Théorie de la démonstration linéaire et logique classique

Olivier LAURENT
Preuves, Programmes et Systèmes
CNRS – Université Paris VII
UMR 7126 – Case 7014
175, rue du Chevaleret – 75013 Paris – FRANCE
Olivier.Laurent@pps.jussieu.fr

Notes de septembre 1999

Introduction

La théorie de la démonstration, introduite suite au deuxième problème posé par D. Hilbert en 1900, a pour objet l'étude des systèmes de démonstrations formelles (assemblages de règles syntaxiques simples). Elle s'est avérée par la suite être fortement liée à l'informatique théorique, et plus particulièrement à la théorie des langages de programmation.

L'un des outils de base de cette dernière est le λ-calcul. Il s'agit d'un langage très simple (trois règles syntaxiques) mais permettant de coder n'importe quelle fonction récursive. La version typée ajoute de la modularité et représente le fondement théorique de nombreux langages de programmation (ML, Miranda, ...).

Le λ-calcul typé entretient des rapports étroits avec la logique à travers la *correspondance de Curry-Howard* qui identifie les types du langage à des formules de la *logique intuitionniste* (une formalisation des mathématiques intuitionnistes de L. E. J. Brouwer) et les termes (programmes) à des démonstrations formelles intuitionnistes. Cette idée coïncide avec celle de la *sémantique de A. Heyting* qui consiste à interpréter les preuves de formules de la forme $A \to B$ comme des fonctions qui transforment toute preuve de A en une preuve de B, ce qui est proche de l'idée d'un programme transformant des paramètres d'entrée en une réponse.

Ainsi on a transformé l'étude des langages de programmation en celle des systèmes logiques, permettant :

- d'importer en informatique théorique les techniques développées par les logiciens,
- de développer de nouveaux systèmes logiques basés sur le point de vue calculatoire.

Une logique peut alors se définir comme un formalisme (*calcul des séquents*) muni d'une procédure de transformation des preuves (*élimination des coupures*) qui constitue le pendant logique de l'exécution des programmes par la correspondance de Curry-Howard.

La règle logique de coupure correspond à l'utilisation de lemmes ou de théorèmes généraux dans une démonstration ce qui constitue un procédé fondamental en mathématiques. Le processus de réécriture des preuves qu'est l'élimination des coupures effectue l'explicitation du contenu vraiment utile du lemme dans la preuve dans laquelle il est utilisé. Cette transformation des démonstrations formelles se traduit en général par une explosion de la taille des preuves montrant à quel point l'utilisation d'un lemme peut être synthétique.

Une des applications les plus marquantes de l'introduction d'idées calculatoires en logique est la *logique linéaire* due à J.-Y. Girard en 1986. Elle définit de nouveaux connecteurs plus primitifs que les connecteurs usuels. Ce système, encore plus contraint que le système intuitionniste, n'en est pas moins suffisamment puissant pour que l'on puisse y traduire la logique intuitionniste et la logique classique. En particulier, la découverte de bonnes traductions de la logique classique dans la logique linéaire donne une solution élégante au problème de l'extension de la correspondance de Curry-Howard au travers des systèmes classiques constructifs (comme LC).

Nous allons étudier les différents systèmes logiques que l'on a cités à travers différentes syntaxes. Le *calcul des séquents* de G. Gentzen restera pour nous la syntaxe de référence parce qu'il possède une règle de coupure. Cependant l'élimination des coupures est compliquée par des étapes de calcul sans grand intérêt. La *déduction naturelle* résout ces problèmes pour la logique intuitionniste mais au prix d'une rupture des symétries. Par ailleurs elle ne s'applique que mal à la logique classique. Pour ce qui est de la logique linéaire, une nouvelle syntaxe a été introduite : les *réseaux de preuve* qui, tout en préservant les symétries du calcul des séquents, possèdent une élimination des coupures plus facile à étudier. Cette syntaxe de graphes a permis d'introduire une interprétation géométrique des preuves. Une variante de ces réseaux, les *réseaux polarisés*, offre une syntaxe alternative au calcul des séquents pour les systèmes constructifs de logique classique.

On peut citer comme références dans le domaine le livre de J.-L. Krivine sur le λ-calcul [4] et celui de J.-Y. Girard, Y. Lafont et P. Taylor pour la théorie de la démonstration [3].

1 Logique classique

La logique classique correspond à la notion usuelle de formule vraie en mathématiques c'est-à-dire la vérité obtenue par les traditionnelles tables de vérité. De nombreux systèmes de preuve ont été proposés pour prouver cet ensemble de formules par des démonstrations formelles parmi lesquels on peut citer celui de D. Hilbert qui utilise trois axiomes et une seule règle de déduction (*modus ponens*). Cependant ce système est très difficilement utilisable pour étudier la structure des preuves (démontrer $A \to A$ n'est déjà pas totalement trivial...).

On va s'intéresser ici au système introduit plus tard par G. Gentzen : le *calcul des séquents* qui est l'un des plus utilisés en particulier parce qu'il peut être appliqué à la plupart des systèmes logiques.

1.1 Formules et séquents

On considère les formules construites à l'aide des *connecteurs* $\neg$ (négation), $\wedge$ (conjonction), $\vee$ (disjonction) et $\to$ (implication) à partir de variables (ou formules atomiques). Ces formules correspondent à ce que l'on appelle le *calcul propositionnel*.

Même si l'on souhaite simplement démontrer des formules, il est nécessaire d'utiliser à l'intérieur des preuves des structures un peu plus complexes que sont les *séquents*. Ce sont des paires de suites de formules que l'on note $A_1, ..., A_n \vdash B_1, ..., B_m$ où l'ordre entre les A_i et entre les B_j n'a pas d'importance. Un tel séquent signifie qu'à partir des hypothèses A_1, ..., A_n on peut déduire l'une des conclusions B_1, ..., B_m, ce qui peut se représenter par la formule $(A_1 \wedge ... \wedge A_n) \to (B_1 \vee ... \vee B_m)$. Le cas d'une démonstration d'une simple formule A correspond à une démonstration du séquent $\vdash A$.

1.2 Calcul des séquents LK

Groupe identité

$$\frac{}{A \vdash A}\,ax \qquad \frac{\Gamma \vdash A, \Delta \quad \Sigma, A \vdash \Xi}{\Gamma, \Sigma \vdash \Delta, \Xi}\,cut$$

La règle *ax* est indispensable dans ce système , c'est elle qui commence toute preuve car elle est la seule sans *prémisses* (c'est-à-dire sans hypothèses), elle permet de former les feuilles des arbres de démonstration.

La règle de *coupure* (*cut*) est la "clef du raisonnement", elle correspond à l'utilisation des lemmes en mathématiques. En effet, considérons le cas particulier où Δ est vide et Ξ est constitué d'une seule formule B et lisons la règle obtenue $\frac{\Gamma \vdash A \quad \Sigma, A \vdash B}{\Gamma, \Sigma \vdash B}$ de bas en haut : on cherche à démontrer la formule B à l'aide des hypothèses Γ, Σ, pour cela on peut tout d'abord démontrer le lemme A à partir d'une partie des hypothèses (Γ) puis utiliser ce lemme dans la suite de la preuve ainsi que le reste des hypothèses (Σ) pour démontrer B.

Groupe structurel

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\,\mathit{aff}_g \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\,\mathit{aff}_d$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\,\mathit{contr}_g \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\,\mathit{contr}_d$$

Ces règles spécifient la manière dont peuvent être utilisées les formules. En particulier les règles à gauche sont particulièrement simples à comprendre : l'affaiblissement dit que si l'on ajoute des hypothèses la preuve reste valide et la contraction dit qu'il est inutile d'avoir plusieurs fois la même hypothèse (ou encore que l'on peut utiliser une hypothèse autant de fois que l'on veut). Les versions à droite de ces règles sont nécessaires afin de rendre compte de la symétrie de la logique classique.

On verra, avec l'introduction d'autres systèmes logiques, comment les modifications de ces règles ont des conséquences fondamentales sur les propriétés obtenues.

Groupe logique Les règles précédentes montrent comment on gère les formules à travers les preuves, nous allons maintenant voir comment on construit ces formules à l'aide des connecteurs logiques.

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\,\neg_g \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\,\neg_d$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,\wedge_g^1 \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,\wedge_g^2 \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\,\wedge_d$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\,\vee_g \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}\,\vee_d^1 \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\,\vee_d^2$$

$$\frac{\Gamma \vdash A, \Delta \quad \Sigma, B \vdash \Xi}{\Gamma, \Sigma, A \rightarrow B \vdash \Delta, \Xi}\,\rightarrow_g \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \rightarrow B, \Delta}\,\rightarrow_d$$

Dans le cas où Δ est vide, la règle $\wedge_d$ signifie, par exemple, que si l'on démontre A à partir des hypothèses Γ et par ailleurs B à partir des mêmes hypothèses alors Γ permet de démontrer $A \wedge B$.

Dans les règles qui précèdent, Γ, Δ, Σ et Ξ sont appelés *contextes* des règles, ce sont des ensembles de formules qui ne sont pas directement modifiées par la règle.

Théorème 1 (Complétude et correction)
Le séquent $\vdash A$ est prouvable en calcul des séquents si et seulement si pour toute valeur vrai ou faux donnée à chaque variable de la formule A, la valeur de A obtenue par les tables de vérité est vrai.

Ce résultat montre que le calcul des séquents prouve bien exactement les formules de la logique classique comme attendu.

Théorème 2 (Hauptsatz de Gentzen)
Si le séquent $\Gamma \vdash \Delta$ est prouvable en calcul des séquents alors il l'est sans utiliser la règle de coupure.

Ce théorème d'*élimination des coupures* reflète les profondes symétries du calcul des séquents (entre règles à droite du $\wedge$ et règles à gauche du $\vee$ et inversement, entre gauche et droite du signe $\vdash$, ...). Il constitue un des résultats les plus importants de la théorie de la démonstration et entraîne en particulier le résultat suivant :

Corollaire 2.1 (Propriété de la sous-formule)
Si le séquent $\Gamma \vdash \Delta$ est prouvable en calcul des séquents alors il l'est en n'utilisant que des sous-formules de Γ et Δ.

Cette propriété montre que, pour prouver une formule, il est inutile d'utiliser des principes qui n'apparaissent pas dans la formule en question. En effet, si l'on regarde les règles de bas en haut, la règle de coupure est la seule à introduire une nouvelle formule. Ainsi la règle de coupure, pourtant essentielle dans les raisonnements mathématiques, n'est jamais indispensable pour prouver une formule.

En terme de recherche (automatique par exemple) de preuve, il suffit de chercher une démonstration n'utilisant que des sous-formules ce qui restreint l'espace dans lequel on la recherche.

On peut voir maintenant un exemple simple, mais tout à fait caractéristique, d'utilisation de ce théorème :

Corollaire 2.2 (Non contradiction)
La formule $A \wedge \neg A$ n'est prouvable pour aucune formule A en calcul des séquents.

Démonstration : On raisonne par l'absurde en supposant qu'il existe une preuve du séquent $\vdash A \wedge \neg A$ en calcul des séquents. En introduisant une règle de coupure entre cette preuve et la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax}{A, \neg A \vdash}\,\neg_g}{A \wedge \neg A, \neg A \vdash}\,\wedge_g^1}{A \wedge \neg A, A \wedge \neg A \vdash}\,\wedge_g^2}{A \wedge \neg A \vdash}\,contr_g$$

on obtient une démonstration du séquent $\vdash$, or d'après le théorème 2 ce séquent est alors prouvable sans règle de coupure ce qui est impossible puisqu'aucune dernière règle ne pourrait s'appliquer.

La preuve du théorème 2 explicite un algorithme qui fait remonter les règles de coupure jusqu'aux axiomes et les élimine. Les exemples ci-dessous donnent quelques étapes de cet algorithme.

Exemple 1 (Élimination des coupures : cas clefs)
Élimination d'une coupure sur un axiome :

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\AxiomC{π}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma \vdash A, \Delta$}
\RightLabel{cut}
\BinaryInfC{$\Gamma \vdash A, \Delta$}
\end{prooftree}
\quad \rightarrow \quad
\begin{prooftree}
\AxiomC{π}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma \vdash A, \Delta$}
\end{prooftree}
$$

Élimination de la coupure entre une règle à gauche et une règle à droite du $\wedge$:

$$
\begin{prooftree}
\AxiomC{π_1}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma, A \vdash \Delta$}
\RightLabel{$\wedge_g^1$}
\UnaryInfC{$\Gamma, A \wedge B \vdash \Delta$}
\AxiomC{π_2}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Sigma \vdash A, \Xi$}
\AxiomC{π_3}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Sigma \vdash B, \Xi$}
\RightLabel{$\wedge_d$}
\BinaryInfC{$\Sigma \vdash A \wedge B, \Xi$}
\RightLabel{cut}
\BinaryInfC{$\Gamma, \Sigma \vdash \Delta, \Xi$}
\end{prooftree}
\quad \rightarrow \quad
\begin{prooftree}
\AxiomC{π_1}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma, A \vdash \Delta$}
\AxiomC{π_2}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Sigma \vdash A, \Xi$}
\RightLabel{cut}
\BinaryInfC{$\Gamma, \Sigma \vdash \Delta, \Xi$}
\end{prooftree}
$$

Outre les cas dits *clefs* (ou *étapes logiques*) dont font partie les deux exemples ci-dessus, de nombreuses étapes de l'élimination des coupures en calcul des séquents se contentent de faire permuter la règle de coupure avec la règle qui précède quand la formule coupée est dans le contexte de cette règle.

Exemple 2 (Élimination des coupures : cas commutatif)
Élimination d'une coupure dans le contexte d'une règle de $\vee$ à droite :

$$
\begin{prooftree}
\AxiomC{π_1}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma \vdash A, C, \Delta$}
\RightLabel{$\vee_d^1$}
\UnaryInfC{$\Gamma \vdash A \vee B, C, \Delta$}
\AxiomC{π_2}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Sigma, C \vdash \Xi$}
\RightLabel{cut}
\BinaryInfC{$\Gamma, \Sigma \vdash A \vee B, \Delta, \Xi$}
\end{prooftree}
\quad \rightarrow \quad
\begin{prooftree}
\AxiomC{π_1}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma \vdash A, C, \Delta$}
\AxiomC{π_2}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\Sigma, C \vdash \Xi$}
\RightLabel{cut}
\BinaryInfC{$\Gamma, \Sigma \vdash A, \Delta, \Xi$}
\RightLabel{$\vee_d^1$}
\UnaryInfC{$\Gamma, \Sigma \vdash A \vee B, \Delta, \Xi$}
\end{prooftree}
$$

Certaines de ces *étapes commutatives* n'ont, d'un point de vue logique, que peu d'intérêt ce qui encourage à rechercher d'autres syntaxes comme la *déduction naturelle* (section 2.2) ou les *réseaux de preuve* (section 5.2) pour lesquelles il y ait le moins possible de ces étapes.

Exemple 3 (Non identification de preuves sans coupures)
Même dans le cas sans coupures, le calcul des séquents différencie par des commutations de règles des preuves dont la structure est fondamentalement la même.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\wedge_g^1$}
\UnaryInfC{$A \wedge B \vdash A$}
\RightLabel{$\vee_d^1$}
\UnaryInfC{$A \wedge B \vdash A \vee B$}
\end{prooftree}
\qquad
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$A \vdash A$}
\RightLabel{$\vee_d^1$}
\UnaryInfC{$A \vdash A \vee B$}
\RightLabel{$\wedge_g^1$}
\UnaryInfC{$A \wedge B \vdash A \vee B$}
\end{prooftree}
$$

Comme on le verra à la section 4, l'élimination des coupures est un processus de calcul d'où son intérêt pour l'informatique. De ce point de vue calculatoire, le théorème 2 montre la terminaison du processus. Une autre propriété que l'on peut attendre est le *déterminisme*, c'est-à-dire l'unicité du résultat du calcul. Dans le cas du calcul des séquents pour la logique classique, cette propriété est loin d'être vérifiée.

Exemple 4 (Paire critique de Lafont)

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma\vdash\Delta\end{array}}{\Gamma\vdash\Delta,A}\,\mathit{aff}_d \qquad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \Sigma\vdash\Xi\end{array}}{\Sigma,A\vdash\Xi}\,\mathit{aff}_g}{\Gamma,\Sigma\vdash\Delta,\Xi}\,\mathit{cut}$$

$$\swarrow \qquad\qquad \searrow$$

$$\begin{array}{cl}\pi_1 & \\ \vdots & \\ \Gamma\vdash\Delta & \\ \hline\hline \Gamma,\Sigma\vdash\Delta & \mathit{aff}_g \\ \hline\hline \Gamma,\Sigma\vdash\Delta,\Xi & \mathit{aff}_d\end{array} \qquad\qquad \begin{array}{cl}\pi_2 & \\ \vdots & \\ \Sigma\vdash\Xi & \\ \hline\hline \Gamma,\Sigma\vdash\Xi & \mathit{aff}_g \\ \hline\hline \Gamma,\Sigma\vdash\Delta,\Xi & \mathit{aff}_d\end{array}$$

π_1 et π_2 étant quelconques, on ne peut identifier les deux résultats ce qui montre le non déterminisme du calcul.

1.3 Quelques preuves formelles

Pour mieux comprendre ce qu'est une preuve formelle et comment on utilise les règles du calcul des séquents, nous allons détailler quelques exemples simples qui sont par ailleurs intéressants d'un point de vue logique.

Involutivité de la négation

$$\dfrac{\dfrac{\dfrac{}{A\vdash A}\,ax}{A,\neg A\vdash}\,\neg_g}{A\vdash\neg\neg A}\,\neg_d \qquad\qquad \dfrac{\dfrac{\dfrac{}{A\vdash A}\,ax}{\vdash\neg A,A}\,\neg_d}{\neg\neg A\vdash A}\,\neg_g$$

On a démontré que A et $\neg\neg A$ sont équivalents, principe tout à fait usuel en logique classique qui montre que l'on passe simplement par négation d'un côté à l'autre du séquent (de manière réversible) ce qui deviendra faux en logique intuitionniste.

Dualité $\wedge$-$\vee$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A\vdash A}\,ax}{A\wedge B\vdash A}\,\wedge_g^1}{\neg A,A\wedge B\vdash}\,\neg_g \qquad \dfrac{\dfrac{\dfrac{}{B\vdash B}\,ax}{A\wedge B\vdash B}\,\wedge_g^2}{\neg B,A\wedge B\vdash}\,\neg_g}{\neg A\vee\neg B,A\wedge B\vdash}\,\vee_g}{\neg A\vee\neg B\vdash\neg(A\wedge B)}\,\neg_d \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A\vdash A}\,ax}{\vdash\neg A,A}\,\neg_d}{\vdash\neg A\vee\neg B,A}\,\vee_d^1 \qquad \dfrac{\dfrac{\dfrac{}{B\vdash B}\,ax}{\vdash\neg B,B}\,\neg_d}{\vdash\neg A\vee\neg B,B}\,\vee_d^2}{\vdash\neg A\vee\neg B,A\wedge B}\,\wedge_d}{\neg(A\wedge B)\vdash\neg A\vee\neg B}\,\neg_g$$

On retrouve le principe connu : $\neg(A\wedge B)$ et $\neg A\vee\neg B$ sont équivalents, on pourrait de même montrer que $\neg(A\vee B)$ et $\neg A\wedge\neg B$ sont équivalents. On dit que $\wedge$ et $\vee$ sont des connecteurs *duaux*, ce qui se retrouve dans la parfaite similitude entre règle à droite (resp. à gauche) du $\wedge$ et règle à gauche (resp. à droite) du $\vee$.

Définissabilité de $\to$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax}{\neg A, A \vdash}\,\neg_g}{\neg A, A \vdash B}\,\mathit{aff}_d \qquad \dfrac{\dfrac{}{B \vdash B}\,ax}{B, A \vdash B}\,\mathit{aff}_g}{\neg A \vee B, A \vdash B}\,\vee_g}{\neg A \vee B \vdash A \to B}\,\to_d \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax}{\vdash A, \neg A}\,\neg_d}{\vdash A, \neg A \vee B}\,\vee_d^1 \qquad \dfrac{\dfrac{}{B \vdash B}\,ax}{B \vdash \neg A \vee B}\,\vee_d^2}{A \to B \vdash \neg A \vee B, \neg A \vee B}\,\to_g}{A \to B \vdash \neg A \vee B}\,\mathit{contr}_d$$

Le connecteur $\to$ peut donc être défini en logique classique à partir de $\neg$ et de $\vee$ par $A \to B \stackrel{def}{=} \neg A \vee B$. Par conséquent, on peut se passer d'avoir ce connecteur $\to$ avec des règles spécifiques.

1.4 Calcul monolatère

Si l'on oublie le connecteur $\to$, l'existence d'une négation involutive et la symétrie entre règles à gauche et règles à droite aussi bien dans le groupe structurel qu'entre les connecteurs $\wedge$ et $\vee$ permet d'introduire des séquents *monolatères*. L'idée est que les règles à gauche et à droite sont redondantes dans la mesure où la négation (passage d'un côté à l'autre) les échange parfaitement.

Dans ce calcul monolatère, les séquents sont de la forme $\vdash \Gamma$ et la négation n'est plus un connecteur mais est définie par les lois de De Morgan permettant de passer d'un connecteur à son dual : $\neg(A \wedge B) \stackrel{def}{=} \neg A \vee \neg B$, ... elle n'apparaît alors qu'au niveau des formules atomiques. Les règles que l'on obtient sont :

$$\dfrac{}{\vdash A, \neg A}\,ax \qquad \dfrac{\vdash \Gamma, A \quad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}\,cut$$

$$\dfrac{\vdash \Gamma}{\vdash \Gamma, A}\,\mathit{aff} \qquad \dfrac{\vdash \Gamma, A, A}{\vdash \Gamma, A}\,\mathit{contr}$$

$$\dfrac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\,\wedge \qquad \dfrac{\vdash \Gamma, A}{\vdash \Gamma, A \vee B}\,\vee_1 \qquad \dfrac{\vdash \Gamma, B}{\vdash \Gamma, A \vee B}\,\vee_2$$

1.5 Règles multiplicatives et additives

Les règles de déduction que l'on a données à la section 1.2 pour les connecteurs $\wedge$ et $\vee$ sont dites *additives*, il est également possible de donner d'autres règles dites *multiplicatives* qui diffèrent principalement sur la gestion des contextes.

$$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge^m B \vdash \Delta}\,\wedge_g(m) \qquad \dfrac{\Gamma \vdash A, \Delta \quad \Sigma \vdash B, \Xi}{\Gamma, \Sigma \vdash A \wedge^m B, \Delta, \Xi}\,\wedge_d(m)$$

$$\dfrac{\Gamma, A \vdash \Delta \quad \Sigma, B \vdash \Xi}{\Gamma, \Sigma, A \vee^m B \vdash \Delta, \Xi}\,\vee_g(m) \qquad \dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee^m B, \Delta}\,\vee_d(m)$$

Ces règles exprime clairement ce que l'on disait à la section 1.1 sur la signification des séquents : la virgule à gauche est un $\wedge$ et la virgule à droite est un $\vee$.

En logique classique, les connecteurs additifs et multiplicatifs sont équivalents :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax}{A, B \vdash A}\,\mathit{aff}_g}{A \wedge^m B \vdash A}\,\wedge_g(m) \qquad \dfrac{\dfrac{\dfrac{}{B \vdash B}\,ax}{A, B \vdash B}\,\mathit{aff}_g}{A \wedge^m B \vdash B}\,\wedge_g(m)}{A \wedge^m B \vdash A \wedge^a B}\,\wedge_d(a) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax \quad \dfrac{}{B \vdash B}\,ax}{A, B \vdash A \wedge^m B}\,\wedge_d(m)}{A \wedge^a B, B \vdash A \wedge^m B}\,\wedge_g^1(a)}{A \wedge^a B, A \wedge^a B \vdash A \wedge^m B}\,\wedge_g^2(a)}{A \wedge^a B \vdash A \wedge^m B}\,\mathit{contr}_g$$

Ces preuves ne font intervenir que des règles structurelles et même plus précisément toutes les règles structurelles si l'on veut obtenir les équivalences entre les deux $\wedge$ et les deux $\vee$.

2 Logique intuitionniste

Comme on l'a souligné dans l'exemple 4, le calcul des séquents classique n'a pas de propriété de déterminisme. De plus, la logique classique n'est pas un système *constructif* : si on a démontré la formule $A \vee B$ sans coupure, il n'en découle pas directement une preuve de A ou de B.

Exemple 5 (Tiers exclus)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,ax}{\vdash A, \neg A}\,\neg_d}{\vdash A \vee \neg A, \neg A}\,\vee_d^1}{\vdash A \vee \neg A, A \vee \neg A}\,\vee_d^2}{\vdash A \vee \neg A}\,\mathit{contr}_d$$

Même si $A \vee \neg A$ est toujours prouvable savoir si A ou $\neg A$ l'est ne peut absolument pas être déduit de cette démonstration.

Du point de vue de la théorie de la démonstration, la logique intuitionniste s'avère posséder de bien meilleures propriétés que son ancêtre classique, au prix d'une rupture des symétries gauche/droite. On obtient un système constructif muni d'un processus d'élimination des coupures déterministe et qui de plus permet l'introduction d'une meilleure syntaxe et une meilleure interprétation des preuves.

2.1 Calcul des séquents LJ

Les séquents intuitionnistes sont les séquents de la forme $\Gamma \vdash A$ ou $\Gamma \vdash$, pour le reste le calcul est le même que dans le cas classique, cette forme particulière des séquents interdisant la règle de contraction à droite.

Groupe identité

$$\dfrac{}{A \vdash A}\,ax \qquad \dfrac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\,cut$$

Groupe structurel

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\,\mathit{aff}_g \qquad \frac{\Gamma \vdash}{\Gamma \vdash A}\,\mathit{aff}_d \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\,\mathit{contr}_g$$

Groupe logique

$$\frac{\Gamma \vdash A}{\Gamma, \neg A \vdash}\,\neg_g \qquad \frac{\Gamma, A \vdash}{\Gamma \vdash \neg A}\,\neg_d$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_g^1 \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_g^2 \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_d$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\,\vee_g \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_d^1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_d^2$$

$$\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \to B \vdash C}\,\to_g \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to_d$$

La rupture des symétries du calcul et la restriction des règles structurelles permettent d'introduire un processus déterministe naturel d'élimination des coupures (contrairement au cas classique comme le montre l'exemple 4) : pour éliminer la coupure, on remplace les utilisations de l'hypothèse A dans la preuve de $\Delta, A \vdash C$ par des copies de la preuve de $\Gamma \vdash A$.

Le prix à payer est la perte de la négation involutive à cause de la forte dissymétrie gauche/droite ainsi que la non interdéfinissabilité des connecteurs : $\to$ doit avoir ses propres règles, l'équivalence avec $\neg A \vee B$ n'est plus démontrable (même si $A \vee \neg A$ n'est plus prouvable pour toute formule A, $A \to A$ doit bien entendu le rester). Parmi les preuves de la section 1.3, seules celles de $A \vdash \neg\neg A$ et de $\neg A \vee \neg B \vdash \neg(A \wedge B)$ restent valables, les autres séquents n'étant pas prouvables en logique intuitionniste.

Corollaire 2.3 (Cas intuitionniste)
Le calcul des séquents intuitionniste admet l'élimination des coupures.

Le processus de calcul qui en découle est déterministe modulo des permutations de règles, ce qui est beaucoup plus fort que dans le cas classique où le non déterminisme est intrinsèque.

Corollaire 2.4 (Propriété de la disjonction)
De toute preuve de $A \vee B$ en calcul des séquents intuitionniste on peut extraire une preuve de A ou une preuve de B.

2.2 Déduction naturelle

Comme on l'a vu l'élimination des coupures en calcul des séquents nécessite de nombreuses étapes commutatives dont certaines n'ont que peu d'intérêt. Il est naturel de s'intéresser à d'autres syntaxes correspondant à la même logique c'est-à-dire permettant de prouver les mêmes formules mais possédant une élimination des coupures plus claire.

Pour la logique intuitionniste, la *déduction naturelle* a été introduite par G. Gentzen et particulièrement étudiée par D. Prawitz. Elle s'obtient à partir du calcul des séquents en ne gardant que les règles à droite des connecteurs (appelées *règles d'introduction*) et en ajoutant les règles suivantes, appelées *règles d'élimination.*

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

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge_e^1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge_e^2$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\vee_e$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\,\rightarrow_e$$

Remarque : La règle d'élimination de $\rightarrow$ est souvent appelée *modus ponens* et constitue l'unique règle du système de D. Hilbert.

Ces règles d'élimination ont l'avantage d'être plus proches des raisonnements mathématiques usuels que les règles à gauche du calcul des séquents qu'elles remplacent.

Bien qu'il n'y ait pas dans ce système de règle de coupure explicite, on peut tout de même définir une telle notion comme étant la succession d'un règle d'introduction et d'une règle d'élimination portant sur le même connecteur de la même formule.

Théorème 3 (Normalisation forte)
La déduction naturelle admet l'élimination des coupures (et ce indépendemment de l'ordre dans lequel on les élimine).

Ce processus est cette fois totalement déterministe, ce qui montre que le petit problème de commutation de règles qui persiste dans le système LJ est purement superficiel. De plus en l'absence du connecteur $\vee$ aucune commutation de règles n'est nécessaire, on ne fait que des étapes logiques.

En terme de prouvabilité, on a vu que la logique intuitionniste est strictement plus faible que la logique classique, cependant il suffit de rajouter un axiome $\overline{\vdash A \vee \neg A}$ (tiers exclus) ou $\overline{\vdash \neg\neg A \rightarrow A}$ (raisonnement par l'absurde) pour retrouver la prouvabilité classique, mais on perd alors les propriétés de la logique intuitionniste.

3 λ-calcul

Le λ-calcul a été introduit vers 1930 par A. Church, il constitue un langage de programmation théorique défini par seulement trois constructions. Il est cependant suffisamment puissant pour coder n'importe quel langage de programmation (en particulier toutes les fonctions récursives).

Ce langage reste fondamentalement un outil théorique à cause de la difficulté à écrire de gros programmes mais il est à la base des langages de programmation dits *fonctionnels* comme LISP, ML, ...

3.1 λ-calcul pur

Définition 1 (λ-terme)
Étant donné un ensemble dénombrable de *variables* $\mathcal{V}$, l'ensemble des λ-*termes* est le plus petit ensemble tel que :

- **variable :** Si $x \in \mathcal{V}$, x est un λ-terme.
- **abstraction :** Si u est un λ-terme et $x \in \mathcal{V}$, $\lambda x.u$ est un λ-terme.
- **application :** Si u et v sont des λ-termes, $(u)v$ est un λ-terme.

La signification de la construction d'abstraction est la suivante : si u est une expression faisant intervenir x, $\lambda x.u$ est la fonction qui à x associe la valeur de u pour cet x ce que l'on note habituellement en mathématiques $x \mapsto u$. Le nom donné à x n'a donc plus d'importance, on parle de *variable liée* et λ est appelé un *lieur*.

Le cas de l'application correspond à l'application de la fonction u à l'argument v (que l'on note plutôt $u(v)$ en mathématiques).

Définition 2 (β-réduction)
L'évaluation des λ-termes est définie par un règle de réécriture appelée β-*réduction* :

$$(\lambda x.u)v \rightarrow u[^v/_x]$$

où $u[^v/_x]$ est le λ-terme u dans lequel chaque occurrence de la variable x a été remplacée par v.

Lors de la substitution $u[^v/_x]$, il est nécessaire de faire attention aux problèmes de *capture de variable* c'est pourquoi il est sous-entendu que l'on effectue la substitution avec renommage des variables liées. Une définition précise nécessiterait de rentrer dans des détails fastidieux, on se référera au livre de J.-L. Krivine [4] pour une présentation tout à fait rigoureuse.

La β-réduction correspond au calcul à effectuer lorsque l'on applique une fonction à une valeur ou un programme à un argument, ainsi le passage de la valeur 3 à la fonction $x \mapsto x + x$ donnera $3 + 3$ et enfin par évaluation de l'addition : 6.

Exemple 6 (Quelques λ-termes)
Voici différents exemples de λ-termes simples :

$id = \lambda x.x$	fonction identité : elle rend son argument
$p1 = \lambda x.\lambda y.x$	première projection : prend deux arguments et rend le premier
$p2 = \lambda x.\lambda y.y$	deuxième projection : prend deux arguments et rend le deuxième
$ap = \lambda f.\lambda x.(f)x$	application : prend deux arguments et applique le premier (fonction) au deuxième (valeur)

On peut effectuer un calcul avec ces termes :

$$((ap)((p2)p1)id)z \rightarrow (\lambda x.(((p2)p1)id)x)z \rightarrow (((p2)p1)id)z \rightarrow ((\lambda y.y)id)z \rightarrow (id)z \rightarrow z$$

Pour assurer un peu de lisibilité et de concision on a donné des noms à certains sous-termes mais le véritable λ-terme que l'on a réduit est $((\lambda f.\lambda x.(f)x)((\lambda x.\lambda y.y)\lambda x.\lambda y.x)\lambda x.x)z$. La pauvreté du langage fait que les programmes deviennent vite compliqués et incompréhensibles, c'est pourquoi le λ-calcul n'est pas utilisable comme vrai langage de programmation.

Théorème 4 (Church-Rosser)
La β-réduction est confluente.

Autrement dit, si t se réduit en u et en v par des suites de β-réductions, alors il existe un terme w tel que u et v peuvent tous les deux se β-réduire en w.

Définition 3 (Forme normale et terme normalisable) – Un λ-terme est en *forme normale* si on ne peut pas lui appliquer de β-réduction.
- Un λ-terme est dit *normalisable* s'il existe une suite de β-réductions qui le transforme en une forme normale.
- Un λ-terme t est dit *fortement normalisable* si toute suite de β-réductions de t est finie.

Ainsi une forme normale est un terme sur lequel on ne peux plus effectuer de calcul c'est-à-dire une valeur, un résultat final.

Corollaire 4.1 (Unicité de la forme normale)
Il existe au plus une forme normale en laquelle un λ-terme peut se réduire.

Cette propriété est le déterminisme du calcul : un programme ne peut donner qu'un seul résultat, quel que soit l'ordre dans lequel les calculs ont été effectués à l'intérieur du programme.

Exemple 7
Il existe des λ-termes non normalisables, c'est-à-dire des programmes dont l'exécution ne termine pas :

soit $\delta = \lambda x.(x)x$, on a $(\delta)\delta \to (x)x[^{\delta}/_{x}] = (\delta)\delta$ ...

Il existe également des λ-termes normalisables non fortement normalisables : $(\lambda x.\lambda y.y)(\delta)\delta \to \lambda y.y$ mais également $(\lambda x.\lambda y.y)(\delta)\delta \to (\lambda x.\lambda y.y)(\delta)\delta$ ... en réduisant dans $(\delta)\delta$.

3.2 Systèmes de λ-calcul typé

Un système de types consiste à associer aux λ-termes des formules d'un langage logique par des règles d'inférence. Ce typage permet de caractériser des propriétés des programmes par exemple, dans un système de types suffisamment riche, on pourra montrer qu'un programme effectue bien le calcul que l'on souhaite et ne produit pas d'erreurs.

Système des types simples Ce premier système de types introduit par H. Curry établit une hiérarchie entre les termes : termes de base, fonctions, fonctionnelles, ...

On considère les types formés uniquement à l'aide de $\to$ à partir de variables.

Le séquent (ou jugement) $x_1 : A_1, ..., x_n : A_n \vdash u : B$ où x_1, ..., x_n sont des variables et u est un λ-terme signifie que dans un environnement dans lequel x_1 a le type A_1, ..., x_n a le type A_n, u a lui le type B.

$$\frac{}{x_1 : A_1, ..., x_n : A_n \vdash x_i : A_i}\,var \qquad \frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x.u : A \to B}\,abs \qquad \frac{\Gamma \vdash u : A \to B \quad \Gamma \vdash v : A}{\Gamma \vdash (u)v : B}\,app$$

Un terme de type $A \to B$, par exemple, correspond à une fonction qui prend comme argument un programmes de type A et rend un programme de type B.

Alors que le λ-calcul pur n'assure aucune garantie de terminaison des programmes un certain nombre de systèmes de types assurent cette propriété.

Théorème 5 (Normalisation forte)
Si t est typable dans le système des types simples alors t est fortement normalisable.

Exemple 8
Si l'on reprend l'exemple 7, on constate que δ n'est pas typable dans ce système, sinon x aurait un type A et pour typer $(x)x$ il faudrait que x soit également de type $A \to B$. Or $A = A \to B$ est impossible dans le langage des types simples.

Systèmes F et AF_2 En ajoutant le quantificateur universel $\forall$ du second ordre (système F) ou du premier et du second ordre (système AF_2) on obtient un ensemble beaucoup plus grand de λ-termes typés qui constitue un langage tout à fait expressif (polymorphisme), avec lequel on peut définir tous les types de données usuels : booléens, entiers, paires, listes, arbres,...

Théorème 6 (Normalisation forte)
Si t est typable dans le système F (ou, ce qui est équivalent, dans le système AF_2) alors t est fortement normalisable.

4 Isomorphisme de Curry-Howard

L'*isomorphisme* (ou *correspondance*) de Curry-Howard fait apparaître les liens très forts qui existent entre théorie de la démonstration et programmation.

Commençons tout d'abord par des constatations élémentaires :

1. les types utilisés dans le système des types simples du λ-calcul sont des formules de la logique intuitionniste n'utilisant que le connecteur $\to$ (c'est-à-dire la logique intuitionniste minimale),
2. si l'on oublie les λ-termes, les règles du système des types simples sont les règles pour le connecteur $\to$ et la règle d'axiome de la déduction naturelle,
3. étant donnée une preuve en déduction naturelle n'utilisant que le connecteur $\to$, il existe une unique manière (à renommage des variables près) d'associer un λ-terme à chaque formule de la preuve pour obtenir une dérivation de typage dans le système des types simples.

On a correspondance exacte entre preuves de la logique intuitionniste minimale en déduction naturelle et λ-termes simplement typés, mais cette correspondance n'est pas seulement “statique”, l'élimination des coupures correspond exactement à la β-réduction du λ-calcul. Dans ce cas particulier, les preuves sont donc des programmes et éliminer les coupures revient à évaluer ces programmes en un résultat. On peut donner le tableau de correspondance suivant :

Logique	Informatique
preuve	programme (λ-terme)
formule	spécification (type)
élimination des coupures	évaluation (β-réduction)
preuve sans coupures	résultat (forme normale)

Cette correspondance ne s'arrête pas à la logique intuitionniste minimale et de très nombreux travaux ont été faits pour l'étendre. Ainsi, à l'ajout des quantificateurs $\forall$ du premier et du second ordre à la logique sont associés les systèmes F et AF_2. Vers 1990 un nouveau pas a été franchi avec l'extension à la logique classique correspondant au principe des continuations en programmation.

logique intuitionniste minimale	$\longleftrightarrow$	λ-calcul simplement typé
logique intuitionniste propositionnelle du 2^{nd} ordre	$\longleftrightarrow$	système F
logique intuitionniste du 2^{nd} ordre avec raisonnement équationnel	$\longleftrightarrow$	système AF_2
logique classique minimale	$\longleftrightarrow$	$\lambda\mu$-calcul

5 Logique linéaire

On peut continuer dans la direction prise par la logique intuitionniste concernant la restriction des règles structurelles, en adoptant la position radicale qui consiste à interdire toutes les règles structurelles. Afin de conserver un pouvoir expressif au moins équivalent à celui de la logique intuitionniste, on réintroduit les règles structurelles sous formes de règles logiques. On ajoute pour cela des connecteurs unaires ? et ! appelés *modalités*, avec la contrainte que seules les formules marquées d'un ? ont le droit de subir des contractions et des affaiblissements.

La parfaite symétrie gauche/droite de cette contrainte permet de retrouver en logique linéaire des propriétés de la logique classique perdues en logique intuitionniste comme la négation involutive $(.)^{\perp}$, les connecteurs interdéfinissables, ...

Cette nouvelle logique, la logique linéaire introduite en 1986 par J.-Y. Girard [1], a permis d'affiner considérablement l'étude des autres systèmes et cumule de nombreuses bonnes propriétés.

5.1 Calcul des séquents LL

L'absence de règles structurelles a pour conséquence que les connecteurs multiplicatifs et additifs tels qu'on les a vus à la section 1.5 ne sont plus équivalents, on introduit donc deux connecteurs de conjonction : le *tenseur* $\otimes$ (multiplicatif) et le *avec* & (additif) et deux connecteurs de disjonction : le *par* $\parr$ (multiplicatif) et le *plus* $\oplus$ (additif).

La négation $(.)^{\perp}$ étant involutive, on se restreint à un calcul des séquents monolatère où la négation est définie par

$$\begin{array}{ccccccc}
(A \otimes B)^{\perp} & \stackrel{def}{=} & A^{\perp} \parr B^{\perp} & \qquad & (A \parr B)^{\perp} & \stackrel{def}{=} & A^{\perp} \otimes B^{\perp} \\
(A \oplus B)^{\perp} & \stackrel{def}{=} & A^{\perp} \,\&\, B^{\perp} & & (A \,\&\, B)^{\perp} & \stackrel{def}{=} & A^{\perp} \oplus B^{\perp} \\
(!A)^{\perp} & \stackrel{def}{=} & ?A^{\perp} & & (?A)^{\perp} & \stackrel{def}{=} & !A^{\perp}
\end{array}$$

Groupe identité

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

Groupe multiplicatif

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

Groupe additif

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,\& \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,\oplus_1 \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,\oplus_2$$

Groupe exponentiel

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\,?d \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\,?a \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\,?c \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,!$$

On peut définir le connecteur d'implication linéaire par $A \multimap B \stackrel{def}{=} A^{\perp} \parr B$.

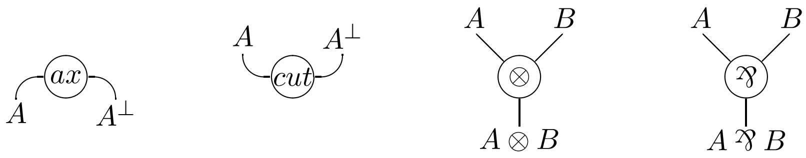
Théorème 7 (Normalisation forte)
Le calcul des séquents LL *admet l'élimination des coupures et ce indépendamment de l'ordre d'élimination.*

On retrouve des propriétés du calcul intuitionniste : le $\oplus$ est une disjonction constructive, l'élimination des coupures est déterministe (à des permutations de règles près sur la forme normale), ...

5.2 Réseaux de preuve

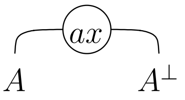
Tout comme, la logique intuitionniste possède une syntaxe particulièrement efficace : la déduction naturelle, la logique linéaire a permis l'introduction des *réseaux de preuve* corrigeant une nouvelle fois les problèmes de commutation de règles du calcul des séquents.

Les réseaux de preuve sont des graphes dont les sommets correspondent aux règles du calcul des séquents et dont les arêtes sont étiquetées par des formules. Nous présentons ici le cas particulièrement simple des réseaux de preuve pour la partie multiplicative (sans additifs ni exponentiels). On dispose des quatre nœuds suivants (un par règle) :

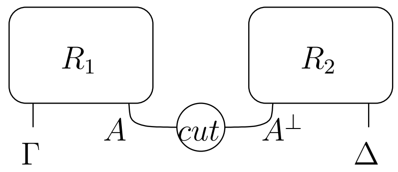


La traduction du calcul des séquents en réseaux se fait par induction sur la structure de la preuve en associant à chaque preuve π_i un réseau R_i de mêmes conclusions.

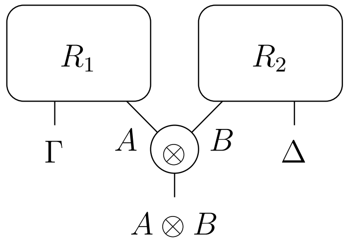
$$\dfrac{}{\vdash A, A^\perp}\,ax \quad \longrightarrow$$



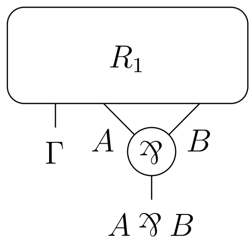
$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Gamma, A\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Delta, A^\perp\end{array}}{\vdash \Gamma, \Delta}\,cut \quad \longrightarrow$$



$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Gamma, A\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \vdash \Delta, B\end{array}}{\vdash \Gamma, \Delta, A \otimes B}\,\otimes \quad \longrightarrow$$

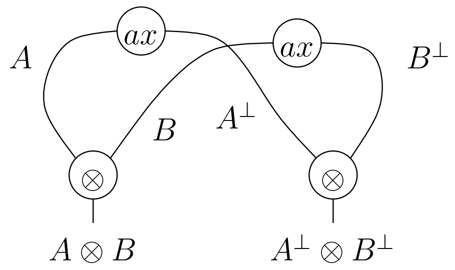


$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \vdash \Gamma, A, B\end{array}}{\vdash \Gamma, A \parr B}\,\parr \quad \longrightarrow$$



Cependant n'importe quel graphe construit avec ces nœuds n'est pas la traduction d'une preuve en calcul des séquents.

Exemple 9 (Réseau incorrect)



Si ce réseau correspondait à une preuve, il y aurait une dernière règle or aucun des deux nœuds $\otimes$ n'a pu être fait en dernier. En effet, si c'était le cas en le supprimant on obtiendrait deux réseaux correspondant aux deux prémisses de la règle du $\otimes$ or quel que soit celui des deux que l'on supprime on obtient un seul graphe.

On est alors amené à introduire la notion de *critère de correction* qui permet de distinguer les réseaux qui correspondent à des preuves (réseaux *séquentialisables*) et les autres.

Définition 4 (Graphe de correction)
Un graphe de correction d'un réseau multiplicatif est un sous-graphe obtenu en ne conservant qu'une des deux arêtes situées au-dessus de chaque nœud $\parr$.

Théorème 8 (Séquentialisation)
Un réseau est séquentialisable si et seulement si tous ses graphes de correction sont acycliques et connexes.

Le réseau de l'exemple 9 n'a par exemple qu'un graphe de correction et celui-ci est cyclique.

Théorème 9 (Élimination des coupures)
Les réseaux de preuve vérifient l'élimination forte des coupures et celle-ci est confluente.

La syntaxe des réseaux s'étend en ajoutant une notion de boîtes au cas des connecteurs exponentiels, le cas des connecteurs additifs est lui plus délicat et une solution a été présentée dans [2]. Cependant, celle-ci est nettement plus complexe que pour les autres connecteurs.

6 Logique classique constructive

Les apports de la logique linéaire ont permis d'affiner l'étude de la logique classique : V. Danos, J.-B. Joinet et H. Schellinx ont ainsi introduit le système $\mathsf{LK}^{\mathsf{tq}}$ et par ailleurs J.-Y. Girard a défini le calcul LC qui constituent tous les deux des systèmes de preuve pour la logique classique avec une élimination déterministe des coupures basés sur la logique linéaire.

L'idée de LC est d'introduire une notion de polarité des formules de la logique classique définie de la manière suivante :

A	B	$A \wedge B$	$A \vee B$	$\neg A$
$+$	$+$	$+$	$+$	$-$
$-$	$+$	$+$	$-$	$+$
$+$	$-$	$+$	$-$	
$-$	$-$	$-$	$-$	

en supposant que les atomes sont positifs.

Remarque : Le tableau ci-dessus ne constitue pas une table de vérité malgré une forte ressemblance. En effet, les deux constantes V et F ont toutes les deux la polarité $+$.

Pour simplifier la syntaxe, on utilise un calcul des séquents monolatère. Le non déterminisme de l'élimination des coupures est résolu par le fait qu'une coupure a nécessairement lieu entre une formule positive et une formule négative (puisque la négation inverse la polarité) ce qui brise la symétrie.

6.1 Calcul des séquents LC

Les séquents utilisés sont de la forme $\vdash \Gamma; \Pi$ où Γ est un multi-ensemble de formules positives et négatives et Π (le *bénitier*) est soit vide soit constitué d'une unique formule positive.

$$\frac{}{\vdash \neg P; P} \qquad \frac{\vdash \Gamma; P \quad \vdash \neg P, \Delta; \Pi}{\vdash \Gamma, \Delta; \Pi} \qquad \frac{\vdash \Gamma, N; \quad \vdash \neg N, \Delta; \Pi}{\vdash \Gamma, \Delta; \Pi}$$

$$\frac{\vdash \Gamma; P}{\vdash \Gamma, P;} \qquad \frac{\vdash \Gamma; \Pi}{\vdash \Gamma, N; \Pi} \qquad \frac{\vdash \Gamma, N, N; \Pi}{\vdash \Gamma, N; \Pi} \qquad \frac{\vdash \Gamma; \Pi}{\vdash \Gamma, P; \Pi} \qquad \frac{\vdash \Gamma, P, P; \Pi}{\vdash \Gamma, P; \Pi}$$

$$\frac{\vdash \Gamma; P \quad \vdash \Delta; Q}{\vdash \Gamma, \Delta; P \wedge Q} \qquad \frac{\vdash \Gamma; P \quad \vdash \Delta, N;}{\vdash \Gamma, \Delta; P \wedge N} \qquad \frac{\vdash \Gamma, M; \quad \vdash \Delta; Q}{\vdash \Gamma, \Delta; M \wedge Q}$$

$$\frac{\vdash \Gamma, M; \Pi \quad \vdash \Gamma, N; \Pi}{\vdash \Gamma, M \wedge N; \Pi}$$

$$\frac{\vdash \Gamma, M, N; \Pi}{\vdash \Gamma, M \vee N; \Pi} \qquad \frac{\vdash \Gamma, N, P; \Pi}{\vdash \Gamma, N \vee P; \Pi} \qquad \frac{\vdash \Gamma, P, N; \Pi}{\vdash \Gamma, P \vee N; \Pi}$$

$$\frac{\vdash \Gamma; P}{\vdash \Gamma; P \vee Q} \qquad \frac{\vdash \Gamma; Q}{\vdash \Gamma; P \vee Q}$$

Théorème 10 (Complétude et correction)
Le séquent $\vdash \Gamma;$ est prouvable dans LC *si et seulement si $\vdash \Gamma$ est prouvable dans* LK*.*

La prouvabilité dans LC pour les séquents à bénitier vide correspond donc à celle de LK ce qui justifie qu'on a à faire à un système classique.

Théorème 11 (Élimination des coupures)
L'élimination des coupures dans LC *termine et est déterministe (à permutations de règles près).*

6.2 Réseaux de preuve polarisés

La syntaxe du système LC a été donnée par J.-Y. Girard par un calcul des séquents ce qui pose les problèmes que nous avons déjà rencontrés quant aux commutations de règles lors de l'élimination des coupures pour atteindre les étapes intéressantes.

La sémantique définie pour LC montre comment ce système peut être vu comme un fragment de LL où seules certaines formules dites *polarisées* sont autorisées. Pour ce fragment de la logique linéaire, on obtient une notion de réseaux de preuve particulièrement simple ce qui permet donc d'avoir une syntaxe efficace pour LC.

De plus, le fragment polarisé de LL s'avère avoir de bonnes propriétés et contenir la plupart des systèmes classiques avec élimination des coupures déterministe existants : LC ainsi que nous venons de le dire, mais également $\mathsf{LK}^{\eta}_{\mathrm{pol}}$ ou le $\lambda\mu$-calcul.

Références

[1] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[2] Jean-Yves Girard. Proof-nets : the parallel syntax for proof-theory. In Ursini and Agliano, editors, *Logic and Algebra*, New York, 1996. Marcel Dekker.

[3] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and types*. Number 7 in Cambridge tracts in theoretical computer science. Cambridge University Press, 1989.

[4] Jean-Louis Krivine. *Lambda-Calcul : Types et Modèles*. Études et Recherches en Informatique. Masson, 1990.