Introduction

Table des matières

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

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

On se réfère également au calcul **LJ** défini dans cette même section, mais on travaille ici sur une représentation légèrement différente. On s'intéresse en effet à des *multiensembles*, c'est-à-dire des collections où le nombre d'occurrences est pris en compte, mais non l'ordre des éléments. Un séquent de Γ consiste alors en un multiensemble (au lieu d'une liste) de formules Γ et une formule D, toujours noté $\Gamma \vdash D$. On utilise la notation X, Y pour désigner la réunion de deux multiensembles X et Y, au lieu de la concaténation de deux listes, X ou Y pouvant encore une fois être une formule représentant ici un multiensemble à un élément.

Malgré le changement de structure et grâce à la réutilisation de la notation X, Y , les règles de **LJ** sont toujours données par la figure **??** page **??**, à l'exception de la règles d'échange $\dfrac{\Gamma_1, A, B, \Gamma_2 \Rightarrow D}{\Gamma_1, B, A, \Gamma_2 \Rightarrow D}\ (permut.\ L)$ qui peut être oubliée puisqu'un multiensemble ne tient pas compte de l'ordre.

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

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

1.1 Algorithme de recherche de preuve

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

Cet algorithme est correct par construction et d'après la définition de la prouvabilité d'un séquent. En revanche, il y a des causes possibles de non terminaison, qui se regroupent en deux catégories : "largeur" infinie, "profondeur" infinie. Pour éviter une "largeur" infinie, il faut que pour un séquent donné, le nombre d'instances dont il est conclusion soit fini. En ce qui concerne le problème de "profondeur" dû à la récursivité, on peut souvent munir les séquents d'un ordre bien fondé, de sorte que pour toute instance de règle, les prémisses sont toutes strictement inférieures à la conclusion.

Remarque : la règle de coupure. La règle de coupure, par exemple pour **LJ** : $\dfrac{\Gamma_1 \Rightarrow A \qquad A, \Gamma_2 \Rightarrow D}{\Gamma_1, \Gamma_2 \Rightarrow D}\ (cut)$, rend l'algorithme proposé inutilisable parce qu'il ne termine jamais. En effet, on explore indéfiniment en "largeur", car le nombre d'instances dont un séquent donné est conclusion est infini, A pouvant être n'importe quelle formule. Heureusement, cette règle est souvent non nécessaire. De nombreux calculs la formulent car c'est une

bonne chose que l'implication représentée par un séquent soit transitive, mais s'en passent ensuite grâce à un *théorème d'élimination de la coupure*, souvent difficile à établir. Voir la section **??** du mémoire pour plus de détails. Quoi qu'il en soit, on ne s'intéresse désormais qu'à des calculs dans lesquels cette règle ne figure pas.

1.2 Propriétés intéressantes des calculs de séquents

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

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

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

Inversibilité de certaines règles ou prémisses. Dans l'algorithme proposé, il peut être assez long de montrer qu'un séquent n'est pas prouvable, puisqu'on essaie toutes les instances dont il est la conclusion. La notion d'inversibilité permet de terminer beaucoup plus rapidement dans certains cas. Une prémisse $prem_k$ d'une règle $\dfrac{prem_1 \quad \dots \quad prem_p}{concl}\,(\mathcal{R})$ (aussi appelée *k-ième prémisse de* $\mathcal{R}$) est ***inversible*** si on a : si $prem_k$ est non prouvable, alors *concl* est non prouvable. Une règle est ***inversible*** si toutes ses prémisses sont inversibles. Ainsi, si au cours de la recherche de preuve, on obtient qu'une prémisse inversible est non prouvable, on peut directement conclure que la conclusion ne l'est pas non plus, sans avoir besoin d'essayer d'autre règle.

Localité des règles. L'algorithme nécessite de savoir déterminer, pour un séquent donné, toutes les instances dont il est conclusion, et en particulier calculer les prémisses de ces instances. Pour une instance, la *formule principale* est la formule de la conclusion qui est remplacée dans les prémisses par d'autres formules (règles logiques), ou dupliquée ou supprimée (règles structurelles). Dans le cas des règles logiques, la formule principale doit avoir une

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

1.3 Deux exemples de calculs de séquents pour la logique intuitionniste

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

LJT. Le calcul **LJT** est introduit par R. Dyckhoff dans [1] pour pallier le problème de cycles de **LJ**. Il n'y a pas de règle de contraction, et la règle $\to L$ est remplacée par quatre règles selon la structure de A dans la formule principale $A \to B$: par exemple $\dfrac{B, A, \Gamma \;\Rightarrow\; D}{A \to B, A, \Gamma \;\Rightarrow\; D}\,(\to L_1)$ où A doit être réduite à une variable, ou encore $\dfrac{A_1 \to (A_2 \to B)\,,\; \Gamma \;\Rightarrow\; D}{(A_1 \wedge A_2) \to B\,,\; \Gamma \;\Rightarrow\; D}\,(\to L_2)$. On n'a pas la propriété de la sous-formule, mais on peut quand même déterminer toutes les formules susceptibles d'apparaître lorsqu'on essaie de prouver un séquent donné. Dyckhoff

montre que l'algorithme termine en choisissant bien un bon ordre sur les formules puis sur les séquents. Toutes les règles sont inversibles et locales sauf une des règles qui remplacent $\to L$, qui comporte deux prémisses dont seule la deuxième est inversible.

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

Le calcul de 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 [2]. Il présente des priopriétés 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 1. *Un* **séquent** *de* **LSJ** *est la donnée de trois multiensembles* Θ, Γ *et* Δ *de formules ; on écrit* $\Theta\,;\,\Gamma \Rightarrow \Delta$.

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

Proposition 2. *Soit A une formule, elle est valide en logique intuitionniste si et seulement si le séquent* $\emptyset\,;\,\emptyset \Rightarrow A$ *est prouvable dans* **LSJ**.

Les ***règles*** du calcul **LSJ** sont données dans la figure 1. Toutes les règles sont des *axiomes* ou des *règles logiques*. Il n'y a pas de *règle structurelle* ni de règle de *coupure*.

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

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

2.2 Séquents et règles de LSJℓ

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

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

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

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

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

FIGURE 1 – Les règles du calcul **LSJ**

Définition 3. *Un* **séquent** *de* **LSJℓ** *est la donnée de deux multiensembles Γ et Δ de couples* " entier : formule "*, et d'un entier naturel n, tels que tous les entiers présents dans Γ sont $\leq n+1$ et tous ceux présents dans Δ sont $\leq n$; on écrit $\Gamma \Rightarrow_n \Delta$.*

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

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

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

2.3 Équivalence entre LSJℓ et LSJ

On note $\mathfrak{S}$ l'ensemble des séquents de **LSJ**, et $\mathfrak{S}'$ l'ensemble des séquents de **LSJℓ**. Soit $\sigma \in \mathfrak{S}$, on note $\vdash \sigma$ si σ est prouvable dans **LSJ** ; soit $\sigma' \in \mathfrak{S}'$, on note $\vdash' \sigma'$ si σ' est prouvable

dans **LSJℓ**. Montrons que pour tous $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$, on a $\vdash \sigma$ si et seulement si $\vdash' \sigma'$, où $\mathbf{\Phi}$ est la surjection de $\mathfrak{S}'$ sur $\mathfrak{S}$ définie précédemment.

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

Lemme 4. *Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$ et soit $\mathcal{R}$ une règle de* **LSJ**.

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

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

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

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

$\to$L : 1) Si $\dfrac{\sigma_1 \quad \sigma_2 \quad \sigma_3}{\sigma}\,(\to L)$ alors il existe A, B et $\widetilde{\Gamma}$ tels que $\Gamma = A \to B, \widetilde{\Gamma}$ et $\sigma_1 = \Theta\,; B, \widetilde{\Gamma} \Rightarrow \Delta$ et $\sigma_2 = B, \Theta\,; \widetilde{\Gamma} \Rightarrow A, \Delta$ et $\sigma_3 = B\,; \Theta, \widetilde{\Gamma} \Rightarrow A$; et il existe $i \leq n$ tel que $i : A \to B \in \Gamma'$. On pose $\widetilde{\Gamma}' = \Gamma' - i : A \to B$ (on retire une seule occurrence de $i : A \to B$ de Γ') et $\sigma'_1 = i : B, \widetilde{\Gamma}' \Rightarrow_n \Delta'$ et $\sigma'_2 = n+1 : B, \widetilde{\Gamma}' \Rightarrow_n n : A, \Delta'$ et $\sigma'_3 = n+2 : B, \widetilde{\Gamma}' \Rightarrow_{n+1} n+1 : A, \Delta'$ et on vérifie qu'on a bien $\sigma_1 = \mathbf{\Phi}(\sigma'_1)$ et $\sigma_2 = \mathbf{\Phi}(\sigma'_2)$ et $\sigma_3 = \mathbf{\Phi}(\sigma'_3)$ (en remarquant que $\widetilde{\Gamma} = \mathsf{forget}(\widetilde{\Gamma}'_{\leq n})$), et aussi $\dfrac{\sigma'_1 \quad \sigma'_2 \quad \sigma'_3}{\sigma'}\,(\to L')$).
2) Si $\dfrac{\sigma'_1 \quad \sigma'_2 \quad \sigma'_3}{\sigma'}\,(\to L')$ alors il existe i, A, B et $\widetilde{\Gamma}'$ tels que $\Gamma' = i : A \to B, \widetilde{\Gamma}'$ et σ'_1, σ'_2 et σ'_3 ont la forme donnée ci-dessus ; on pose $\widetilde{\Gamma} = \Gamma - A \to B$ (on retire une seule occurrence de $A \to B$ de Γ), alors les images σ_1, σ_2 et σ_3 par $\mathbf{\Phi}$ de σ'_1, σ'_2 et σ'_3 respectivement s'écrivent comme ci-dessus et donc $\dfrac{\sigma_1 \quad \sigma_2 \quad \sigma_3}{\sigma}\,(\to L)$. □

Théorème 5. *Soit $\sigma \in \mathfrak{S}$ et $\sigma' \in \mathfrak{S}'$ tels que $\sigma = \mathbf{\Phi}(\sigma')$, alors $\vdash \sigma$ si et seulement si $\vdash' \sigma'$.*

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

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

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

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

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

3 Éléments d'implémentation

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

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

Ce qui est précisé dans [2], mais pas en 1.1 puisque c'est propre au calcul **LSJ**, est l'ordre dans lequel on s'intéresse aux différentes instances dont un séquent donné est conclusion. On privilégie naturellement celles qui offrent une possibilité de terminer rapidement la recherche de preuve, ce qui induit une priorité en fonction de la règle associée : d'abord les axiomes, puis les règles inversibles à une seule prémisse $\wedge L$ et $\vee R$, puis celles à deux prémisses $\wedge R$ et $\vee L$, et enfin les règles non inversibles $\to L$ et $\to R$.

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

Pour cela, on associe à chaque formule de Γ et de Δ une **priorité** selon la figure 3. Plus l'entier est petit, plus la formule est prioritaire. On choisit alors la formule la plus prioritaire parmi les formules "accessibles", c'est-à-dire les formules de Γ auxquelles est associé un indice inférieur à l'indice n du séquent, ou celles de Δ avec un indice égal à n ; ce sont exactement les formules qui auraient figuré dans Γ ou Δ d'un séquent $\Theta\,;\Gamma \Rightarrow \Delta$ de **LSJ**. Ce choix de formule principale n'intervient qu'après avoir testé les axiomes. Si la formule la plus prioritaire est de priorité 6, il n'y a aucune formule principale possible, donc le séquent est non prouvable. Le seul cas où on peut avoir plusieurs instances à tester pour un séquent donné est celui où la formule la plus prioritaire est de priorité 5. Dans ce cas, on teste successivement les formules "accessibles" de forme $A \to B$ 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".

3.2 Indexation

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

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

FIGURE 3 – Priorité selon la forme de la formule et son appartenance à Γ ("côté L") ou Δ ("côté R")

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

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

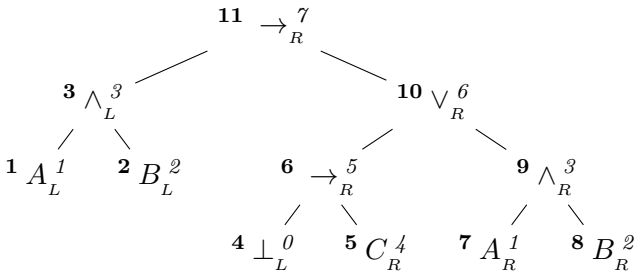

FIGURE 4 – Indexation de la formule $(a \wedge b) \wedge (\neg c \vee (a \wedge b))$. "sf" comme "sous-formule" est le numéro associé à une formule (en gras dans l'arbre), "description" ce qui est stocké dans le tableau. La classe d'une formule (en italique) est expliquée dans la prochaine sous-section.

3.3 Classes d'égalité structurelle pour l'axiome id

L'axiome $\dfrac{}{i : A, \Gamma \Rightarrow_n n : A, \Delta}\,(id)$ $(i \leq n)$ affirme qu'un séquent est prouvable si une même formule A apparaît, de façon "accessible" (condition sur les indices), de part et d'autre du séquent. "même formule" est à comprendre au sens de l'égalité structurelle, c'est-à-dire l'égalité des arbres associés. On teste très souvent cette égalité au cours de la recherche de preuve. Le faire sur des arbres est long (de l'ordre du nombre de nœuds du plus petit). On préfère, au cours de l'indexation, construire un deuxième tableau, qui à chaque numéro de formule associe un numéro de *classe*, à comprendre comme les classes d'équivalence de l'égalité structurelle ; voir a figure 4. Ainsi, on détermine ensuite en temps constant si les formules associées à deux entiers donnés sont égales structurellement.

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

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

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

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

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

3.5 Informations supplémentaires pour un test rapide des axiomes

Les axiomes sont testés à chaque appel récursif sur un nouveau séquent. L'axiome id notamment est long à tester naïvement : il faudrait tester, pour tout couple (A, B) de formules

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

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

4 Perspective de certification

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

4.1 Un langage simple pour faciliter la certification

Un premier pas vers la certification est l'utilisation d'un langage relativement simple, afin de limiter le nombre de propriétés à démontrer. Le langage que nous avons employé pour cette raison, qu'on notera $\boldsymbol{T}$, comporte un seul type de données : des arbres binaires, qui consistent en l'arbre vide ou une feuille étiquetée par un entier naturel ou un couple d'arbres. Les expressions sont données dans la figure 5, *var* désignant une variable et *nomf* un nom de fonction, des chevrons $<.,.>$ étant utilisés pour les couples d'arbres. La condition du **if** doit être une feuille d'un entier n, correspondant à *faux* si $n = 0$ et *vrai* sinon ; **isnull**, **isint** et la comparaison renvoient de même une feuille contenant 0 ou 1. La première expression du **match** doit s'évaluer à un couple, sinon il y a une erreur à l'exécution. De même, dans la comparaison ou dans **succ** (successeur) ou **pred** (prédécesseur), les expressions doivent être des feuilles. Une déclaration de fonction est un triplet de la forme (*nomf*, *var*, *expr*) comportant le nom de la fonction, celui de son argument, et enfin son corps dans lequel l'argument peut apparaître comme une variable libre. Un programme consiste en une liste de déclarations de fonctions considérées comme mutuellement récursives, puis une expression à évaluer.

L'intérêt d'utiliser ce langage est que mon encadrant D. Larchey-Wendling a réalisé pour celui-ci un compilateur certifié vers une machine abstraite, ainsi qu'un interpréteur certifié de cette machine abstraite. Son objectif est justement l'écriture grâce à ce langage d'un prouveur certifié.

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

FIGURE 5 – Les expressions du langage $\boldsymbol{T}$

4.2 Compilation d'un programme spécifique à une formule donnée

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

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

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

4.3 Comparaison de différentes implémentations

Conclusion

Références

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

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