformation naturelle** θ de F vers G est une famille $(\theta_A : F(A) \rightarrow G(A))_{A \in \text{Obj}(\mathcal{C})}$ de morphismes de \mathcal{D} , indexée par les objets de \mathcal{C} , telle que le diagramme suivant commute dans \mathcal{D} pour tout morphisme $f : A \rightarrow B$ de \mathcal{C}

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

On note $\theta : F \Rightarrow G$, voire $\theta : F \Rightarrow G : \mathcal{C} \rightarrow \mathcal{D}$. On appelle **isomorphisme naturel** une transformation naturelle qui est une famille d'isomorphismes.

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

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

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

$$\begin{array}{ccccc} & & (A \otimes B) \otimes (C \otimes D) & & \\ & \nearrow \alpha_{A \otimes B, C, D} & & \searrow \alpha_{A, B, C \otimes D} & \\ ((A \otimes B) \otimes C) \otimes D & & & & A \otimes (B \otimes (C \otimes D)) \\ \downarrow \alpha_{A, B, C \otimes id_D} & & & & \uparrow id_A \otimes \alpha_{B, C, D} \\ (A \otimes (B \otimes C)) \otimes D & \xrightarrow{\alpha_{A, B \otimes C, D}} & & & A \otimes ((B \otimes C) \otimes D) \\ & & (A \otimes e) \otimes B & \xrightarrow{\alpha_{A, e, B}} & A \otimes (e \otimes B) \\ & \searrow \rho_A \otimes id_B & & \swarrow id_A \otimes \lambda_B & \\ & & A \otimes B & & \end{array}$$

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

Précision. En fait la condition sur α est $\alpha : F \Rightarrow G : \mathcal{C} \times \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ 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 $\lambda : (e \otimes \bullet) \Rightarrow Id_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{C}$ 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 .

Le nom de *catégorie monoïdale* vient du fait que l'ensemble des objets de la catégorie (s'il s'agit bien d'un ensemble) muni de \otimes devient un monoïde si on identifie la source et la cible de chaque isomorphisme donné par la définition.

On munit la catégorie \mathcal{CP} du bifoncteur \otimes et on choisit comme objet unité [1], la dénotation de la formule **1**, élément neutre pour le connecteur \otimes en logique linéaire intuitionniste. 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 dénnotations des preuves suivantes

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

Il s'agit bien de transformations naturelles, et on a bien la propriété de cohérence (cf. [1], chapitre 2). 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 $\bar{\alpha}_{A,B,C} : A \otimes (B \otimes C) \rightarrow (A \otimes B) \otimes C$, $\bar{\lambda}_A : A \rightarrow e \otimes A$, $\bar{\rho}_A : A \rightarrow A \otimes e$, par exemple $\bar{\lambda}_A$ est la dénotation de $\frac{\frac{}{A \vdash A} (id)}{\frac{1 \vdash A}{A \vdash 1 \otimes A} (1R)} (\otimes R)$. 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 : $\bar{\lambda} \circ \lambda = id_{e \otimes A}$, $\bar{\rho} \circ \rho = id_{A \otimes e}$, $\bar{\alpha} \circ \alpha = id_{(A \otimes B) \otimes C}$, $\alpha \circ \bar{\alpha} = id_{A \otimes (B \otimes C)}$. Si on considère au contraire la version plus "large" du p.é.c., on a bien toutes ces égalités, donc α , λ et ρ sont bien des isomorphismes. 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.

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

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

$$\begin{array}{ccccc} & & \alpha_{A,B,C} & \nearrow & A \otimes (B \otimes C) \xrightarrow{\gamma_{A,B \otimes C}} (B \otimes C) \otimes A \xrightarrow{\alpha_{B,C,A}} B \otimes (C \otimes A) \\ & & & & \searrow \\ (A \otimes B) \otimes C & & & & \\ & \searrow & \gamma_{A,B} \otimes id_C & \rightarrow & (B \otimes A) \otimes C \xrightarrow{\alpha_{B,A,C}} B \otimes (A \otimes C) \xrightarrow{id_B \otimes \gamma_{A,C}} B \otimes (C \otimes A) \\ & & \alpha_{A,B,C}^{-1} & \nearrow & (A \otimes B) \otimes C \xrightarrow{\gamma_{A \otimes B,C}} C \otimes (A \otimes B) \xrightarrow{\alpha_{C,A,B}^{-1}} (C \otimes A) \otimes B \\ A \otimes (B \otimes C) & & & & \\ & \searrow & id_A \otimes \gamma_{B,C} & \rightarrow & A \otimes (C \otimes B) \xrightarrow{\alpha_{A,C,B}^{-1}} (A \otimes C) \otimes B \xrightarrow{\gamma_{C,A} \otimes id_B} (C \otimes A) \otimes B \end{array}$$

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 $\gamma_{B,A} = \gamma_{A,B}^{-1}$. 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**.

4.5 Sur une généralisation à des preuves de n'importe quel séquent

Dans la définition de \mathcal{CP} donnée au début de cette section, on ne s'intéresse qu'aux preuves de séquents qui ont exactement une formule de chaque côté. Pour considérer la dénotation d'une preuve d'un séquent quelconque $\Gamma \vdash D$ comme un morphisme de \mathcal{CP} , on a besoin d'associer une dénotation à la liste Γ . Il serait naturel de choisir la dénotation de $G_1 \otimes \dots \otimes G_k$, mais ceci n'est pas une formule tant qu'on n'a pas ajouté de parenthèses. Lorsqu'on en ajoute, on obtient des formules qui sont certes équivalentes d'un point de vue logique, mais différentes d'un point de vue structurel, et les dénotations associées n'ont a priori pas de raison d'être égales. On peut résoudre ce problème grâce à une propriété de cohérence qui affirme que \mathcal{CP} est équivalente, dans un certain sens qui serait trop long à définir ici, à une catégorie monoïdale symétrique stricte \mathcal{C} ; *stricte* signifie que pour tous objets A, B, C de \mathcal{C} , avec les notations habituelles, on a $(A \otimes B) \otimes C = A \otimes (B \otimes C)$ et $e \otimes A = A = A \otimes e$.

Conclusion

Après avoir étudié le formalisme du calcul des séquents à travers différents exemples, on a transporté des propriétés sur un calcul des séquents pour ILL donné vers le langage mathématique des catégories. On peut encore interpréter le connecteur \multimap d'ILL en terme de catégorie monoïdale fermée, et le connecteur $\&$ en terme de catégorie cartésienne fermée, et la modalité $!$ en terme d'adjonction entre une catégorie monoïdale symétrique fermée et une catégorie cartésienne; mais étudier ces propriétés serait trop long pour le cadre de ce mémoire. Les catégories sont omniprésentes en mathématiques; ces résultats permettent donc d'envisager des liens entre le calcul des séquents et des domaines qui semblent très différents au premier abord. Il y a une analogie forte entre la théorie de la preuve vue à travers le calcul des séquents et la théorie des nœuds; les deux permettent notamment de construire de façon similaire une catégorie libre, présentant de nombreuses propriétés, à partir d'une catégorie donnée ([1], pp.34-38).

Rapport de stage : Recherche de preuves compilée et certifiée

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 des séquents bien choisis. Par exemple, l'équipe TYPES a développé un prouveur automatisé pour ILP appelé STRIP [5], basé sur une variante du calcul des séquents **LJT** [6]. Cependant, des structures de données sophistiquées sont nécessaires, si bien qu'une certification de ce prouveur serait difficile. Le calcul des séquents **LSJ** [7], 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 des séquents **LSJ**. Je l'ai testé sur les formules de la banque ILTP [8], 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 des séquents, et quelles propriétés du calcul des séquent peuvent rendre cet algorithme effectif et efficace. Je donne pour exemples **LJ** et **LJT**, deux calculs dont des variantes ont été effectivement employées 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 **LSJℓ**, qui permet d'obtenir facilement une bonne complexité spatiale, et je prouve que **LSJℓ** est bien équivalent à **LSJ**. En ce qui concerne l'implémentation, je donne quelques détails essentiels ou intéressants. 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 d'efficacité.

Table des matières

1 Propriétés favorisant l'application d'un calcul des séquents à la recherche automatisée de preuve et exemples de calculs	17
2 Le calcul des séquents utilisé pour l'implémentation : LSJ, légèrement modifié en LSJℓ	20
3 Éléments d'implémentation	24
4 Perspective de certification	27
5 Différentes variantes de prouveur et résultats obtenus	28

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

On utilise les notions et définitions suivantes, introduites dans la section 1 du mémoire : *calcul des séquents*, *séquent*, *règle*, *prémisse*, *conclusion*, *instance*, *preuve* ou *arbre de preuve*, *séquent prouvable dans un calcul*. De plus, une règle n'ayant aucune prémisse est appelée un *axiome*.

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 4, à l'exception de la règle d'échange $\frac{\Gamma_1, A, B, \Gamma_2 \vdash D}{\Gamma_1, B, A, \Gamma_2 \vdash D}$ (*permut. L*) qui peut être oubliée puisqu'un multiensemble ne tient pas compte de l'ordre.

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

Il existe plusieurs calculs des séquents pour la logique intuitionniste, sans parler des nombreuses autres logiques existantes. Un tel calcul se caractérise par sa propre définition des séquents, et ses propres règles. On veut aussi disposer d'une construction, pour chaque formule, d'un séquent qui est prouvable par le calcul si et seulement si la formule est prouvable en logique intuitionniste. Dériver d'un calcul des séquents pour la logique intuitionniste 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 des 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 des

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 des 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, c'est-à-dire 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 prémisses, 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 prémisses inversibles 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 prémisses 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** :

$$\frac{\Gamma \vdash A \quad \Gamma', A \vdash D}{\Gamma, \Gamma' \vdash D} \text{ (cut)}, \text{ rend l'algorithme proposé inutilisable parce qu'il ne termine jamais.}$$

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 3 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 des séquents

Un calcul des 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** :

$$\frac{A, A, \Gamma \vdash D}{A, \Gamma \vdash D} \text{ (contraction } L), \text{ sont problématiques pour la terminaison de l'algorithme. En}$$

effet, pour essayer de prouver $A, \Gamma \vdash D$, on peut être amené à essayer de prouver $A, A, \Gamma \vdash D$, puis en appliquant encore la même règle à essayer de prouver $A, A, A, \Gamma \vdash D$, 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. On définit récursivement que 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 des 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 des séquents. Souvent, elle fait partie des arguments qui permettent de montrer la terminaison. Elle fournit en effet un ordre bien fondé sur les formules ("est une sous-formule stricte de" étant une relation bien fondée), 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 section 3.2 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 prémissse $prem_k$ d'une règle $\frac{prem_1 \dots prem_p}{concl}(\mathcal{R})$ (aussi appelée *k-ième prémissse 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 prémissse 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 prémissse*, 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 pour toutes les prémisses de toutes ses instances, la différence d'information entre la prémissse et la conclusion est bornée par la taille en mémoire de la formule principale. 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 (par exemple numéro de prémissse et 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 prémissse, sur laquelle on relance l'algorithme. Et inversement, on a parfois besoin de revenir à la conclusion à partir d'une prémissse et des informations supplémentaires retenues : par exemple pour ensuite calculer une autre prémissse de l'instance, ou encore pour essayer d'appliquer une autre règle à la conclusion si on a trouvé une prémissse 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 des 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 $\frac{}{\Gamma, A \vdash A} (id)$. En revanche, on ne peut pas supprimer purement et simplement la règle $\frac{\Gamma, A, A \vdash D}{\Gamma, A \vdash D} (contraction L)$. 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 $\frac{\Gamma \vdash A \quad \Gamma, B \vdash D}{\Gamma, A \rightarrow B \vdash D} (\rightarrow L)$ par une seule règle $\frac{\Gamma, A \rightarrow B \vdash A \quad \Gamma, B \vdash D}{\Gamma, A \rightarrow B \vdash D} (\rightarrow L)$. On n'a alors plus d'appels récursifs sur des séquents strictement croissants $\Gamma, A \vdash D$ puis $\Gamma, A, A \vdash D$ puis $\Gamma, A, A, A \vdash D$ 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 $\frac{\Gamma, A \rightarrow B \vdash A \quad \Gamma, B \vdash A}{\Gamma, A \rightarrow B \vdash A} (\rightarrow L)$. On ajoute alors un système de détection de cycles en retenant tous les séquents rencontrés [9]. L'algorithme obtenu est correct et termine. On a la propriété de la sous-formule. Les règles sont toutes inversibles et locales sauf $\rightarrow L$, 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 [6] pour pallier le problème de cycles de **LJ**. Il n'y a pas de règle de contraction, et la règle $\rightarrow L$ est remplacée par quatre règles selon la structure de A dans la formule principale $A \rightarrow B$: par exemple $\frac{B, A, \Gamma \vdash D}{A \rightarrow B, A, \Gamma \vdash D} (\rightarrow L_1)$ où A doit être réduite à une variable, ou encore $\frac{A_1 \rightarrow (A_2 \rightarrow B), \Gamma \vdash D}{(A_1 \wedge A_2) \rightarrow B, \Gamma \vdash D} (\rightarrow L_2)$. 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 ordre bien fondé sur les formules puis sur les séquents. Toutes les règles sont inversibles et locales sauf une des règles qui remplacent $\rightarrow L$, qui comporte deux prémisses dont seule la deuxième est inversible. Une variante de **LJT** est le fondement du prouveur STRIP [5].

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

Le calcul des séquents utilisé pour l'implémentation est **LSJℓ**, une variante de **LSJ**. Le calcul **LSJ** est présenté par M. Ferrari, C. Fiorentini et G. Fiorino dans [7]. Il présente des propriétés très intéressantes pour l'application à la recherche de preuve. **LSJℓ** 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. **LSJℓ** hérite de toutes les bonnes propriétés de **LSJ**, en ajoutant la localité des règles.

2.1 Séquents et règles de **LSJ**

Définition 16. *Un séquent de **LSJ** est la donnée de trois multiensembles Θ , Γ et Δ de formules ; on écrit $\Theta ; \Gamma \vdash \Delta$.*

On a vu que dans les calculs **LK** et **LJ**, le séquent $\Gamma \vdash \Delta$ représente la formule $(\bigwedge_{G \in \Gamma} G) \rightarrow (\bigvee_{D \in \Delta} D)$, respectivement en logique classique et en logique intuitionniste, avec Δ contenant exactement une formule pour **LJ**. C'est une interprétation courante en calcul des séquents. Pour **LSJ**, on ne sait pas représenter un séquent $\Theta ; \Gamma \vdash \Delta$ par une seule formule. On a cependant le résultat suivant : un séquent $\emptyset ; \Gamma \vdash \Delta$ est prouvable dans **LSJ** si et seulement si la formule $(\bigwedge_{G \in \Gamma} G) \rightarrow (\bigvee_{D \in \Delta} D)$ 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 \rightarrow . L'article [7] propose bien une interprétation du séquent $\Theta ; \Gamma \vdash \Delta$ 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 17. *Soit A une formule, elle est valide en logique intuitionniste si et seulement si le séquent $\emptyset ; \emptyset \vdash A$ 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*.

$$\begin{array}{c}
\frac{}{\Theta ; \perp, \Gamma \vdash \Delta} (\perp L) \qquad \frac{}{\Theta ; A, \Gamma \vdash A, \Delta} (id) \\
\frac{\Theta ; A, B, \Gamma \vdash \Delta}{\Theta ; A \wedge B, \Gamma \vdash \Delta} (\wedge L) \qquad \frac{\Theta ; \Gamma \vdash A, \Delta \quad \Theta ; \Gamma \vdash B, \Delta}{\Theta ; \Gamma \vdash A \wedge B, \Delta} (\wedge R) \\
\frac{\Theta ; A, \Gamma \vdash \Delta \quad \Theta ; B, \Gamma \vdash \Delta}{\Theta ; A \vee B, \Gamma \vdash \Delta} (\vee L) \qquad \frac{\Theta ; \Gamma \vdash A, B, \Delta}{\Theta ; \Gamma \vdash A \vee B, \Delta} (\vee R) \\
\frac{\Theta ; B, \Gamma \vdash \Delta \quad B, \Theta ; \Gamma \vdash A, \Delta \quad B ; \Theta, \Gamma \vdash A}{\Theta ; A \rightarrow B, \Gamma \vdash \Delta} (\rightarrow L) \\
\frac{\Theta ; A, \Gamma \vdash B, \Delta \quad \emptyset ; A, \Theta, \Gamma \vdash B}{\Theta ; \Gamma \vdash A \rightarrow B, \Delta} (\rightarrow R)
\end{array}$$

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

$\wedge L$, $\wedge R$, $\vee L$ et $\vee R$ sont inversibles ; les deux premières prémisses de $\rightarrow L$ et la première prémisses de $\rightarrow R$ sont inversibles ; la troisième prémisses de $\rightarrow L$ et la deuxième prémisses de $\rightarrow R$ ne sont pas inversibles.

Non localité de certaines règles. Les règles $\rightarrow L$ et $\rightarrow R$ ne sont pas locales : pour chacune, les formules représentées par Δ dans la conclusion n'apparaissent nulle part dans la dernière prémisses, il n'est donc pas possible de retrouver la conclusion en connaissant uniquement cette prémisses, la formule principale et le numéro de la prémisses, 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 prémisses. Cette représentation est exhaustive et correcte. On définit en effet une surjection Φ 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 Φ est prouvable dans **LSJ**.

Définition 18. 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 Φ . 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 $\text{forget}(M)$ le multiensemble de formules obtenu en oubliant l'indice et ne gardant que la formule de chaque couple de M . On définit l'application Φ de l'ensemble des séquents de **LSJℓ** dans l'ensemble des séquents de **LSJ**,

qui à $\Gamma' \vdash_n \Delta'$ associe $\Theta; \Gamma \vdash \Delta$ où :

$$\begin{cases} \Theta = \text{forget}(\Gamma'_{n+1}) \\ \Gamma = \text{forget}(\Gamma'_{\leq n}) \\ \Delta = \text{forget}(\Delta'_n) \end{cases} . \text{ C'est une surjection : en}$$

effet tout séquent $\Theta; \Gamma \vdash \Delta$ de **LSJ** a au moins pour antécédent le séquent $\Gamma' \vdash_0 \Delta'$, 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 .	n et parfois i désignent toujours des entiers naturels, avec $i \leq n$	
	$\frac{}{i : \perp, \Gamma \vdash_n \Delta} (\perp L)$	$\frac{}{i : A, \Gamma \vdash_n n : A, \Delta} (id)$
	$\frac{i : A, i : B, \Gamma \vdash_n \Delta}{i : A \wedge B, \Gamma \vdash_n \Delta} (\wedge L)$	$\frac{\Gamma \vdash_n n : A, \Delta \quad \Gamma \vdash_n n : B, \Delta}{\Gamma \vdash_n n : A \wedge B, \Delta} (\wedge R)$
	$\frac{i : A, \Gamma \vdash_n \Delta \quad i : B, \Gamma \vdash_n \Delta}{i : A \vee B, \Gamma \vdash_n \Delta} (\vee L)$	$\frac{\Gamma \vdash_n n : A, n : B, \Delta}{\Gamma \vdash_n n : A \vee B, \Delta} (\vee R)$
	$\frac{i : B, \Gamma \vdash_n \Delta \quad n+1 : B, \Gamma \vdash_n n : A, \Delta \quad n+2 : B, \Gamma \vdash_{n+1} n+1 : A, \Delta}{i : A \rightarrow B, \Gamma \vdash_n \Delta} (\rightarrow L)$	
LSJℓ présente les mêmes propriétés que LSJ , auxquelles s'ajoute la localité de toutes les règles.	$\frac{0 : A, \Gamma \vdash_n n : B, \Delta \quad 0 : A, \Gamma \vdash_{n+1} n+1 : B, \Delta}{\Gamma \vdash_n n : A \rightarrow B, \Delta} (\rightarrow R)$	

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 $\sigma = \Phi(\sigma')$, on a $\vdash \sigma$ si et seulement si $\vdash' \sigma'$, où Φ 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 $\frac{\sigma_1 \quad \dots \quad \sigma_p}{\sigma} (\mathcal{R})$ et $\frac{\sigma'_1 \quad \dots \quad \sigma'_p}{\sigma'} (\mathcal{R}')$ des instances de ces règles.

Lemme 19. Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \Phi(\sigma')$ et soit \mathcal{R} une règle de **LSJ**.

- 1) Si $\frac{\sigma_1 \quad \dots \quad \sigma_p}{\sigma} (\mathcal{R})$ alors il existe $\sigma'_1, \dots, \sigma'_p$ tels que pour tout k , $\sigma_k = \Phi(\sigma'_k)$, et $\frac{\sigma'_1 \quad \dots \quad \sigma'_p}{\sigma'} (\mathcal{R}')$.
- 2) Si $\frac{\sigma'_1 \quad \dots \quad \sigma'_p}{\sigma'} (\mathcal{R}')$, posons pour tout k , $\sigma_k = \Phi(\sigma'_k)$, alors $\frac{\sigma_1 \quad \dots \quad \sigma_p}{\sigma} (\mathcal{R})$.

Pour un axiome \mathcal{A} , cela signifie simplement : $\frac{}{\sigma}(\mathcal{A})$ si et seulement si $\frac{}{\sigma'}(\mathcal{A}')$.

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

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

$\rightarrow L$: 1) Si $\frac{\sigma_1 \sigma_2 \sigma_3}{\sigma}(\rightarrow L)$ alors il existe A, B et $\tilde{\Gamma}$ tels que $\Gamma = A \rightarrow B, \tilde{\Gamma}$ et $\sigma_1 = \Theta; B, \tilde{\Gamma} \vdash \Delta$ et $\sigma_2 = B, \Theta; \tilde{\Gamma} \vdash A, \Delta$ et $\sigma_3 = B; \Theta, \tilde{\Gamma} \vdash A$; et il existe $i \leq n$ tel que $i : A \rightarrow B \in \Gamma'$. On pose $\tilde{\Gamma}' = \Gamma' - i : A \rightarrow B$ (on retire une seule occurrence de $i : A \rightarrow B$ de Γ') et $\sigma'_1 = i : B, \tilde{\Gamma}' \vdash_n \Delta'$ et $\sigma'_2 = n + 1 : B, \tilde{\Gamma}' \vdash_n n : A, \Delta'$ et $\sigma'_3 = n + 2 : B, \tilde{\Gamma}' \vdash_{n+1} n + 1 : A, \Delta'$ et on vérifie qu'on a bien $\sigma_1 = \Phi(\sigma'_1)$ et $\sigma_2 = \Phi(\sigma'_2)$ et $\sigma_3 = \Phi(\sigma'_3)$ (en remarquant que $\tilde{\Gamma} = \text{forget}(\tilde{\Gamma}'_{\leq n})$), et aussi $\frac{\sigma'_1 \sigma'_2 \sigma'_3}{\sigma'}(\rightarrow L')$.

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

Théorème 20. Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \Phi(\sigma')$, 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 $\sigma = \Theta; \Gamma \vdash \Delta$ tel que toutes les formules dans Γ et dans Δ sont atomiques : soit $\sigma' = \Gamma' \vdash_n \Delta' \in \mathfrak{S}'$ tel que $\sigma = \Phi(\sigma')$. 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 $\sigma = \Theta; \Gamma \vdash \Delta \in \mathfrak{S}$. Soit $\sigma' = \Gamma' \vdash_n \Delta' \in \mathfrak{S}'$ tel que $\sigma = \Phi(\sigma')$.

On suppose $\vdash \sigma$. Alors il existe une règle \mathcal{R} de **LSJ** et $\sigma_1, \dots, \sigma_p \in \mathfrak{S}$ (avec éventuellement p nul) tels que $\vdash \sigma_k$ pour tout k et $\frac{\sigma_1 \dots \sigma_p}{\sigma}(\mathcal{R})$. D'après le lemme, il existe $\sigma'_1, \dots, \sigma'_p \in \mathfrak{S}'$ tels que $\sigma_k = \Phi(\sigma'_k)$ pour tout k et $\frac{\sigma'_1 \dots \sigma'_p}{\sigma'}(\mathcal{R}')$. 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 $\sigma'_1, \dots, \sigma'_p \in \mathfrak{S}'$ tels que $\vdash' \sigma'_k$ pour tout k et $\frac{\sigma'_1 \dots \sigma'_p}{\sigma'}(\mathcal{R}')$. On pose $\sigma_k = \Phi(\sigma'_k)$ pour tout k . D'après le lemme on a $\frac{\sigma_1 \dots \sigma_p}{\sigma}(\mathcal{R})$, en particulier on peut appliquer l'hypothèse de récurrence aux σ_k donc $\vdash \sigma_k$ pour tout k , d'où $\vdash \sigma$. \square

3 Éléments d'implémentation

On utilise l'algorithme dont le pseudo-code est donné dans [7]. Il s'agit de l'algorithme "générique" 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 [7], 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 \perp 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 admet une formule principale 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 $\Theta ; \Gamma \vdash \Delta$ 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
<i>atomique</i>	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 \perp 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 $\vdash A$, 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 \perp 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 cellules 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 $H = A \rightarrow B$ 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 $\vdash A$, on peut associer un côté à chaque numéro représentant une sous-formule de A .

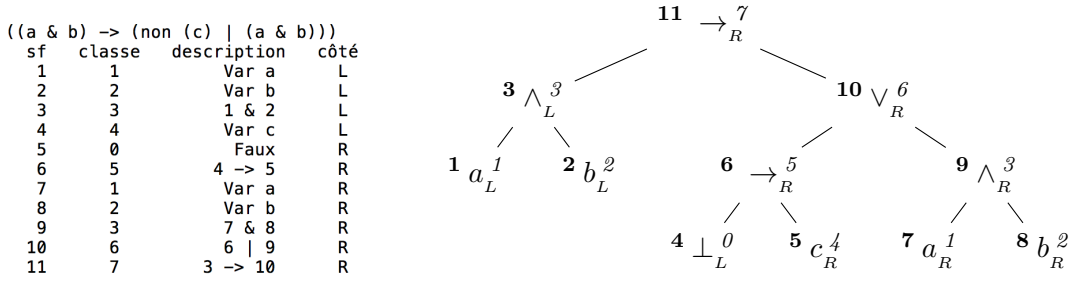


FIGURE 9 – Indexation de la formule $(a \wedge b) \rightarrow (\neg c \vee (a \wedge b))$. “sf” comme “sous-formule” est le numéro associé à une formule (en gras dans l’arbre), “description” ce qui est stocké dans le tableau. La classe d’une formule (en italique) est expliquée dans la prochaine sous-section.

3.3 Classes d’égalité structurelle pour l’axiome id

L’axiome $\frac{i : A, \Gamma \vdash_n n : A, \Delta}{i : A, \Gamma \vdash_n n : A, \Delta} (id) \quad (i \leq n)$ 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 la 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 σ , de la forme $\Gamma \vdash \Delta$, 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 $H = A \text{ c } B$, 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 cellules, une par priorité. Dans la cellule 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 utile 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 prémisses, et celui où on repasse de la prémisses à la conclusion, la représentation des multiensembles de la prémisses 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 “implicates”, 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 $\tilde{\Gamma}$ et $\tilde{\Delta}$. On retient également deux variables booléennes *fauxL* et *id*. Lorsqu'on ajoute une formule à Δ , on met à jour $\tilde{\Delta}$, et on vérifie si la classe de la formule ajoutée est présente et “accessible” dans $\tilde{\Gamma}$, auquel cas on assigne *vrai* à *id*. De même pour un ajout à Γ en échangeant les rôles de $\tilde{\Delta}$ et $\tilde{\Gamma}$, mais en plus, si la formule ajoutée est \perp avec un indice inférieur à celui du séquent, on assigne *vrai* à *fauxL*. Lorsqu'on transforme une prémisses 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 $\tilde{\Gamma}$ et $\tilde{\Delta}$ avec une structure de données similaire à celle pour Γ et Δ : un tableau indexé par les classes

et dans la cellule cl , une liste de couples (*un indice i , le nombre de couples $i:cl$ dans $\tilde{\Gamma}$ resp. $\tilde{\Delta}$*). 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 (en Coq) 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. Voici deux changements favorables à la certification qui ont été utilisés dans l’implémentation de certaines variantes de prouveurs.

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 **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 $\langle ., . \rangle$ 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.

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

FIGURE 10 – Les expressions du langage **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 **M** le langage associé à cette machine abstraite. Justement, l’objectif de ces travaux sur **T** serait, à terme, l’utilisation de ce langage (ou plutôt d’une version améliorée de ce langage) pour écrire des prouveurs certifiés.

4.2 Compilation d’un programme spécifique à une formule donnée

On a vu en 3.1 que pour une formule non atomique donnée d’un séquent, il y a exactement une instance dont le séquent est conclusion avec cette formule principale, et la règle associée ne dépend 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 prémisses, de numéro k , de l’unique règle associée, des fonctions de transformation des 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 \mathbf{T} des fonctions associées aux sous-formules. On ajoute ensuite ces fonctions à du code en \mathbf{T} ne dépendant pas de la formule, pour obtenir un code intégralement en \mathbf{T} dont l'exécution doit renvoyer la prouvabilité de la formule initiale.

5 Différentes variantes de prouveur et résultats obtenus

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 \mathbf{T} (langage défini, comme \mathbf{M} , en 4.1). 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 \mathbf{T} , donc on a de toute façon besoin d'un analyseur pour \mathbf{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 5.1.

Afin de tester ces prouveurs, on a utilisé la bibliothèque ILTP [8], 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.

Les tests ont été effectués sous MacOS avec un processeur 2.7 GHz Intel Core i7.

En 5.2, on compare notre prouveur le plus efficace ("prouveur simple" dans 5.1) aux prouveurs répertoriés par ILTP. En 5.3, 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** [7], en utilisant tout ce qui est décrit dans la section 3, notamment les structures de données élaborées de 3.4. En revanche, aucun effort n’est fait pour faciliter la certification.
- **“Prouveur simple avec listes”**. Ce prouveur est similaire à celui ci-dessus, mais les structures de données utilisées pour le séquent sont de simples listes. Cela permet une comparaison plus juste avec les prouveurs suivants, pour lesquelles les structures de données élaborées de 3.4 ne sont pas implémentées. Ce serait possible de les implémenter, mais ce serait fastidieux, et n’apporterait pas grand-chose par rapport à l’investissement, dans la mesure où ce qui nous intéresse est la variation d’efficacité liée aux changements qu’on introduit ; utiliser ce prouveur comme témoin est une solution plus facile et tout à fait convenable.
- **“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 4.2, 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 l’exécution du code en M par la machine abstraite simple à laquelle ce langage correspond, en utilisant les arbres de caml plutôt qu’en manipulant explicitement des pointeurs. (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 [7], avec de légers changements comme la transformation du calcul en **LSJℓ**, se compare honorablement aux quatre prouveurs répertoriés par ILTP, malgré une implémentation pas toujours optimisée et l’utilisation d’une machine personnelle peu performante. Pour chaque formule de la bibliothèque est indiqué le nombre de prouveurs, parmi les quatre répertoriés, qui terminent sur cette entrée en moins de cinq minutes. On a effectué les tests sur notre prouveur avec la même limitation de cinq minutes. Comme on peut s’y attendre, il y a de nombreuses formules sur lesquelles ce prouveur ne termine pas, contrairement à tous ceux d’ILTP. Mais il y a aussi deux formules pour lesquels notre prouveur termine alors que deux des quatre prouveurs répertoriés ne terminent pas, et une autre pour lequel le nôtre termine mais un des quatre non. Et enfin, il y a la famille de formules SYJ209, dont on parle plus loin.

Cette efficacité est due aux priorités de **LSJ**. On remarque que les formules sur lesquelles notre prouveur est particulièrement performant sont les formules comportant beaucoup de \rightarrow , ce qui n'est pas surprenant vu que les nouveautés de ce calcul concernent uniquement les règles liées à ce connecteur.

Sur une dizaine de formules de la famille SYJ209 (formules qui ont la même forme mais avec un nombre de variables variant entre une dizaine et une vingtaine), aucun des quatre prouveurs répertoriés par ILTP ne termine en moins de cinq minutes, alors que notre prouveur termine en moins d'une milliseconde. Ce sont des formules non prouvables pour lesquelles, au cours de la recherche de preuve, on se retrouve rapidement avec un séquent avec environ autant de formules de la forme $A \rightarrow B$ à droite que le nombre de variables, et dans lequel il n'y a pas de formule plus prioritaire que celles avec \rightarrow . Il se trouve qu'ici, lorsqu'on applique la règle $\rightarrow R$ avec n'importe quelle formule principale, on obtient que la première prémisse, celle qui est inversible, est non prouvable ; de plus on ne rajoute que des formules atomiques, donc on se retrouve avec un séquent similaire au précédent, avec une formule composée de moins. La recherche de preuve termine alors en temps **linéaire** en le nombre de variables, une fois qu'on a appliqué $\rightarrow R$ à chacune des formules $A \rightarrow B$ dans n'importe quel ordre. Ce qui arrive vraisemblablement avec les prouveurs d'ILTP qui sont aussi basés sur un calcul des séquents, c'est que les prémisses non prouvables qu'on rencontre ne sont pas inversibles. Ainsi, lorsqu'on a n formules en \rightarrow dans le séquent, on doit toutes les essayer successivement, et lorsqu'on en essaie une, on se retrouver avec $n - 1$ formules qu'on doit de nouveau toutes essayer successivement, et ainsi de suite, ce qui représente **au moins $n!$** essais de règles, où n est à peu près le nombre de variables : on comprend que ce soit long déjà pour une dizaine de variables.

5.3 Comparaison entre les différents prouveurs implémentés

On a effectué des tests avec les différents prouveurs réalisés. Pour ceux qui sont en plusieurs étapes, on détaille les temps de chacune. On renvoie également le nombre d'appels à la fonction récursive principale, qui applique la stratégie de recherche de preuve à un séquent. Ci-contre un exemple de résultats (formule SYJ201+1.002 d'ILTP [8]) ; “1ère compil.” est la compilation d'un programme en **T** ou en OCaml adapté à la formule en entrée, “2e compil.” la compilation de **T** vers **M** ou la compilation par ocamlc ; quelle que soit la méthode, on a ici 71036 appels à la fonction récursive principale. En général, comme on peut s'y attendre, l'exécution, au cours de laquelle a lieu la recherche de preuve à proprement parler, est ce qui prend le plus de temps, sauf dans le cas de petites formules où tout est négligeable sauf la compilation par ocamlc.

	“simple listes”	“compilé Caml”	“ T ”	“ T M ”
total	0.08 s	.61 s	14 s	57 s
1ère compil.	—	0.01 s	0.02 s	0.02 s
2ème compil.	—	0.28 s	—	0.02 s
exécution	—	0.32 s	14 s	57 s

Le “prouveur simple”, même dans la version qui utilise de simples listes, reste le plus efficace. Plus on y apporte de modifications, plus l'efficacité diminue, et elle diminue beaucoup. Ce n'est pas très surprenant, ni rédhibitoire. En effet, ces variantes ont encore des défauts flagrants, dont certains ne seraient pas trop difficiles à corriger. Par exemple, le langage **T** ne permet pas d'effets de bords, si bien qu'on crée une nouvelle variable à chaque fois qu'on modifie le séquent ; finalement, bien qu'il n'y ait qu'un seul séquent à la fois sur lequel on est susceptible de travailler, on garde de nombreux séquents en mémoire au cours des appels récursifs. L'ajout d'effets de bord est en général défavorable à la certification, mais on pourrait ici n'avoir que des effets de bords très limités, donc pas trop de difficultés supplémentaires : par exemple, ne disposer que d'une seule variable globale modifiable, dans laquelle on stockerait

le séquent. L'utilisation de la machine abstraite liée au langage \mathbf{M} est aussi très décevante, car beaucoup moins efficace qu'un simple interpréteur. Lorsqu'on observe un code en \mathbf{M} autogénéré, on constate qu'il y a de très nombreuses opérations inutiles, comme la sauvegarde, lors d'appels de fonctions ou de "let ... in", de variables qui en fait ne seront plus utilisées. Encore une fois, cela s'arrangerait un peu avec des effets de bords, ou même une possibilité d'autoriser l'oubli de certaines variables. Mais le mieux serait encore de coder la recherche de preuve en récursif terminal, ce qui est possible grâce à la localité des règles de $\mathbf{LSJ\ell}$, et d'optimiser le code en conséquence.

Il reste donc beaucoup à faire, mais l'implémentation d'une recherche de preuve compilée et certifiée basée sur le calcul des séquents \mathbf{LSJ} semble vraiment envisageable. Bien sûr, un prouveur facile à certifier restera sans doute moins efficace qu'un prouveur qu'on pourrait réaliser sans cette contrainte, mais peut-être pourra-t-on obtenir une perte d'efficacité suffisamment faible pour être compensée par l'intérêt que présenterait un prouveur certifié.

Conclusion

Au cours de ce stage, j'ai confirmé que \mathbf{LSJ} est un calcul des séquents particulièrement intéressant dans le cadre de la recherche automatisée de preuves en logique intuitionniste, puisque même avec une implémentation peu optimisée et une machine personnelle peu performante, la comparaison avec d'autres prouveurs existants est encourageante. J'ai même trouvé une formule (appartenant à la banque ILTP) sur laquelle aucun prouveur répertorié ne termine en moins de cinq minutes, alors que celui basé sur \mathbf{LSJ} termine en moins d'une milliseconde, ce qui illustre l'intérêt d'avoir des prémisses inversibles dans les règles liées au connecteur \rightarrow . J'ai aussi exploré des modifications à apporter à ce prouveur afin de le rendre plus facile à certifier ; notamment, j'ai implémenté la compilation, à partir d'une formule donnée, d'un programme dont l'exécution renvoie un booléen indiquant si la formule est prouvable, écrit dans un langage très simple, où le seul type de données consiste en des arbres binaires construits à partir d'un arbre vide et de feuilles étiquetées par des entiers. Les méthodes employées pour faciliter la certification ont pour l'instant un impact fort sur l'efficacité, mais il y a de nombreuses améliorations à envisager, par exemple l'ajout d'effets de bords dans le langage \mathbf{T} , qui compliquerait un peu la certification mais pas trop car ils pourraient être très limités : ne disposer que d'une seule variable globale modifiable permettrait déjà une amélioration importante.

Ce stage m'a appris à travailler sur un programme qu'on n'a jamais fini d'améliorer, pour lequel une factorisation la plus importante possible du code et une automatisation d'un grand nombre de tests sont extrêmement utiles. Factoriser le code était particulièrement souhaitable pour les nombreux changements apportés aux structures de données représentant les multiensembles du séquent, et puis, surtout, pour implémenter plusieurs prouveurs distincts qui partagent, par exemple, la phase d'indexation. En plus de la recherche de preuve à proprement parler, il y avait aussi beaucoup de fonctionnalités auxiliaires à implémenter, par exemple deux analyseurs : un pour les fichiers d'ILTP et un pour le langage \mathbf{T} . Enfin, j'ai eu à choisir, à la suite de discussions avec mes encadrants, quelles variantes de prouveur me semblaient les plus intéressantes et pratiques à implémenter.

Références

- [1] P.-A. Melliès, “Categorical semantics of linear logic,” *Panoramas et Synthèses*, vol. 27, pp. 15–215, 2009.
- [2] A. S. Troelstra, *Lectures on linear logic*. 1992.
- [3] J.-Y. Girard, “Linear logic : Its syntax and semantics,” in *Proceedings of the Workshop on Advances in Linear Logic*, (New York, NY, USA), pp. 1–42, Cambridge University Press, 1995.
- [4] D. Larchey-Wendling, *Preuves, réfutations et contre-modèles dans des logiques intuitionnistes*. PhD thesis, Nancy 1, 2000. Th. doct. : informatique.
- [5] D. Galmiche, D. Larchey-Wendling, and D. Méry, “STRIP : Structural sharing and intuitionistic proof-search,” Interne A00-R-342 — galmiche00h, 2000. Rapport interne.
- [6] R. Dyckhoff, “Contraction-free sequent calculi for intuitionistic logic.,” *J. Symb. Log.*, vol. 57, no. 3, pp. 795–807, 1992.
- [7] 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.
- [8] T. Rath, J. Otten, and C. Kreitz, “The ILTP Library.” <http://www.cs.uni-potsdam.de/ti/iltp/>.
- [9] J. M. Howe, “Two loop detection mechanisms : a comparison,” in *Automated Reasoning with Analytic Tableaux and Related Methods*, pp. 188–200, Springer, 1997.