N° d'ordre : 7986

Thèse de doctorat

présentée à

L'Université de Paris-Sud

U.F.R. Scientifique d'Orsay

par

Pierre Corbineau

pour obtenir

le grade de docteur en sciences
de l'Université de Paris XI Orsay
spécialité : Informatique

Sujet :

Démonstration automatique en Théorie des Types

Soutenue le 30 septembre 2005 devant la commission d'examen composée de :

Mme	Rozoy Brigitte	(Présidente)
M.	Bezem Marc	
M.	Monin Jean-François	(Rapporteurs)
M.	Dyckhoff Roy	
M.	Gonthier Georges	
M.	Marché Claude	
Mme	Paulin Christine	

Démonstration Automatique en Théorie des Types

Pierre Corbineau

septembre 2005

Remerciements

Je tiens tout d'abord à remercier Christine Paulin qui a su me donner de judicieux conseils à des moments cruciaux de mon travail de thèse. Je souhaite également la remercier pour la qualité des discussions scientifiques que nous avons eues. Ensuite, je remercie Judicaël Courant d'avoir encadré mon travail pendant les deux premières années et d'avoir su guider mes activités doctorales.

Je remercie tout particulièrement Claude Marché d'avoir pris la succession de Judicaël Courant pour m'aider à mener à bien cette thèse. Je le remercie notamment pour le soutien et la rigueur qu'il a su mettre dans la relecture de mes articles et de mon mémoire.

Je remercie les membres du jury d'avoir accepté de se pencher sur mes travaux ; et en premier lieu les rapporteurs, Marc Bezem et Jean-François Monin, que je remercie pour leur lecture attentive et leurs remarques et questions pertinentes. Je remercie Roy Dyckhoff, dont les articles furent une importante source d'inspiration, d'avoir accepté d'être présent pour ma soutenance. Je remercie également Georges Gonthier d'avoir accepté de participer à ce jury et Brigitte Rozoy, avec qui j'ai effectué mes premiers enseignements, d'avoir accepté d'en être la présidente.

Mes remerciements vont aussi aux personnes que j'ai cotoyées au sein des équipes Démons et LogiCal durant ces années : Évelyne avec qui la collaboration fut à la fois fructueuse et agréable, Jean-Christophe jamais à court d'idées pour les algorithmes et structures de données, Hugo qui fut mon guide dans les arcanes de code source de Coq, mais également tous les autres, notamment mes collègues doctorants Xavier, Benjamin, les deux autres Pierre et les trois Julien, Nicolas, June, Thierry, Clément, Olivier, François-Régis ainsi que les chercheurs : Sylvain, Bruno, Gilles, Ralf, Delia et Jean-Pierre.

Je tiens également à exprimer ma reconnaissance envers Joëlle Despeyroux et Jean Goubault qui, en encadrant mes stages de Maîtrise et de DEA, furent les artisans de mon initiation à l'usage du système Coq. De façon plus large, je tiens à saluer les professeurs de sciences qui, notamment au lycée, ont développé ma curiosité scientifique en tâchant de la satisfaire.

Je remercie mes parents qui, après avoir fait avec moi le pari des classes préparatoires, ont continué à me soutenir et à suivre mon parcours, malgré le caractère abscons et peu tangible de mes études universitaires.

Je remercie Michèle et Henri de m'avoir accueilli chez eux pour des périodes de vacances, studieuses ou non, qui m'ont permis de couper avec le stress parisien.

Enfin, je remercie Hélène, pendant ces années de thèse, de m'avoir soutenu grâce à une écoute attentive, d'avoir su me faire partager avec elle mes moments de doutes et d'avoir supporté avec patience les périodes où le travail m'absorbait totalement.

Table des matières

Introduction

Les débuts de la logique formelle

Depuis la fin du XIX$^{\text{ème}}$ siècle, de nombreux philosophes et mathématiciens se sont penchés sur la question suivante : qu'est-ce qu'un raisonnement logique ? En effet, les progrès des mathématiques, notamment en analyse numérique et en algèbre, s'étaient accomplis en se fondant sur une notion *intuitive* de ce qu'est une démonstration valide. Le développement de théories mathématiques de plus en plus complexes souleva alors chez les philosophes la question de la validité de la démarche mathématique : ils remirent en question ses bases logiques.

En réponse à ce questionnement, plusieurs philosophes et mathématiciens proposèrent des descriptions très précises de ce qu'était pour eux un raisonnement logique, en créant un *langage* pour exprimer des concepts et propriétés logiques ainsi que des règles permettant de déduire la validité d'une proposition de celle d'autres propositions. On considère que c'est Gottlob Frege, dans son ouvrage de 1879 intitulé *Begriffschrift*, qui donna la première présentation formelle du calcul des prédicats du premier ordre, ou logique du premier ordre. Son second ouvrage, *Grundsetze der Arithmetik* [Fre03], étendit et précisa ce système.

Même si Bertrand Russell démontra l'incohérence du système formel de Frege, celui-ci inspira de nombreux travaux. Le mathématicien David Hilbert établit alors un programme formaliste destiné à encadrer les recherches dans le domaine des fondements de mathématiques. En 1933, le théorème de complétude de Kurt Gödel [Göd34] permit de fixer les bases conceptuelles de la logique du premier ordre.

Ce théorème établissait qu'il existait un système de déduction D simple et effectif (calculable) tel que toute proposition valide de la logique du premier ordre, c'est-à-dire satisfaite dans tous les modèles booléens, était démontrable dans ce système D. En revanche, son théorème d'incomplétude de l'arithmétique du premier ordre mettait à bas le programme de Hilbert en établissant l'impossibilité de créer un système formel «idéal».

En 1934, Gerhard Gentzen publia le livre *Untersuchungen über das logische Schliessen* [Gen34] qui fonda la théorie de la démonstration en définissant la notion de règle de déduction et de séquent. Ces concepts permirent de prouver facilement les propriétés de systèmes déductifs. Gentzen proposa ainsi les calculs de séquents LK et LJ pour le calcul des prédicats respectivement classique et intuitionniste. La logique intuitionniste [Hey56] se caractérise principalement par l'impossibilité de prouver qu'une propriété A quelconque satisfait le tiers exclus : « ou bien A est vrai ou alors A est faux ».

L'approche purement automatique

Un succès indéniable

L'avènement de l'informatique, dans les années 1950-1960, provoqua l'apparition de systèmes de démonstration automatique (*automated theorem provers*). Malgré les résultats d'Alan Turing sur l'indécidabilité de la logique du premier ordre, les informaticiens parvinrent à mettre au point des méthodes efficaces en pratique pour résoudre partiellement ce problème indécidable.

La méthode de démonstration automatique la plus populaire fut incontestablement la *résolution* [Rob65], que l'on doit à J. A. Robinson. Cette technique, particulièrement adaptée à la logique classique, passe par une réduction du problème considéré à une forme spécifique appelée forme clausale. Elle utilise ensuite une unique règle de déduction, la règle de résolution, pour démontrer que la négation de la proposition de départ permet de dériver la clause vide et donc que cette négation est contradictoire.

La popularité de la résolution fut telle qu'elle conduisit à l'invention de la programmation logique, fondement théorique du langage Prolog [KvE76]. Un programme Prolog se compose d'un ensemble de clauses, et l'exécution de ce programme est l'application de l'algorithme de résolution à cet ensemble de clauses avec une stratégie bien précise.

La résolution du premier ordre et ses avatars constituent encore aujourd'hui la base d'un nombre important de systèmes de recherche de preuve. Une autre méthode très utilisée est la méthode des tableaux. Cette approche consiste à optimiser la recherche de dérivations d'une proposition donnée dans un calcul de séquents, en accumulant des contraintes d'unification, puis à essayer dans un second temps de résoudre ces contraintes avec une stratégie déterministe.

Limites des méthodes automatiques

Malgré le succès de la résolution et des tableaux, ces méthodes souffrent de la faiblesse du langage de base : la logique du premier ordre. En premier lieu, on rencontre de manière quasiment systématique le prédicat d'égalité, or la résolution fonctionne très mal en présence de la transitivité de l'égalité. Ce problème a conduit à proposer des règles de déduction spécifiques pour l'égalité, comme la paramodulation [RW69], et de nombreux travaux sur la logique équationnelle [NR01].

En second lieu, la résolution et la paramodulation, ainsi que la méthode des tableaux, déstructurent complètement le problème pour arriver à le résoudre. En effet, en cas d'échec, il est difficile de remonter à la cause de cet échec, et en cas de succès, il est également difficile de comprendre pourquoi la proposition que l'on veut prouver est valide. Enfin, ces méthodes ne permettent pas de continuer « à la main » des démonstrations de propriétés simplifiées automatiquement.

En troisième lieu, la logique du premier ordre considère la validité dans tous les modèles, or on souhaite parfois raisonner dans des modèles spécifiques : l'exemple le plus usité — notamment pour la preuve de programmes — est l'arithmétique linéaire sur les entiers.

Pour ces théories plus complexes, on cherche plutôt à mettre au point des procédures spécifiques. Parmi les problèmes toujours à l'étude, citons d'une part le mélange de plusieurs théories [NO79, CK04], et d'autre part les raisonnements à base de récurrence qui permettent d'établir la validité des formules vraies dans la classe des modèles engendrés par des constructeurs.

De plus, la communauté mathématique reste très sceptique vis-à-vis de la capacité des ordinateurs à effectuer des raisonnements corrects. L'exemple le plus célèbre de cette réticence est le refus de considérer comme valide la vérification par ordinateur des cas irréductibles dans la preuve du théorème des quatre couleurs [AH76]. En effet, le programme en question ne fournit pas un objet témoignant de la véracité de la preuve et vérifiable indépendamment par l'homme ou par un autre programme, c'est-à-dire qu'il ne satisfait pas ce que l'on appelle le critère de deBruijn. La preuve formelle récente du théorème des quatre couleurs par George Gonthier et Benjamin Werner [Gon] satisfait ce critère.

L'approche interactive

Parallèlement à l'essor des démonstrateurs automatiques de théorèmes, d'autres logiciels furent développés afin de résoudre le problème — décidable — de la vérification de preuve, et ce pour des langages et des systèmes déductifs de plus en plus complexes.

Mentionnons notamment le vérificateur de preuve de Boyer et Moore qui, bien que basé sur la résolution, permet à l'utilisateur de donner des directives pour guider le système de preuve. On peut considérer ce logiciel comme un vérificateur de preuve au sens où l'ensemble de ces directives constitue la preuve. Cependant la forme des preuves acceptables dépend fortement de l'implantation du système et par conséquent il satisfait difficilement le critère de deBruijn.

Nous allons maintenant décrire une classe de systèmes de preuve dont l'architecture est beaucoup plus uniforme et qui, eux, satisfont le critère de deBruijn : les assistants de preuve fondés sur la théorie des types.

Du λ-calcul à la Théorie des Types

Afin d'éviter l'incohérence que Frege avait introduit dans son système, Russell et Whitehead proposèrent dans le *Principia mathematica* [WR10] une *Théorie des types* reposant sur une stratification des formules — chaque strate étant désignée par un type — et l'interdiction des substitutions qui ne respecteraient pas cette stratification. Après la découverte du λ-calcul [Bar84] par Alonzo Church, Curry et Howard [How80]. parvinrent à établir une correspondance entre les règles de déduction logique et certains systèmes de types conçus pour ce calcul. Per Martin-Löf s'appuie sur cet isomorphisme pour proposer une Théorie des Types Intuitionniste [ML84] plus riche que celle de Russell et Whitehead : elle permet de quantifier sur tous les ensembles — et ainsi d'exprimer la théorie des catégories — et elle est plus proche des langages de programmation : les formules logiques peuvent contenir des

termes du λ-calcul, expressions que l'on peut calculer. Notre travail se sert particulièrement de ce second aspect.

Le λ-calcul est une théorie de fonctions construites par application et abstraction sur une variable. Sa description, bien que relativement simple, conduit à une théorie des modèles très complexe et très riche. Le langage du λ-calcul est défini par :

$$\mathcal{L} := x \mid (\mathcal{L}\,\mathcal{L}) \mid \lambda x.\mathcal{L}$$

où $\lambda x.u$ désigne la fonction qui à x associe u et $(f\,c)$ désigne l'application de la fonction f à l'objet c. Ce langage est muni de la règle de calcul suivante : un terme de la forme $(\lambda x.u)t$, appelé β-redex, se réduit en $u\{t/x\}$, c'est-à-dire le terme u dans lequel x a été remplacé par t. Cette règle de calcul s'appelle la β-réduction. Grâce à ce système de calcul, il est possible de simuler une machine de Turing et, en particulier, on peut écrire un terme dont le calcul ne termine pas, par exemple $\Omega = (\lambda x.(x\,x)\,\lambda x.(x\,x))$.

Un système de types pour le λ-calcul est la donnée d'un ensemble d'objets, appelés types, permettant d'annoter les λ-termes et accompagnés de contraintes donnant le type d'un terme en fonction des types de ses sous-termes. Ainsi, l'expression $\Gamma \vdash t : \tau$ se lit « Le terme t a le type τ dans le contexte Γ ». Parmi les systèmes des types existants, un certain nombre ont la propriété de garantir la terminaison des termes bien typés.

La Théorie des Types repose sur la découverte suivante : il est possible de choisir un système de types dont les types sont des propositions logiques, et tel que les règles de typage n'autorisent que des déductions correctes. Le système le plus simple pour lequel ces remarques s'appliquent est le λ-calcul simplement typé : les types sont formés à partir d'un ensemble de types atomiques $\alpha, \beta...$ et de l'opérateur $\rightarrow$. Les règles de base de ce système sont :

$$\frac{\Gamma, x : \alpha \vdash u : \beta}{\Gamma \vdash \lambda x.u : \alpha \rightarrow \beta}\ \mathit{Abs} \qquad \frac{\Gamma \vdash t : \alpha \rightarrow \beta \quad \Gamma \vdash u : \alpha}{\Gamma \vdash (t)u : \beta}\ \mathit{App}$$

La règle *Abs* dit que si un terme u a le type β, sachant que la variable x a le type α, alors on peut donner au terme $\lambda x.u$ le type $\alpha \rightarrow \beta$. À l'inverse, la règle *App* dit que si l'on applique une fonction de type $\alpha \rightarrow \beta$ à un terme de type α, on obtient un terme de type β. Dans ce système, on peut lire le symbole $\rightarrow$ comme l'implication logique et identifier les types atomiques à des propositions logiques. On obtient alors, en effaçant les termes dans les règles *Abs* et *App*, deux règles de raisonnement logique :

$$\frac{\Gamma; \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta}\ \rightarrow_I \qquad \frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta}\ \rightarrow_E$$

Grâce à cette correspondance, on déduit que s'il existe un terme du λ-calcul tel que $\vdash t : \tau$, alors τ est une tautologie propositionnelle intuitionniste. Cette remarque constitue la base de l'isomorphisme de Curry-Howard-deBruijn, qui met en correspondance ces deux lectures du λ-calcul : d'abord comme un langage de programmation, puis comme un système logique. Il permet ainsi d'identifier preuves et programmes, de même que types de données et propositions.

En 1969, Howard proposa d'étendre le système de types par des types dépendants, c'est-à-dire des types annotés par des termes du λ-calcul. Il permit ainsi d'étendre l'isomorphisme à la logique intuitionniste du premier ordre. Le système F de Jean-Yves Girard apporta l'extension au second ordre, puis le système $F\omega$ et le Calcul des Constructions [Coq85] (Thierry Coquand) permirent de traiter la logique d'ordre supérieur.

Parallèlement, des outils logiciels furent développés pour profiter de ces avancées théoriques. Le premier vérificateur de preuve en théorie des types s'appelle *Automath* [dB80], il fut écrit en 1968 et est dû à N. G. deBruijn. Le défaut principal de cette première tentative est qu'il fallait saisir à la main l'intégralité des termes de preuve à vérifier. Le système LCF, fondé sur un formalisme de Dana Scott et dont le langage de preuve est à l'origine des langages de programmation fonctionnels modernes tels Standard ML [MTHM97] ou Objective Caml [Obj], apporta les concepts d'architecture logicielle encore utilisés de nos jours, notamment la saisie de preuve par composition de commandes appelées *tactiques* qui se réduisent à un petit nombre d'opération élémentaires dont l'interface, lesquelles sont vérifiées par un noyau de taille réduite (critère de deBruijn). *LCF* utilise des types abstraits pour empêcher la création de faux théorèmes en dehors de ce noyau.

Dans les années 80, deux familles de prouveurs basées sur les systèmes de types ont vu le jour. D'une part, les *Logical Frameworks* [Pfe91], qui utilisent la Théorie des Types comme une *méta-logique* permettant de décrire les règles d'inférence d'une logique axiomatisée appelée *logique objet*. Ces systèmes offrent l'avantage d'une grande souplesse malgré leur caractère axiomatique. Les outils les plus connus dans cette famille sont Isabelle [Isa],Agda [Coq00] et plus récemment TweLF [PS00].

L'autre famille, elle, a tiré parti de la richesse expressive des systèmes de types pour construire des systèmes de preuve qui sont en fait des vérificateurs de types pour la Théorie des Types de Martin-Löf [ML84] (notamment NuPRL [Kre02]) pour le Calcul des Constructions (Lego [Leg] et Coq [Coq04, BC04]). Ces derniers outils sont fondés sur un Calcul des Constructions étendu par des types de données dits *types inductifs* : le Calcul des Constructions Inductives qui est le calcul à la base du système Coq [Coq].

Logique et calcul : le système Coq

Le système Coq est composé de deux parties bien distinctes : d'une part, le noyau, de taille réduite, qui contient uniquement l'algorithme de vérification de types dans le Calcul des Constructions Inductives, et d'autre part, le moteur de preuve, qui permet d'indiquer à Coq comment construire le λ-terme de preuve à l'aide de commandes.

Comme dans toute extension du λ-calcul, les termes du Calcul des Constructions Inductives peuvent se réduire, c'est-à-dire que l'on peut effectuer des calculs à l'aide de ces termes. Par exemple, on peut formaliser l'arithmétique de Peano [Pea89] de façon à ce que le terme $2 + 2$ puisse se réduire sur le terme 4. Ceci entraîne que les termes apparaissant dans les types dépendants peuvent eux aussi se réduire par calcul pour donner d'autres termes. Si nous lisons ces types dépendants comme des propositions logiques, cela implique que l'on peut par calcul obtenir des propositions logiques différentes : par exemple, si Pair

est un prédicat sur les entiers, alors la proposition Pair$(2+2)$ se réduit par calcul sur la propriété Pair(4).

Pour tirer parti de ce constat, le Calcul des Constructions Inductives possède une règle de typage appelée règle de conversion qui dit que deux types équivalents par calcul — c'est-à-dire pouvant se réduire sur un même type — sont des types pour les mêmes termes. Autrement dit, un terme de type τ est également un terme de tous les types équivalents à τ. Ainsi, une preuve de Pair$(2+2)$ sera également une preuve de Pair(4) et vice-versa, ce qui peut être utile si l'on a prouvé par exemple que $\forall x.\,\text{Pair}(x+x)$ et que l'on veut prouver Pair(4). Ainsi, il existe dans Coq un nombre important de tactiques qui ne construisent pas le terme de preuve mais modifient uniquement la propriété à prouver en effectuant des calculs dans son énoncé.

Limites de l'approche interactive

L'expressivité du Calcul des Constructions Inductives permet de s'assurer plus facilement de l'adéquation entre les propriétés formelles énoncées dans le système et les idées informelles de l'utilisateur. En effet, plus un formalisme est expressif et plus les concepts informels de l'utilisateur s'y expriment de façon naturelle, assurant ainsi l'adéquation entre visions formelle et informelle. Cette expressivité a un inconvénient de taille : les étapes de raisonnement interactif sont élémentaires et effectuer une preuve peut s'avérer extrêmement pénible et répétitif.

De plus, la puissance de la logique rend l'automatisation difficile. En effet, dans le cas du Calcul des Constructions Inductives et des autres calculs avec types dépendants, on se heurte à l'indécidabilité de l'unification d'ordre supérieur : on sait vérifier si deux termes sont équivalents par calcul, mais on ne sait pas si l'on peut donner aux variables de deux termes des valeurs rendant ces termes équivalents. Or, l'unification est à la base de la plupart des méthodes de recherche de preuves, que ce soit par résolution ou par tableaux. On en est alors réduit à utiliser des variantes de l'unification du premier ordre, qui sont incomplètes.

En outre, la logique induite par le Calcul des Constructions Inductives est intuitionniste, comme dans le cas du λ-calcul simplement typé. Or la résolution, comme beaucoup d'autres méthodes, s'adapte très mal aux logiques constructives. Plus encore, les connecteurs logiques habituels ($\wedge$, $\vee$, $\exists\ldots$) ne sont pas primitifs dans le calcul mais sont définis par des constructions inductives (voir chapitre 3). Une direction naturelle pour automatiser la construction de termes de preuve est d'utiliser les méthodes automatiques connues, sur les fragments logiques concernés. Mais alors celles-ci doivent être instrumentées de manière à produire des termes de preuve ou toute autre forme de *trace* de preuve revérifiable automatiquement dans Coq. Les premiers fragments ainsi automatisés dans Coq ont été le calcul propositionnel — César Muñoz [Muñ95] a développé la tactique `tauto` qui produit directement un terme de preuve —, et l'arithmétique linéaire sur $\mathbb{Z}$ — Pierre Crégut a développé la tactique `omega` qui fournit une preuve par application successive de lemmes arithmétiques élémentaires.

On peut noter la tentative de Huang visant à automatiser toute la logique du premier ordre en utilisant la procédure de semi-décision `Jprover` et en produisant directement des termes de preuve. Cuihtlauac Alvarado [Alv02] a proposé une automatisation des preuves par réécriture, en produisant une trace d'une forme spécifique dont la revérification passait par l'application d'un lemme général de validité des traces, ce que l'on appelle *principe de réflexion*. Pierre Crégut a récemment proposé une version de `omega` utilisant un tel principe [Cré01]. Samuel Boutin [Bou97] a utilisé une forme plus avancée de réflexion pour automatiser certaines théories équationnelles en prouvant la correction de l'algorithme de décision plutôt que celui de vérification de trace.

Preuves par réflexion

Réflexion logique

La *réflexion logique* est un outil dont la principale application fut la preuve du théorème d'incomplétude de Gödel. Gödel utilisa pour prouver son théorème un encodage concret des formules logiques à l'aide d'entiers. Dans le cadre d'un système formel, cet encodage appelé *réification* permet d'exprimer une propriété logique ϕ à l'aide d'un objet du système formel que l'on note $\dot{\phi}$. Gödel se plaça dans l'arithmétique du premier ordre et utilisa des entiers — nommés par la suite nombres de Gödel — pour les représenter.

Gödel parvint alors à définir une représentation pour une dérivation de son système formel, c'est-à-dire un encodage p de la preuve d'une propriété ϕ. Il relia cette représentation à celle des formules par la définition d'un prédicat $Prov$ représentant la relation « être une preuve valide de la formule ... ». À l'aide de $Prov$, on peut construire le prédicat de démontrabilité $Dem(\dot{\phi}) \equiv \exists p.Prov(p, \dot{\phi})$.

Dès lors, on va s'intéresser à deux types de propriétés de cet encodage :

– la correction : toute formule ϕ telle que $Dem(\dot{\phi})$ est une formule valide.

– la complétude : toute formule valide ϕ vérifie $Dem(\dot{\phi})$.

Dans un système du premier ordre, ces propriétés ne sont pas exprimables à l'intérieur du système car elles nécessiteraient de pouvoir quantifier sur les formules logiques, or on ne peut quantifier que sur les objets. De plus, la réification d'une formule n'est pas exprimable à l'aide d'une fonction ou d'un prédicat fonctionnel du premier ordre puisqu'une telle fonction ou un tel prédicat serait appliqué à une formule.

Les propriétés de correction et de complétude ne sont donc pas des propriétés *dans* le système formel mais *à propos* du système formel, c'est pourquoi on parle parfois de méta-propriétés et de méta-théorèmes. La propriété de correction est celle qui va nous intéresser ici, et c'est celle que l'on nomme le plus souvent *principe de réflexion*.

Réflexion calculatoire

Dans son article récapitulatif *Metatheory and Reflection in Theorem Proving : A Survey and Critique* [Har95], John Harrison explique que lorsque l'on se place dans le cadre d'une logique d'ordre supérieur, la distinction entre propriétés et méta-propriétés disparaît. Il

explique ainsi ce phénomène : on entend souvent par propriétés celles exprimables dans le cadre de la logique ou de l'arithmétique du premier ordre, et les méta-propriétés à propos de ces fragments sont souvent suffisamment simples pour entrer dans le cadre de la logique d'ordre supérieur.

Cela permet, *dans* la logique d'ordre supérieur, de prouver des résultats *à propos* d'un fragment du premier ordre de cette logique. C'est notamment possible pour le principe de réflexion modifié que nous allons décrire maintenant.

Dans un principe de réflexion calculatoire, la quantification ne porte plus sur les propositions logiques, mais sur leur représentation par les objets de la logique. Pour cela, il est nécessaire d'utiliser une fonction réciproque de la réification : l'interprétation réflexive, que l'on note $[\![\_]\!]$, et qui vérifie $[\![\dot{\phi}]\!] = \phi$ pour toute formule ϕ réifiable. Par ce moyen, le champ d'application du principe de réflexion est restreint à l'image de la fonction d'interprétation. Dans ce contexte, le principe de réflexion modifié s'énonce de la façon suivante :

$$\forall F, (\exists \pi, Prov(\pi, F)) \rightarrow [\![F]\!]$$

Supposons que l'on ait défini le prédicat $Prov$ de sorte que si π est une preuve de F, $Prov(\pi, F)$ donne par calcul la propriété triviale $\top$ par calcul, on pourra par ce moyen obtenir une preuve de $[\![F]\!]$ par un calcul : la vérification de la trace π.

Vers une combinaison des approches interactive et automatique

La première application des assistants de preuve tels que Coq a été la formalisation des preuves de théorèmes mathématiques, mais ces assistants de preuve sont également utilisés pour la certification de logiciels, c'est-à-dire pour la preuve de programmes. Dans ce genre d'applications, les problèmes rencontrés sont relativement simples du point de vue conceptuel puisqu'ils font appel à des notions mathématiques élémentaires, mais complexes du point de vue de la preuve automatique dans la mesure où ils font appel à une combinaison d'arithmétique et de théorie des structures de données (tableaux, listes chaînées, etc.), et sont souvent de grande taille (beaucoup d'hypothèses dont peu sont utiles à la preuve).

Pour résoudre ce genre de problèmes, il existe un grand nombre de démonstrateurs automatiques spécialisés dont les résultats sont remarquables. Face au succès de ces outils automatiques, les développeurs ont adopté des attitudes différentes. Tandis que les uns optent pour la confiance aveugle vis-à-vis de la réponse de ces outils, les considérant comme des oracles, les autres optent pour l'approche sceptique : ils ne font confiance à la réponse d'un outil externe que si celui-ci fournit à l'assistant une trace vérifiable de la validité de sa réponse. Cette attitude sceptique est celle adoptée par l'outil Coq.

Apports de la présente thèse

Le but de nos travaux est d'accroître l'automatisation de la preuve de théorèmes en Théorie des Types, avec deux objectifs majeurs : la couverture de fragments logiques les plus larges possibles, et l'utilisation de la réflexion pour construire de façon flexible et efficace des preuves à partir de traces fournies par les procédures automatiques. Nous commençons, dans le chapitre 1, par présenter brièvement les notions de base utilisées dans cette thèse, notamment la logique du premier ordre et le Calcul des Constructions Inductives. Les trois chapitres suivants présentent trois contributions distinctes vers l'objectif fixé. Enfin, nous concluons et envisageons des perspectives dans le dernier chapitre.

Preuves d'égalités entre termes clos avec constructeurs

Dans le Calcul des Constructions Inductives, les constructeurs de types inductifs et co-inductifs possèdent des propriétés spécifiques : injectivité et élimination forte (deux termes construits par application de constructeurs distincts ne peuvent être égaux). Ces propriétés induisent une théorie équationnelle particulière. De plus le Calcul des Constructions Inductives permet d'écrire des égalités entre objets de type fonctionnel et d'appliquer partiellement des symboles de fonction.

Dans le chapitre 2, nous formalisons la notion de termes avec application partielle et celle de constructeur et nous montrons que la satisfaisabilité d'une conjonction d'égalités et de diségalités entre termes clos avec constructeurs peut être réduite à l'existence d'un modèle pour une partie finie de l'univers des termes. Nous expliquons comment l'algorithme de clôture de congruence peut être étendu pour tenir compte de la sémantique des types inductifs en propageant les conséquences équationelles de l'injectivité et dérivant des contradictions par élimination forte, afin de rechercher un tel modèle, ou, au contraire, de dériver une contradiction.

Nous prouvons la correction et la complétude de cet algorithme et décrivons son implantation dans le système Coq sous la forme d'une tactique : `congruence`.

Preuves intuitionnistes dans le Calcul des Constructions

L'utilisation dans Coq du calcul de séquents sans contraction LJT (ou $G4ip$) de Roy Dyckhoff est à la base de la procédure de décision `tauto` pour la logique intuitionniste propositionnelle. Pour traiter la logique du premier ordre, Dyckhoff a étudié l'extension $G4i$ de ce système à la logique du premier ordre. Ces calculs de séquents sont limités au langage logique utilisant les connecteurs standard, sans tenir compte des possibilités offertes par les types inductifs de Coq.

Dans le chapitre 3, nous expliquons comment étendre le langage logique de manière à traiter de façon uniforme les connecteurs logiques définis par des types inductifs non-récursifs. Nous appliquons à ce système la technique des calculs de séquents sans contraction. Nous établissons les propriétés fondamentales de ce nouveau système : élimination

des contractions (théorème 3.15) et des coupures (théorème 3.17). Ces résultats nous permettent de dériver une procédure de recherche de preuve en logique du premier ordre, `firstorder` dont nous décrivons les principes de base.

Nous étendons également ces résultats à une extension du calcul avec le prédicat d'égalité (théorème 3.29).

Preuves réflexives en logique du premier ordre avec égalité

L'interfaçage de Coq avec des procédures de décision externes s'avère souvent complexe en raison d'une part de la différence entre les formalismes et entre les formats de traces, et d'autre part en raison de la taille excessive de ces traces, qui peut nécessiter une quantité importante de ressources — temps de calcul et espace mémoire — pour être vérifiée.

Dans le chapitre 4, nous décrivons une architecture générique permettant de définir une structure d'arbres de preuve ainsi qu'une fonction vérifiant leur validité et nous utilisons une preuve de la correction de cette procédure comme un principe de réflexion. Dans un premier temps, nous nous limitons à la logique propositionnelle, et nous décrivons l'implantation d'une variante `rtauto` de la tactique `tauto` utilisant ce principe de réflexion pour produire ses preuves.

Ensuite nous montrons comment l'on peut passer au premier ordre dans le cadre d'une logique typée comme celle induite par le Calcul des Constructions Inductives, et comment nous traitons le cas de l'égalité. Le résultat principal est le théorème 4.10 qui établit la correction de notre schéma de réflexion.

Nous expliquons également comment ce cadre permet l'interprétation de traces de réécriture obtenues lors de la résolution de problèmes équationnels par complétion ordonnée. Les traces vérifiées permettent d'obtenir un certificat de validité du calcul de complétion ; la génération de ces traces est validée par une série de tests sur des problèmes de la librairie TPTP.

Chapitre 1

Éléments de logique du premier ordre et de Théorie des Types

Dans ce chapitre, nous rappelons d'abord les définitions des notions de base que sont les termes et les formules du premier ordre, et ensuite les notions de système d'inférence et de calcul de séquents afin de décrire le système LJ de Gentzen. Enfin, nous donnons une courte introduction au Calcul des Constructions Inductives et au système Coq.

1.1 Logique du premier ordre

1.1.1 Termes du premier ordre

Pour construire un langage du premier ordre, il faut d'abord disposer d'un ensemble de sortes dans lesquelles on peut définir les termes. Ces sortes sont le support d'un ensemble de fonctions que l'on représente par des symboles regroupés dans une signature.

Définition 1.1 (signature)
Une signature du premier ordre est une paire $\mathcal{S}, \Sigma$, où $\mathcal{S}$ est un ensemble fini d'objets appelés sortes *et Σ un ensemble de triplets contenant chacun :*

- *un symbole de fonction f.*
- *une liste (finie) de sortes $[s_1, \ldots, s_p]$ représentant son domaine, c'est-à-dire la liste des sortes des arguments de la fonction représentée par le symbole f. L'entier p est appelé arité de f.*
- *une sorte s représentant le codomaine de la fonction f, c'est-à-dire le domaine des valeurs prises par la fonction f.*

Pour simplifier les notations, nous supposerons dans la suite $\mathcal{S}$ et Σ fixés et nous noterons $f : (s_1, \ldots, s_n) \rightarrow s$ au lieu de $\langle f, [s_1, \ldots, s_n], s\rangle \in \Sigma$. Les symboles d'arité 0 seront appelés constantes.

Définition 1.2 (terme)
On suppose que l'on dispose pour chaque sorte $s \in \mathcal{S}$ d'un ensemble infini $\mathcal{X}_s$ de symboles

distincts des symboles de fonctions de Σ tels que $\mathcal{X}_s \cap \mathcal{X}_{s'} = \emptyset$ si $s \neq s'$. Un élément de $\mathcal{X}_s$ est appelé variable de sorte s. Un terme de sorte s est un arbre fini t dont la racine est étiquetée par un symbole qui peut être :

- *soit une variable $x \in \mathcal{X}_s$, et alors l'arbre est réduit à une feuille.*
- *soit un symbole de fonction $f : (s_1, \ldots, s_n) \rightarrow s$, et alors la racine de l'arbre doit avoir n fils $t_1, \ldots, t_n$ qui doivent eux-mêmes être des termes de sorte respective $s_1, \ldots, s_n$.*

Pour désigner les sous-termes d'un terme, on a besoin de pouvoir noter leur position. Pour définir ces positions, plusieurs solutions sont possibles, voici celle que nous avons retenue ici :

DÉFINITION 1.3 (POSITION)
Une position est une liste finie d'entiers $i_1; \ldots; i_n$, potentiellement vide (on note Λ la liste vide). On note $u|_p$ le sous-terme de u à la position p, que l'on définit ainsi :

- *$u|_\Lambda = u$*
- *Si $p = i; q$ et $u = f(t_1, \ldots, t_n)$ avec $i \leq n$, alors $u|_p = t_i|_q$.*

Dans tous les autres cas, la position est invalide pour le terme u. On note $\mathrm{Pos}(u)$ l'ensemble des positions valides du terme u, et $u[v]_p$ le terme u dans lequel on a mis le terme v à la position p.

Pour pouvoir remplacer une variable par un terme dans un autre terme, on définit la notion de substitution :

DÉFINITION 1.4 (SUBSTITUTION)
Une substitution σ est une fonction de l'ensemble des variables dans l'ensemble des termes, telle que $\sigma(x) = x$ sauf pour un nombre fini de variables qui forment le domaine de σ, noté $\mathrm{Dom}(\sigma)$. On définit l'application $t\sigma$ de la substitution σ au terme t par les équations suivantes :

- *$x\sigma = \sigma(x)$*
- *$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$*

La composition de deux substitutions σ et τ s'écrit $\tau\sigma$, elle est définie par $(\tau\sigma)(x) = (\tau(x))\sigma$. En particulier on voit par définition que le domaine de $\tau\sigma$ est bien fini.

DÉFINITION 1.5 (VARIABLES INDÉPENDANTES)
On note $Fv(t)$ l'ensemble des variables apparaissant dans le terme t. On note $Fv(\sigma)$ l'ensemble défini par :

$$Fv(\sigma) = \bigcup_{x \in \mathrm{Dom}(\sigma)} \sigma(x)$$

Une variable x est dite indépendante d'une substitution σ si, et seulement si, $x \notin Fv(\sigma) \cup \mathrm{Dom}(\sigma)$.

1.1.2 Logique du premier ordre

Définition 1.6 (prédicat, formule atomique)
Pour définir les prédicats, on utilise une sorte distinguée $*$ n'apparaissant pas dans $\mathcal{S}$. Un prédicat est alors un symbole de fonction à valeurs dans cette sorte $*$, appelée sorte des propositions. On appelle formule atomique tout terme de sorte $*$.

Définition 1.7 (formule)
Une formule est un objet défini par la syntaxe suivante :

$$\begin{aligned} formule := \;& formule\ atomique \\ \mid\; & formule \rightarrow formule \\ \mid\; & formule \wedge formule \\ \mid\; & formule \vee formule \\ \mid\; & \bot \\ \mid\; & \forall_s x.formule \\ \mid\; & \exists_s x.formule \end{aligned}$$

Une formule est bien formée si toutes ses formules atomiques sont des termes de sorte $*$ utilisant le cas échéant les variables liées par les quantificateurs $\forall$ et $\exists$.

Dans une formule, le symbole $\bot$ désigne la proposition absurde, les symboles $\wedge$ et $\vee$ correspondent respectivement au *et logique* (conjonction) et au *ou logique* (disjonction). Le symbole $\rightarrow$ désigne l'*implication logique* : $A \rightarrow B$ est valide si B est toujours valide lorsque A est valide. Enfin les connecteurs $\forall_s$ et $\exists_s$ sont les quantificateurs respectivement universel et existentiel sur les objets de sorte s : $\forall_s x.Px$ exprime la validité de $P\,t$ quel que soit l'objet t de sorte s, tandis que $\exists_s x.Px$ exprime l'existence d'un objet t de sorte s tel que $P\,t$.

Définition 1.8 (variables libres, variables liées)
L'ensemble $Fv(F)$ des variables libres de la formule F est défini par les équations récursives suivantes :

$$\begin{aligned} Fv(P) &= \{x \mid \exists p \in Pos(P) : P|_p = x\} \text{ si } P \text{ est atomique} \\ Fv(A \rightarrow B) &= Fv(A) \cup Fv(B) \\ Fv(A \wedge B) &= Fv(A) \cup Fv(B) \\ Fv(A \vee B) &= Fv(A) \cup Fv(B) \\ Fv(\bot) &= \emptyset \\ Fv(\forall_s x.A) &= Fv(A) \setminus \{x\} \\ Fv(\exists_s x.A) &= Fv(A) \setminus \{x\} \end{aligned}$$

Les variables apparaissant dans F sans y être libres sont appelées variables liées dans F.

L'extension de la notion de substitution aux formules du premier ordre nécessite que l'on prenne des précautions à cause de la présence des quantificateurs. Une définition incorrecte peut conduire à la capture de variables comme le montre l'exemple de la variable y dans la figure 1.1.

DÉFINITION 1.9 (SUBSTITUTION SANS CAPTURE)
L'application $F\sigma$ d'une substitution σ (voir définition 1.4) à une formule F est définie par les équations suivantes :

$$\begin{array}{ll} P\sigma & \text{avec } P \text{ atomique, est défini comme dans la définition 1.4.} \\ A{\rightarrow}B & = A\sigma{\rightarrow}B\sigma \\ A \wedge B & = A\sigma \wedge B\sigma \\ A \vee B & = A\sigma \vee B\sigma \\ \bot\sigma & = \bot \\ (\forall_s x.A)\sigma & = \forall y.(F\{y/x\}\sigma) \\ (\exists_s x.A)\sigma & = \exists y.(F\{y/x\}\sigma) \\ & \quad \text{avec } y \notin Fv(F) \cup Fv(\sigma) \cup Dom(\sigma) \end{array}$$

Comme la définition de la substitution sans capture dépend du choix de la variablefraîche y, on définit une relation d'équivalence permettant d'identifier deux formules ne différant que par le nom de leurs variables liées.

DÉFINITION 1.10 (α-CONVERSION)
Deux formules sont dites α-convertibles si, et seulement si, elles sont équivalentes pour la plus petite relation $=_\alpha$ satisfaisant les conditions suivantes :

- *Pour toute formule F, $F =_\alpha F$.*
- *Pour toutes formules A, A', B et B' telles que $A =_\alpha A'$ et $B =_\alpha B'$, on a $A{\rightarrow}B =_\alpha A'{\rightarrow}B'$, $A \wedge B =_\alpha A' \wedge B'$ et $A \vee B =_\alpha A' \vee B'$.*
- *Pour toutes formules F, F' telles que $F =_\alpha F'$ et $y \notin (Fv(F) \cup Fv(F')) \setminus \{x\}$, on a $\forall x.F =_\alpha \forall y.F'\{y/x\}$ et $\exists x.F =_\alpha \exists y.F'\{y/x\}$.*

Par construction, si $F =_\alpha F'$, alors $Fv(F) = Fv(F')$, et la relation $=_\alpha$ est une relation d'équivalence passant au contexte, c'est-à-dire une congruence. En outre, toutes les formules possibles comme résultats d'une même substitution sont α-équivalentes (changer la variable y choisie revient à renommer une variable liée). C'est pourquoi les formules que l'on manipule sont en réalité des classes d'équivalence pour la relation $=_\alpha$, et cela donne un sens à la notion de substitution sans capture.

$$\begin{array}{ccc} (\exists y.y > x)\{y+1/x\} & \xrightarrow{\text{substitution avec capture}} & \exists y.y > y+1 \\ \Big\downarrow{\scriptstyle \text{renommage de la variable liée}} & & \neq \\ (\exists z.z > x)\{y+1/x\} & \xrightarrow{\text{substitution sans capture}} & \exists z.z > y+1 \end{array}$$

FIG. 1.1 – Substitution naïve et substitution sans capture

1.2 Systèmes formels

1.2.1 Système d'inférence

Définition 1.11 (jugement, instance)
Un jugement est un mot d'un langage $\mathfrak{J}$ muni d'une notion de substitution. Un jugement exprime une propriété logique. On appelle instance d'un jugement $\mathcal{J} \in \mathfrak{J}$ tout jugement $\mathcal{J}\sigma$ résultant de l'application d'une substitution σ au jugement $\mathcal{J}$.

Définition 1.12 (règle d'inférence)
Un règle d'inférence $\mathcal{R}$ est la donnée d'un nombre fini (éventuellement nul) de jugements $\mathcal{H}_1, \ldots, \mathcal{H}_n$ appelés prémisses de la règle et d'un jugement $\mathcal{C}$ appelé conclusion de la règle. On note alors une telle règle sous la forme d'une fraction :

$$\frac{\mathcal{H}_1 \quad \ldots \quad \mathcal{H}_n}{\mathcal{C}}\ \mathcal{R}$$

On appelle instance de la règle $\mathcal{R}$ toute règle formée d'instances des prémisses et de la conclusion par une même substitution σ :

$$\frac{\mathcal{H}_1\sigma \quad \ldots \quad \mathcal{H}_n\sigma}{\mathcal{C}\sigma}\ \mathcal{R}$$

Par abus de langage, une instance de $\mathcal{R}$ sera également notée $\mathcal{R}$.

Définition 1.13 (système d'inférence, dérivation)
Un système d'inférence ou calcul de dérivation $\mathfrak{C}$ est un couple $(\mathfrak{J}, \mathfrak{R})$ constitué d'un langage $\mathfrak{J}$ de jugements et d'un ensemble $\mathfrak{R}$ de règles d'inférence.

On appelle arbre de dérivation du calcul $\mathfrak{C}$ un arbre fini dont chaque noeud $\mathcal{N}$ est étiqueté par un couple $(\mathcal{C}, \mathcal{R})$ constitué d'un jugement et d'un nom de règle, et vérifie la propriété de bonne formation suivante :

Soient $\mathcal{H}_1, \ldots, \mathcal{H}_n$ les jugements étiquetant les fils de $\mathcal{N}$, alors la règle :

$$\frac{\mathcal{H}_1 \quad \ldots \quad \mathcal{H}_n}{\mathcal{C}}$$

est une instance de $\mathcal{R}$.

Un arbre de dérivation peut donc être vu comme un empilement d'instances de règles tel que la conclusion de chaque règle est une prémisse de la règle du noeud père :

$$\dfrac{\dfrac{\vdots \qquad\qquad \vdots}{\mathcal{H}_{1,1} \quad \ldots \quad \mathcal{H}_{1,n_1}}\ \mathcal{R}_1 \qquad \ldots \qquad \dfrac{\vdots \qquad\qquad \vdots}{\mathcal{H}_{n,1} \quad \ldots \quad \mathcal{H}_{n,n_1}}\ \mathcal{R}_n}{\mathcal{C}}$$

Les noeuds terminaux d'un tel arbre sont étiquetés par des règles sans prémisse. Le jugement étiquetant la racine d'un arbre est appelé conclusion de cet arbre.

On dit qu'un jugement $\mathcal{C}$ est dérivable dans le calcul $\mathfrak{C}$ s'il existe un arbre de dérivation de $\mathfrak{C}$ dont la conclusion est $\mathcal{C}$. Par abus de langage, un tel arbre est appelé dérivation du jugement $\mathcal{C}$.

DÉFINITION 1.14 (RÈGLE (FORTEMENT) ADMISSIBLE)
Soit $\mathfrak{C}$ un calcul de dérivation et soit une règle $\mathcal{R}$ dont le schéma est :

$$\frac{\mathcal{H}_1 \quad \dots \quad \mathcal{H}_n}{\mathcal{C}}\,\mathcal{R}$$

On dit que la règle $\mathcal{R}$ est admissible dans le calcul $\mathfrak{C}$ si, à partir de toute dérivation des hypothèses de $\mathcal{R}$ dans le calcul $\mathfrak{C}$, on peut construire, dans le calcul $\mathfrak{C}$, *une dérivation de la conclusion de $\mathcal{R}$, ce qui peut être illustré par le schéma suivant :*

$$\begin{array}{ccc} \Delta_1 & & \Delta_n \\ \vdots & \dots & \vdots \\ \mathcal{H}_1 & & \mathcal{H}_n \end{array} \quad \Longrightarrow \quad \begin{array}{c} \Delta \\ \vdots \\ \mathcal{C} \end{array}$$

De plus, la règle $\mathcal{R}$ est dite fortement admissible si la dérivation Δ a une hauteur inférieure à la plus grande des hauteurs des dérivations $\Delta_1, \dots, \Delta_n$.

Un exemple simple de règle fortement admissible est la règle de projection :

$$\frac{\mathcal{H}_1 \quad \dots \quad \mathcal{H}_n}{\mathcal{H}_k}\,\mathcal{R}$$

Un exemple de règle admissible mais non fortement admissible est, par exemple, la règle suivante :

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B \quad \Gamma \vdash C}{\Gamma \vdash A \wedge (B \wedge C)}\,\wedge\wedge_I$$

dans un système contenant la règle :

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_I$$

La transformation suivante montre l'admissibilité de $\wedge\wedge_I$ dans ce système :

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B \quad \Gamma \vdash C}{\Gamma \vdash A \wedge (B \wedge C)}\,\wedge\wedge_I \quad \Longrightarrow \quad \frac{\Gamma \vdash A \quad \dfrac{\Gamma \vdash B \quad \Gamma \vdash C}{\Gamma \vdash B \wedge C}\,\wedge_I}{\Gamma \vdash A \wedge (B \wedge C)}\,\wedge_I$$

Pourtant, on constate que la nouvelle dérivation peut être plus haute que l'ancienne car on rajoute une étape de déduction : la règle $\wedge\wedge_I$ n'est donc pas fortement admissible.

Définition 1.15 (règle inversible)
Soit $\mathfrak{C}$ un calcul de dérivation et soit $\mathcal{R}$ une règle de schéma :

$$\frac{\mathcal{H}}{\mathcal{C}}\ \mathcal{R}$$

On dit que la règle $\mathcal{R}$ est inversible dans $\mathfrak{C}$ si, et seulement si, la règle inverse $\mathcal{R}^{-1}$ de schéma

$$\frac{\mathcal{C}}{\mathcal{H}}\ \mathcal{R}^{-1}$$

est admissible dans $\mathfrak{C}$, ce qui équivaut à dire que la dérivabilité de $\mathcal{C}$ et celle de $\mathcal{H}$ sont équivalentes.

Dans le cas d'une règle à plusieurs prémisses de schéma

$$\frac{\mathcal{H}_1 \quad \dots \quad \mathcal{H}_n}{\mathcal{C}}\ \mathcal{R}$$

on parle de règle partiellement inversible par rapport à l'hypothèse $\mathcal{H}_k$ si la règle

$$\frac{\mathcal{C}}{\mathcal{H}_k}\ \mathcal{R}_k^{-1}$$

est admissible dans $\mathfrak{C}$. On parlera d'inversibilité totale dans le cas où la règle est inversible par rapport à toutes ses hypothèses.

La notion de règle inversible est capitale pour la recherche de dérivations d'un jugement. En effet, si l'on considère l'ensemble des dérivations d'un jugement qui est la conclusion d'une règle inversible, alors on sait que deux cas sont possibles : soit cet ensemble est vide et le jugement n'est pas dérivable, soit il contient une dérivation terminant par cette règle. Dans le cas d'un calcul de séquents, on parle d'équiprouvabilité de l'hypothèse et de la conclusion. L'utilisation de cette propriété permet d'utiliser en priorité les règles inversibles pour rechercher la dérivation d'un jugement. Cette propriété sera utilisée dans la section 3.4.1 pour optimiser la recherche de preuve.

1.2.2 Calcul de séquents

Les systèmes d'inférence sont utilisés principalement dans deux domaines : la sémantique des langages de programmation et la logique formelle. Dans ce dernier domaine, la forme de jugement la plus fréquemment utilisée est le séquent, qui a été inventé par Gentzen [Gen34].

Définition 1.16 (séquent)
Un séquent est un couple noté $\Gamma \vdash \Delta$ formé de deux listes finies de formules Γ et Δ. Γ est appelé contexte du séquent et Δ conclusion. Les formules de Γ sont appelées hypothèses du séquent. Un calcul de dérivation dont les jugements sont des séquents est appelé calcul de séquents. Un calcul de séquents est dit mono-successeur si les règles manipulent des séquents dont la conclusion contient au plus une formule ; sinon, il est dit multi-successeurs.

Gentzen a mis au point le calcul de séquents LK pour la logique classique du premier ordre, que nous présentons dans la figure 1.2. Parmi les règles de LK, on distingue les règles X_L et X_R d'échange, les règles C_L et C_R de contraction et les règles W_L et W_R d'affaiblissement, qui sont ce que l'on appelle des règles structurelles. Ces règles ont un rôle plus mécanique que les autres car elle ne précisent pas le rôle des connecteurs logiques.

Les règles d'échange permettent de réordonner le contexte à volonté. Souvent, on considère que le contexte est un multi-ensemble et on omet d'énoncer ces règles. Les règles de contraction signifient qu'une formule peut être répliquée à volonté. Les règles d'affaiblissement sont souvent omises car on utilise une formulation plus générale de la règle d'axiome :

$$\frac{}{\Gamma, A \vdash A, \Delta}\ Ax$$

Avec cette règle généralisée, on peut se passer de l'affaiblissement en ajoutant les nouvelles formules dans toutes les prémisses jusqu'aux règles Ax ou $\bot_L$.

DÉFINITION 1.17 (CONDITION DE VARIABLE FRAÎCHE)
*Dans une règle de calcul de séquents, une variable satisfait la condition de variable fraîche (*eigenvariable condition*) si, et seulement si, elle n'apparaît pas libre dans la conclusion de la règle.*

Dans les règles $\forall_R$ et $\exists_L$, la variable x doit satisfaire la condition de variable fraîche, cela permet d'assurer que, dans le cas de $\forall_R$, on a bien prouvé Px pour x quelconque (on ne dispose d'aucune information particulière concernant x).

Gentzen a établi que le calcul LK est correct et complet vis-à-vis de la sémantique booléenne de la logique classique du premier ordre. Il a également montré que le système LK était équivalent au système LK privé de la règle de coupure Cut, établissant ainsi la propriété d'élimination des coupures en logique classique du premier ordre. Le calcul est bien classique malgré l'absence de mention explicite à l'axiome du tiers exclus, en effet celui-ci peut être dérivé dans LK :

$$\dfrac{\dfrac{\dfrac{}{A \vdash A, \bot}\ Ax}{\vdash A, (A \to \bot)}\ \to_R}{\vdash A \vee (A \to \bot)}\ \vee_R$$

1.2.3 Calculs de séquents intuitionnistes

Gentzen s'aperçut que si l'on transformait le calcul LK pour en faire un calcul mono-successeur LJ, on obtenait un système de déduction correct et complet pour la logique intuitionniste [Hey56]. Pour cela, il fallait supprimer les règles W_R, X_R et C_R et diviser la règle $\vee_R$ en deux :

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee_{R_1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee_{R_2}$$

$$\frac{}{P \vdash P}\, Ax \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\, Cut$$

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

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\, C_L \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\, C_R$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\, W_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\, W_R$$

$$\frac{\Gamma, B \vdash \Delta \quad \Gamma \vdash A, \Delta}{\Gamma, A{\to}B \vdash \Delta}\, {\to}_L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A{\to}B, \Delta}\, {\to}_R$$

$$\frac{}{\Gamma, \bot \vdash \Delta}\, \bot_L$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\, \wedge_L \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\, \wedge_R$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\, \vee_L \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\, \vee_R$$

$$\frac{\Gamma, A\{t/x\} \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\, \forall_L \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x.A, \Delta}\, \forall^*_R$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\, \exists^*_L \qquad \frac{\Gamma \vdash A\{t/x\}, \Delta}{\Gamma \vdash \exists x.A, \Delta}\, \exists_R$$

Fig. 1.2 – Le système LK

Le défaut de cette approche mono-successeur est qu'elle oblige à choisir immédiatement entre A et B lors de l'utilisation de la règle $\vee_R$, ce qui peut conduire à revenir sur nos pas (*backtracking*) en cas d'échec, alors que le calcul multi-successeurs a l'avantage de permettre d'explorer « en parallèle » les deux possibilités. Pour combiner logique intuitionniste et calcul de séquents multi-successeurs, Dragalin [Dra87] proposa une modification de la règle $R\rightarrow$ dans son système $GHPC$, qui efface tout Δ dans la prémisse :

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B, \Delta}\ R\rightarrow$$

De fait, cette modification empêche que l'on se serve de l'hypothèse A pour déduire une conclusion de Δ, or c'est précisément ce que l'on fait dans la dérivation du tiers exclus (voir plus haut). Ainsi, les formules de Δ se retrouvent isolées, et on peut extraire d'une dérivation $GHPC$ une dérivation LJ équivalente.

1.3 Théorie des types et logique du premier ordre

1.3.1 Calcul des Constructions Inductives

Le Calcul des Constructions Inductives est un λ-calcul typé avec des types dépendants, étendu par des types inductifs munis d'une construction de filtrage et de récurrence primitive. Coq est un interpréteur et un vérificateur de types pour ce langage. L'interprétation des programmes de ce calcul comme des preuves de propriétés logiques fait de Coq un assistant de preuve interactif : la relation de typage $\Gamma \vdash t : T$ peut se lire aussi bien comme *« d'après les types de variables de* Γ*,* t *est un élément de l'ensemble* T *»* que comme *« des hypothèses présentes dans* Γ*, on peut déduire que* t *est une preuve de* T *»*. Avec le lieur de définition de fonction λ, l'objet le plus important du langage de Coq est le lieur $\forall$ qui construit un produit dépendant, que l'on peut lire aussi bien comme un type de fonction dépendant que comme une proposition logique universellement quantifiée. Nous écrirons le terme $\forall x : A.B$ sous la forme $A \rightarrow B$ chaque fois que x n'apparaît pas libre dans B. Dans ce cas, $A \rightarrow B$ représente l'espace de fonctions du type A vers le type B ou l'implication entre les propositions A et B. Le détail des règles de typage pour le Calcul des Constructions est donné dans la figure 1.3. Les types inductifs sont présentés plus bas.

Afin d'établir une séparation claire entre les objets informatifs (types de données et fonctions) et les objets non-informatifs (preuves de propositions), les types utilisés comme des types de données sont typés avec la sorte (type de type) Set et les propositions logiques le sont avec la sorte Prop. Pour assurer la cohérence de cette séparation, un ensemble de restrictions sur la règle de filtrage empêche toute fuite d'information entre Prop et Set. En particulier, il est impossible de construire des objets dont le type est dans Set par analyse de cas sur une preuve.

Pour permettre de parler d'objets plus complexes, les sortes Prop et Set sont placées dans une sorte notée Type_0. Cette sorte est elle-même placée dans Type_1, etc. On dispose ainsi d'une infinité de sortes Type_i, où i est un entier naturel. Dans le système Coq, l'indice

$$\mathcal{S} = \{\mathsf{Set}; \mathsf{Prop}; \mathsf{Type}_i | i \in \mathbb{N}\}$$

$$\frac{}{\vdash_{BF} \emptyset}\ \textsc{cont}_1 \qquad \frac{\vdash_{BF} \Gamma \quad \Gamma \vdash t : s}{\vdash_{BF} \Gamma, x : t}\ \textsc{cont}_2 \quad s \in \mathcal{S}, x \notin \Gamma$$

$$\frac{}{\Gamma \vdash s : \mathsf{Type}_j}\ \textsc{Sort} \quad s \in \{\mathsf{Set}; \mathsf{Prop}; \mathsf{Type}_i | i < j\}$$

$$\frac{\Gamma \vdash t : s}{\Gamma \vdash t : \mathsf{Type}_j}\ \textsc{Cumul} \quad s \in \{\mathsf{Set}; \mathsf{Prop}; \mathsf{Type}_i | i < j\}$$

$$\frac{\Gamma \vdash T : s \quad \Gamma, x : T \vdash U : \mathsf{Prop}}{\Gamma \vdash \forall x : T.U : \mathsf{Prop}}\ \textsc{Prod}_1 \quad s \in \mathcal{S}$$

$$\frac{\Gamma \vdash T : s \quad \Gamma, x : T \vdash U : \mathsf{Set}}{\Gamma \vdash \forall x : T.U : \mathsf{Set}}\ \textsc{Prod}_2 \quad s \in \{\mathsf{Set}; \mathsf{Prop}\}$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \forall x : T.U : \mathsf{Type}_k}\ \textsc{Prod}_2 \quad s_1, s_2 \in \{\mathsf{Set}; \mathsf{Prop}; \mathsf{Type}_i | i \leq k\}$$

$$\frac{\vdash_{BF} \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}\ \textsc{Var} \qquad \frac{\Gamma \vdash u : \forall x : T.U \quad \Gamma \vdash t : T}{\Gamma \vdash (u\, t) : U\{t/x\}}\ \textsc{App}$$

$$\frac{\Gamma, x : T \vdash u : U \quad \Gamma \vdash \forall x : T.U : s}{\Gamma \vdash \lambda x : T.u : \forall x : T.U}\ \textsc{Abs} \quad s \in \mathcal{S}$$

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s \quad T =_\beta T'}{\Gamma \vdash t : T'}\ \textsc{Conv} \quad s \in \mathcal{S}$$

Fig. 1.3 – Règles de typage du Calcul des Constructions

des sortes Type est omis, et le système s'assure à chaque opération que l'on peut donner à chaque occurrence de Type un indice de manière à rendre typables les termes manipulés. La règle de cumulativité CUMUL indique que les sortes plus petites sont inclues dans les sortes plus grandes. Dans le système Coq, cela implique que si un objet a pour type la sorte Prop ou Set, il a également pour type la sorte Type.

Une particularité de la théorie des types de Coq qui est fondamentale pour notre travail est la règle de conversion CONV : si $\Gamma \vdash t : T$ alors $\Gamma \vdash t : T'$ pour tout type T' équivalent à T, c'est-à-dire ayant la même forme normale par réduction selon la théorie des types. Cette dernière propriété est due au fait que les règles de réduction du langage de Coq en forment un système confluent et fortement normalisant. En particulier, tout terme t est, de façon démontrable dans la théorie de Coq, égal à sa forme normale $t_\downarrow$: nous utilisons l'axiome de réflexivité `refl_equal` pour construire la preuve (`refl_equal` t) : $t = t$ et grâce à la règle de conversion, cette preuve est aussi une preuve de $t = t_\downarrow$ (ces deux propositions sont équivalentes car elles ont la même forme normale $t_\downarrow = t_\downarrow$). Par exemple, si l'on considère le terme 2×2 qui se réduit en 4, (`refl_equal` 4) est à la fois une preuve de $4 = 4$ et une preuve de $2 \times 2 = 4$.

1.3.2 Représentation de la logique du premier ordre dans le Calcul des Constructions Inductives

Pour donner aux utilisateurs de Coq une base de travail commune, la librairie standard fournit aux utilisateurs de Coq un ensemble d'objets prédéfinis et de notations permettant d'écrire des propositions logiques du premier ordre.

Tout d'abord les sortes de la logique du premier ordre sont représentées en Coq par des types de données, de sorte Set. Les prédicats sont des objets de type $\tau_1 \to \ldots \to \tau_n \to$ Prop, où $\tau_1, \ldots, \tau_n$ sont des types dans Set. Les fonctions du premier ordre sont des objets Coq de type $\tau_1 \to \ldots \to \tau_n \to \tau$, où $\tau_1, \ldots, \tau_n, \tau$ sont des types dans Set.

Par exemple, les entiers de Peano sont définis par un type `nat`, la fonction d'addition a le type `nat`→`nat`→`nat`, et le prédicat de parité a pour type `nat`→Prop.

Afin de construire des formules du premier ordre, on dispose déjà du produit dépendant et de sa version non-dépendante, pour représenter le quantificateur universel $\forall$ et l'implication $\to$. Pour représenter la disjonction $\vee$ dans Coq, la bibliothèque standard définit un type inductif correspondant à l'union disjointe :

```
Inductive or (A B:Prop) : Prop :=
  | or_introl : A → A ∨ B
  | or_intror : B → A ∨ B
where "A ∨ B" := (or A B) : type_scope.
```

Cette déclaration définit en fait quatre objets Coq différents :
- Un type paramétrique `or` de type Prop→Prop→Prop
- Un constructeur `or_introl` de type $\forall A, B :$ Prop.$A \to A \vee B$

- Un constructeur `or_intror` de type $\forall A, B : \mathsf{Prop}. B \rightarrow A \vee B$
- Un principe d'élimination `or_ind` de type :

$$\forall A, B, P : \mathsf{Prop}. (A \rightarrow P) \rightarrow (B \rightarrow P) \rightarrow A \vee B \rightarrow P$$

Cette déclaration permet aussi d'ajouter du « sucre syntaxique » au langage de Coq, en définissant la notation `A ∨ B` comme équivalente à `(or A B)`. Le principe d'élimination `or_ind` est la formulation au second ordre de la sémantique de la disjonction `or` : il affirme que si chacune des propriétés `A` et `B` implique une troisième propriété P, alors la disjonction de ces propriétés implique P également.

Exemple 1.18 (commutativité de $\vee$)
Dans le Calcul des Constructions Inductives, la commutativité de $\vee$ dont l'énoncé est $\forall A, B : \mathsf{Prop}. A \vee B \rightarrow B \vee A$ est prouvée par le terme suivant :

```
λ (A,B:Prop) (H:A ∨ B),
        (or_ind A B (B ∨ A) (or_intror B A) (or_introl B A) H)
```

Dans le Calcul des Constructions Inductives, les principes d'élimination sont en fait définis à l'aide d'une construction syntaxique appelée filtrage, qui est inspirée de celle définie dans le langage Objective Caml. Ainsi, la preuve de commutativité de $\vee$ s'écrit sous la forme d'un filtrage sur l'hypothèse $A \vee B$.

```
λ (A,B:Prop) (H:A ∨ B),
  match H with
    | or_introl HA => or_intror B A HA
    | or_intror HB => or_introl B A HB
  end.
```

La conjonction de deux propriétés est définie de manière analogue par le type inductif `and` dont le constructeur est `conj` :

```
Inductive and (A B:Prop) : Prop :=
    conj : A → B → A ∧ B
where "A ∧ B" := (and A B) : type_scope.
```

La proposition absurde est définie comme un type inductif sans constructeur, et la proposition triviale comme un type inductif ayant un unique constructeur constant `I`.

```
Inductive True : Prop :=
    I : True.
Inductive False : Prop :=.
```

Pour représenter le quantificateur existentiel, on utilise un type inductif de paire dépendante : le type du second membre (la propriété `P x`) dépend de la valeur du premier membre (l'objet `x` témoin de l'existence).

```
Inductive ex (A:Type) (P:A -> Prop) : Prop :=
    ex_intro : forall x:A, P x -> ex A P.
```

Pour compléter cet ensemble de définitions, la bibliothèque standard fournit également des définitions pour la négation et l'équivalence logique :

```
Definition not (A:Prop) := A → False.
Definition iff (A B:Prop) := (A → B) ∧ (B → A).
```

Traitement de l'égalité

L'égalité Coq est définie par un type inductif ayant un paramètre. L'unique constructeur de ce type est l'axiome de réflexivité, et le principe d'élimination est l'axiome de remplacement :

```
Inductive eq (A:Type) (x:A) : A -> Prop :=
    refl_equal : x = x :>A
where "x = y :> A" := (eq A x y) : type_scope.

Notation "x = y" := (x = y :>_) : type_scope.
```

La déclaration finale permet d'omettre le type des membres de l'égalité. Le principe d'élimination de l'égalité est le suivant :

```
eq_ind : ∀ (A : Type) (x : A) (P : A → Prop),
       P x → forall y : A, x = y → P y
```

Il exprime le principe de remplacement : deux objets égaux ont les mêmes propriétés.

Chapitre 2

Égalités closes avec constructeurs

Parmi les problèmes posés par la théorie de l'égalité, le plus important est le problème du mot : une égalité est-elle conséquence d'une conjonction d'égalités ? En 1947, Post et Markov [Pos47, Mar47] démontrèrent que ce problème était indécidable. À l'inverse, on peut décider si une égalité entre termes clos est conséquence d'une conjonction finie d'égalités entre termes clos. Ce dernier problème s'appelle clôture de congruence et sa décidabilité fut démontrée par Nelson et Oppen [NO80], ainsi que par Downey, Sethi et Tarjan [DST80]. Les algorithmes proposés dans ces deux articles sont d'une complexité quasi-linéaire. Nelson et Oppen, ainsi que Shostak [Sho82, Sho84] proposèrent des méthodes pour étendre ce cas clos avec des théories équationnelles particulières : théorie des listes, des n-uplets ...

En général, les procédures de semi-décision sur des égalités font appel à des approches syntaxiques (remplacement, réécriture ...). L'algorithme utilisé pour la clôture de congruence fait donc figure d'exception puisque son principe est de construire directement un ensemble de classes d'équivalence de termes. Le lien entre clôture de congruence et méthodes syntaxiques fut établi par la notion de clôture abstraite de congruence [BT00, BTV03].

La simplicité et l'efficacité de l'algorithme en font un candidat intéressant pour l'implantation d'une tactique en Coq. En outre, on souhaite enrichir la procédure de décision afin qu'elle puisse prendre en compte les propriétés spécifiques des constructeurs des types inductifs et co-inductifs de Coq :

- propriété d'injectivité : $C\,x_1\,\ldots\,x_n = C\,y_1\,\ldots\,y_n \Rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n$
- propriété de discrimination : $C\,x_1\,\ldots\,x_n \neq D\,y_1\,\ldots\,y_p$

L'assistant de preuve Coq disposait déjà d'une part des tactiques `injection` et `discriminate` qui déduisent les conséquences immédiates d'une égalité entre termes avec constructeurs, et d'autre part de la tactique `autorewrite` permettant de normaliser un terme vis-à-vis d'un ensemble d'égalités orientées. Ces tactiques ne coopèrent pas du tout, une manière naturelle de les faire coopérer serait une tactique de clôture par congruence intégrant la théorie des constructeurs.

Lors de mon stage de DEA en 2001, j'ai prouvé dans Coq la correction de l'algorithme de clôture de congruence et j'ai mis au point un système d'annotation de la structure de données permettant d'en tirer une preuve d'égalité. Cette méthode fut implantée dans Coq en 2002 sous la forme de la tactique `congruence`. Cette méthode de production de preuve

fut découverte indépendamment et étudiée en détail par Nieuwenhuis et Oliveras [NO05].

En 2003, j'ai produit une version étendue de la tactique `congruence` supportant la théorie des constructeurs. Aucune base théorique n'avait été décrite sur cette extension et ce chapitre comble cette lacune. Notre propos est donc de décrire une procédure de décision capable de traiter complètement la théorie des constructeurs par une extension de la clôture de congruence. Ceci est possible grâce à un théorème montrant que le problème peut être ramené à un ensemble fini de termes. De plus, la logique d'ordre supérieur de Coq nous pousse à résoudre ce problème dans le cadre d'une théorie simplement typée avec égalité entre objets fonctionnels.

Dans l'ensemble de ce chapitre, nous manipulerons uniquement des termes clos (sans variables), que nous appellerons simplement termes par souci de clarté. De même, nous parlerons d'algèbre de termes pour désigner les algèbres de termes clos.

2.1 Le problème de clôture de congruence

Le langage de la théorie de l'égalité utilise un prédicat binaire distingué $\approx$ que l'on suppose bien sorté, c'est-à-dire qu'il met en relation uniquement des termes de même sorte. Dans notre langage de formules, ce prédicat polymorphe est une notation pour désigner un ensemble de prédicats monomorphes $(\approx_s)_{s\in\mathcal{S}}$.

La définition du problème de clôture de congruence nécessite tout d'abord que l'on définisse ce qu'est une congruence.

DÉFINITION 2.1 (CONGRUENCE)
Une relation bien sortée $\approx$ sur une algèbre de termes du premier ordre est appelée congruence si, et seulement si elle vérifie deux conditions :

1. *$\approx$ est une relation d'équivalence, c'est-à-dire qu'elle satisfait les règles d'inférence suivantes :*

$$\frac{}{t \approx t}\ \text{REFL} \qquad \frac{s \approx t}{t \approx s}\ \text{SYM} \qquad \frac{s \approx u \qquad u \approx t}{s \approx u}\ \text{TRANS}$$

2. *$\approx$ est une relation passant au contexte, ce qui signifie qu'elle satisfait la règle d'inférence suivante pour tout symbole f, où n est l'arité du symbole f.*

$$\frac{s_1 \approx t_1 \qquad \dots \qquad s_n \approx t_n}{f(s_1,\dots,s_n) \approx f(t_1,\dots,t_n)}\ \text{CONGR}_f$$

Les congruences possèdent un certain nombre de propriétés, elles sont notamment stables par intersection :

LEMME 2.2
Soient $(\approx_l), l \in L$ une famille de relations indexées par un ensemble quelconque L. Soit $\approx$ leur intersection, définie par $u \approx v \Leftrightarrow \forall l \in L, u \approx_l v$. Si chacune des relations $\approx_l$ est une congruence, alors $\approx$ est une congruence.

Démonstration : Immédiate d'après la définition. □

De ce résultat on déduit que pour toute relation binaire R sur une algèbre de termes, il existe une congruence minimale $\approx_R$ contenant R, qui est l'intersection de toutes les congruences contenant R (il en existe toujours au moins une : la relation contenant tous les couples de termes de même sorte).

Le problème de clôture de congruence peut alors s'énoncer des deux manières suivantes :

- L'ensemble fini de propositions logiques $\{s_1 \approx t_1, \ldots, s_n \approx t_n, u \not\approx v\}$ est-il satisfaisable dans la théorie de l'égalité ?
- Existe-t-il une congruence satisfaisant les axiomes $s_1 \approx t_1, \ldots, s_n \approx t_n$ et $u \not\approx v$?

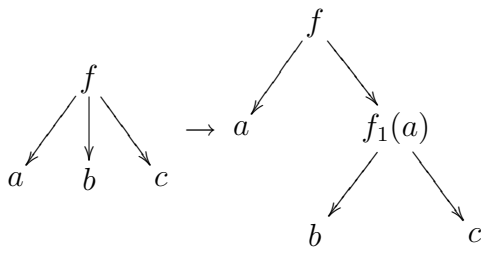
Comme le suggèrent ces formulations du problème, on peut procéder immédiatement à une généralisation du problème dans laquelle on peut avoir plusieurs propositions de la forme $u \not\approx v$, la question étant alors de déterminer si l'une d'entre elles est contradictoire avec les axiomes de la forme $s \approx t$.

L'équivalence de la seconde formulation avec la première repose sur deux arguments : d'une part, si l'ensemble des propositions est satisfaisable par une interprétation I, alors la relation $\approx$ définie par $s \approx t \Leftrightarrow I(s) = I(t)$ est une congruence. L'égalité d'interprétation donne une congruence satisfaisant les axiomes $s_i \approx t_i$ mais pas $u \approx v$. D'autre part, s'il existe une congruence satisfaisant les axiomes $s_i \approx t_i$ mais pas $u \approx v$, on peut alors construire une interprétation qui envoie chaque terme sur sa classe d'équivalence. La propriété de congruence montre que l'interprétation des symboles de fonction est bien définie et que cette interprétation satisfait les axiomes $s_i \approx t_i$ et pas $u \approx v$.

La remarque fondamentale permettant de décider le problème est la suivante : si T est l'ensemble des termes et sous-termes apparaissant dans les axiomes et la conjecture, alors toute dérivation de celle-ci à partir des axiomes, à l'aide des règles REFL, SYM, TRANS, CONGR, peut être transformée en une dérivation dans laquelle n'apparaissent que des termes dans T. De manière équivalente, toute interprétation de l'ensemble de termes T satisfaisant les équations $s \approx t$ et les diséquations $u \not\approx v$ peut être étendue à l'algèbre de termes tout entière.

Nelson et Oppen [NO79, NO80], ainsi que Downey, Sethi et Tarjan [DST80], se basent alors sur cette propriété pour proposer des algorithmes utilisant la représentation de classes d'équivalence de termes par une forêt d'arbres (structure UNION-FIND), afin d'obtenir une complexité optimale.

Pour simplifier l'algorithme, les deux articles proposent une représentation arborescente des termes, et décrivent comment transformer ces arbres en ajoutant des noeuds pour qu'ils deviennent binaires :



Ici, nous souhaitons traiter le cas de termes d'ordre supérieur, c'est pourquoi nous allons utiliser un encodage différent, qui correspond à la vision curryfiée des fonctions à plusieurs arguments. Ce type de représentation est particulièrement adapté pour les formalismes d'ordre supérieur car il permet de représenter des fonctions partiellement appliquées.

2.1.1 Algèbre de termes simplement typés

Pour introduire la représentation curryfiée de nos termes, nous allons introduire un type particulier de signatures du premier ordre : les signatures simplement typées.

DÉFINITION 2.3 (TERMES SIMPLEMENT TYPÉS)
Une algèbre de termes simplement typés est définie par une signature $(\mathcal{S}, \Sigma)$ *telle que :*

- $\mathcal{S} = \bigcup_{n\in\mathbb{N}} \mathcal{S}_n$, *où* $\mathcal{S}_{n+1} = S_n \cup \{s \to s' | (s, s') \in \mathcal{S}_n \times \mathcal{S}_n\}$ *et* $\mathcal{S}_0$ *est un ensemble de sortes* $\alpha_1, \ldots, \alpha_n$ *dites sortes de base. Les sortes de* $\mathcal{S} \setminus \mathcal{S}_0$ *sont appelées sortes fonctionnelles.*
- Σ *ne contient que des constantes et un ensemble de symboles de fonctions d'arité 2 appelés applications, de la forme* $@_{s,s'} : (s \to s', s) \to s'$.

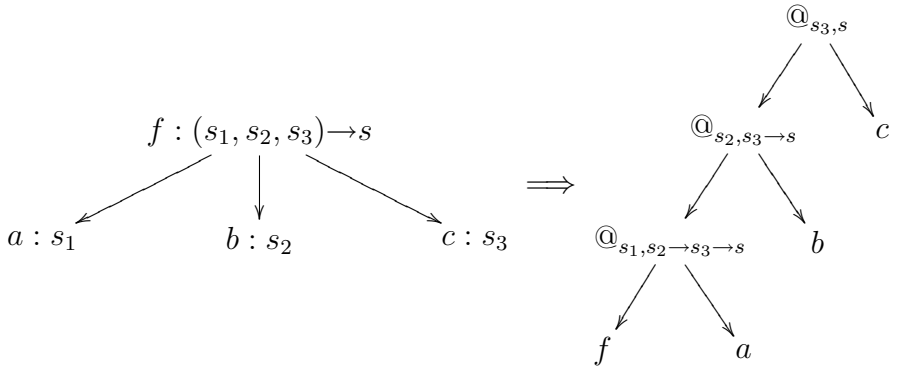
Dans la pratique, l'ensemble des sortes nécessaires pour construire les termes à partir d'un nombre fini de constantes est fini, de même que l'ensemble des symboles d'application. Par abus de notation, nous n'écrirons pas les indices des symboles d'application, et dans la suite, nous écrirons parfois $(f\, x\, y)$ au lieu de $@(@(f, x), y)$ pour simplifier.

A toute signature du premier ordre on peut faire correspondre une signature simplement typée par l'opération dite de curryfication que nous décrivons ainsi :

DÉFINITION 2.4 (TERMES CURRYFIÉS)
Soit $(\mathcal{S}, \Sigma)$ *une signature du premier ordre. On définit la signature curryfiée correspondante par* $(\hat{\mathcal{S}}, \hat{\Sigma})$, *où* $\hat{\mathcal{S}}$ *est l'ensemble des sortes simples engendrées en utilisant* $\mathcal{S}$ *pour les sortes de base, et où* $\hat{\Sigma}$ *contient, pour chaque symbole de fonction* $f : (s_1, \ldots, s_n) \to s \in \Sigma$, *un symbole de constante* $\hat{f} : s_1 \to (\ldots \to (s_n \to s))$, *ainsi que les symboles d'applications relatifs à la sorte de* f : $@_{s_1, s_2 \to \ldots \to s}, \ldots, @_{s_n, s}$.

Dans la suite de ce chapitre, nous supposerons le symbole $\to$ associatif à droite et omettrons les parenthèses en conséquence.

Chaque terme t de sorte s dans l'algèbre de départ peut être représenté par un terme curryfié $\hat{t}$ de même sorte dans l'algèbre engendrée par Σ_c :



La réciproque n'est pas vraie, y compris lorsque la signature est du premiere ordre, puisque l'on peut construire des termes curryfiés représentant des fonctions partiellement appliquées. Cependant, tout terme curryfié ayant pour sorte une sorte de $\mathcal{S}$, dite sorte de base, est le résultat de la curryfication d'un unique terme t de l'algèbre engendrée par Σ.

Exemple 2.5
La signature du premier ordre correspondant à l'arithmétique de Peano est composée de l'ensemble de sortes $\mathcal{S}_0 = \{\mathbb{N}\}$ *et de la signature :*

$$\begin{array}{l}\{\,\mathbf{0} : () \to \mathbb{N}\,;\\ \quad \mathrm{S} : (\mathbb{N}) \to \mathbb{N}\,;\\ \quad + : (\mathbb{N}, \mathbb{N}) \to \mathbb{N}\,;\\ \quad \times : (\mathbb{N}, \mathbb{N}) \to \mathbb{N}\}\end{array}$$

L'ensemble de sortes de la signature curryfiée correspondante est :

$$\mathcal{S} = \{\mathbb{N}\,; \mathbb{N} \to \mathbb{N}\,; \mathbb{N} \to \mathbb{N} \to \mathbb{N}\,\}$$

L'ensemble de symboles curryfiés est alors :

$$\begin{array}{rl}\{\quad \hat{\mathbf{0}} & : \mathbb{N}\,;\\ \quad \hat{\mathrm{S}} & : \mathbb{N} \to \mathbb{N}\,;\\ \quad \hat{+} & : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\,;\\ \quad \hat{\times} & : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\,;\\ \quad @_{\mathbb{N},\mathbb{N}} & : (\mathbb{N} \to \mathbb{N}, \mathbb{N}) \to \mathbb{N}\,;\\ \quad @_{\mathbb{N},\mathbb{N}\to\mathbb{N}} & : (\mathbb{N} \to \mathbb{N} \to \mathbb{N}, \mathbb{N}) \to \mathbb{N} \to \mathbb{N}\}\end{array}$$

On remarque que l'on peut exprimer, à l'aide de la signature curryfiée, des égalités sur les fonctions telles que $@(+, @(\mathrm{S}, \mathbf{0})) \approx S$.

On pourrait penser que le fait de pouvoir exprimer plus de propriétés dans le système curryfié rend la théorie de l'égalité plus puissante dans celui-ci que le système de départ. Il n'en est rien, comme le démontre le résultat suivant.

Lemme 2.6 (conservativité de la curryfication)
La curryfication est conservative par rapport à la clôture de congruence, c'est-à-dire que toute dérivation de $\hat{u} \approx \hat{v}$ *à partir d'axiomes* $\hat{s}_1 \approx \hat{t}_1, \ldots, \hat{s}_n \approx \hat{t}_n$ *dans le système curryfié peut être transformée en une dérivation dans le système de départ et réciproquement.*

Démonstration : Tout d'abord, on remarque que l'on peut transformer la dérivation de telle sorte que la prémisse gauche de chaque application de la règle $\textsc{congr}_@$ ne se termine ni par la règle sym, ni par la règle trans. Cela est possible par utilisation répétée des transformations suivantes :

$$\dfrac{\dfrac{g \approx f}{f \approx g}\,\textsc{sym} \qquad x \approx y}{(f\,x) \approx (g\,y)}\,\textsc{congr}_@ \quad\Longrightarrow\quad \dfrac{g \approx f \qquad \dfrac{x \approx y}{y \approx x}\,\textsc{sym}}{\dfrac{(g\,y) \approx (f\,x)}{(f\,x) \approx (g\,y)}\,\textsc{sym}}\,\textsc{congr}_@$$

$$\dfrac{\dfrac{f \approx g \quad g \approx h}{f \approx h}\,\text{TRANS} \qquad x \approx y}{(f\,x) \approx (h\,y)}\,\text{CONGR}_@$$

$$\Downarrow$$

$$\dfrac{\dfrac{f \approx g \quad \dfrac{}{x \approx x}\,\text{REFL}}{(f\,x) \approx (g\,x)}\,\text{CONGR}_@ \qquad \dfrac{g \approx h \quad x \approx y}{(g\,x) \approx (h\,y)}\,\text{CONGR}_@}{(f\,x) \approx (h\,y)}\,\text{TRANS}$$

On appelle hauteur principale d'un arbre de dérivation le nombre maximal d'étapes de sa racine à la plus profonde de ses feuilles, en excluant les branches qui passent par la prémisse de droite d'une règle de congruence. L'application répétée des transformations décrites plus haut termine quelle que soit la stratégie utilisée car elles ne modifient pas la hauteur principale des arbres, excepté pour la règle $\text{CONGR}_@$ concernée par la transformation : la prémisse de gauche de la (des) nouvelle(s) règle(s) engendrée(s) a une hauteur strictement plus petite que celle de la règle de départ. Les transformations font donc décroître pour l'ordre multi-ensemble la hauteur principale des prémisses gauches des règles CONGR dans l'arbre.

On montre ensuite par récurrence structurelle que tout arbre de dérivation pour l'algèbre curryfiée dont la conclusion et les axiomes sont des égalités dans une sorte de base est transformable en un arbre équivalent dans l'algèbre de base.

- Si l'arbre est réduit à un axiome ou à une application de la règle de réflexivité, alors, comme les termes sont d'une sorte de base, on peut utiliser la version non curryfiée de ces termes pour reconstruire une arbre réduit, soit à un axiome, soit à une application de la règle de récursivité.
- Si l'arbre se termine par une application de la règle de symétrie ou de transitivité, alors, comme les termes de la conclusion sont d'une sorte de base, ceux des prémisses le sont également, et l'on peut appliquer récursivement la transformation aux prémisses et utiliser la version non curryfiée des règles de transitivité et de symétrie.
- Si l'arbre se termine par une règle de congruence, alors si l'on suit les prémisses gauches, on trouve une séquence de règles de congruence éventuellement vide et terminée par une règle de réflexivité : par hypothèse, une prémisse gauche ne peut être un axiome car les termes n'y sont pas dans une sorte de base. On est donc dans la situation suivante :

$$\dfrac{\dfrac{\dfrac{}{(f\,s_1\,\ldots\,s_k) \approx (f\,s_1\,\ldots\,s_k)}\,\text{REFL} \qquad \vdots\ s_{k+1} \approx s'_{k+1}}{(f\,s_1\,\ldots\,s_k\,s_{k+1}) \approx (f\,s_1\,\ldots\,s_k\,s'_{k+1})}\,\text{CONGR}_@ \qquad \ddots \qquad \vdots\ s_n \approx s'_n}{(f\,s_1\,\ldots\,s_k\,s_{k+1}\,\ldots\,s_n) \approx (f\,s_1\,\ldots\,s_k\,s'_{k+1}\,\ldots\,s'_n)}\,\text{CONGR}_@$$

Comme les termes $s_1, \ldots, s_n, s'_k, \ldots s'_n$ sont dans une sorte de base, on peut construire une dérivation non curryfiée de $s_i \approx s_i$ par réflexivité pour $1 \leq i < k$, et de $s_i \approx s'_i$ par hypothèse de récurrence pour $k \leq i \leq n$, ce qui donne :

$$\dfrac{\dfrac{}{s_1 \approx s_1}\,\text{REFL} \quad \ldots \quad \dfrac{}{s_k \approx s_k}\,\text{REFL} \quad \begin{array}{c}\vdots\\ s_{k+1} \approx s'_{k+1}\end{array} \quad \ldots \quad \begin{array}{c}\vdots\\ s_n \approx s'_n\end{array}}{f(s_1, \ldots, s_k, s_{k+1}, \ldots, s_n) \approx f(s_1, \ldots, s_k, s'_{k+1}, \ldots, s'_n)}\,\text{CONGR}_f$$

Ceci clôt notre démonstration par récurrence.

Pour la réciproque, on remarque d'abord que toutes les règles sauf CONGR ne sont pas modifiées par le changement de signature. Ensuite, on peut obtenir la règle CONGR_f à partir de la règle $\text{CONGR}_@$ de la manière suivante :

$$\dfrac{a \approx a' \quad b \approx b' \quad c \approx c'}{f(a,b,c) \approx f(a',b',c')}\,\text{CONGR}_f$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f \approx f}\,\text{REFL} \quad a \approx a'}{(f\,a) \approx (f\,a')}\,\text{CONGR}_@ \quad b \approx b'}{(f\,a\,b) \approx (f\,a'\,b')}\,\text{CONGR}_@ \quad c \approx c'}{(f\,a\,b\,c) \approx (f\,a'\,b'\,c')}\,\text{CONGR}_@$$

Ceci nous permet de conclure que la curryfication est conservative par rapport à la clôture de congruence. □

2.2 Extension à la théorie des constructeurs libres

Nous souhaitons maintenant étendre la procédure classique pour qu'elle prenne en compte l'existence de constructeurs de types de données. De tels constructeurs ont pour propriété d'être injectifs, ce qui les rapproche des constructeurs de n-uplets, mais ils ont également la propriété de séparation ou élimination forte : les constructeurs d'un même type de données ont des codomaines disjoints. Ces caractéristiques sont communes aux deux types de constructeurs du système Coq : les constructeurs de types inductifs et de types co-inductifs. Nous ne prétendons pas que ces deux propriétés axiomatisent totalement les constructeurs. En effet, les constructeurs inductifs vérifient également des axiomes d'acyclicité, par exemple, pour les entiers de Peano, la fonction successeur `S` vérifie $x \not\approx \texttt{S}\,x$. Ces propriétés d'acyclicité ont déjà été étudiées, notamment par Oppen dans [Opp78].

D'autres types d'inférences peuvent être effectuées si l'on suppose que les types de données sont totalement engendrés par les constructeurs : en effet, si l'on considére un type fini tel que le type des booléens, qui a deux constructeurs constants T et F, on peut déduire que l'ensemble de propositions $\{g\,T \approx a, g\,F \approx a, g\,c \not\approx a\}$ est insatisfaisable. Nous n'incluons pas non plus ces propriétés dans notre théorie des constructeurs.

DÉFINITION 2.7 (SIGNATURE AVEC CONSTRUCTEURS LIBRES)
Une signature avec constructeurs libres est une signature de types simples à laquelle on ajoute, pour certaines sortes de base, un nombre fini de constantes que l'on appelle constructeurs et qui sont de la forme $C_i^\alpha : \tau_1 \to (\ldots \to (\tau_n \to \alpha))$. C_i^α *est alors un constructeur d'arité* n *de la sorte* α.

La notion de congruence doit alors être étendue pour tenir compte de l'injectivité et de la séparation induite par les constructeurs :

DÉFINITION 2.8 (THÉORIE DES CONSTRUCTEURS)
Une congruence satisfait la théorie des constructeurs libres sur la signature Σ *si, et seulement si elle satisfait la règle d'injectivité :*

$$\frac{C_i^\alpha\, s_1 \ldots s_n \approx C_i^\alpha\, t_1 \ldots t_n}{s_i \approx t_i}\ \text{INJ}_i \text{ si} \begin{cases} C_i^\alpha\, s_1 \ldots s_n : \alpha \\ C_i^\alpha\, t_1 \ldots t_n : \alpha \end{cases}$$

De plus, une telle congruence doit séparer les images directes de constructeurs distincts du même type, c'est-à-dire que l'on doit avoir $C_i^\alpha\, s_1 \ldots s_n \not\approx C_j^\alpha\, t_1, \ldots, t_p$ *dès que* $i \neq j$ *et que les termes* $C_i^\alpha\, s_1 \ldots s_n$ *et* $C_j^\alpha\, t_1, \ldots, t_p$ *sont de sorte* α.

Du fait de l'existence de congruence entre termes dans des sortes fonctionnelles, la notion de terme résultant de l'application d'un constructeur, ou terme inductif, doit être définie relativement à cette congruence.

DÉFINITION 2.9 (TERME INDUCTIF)
Un terme t *est dit inductif pour une congruence* $\approx$ *s'il est* $\approx$*-équivalent à un constructeur* C_i^α *ou à un terme de la forme* $@(f, x)$ *avec* f *inductif.*

Le lemme suivant permet de parler de classe d'équivalence inductive :

LEMME 2.10
Si s *est inductif pour* $\approx$ *et que* $t \approx s$, *alors* t *est inductif.*

Démonstration : Évidente par transitivité de $\approx$. □

Le problème que l'on souhaite résoudre maintenant est la satisfaisabilité d'un ensemble fini d'égalités et de diségalités de termes dans la théorie des constructeurs libres. De manière équivalente, on souhaite savoir s'il existe une congruence satisfaisant la théorie des constructeurs libres ainsi que l'ensemble des égalités et diségalités du problème.

De même que pour la clôture de congruence simple, on constate qu'une famille quelconque de congruences satisfaisant la règle d'injectivité a pour intersection une congruence satisfaisant la règle d'injectivité. Il existe donc, pour toute relation R, une congruence minimale $\approx_R$ satisfaisant la règle d'injectivité, mais pas nécessairement les axiomes de séparation de l'image des constructeurs. On en déduit que s'il existe une congruence contenant R et satisfaisant à la fois les axiomes de séparation et les diségalités du problème, alors cette congruence minimale les satisfait également.

Avant de transformer toutes ces remarques en une procédure effective, il faut vérifier que l'on peut, comme dans le cas de la clôture de congruence, restreindre le problème de décision à un ensemble fini de termes, pour pouvoir le calculer. Malheureusement, l'ensemble des termes et sous-termes du problème ne suffit pas à résoudre la question : supposons que l'on veuille décider si l'ensemble $\{C_1^\alpha \approx f; f \approx C_2^\alpha; a \approx a\}$ est satisfaisable, avec $f : \beta \to \alpha$ et $a : \beta$, alors on remarque que tel n'est pas le cas puisque l'on peut dériver une égalité contredisant les axiomes de séparation, mais uniquement si l'on utilise les termes $C_1^\alpha\, a$ et $C_2^\alpha\, a$ qui n'apparaissent pas dans le problème posé :

$$\dfrac{\dfrac{C_1^\alpha \approx f \quad f \approx C_2^\alpha}{C_1^\alpha \approx C_2^\alpha}\ \text{TRANS} \qquad a \approx a}{C_1^\alpha\, a \approx C_2^\alpha\, a}\ \text{CONGR}_@$$

Pour résoudre cette difficulté, il faut considérer des ensembles de termes clos par application :

Définition 2.11 (clôture par application)
Un ensemble T de termes simplement typés avec constructeurs est dit clos par application pour la congruence $\approx$ si, dans toute classe d'équivalence $T/_\approx$ dont les termes ont une sorte fonctionnelle $\tau \to \tau'$ et qui contient un terme inductif, ou bien il existe un terme t tel que $(tu) \in T$ pour un certain $u \in T$, ou alors il n'existe aucun terme clos de sorte τ (y compris en dehors de T).

Nous pouvons alors définir une version locale de la théorie des constructeurs libres, qui fait appel à une relation secondaire $\rightsquigarrow$ afin de ne pas étendre l'ensemble de termes considérés plus que nécessaire.

Définition 2.12 (théorie locale des constructeurs)
Soit T un ensemble de termes clos par sous-terme et $\approx$ une relation bien sortée sur T, on dit que $\approx$ est une congruence locale sur T si, et seulement si $\approx$ est une relation d'équivalence sur T qui satisfait toute instance de la règle de congruence CONGR$_@$ *où n'apparaissent que des termes de T.*

De plus, on dit qu'une congruence locale $\approx$ satisfait localement la théorie des constructeurs libres si, et seulement si, il existe une relation $\rightsquigarrow$ satisfaisant les règles suivantes :

$$\dfrac{C_i^\alpha \approx f}{C_i^\alpha \rightsquigarrow f}\ \text{PROMOTION} \qquad \dfrac{C_i^\alpha\, t_1 \ldots t_k \rightsquigarrow u \quad u\, t_{k+1} \approx v}{C_i^\alpha\, t_1 \ldots t_k\, t_{k+1} \rightsquigarrow v}\ \text{COMPOSE}$$

$$\dfrac{C_i^\alpha\, s_1 \ldots s_n \rightsquigarrow u \quad C_i^\alpha\, t_1 \ldots t_n \rightsquigarrow u}{s_i \approx t_i}\ \text{INJECTION}$$

et telle qu'il n'y ait aucun terme $t \in T$ tel que $C_i^\alpha\, s_1 \ldots s_n \rightsquigarrow t$ et $C_j^\alpha\, t_1 \ldots t_p \rightsquigarrow t$, où n et p sont les arités respectives de C_i^α et C_j^α.

Ceci nous permet d'énoncer le théorème qui ramène le problème à un ensemble fini de termes :

THÉORÈME 2.13 (RÉDUCTION À UN ENSEMBLE FINI)
Soit E un ensemble fini non vide d'équations et de diséquations et T un ensemble de termes contenant tous les termes et sous-termes apparaissant dans E.

Soit $\approx$ une congruence locale à T satisfaisant E et satisfaisant localement la théorie des constructeurs libres, et telle que T soit clos par application pour $\approx$.

Il existe une congruence $\approx^*$ sur l'ensemble de l'algèbre de termes qui, d'une part, satisfait E et la théorie des constructeurs libres, et qui, d'autre part, est une extension conservative de $\approx$, c'est-à-dire que pour tous termes u et v dans T, $u \approx^* v \Leftrightarrow u \approx v$.

Pour prouver ce résultat, nous allons construire une relation $\approx^*$ à l'aide d'une interprétation des termes de l'algèbre complète compatible avec la relation $\approx$.

Construction de la relation $\approx^*$

Soit S l'ensemble des arbres binaires dont les feuilles sont l'ensemble $T/_\approx$ des classes d'équivalence selon $\approx$, auxquelles on ajoute un élément supplémentaire e_τ pour chaque sorte τ. De plus, on limite S à l'ensemble des arbres bien sortés, c'est-à-dire tels que :

1. e_τ est un arbre bien sorté de sorte τ.
2. Chaque classe d'équivalence pour $\approx$ est bien sortée et sa sorte est la sorte de ses éléments.
3. Si un arbre g est bien sorté de sorte $\tau \to \tau'$ et d est bien sorté, de sorte τ, alors l'arbre (g, d) est bien sorté, de sorte τ'.

On note S_τ le sous-ensemble des arbres de S de la sorte τ.

Un arbre a est inductif s'il n'est pas réduit à une feuille, ou s'il est une classe d'équivalence qui contient un terme inductif.

On définit la fonction $App_{\tau,\tau'} : S_{\tau\to\tau'} \times S_\tau \to S'_\tau$ de la manière suivante : Soient $a \in S_{\tau\to\tau'}$ et $b \in S_\tau$:

- Si $a = e_{\tau\to\tau'}$, alors $App_{\tau,\tau'}(e_{\tau\to\tau'}, b) = e'_\tau$
- Si a n'est pas une feuille, alors $App_{\tau,\tau'}(a, b) = (a, b)$
- Si $a \in T/_\approx$:
  - Si $b \in T/_\approx$ et s'il existe $t_a \in a$ et $t_b \in b$ tels que $@_{\tau,\tau'}(t_a, t_b) \in T$, alors soit c la classe d'équivalence de $@_{\tau,\tau'}(t_a, t_b)$, on pose $App_{\tau,\tau'}(a, b) = c$. Cette définition est non ambiguë car $\approx$ est une congruence.
  - Sinon :
    - Si a est inductif, alors $App_{\tau,\tau'}(a, b) = (a, b)$
    - Sinon $App_{\tau,\tau'}(a, b) = e_{\tau'}$.

Le domaine d'interprétation que nous utilisons est la famille d'ensembles S_τ. L'interprétation des constantes et constructeurs est leur classe d'équivalence dans $T/_\approx$: $I(f) = \bar{f}$. Enfin on pose $I(@_{\tau,\tau'}) = \mathrm{App}_{\tau,\tau'}$.

On peut alors définir la congruence $\approx^*$ par l'équivalence $s \approx^* t \Leftrightarrow I(s) = I(t)$.

Preuve du théorème 2.13

Tout d'abord, $\approx^*$ est bien une congruence car les ensembles S_τ sont disjoints, l'égalité dans le modèle est une relation d'équivalence et comme $I(@(f,x)) = App(I(f), I(x))$, $\approx^*$ satisfait la règle de congruence..

Ensuite, par construction, les termes de T s'interprètent sur leur classe d'équivalence pour $\approx$, donc $\approx^*$ est conservative par rapport à $\approx$ et, en outre, $\approx^*$ satisfait les équations et diséquations de E.

Enfin, il nous faut montrer que $\approx^*$ satisfait la propriété d'injectivité et les axiomes de séparation.

Lemme 2.14 (inversion)
Soient $u, v, w \in T/_\approx$ tels que $\mathrm{App}(u,v) = w$*, alors il existe $s \in u$ et $t \in v$ tels que* $@(s,t) \in w$.

Démonstration : Par définition, si tel n'est pas le cas, ou bien $\mathrm{App}(u,v)$ est un arbre non réduit à une feuille, ou alors $\mathrm{App}(u,v) = e^\tau$. □

Lemme 2.15 (termes inductifs)
Tout terme t de la forme $C_i^\alpha s_1 \dots s_k$ est inductif pour $\approx^*$ et a une interprétation c inductive.

Démonstration : Nous allons montrer le résultat par induction sur k.

1. C_i^α est inductif, $I(C_i^\alpha)$ est une classe d'équivalence contenant C_i^α donc est inductif.
2. Posons $c_k = I(C_i^\alpha s_1 \dots s_k)$, $c_{k+1} = I(C_i^\alpha s_1 \dots s_{k+1})$ et $d = I(s_{k+1})$. Supposons que $C_i^\alpha s_1 \dots s_k$ est inductif, alors par définition $C_i^\alpha s_1 \dots s_{k+1}$ est aussi inductif. Supposons que c_k est inductif. On sait que c_{k+1} est une classe de $\approx$-équivalence ou un arbre non réduit à une feuille.
   (a) Si $c_{k+1} \in T/_\approx$, alors nécessairement c_k et d sont aussi des classes d'équivalence. D'après le lemme précédent, il y a dans ces classes deux termes f et x tels que $@(f,x) \in c_{k+1}$. Comme c_k est inductif, il contient un terme inductif t or $f \approx t$, d'où f est inductif. Ainsi le terme $@(f,x)$ est lui aussi inductif et donc c_{k+1} également.
   (b) Sinon c_k n'est pas une feuille, donc c_{k+1} non plus par définition de App.

Ceci clôt la démonstration. □

Lemme 2.16 (injectivité, cas de base)
Si $C_i^\alpha s_1 \dots s_n \approx^* C_i^\alpha t_1 \dots t_n$ et que l'interprétation commune de ces deux termes est une classe d'équivalence, alors pour tout $j \le n$, $s_j \approx^* t_j$.

Démonstration : Posons tout d'abord $u_k = C_i^\alpha s_1 \dots s_k$ et $v_k = C_i^\alpha t_1 \dots t_k$ pour tout $k \le n$. Soient $(u'_k, f_k, x_k)_{k \le n}$ trois suites de termes définies par le processus suivant :
– Si $k = 0$, alors $u'_0 = u_0 = C_i^\alpha$.

– Sinon, par construction de App, on déduit de manière immédiate que $I(u_{k-1})$ et $I(s_k)$ sont des classes d'équivalence, donc d'après le lemme 2.14, il existe $f_{k-1} \in I(u_{k-1})$ et $x_k \in I(s_k)$ tels que $(f_{k-1}x_k) \in I(u_k)$. Par récurrence, on suppose u'_{k-1} défini et on pose $u'_k = u'_{k-1}\, x_k$.

Enfin, on pose $f_n = t$ pour t arbitraire dans $I(u_n)$. Soit $\rightsquigarrow$ la relation qui fait que $\approx$ satisfait localement la théorie des constructeurs ; on peut montrer par récurrence que pour tout $k \leq n$, on a $u'_k \rightsquigarrow f_k$:

– Si $k = 0$, alors $f_0 \in I(C_i^\alpha)$, donc $f_0 \approx C_i^\alpha$ et comme $\approx$ satisfait la règle PROMOTION, on conclut que $u'_0 \rightsquigarrow f_0$:

$$\frac{C_i^\alpha \approx f_0}{C_i^\alpha \rightsquigarrow f_0}\ \text{PROMOTION}$$

– Supposons que $u'k \rightsquigarrow f_k$, on sait que $(f_k\, x_{k+1}) \approx f_{k+1}$ car ces deux termes sont dans la même classe d'équivalence, et comme $\approx$ satisfait la règle COMPOSE, on en déduit que $u'_k \rightsquigarrow f_k$:

$$\frac{C_i^\alpha x_1 \ldots x_k \rightsquigarrow f_k \quad (f_k\, x_{k+1}) \approx f_{k+1}}{C_i^\alpha x_1 \ldots x_{k+1} \rightsquigarrow f_{k+1}}\ \text{COMPOSE}$$

Ce qui clôt la récurrence. On construit de manière analogue trois suites de termes $(v'_k, g_k, y_k)_{k\leq n}$ en choisissant le même terme t pour g_n. On peut alors appliquer la règle d'injection :

$$\frac{C_i^\alpha x_1 \ldots x_n \rightsquigarrow t \quad C_i^\alpha y_1 \ldots y_n \rightsquigarrow t}{x_i \approx y_i}\ \text{INJECTION}$$

Or, par construction, $s_i \approx^* x_i$ et $t_i \approx^* y_i$, donc on peut conclure que $s_i \approx^* t_i$. □

LEMME 2.17 (INJECTIVITÉ)
$\approx^*$ satisfait la règle d'injectivité des constructeurs.

Démonstration : Supposons que $C_i^\alpha s_1 \ldots s_n \approx^* C_i^\alpha t_1 \ldots t_n$ et montrons par récurrence sur $k \leq n$ que si $C_i^\alpha s_1 \ldots s_k \approx^* C_i^\alpha t_1 \ldots t_k$, alors pour tout $j \leq k$, $s_j \approx t_j$.

Si $k = 0$, il n'y a rien à prouver.

Supposons l'hypothèse de récurrence vraie au rang k, soit c_{k+1} l'interprétation commune des termes $C_i^\alpha s_1 \ldots s_{k+1}$ et $C_i^\alpha t_1 \ldots t_{k+1}$. Deux cas sont possibles :

– Si c_{k+1} est un arbre dont les fils sont c_k et d, alors on a par construction :

$$I(C_i^\alpha s_1 \ldots s_k) = c_k = I(C_i^\alpha t_1 \ldots t_k)$$

On en déduit que $C_i^\alpha s_1 \ldots s_k \approx^* C_i^\alpha t_1 \ldots t_k$ et par hypothèse de récurrence on a $s_j \approx^* t_j$ pour tout $j \leq k$. De plus, on a $I(s_{k+1}) = I(t_{k+1})$ donc $s_{k+1} \approx^* t_{k+1}$.

– Sinon, c_{k+1} est une classe d'équivalence qui est inductive (d'après le lemme 2.15). Si $k + 1 = n$, on peut utiliser le lemme précédent, sinon on s'y ramène : soit τ la sorte de s_{k+2}, alors, comme T est clos par application et que la sorte τ est habitée par le terme s_{k+2}, il existe $f_{k+1} \in c_{k+1}$ et $x_{k+2} \in T$ tels que $f_{k+1}\, x_{k+2} \in T$. Posons alors :

$$c'_{k+2} = I(C_i^\alpha s_1 \ldots s_{k+1}\, x_{k+2}) = I(C_i^\alpha t_1 \ldots t_{k+1}\, x_{k+2})$$

On peut affirmer que c'_{k+2} est une classe d'équivalence. En itérant le processus $n - k - 1$ fois, on obtient deux termes équivalents dont l'interprétation est une classe d'équivalence :

$$C_i^\alpha s_1 \dots s_{k+1}\, x_{k+2} \dots x_n \approx^* C_i^\alpha t_1 \dots t_{k+1}\, x_{k+2} \dots x_n$$

On peut alors leur appliquer le lemme précédent et ainsi conclure.
Ce qui clôt notre démonstration. □

Lemme 2.18 (séparation, cas de base)
Pour tous constructeurs C_i^α et C_j^α d'arité respective n et p, et pour toutes suites de termes $(s_k)_{k\leq n}$ et $(t_l)_{l\leq p}$, si $I(C_i^\alpha s_1 \dots s_n) = I(C_j^\alpha t_1 \dots t_p) = c$ avec $c \in T/_\approx$, alors la relation $\approx$ ne satisfait pas localement les axiomes de séparation.

Démonstration : Soit $t \in c$ un terme arbitraire et $\rightsquigarrow$ la relation permettant à $\approx$ de satisfaire localement la théorie des constructeurs. De manière analogue à la preuve du lemme 2.16, on peut trouver des termes $(x_k)_{k\leq n}$ et $(y_l)_{l\leq p}$ dans T tels que $C_i^\alpha\, x_1 \dots x_n \rightsquigarrow t$ et $C_j^\alpha\, y_1 \dots y_p \rightsquigarrow t$, ce qui contredit l'hypothèse que $\rightsquigarrow$ satisfait les axiomes de séparation. □

Lemme 2.19 (séparation)
$\approx^*$ satisfait les axiomes de séparation.

Démonstration : Supposons $C_i^\alpha s_1 \dots s_n \approx^* C_j^\alpha t_1 \dots t_p$ et montrons que $\approx$ ne satisfait pas localement les axiomes de séparation. Sans perte de généralité, supposons que $n \leq p$.

Montrons par récurrence sur $k \leq n$ que $C_i^\alpha s_1 \dots s_k \approx^* C_j^\alpha t_1 \dots t_{p-n+k}$ est contradictoire avec les hypothèses.

- Si $k = 0$, alors comme $I(C_i^\alpha)$ est une classe d'équivalence, on peut utiliser un processus analogue à celui de la démonstration du lemme 2.17 pour se ramener à des constructeurs totalement appliqués, pour lesquels on peut appliquer le lemme 2.18.
- Supposons l'hypothèse de récurrence vraie au rang k, soit c_{k+1} l'interprétation commune des termes $C_i^\alpha s_1 \dots s_{k+1}$ et $C_j^\alpha t_1 \dots t_{p-n+k+1}$. Deux cas sont possibles :
  - Si c_{k+1} est un arbre dont les fils sont c_k et d, alors par construction on a :

    $$I(C_i^\alpha s_1 \dots s_k) = c_k = I(C_j^\alpha t_1 \dots t_{p-n+k})$$

    On applique l'hypothèse de récurrence.
  - Sinon, c_{k+1} est une classe d'équivalence qui est inductive, on peut alors procéder comme pour le cas $k = 0$.

Ce qui clôt notre démonstration. □

Démonstration du théorème 2.13 : Grâce aux lemmes 2.17 et 2.19, nous pouvons enfin affirmer que $\approx^*$ est une congruence satisfaisant la théorie des constructeurs libres, ce qui clôt la démonstration du théorème 2.13. □

règle d'initialisation

$$\frac{Sat(\mathcal{A})?}{T_0, E, M, U, \emptyset}\ \textsc{Init}\begin{cases} T_0 = \{@(u,v) \in \mathcal{A}\downarrow\} \\ E = \{(u,v) | u \approx v \in \mathcal{A}\} \\ M = \{(C_i^\alpha, C_i^\alpha) | C_i^\alpha \in \mathcal{A}\downarrow\} \\ U = u \mapsto (u, \{@(u,v) \in A\downarrow\}, \{@(v,u) \in A\downarrow\}, \emptyset) \end{cases}$$

règles de conclusion

$$\frac{\emptyset, \emptyset, \emptyset, U, S}{SAT}\ \textsc{Ok}\ \Big\{ \forall u \not\approx v \in \mathcal{A}, \bar{u} \neq \bar{v}$$

$$\frac{T, E, M, U, S}{INSAT}\ \textsc{Refute}\ \Big\{ \exists u \not\approx v \in \mathcal{A}, \bar{u} = \bar{v}$$

$$\frac{T, E, M \cup \{(t, C_i^\alpha\, \vec{u})\}, U, S}{INSAT}\ \textsc{Conflict}\begin{cases} C_i^\alpha \vec{u} : \alpha \\ C_j^\alpha \vec{v} \in C(t) \\ i \neq j \end{cases}$$

règles de déduction

$$\frac{T, E \cup \{(s,t)\}, M, U, S}{T \cup G(s) \cup D(s), E, M \cup \{(t,c) | c \in C(s)\}, \mathrm{union}(U,s,t), \mathrm{suppr}(S, G(s) \cup D(s))}\ \textsc{Merge}\ \Big\{ \bar{s} \neq \bar{t}$$

$$\frac{T, E \cup \{(s,t)\}, M, U, S}{T, E, M, U, S}\ \textsc{no-Merge}\ \Big\{ \bar{s} = \bar{t}$$

$$\frac{T, E, M \cup \{(t, C_i^\alpha\, \vec{u})\}, U, S}{T, E \cup \{(\vec{u}, \vec{v})\}, M, U, S}\ \textsc{Match}\begin{cases} C_i^\alpha \vec{u} : \alpha \\ C_i^\alpha \vec{v} \in C(t) \end{cases}$$

$$\frac{T, E, M \cup \{(t, C_i^\alpha\, \vec{u})\}, U, S}{T \cup G(t), E, M, U\{C(t) \leftarrow C(t) \cup C_i^\alpha\, \vec{u}\}, S}\ \textsc{Mark}\begin{cases} C_i^\alpha \vec{u} : \tau \to \tau' \\ \quad \text{ou} \\ C(t) = \emptyset \end{cases}$$

$$\frac{T \cup \{t\}, E, M, U, S}{T, E \cup \{(t,s)\}, M \cup \{(t, F\, v) | F \in \mathcal{C}\}, U, S}\ \textsc{Update}_1\begin{cases} t = @(u,v) \\ S(\bar{u}, \bar{v}) = s \\ C(u) = \mathcal{C} \end{cases}$$

$$\frac{T \cup \{t\}, E, M, U, S}{T, E, M \cup \{(t, F\, v) | F \in \mathcal{C}\}, U, S \cup \{(\bar{u}, \bar{v}) \mapsto t\}}\ \textsc{Update}_2\begin{cases} t = @(u,v) \\ S(\bar{u}, \bar{v}) = \bot \\ C(u) = \mathcal{C} \end{cases}$$

FIG. 2.1 – Algorithme de clôture de congruence avec constructeurs et applications partielles

2.3 L'algorithme de décision

2.3.1 Description

On souhaite maintenant donner un algorithme permettant de décider la satisfaisabilité dans la théorie de l'égalité sans quantificateurs avec des constructeurs libres.

L'algorithme est décrit par un ensemble de règles de transition (lues de haut en bas) entre des états composés d'un quintuplet (T, E, M, U, S) contenant les objets suivants :

- T : Ensemble de **T**ermes à mettre à jour (dans S et pour les marquages)
- E : Ensemble de couples de termes à rendre **É**gaux
- M : Ensemble de **M**arquages de constructeurs à effectuer
- U : Structure **U**NION-FIND associant à chaque terme sa classe
- S : Table des **S**ignatures (La signature d'un terme est le couple des classes d'équivalences de ses deux sous-termes.)

De plus, une classe est un quadruplet (r, G, D, C) contenant :

- r : **r**eprésentant de la classe
- G : Ensemble des ancêtres **G**auches de la classe
- D : Ensemble des ancêtres **D**roits de la classe
- C : Ensemble des marquages de **C**onstructeurs de la classe d'équivalence, c'est-à-dire de termes (potentiellement extérieurs au problème de départ) équivalents aux termes de cette classe.

Par abus de langage, on notera $U(u) = (\bar{u}, G(u), D(u), C(u))$. Un ancêtre gauche d'une classe c est un terme dont le sous-terme gauche est dans la classe c ; un ancêtre droit d'une classe c est un terme dont le sous-terme droit est dans la classe c.

La structure UNION-FIND est composée d'une forêt d'arbres dont les noeuds sont des termes et les racines les représentants de la classe. L'opération de recherche de la classe consiste alors à remonter l'arbre jusqu'à sa racine. L'opération union(U, s, t) qui fusionne deux classes d'équivalence consiste à placer $\bar{s}$ comme sous-noeud immédiat de $\bar{t}$, faisant ainsi de $\bar{t}$ le représentant des éléments de l'ancienne classe de s. L'ensemble des ancêtres gauches et droits de la classe de s sont alors ajoutés à ceux de la classe de t, mais les constructeurs de la classe de s ne sont pas ajoutés à l'ensemble des marquages.

La table des signatures associe à un couple de représentants de classe $(\bar{u}, \bar{v})$ un terme dont les sous-termes sont respectivement dans la classe de u et dans celle de v, si un tel terme existe, et $\perp$ sinon. L'opération suppr(S, D) supprime les liaisons qui associent un couple à un terme de l'ensemble D.

Les règles de transition de l'algorithme sont décrites dans la figure 2.1. La règle INIT crée la structure initiale, où les termes identiques sont partagés, où les constructeurs sont à marquer et où chaque axiome d'égalité est ajouté à l'ensemble E. De plus la table des signatures est vide et tous les termes non constants sont marqués comme étant à mettre à jour. L'ensemble $\mathcal{A}\downarrow$ est l'ensemble des termes et sous-termes apparaissant dans $\mathcal{A}$.

L'algorithme peut se terminer de trois façons différentes : la règle REFUTE détecte si une diségalité du problème est contredite et répond alors que le problème est insatisfaisable. Dans le cas contraire, et s'il ne reste plus de contraintes à propager, la règle OK conclut

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{SAT(f\,a \approx a, f(f\,a) \not\approx a)?}{\{f\,a, f(f\,a)\}, \{(f\,a, a)\}, \emptyset, \emptyset, \emptyset}\ \textsc{Init}}{\{f\,a, f(f\,a)\}, \emptyset, \emptyset, \{f\,a \mapsto a\}, \emptyset}\ \textsc{Merge}}{\{f(f\,a)\}, \emptyset, \emptyset, \{f\,a \mapsto a\}, \{(f, a) \mapsto f\,a\}}\ \textsc{Update}_2}{\emptyset, \{(f(f\,a), f\,a)\}, \emptyset, \{f\,a \mapsto a\}, \{(f, a) \mapsto f\,a\}}\ \textsc{Update}_1 \quad S(\bar{f}, \overline{f\,a}) = f\,a}{\emptyset, \emptyset, \emptyset, \left\{ \begin{array}{c} f\,a \mapsto a \\ f(f\,a) \mapsto a \end{array} \right\}, \{(f, a) \mapsto f\,a\}}\ \textsc{Merge}}{INSAT}\ \textsc{Refute} \quad \overline{f(f\,a)} = a = \bar{a}$$

FIG. 2.2 – Calcul de satifaisabilité pour $\{f\,a \approx a, f(f\,a) \not\approx a\}$

que l'ensemble d'axiomes est satisfaisable. La règle CONFLICT est déclenchée lorsque l'on déduit une égalité entre constructeurs distincts et le problème est déclaré insatisfaisable.

Les règles MERGE et NO-MERGE servent à effectuer les fusions de classes d'équivalence lorsqu'une congruence est détectée. La règle NO-MERGE permet d'éviter des opérations inutiles.

Les règles MATCH et MARK permettent d'annoter les classes d'équivalence par les termes inductifs qu'elles contiennent. Dans le cas de la règle MATCH, on ajoute à l'ensemble E les conséquences de la propriété d'injectivité.

Les règles $\textsc{Update}_{1/2}$ effectuent la mise à jour de la table des signatures en ajoutant une demande de marquage si le sous-terme droit est un constructeur, ou de fusion si un terme a une signature qui est déjà dans la table.

2.3.2 Quelques exemples

Nous faisons la convention de noter les symboles de fonction ordinaires avec une minuscule et les symboles de constructeurs avec une majuscule. Le premier exemple est une exemple très simple destiné à illustrer le fonctionnement de la table de signature et de la propagation des congruences. Dans l'ensemble U, la notation $s \mapsto t$ indique que $\bar{s} = t$, pour les termes non mentionnés, il est implicite que $\bar{u} = u$. De même, sauf mention contraire, $C(t) = \emptyset$.

EXEMPLE 2.20
On considère la signature $\mathcal{S}_0 = \{s\}, \Sigma = \{a : s, f : s \rightarrow s\}$ et le problème simple suivant : $\{f\,a \approx a, f(f\,a) \not\approx a\}$. Une exécution possible de l'algorithme est donnée dans la figure 2.2.

EXEMPLE 2.21
Pour illustrer l'utilisation des marquages de classes d'équivalence par des termes avec constructeurs, nous allons donner un exemple qui utilise à la fois l'injectivité et l'élimi-

nation forte. Considérons la signature suivante :

$$\mathcal{S}_0 = \{\texttt{bool}, \texttt{box}\},$$
$$\Sigma = \{g : \texttt{bool}\to\texttt{box}, B : \texttt{bool}\to\texttt{box}, T : \texttt{bool}, F : \texttt{bool}\}$$

Le problème de satisfaisabilité $\{B \approx g, g\,T \approx g\,F\}$ est résolu par notre algorithme dans la figure 2.3.

2.3.3 Terminaison

THÉORÈME 2.22 (TERMINAISON)
Toute application des règles de l'algorithme jusqu'à saturation débouche en un nombre fini d'étapes sur SAT ou $INSAT$.

Démonstration : On appelle taille d'une sorte la fonction définie par $|\tau\to\tau'| = 1 + |\tau'|$ et $|\alpha| = 0$ si α est une sorte de base. On définit alors un ordre $\prec$ sur les couples de termes de même sorte et sur les termes. On note τ_u la sorte d'un terme u et $\tau_{(u,v)}$ la sorte commune aux deux termes u et v ; on pose alors $x \prec y$ si, et seulement si, ou bien $|\tau_x| < |\tau_y|$ pour x et y quelconques, ou alors $|\tau_x| = |\tau_y|$ et x est un couple et y un terme. Par construction l'ordre $\prec$ est bien fondé.

L'argument de terminaison pour l'algorithme est un ordre lexicographique sur :

1. le nombre de classes d'équivalence
2. l'union $T \cup M$, ordonnée par l'extension multi-ensemble de $\prec$
3. le cardinal de l'ensemble E

Examinons maintenant les règles de transition récursives :

MERGE Le nombre de classes d'équivalence décroît strictement car on fusionne deux classes distinctes (grâce à la condition de bord). C'est la seule règle qui modifie le nombre de classes d'équivalence.

MATCH Comme on enlève un terme de M, $T \cup M$ décroît pour l'ordre multi-ensemble.

MARK On remplace un couple (t, u) de M par un ensemble fini de termes $G(t)$ qui sont des ancêtres gauches de t. Si cet ensemble est non vide, alors cela signifie que t a une sorte fonctionnelle $\tau\to\tau'$ et donc que ses ancêtres gauches sont de sorte τ'. On en conclut que $\forall v \in G(t), v \prec t$ et que $T \cup M$ décroît strictement.

UPDATE$_{1/2}$ Dans l'union $T \cup M$, on remplace un terme t par un ensemble fini de couples de termes (t, v) de la même sorte. Pour chacun de ces couples, on a donc $(t, v) \prec t$ et donc $T \cup M$ décroît pour l'extension multi-ensemble de $\prec$.

NO-MERGE Le cardinal de E décroît strictement, les ensembles T et M sont inchangés.

De plus il y a toujours une règle qui s'applique :
- Si E est non vide, on peut appliquer la règle MERGE ou NO-MERGE,
- Sinon :
  - si M est non vide, on peut appliquer l'une des règles CONFLICT, MATCH ou MARK.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{SAT(B \approx g, g\,T \approx g\,F)?}{\{g\,T, g\,F\}, \{(B,g),(g\,T,g\,F)\}, \{(T,T),(F,F),(B,B)\}, \emptyset, \emptyset}\ \textsc{Init}}{\{g\,T, g\,F\}, \{(g\,T,g\,F)\}, \{(T,T),(F,F),(B,B)\}, \{B \mapsto g\}, \emptyset}\ \textsc{Merge}}{\{g\,T, g\,F\}, \emptyset, \{(T,T),(F,F),(B,B)\}, \left\{ \begin{array}{c} g\,T \mapsto g\,F \\ B \mapsto g \end{array} \right\}, \emptyset}\ \textsc{Merge}}{\{g\,T, g\,F\}, \emptyset, \{(F,F),(B,B)\}, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & \end{array} \right\}, \emptyset}\ \textsc{Mark}}{\{g\,T, g\,F\}, \emptyset, \{(B,B)\}, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(F) = \{F\} \end{array} \right\}, \emptyset}\ \textsc{Mark}}{\{g\,T, g\,F\}, \emptyset, \emptyset, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g_B & C(F) = \{F\} \\ & C(g) = \{B\} \end{array} \right\}, \emptyset}\ \textsc{Update}_2}{\{g\,F\}, \emptyset, \{(g\,T, B\,T)\}, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(F) = \{F\} \\ & C(g) = \{B\} \end{array} \right\}, \emptyset}\ \textsc{Mark}}{\{g\,F\}, \emptyset, \emptyset, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(F) = \{F\} \\ C(g\,F) = \{B\,T\} & C(g) = \{B\} \end{array} \right\}, \emptyset}\ \textsc{Update}_2}{\emptyset, \emptyset, \{(g\,F, B\,F)\}, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(F) = \{F\} \\ C(g\,F) = \{B\,T\} & C(g) = \{B\} \end{array} \right\}, \emptyset}\ \textsc{Match} \quad B\,F \approx B\,T}{\emptyset, \{(F,T)\}, \emptyset, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(F) = \{F\} \\ C(g\,F) = \{B\,T\} & C(g) = \{B\} \end{array} \right\}, \emptyset}\ \textsc{Merge}}{\{g\,F\}, \emptyset, \{(F,F)\}, \left\{ \begin{array}{cc} g\,T \mapsto g\,F & C(T) = \{T\} \\ B \mapsto g & C(g) = \{B\} \\ F \mapsto T & C(g\,F) = \{B\,T\} \end{array} \right\}, \emptyset}\ \textsc{Conflict} \quad F \approx T}{INSAT}$$

FIG. 2.3 – Calcul de satisfaisabilité pour $\{B \approx g, g\,T \approx g\,F\}$

- Sinon :
  - si T est non vide, on peut appliquer l'une des règles UPDATE_1 ou UPDATE_2
  - sinon on peut appliquer REFUTE ou OK.

On en déduit que pour toute entrée $\mathcal{A}$, l'algorithme se termine toujours par SAT ou $INSAT$, et ce quelle que soit la stratégie utilisée. □

2.3.4 Correction

Pour nous assurer de la correction, il faut prouver que si l'algorithme retourne la réponse $INSAT$, alors le problème est insatisfaisable :

Pour cela nous allons prouver les invariants suivants :

1. Si $\bar{u} = \bar{v}$, alors $u \approx v$ est une conséquence de $\mathcal{A}$ dans la théorie de l'égalité avec constructeurs libres.
2. Si $c \in C(u)$, alors $c \approx u$ est une conséquence de $\mathcal{A}$ dans la théorie de l'égalité avec constructeurs libres.
3. Si $(t, c) \in M$, alors $t \approx c$ est une conséquence de $\mathcal{A}$ dans la théorie de l'égalité avec constructeurs libres.
4. Si $(u, v) \in E$, alors $u \approx v$ est une conséquence de $\mathcal{A}$ dans la théorie de l'égalité avec constructeurs libres.
5. Si $S(u, v) = w$, alors il existe u', v' et w' tels que $\bar{u}' = \bar{u}$, $\bar{v}' = \bar{v}$, $\bar{w}' = \bar{w}$ et $w' = @(u, v)$.
6. Les ensembles $G(u)$ et $D(u)$ contiennent tous les ancêtres des termes de la classe de u et rien que ceux-ci.

Démonstration des invariants : La règle INIT établit ces invariants : si $\bar{u} = \bar{v}$, alors $u = v$ syntaxiquement et si $(u, v) \in E$, alors $u \approx v \in \mathcal{A}$. L'invariant 3 est vrai par réflexivité.

On donne ici les arguments de préservation pour les cas non triviaux :

MERGE L'invariant 1 reste vrai en utilisant la règle TRANS et grâce au fait qu'il était vrai avant l'application de la règle ainsi que grâce à l'invariant 4. L'invariant 5 reste vrai car seuls les ancêtres de la classe de s ont changé de signature (invariant 6). L'invariant 6 reste vrai car l'opération union fusionne aussi les ensembles d'ancêtres des deux classes fusionnées.

NO-MERGE Les invariants sont trivialement préservés.

MATCH L'invariant 4 est vérifié en utilisant la propriété d'injectivité ainsi que les invariants 2 et 3 à l'étape précédente.

MARK L'invariant 2 est une conséquence de l'invariant 3 à l'étape précédente.

UPDATE_1 Pour l'invariant 4, il faut montrer que le nouveau couple (t, s) est correct. Comme $t = @(u, v)$, on trouve, grâce à l'invariant 5, des termes u',v' et s' tels que $s' = @(u', v')$. Grâce à l'invariant 1, on a $u \approx u'$, $v \approx v'$ et $s' \approx s$ et on conclut par les règles $\text{CONGR}_@$ et TRANS. L'invariant 3 est vérifié par congruence.

UPDATE$_2$ L'invariant 5 est trivialement vérifié par t.

Ceci clôt la démonstration de préservation des invariants. □

THÉORÈME 2.23 (CORRECTION)
L'algorithme est correct.

Démonstration : Si l'on conclut par la règle CONFLICT, alors grâce aux invariants 2 et 3, on peut déduire que $C_i^\alpha \vec{u} \approx t$ et $C_j^\alpha \vec{v} \approx t$, donc par transitivité on déduit que l'on ne peut satisfaire l'axiome de séparation $C_i^\alpha \vec{u} \not\approx C_j^\alpha \vec{v}$.

Sinon, on conclut par la règle REFUTE et alors par l'invariant 1, on sait que l'on peut dériver $u \approx v$ des axiomes de $\mathcal{A}$, ce qui fait que $u \not\approx v$ n'est pas satisfaisable. □

2.3.5 Complétude

Pour la complétude, il nous faut montrer que si l'algorithme retourne SAT, alors $\mathcal{A}$ est satisfaisable dans la théorie des constructeurs. Pour cela, nous allons montrer que l'équivalence induite par la structure UNION-FIND à la fin de l'algorithme est une congruence satisfaisant localement la théorie des constructeurs libres.

LEMME 2.24 (COMPLÉTUDE LOCALE)
Si l'algorithme retourne SAT, alors, à la fin de l'algorithme, la relation d'équivalence $\approx$ définie par $u \approx v \Leftrightarrow \bar{u} = \bar{v}$ est une congruence satisfaisant localement la théorie des constructeurs libres.

Démonstration : A toutes les étapes de l'algorithme, $\approx$ est une classe d'équivalence car elle est induite par une partition de l'ensemble des termes en une forêt d'arbres. On remarque que les règles de déduction préservent l'unicité de la sorte des termes d'une classe d'équivalence, donc $\approx$ est toujours bien sortée.

Nous noterons $\sim$ la plus petite relation d'équivalence contenant $\approx$ et les couples de termes de l'ensemble E. On souhaite alors établir les invariants suivants :

1. Si $@(f,x) \in T_0 \setminus T$, $@(g,y) \in T_0 \setminus T$, $f \approx g$ et $x \approx y$ alors $@(f,x) \sim @(g,y)$.
2. Si $@(f,x) \in T_0 \setminus T$, alors il existe w tel que $S(\bar{f},\bar{x}) = w$ et $w \sim @(f,x)$.
3. Si $@(f,t_{k+1}) \in T_0 \setminus T$ et $C_i^\alpha\, t_1 \ldots t_k \in C(f)$, alors il existe $u_1,\ldots,u_{k+1}$ tels que pour tout j, $u_j \sim t_j$ et ou bien $C_i^\alpha\, u_1 \ldots u_{k+1} \in C(@(f,t_{k+1}))$ ou alors il existe $v \approx @(f,t_{k+1})$ tel que $(v, C_i^\alpha\, u_1 \ldots u_{k+1}) \in M$.
4. Pour tout constructeur, ou $C_i^\alpha \in C(C_i^\alpha)$, ou il existe $c \approx C_i^\alpha$ tel que $(c, C_i^\alpha) \in M$.
5. Si u est dans une sorte de base, $C(u)$ contient au plus un élément.
6. Si $u \approx v \in \mathcal{A}$, alors $u \sim v$.

On remarque que les relations $\approx$ et $\sim$ croissent au cours du temps, c'est-à-dire que deux termes équivalents à une étape donnée le restent toujours.

INIT Comme $T = T_0$, les invariants 1, 2 et 3 sont triviaux. Les invariants 4, 5 et 6 sont vérifiés par construction.

MERGE On remarque que MERGE préserve la relation $\sim$, et comme les ancêtres de s sont exclus de l'invariant par leur ajout dans T, l'invariant 1 est préservé. L'invariant 2 est préservé car les termes effacés de la signature sont justement ceux ajoutés dans T, et les autres termes n'ont pas changé de signature. L'invariant 4 est préservé car si $C_i^\alpha \approx s$, alors (s, C_i^α) est ajouté dans M. L'invariant 3 est préservé car même si $C(s)$ est maintenant égal à $C(t)$, les ancêtres de s sont maintenant dans T d'une part, et on a ajouté les éléments de $C(s)$ dans M d'autre part. Enfin 5 et 6 sont préservés.

NO-MERGE Les invariants sont trivialement préservés.

MATCH Les invariants 1, 2, 4, 5, 6 sont trivialement respectés. L'invariant 3 est respecté car on a ajouté les couples (u_k, v_k) dans E.

MARK Les invariants 1 et 2 sont affaiblis donc respectés. Les invariants 4, 5 et 6 sont préservés. L'invariant 3 est respecté car les ancêtres gauches de t en sont exclus par leur ajout dans T.

UPDATE$_1$ L'invariant 1 est vérifié pour $@(u, v)$ car si $u \approx g$ et $v \approx y$, alors $S(x, y) = S(u, v)$ et comme on a ajouté (t, s) dans E , on a bien $@(u, v) \sim @(t, s)$. D'après les conditions de bord, $@(u, v)$ satisfait l'invariant 2. Les invariants 4, 5 et 6 sont toujours vérifiés. L'invariant 3 est vérifié par t puisque l'on ajoute les paires (t, Fv) dans M.

UPDATE$_2$ L'invariant 1 est vérifié puisque l'on déduit de l'invariant 2 précédent, qu'aucun terme n'est congruent à t. L'invariant 2 est verifié car on ajoute t et sa signature dans S. Les autres invariants sont maintenus pour les mêmes raisons que pour la règle précédente.

À la fin de l'exécution de l'algorithme, on remarque que $E = \emptyset$ donc $(\approx) = (\sim)$. Comme $T = \emptyset$, l'invariant 1 permet de conclure que $\approx$ est une congruence. Nous noterons ici $\rightsquigarrow$ la relation définie par $C_i^\alpha s_1 \ldots s_k \rightsquigarrow t$ si, et seulement si, il existe un terme $C_i^\alpha t_1 \ldots t_k$ dans $C(t)$ tel que $s_j \approx t_j$ pour tout $j \leq k$. La relation $\rightsquigarrow$ satisfait alors la règle PROMOTION puisque $M = \emptyset$ et que l'invariant 4 est préservé.

Pour la règle COMPOSE, supposons que $C_i^\alpha u_1 \ldots u_k \rightsquigarrow f$ et que $@(f, u_{k+1}) \approx g$, alors il existe $v_1, \ldots, v_k$ tel que $u_j \approx v_j$ pour tout $j \leq k$ et $C_i^\alpha v_1 \ldots v_k \in C(f)$. Posons $v_{k+1} = u_{k+1}$. D'après l'invariant 4, il existe $w_j \approx v_j$ pour tout $j \leq k+1$ tel que $C_i^\alpha w_1 \ldots w_{k+1} \in C(@(f, u_{k+1}))$, on en déduit que $C_i^\alpha v_1 \ldots v_{k+1} \in C(g)$ et donc que $C_i^\alpha u_1 \ldots u_{k+1} \rightsquigarrow g$.

Pour la règle INJECTION, supposons que $C_i^\alpha u_1 \ldots u_n \rightsquigarrow w$ et que $C_i^\alpha v_1 \ldots v_n \rightsquigarrow w$, où n est l'arité de C_i^α, alors il existe $s_1, \ldots, s_n$ et $t_1, \ldots, t_n$ tels que pour tout $j \leq n$, $u_j \approx s_j$ et $v_j \approx t_j$, et $C_i^\alpha u_1 \ldots u_n \in C(w)$ et $C_i^\alpha v_1 \ldots v_n \in C(w)$. D'après l'invariant 5, ces derniers termes sont égaux car $C(w)$ contient au plus un élément (w est de sorte α), et donc pour tout $j \leq n$, $u_j = v_j$ donc on peut conclure que $s_j \approx t_j$.

Les axiomes de séparation sont satisfaits car, d'après l'invariant 5, on ne peut avoir deux constructeurs distincts totalement appliqués dans le même $C(u)$.

On en conclut que $\approx$ satisfait localement la théorie des constructeurs libres. □

Pour établir la complétude globale, il nous manque une hypothèse supplémentaire : l'ensemble T doit être clos par application pour la congruence induite par la structure UNION-FIND. Ceci n'est pas vrai en général, mais l'algorithme peut être adapté pour le vérifier, grâce aux ajouts suivants :

- Au début de l'algorithme, chaque classe d'équivalence est marquée comme étant de statut *INCONNU*.
- Lorsqu'on marque une classe de type *INCONNU* avec un constructeur partiellement appliqué, on marque la classe avec le statut *INCOMPLET*.
- Lorsqu'on applique l'une des règles UPDATE à un terme $(f\,x)$ et que la classe de f est de statut *INCOMPLET*, marquer cette classe avec le statut *COMPLET*.
- Lorsqu'on fusionne deux classes, la classe fusionnée a le statut le plus évolué pour l'ordre $\sqsubseteq$ de ceux des classes fusionnées, avec :

$$INCONNU \sqsubset INCOMPLET \sqsubset COMPLET$$

Ainsi, lorsque l'algorithme répond *SAT*, l'ensemble final est clos par application si, et seulement aucune classe n'est de statut *INCOMPLET*.

Nous pouvons maintenant démontrer le théorème suivant :

THÉORÈME 2.25 (COMPLÉTUDE)
Soit $\mathcal{A}$ un ensemble fini non vide d'axiomes : si l'exécution de l'algorithme sur $\mathcal{A}$ retourne SAT et que l'ensemble final est clos par application pour la congruence $\approx$ construite par l'algorithme, alors $\mathcal{A}$ est satisfaisable.

Démonstration : En utilisant le lemme précédent en conjonction avec le théorème 2.13. □

Si l'on dispose d'un oracle capable de retourner un terme d'une sorte habitée, on peut ajouter à l'état final de l'algorithme l'application des termes inductifs incomplètement appliqués aux constantes données par l'oracle. On peut ensuite lancer à nouveau l'algorithme sur l'ensemble complété. La relation construite par la nouvelle exécution de l'algorithme fera de l'ensemble complété un ensemble clos par application pour $\approx$.

2.4 La tactique `congruence`

Une version précédente de l'algorithme décrit plus haut est implantée et distribuée dans l'assistant de preuve Coq (version 8). Cette version souffrait de quelques limitations dont ne souffre plus la version courante qui est disponible dans la version de développement de Coq. La tactique `congruence` se compose d'une procédure extrayant d'un but Coq l'ensemble des égalités et diségalités qu'il contient, et ajoute la négation du but si celui-ci est une égalité. Si le but n'est pas une égalité, alors on ajoute au problème les diségalités $H \not\approx C$ où H est une hypothèse qui n'est pas une égalité, et C la conclusion du but. De plus, on ajoute $H \not\approx H'$ pour tout couple d'hypothèses $h : H$ et $h' : \neg H'$, si H et H' ne sont pas des égalités.

Ensuite la procédure exécute l'algorithme de la figure 2.1 avec la stratégie suivante : on applique Merge et no-Merge si E est non-vide, puis Mark, Match ou Conflict si E est vide et M est non-vide, puis Update$_{1/2}$ si E et M sont vides et T non-vide ; enfin on conclut par Refute ou Ok.

Si l'algorithme retourne $INSAT$, il est instrumenté pour produire une preuve de l'égalité niée par l'hypothèse. La méthode utilisée pour produire la preuve est une extension de celle décrite dans [Cor01], ainsi que plus récemment dans [NO05].

En outre, la tactique `congruence` peut détecter si l'ensemble final n'est pas clos par application et tente de le compléter avec des constantes sans signification, de telle sorte que si l'ensemble complété est insatisfaisable, la procédure suggère à l'utilisateur comment résoudre son problème. Nous illustrons cela avec un type inductif représentant un triplet d'entiers :

```
Inductive Cube:Set :=
| Triple: nat -> nat -> nat -> Cube.
```

Nous souhaitons alors prouver le théorème suivant :

```
Theorem incomplete : ∀ a b c d : nat,
  Triple a = Triple b → Triple d c = Triple d b → a = c.
```

Grâce à la commande `intros`, notre but contient un ensemble d'égalités sans quantificateurs :

```
a b c d : nat
H : Triple a = Triple b
H0 : Triple d c = Triple d b
______________________________________
a = c
```

La tactique `congruence` peut alors construire l'ensemble d'axiomes correspondant :

$$\{\texttt{Triple a} \approx \texttt{Triple b}, \texttt{Triple d c} \approx \texttt{Triple d b}, \texttt{a} \not\approx \texttt{c}\}$$

L'algorithme détecte que l'ensemble d'axiomes est satisfaisable mais que l'ensemble de termes n'est pas clos par application. Il tente de le compléter et parvient à une contradiction, il en informe donc l'utilisateur :

```
Goal is solvable by congruence but some arguments are missing.
Try "congruence with (Triple a ?1 ?2) (Triple d c ?3)",
replacing metavariables by arbitrary terms.
```

L'utilisateur entre alors la commande suivante qui résout sans problème la question posée, grâce à l'addition de termes supplémentaires dans l'ensemble considéré, qui le rendent clos par application.

```
congruence with (Triple a 0 0) (Triple d c 0).
```

2.5 Conclusion

La tactique `congruence` est une tactique rapide résolvant un problème très précis, c'est pourquoi elle est particulièrement efficace pour « nettoyer » un ensemble de sous-buts dont beaucoup sont triviaux. Cette caractéristique est particulièrement utile pour travailler sur les égalités entre booléens, notamment dans les preuves Coq présentées dans le chapitre 4.

Notre étude peut être prolongée dans plusieurs directions :

- une analyse détaillée de complexité
- une analyse de performances (*benchmarks*)
- l'utilisation de l'algorithme pour produire des traces de preuve réflexives (cf. chapitre 4)
- l'étude du cas des algèbres de termes avec types dépendants, et le problème de la conservativité de l'égalité hétérogène [McB99] par rapport à l'égalité classique dans le cas clos.

Chapitre 3

Preuves intuitionnistes dans le Calcul des Constructions

La classe de problèmes considérés dans ce chapitre est d'abord l'ensemble des énoncés du premier ordre, tels que définis au chapitre 1, puis des extensions de ce langage, que nous décrivons.

Tout d'abord, nous rappelons dans la section 3.1 les apports des travaux de Roy Dyckhoff (notamment [Dyc92]) sur les calculs de séquents sans contraction pour la logique intuitionniste propositionnelle (calcul LJT) et du premier ordre ($G4i$).

Dans la section 3.2, nous décrivons une extension de ce système appelée $LJTI$, et nous montrons en quoi cette extension est adaptée pour représenter le fragment du premier ordre du système Coq incluant une certaine forme de définitions inductives *non-récursives*.

Nous établissons les propriétés fondamentales de $LJTI$ dans la section 3.3.

Ensuite nous décrivons dans la section 3.4 l'implantation dans Coq d'une tactique utilisant ce système logique. Nous expliquons les choix d'implantation.

Nous montrons ensuite, dans la section 3.5, comment on peut étendre le calcul $LJTI$ pour la logique équationnelle, nous y étendons également les résultats théoriques obtenus.

3.1 Calculs de Séquents sans Contraction

La plupart des calculs de prédicats intuitionnistes présentés à l'aide de séquents [Kle52, Dra87] — et notamment le calcul LJ — de Gentzen [Gen34] incluent une règle structurelle de contraction :

$$\frac{\Gamma, A, A \vdash G}{\Gamma, A \vdash G}\ Contr$$

D'autres évitent cette règle et utilisent une version modifiée de la règle habituelle d'élimination de l'implication :

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash G}{\Gamma, A{\rightarrow}B \vdash G}\ L{\rightarrow} \quad \Longrightarrow \quad \frac{\Gamma, \boxed{A{\rightarrow}B} \vdash A \quad \Gamma, B \vdash G}{\Gamma, A{\rightarrow}B \vdash G}\ L{\rightarrow}^{*}$$

Le problème est que ces règles *Contr* ou $L{\to}^*$ ont la désagréable propriété, si on les voit comme des règles de transformation de séquents du bas vers le haut ainsi que le font certaines procédures de recherche de preuve, de permettre des dérivations infinies si l'on ne contrôle pas leur utilisation, comme le montre la figure 3.1.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, A, A, A, A \vdash G}\,Contr}{\Gamma, A, A, A \vdash G}\,Contr}{\Gamma, A, A \vdash G}\,Contr$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, A{\to}B \vdash A} \quad \Gamma, B \vdash G}{\Gamma, A{\to}B \vdash A}\,L{\to}^* \qquad \Gamma, B \vdash A}{\Gamma, A{\to}B \vdash A}\,L{\to}^* \qquad \Gamma, B \vdash A}{\Gamma, A{\to}B \vdash G}\,L{\to}^*$$

FIG. 3.1 – Dérivations infinies utilisant les règles *Contr* et $L{\to}^*$

Une solution évidente à ce problème est la définition de stratégies qui limitent l'utilisation de ces règles suivant la forme des formules A et B. L'introduction de ces stratégies pose alors le problème de la complétude : peut-on encore prouver les mêmes théorèmes tout en respectant une stratégie interdisant certaines dérivations ? Dans son article *Contraction-Free Sequent Calculi for Intuitionistic Logic* [Dyc92], Roy Dyckhoff apporta une solution différente[1] en proposant, dans le cadre de la logique propositionnelle, un calcul de séquents sans contraction appelé LJT, qui a la propriété de terminaison : toutes les dérivations de ce calcul sont finies.

Pour arriver à ce résultat, il décida de séparer la règle $L{\to}$ en plusieurs sous-cas selon la forme de la sous-formule A de la formule principale $A{\to}B$ (voir au bas de la figure 3.3). Par ce moyen, il réussit à obtenir un système complet sans avoir recours à la règle $L{\to}^*$. Ce système est présenté à la figure 3.3 dans sa version mono-successeur.

Dyckhoff établit l'admissibilité[2] de la contraction dans LJT et l'équivalence entre la dérivabilité dans LJ et celle dans LJT. Cependant, il ne donne de preuve de l'élimination des coupures dans LJT que par le moyen d'une traduction dans LJ sans coupures puis d'une traduction inverse de LJ vers LJT comme indiqué dans la figure 3.2.

Le calcul LJT mono-successeur (ou *G4ip*) a été utilisée par César Muñoz dans l'assistant de preuve Coq pour implanter un système de recherche de preuves : la tactique `tauto`. Cette procédure recherche en profondeur parmi toutes les dérivations possibles, avec pour stratégie de commencer par les règles inversibles. Cette tactique est aussi utilisée par la

[1] Dyckhoff se fonde sur les travaux de Vorobjev [Vor58] et Hudelmaier [Hud93].

[2] cf. définition 1.14.

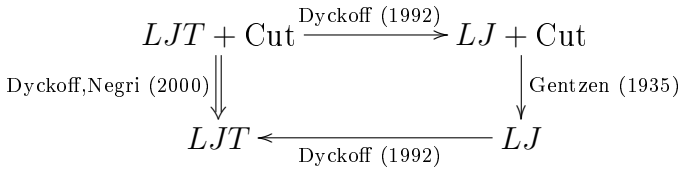


Fig. 3.2 – Preuves d'élimination des coupures pour LJT

$$\frac{}{\Gamma, P \vdash P}\ Ax \qquad \frac{}{\Gamma, \bot \vdash G}\ L\bot \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ R\to$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ R\wedge \qquad \frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ L\wedge$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ R\vee_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ R\vee_2 \qquad \frac{\Gamma, A \vdash G \quad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ L\vee$$

$$\frac{\Gamma, P, B \vdash G}{\Gamma, P, P \to B \vdash G}\ La\to \qquad \frac{\Gamma, B \to C, A \vdash B \quad \Gamma, C \vdash G}{\Gamma, (A \to B) \to C \vdash G}\ L\to\to$$

$$\frac{\Gamma, A \to B \to C \vdash G}{\Gamma, (A \wedge B) \to C \vdash G}\ L\wedge\to \qquad \frac{\Gamma, A \to C, B \to C \vdash G}{\Gamma, A \vee B \to C \vdash G}\ L\vee\to$$

Fig. 3.3 – Le système LJT

procédure de simplification `intuition` (J. Courant, P. Corbineau) en coupant les parties non-inversibles des branches de calcul qui échouent, et en laissant les buts correspondants à l'utilisateur. Une autre possibilité offerte par `intuition` est l'appel d'une tactique arbitraire comme solveur pour les buts pour lesquels aucune règle propositionnelle ne peut être appliquée. Ces tactiques ont apporté une réponse satisfaisante pour la logique intuitionniste propositionnelle, aussi a-t-on cherché à étendre leur portée aux quantificateurs du premier ordre.

Deux tentatives d'automatisation du calcul des prédicats dans Coq ont eu lieu précédemment : la première fut l'implantation d'une procédure de décision pour le Calcul Direct des Prédicats DPC [KW84, BK92], un fragment linéaire du calcul des prédicats, qui donna la tactique `linear` implantée par Jean-Christophe Filliâtre dans des versions précédentes de Coq. Cette tactique a ensuite été abandonnée à cause du manque d'expressivité du calcul DPC.

La seconde tentative fut l'intégration par Huang du module `Jprover` [SLKN01], partie du système Nuprl [Kre02] dans Coq. Le module `Jprover` est constitué d'un prouveur par tableaux pour la logique classique auquel est associé un solveur de contraintes destiné à restreindre les preuves trouvées aux seules preuves intuitionnistes.

De même que `linear`, cette tactique se comporte comme une boîte noire qui construit directement la preuve complète. Cette opacité de fonctionnement empêche d'en faire une procédure de simplification de buts. De plus, l'implantation est restreinte aux connecteurs logiques standard alors que même `tauto` traite certaines extensions comme la disjonction informative `sumbool`. Il lui manque également un bon traitement des variables existentielles implicites qui apparaissent parfois dans des preuves de propriétés telles que $(\forall x.Px) \rightarrow (\exists x.Px)$.

Selon nous, ces tentatives ne sont pas très satisfaisantes, une approche plus naturelle serait d'étendre directement le calcul LJT au premier ordre. Dans un article postérieur au précédent [DN00], coécrit avec Sara Negri, Roy Dyckoff parvint à donner une preuve directe de l'élimination des coupures pour LJT, auquel Troelstra et Schwichtenberg, dans leur ouvrage de référence *Basic Proof Theory* [TS96], ont donné le nom $G4ip$, en référence au calcul $G3$ de Kleene [Kle52]. Dans cet article, il traite également le cas du calcul $G4i$, l'extension au premier ordre de $G4ip$, dans les versions mono- et multi-successeurs. Bien évidemment ce calcul ne peut avoir la propriété de terminaison à cause de l'indécidabilité de la logique du premier ordre : le système $G4i$ contient l'instance $L\forall\rightarrow$ de $L\rightarrow^*$ où A est de la forme $\forall x.P$, et la règle $L\forall$ qui est l'analogue, pour le quantificateur $\forall$, de la règle $L\rightarrow^*$:

$$\frac{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A \quad \Gamma, B \vdash G}{\Gamma, (\forall x.A)\rightarrow B \vdash G}\, L\forall\rightarrow \qquad \frac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \forall x.A \vdash G}\, L\forall$$

Cette extension au premier ordre est bien celle qui nous intéresse, mais pas directement car elle n'utilise pas le caractère inductif des connecteurs logiques de Coq. Les langages de propositions logiques habituels, comme celui décrit au chapitre 1.7, utilisent presque tous des connecteurs logiques tels que $\wedge$, $\vee$, $\rightarrow$. Les travaux sur la logique classique s'appuient notamment sur les formes normales conjonctives et disjonctives et transforment $A\rightarrow B$ en $\neg A \vee B$. À l'inverse, en logique intuitionniste, l'implication $\rightarrow$ et la quantification universelle $\forall$ jouent un rôle crucial. Ceci est évident lorsque l'on examine les différences entre les modèles booléens de la logique classique et la sémantique de Kripke de la logique intuitionniste : la notion de noeud de structure de Kripke (monde de Kripke) n'intervient que dans la sémantique de $\rightarrow$ et $\forall$, le reste des contraintes ne font d'un modèle de Kripke qu'un simple produit cartésien de modèles booléens.

De plus, les opérateurs $\rightarrow$ et $\forall$ ont des interprétations naturelles par l'isomorphisme de Curry-Howard-deBruijn [How80] : $\rightarrow$ et $\forall$ sont les équivalents logiques des types des λ-abstractions non-dépendantes et dépendantes alors que l'extension de l'isomorphisme de Curry-Howard aux connecteurs $\wedge$ et $\vee$ nécessite l'adjonction de constructions algébriques au λ-calcul (les paires et les sommes disjointes) ainsi que celle des constructeurs et destructeurs correspondants, qui permettent d'exprimer les règles d'élimination et d'introduction.

Le Calcul des Constructions Inductives suit ce principe : les symboles $\rightarrow$ et $\forall$ sont des constructions primitives et l'utilisateur peut, à son gré, définir des constructions inductives de complexité arbitraire malgré certaines restrictions destinées à obtenir un calcul cohérent. Les connecteurs logiques habituels sont définis comme des types inductifs, et les règles d'introduction et d'élimination s'expriment de façon uniforme par rapport à leur définition.

Ceci met à la disposition de l'utilisateur un langage logique qui, s'il ne gagne pas en expressivité, peut être étendu par des connecteurs logiques plus complexes qui auront leurs propres règles d'introduction et d'élimination. D'une part, les nouvelles règles d'inférence sont plus concises et les preuves en deviennent plus claires et plus courtes que celles qui utilisent des connecteurs simples imbriqués. D'autre part, les propriétés du système formel peuvent être établies une fois pour toutes en ce qui concerne le schéma de règles pour les connecteurs inductifs, ce qui donne des preuves plus courtes et des systèmes d'inférence plus concis.

Le travail réalisé dans ce chapitre est l'adaptation du système $G4i$ afin qu'il puisse être utilisé de façon naturelle pour chercher des preuves intuitionnistes du premier ordre dans Coq, où les symboles logiques $\wedge,\vee,\bot$ et le quantificateur existentiel $\exists$ sont des définitions inductives. Nous proposons en conséquence une variante du calcul $G4i$, appelée $LJTI$ où les constructions logiques primitives sont uniquement $\forall,\rightarrow$ et les définitions inductives, *excluant toutefois la récursivité*, tandis que les autres connecteurs sont traités comme des cas particuliers de définitions inductives.

3.2 Le système $LJTI$

3.2.1 Les connecteurs inductifs

Dans la suite du document, nous aurons à traiter des expressions de taille fixée *a priori* mais arbitraire, c'est pourquoi nous recourrons de façon massive aux notations suivantes :

$$
\begin{aligned}
\mathbf{H_i} &\equiv H_{i,1}, \ldots, H_{i,p} \\
\mathbf{H_i}{\rightarrow}X &\equiv H_{i,1}{\rightarrow}(\ldots{\rightarrow}(H_{i,p}{\rightarrow}X)) \\
\mathbf{x} &\equiv x_1, \ldots, x_p \\
\forall \mathbf{y_i}.X &\equiv \forall y_{i,1} \ldots \forall y_{i,p}.X \\
\{H_i\}_i &\equiv H_1, \ldots, H_p \\
\frac{\{\mathcal{S}_i\}_i}{\mathcal{S}} &\equiv \frac{\mathcal{S}_1 \quad \ldots \quad \mathcal{S}_p}{\mathcal{S}}
\end{aligned}
$$

Il est à noter que l'expression $\mathbf{H_i}$ change de sens quand elle est située à gauche d'une flèche, de même que l'expression $\mathbf{x}$ lorsqu'elle est précédée de $\forall$. Les formules que nous allons utiliser ici sont définies de manière mutuellement récursive avec les familles inductives. Nous supposons fixée une signature Σ pour les sortes, les symboles de fonction et les symboles de prédicat. Les termes et les formules atomiques gardent donc leur définition habituelle. Nous allons définir d'abord les familles inductives et ensuite les formules.

DÉFINITION 3.1 (FAMILLE INDUCTIVE)

$$\begin{aligned} \mathit{type} &:= (\mathit{sorte}, \ldots, \mathit{sorte}) \rightarrow * \\ &\mid (\mathit{sorte}, \ldots, \mathit{sorte}) \rightarrow \mathit{sorte} \\ \mathit{constructeur} &:= \mathcal{C} : \mathit{clôture} \\ \mathit{clôture} &:= \forall x. \mathit{clôture} \\ &\mid \mathit{clause} \\ \mathit{clause} &:= \mathit{formule} \rightarrow \mathit{clause} \\ &\mid \square \\ \mathit{définition} &:= \mathit{inductif}(X_1 : \mathit{type}, \ldots, X_n : \mathit{type})\mathit{constructeur}^* \end{aligned}$$

Les formules présentes dans les clauses définissant les constructeurs sont appelées hypothèses de ce constructeur et sont des formules construites sur la signature Σ augmentée des paramètres de la famille inductive.

L'arité des familles inductives, de la forme $I : Ind(p_1 : \tau_1, \ldots, p_n : \tau_n)$, est alors ajoutée à la signature Σ. Grâce à ces familles inductives, nous allons pouvoir construire de nouvelles formules, dont la définition est donnée ci-dessous.

DÉFINITION 3.2 (FORMULES)
Nous partons de la notion de formule définie en 1.1, dont nous enlevons les constructions $\wedge$, $\vee$, $\bot$ et $\exists$ et à laquelle nous ajoutons les formules inductives :

$$\begin{aligned} \mathit{formule} &:= \forall x. \mathit{formule} \\ &\mid \mathit{formule} \rightarrow \mathit{formule} \\ &\mid \textit{formule atomique} \\ &\mid I(\mathit{param}, \ldots, \mathit{param}) \\ \mathit{param} &:= \lambda x_1 \ldots x_n. \mathit{terme} \\ &\mid \lambda x_1 \ldots x_n. \mathit{formule} \end{aligned}$$

Les formules inductives sont construites par l'application d'un connecteur inductif à un nombre fixe de paramètres, qui sont eux-mêmes des contextes de termes ou de formules. Nous matérialisons les places vides de ces contextes en les représentant sous forme de λ-abstractions.

Les formules définissant les constructeurs doivent être closes. De plus les paramètres des formules inductives doivent avoir un type correspondant à celui des paramètres formels. L'ensemble des contraintes de bonne formation des signatures, formules, familles inductives est résumé dans la figure 3.4.

Les contraintes de bonne formation de la signature Σ induisent un ordre total $\prec_\Sigma$ sur les familles inductives de cette signature. Cet ordre est défini par $I_1 \prec_\Sigma I_2$ si, et seulement si I_1 apparaît dans la signature au moment de l'ajout de I_2. On constate ainsi que si I_1 apparaît dans la clause de définition d'un constructeur de I_2, alors nécessairement $I_1 \prec_\Sigma I_2$. Dans le Calcul des Constructions Inductives, les définitions peuvent être récursives ou mutuellement récursives si les occurrences de l'inductif satisfont la condition de stricte positivité.

Ici, nous interdisons purement et simplement la récursivité parce que d'une part cette hypothèse est nécessaire pour assurer la terminaison du processus d'élimination des coupures, et que d'autre part notre propos ici n'est pas de traiter le raisonnement par récurrence.

Définition 3.3 (variables libres)
L'ensemble des variables libres dans un objet est défini par les équations suivantes :

$$\begin{aligned} Fv(\Box) &= \emptyset \\ Fv(\lambda x_1 \ldots x_n.C) &= Fv(C) \setminus \{x_1, \ldots, x_n\} \\ Fv(\forall x.F) &= Fv(F) \setminus \{x\} \\ Fv(F_1 \rightarrow F_2) &= Fv(F_1) \cup Fv(F_2) \\ Fv(P(t_1, \ldots, t_n) &= \bigcup_{i=1\ldots n} Fv(t_i) \\ Fv(I(p_1, \ldots, p_n) &= \bigcup_{i=1\ldots n} Fv(p_i) \end{aligned}$$

Grâce à ces définitions, nous pouvons définir les familles inductives correspondant aux connecteurs habituels (voir figure 3.5). Les familles inductives $\wedge$ et $\vee$ ont deux paramètres propositionnels ; $\top$ et $\bot$ n'en ont aucun, et $\exists_s$ en a un d'arité $s \rightarrow *$. Nous donnons ici les dérivations de bonne formation pour $\bot$ et $\exists_s$.

Exemple 3.4 (bonne formation de $\bot$ et $\exists$)

- $\bot$ *est bien formé :* $\dfrac{}{\emptyset \vdash_{BF} \bot()\{\}}$
- $\exists$ *est bien formé :*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{x:s\}, \{P:s\rightarrow *\} \vdash_{BF} \Box : clause} \quad \dfrac{\dfrac{}{\{x:s\}, \{P:s\rightarrow *\} \vdash_{BF} x}}{\{x:s\}, \{P:s\rightarrow *\} \vdash_{BF} P(x) : *}}{\{x:s\}, \{P:s\rightarrow *\} \vdash_{BF} P(x)\rightarrow\Box : clause}}{\emptyset \vdash_{BF} \forall_s x.P(x)\rightarrow\Box : constr(P:s\rightarrow *)}}{\emptyset \vdash_{BF} \exists(P:s\rightarrow *)\{témoin : \forall x.P(x)\rightarrow\Box\}}$$

On peut aussi définir des familles plus exotiques : beaucoup de prédicats peuvent être définis par des familles inductives non récursives. Si nous voulons dire qu'une proposition A satisfait la propriété de tiers exclus, nous allons définir :

$$(\text{Dec}, (A : *), \{Vrai : A\rightarrow\Box;\ Faux : (A\rightarrow\bot)\rightarrow\Box\})$$

Dec est bien formée dans la signature $\Sigma = \{\bot : Ind()\}$, comme en témoigne la dérivation de la figure 3.6.

Un autre exemple pourrait être un prédicat $\text{Const}_{s,s'}(f, P)$ signifiant « La fonction f est constante sur le domaine $\{x \in s | Px\}$ ».

$$\begin{aligned} (\text{Const}_{s,s'}, \quad & (f : s\rightarrow s', P : s\rightarrow *), \\ & \{Empty\_domain : (\forall_s x.P(x)\rightarrow\bot)\rightarrow\Box; \\ & \ Unique\_value : \forall_s y.P(y)\rightarrow(\forall x.P(x)\rightarrow f(x) = f(y))\rightarrow\Box\}) \end{aligned}$$

signature

$$\frac{}{\vdash_{BF} \emptyset}$$

$$\frac{\vdash_{BF} \Sigma}{\vdash_{BF} \Sigma \cup \{f : (s_1, \ldots, s_n) \rightarrow s\}}\ f \notin \Sigma$$

$$\frac{\vdash_{BF} \Sigma}{\vdash_{BF} \Sigma \cup \{P : (s_1, \ldots, s_n) \rightarrow *\}}\ P \notin \Sigma$$

$$\frac{\vdash_{BF} \Sigma \quad \Sigma \vdash_{BF} I(p_1 : \tau_1, \ldots, p_n : \tau_n)\{\mathcal{C}_1 : F_1; \ldots; \mathcal{C}_p : F_p\}}{\vdash_{BF} \Sigma \cup \{I : Ind(\tau_1, \ldots, \tau_n)\}}\ I \notin \Sigma$$

terme

$$\frac{}{\Xi, \Sigma \vdash_{BF} x : s}\ x : s \in \Xi$$

$$\frac{\Xi, \Sigma \vdash_{BF} t_1 : s_1 \quad \ldots \quad \Xi, \Sigma \vdash_{BF} t_n : s_n}{\Xi, \Sigma \vdash_{BF} f(t_1, \ldots, t_n) : s}\ f : (s_1, \ldots, s_n) \rightarrow s \in \Sigma$$

contexte

$$\frac{\Xi \cup \{x_1 : s_1, \ldots, x_n : s_n\}, \Sigma \vdash_{BF} t : \tau}{\Xi, \Sigma \vdash_{BF} \lambda x_1 \ldots x_n.t : (s_1, \ldots, s_n) \rightarrow \tau}\ \tau \in \mathcal{S} \cup \{*\}$$

formule

$$\frac{\Xi, \Sigma \vdash_{BF} t_1 : s_1 \quad \ldots \quad \Xi, \Sigma \vdash_{BF} t_n : s_n}{\Xi, \Sigma \vdash_{BF} P(t_1, \ldots, t_n) : *}\ P : (s_1, \ldots, s_n) \rightarrow * \in \Sigma$$

$$\frac{\Xi \cup \{x : s\}, \Sigma \vdash_{BF} F : *}{\Xi, \Sigma \vdash_{BF} \forall x.F : *}$$

$$\frac{\Xi, \Sigma \vdash_{BF} F_1 : * \quad \Xi, \Sigma \vdash_{BF} F_1 : *}{\Xi, \Sigma \vdash_{BF} F_1 \rightarrow F_2 : *}$$

$$\frac{\Xi, \Sigma \vdash_{BF} p_1 : \tau_1 \quad \ldots \quad \Xi, \Sigma \vdash_{BF} p_n : \tau_n}{\Xi, \Sigma \vdash_{BF} I(p_1, \ldots, p_n) : *}\ I : Ind(\tau_1, \ldots, \tau_n)$$

clause de définition de constructeur

$$\frac{}{\Xi, \Sigma \vdash_{BF} \square : clause}$$

$$\frac{\Xi, \Sigma \vdash_{BF} A : clause \quad \Xi, \Sigma \vdash_{BF} H : *}{\Xi, \Sigma \vdash_{BF} H \rightarrow A : clause}$$

$$\frac{\{x_1 : s_1, \ldots, x_n : s_n\}, \Sigma \cup \{p_1 : \tau_1, \ldots, p_n : \tau_n\} \vdash_{BF} F : clause}{\Sigma \vdash_{BF} \forall x_1, \ldots, x_n.F : constr(p_1 : \tau_1, \ldots, p_n : \tau_n)}$$

famille inductive

$$\frac{\Sigma \vdash_{BF} F_1 : constr(p_1 : \tau_1, \ldots, p_n : \tau_n) \quad \ldots \quad \Sigma \vdash_{BF} F_p : constr(p_1 : \tau_1, \ldots, p_n : \tau_n)}{\Sigma \vdash_{BF} I(p_1 : \tau_1, \ldots, p_n : \tau_n)\{\mathcal{C}_1 : F_1; \ldots; \mathcal{C}_p : F_p\}}$$

avec $\mathcal{C}_1, \ldots, \mathcal{C}_n$ distincts deux à deux

FIG. 3.4 – Règles de bonne formation pour les inductifs

$$(\wedge, (A:*, B:*), \{\texttt{pair} : A{\to}B{\to}\Box\}) : Ind(*,*)$$
$$(\vee, (A:*, B:*), \{\texttt{or\_introl} : A{\to}\Box;\ \texttt{or\_intror} : B{\to}\Box\}) : Ind(*,*)$$
$$(\bot, (), \{\}) : Ind()$$
$$(\top, (), \{I:\Box\}) : Ind()$$
$$(\exists_s, (P : s{\to}*), \{\texttt{ex\_intro} : \forall_s y.P(y){\to}\Box\}) : Ind(s{\to}*)$$

Fig. 3.5 – Connecteurs logiques habituels sous forme de familles inductives

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset,\Sigma;A:*\vdash_{BF} A:*}\quad\dfrac{}{\emptyset,\Sigma;A:*\vdash_{BF}\bot:*}}{\emptyset,\Sigma;A:*\vdash_{BF} A{\to}\bot:*}\ \vdots}{\emptyset,\Sigma;A:*\vdash_{BF}\Box:clause}}{\emptyset,\Sigma;A:*\vdash_{BF}(A{\to}\bot){\to}\Box:clause}}{\Sigma\vdash_{BF}(A{\to}\bot){\to}\Box:constr(A:*)}\ \vdots}{\dfrac{\dfrac{\dfrac{}{\emptyset,\Sigma;A:*\vdash_{BF}\Box:clause}\quad\dfrac{}{\emptyset,\Sigma;A:*\vdash_{BF} A:*}}{\emptyset,\Sigma;A:*\vdash_{BF} A{\to}\Box:clause}}{\Sigma\vdash_{BF} A{\to}\Box:constr(A:*)}}{\Sigma\vdash_{BF}\mathrm{Dec}(A:*)\{\mathrm{Vrai}:A{\to}\Box;\mathrm{Faux}:(A{\to}\bot){\to}\Box\}}$$

Fig. 3.6 – Bonne formation de Dec

De manière générale, une famille inductive $I(p_1:\tau_1,\ldots,p_n:\tau_n)\{\mathcal{C}_1:F_1,\ldots,\mathcal{C}_p:F_p\}$ avec :

$$F_i \equiv \forall_{\mathbf{s_i}}\mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}\Box$$

a une signification intuitive qui s'exprime avec les connecteurs habituels par l'équivalence logique :

$$I(\mathbf{p}) \Leftrightarrow \bigvee_i (\exists_{\mathbf{s_i}}\mathbf{y_i}.\bigwedge_j H_{i,j}(\mathbf{p},\mathbf{y_i}))$$

Définition 3.5 (substitution)
La définition des substitutions (1.4) est étendue aux paramètres et aux formules inductives comme suit :

paramètre

$(\lambda x.C)\sigma = \lambda y.(C\rho\sigma)$ *avec* $\rho = \{y/x\}$ *et* $y \notin Fv(C)\cup Fv(\sigma)\cup Dom(\sigma)$

formule

$I(p_1,\ldots,p_n)\sigma = I(p_1\sigma,\ldots,p_n\sigma)$

Hypothèses de constructeurs

DÉFINITION 3.6 (HYPOTHÈSES D'UN CONSTRUCTEUR INSTANCIÉ)
Soit la famille inductive $I(p_1 : \tau_1, \ldots, p_n : \tau_n)\{\mathcal{C}_1 : F_1, \ldots, \mathcal{C}_p : F_p\}$, *avec*

$$F_i \equiv \forall_{s_1} y_{i,1}, \ldots, \forall_{s_{k_i}} y_{i,k_i}.H_{i,1} \rightarrow \ldots \rightarrow H_{i,h_i} \rightarrow \Box$$

Soit $X_1 : \tau_1, \ldots, X_n : \tau_n$ *une liste de paramètres. La clause de définition du constructeur* $\mathcal{C}_i$ *instancié par les paramètres* $X_1, \ldots, X_n$ *est définie par :*

$$F_i(\mathbf{X}) \equiv \forall_{s_1} z_{i,1}, \ldots, \forall_{s_{k_i}} z_{i,k_i}.inst(H_{i,1}, \mathbf{X})\rho \rightarrow \ldots \rightarrow inst(H_{i,h_i}, \mathbf{X})\rho \rightarrow \Box$$

Où $\rho = \{z_1/y_1; \ldots; z_{k_i}/y_{k_i}\}$ *avec* $z_1, \ldots, z_{k_i}$ *des variables distinctes et n'apparaissant pas dans* $\bigcup_{i=1\ldots n} Fv(X_i)$, *et où l'opérateur inst est défini par les équations suivantes.*

terme

$inst(x, \mathbf{X}) = x$
$inst(f(t_1, \ldots, t_m)) = u\{inst(t_1, \mathbf{X})/x_1; \ldots; inst(t_m, \mathbf{X})/x_m\}$
si $f = p_i$ *et* $X_i = \lambda x_1 \ldots x_m.u$
$inst(f(t_1, \ldots, t_m)) = f(inst(t_1, \mathbf{X}); \ldots; inst(t_m, \mathbf{X}))$
sinon

paramètre

$inst(\lambda x_1 \ldots x_m.C, \mathbf{X}) = \lambda y_1 \ldots y_m.inst(C\rho, \mathbf{X})$
avec $\rho = \{y_1/x_1; \ldots; y_m/x_m\}$
et $y_1, \ldots, y_m \notin Fv(C) \cup \bigcup_{i=1\ldots n} Fv(X_i)$

formule

$inst(\forall x.F, \mathbf{X}) = \forall y.inst(F\rho, \mathbf{X})$
avec $\rho = \{y/x\}$ *et* $y \notin Fv(F) \cup \bigcup_{i=1..n} Fv(X_i)$
$inst(F_1 \rightarrow F_2, \mathbf{X}) = inst(F_1, \mathbf{X}) \rightarrow inst(F_2, \mathbf{X})$
$inst(I(q_1, \ldots, q_m), \mathbf{X}) = I(inst(q_1, \mathbf{X}), \ldots, inst(q_m, \mathbf{X}))$
$inst(A(t_1, \ldots, t_m), \mathbf{X}) = G\{inst(t_1, \mathbf{X})/x_1; \ldots; inst(t_m, \mathbf{X})/x_m\}$
si $A = p_i$ *et* $X_k = \lambda x_1 \ldots x_n.G$
$inst(A(t_1, \ldots, t_n), \mathbf{X}) = A(inst(t_1, \mathbf{X}); \ldots; inst(t_m, \mathbf{X}))$
sinon

L'ensemble des hypothèses du constructeur $\mathcal{C}_i$ *instancié par les paramètres* $X_1, \ldots, X_n$ *et les termes* $t_1, \ldots, t_{k_i}$ *est défini comme :*

$$\mathrm{Hyps}(\mathcal{C}_i, \mathbf{X}, \mathbf{u}) = \{inst(H_{i,1}\rho, \mathbf{X})\sigma, \ldots, inst(H_{i,h_i}\rho, \mathbf{X})\sigma\}$$

avec $\sigma = \{t_1/z_1; \ldots; t_{k_i}/z_{k_i}\}$. *Dans la suite, nous utiliserons la notation* $H_{i,j}(\mathbf{X}, \mathbf{u})$ *pour désigner la formule* $inst(H_{i,j}\rho, \mathbf{X})\sigma$.

$$\frac{}{\Gamma, P \vdash P}\ Ax$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\ R{\to} \qquad \frac{\Gamma, P, B \vdash G}{\Gamma, P, P{\to}B \vdash G}\ La{\to}$$

$$\frac{\Gamma, A, B{\to}C \vdash B \quad \Gamma, C \vdash G}{\Gamma, (A{\to}B){\to}C \vdash G}\ L{\to}{\to}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ R\forall \qquad \frac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \forall x.A \vdash G}\ L\forall$$

$$\frac{\Gamma, (\forall x.A){\to}B \vdash \forall x.A \quad \Gamma, B \vdash G}{\Gamma, (\forall x.A){\to}B \vdash G}\ L\forall{\to}$$

$$\frac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_{j=1..h_i}}{\Gamma \vdash I(\mathbf{p})}\ RI_i \qquad \frac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_{i=1..n}}{\Gamma, I(\mathbf{p}) \vdash G}\ LI$$

$$\frac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B\}_{i=1..n} \vdash G}{\Gamma, I(\mathbf{p}){\to}B \vdash G}\ LI{\to}$$

Fig. 3.7 – Le calcul $LJTI$

Le calcul de séquents $LJTI$ est défini, à l'aide des familles inductives, dans la figure 3.7. Par ce moyen, nous obtenons un calcul où le nombre de règles est très réduit. Dans les règles Ax et $La{\to}$, P est une formule atomique. Dans les règles RI et LI, H_{ij} désigne une hypothèse du constructeur $\mathcal{C}_i$ de la famille inductive I. Dans la règle $LI{\to}$, la formule $\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B$ est la clause de définition du constructeur $\mathcal{C}_i$ instancié par $\mathbf{X}$, dans laquelle on remplace $\Box$ par la formule B. Dans les règles $R\forall$ et LI, les variables x et $\mathbf{y_i}$ sont choisies de sorte qu'elles n'apparaissent pas libres dans les conclusions respectives de ces règles.

Par exemple, si nous essayons d'appliquer les schémas de règles RI, LI et $LI{\to}$ au connecteur inductif $\bot$, nous obtenons les règles suivantes :

$$\textit{(pas de règle } R\bot\textit{)} \qquad \frac{}{\Gamma, \bot \vdash G}\ L\bot \qquad \frac{\Gamma \vdash G}{\Gamma, \bot{\to}A \vdash G}\ L\bot{\to}$$

Comme les familles inductives ont une règle d'introduction par constructeur, il n'y a pas de règle $R\bot$, ce qui était aussi le cas dans LJT. La règle $L\bot$ est la même que dans LJT, à part $L\bot{\to}$ qui est en fait une instance de la règle d'affaiblissement (voir section 1.2.2), qui est admissible dans $G4i$ (voir [DN00]). Pour les autres connecteurs habituels ($\wedge$, $\vee$, $\exists$), on obtient des règles identiques à celles de $G4i$.

Les règles pour Dec sont :

$$\frac{\Gamma \vdash A}{\Gamma \vdash \mathrm{Dec}(A)}\ R\mathrm{Dec}_1 \qquad \frac{\Gamma \vdash A{\to}\bot}{\Gamma \vdash \mathrm{Dec}(A)}\ R\mathrm{Dec}_2$$

$$\frac{\Gamma, A \vdash G \quad \Gamma, A{\rightarrow}\bot \vdash G}{\Gamma, \mathrm{Dec}(A) \vdash G}\ L\mathrm{Dec} \qquad \frac{\Gamma, A{\rightarrow}C, (A{\rightarrow}\bot){\rightarrow}C \vdash G}{\Gamma, \mathrm{Dec}(A){\rightarrow}C \vdash G}\ L\mathrm{Dec}{\rightarrow}$$

Pour Const, on obtient :

$$\frac{\Gamma \vdash \forall_s x, P(x){\rightarrow}\bot}{\Gamma \vdash \mathrm{Const}_{s,s'}(P,f)}\ R\mathrm{Const}_{s,s'1}$$

$$\frac{\Gamma \vdash P(t) \quad \Gamma \vdash \forall_s x.P(x){\rightarrow}f(x) = f(t)}{\Gamma \vdash \mathrm{Const}_{s,s'}(P,f)}\ R\mathrm{Const}_{s,s'2}$$

$$\frac{\Gamma, \forall_s x.P(x){\rightarrow}\bot \vdash G \quad \Gamma, P(y), \forall_s x.P(x){\rightarrow}f(x) = f(y) \vdash G}{\Gamma, \mathrm{Const}_{s,s'}(P,f) \vdash G}\ L\mathrm{Const}$$

$$\frac{\Gamma, (\forall_s x.P(x){\rightarrow}\bot){\rightarrow}A, \forall_s y, P(y){\rightarrow}(\forall_s x.P(x){\rightarrow}f(x) = f(y)){\rightarrow}A \vdash G}{\Gamma, \mathrm{Const}(P,f){\rightarrow}A \vdash G}\ L\mathrm{Const}_{s,s'}{\rightarrow}$$

Dans la règle $L\mathrm{Const}_{s,s'}$, x ne doit être libre ni dans Γ, ni dans G, ni dans P ou f.

Si nous avons choisi de travailler sur un calcul mono-successeur plutôt qu'avec un calcul multi-successeurs, c'est parce qu'une version acceptable de la règle RI aurait alors un nombre de prémisses égal au produit du nombre d'hypothèses de tous les constructeurs. Prenons l'exemple simple du connecteur ♣ qui signifie qu'il existe un objet satisfaisant deux des trois prédicats passés en paramètres :

$$\clubsuit(P,Q,R : s{\rightarrow}*)\{\clubsuit_1 : \forall x, Px{\rightarrow}Qx{\rightarrow}\Box, \clubsuit_2 : \forall x, Qx{\rightarrow}Rx{\rightarrow}\Box, \clubsuit_3 : \forall x, Rx{\rightarrow}Px{\rightarrow}\Box\}$$

Dans notre calcul $LJTI$ cette famille a trois règles d'introduction :

$$\frac{\Gamma \vdash Pt \quad \Gamma \vdash Qt}{\Gamma \vdash \clubsuit(P,Q,R)}\ R\clubsuit_1 \qquad \frac{\Gamma \vdash Qu \quad \Gamma \vdash Ru}{\Gamma \vdash \clubsuit(P,Q,R)}\ R\clubsuit_2 \qquad \frac{\Gamma \vdash Rv \quad \Gamma \vdash Pv}{\Gamma \vdash \clubsuit(P,Q,R)}\ R\clubsuit_3$$

Si nous voulons mettre ces trois règles en une seule, nous aurons alors une règle multi-successeurs à huit prémisses car il faut considérer toutes les combinaisons d'hypothèses possibles.

$$\frac{\begin{array}{c}\Gamma \vdash Pt, Qu, Rv, \Delta \quad \Gamma \vdash Qt, Qu, Rv, \Delta \quad \Gamma \vdash Pt, Ru, Rv, \Delta \quad \Gamma \vdash Qt, Ru, Rv, \Delta \\ \Gamma \vdash Pt, Qu, Pv, \Delta \quad \Gamma \vdash Qt, Qu, Pv, \Delta \quad \Gamma \vdash Pt, Ru, Pv, \Delta \quad \Gamma \vdash Qt, Ru, Pv, \Delta\end{array}}{\Gamma \vdash \clubsuit(P,Q,R), \Delta}\ R\clubsuit$$

Nous pensons qu'il est préférable de s'en tenir au calcul mono-successeur, car l'inconvénient d'avoir à choisir une des branches dans une disjonction, est moins désagréable que ce phénomène d'explosion combinatoire.

3.3 Propriétés de $LJTI$

Nous allons maintenant établir les propriétés structurelles de $LJTI$, notamment les propriétés d'élimination des contractions et des coupures. La démarche que nous suivons est similaire à celle de l'article [DN00], avec globalement moins de sous-cas car $LJTI$ contient moins de règles que $G4i$.

Dans nos démonstrations, nous procéderons souvent en explicitant des commutations de règles. Pour matérialiser l'utilisation des hypothèses de récurrence, nous marquerons les étapes en question par une double barre, avec l'indication Ind ou le nom de la règle dont nous prouvons l'admissibilité. Les règles dont l'admissibilité est déjà prouvée seront parfois dénotées par le numéro du lemme de la preuve d'admissibilité.

3.3.1 Lemmes d'inversion

Nous commençons par prouver que le calcul de séquents $LJTI$ est stable par substitution, propriété que l'on peut illustrer à l'aide de l'exemple suivant : prenons le séquent :

$$Homme(x) \vdash Mortel(x)$$

« Si x est un homme alors x est mortel. »

Le lemme de substitution nous garantit qu'il existe une dérivation d'un séquent plus précis en utilisant la substitution $\{x/Socrate\}$:

$$Homme(Socrate) \vdash Mortel(Socrate)$$

« Si Socrate est un homme alors Socrate est mortel. »

Ce lemme est très utile car il va nous permettre, en particulier, de renommer les variables libres dans les prémisses des règles à condition de variable fraîche ($R\forall$ et LI), ce afin d'éviter des conflits avec de nouvelles variables introduites notamment par l'utilisation de la règle d'affaiblissement.

LEMME 3.7 (SUBSTITUTION)
Pour toute substitution σ, la règle suivante est fortement admissible dans $LJTI$.

$$\frac{\Gamma \vdash G}{(\Gamma \vdash G)\sigma}\ Subst_\sigma$$

Démonstration : Nous allons prouver par récurrence sur la hauteur de l'arbre de dérivation de la prémisse que pour toute substitution σ, la règle $Subst_\sigma$ est fortement admissible.

– Pour les règles simples, on effectue les transformations suivantes :

$$\frac{}{\Gamma, P \vdash P}\ Ax \quad \Longrightarrow \quad \frac{}{\Gamma\sigma, P\sigma \vdash P\sigma}\ Ax$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\, R\to \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, A \vdash B}{\Gamma\sigma, A\sigma \vdash B\sigma}\, Subst_\sigma}{\Gamma\sigma \vdash A\sigma \to B\sigma}\, R\to$$

$$\frac{\Gamma, P, B \vdash G}{\Gamma, P, P \to B \vdash G}\, La\to \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, P, B \vdash G}{\Gamma\sigma, P\sigma, B\sigma \vdash G\sigma}\, Subst_\sigma}{\Gamma\sigma, P\sigma, P\sigma \to B\sigma \vdash G\sigma}\, La\to$$

$$\frac{\Gamma, A, B \to C \vdash B \quad \Gamma, C \vdash G}{\Gamma, (A \to B) \to C \vdash G}\, L\to\to$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, A, B \to C \vdash B}{\Gamma\sigma, A\sigma, B\sigma \to C\sigma \vdash B\sigma}\, Subst_\sigma \quad \dfrac{\Gamma, C \vdash G}{\Gamma\sigma, C\sigma \vdash G\sigma}\, Subst_\sigma}{\Gamma\sigma, (A\sigma \to B\sigma) \to C\sigma \vdash G\sigma}\, L\to\to$$

- Pour la règle $R\forall$, soit z une variable indépendante de σ qui n'est pas libre dans Γ. Soit $\rho = \{z/x\}$. Comme x n'est pas libre dans Γ, on a $\Gamma\rho = \Gamma$, et donc $\Gamma\rho\sigma = \Gamma\sigma$. De plus, on a $\forall z.(A\rho\sigma) = (\forall x.A)\sigma$. On effectue donc la transformation suivante :

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\, R\forall \quad \Longrightarrow \quad \begin{array}{c} \dfrac{\dfrac{\Gamma \vdash A}{\Gamma\rho\sigma \vdash A\rho\sigma}\, Subst_{\rho\sigma}}{\Gamma\rho\sigma \vdash \forall z.(A\rho\sigma)}\, R\forall \\ = \\ \Gamma\sigma \vdash (\forall x.A)\sigma \end{array}$$

  L'application de la règle $R\forall$ reste valide car z ne peut pas être libre dans $\Gamma\rho\sigma$ par construction.
- Pour la règle $L\forall$, soit z une variable indépendante de σ qui n'est libre ni dans Γ ni dans G. Soit $\rho = \{z/x\}$.

$$\frac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \forall x.A \vdash G}\, L\forall$$

$$\Downarrow$$

$$\begin{array}{c} \dfrac{\dfrac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma\sigma, (\forall x.A)\sigma, (A\{t/x\})\sigma \vdash G\sigma}\, Subst_\sigma}{\Gamma\sigma, \forall z.(A\rho\sigma), A\rho\sigma\{t\sigma/z\} \vdash G\sigma}\, L\forall \\ = \\ \Gamma\sigma, \forall z.(A\rho\sigma) \vdash G\sigma \\ = \\ \Gamma\sigma, (\forall x.A)\sigma \vdash G\sigma \end{array}$$

– $L\forall\rightarrow$:

$$\frac{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A \quad \Gamma, B \vdash G}{\Gamma, (\forall x.A)\rightarrow B \vdash G}\ L\forall\rightarrow$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A}{\Gamma\sigma, (\forall x.A)\sigma\rightarrow B\sigma \vdash (\forall x.A)\sigma}\ Subst_\sigma \quad \dfrac{\Gamma, B \vdash G}{\Gamma\sigma, B\sigma \vdash G\sigma}\ Subst_\sigma}{\Gamma\sigma, (\forall x.A)\sigma\rightarrow B\sigma \vdash G\sigma}\ L\forall\rightarrow$$

– RI_i :

$$\frac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j}{\Gamma \vdash I(\mathbf{p})}\ RI_i \quad \Longrightarrow \quad \frac{\dfrac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j}{\{\Gamma\sigma \vdash (H_{i,j}(\mathbf{p}, \mathbf{t_i}))\sigma\}_j}\ Subst_\sigma}{\{\Gamma\sigma \vdash H_{i,j}(\mathbf{p}\sigma, \mathbf{t_i}\sigma)\}_j}\ RI_i$$

$$=$$

$$\Gamma\sigma \vdash I(\mathbf{p}\sigma)$$

– LI : Pour tout i, on construit le renommage $\rho_i = \{\mathbf{z_i}/\mathbf{y_i}\}$, où les variables $\mathbf{z_i}$ sont indépendantes de σ, deux à deux distinctes et n'apparaissent pas libres dans Γ, $\mathbf{p}$ ni dans G.

$$\frac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_i}{\Gamma, I(\mathbf{p}) \vdash G}\ LI$$

$$\Downarrow$$

$$\frac{\dfrac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_i}{\{\Gamma\rho_i\sigma, (\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}))\rho_i\sigma \vdash G\rho_i\sigma\}_i}\ Subst_{\rho_i\sigma}}{\{\Gamma\sigma, (\mathbf{H_i}(\mathbf{p}\sigma, \mathbf{z_i})) \vdash G\sigma\}_i}\ LI$$

$$=$$

$$\Gamma\sigma, I(\mathbf{p}\sigma) \vdash G\sigma$$

– $LI\rightarrow$: Pour tout i, soient $\mathbf{z_i}$ des variables indépendantes de σ, deux à deux distinctes, et n'apparaissant libres ni dans Γ ou $\mathbf{p}$, ni dans G ou B.

$$\frac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i \vdash G}{\Gamma, I(\mathbf{p})\rightarrow B \vdash G}\ LI\rightarrow$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i \vdash G}{\Gamma\sigma, (\{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i)\sigma \vdash G\sigma}\ Subst_\sigma}{\Gamma\sigma, \{\forall \mathbf{z_i}.\mathbf{H_i}(\mathbf{p}\sigma, \mathbf{z_i})\rightarrow B\sigma\}_i \vdash G}\ LI\rightarrow$$

$$=$$

$$\Gamma\sigma, I(\mathbf{p}\sigma)\rightarrow B\sigma \vdash G\sigma$$

Ce qui clôt la preuve par récurrence. □

On remarque que cette preuve est constructive et nous donne donc un algorithme permettant de calculer la dérivation de $(\Gamma \vdash G)\sigma$ à partir de celle de $\Gamma \vdash G$.

L'étape suivante concerne la règle d'affaiblissement (*Weakening*). Cette règle permet de dire que l'ajout d'hypothèses supplémentaires ne peut rendre une propriété invalide : elle permet donc de transformer une preuve de :

$$Homme(x) \vdash Mortel(x)$$

« Si x est un homme, alors x est mortel. »

en une preuve de :

$$Homme(x), Plate(Terre) \vdash Mortel(x)$$

« Si x est un homme et si la terre est plate, alors x est mortel. »

THÉORÈME 3.8
La règle d'affaiblissement W ci-dessous est fortement admissible dans LJTI.

$$\frac{\Gamma \vdash G}{\Gamma, \Gamma' \vdash G}\, W$$

Démonstration : Par récurrence sur la hauteur de l'arbre de dérivation :

– Pour les règles simples, on effectue les transformations suivantes :

$$\frac{}{\Gamma, P \vdash P}\, Ax \quad \Longrightarrow \quad \frac{}{\Gamma, \Gamma', P \vdash P}\, Ax$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\, R{\to} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, A \vdash B}{\Gamma, \Gamma', A \vdash B}\, W}{\Gamma, \Gamma' \vdash A{\to}B}\, R{\to}$$

$$\frac{\Gamma, P, B \vdash G}{\Gamma, P, P{\to}B \vdash G}\, La{\to} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, P, B \vdash G}{\Gamma, \Gamma', P, B \vdash G}\, W}{\Gamma, \Gamma', P, P{\to}B \vdash G}\, La{\to}$$

$$\frac{\Gamma, A, B{\to}C \vdash B \quad \Gamma, C \vdash G}{\Gamma, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, A, B{\to}C \vdash B}{\Gamma, \Gamma', A, B{\to}C \vdash B}\, W \quad \dfrac{\Gamma, C \vdash G}{\Gamma, \Gamma', C \vdash G}\, W}{\Gamma, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

– Pour la règle $R\forall$, soit z une variable qui n'est libre ni dans Γ ni dans Γ'. Soit $\rho = \{z/x\}$. Comme x n'est pas libre dans Γ, on a $\Gamma\rho = \Gamma$. De plus, on a $\forall z.(A\rho) = (\forall x.A)$. On effectue donc la transformation suivante :

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\, R\forall \quad \Longrightarrow \quad \frac{\dfrac{\dfrac{\Gamma \vdash A}{\Gamma\rho \vdash A\rho}\, Subst_\rho}{\Gamma\rho, \Gamma' \vdash A\rho}\, W}{\Gamma\rho, \Gamma' \vdash \forall z.(A\rho) \;=\; \Gamma, \Gamma' \vdash \forall x.A}\, R\forall$$

L'application de la règle $R\forall$ reste valide car z ne peut pas être libre dans $\Gamma\rho$ ni dans Γ' par construction.

– Pour la règle $L\forall$:

$$\frac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \forall x.A \vdash G}\, L\forall \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \Gamma', \forall x.A, A\{t/x\} \vdash G}\, W}{\Gamma, \Gamma', \forall x.A \vdash G}\, L\forall$$

– $L\forall\rightarrow$:

$$\frac{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A \quad \Gamma, B \vdash G}{\Gamma, (\forall x.A)\rightarrow B \vdash G}\, L\forall\rightarrow$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A}{\Gamma, \Gamma', (\forall x.A)\rightarrow B \vdash \forall x.A}\, W \quad \dfrac{\Gamma, B \vdash G}{\Gamma, \Gamma', B \vdash G}\, W}{\Gamma, \Gamma', (\forall x.A)\rightarrow B \vdash G}\, L\forall\rightarrow$$

– RI_i :

$$\frac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j}{\Gamma \vdash I(\mathbf{p})}\, RI_i \quad \Longrightarrow \quad \frac{\dfrac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j}{\{\Gamma, \Gamma' \vdash (H_{i,j}(\mathbf{p}, \mathbf{t_i}))\}_j}\, W}{\Gamma, \Gamma' \vdash I(\mathbf{p})}\, RI_i$$

– LI : Pour tout i, on construit le renommage $\rho_i = \{\mathbf{z_i}/\mathbf{y_i}\}$, où les variables $\mathbf{z_i}$ sont deux à deux distinctes et n'apparaissent pas libres dans Γ, $\mathbf{p}$ ni dans G.

$$\frac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_i}{\Gamma, I(\mathbf{p}) \vdash G}\, LI \quad \Longrightarrow \quad \frac{\dfrac{\dfrac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_i}{\{\Gamma\rho_i, (\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}))\rho_i \vdash G\rho_i\}_i}\, Subst_{\rho_i}}{\{\Gamma\rho_i, \Gamma', (\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}))\rho_i \vdash G\rho_i\}_i}\, W}{\{\Gamma, (\mathbf{H_i}(\mathbf{p}, \mathbf{z_i})) \vdash G\}_i = \Gamma, \Gamma', I(\mathbf{p}) \vdash G}\, LI$$

– $LI{\rightarrow}$:

$$\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}{\Gamma, I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}{\Gamma, \Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\, W}{\Gamma, \Gamma', I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

Ce qui clôt la preuve par récurrence. □

LEMME 3.9 (RÈGLES INVERSIBLES)
Les règles suivantes sont fortement admissibles (cf. 1.14) dans LJTI :

$$\dfrac{\Gamma \vdash A{\rightarrow}B}{\Gamma, A \vdash B} \quad (3.1) \qquad \dfrac{\Gamma, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, B \vdash G} \quad (3.4)$$

$$\dfrac{\Gamma, P{\rightarrow}B \vdash G}{\Gamma, B \vdash G} \quad (3.2) \qquad \dfrac{\Gamma, I(\mathbf{p}) \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}) \vdash G} \quad (3.5)$$

$$\dfrac{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, B \vdash G} \quad (3.3) \qquad \dfrac{\Gamma, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G} \quad (3.6)$$

Démonstration : Par récurrence sur la hauteur de la dérivation, en utilisant la règle *Subst* pour renommer les variables et pour prouver la règle 3.5. Le détail de la preuve est donné en annexe, dans la section A.1. □

3.3.2 Élimination des contractions

Pour commencer, nous démontrons que la règle d'axiome non restreinte est admissible dans $LJTI$. Nous en déduisons ensuite que l'on peut éliminer les contractions d'une preuve $LJTI$. De cela nous concluons que la règle $L{\rightarrow}$ du calcul LJ est admissible dans $LJTI$. Afin d'effectuer nos démonstrations par récurrence sur les formules, nous définissons la notion de poids d'une formule :

Définition 3.10 (poids d'une formule)
Le poids d'une formule est défini par les équations suivantes :

$$\begin{aligned}
\text{poids}(P) &= 1, \text{ si } P \text{ est atomique} \\
\text{poids}(A{\rightarrow}B) &= 1 + \text{poids}(A) + \text{poids}(B) \\
\text{poids}(\forall x.A) &= 1 + \text{poids}(A) \\
\text{poids}(I(\mathbf{p})) &= \sum_i \text{poids}(\mathcal{C}_i(\mathbf{p})) \\
\text{poids}(\mathcal{C}_i(\mathbf{p})) &= (2 \times \text{taille}(\mathbf{y_i})) + \sum_j 1 + \text{poids}(H_{i,j}(\mathbf{p})) \\
&\text{si } \mathcal{C}_i(\mathbf{p}) : \forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}I(\mathbf{p})
\end{aligned}$$

Il est à noter que le résultat ne dépend pas du choix des variables $\mathbf{y_i}$*. L'expression taille*$(\mathbf{y_i})$ *désigne le nombre de variables dans la liste* $\mathbf{y_i}$*.*

Le poids ainsi défini est inférieur à celui utilisé par Roy Dyckhoff [DN00] dans le cas de la disjonction $\vee$ mais les démonstrations faites par Dyckhoff sont compatibles avec notre poids. En revanche, il reste à établir que ce système d'équations définit bien une fonction à valeurs entières, dont la propriété principale est que l'application des règles concernant les formules inductives conduit au remplacement de celles-ci par des formules de poids plus faible. Il s'agit là d'une difficulté nouvelle par rapport à la preuve de Dyckhoff puisque les hypothèses des constructeurs d'inductifs usuels ($\wedge$,$\vee$,$\exists$) sont des paramètres ou des instances de leurs paramètres, et dans ce cas la définition du poids est purement structurelle. En revanche dans le cas de nos inductifs, une sous-formule peut avoir une taille arbitraire et réutiliser ses paramètres un nombre arbitraire de fois.

Lemme 3.11 (Bonne définition de la fonction poids)
Soit Σ *une signature bien formée, alors la fonction* poids *est bien définie (c'est-à-dire que son calcul termine).*

Démonstration : Nous allons montrer qu'il existe une relation bien fondée $<_\Sigma$ sur les formules telle que :

$$A <_\Sigma A{\rightarrow}B \tag{3.7}$$
$$B <_\Sigma A{\rightarrow}B \tag{3.8}$$
$$A <_\Sigma \forall_s x.A \tag{3.9}$$
$$H_{i,j}(\mathbf{p}, \mathbf{y_i}) <_\Sigma I(\mathbf{p}) \tag{3.10}$$

À partir de la signature Σ, on définit une signature également bien formée $\widehat{\Sigma}$ telle que :

$$\widehat{\Sigma} = \{c_s\}_{s\in\mathcal{S}} \cup \{\widehat{P} : *\}_{P:(s_1,\ldots,s_n)\rightarrow *\in\Sigma} \cup \{\widehat{I} : Ind(p_1 : \widehat{\tau}_1, \ldots, p_n : \widehat{\tau}_n)\}_{I:Ind(p_1:\tau_1,\ldots,p_n:\tau_n))\in\Sigma}$$

Pour toute arité de paramètre, on définit $\widehat{(s_1, \ldots, s_n)} \to \tau = \tau$, où $\tau \in \mathcal{S} \cup \{*\}$. Pour chaque constructeur $\mathcal{C}_i : \forall y_{i,1} : s_{i,1}, \ldots, y_{i,k_i} : s_{i,k_i}.H_{i,1} \to \ldots \to H_{i,h_i} \to \square$ de la famille I, on a un constructeur $\widehat{\mathcal{C}_i} : \widehat{H}_{i,1} \to \ldots \to \widehat{H}_{i,h_i} \to \square$ dans la famille $\widehat{I}$.

On définit également la transformation $\widehat{F}$ de la formule F par :

$$\begin{aligned}
\widehat{P(t_1, \ldots, t_n)} &= \widehat{P} \\
\widehat{A \to B} &= \widehat{A} \to \widehat{B} \\
\widehat{\forall_s x.A} &= \forall_s x.\widehat{A} \\
\widehat{I(p_1, \ldots, p_n)} &= \widehat{I}(\widehat{p_{i_1}}, \ldots, \widehat{p_{i_n}}) \\
\widehat{\lambda x_1 \ldots x_n.u} &= \widehat{u} \\
\widehat{t} &= c_s \text{ si } t \text{ est un terme de sorte } s
\end{aligned}$$

On constate que $\widehat{\Sigma}$ est bien formée si on la construit avec l'ordre de définition $\prec_{\widehat{\Sigma}}$ tel que $\widehat{I_1} \prec_{\widehat{\Sigma}} \widehat{I_2}$ si, et seulement si, $I_1 \prec_\Sigma I_2$; pour cela il suffit de constater qu'il n'y a plus de variable libre après effacement. Par construction, $\prec_{\widehat{\Sigma}}$ est alors bien fondé puisque $\prec_\Sigma$ l'est.

On peut alors étendre $\prec_{\widehat{\Sigma}}$ en un nouvel ordre bien fondé $\prec'_{\widehat{\Sigma}}$ en ajoutant, pour toute famille inductive $\widehat{I} \in \widehat{\Sigma}$, les relations :

$$\begin{aligned}
&\forall_s \prec'_{\widehat{\Sigma}} \widehat{I} \\
&\to \prec'_{\widehat{\Sigma}} \widehat{I} \\
&c_s \prec'_{\widehat{\Sigma}} \widehat{I} \text{ pour tout } s \in \mathcal{S} \\
&\widehat{P} \prec'_{\widehat{\Sigma}} \widehat{I} \text{ pour tout } \widehat{P} : * \in \widehat{\Sigma}
\end{aligned}$$

Nous définissons alors $<_{\widehat{\Sigma}}$ comme l'extension par ordre multi-ensemble sur les chemins (MPO) de l'ordre de précédence $\prec'_{\widehat{\Sigma}}$. Cet ordre est par construction bien fondé. On définit alors la relation $<_\Sigma$ comme l'image inverse de $<_{\widehat{\Sigma}}$ par l'opérateur d'effacement $\widehat{\_}$, c'est-à dire que $F_1 <_\Sigma F_2$ si, et seulement si, $\widehat{F_1} <_{\widehat{\Sigma}} \widehat{F_2}$. La relation $<_\Sigma$ est alors bien fondée comme image inverse d'une relation bien fondée.

Il nous reste à montrer que la relation $<_\Sigma$ satisfait bien les conditions 3.7 à 3.10.

- (3.7), (3.8) et (3.9) : On a $\widehat{A} <_{\widehat{\Sigma}} \widehat{A \to B} = \widehat{A} \to \widehat{B}$ car $\widehat{A}$ est un sous-terme strict de $\widehat{A} \to \widehat{B}$. On en déduit par définition que $A <_\Sigma A \to B$. Les cas (3.8) et (3.9) sont traités de manière identique.
- (3.10) : On veut montrer que $H_{i,j}(\mathbf{p}, \mathbf{y_i}) <_\Sigma I(\mathbf{p})$, c'est-à-dire que $\widehat{H_{i,j}(\mathbf{p}, \mathbf{y_i})} <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$. Pour cela, nous allons montrer par récurrence sur la forme des formules que toute sous-formule (au sens large) F de $\widehat{H_{i,j}(\mathbf{p}, \mathbf{y_i})}$ vérifie $F <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$.

  Soit F une sous-formule de $\widehat{H_{i,j}(\mathbf{p}, \mathbf{y_i})}$:
  - Si F est une formule atomique, alors il existe $\widehat{P} \in \widehat{\Sigma}$ tel que $F = \widehat{P} \prec'_{\widehat{\Sigma}} \widehat{I}$ donc comme $\widehat{P}$ n'a pas de sous-termes, on conclut que $\widehat{P} <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$.

- Si $F = A \rightarrow B$, alors par hypothèse de récurrence, $A <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$ et $B <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$. De plus, $\rightarrow \prec'_{\widehat{\Sigma}} \widehat{I}$, donc $A \rightarrow B <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$.
- Si $F = \forall_s x.A$, alors par hypothèse de récurrence, $A <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$. De plus, $\forall_s \prec'_{\widehat{\Sigma}} \widehat{I}$, donc $\forall_s x.A <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$.
- Si $F = \widehat{J}(\widehat{\mathbf{q}})$, deux cas se présentent :
  1. Si $\widehat{J} \prec_{\widehat{\Sigma}} \widehat{I}$, alors examinons les paramètres $\widehat{\mathbf{q}}$. Chaque paramètre est soit une formule Q, auquel cas par hypothèse de récurrence on peut affirmer que $Q <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$, soit une constante $c_s \prec'_{\widehat{\Sigma}} \widehat{I}$ et alors $c_s <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$. On en conclut que $\widehat{J}(\widehat{\mathbf{q}}) <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$.
  2. Sinon, $\widehat{J} \not\prec_{\widehat{\Sigma}} \widehat{I}$ et donc $J \not\prec_{\Sigma} I$, donc par définition, J n'apparaît pas dans le contexte $H_{i,j}(\_, \mathbf{y_i})$, d'où $\widehat{J}$ n'apparaît pas dans $\widehat{H_{i,j}(\_, \mathbf{y_i})}$. On en déduit que F est une sous-formule de l'un des paramètres $\widehat{p_i}$ de $\widehat{I}$, donc $F <_{\widehat{\Sigma}} \widehat{I}(\widehat{\mathbf{p}})$ par sous-terme.

Ceci clôt la preuve par récurrence.
On a bien montré que $<_{\Sigma}$ est une relation bien fondée satisfaisant les conditions 3.7 à 3.10, ce qui permet d'affirmer que la fonction poids est bien définie. □

LEMME 3.12
Les séquents ayant la forme suivante sont dérivables dans LJTI :

1. $\Gamma, A \vdash A$ *(axiome généralisé)*
2. $\Gamma, A, A \rightarrow B \vdash B$ *(*modus ponens*)*

Démonstration :

1. La preuve est faite par récurrence sur le poids de la formule A.
   - Si A est atomique, alors, pour tout Γ, on a :

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

   - Si $A = P \rightarrow B$, où P est atomique, alors on construit :

$$\dfrac{\dfrac{\dfrac{Ind(B)}{\Gamma, P, B \vdash B}}{\Gamma, P, P \rightarrow B \vdash B}\ La\rightarrow}{\Gamma, P \rightarrow B \vdash P \rightarrow B}\ R\rightarrow$$

     $Ind(B)$ désigne l'utilisation de l'hypothèse de récurrence avec la formule B, de poids strictement inférieur à $P \rightarrow B$.

– Si $A = (C{\to}D){\to}B$, on construit :

$$\dfrac{\dfrac{\dfrac{\dfrac{Ind(C{\to}D)}{\Gamma, D{\to}B, C{\to}D \vdash C{\to}D}}{\Gamma, D{\to}B, C{\to}D, C \vdash D}\ \text{lemme 3.9 règle 3.1} \qquad \dfrac{Ind(B)}{\Gamma, B, C{\to}D \vdash B}}{\Gamma, (C{\to}D){\to}B, C{\to}D \vdash B}\ L{\to}{\to}}{\Gamma, (C{\to}D){\to}B \vdash (C{\to}D){\to}B}\ R{\to}$$

– Si $A = \forall x.B$, on construit :

$$\dfrac{\dfrac{\dfrac{Ind(B)}{\Gamma, B \vdash B}}{\Gamma, \forall x.B \vdash B}\ L\forall}{\Gamma, \forall x.B \vdash \forall x.B}\ R\forall$$

– Si $A = (\forall x.C){\to}B$:

$$\dfrac{\dfrac{\dfrac{Ind(\forall x.C)}{\Gamma, (\forall x.C){\to}B, \forall x.C \vdash \forall x.C} \qquad \dfrac{Ind(B)}{\Gamma, B, \forall x.C \vdash B}}{\Gamma, (\forall x.C){\to}B, \forall x.C \vdash B}\ L\forall{\to}}{\Gamma, (\forall x.C){\to}B \vdash (\forall x.C){\to}B}\ R{\to}$$

– Si $A = I(\mathbf{p})$, alors pour tout constructeur $\mathcal{C}_i$, on peut trouver des variables distinctes $\mathbf{y_i}$ libres dans Γ et $\mathbf{p}$. Pour toute hypothèse logique, on a alors :

$$\dfrac{\ldots \qquad \dfrac{\ldots \quad \dfrac{Ind(H_{i,j}(\mathbf{p}, \mathbf{y_i}))}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash H_{i,j}(\mathbf{p}, \mathbf{y_i})} \quad \ldots}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash I(\mathbf{p})}\ RI_i \qquad \ldots}{\Gamma, I(\mathbf{p}) \vdash I(\mathbf{p})}\ LI$$

– Si $A = I(\mathbf{p}){\to}B$, on souhaite construire une dérivation se terminant comme l'indique le schéma suivant :

$$\dfrac{\dfrac{\dfrac{\ldots \quad \Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B\}_i, \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}) \vdash B \quad \ldots}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B\}_i, I(\mathbf{p}) \vdash B}\ LI}{\Gamma, I(\mathbf{p}){\to}B, I(\mathbf{p}) \vdash B}\ LI{\to}}{\Gamma, I(\mathbf{p}){\to}B \vdash I(\mathbf{p}){\to}B}\ R{\to}$$

$\vdots\ \mathfrak{D}_k$

Où les $\mathbf{z_k}$ sont des variables fraîches pour chaque k, rendant valide l'application de la règle LI. Maintenant, il s'agit de construire la dérivation $\mathfrak{D}_k$ correspondant

à chaque constructeur $\mathcal{C}_k$. Pour k donné, soit v_k le nombre de variables et h_k le nombre d'hypothèses du constructeur $\mathcal{C}_k$. On pose :

$$\begin{aligned} \Gamma'_k &= \Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B\}_{i \neq k} \\ \phi_k(z_{k,1}, \ldots, z_{k,l}) &= \forall y_{k,l+1} \ldots y_{k,v_k}. \\ &\qquad \mathbf{H_k}(\mathbf{p}, z_{k,1} \ldots z_{k,l}, y_{k,l+1} \ldots y_{k,v_k}) \rightarrow B \end{aligned}$$

Le séquent que l'on souhaite obtenir s'écrit maintenant :

$$\Gamma'_k, \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}), \phi_k() \vdash B$$

On construit alors la dérivation $\mathfrak{D}_k$ en utilisant v_k fois la règle $L\forall$ avec les dérivés de la formule $\phi_k()$ et les $\mathbf{z_k}$, puis h_k fois la règle 3.1 du lemme 3.9, et enfin en remarquant que $\phi_k(\mathbf{z_k}) = \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}) \rightarrow B$, formule de poids inférieur à $I(\mathbf{p}) \rightarrow B$.

$$\mathfrak{D}_k = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Ind(\mathbf{H_k}(\mathbf{p}, \mathbf{z_k}) \rightarrow B)}{\Gamma'_k, \phi_k(), \phi_k(z_{k,1}), ..., \phi_k(\mathbf{z_k}) \vdash \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}) \rightarrow B}}{\vdots \text{ règle 3.1 } (h_k \text{ fois})}}{\Gamma'_k, \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}), \phi_k(), \phi_k(z_{k,1}), ..., \phi_k(\mathbf{z_k}) \vdash B}}{\vdots\ L\forall}}{\Gamma'_k, \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}), \phi_k(), \phi_k(z_{k,1}) \vdash B}\ L\forall}{\Gamma'_k, \mathbf{H_k}(\mathbf{p}, \mathbf{z_k}), \phi_k() \vdash B}$$

Ce qui clôt la preuve par récurrence.

2. La preuve est immédiate en utilisant la dérivation suivante :

$$\dfrac{\dfrac{\vdots \text{ 1. (ci-dessus)}}{\Gamma, A \rightarrow B \vdash A \rightarrow B}}{\vdots \text{ règle 3.1}}{\Gamma, A, A \rightarrow B \vdash B}$$

□

Le lemme suivant prouve l'admissibilité de la règle d'élimination de l'implication utilisée dans le calcul LJ.

LEMME 3.13
La règle suivante est admissible dans $LJTI$:

$$\dfrac{\Gamma \vdash D \quad \Gamma, B \vdash E}{\Gamma, D \rightarrow B \vdash E}$$

Démonstration : Nous procéderons par récurrence sur la hauteur de la dérivation de la première prémisse $\Gamma \vdash D$, et par cas sur la règle ayant dérivé cette prémisse.

– Si la dernière règle est Ax, alors D est une formule atomique et est présente dans Γ. Soit Γ' tel que $\Gamma = \Gamma', D$; on procède à la transformation suivante :

$$\dfrac{\dfrac{}{\Gamma', D \vdash D}\,Ax \qquad \Gamma', D, B \vdash E}{\Gamma', D, D{\to}B \vdash E} \quad \Longrightarrow \quad \dfrac{\Gamma', D, B \vdash E}{\Gamma', D, D{\to}B \vdash E}\,La{\to}$$

– Si la dernière règle est $R{\to}$, alors D est de la forme $D_1{\to}D_2$. On effectue la transformation suivante :

$$\dfrac{\dfrac{\Gamma, D_1 \vdash D_2}{\Gamma \vdash D_1{\to}D_2}\,R{\to} \qquad \Gamma, B \vdash E}{\Gamma, (D_1{\to}D_2){\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, D_1 \vdash D_2}{\Gamma, D_2{\to}B, D_1 \vdash D_2}\,W \qquad \Gamma, B \vdash E}{\Gamma, (D_1{\to}D_2){\to}B \vdash E}\,L{\to}{\to}$$

– Si la dernière règle est $La{\to}$, alors Γ est de la forme $\Gamma', P, P{\to}C$ avec P atomique.

$$\dfrac{\dfrac{\Gamma', P, C \vdash D}{\Gamma', P, P{\to}C \vdash D}\,La{\to} \qquad \Gamma', P, P{\to}C, B \vdash E}{\Gamma', P, P{\to}C, D{\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma', P, C \vdash D \qquad \Gamma', P, C, B \vdash E}{\Gamma', P, C, D{\to}B \vdash E}\,Ind}{\Gamma', P, P{\to}C, D{\to}B \vdash E}\,La{\to} \qquad \begin{array}{c}\Gamma', P, P{\to}C, B \vdash E\\ \vdots\ \text{règle 3.2}\end{array}$$

– Si c'est $L{\to}{\to}$ alors $\Gamma{=}\Gamma', (F{\to}G){\to}H$.

$$\dfrac{\dfrac{\Gamma', G{\to}H, F \vdash G \quad \Gamma', H \vdash D}{\Gamma', (F{\to}G){\to}H \vdash D}\,L{\to}{\to} \qquad \Gamma', (F{\to}G){\to}H, B \vdash E}{\Gamma', (F{\to}G){\to}H, D{\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma', G{\to}H, F \vdash G}{\Gamma', G{\to}H, F, D{\to}B \vdash G}\,W \qquad \dfrac{\Gamma', H \vdash D \qquad \Gamma', H, B \vdash E}{\Gamma', H, D{\to}B \vdash E}\,Ind}{\Gamma', (F{\to}G){\to}H, D{\to}B \vdash E}\,L{\to}{\to} \qquad \begin{array}{c}\Gamma', (F{\to}G){\to}H, B \vdash E\\ \vdots\ \text{règle 3.3}\end{array}$$

– $R\forall$: On a $D = \forall x.C$.

$$\dfrac{\dfrac{\Gamma \vdash C}{\Gamma \vdash \forall x.C}\, R\forall \quad \Gamma, B \vdash E}{\Gamma, (\forall x.C)\rightarrow B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash C}{\Gamma \vdash \forall x.C}\, R\forall}{\Gamma, (\forall x.C)\rightarrow B \vdash \forall x.C}\, W \quad \Gamma, B \vdash E}{\Gamma, (\forall x.C)\rightarrow B \vdash E}\, L\forall\rightarrow$$

– $L\forall$: Prenons Γ' tel que $\Gamma = \Gamma', \forall x.C$.

$$\dfrac{\dfrac{\Gamma', \forall x.C, C\{t/x\} \vdash D}{\Gamma', \forall x.C \vdash D}\, L\forall \quad \Gamma', \forall x.C, B \vdash E}{\Gamma', \forall x.C, D\rightarrow B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\Gamma', \forall x.C, C\{t/x\} \vdash D \quad \dfrac{\Gamma', \forall x.C, B \vdash E}{\Gamma', \forall x.C, C\{t/x\}, B \vdash E}\, W}{\dfrac{\Gamma', \forall x.C, C\{t/x\}, D\rightarrow B \vdash E}{\Gamma', \forall x.C, D\rightarrow B \vdash E}\, L\forall}\, Ind$$

– $L\forall\rightarrow$: Γ est de la forme $\Gamma', (\forall x.G)\rightarrow C$.

$$\dfrac{\dfrac{\Gamma', (\forall x.G)\rightarrow C \vdash \forall x.G \quad \Gamma', C \vdash D}{\Gamma', (\forall x.G)\rightarrow C \vdash D}\, L\forall\rightarrow \quad \Gamma', (\forall x.G)\rightarrow C, B \vdash E}{\Gamma, (\forall x.G)\rightarrow C, D\rightarrow B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma', (\forall x.G)\rightarrow C \vdash \forall x.G}{\Gamma', (\forall x.G)\rightarrow C, D\rightarrow B \vdash \forall x.G}\, W \quad \dfrac{\Gamma', C \vdash D \qquad \begin{array}{c}\Gamma', (\forall x.G)\rightarrow C, B \vdash E \\ \vdots \text{ règle 3.4} \\ \Gamma', C, B \vdash E\end{array}}{\Gamma', C, D\rightarrow B \vdash E}\, Ind}{\Gamma', (\forall x.G)\rightarrow C, D\rightarrow B \vdash E}\, L\forall\rightarrow$$

– Si la dernière règle est RI_i, alors $D = I(\mathbf{p})$. On a besoin de l'hypothèse de récurrence appliquée à la dérivation de chacun des séquents de la forme $\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})$ pour

$j = 1 \dots h_k$.

$$\dfrac{\dfrac{\dfrac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j}{\Gamma \vdash I(\mathbf{p})}\, RI_i \qquad \Gamma, B \vdash E}{\Gamma, I(\mathbf{p}) \rightarrow B \vdash E}}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_j \quad \Gamma, B \vdash E}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t_i}) \rightarrow B \vdash E}\, Ind \text{ (h_k fois})}{\Gamma, \forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B, \dots, \mathbf{H_i}(\mathbf{p}, \mathbf{t_i}) \rightarrow B \vdash E}\, W}{\vdots\ L\forall \text{ (v_k fois)}}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B \vdash E}{\Gamma, \{\forall \mathbf{y_k}.\mathbf{H_k}(\mathbf{p}, \mathbf{y_k}) \rightarrow B\}_k \vdash E}\, W}{\Gamma, I(\mathbf{p}) \rightarrow B \vdash E}\, LI\rightarrow}$$

– LI : Γ est de la forme $\Gamma', I(\mathbf{p})$. Pour chaque constructeur $\mathcal{C}_k$, on choisit des variables distinctes $\mathbf{z_k}$ n'apparaissant pas libres dans D,B,E,Γ' ni dans $\mathbf{p}$.

$$\dfrac{\dfrac{\dfrac{\{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash D\}_i}{\Gamma', I(\mathbf{p}) \vdash D}\, LI \qquad \Gamma', I(\mathbf{p}), B \vdash E}{\Gamma', I(\mathbf{p}), D \rightarrow B \vdash E}}{\Downarrow}$$

$$\dfrac{\dots \qquad \dfrac{\dfrac{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash D}{\vdots\ Subst_{\{\mathbf{z_i}/\mathbf{y_i}\}}} \qquad \dfrac{\Gamma', I(\mathbf{p}), B \vdash E}{\vdots \text{ règle 3.5}}}{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}) \vdash D \qquad \Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}), B \vdash E}\, Ind}{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}), D \rightarrow B \vdash E} \qquad \dots}{\Gamma', I(\mathbf{p}), D \rightarrow B \vdash E}\, LI$$

Le renommage des $\mathbf{y_k}$ par les $\mathbf{z_k}$ rend l'application de la règle LI valide. L'utilisation de l'hypothèse de récurrence est correcte car la règle $Subst$ est fortement admissible et n'augmente donc pas la hauteur de l'arbre de dérivation de $\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}) \vdash D$.

– $LI{\rightarrow}$: Γ est de la forme $\Gamma', I(\mathbf{p}){\rightarrow}C$.

$$\dfrac{\dfrac{\Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i \vdash D}{\Gamma', I(\mathbf{p}){\rightarrow}C \vdash D}\, LI{\rightarrow} \qquad \Gamma', I(\mathbf{p}){\rightarrow}C, B \vdash E}{\Gamma', I(\mathbf{p}){\rightarrow}C, D{\rightarrow}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i \vdash D \qquad \begin{array}{c}\Gamma', I(\mathbf{p}){\rightarrow}C, B \vdash E \\ \vdots \text{ règle 3.6} \\ \Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i, B \vdash E\end{array}}{\dfrac{\Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i, D{\rightarrow}B \vdash E}{\Gamma', I(\mathbf{p}){\rightarrow}C, D{\rightarrow}B \vdash E}\, LI{\rightarrow}}\, Ind$$

Ce qui clôt la preuve par récurrence. □

LEMME 3.14
La règle suivante est admissible dans $LJTI$.

$$\dfrac{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash E}{\Gamma, C, D{\rightarrow}B, D{\rightarrow}B \vdash E}$$

Démonstration : Par récurrence sur la hauteur de la dérivation et par cas sur la dernière règle. Toutes les règles commutent avec cette nouvelle règle, sauf la règle $L{\rightarrow}{\rightarrow}$ lorsque $(C{\rightarrow}D){\rightarrow}B$ est principale. Dans ce cas, on effectue la transformation suivante :

$$\dfrac{\Gamma, C, D{\rightarrow}B \vdash D \qquad \dfrac{\Gamma, B \vdash E}{\Gamma, C, D{\rightarrow}B, B \vdash E}\, W}{\Gamma, C, D{\rightarrow}B, D{\rightarrow}B \vdash E}\, \text{lemme 3.13}$$

□

Théorème d'élimination des contractions

Grâce à l'admissibilité des règles des lemmes précédents, nous sommes en mesure de prouver l'admissibilité de la contraction, étape importante vers l'élimination des coupures.

THÉORÈME 3.15 (ÉLIMINATION DES CONTRACTIONS)
La règle suivante est admissible dans $LJTI$:

$$\dfrac{\Gamma, A, A \vdash G}{\Gamma, A \vdash G}\, Contr$$

Démonstration : Nous effectuons la preuve par récurrence sur le poids de A d'une part et sur la hauteur de la dérivation de la prémisse d'autre part. Ensuite deux cas se présentent :

- si A n'est pas la formule principale de la dernière règle, alors on peut faire commuter cette règle avec la contraction, que l'on peut ensuite éliminer par récurrence de la dérivation des prémisses puisque le poids de la formule A concernée reste le même alors que la hauteur de la dérivation décroît.
- Si A est la formule principale, alors on procède différemment suivant la forme de A :
  - Si A est une formule atomique P, alors la dernière règle est un axiome et $G = P$. On effectue la transformation suivante :

$$\dfrac{\dfrac{}{\Gamma, P, P \vdash P}\, Ax}{\Gamma, P \vdash P}\, Contr \quad \Longrightarrow \quad \dfrac{}{\Gamma, P \vdash P}\, Ax$$

  - Si A est de la forme $P \to B$ avec P atomique :

$$\dfrac{\dfrac{\dfrac{\Gamma, P, B, P \to B \vdash G}{\Gamma, P, P \to B, P \to B \vdash G}\, La\to}{\Gamma, P, P \to B \vdash G}\, Contr \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P, B, P \to B \vdash G}{\Gamma, P, B, B \vdash G}\ \text{règle 3.2}}{\Gamma, P, B \vdash G}\, Ind(B)}{\Gamma, P, P \to B \vdash G}\, La\to$$

  - $A = (C \to D) \to B$:

$$\dfrac{\Gamma, D \to B, (C \to D) \to B, C \vdash D \quad \Gamma, B, (C \to D) \to B \vdash G}{\dfrac{\Gamma, (C \to D) \to B, (C \to D) \to B \vdash G}{\Gamma, (C \to D) \to B \vdash G}\, Contr}\, L\to\to$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, D \to B, (C \to D) \to B, C \vdash D}{\Gamma, D \to B, C, D \to B, D \to B, C \vdash D}\ \text{lemme 3.14}}{\Gamma, D \to B, C, D \to B \vdash D}\, Ind(D \to B) \quad \dfrac{\Gamma, B, (C \to D) \to B \vdash G}{\Gamma, B, B \vdash G}\ \text{règle 3.3}}{\Gamma, D \to B, C \vdash D}\, Ind(D \to B) \quad \dfrac{\Gamma, B, B \vdash G}{\Gamma, B \vdash G}\, Ind(B)}{\Gamma, (C \to D) \to B \vdash G}\, L\to\to$$

– $A = \forall x.B$:

$$\dfrac{\dfrac{\Gamma, \forall x.B, \forall x.B, B\{t/x\} \vdash G}{\Gamma, \forall x.B, \forall x.B \vdash G}\, L\forall}{\Gamma, \forall x.B \vdash G}\, Contr$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \forall x.B, \forall x.B, B\{t/x\} \vdash G}{\Gamma, \forall x.B, B\{t/x\} \vdash G}\, Ind'}{\Gamma, \forall x.B \vdash G}\, L\forall$$

– $A = (\forall x.C) \rightarrow B$:

$$\dfrac{\dfrac{\Gamma, (\forall x.C)\rightarrow B, (\forall x.C)\rightarrow B \vdash \forall x.C \quad \Gamma, B, (\forall x.C)\rightarrow B \vdash G}{\Gamma, (\forall x.C)\rightarrow B, (\forall x.C)\rightarrow B \vdash G}\, L\forall\rightarrow}{\Gamma, (\forall x.C)\rightarrow B \vdash G}\, Contr$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, (\forall x.C)\rightarrow B, (\forall x.C)\rightarrow B \vdash \forall x.C}{\Gamma, (\forall x.C)\rightarrow B \vdash \forall x.C}\, Ind' \quad \dfrac{\Gamma, B, (\forall x.C)\rightarrow B \vdash G \;\vdots\text{ règle 3.4}\; \Gamma, B, B \vdash G}{\Gamma, B \vdash G}\, Ind(B)}{\Gamma, (\forall x.C)\rightarrow B \vdash G}\, L\forall\rightarrow$$

– $A = I(\mathbf{p})$:

$$\dfrac{\dfrac{\ldots \quad \Gamma, I(\mathbf{p}), \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G \quad \ldots}{\Gamma, I(\mathbf{p}), I(\mathbf{p}) \vdash G}\, LI}{\Gamma, I(\mathbf{p}) \vdash G}\, Contr$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \begin{array}{c}\Gamma, I(\mathbf{p}), \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G \\ \vdots\text{ règle 3.5} \\ \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G \\ \vdots\; Ind(\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})) \\ \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\end{array} \quad \ldots}{\Gamma, I(\mathbf{p}) \vdash G}\, LI$$

– $A = I(\mathbf{p}) \rightarrow C$:

$$\dfrac{\dfrac{\Gamma, I(\mathbf{p})\rightarrow C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash G}{\Gamma, I(\mathbf{p})\rightarrow C, I(\mathbf{p})\rightarrow C \vdash G}\, LI\rightarrow}{\Gamma, I(\mathbf{p})\rightarrow C \vdash G}\, Contr$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\Gamma, I(\mathbf{p})\rightarrow C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash G \\ \vdots \text{ règle 3.6} \\ \Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash G \\ \vdots\ Ind(\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C) \\ \Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash G\end{array}}{\Gamma, I(\mathbf{p})\rightarrow C \vdash G}\, LI\rightarrow$$

Ce qui clôt notre preuve par récurrence. □

LEMME 3.16
La règle suivante est admissible dans LJTI :

$$\dfrac{\Gamma, A\rightarrow B \vdash A \quad \Gamma, B \vdash G}{\Gamma, A\rightarrow B \vdash G}$$

Démonstration : En utilisant les règles admissibles précédentes, on obtient :

$$\dfrac{\dfrac{\Gamma, A\rightarrow B \vdash A \quad \dfrac{\Gamma, B \vdash G}{\Gamma, A\rightarrow B, B \vdash G}\, W}{\Gamma, A\rightarrow B, A\rightarrow B \vdash A}\, \text{lemme 3.13}}{\Gamma, A\rightarrow B \vdash G}\, Contr$$

□

Ce résultat nous montre que le système $LJTI$ est complet par rapport au système LJI de la figure 3.8.

3.3.3 Élimination des coupures pour $LJTI$

THÉORÈME 3.17 (ADMISSIBILITÉ DE LA COUPURE)
La règle de coupure ci-dessous est admissible dans LJTI.

$$\dfrac{\Gamma \vdash A \quad \Gamma', A \vdash E}{\Gamma, \Gamma' \vdash E}\, Cut$$

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

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\rightarrow}B}\ R{\rightarrow} \qquad \frac{\Gamma, A{\rightarrow}B \vdash A \quad \Gamma, B \vdash G}{\Gamma, A{\rightarrow}B \vdash G}\ L{\rightarrow}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ R\forall \qquad \frac{\Gamma, \forall x.A, A\{t/x\} \vdash G}{\Gamma, \forall x.A \vdash G}\ L\forall$$

$$\frac{\{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_{j=1..h_i}}{\Gamma \vdash I(\mathbf{p})}\ RI_i \qquad \frac{\{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\}_{i=1..n}}{\Gamma, I(\mathbf{p}) \vdash G}\ LI$$

FIG. 3.8 – Le calcul LJI

Démonstration : La preuve est réalisée par récurrence d'une part sur le poids de A et d'autre part sur la somme des hauteurs des dérivations des deux prémisses de la règle Cut.

Si la première prémisse est déduite par la règle d'axiome, alors Γ est de la forme Γ'', A. On fait la transformation suivante :

$$\frac{\dfrac{}{\Gamma'', A \vdash A}\ Ax \quad \Gamma', A \vdash E}{\Gamma'', \Gamma', A \vdash E}\ Cut \quad \Longrightarrow \quad \frac{\Gamma', A \vdash E}{\Gamma'', \Gamma', A \vdash E}\ W$$

Si la seconde prémisse est déduite par la règle d'axiome, alors $E \in \Gamma', A$ et deux cas sont possibles :

Si $A = E$:

$$\frac{\Gamma \vdash E \quad \dfrac{}{\Gamma', E \vdash E}\ Ax}{\Gamma, \Gamma' \vdash E}\ Cut \quad \Longrightarrow \quad \frac{\Gamma \vdash E}{\Gamma, \Gamma' \vdash E}\ W$$

Sinon, Γ' est de la forme Γ'', E :

$$\frac{\Gamma \vdash A \quad \dfrac{}{\Gamma'', E, A \vdash E}\ Ax}{\Gamma, \Gamma'', E \vdash E}\ Cut \quad \Longrightarrow \quad \frac{}{\Gamma, \Gamma'', E \vdash E}\ Ax$$

Sinon, aucune prémisse n'est déduite par un axiome.

Si A n'est pas la formule principale de la dernière étape de la première prémisse $\Gamma \vdash A$, alors nous raisonnons par cas sur cette dernière étape.

– $La{\rightarrow}$

$$\frac{\dfrac{\dfrac{\Gamma'', P, B \vdash A}{\Gamma'', P, P{\rightarrow}B \vdash A}\ La{\rightarrow} \quad \Gamma', A \vdash E}{\Gamma'', P, P{\rightarrow}B, \Gamma' \vdash E}\ Cut}{\Downarrow}$$

$$\frac{\dfrac{\Gamma'', P, B \vdash A \quad \Gamma', A \vdash E}{\Gamma'', P, B, \Gamma' \vdash E}\ Ind'}{\Gamma'', P, P{\rightarrow}B, \Gamma' \vdash E}\ La{\rightarrow}$$

– $L{\to}{\to}$

$$\dfrac{\dfrac{\Gamma'', (C{\to}D){\to}B, C \vdash D \quad \Gamma'', B \vdash A}{\Gamma'', (C{\to}D){\to}B \vdash A}\, L{\to}{\to} \qquad \Gamma', A \vdash E}{\Gamma'', (C{\to}D){\to}B, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma'', (C{\to}D){\to}B, C \vdash D}{\Gamma'', (C{\to}D){\to}B, C, \Gamma' \vdash D}\, W \qquad \dfrac{\Gamma'', B \vdash A \quad \Gamma', A \vdash E}{\Gamma'', B, \Gamma' \vdash E}\, Ind'}{\Gamma'', (C{\to}D){\to}B, \Gamma' \vdash E}\, L{\to}{\to}$$

– $L\forall$

$$\dfrac{\dfrac{\Gamma'', \forall x.D, D\{t/x\} \vdash A}{\Gamma'', \forall x.D \vdash A}\, L\forall \qquad \Gamma', A \vdash E}{\Gamma'', \forall x.D, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma'', \forall x.D, D\{t/x\} \vdash A \quad \Gamma', A \vdash E}{\Gamma'', \forall x.D, D\{t/x\}, \Gamma' \vdash E}\, Ind' }{\Gamma'', \forall x.D, \Gamma' \vdash E}\, L\forall$$

– $L\forall{\to}$

$$\dfrac{\dfrac{\Gamma'', (\forall x.D){\to}C \vdash \forall x.D \quad \Gamma'', C \vdash A}{\Gamma'', (\forall x.D){\to}C \vdash A}\, L\forall{\to} \qquad \Gamma', A \vdash E}{\Gamma'', (\forall x.D){\to}C, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma'', (\forall x.D){\to}C \vdash \forall x.D}{\Gamma'', (\forall x.D){\to}C, \Gamma' \vdash \forall x.D}\, W \qquad \dfrac{\Gamma'', C \vdash A \quad \Gamma', A \vdash E}{\Gamma'', C, \Gamma' \vdash E}\, Ind'}{\Gamma'', (\forall x.D){\to}C, \Gamma' \vdash E}\, L\forall{\to}$$

– LI : Pour chaque constructeur $\mathcal{C}_i$, on choisit des variables distinctes z_i non libres dans Γ'',$\mathbf{p}$,Γ',E.

$$\dfrac{\dfrac{\ldots \quad \Gamma'', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash A \quad \ldots}{\Gamma'', I(\mathbf{p}) \vdash A}\, LI \qquad \Gamma', A \vdash E}{\Gamma'', I(\mathbf{p}), \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\ldots \qquad \dfrac{\dfrac{\Gamma'', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash A}{\Gamma'', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}) \vdash A}\, Subst_{\{\mathbf{z_i}/\mathbf{y_i}\}} \qquad \Gamma', A \vdash E}{\Gamma'', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}), \Gamma' \vdash E}\, Ind' \qquad \ldots}{\Gamma'', I(\mathbf{p}), \Gamma' \vdash E}\, LI$$

– $LI{\rightarrow}$

$$\dfrac{\dfrac{\Gamma'', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i \vdash A}{\Gamma'', I(\mathbf{p}){\rightarrow}C \vdash A}\, LI{\rightarrow} \quad \Gamma', A \vdash E}{\Gamma'', I(\mathbf{p}){\rightarrow}C, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\Gamma'', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i \vdash A \quad \Gamma', A \vdash E}{\dfrac{\Gamma'', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}C\}_i, \Gamma' \vdash E}{\Gamma'', I(\mathbf{p}){\rightarrow}C, \Gamma' \vdash E}\, LI{\rightarrow}}\, Ind'$$

Si A est la formule principale de la prémisse gauche, mais pas celle de la prémisse droite, on raisonne alors par cas sur la dernière règle concluant sur cette prémisse :

– $R{\rightarrow}$:

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma', A, B \vdash C}{\Gamma', A \vdash B{\rightarrow}C}\, R{\rightarrow}}{\Gamma, \Gamma' \vdash B{\rightarrow}C}\, Cut \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma \vdash A \quad \Gamma', A, B \vdash C}{\Gamma, \Gamma', B \vdash C}\, Ind'}{\Gamma, \Gamma' \vdash B{\rightarrow}C}\, R{\rightarrow}$$

– $La{\rightarrow}$: Comme la prémisse gauche n'est pas un axiome, A, sa formule principale, ne peut pas être atomique, ce qui fait que $A \neq P$. Γ' est donc de la forme $\Gamma'', P, P{\rightarrow}C$.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, P, C \vdash E}{\Gamma'', A, P, P{\rightarrow}C \vdash E}\, La{\rightarrow}}{\Gamma, \Gamma'', P, P{\rightarrow}C \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma'', A, P, C \vdash E}{\Gamma, \Gamma'', P, C \vdash E}\, Ind'}{\Gamma, \Gamma'', P, P{\rightarrow}C \vdash E}\, La{\rightarrow}$$

– $L{\rightarrow}{\rightarrow}$: Γ' est de la forme $\Gamma'', (C{\rightarrow}D){\rightarrow}B$.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, (C{\rightarrow}D){\rightarrow}B, C \vdash D \quad \Gamma'', A, B \vdash E}{\Gamma'', A, (C{\rightarrow}D){\rightarrow}B \vdash E}\, L{\rightarrow}{\rightarrow}}{\Gamma, \Gamma'', (C{\rightarrow}D){\rightarrow}B \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma'', A, (C{\rightarrow}D){\rightarrow}B, C \vdash D}{\Gamma, \Gamma'', (C{\rightarrow}D){\rightarrow}B, C \vdash D}\, Ind' \quad \dfrac{\Gamma \vdash A \quad \Gamma'', A, B \vdash E}{\Gamma, \Gamma'', B \vdash E}\, Ind'}{\Gamma, \Gamma'', (C{\rightarrow}D){\rightarrow}B \vdash E}\, L{\rightarrow}{\rightarrow}$$

– $R\forall$: On choisit z non libre dans Γ, Γ' et B.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma', A \vdash B}{\Gamma', A \vdash \forall x.B}\, R\forall}{\Gamma, \Gamma' \vdash \forall x.B}\, Cut \quad \Longrightarrow \quad \dfrac{\Gamma \vdash A \quad \dfrac{\Gamma', A \vdash B}{\Gamma', A \vdash B\{z/x\}}\, Subst_{\{z/x\}}}{\dfrac{\Gamma, \Gamma' \vdash B\{z/x\}}{\Gamma, \Gamma' \vdash \forall x.B}\, R\forall}\, Ind'$$

– $L\forall$:

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, \forall x.B, B\{t/x\} \vdash E}{\Gamma'', A, \forall x.B \vdash E}\, L\forall}{\Gamma, \Gamma'', \forall x.B \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\Gamma \vdash A \quad \Gamma'', A, \forall x.B, B\{t/x\} \vdash E}{\dfrac{\Gamma, \Gamma'', \forall x.B, B\{t/x\} \vdash E}{\Gamma, \Gamma'', \forall x.B \vdash E}\, L\forall}\, Ind'$$

– $L\forall\rightarrow$:

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, (\forall x.D)\rightarrow C \vdash \forall x.D \quad \Gamma'', A, C \vdash E}{\Gamma'', A, (\forall x.D)\rightarrow C \vdash E}\, L\forall\rightarrow}{\Gamma, \Gamma'', (\forall x.D)\rightarrow C \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma'', A, (\forall x.D)\rightarrow C \vdash \forall x.D}{\Gamma, \Gamma'', (\forall x.D)\rightarrow C \vdash \forall x.D}\, Ind' \quad \dfrac{\Gamma \vdash A \quad \Gamma'', A, C \vdash E}{\Gamma, \Gamma'', C \vdash E}\, Ind'}{\Gamma, \Gamma'', (\forall x.D)\rightarrow C \vdash E}\, L\forall\rightarrow$$

– RI_i :

$$\dfrac{\Gamma \vdash A \quad \dfrac{\ldots \quad \Gamma', A \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma', A \vdash I(\mathbf{p})}\, RI_i}{\Gamma, \Gamma' \vdash I(\mathbf{p})}\, Cut$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma \vdash A \quad \Gamma', A \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, \Gamma' \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind' \quad \ldots}{\Gamma', A \vdash I(\mathbf{p})}\, RI_i$$

– LI : On choisit pour chaque constructeur $\mathcal{C}_i$ des variables distinctes $\mathbf{z_i}$ non libres dans $\mathbf{p}$, Γ, Γ'' et E.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\ldots \quad \Gamma'', A, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash E \quad \ldots}{\Gamma'', A, I(\mathbf{p}) \vdash E}\, LI}{\Gamma, \Gamma'', I(\mathbf{p}) \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash E}{\Gamma'', A, \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}) \vdash E}\, Subst_{\{\mathbf{z_i}/\mathbf{y_i}\}}}{\Gamma, \Gamma'', \mathbf{H_i}(\mathbf{p}, \mathbf{z_i}) \vdash E}\, Cut \quad \ldots}{\Gamma, \Gamma'', I(\mathbf{p}) \vdash E}\, LI$$

– $LI{\rightarrow}$:

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma'', A, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}C\}_i \vdash E}{\Gamma'', A, I(\mathbf{p}){\rightarrow}C \vdash E}\, LI{\rightarrow}}{\Gamma, \Gamma'', I(\mathbf{p}){\rightarrow}C \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma'', A, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}C\}_i \vdash E}{\Gamma, \Gamma'', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}C\}_i \vdash E}\, Cut}{\Gamma, \Gamma'', I(\mathbf{p}){\rightarrow}C \vdash E}\, LI{\rightarrow}$$

Traitons maintenant le cas où A est la formule principale dans les deux prémisses. Nous procédons par cas sur la forme de A (A n'est pas atomique par hypothèse).

– $A = P{\rightarrow}B$, P atomique :

$$\dfrac{\dfrac{\Gamma, P \vdash B}{\Gamma \vdash P{\rightarrow}B}\, R{\rightarrow} \quad \dfrac{\Gamma', P, B \vdash E}{\Gamma', P, P{\rightarrow}B \vdash E}\, La{\rightarrow}}{\Gamma, \Gamma', P \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, P \vdash B \quad \Gamma', P, B \vdash E}{\Gamma, \Gamma', P, P \vdash E}\, Ind(B)}{\Gamma, \Gamma', P \vdash E}\, Contr$$

– $A = (C{\to}D){\to}B$

$$\dfrac{\dfrac{\Gamma, C{\to}D \vdash B}{\Gamma \vdash (C{\to}D){\to}B}\,R{\to} \qquad \dfrac{\Gamma', D{\to}B, C \vdash D \quad \Gamma', B \vdash E}{\Gamma', (C{\to}D){\to}B \vdash E}\,L{\to}{\to}}{\Gamma, \Gamma' \vdash E}\,Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{axiome généralisé}}{D, C \vdash D}}{D \vdash C{\to}D}\,R{\to} \quad \Gamma, C{\to}D \vdash B}{\Gamma, D \vdash B}\,Ind(C{\to}D)}{\Gamma \vdash D{\to}B}\,R{\to} \quad \Gamma', D{\to}B, C \vdash D}{\Gamma, \Gamma', C \vdash D}\,Ind(D{\to}B)}{\Gamma, \Gamma' \vdash C{\to}D}\,R{\to} \quad \Gamma, C{\to}D \vdash B}{\Gamma, \Gamma, \Gamma' \vdash B}\,Ind(C{\to}D) \quad \Gamma', B \vdash E}{\Gamma, \Gamma, \Gamma', \Gamma' \vdash E}\,Ind(B)$$

$$\vdots\ Contr$$

$$\Gamma, \Gamma' \vdash E$$

– $A = \forall x.B$

$$\dfrac{\dfrac{\Gamma \vdash B}{\Gamma \vdash \forall x.B}\,R\forall \qquad \dfrac{\Gamma', \forall x.B, B\{t/x\} \vdash E}{\Gamma', \forall x.B \vdash E}\,L\forall}{\Gamma, \Gamma' \vdash E}\,Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash B}{\Gamma \vdash B\{t/x\}}\,Subst_{\{t/x\}} \qquad \dfrac{\dfrac{\Gamma \vdash B}{\Gamma \vdash \forall x.B}\,R\forall \quad \Gamma', \forall x.B, B\{t/x\} \vdash E}{\Gamma, \Gamma', \forall x.B, B\{t/x\} \vdash E}\,Ind'}{\Gamma, \Gamma, \Gamma' \vdash E}\,Ind(B\{t/x\})$$

$$\vdots\ Contr$$

$$\Gamma, \Gamma' \vdash E$$

– $A = (\forall x.D) \rightarrow B$

$$\dfrac{\dfrac{\Gamma, \forall x.D \vdash B}{\Gamma \vdash (\forall x.D) \rightarrow B}\, R\rightarrow \quad \dfrac{\Gamma', (\forall x.D) \rightarrow B \vdash \forall x.D \quad \Gamma', B \vdash E}{\Gamma', (\forall x.D) \rightarrow B \vdash E}\, L\forall\rightarrow}{\Gamma, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \forall x.D \vdash B}{\Gamma \vdash (\forall x.D) \rightarrow B}\, R\rightarrow \quad \Gamma', (\forall x.D) \rightarrow B \vdash \forall x.D}{\Gamma, \Gamma' \vdash \forall x.D}\, Ind' \qquad \Gamma, \forall x.D \vdash B}{\Gamma, \Gamma, \Gamma' \vdash B}\, Ind(\forall x.D) \qquad \Gamma', B \vdash E}{\Gamma, \Gamma, \Gamma', \Gamma' \vdash E}\, Ind(B)$$
$$\dfrac{}{\Gamma, \Gamma' \vdash E}\, Contr$$

– $A = I(\mathbf{p})$

$$\dfrac{\dots \quad \Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \dots}{\Gamma \vdash I(\mathbf{p})}\, RI_i \quad \dfrac{\dots \quad \Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash E \quad \dots}{\Gamma', I(\mathbf{p}) \vdash E}\, LI$$
$$\dfrac{}{\Gamma, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dots \quad \Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \dots \quad \dfrac{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash E}{\Gamma', \mathbf{H_i}(\mathbf{p}, \mathbf{t_i}) \vdash E}\, Subst_{\mathbf{t_i}/\mathbf{y_i}}}{\Gamma \dots \Gamma, \Gamma' \vdash E}\, Ind(\mathbf{H_i}(\mathbf{p}, \mathbf{t_i}))}{\Gamma, \Gamma' \vdash E}\, Contr$$

– $A = I(\mathbf{p}) \rightarrow B$

$$\dfrac{\dfrac{\Gamma, I(\mathbf{p}) \vdash B}{\Gamma \vdash I(\mathbf{p}) \rightarrow B}\, R\rightarrow \quad \dfrac{\Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B\}_i \vdash E}{\Gamma', I(\mathbf{p}) \rightarrow B \vdash E}\, LI\rightarrow}{\Gamma, \Gamma' \vdash E}\, Cut$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\dfrac{\dfrac{\Gamma, I(\mathbf{p}) \vdash B}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash B}\, \text{règle 3.5}}{\Gamma \vdash \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B}\, \text{plusieurs } R\rightarrow}{\Gamma \vdash \forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B}\, \text{plusieurs } R\forall \quad \dots \quad \Gamma', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B\}_i \vdash E}{\Gamma \dots \Gamma, \Gamma' \vdash E}\, Ind(\{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \rightarrow B\}_i)$$
$$\vdots\, Contr$$
$$\Gamma, \Gamma' \vdash E$$

Ce qui clôt notre preuve par récurrence. □

L'admissibilité de la règle des coupures nous permet d'énoncer le théorème suivant.

THÉORÈME 3.18 (ÉLIMINATION DES COUPURES)
Tout séquent dérivable dans le système $LJTI + Cut$ l'est également dans $LJTI$.

Démonstration : D'après le théorème 3.17, on peut éliminer toutes les instances de *Cut* de la dérivation, en commençant par les plus hautes dans l'arbre de dérivation. □

3.4 Du calcul $LJTI$ à la tactique `firstorder`

Nous donnons d'abord des indications sur les stratégies de recherche de preuves dans le calcul $LJTI$ mises en œuvre, puis nous expliquons comment nous tirons de ces preuves des termes de preuve acceptables pour Coq.

3.4.1 Recherche de dérivations dans $LJTI$

La recherche de dérivation se fait par application des règles du bas vers le haut, simplifiant ainsi le but jusqu'à pouvoir appliquer une règle sans prémisse. L'espace de recherche à explorer est très important même si l'on fixe une borne de profondeur, car plusieurs règles peuvent s'appliquer à un but donné : aussi faut-il tout mettre en œuvre pour réduire cet espace de recherche.

Stratégies simples

Comme nous le faisions remarquer dans le chapitre 1, l'inversibilité de certaines règles de $LJTI$ nous permet de les appliquer dès que possible puisque s'il existe une dérivation, il en existe une finissant par toute règle inversible applicable. Les règles $L\forall\rightarrow$ et $L\rightarrow\rightarrow$ sont partiellement inversibles par rapport à leur seconde prémisse, nous essayons donc en premier lieu de résoudre la prémisse de gauche, et, en cas de succès, il n'y a pas besoin de remettre en cause la décision d'appliquer cette règle.

De plus, il est possible de se départir de la condition d'atomicité présente dans les règles Ax et $La\rightarrow$: les versions généralisées de ces règles sont admissibles dans $LJTI$ comme en témoignent, pour la règle Ax, le lemme 3.12, règle 1 et pour $La\rightarrow$, la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{\text{lemme 3.12, règle 2}}{\Gamma, A, A{\rightarrow}B \vdash B} \qquad \Gamma, A, B \vdash G}{\Gamma, A, A{\rightarrow}B, \Gamma, A \vdash G}\, Cut}{\vdots \text{ plusieurs } Contr}$$
$$\Gamma, A, A{\rightarrow}B \vdash G$$

Cette dernière règle est inversible comme le montre le schéma suivant :

$$\dfrac{\dfrac{\dfrac{\text{lemme 3.12, règle 1}}{B, A \vdash B}}{B \vdash A{\to}B}\, R{\to} \qquad \Gamma, A, A{\to}B \vdash G}{\Gamma, A, B \vdash G}\, Cut$$

Ces deux règles permettent de raccourcir une dérivation éventuelle en évitant de déstructurer complètement deux occurrences de la formule A jusqu'à obtenir ses atomes. Il faut donc appliquer ces règles prioritairement à toute règle pour les inductifs.

Pour affiner la stratégie employée, il faut également classer les familles inductives selon leur forme. Le premier critère distingue les familles inductives propositionnelles, dont les constructeurs sont définis par des clauses sans quantification prénexe, des autres, que l'on appelle familles du premier ordre.

Ensuite, on peut séparer les familles propositionnelles en trois catégories, selon leur nombre de constructeurs :

- Les familles sans constructeurs sont appelées absurdités (notamment $\bot$).
- Les familles à un constructeur sont appelées conjonctions ($\wedge$, $\top$, etc.).
- Les familles ayant deux contructeurs et plus sont des disjonctions ($\vee$, Dec, etc.).

On peut enfin distinguer les familles triviales, qui possèdent un constructeur défini par la clause $\square$.

Grâce à ces séparations, nous pouvons donner des priorités aux règles applicables à un séquent donné, de la plus prioritaire à la moins prioritaire :

- règles terminales : Ax, LI quand I est absurde, et RI_k lorsque la famille I a un constructeur $\mathcal{C}_k : \square$.
- règles inversibles, avec en priorité celles engendrant le moins de sous-buts : LI, $R{\to}$, $R\forall$, $La{\to}$ généralisée, RI pour I conjonction ...
- règles non-inversibles sans instanciation : $L{\to}{\to}$, $L\forall{\to}$ (avec borne), RI pour I propositionnel
- règles avec instanciation : $L\forall$ et RI

Heuristiques pour l'instanciation

Le théorème d'élimination des coupures nous donne une propriété de clôture pour les preuves dans le système $LJTI$ en affirmant qu'il n'y a pas besoin de faire un détour par une formule étrangère au problème pour résoudre celui-ci. Nous pouvons aller plus loin en définissant la notion de sous-formule atomique.

Dans $LJTI$, il n'y a malheureusement pas de propriété structurelle de sous-formule comme dans LJ puisque la règle $L{\to}{\to}$ a dans sa première prémisse une formule $D{\to}B$ qui n'est pas une sous-formule, au sens strict, de la formule $(C{\to}D){\to}B$ de sa conclusion. Pour pallier cet inconvénient, nous allons parler, non plus de sous-formules au sens strict, mais de sous-formules *atomiques*, similaires à celles que l'on trouve notamment dans [KO99].

$$\begin{array}{ll} \mathcal{SF}^+(A) = +A & \mathcal{SF}^-(A) = -A \text{ (A atomic)} \\ \mathcal{SF}^+(A{\rightarrow}B) = \mathcal{SF}^-(A) \cup \mathcal{SF}^+(B) & \mathcal{SF}^-(A{\rightarrow}B) = \mathcal{SF}^+(A) \cup \mathcal{SF}^-(B) \\ \mathcal{SF}^+(\forall x.P) = \mathcal{SF}^+(P) & \mathcal{SF}^-(\forall x.P) = \mathcal{SF}^-(P\{X/x\}) \\ \mathcal{SF}^+(I(\mathbf{p})) = \bigcup_{i,j} \mathcal{SF}^+(H_{i,j}(\mathbf{p}, \mathbf{Y_{i,k}})) & \mathcal{SF}^-(I(\mathbf{p})) = \bigcup_{i,j} \mathcal{SF}^-(H_{i,j}(\mathbf{p}, \mathbf{y_{i,k}})) \end{array}$$

(où X and $\mathbf{Y_{i,k}}$ sont des méta-variables fraîches)

$$\mathcal{SF}(\Gamma \vdash G) = \mathcal{SF}^+(G) \cup \bigcup_{H \in \Gamma} \mathcal{SF}^-(H)$$

FIG. 3.9 – Sous-formules atomiques pour les formules avec inductifs

DÉFINITION 3.19 (ATOMES SIGNÉS)
Soit un nouvel ensemble de variables $\mathcal{M}$ que l'on nomme méta-variables, on appelle contexte atomique une formule atomique formée à l'aide des termes et variables habituels et des méta-variables.

On appelle instance d'un contexte atomique A l'application $A\sigma$ d'une substitution σ au contexte A, et ce à condition que $Dom(\sigma) \subseteq \mathcal{M}$.

De plus, on définit un atome signé comme étant un contexte atomique précédé du signe $+$ ou $-$. La relation d'instance est étendue aux atomes signés avec en outre la condition de conservation du signe.

La fonction $\mathcal{SF}$ décrite par la figure 3.9 permet d'extraire d'un séquent l'ensemble de ses atomes signés. À l'aide de cette fonction, nous pouvons énoncer une propriété affaiblie de sous-formule pour $LJTI$.

THÉORÈME 3.20 (PROPRIÉTÉ DE LA SOUS-FORMULE)
Soit un arbre de dérivation dans le calcul $LJTI$, de conclusion $\Gamma_0 \vdash G_0$. Tout séquent $\Gamma \vdash G$ qui apparaît dans l'arbre de dérivation est tel que tout atome signé de $\mathcal{SF}(\Gamma \vdash G)$ est une instance d'un atome signé de $\mathcal{SF}(\Gamma_0 \vdash G_0)$ pour un ensemble de méta-variables bien choisies.

Démonstration : Pour chaque règle de $LJTI$, on vérifie que les atomes signés des prémisses sont des instances d'atomes signés de la conclusion.

Ensuite, on procède par récurrence en remarquant que la relation d'instance est transitive. □

Si l'on regarde les règles Ax et $La{\rightarrow}$ du calcul en termes d'atomes signés, cela donne :

$$\frac{}{\Gamma, -P \vdash +P}\ Ax \qquad \frac{\Gamma, -P, C \vdash G}{\Gamma, -P, (+P){\rightarrow}C \vdash G}\ Ax$$

Cela implique que pour chacune des occurrences de ces règles il y a, dans la conclusion de l'arbre de dérivation, une paire d'atomes $(+P_1, -P_2)$ de signe opposé tels que $P = P_1\sigma_1 = P_2\sigma_2$. De plus par définition, on peut mettre σ_1 et σ_2 sous la forme respective $\sigma\tau_1$

et $\rho\sigma\tau_2$, où ρ renomme les méta-variables de P_2 vers des méta-variables distinctes de celles apparaissant dans P_1, et où σ est l'unificateur le plus général de P_1 et $P_2\rho$. Cela nous montre que l'on peut restreindre les termes utilisés dans les règles $L\forall$ et RI_i à des termes obtenus par unification d'atomes de signe opposé.

De la dérivation $LJTI$ au terme de preuve Coq

Un fois que l'on a trouvé une dérivation dans le calcul $LJTI$ pour un séquent $\Gamma \vdash G$, il faut être en mesure de construire à partir de cette dérivation un terme de preuve. Pour cela, nous établissons un ensemble de règles dérivées des règles de typage du Calcul des Constructions Inductives, qui donnent pour chaque règle la façon de construire le terme de preuve pour la conclusion en fonction des termes de preuve des prémisses. L'ensemble de ces règles est donné à la figure 3.10.

Dans les règles RI_i et $L\forall$, il est nécessaire que les termes employés soient bien typés, ce qui ajoute une contrainte supplémentaire par rapport à la logique du premier ordre. Cela peut s'avérer gênant si l'on considère le terme de preuve de l'exemple qui suit : on s'aperçoit que le calcul de séquents a provoqué l'apparition d'une variable libre dans le terme de preuve.

$$\dfrac{\dfrac{\dfrac{}{\forall_s x.Px, Py \vdash Py}\,Ax}{\forall_s x.Px \vdash Py}\,L\forall}{\forall_s x.Px \vdash \exists_s x.Px}\,L\exists \qquad \dfrac{\dfrac{\dfrac{}{f : \forall_s x.Px, h : Py \vdash h : Py}\,Ax}{f : \forall_s x.Px \vdash \texttt{let } h = f\ y \texttt{ in } h : Py}\,L\forall}{f : \forall_s x.Px \vdash \texttt{ex\_intro}^3\ P\ y\ (\texttt{let } h = f\ y \texttt{ in } h) : \exists_s x.Px}\,L\exists$$

Dans le cadre du système Coq, ce problème est résolu par la transformation de ce terme en un autre but, c'est-à-dire que l'on demande à l'utilisateur de nous fournir un objet de la sorte s, ce qui est en général beaucoup plus simple que de résoudre un problème de logique du premier ordre.

3.4.2 La tactique `firstorder`

La tactique `firstorder` implante la procédure décrite ici et est distribuée dans le système Coq (version 8.0). Dans l'annexe C.1, nous effectuons une comparaison entre les tactiques `firstorder` (version 2) et `Jprover`, sur un ordinateur équipé de 512 mégaoctets de mémoire vive et d'un processeur Intel Pentium III cadencé à 1133 MHz, avec une durée maximale de 5 minutes par test.

Cette comparaison utilise ILTP [ROK05], une bibliothèque de problèmes conçus pour tester les prouveurs en logique intuitionniste, et une bonne partie d'entre eux sont purement propositionnels. Nous avons détecté le succès de la recherche de preuve et mesuré le temps total de compilation de chaque problème, ainsi que le temps necéssaire au noyau pour vérifier cette preuve en cas de succès. Dans certains cas, `Jprover` échoue pour donner une

[3] `ex_intro` est l'unique constructeur de la famille $\exists$

$$\frac{}{\Gamma, x : P \vdash x : P}\ Ax$$

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x : A.u : A{\to}B}\ R{\to} \qquad \frac{\Gamma, x : P, y : B \vdash u : G}{\Gamma, x : P, f : P{\to}B \vdash \mathtt{let}\ y = f\ x\ \mathtt{in}\ u : G}\ La{\to}$$

$$\frac{\Gamma, x : A, f : B{\to}C \vdash u : B \quad \Gamma, y : C \vdash v : G}{\Gamma, h : (A{\to}B){\to}C \vdash \begin{cases} \mathtt{let}\ g\,(x : A) = \\ \quad \mathtt{let}\ f\,(z : B) = h\,(\lambda x' : A.z)\ \mathtt{in}\ u\ \mathtt{in} \\ \mathtt{let}\ y = h\,g\ \mathtt{in}\ v \end{cases} : G}\ L{\to}{\to}$$

$$\frac{\Gamma, x : s \vdash u : A}{\Gamma \vdash \lambda x : s.u : \forall_s x.A}\ R\forall \qquad \frac{\Gamma, f : \forall x.A, h : A\{x/t\} \vdash u : G}{\Gamma, f : \forall x.A \vdash \mathtt{let}\ h = f\ t\ \mathtt{in}\ u : G}\ L\forall$$

$$\frac{\Gamma, f : (\forall x.A){\to}B \vdash u : \forall x.A \quad \Gamma, y : B \vdash v : G}{\Gamma, f : (\forall x.A){\to}B \vdash \mathtt{let}\ y = f\ u\ \mathtt{in}\ v : G}\ L\forall{\to}$$

$$\frac{\{\Gamma \vdash u_j : H_{i,j}(\mathbf{p}, \mathbf{t_i})\}_{j=1..h_i}}{\Gamma \vdash \mathcal{C}_i\,\mathbf{p}\,\mathbf{t_i}\,\mathbf{u}\ : I(\mathbf{p})}\ RI_i \qquad \frac{\{\Gamma, \mathbf{y_i}, \mathbf{g_i} : \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash u_i : G\}_{i=1..n}}{\Gamma, x : I(\mathbf{p}) \vdash \begin{cases} \mathtt{match}\ x\ \mathtt{with} \\ \quad \mid \mathcal{C}_1\,\mathbf{y_1}\,\mathbf{g_1} \Rightarrow u_1 \\ \qquad \vdots \\ \quad \mid \mathcal{C}_n\,\mathbf{y_n}\,\mathbf{g_n} \Rightarrow u_n \\ \mathtt{end} \end{cases} : G}\ LI$$

$$\frac{\Gamma, \{g_i : \forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B\}_{i=1..n} \vdash u : G}{\Gamma, f : I(\mathbf{p}){\to}B \vdash \begin{cases} \mathtt{let}\ g_1 = \lambda \mathbf{y_1}.\lambda \mathbf{h_1} : \mathbf{H_1}(\mathbf{p}, \mathbf{y_1}).f\,(\mathcal{C}_1\,\mathbf{y_1}\,\mathbf{h_1})\ \mathtt{in} \\ \qquad \vdots \\ \mathtt{let}\ g_n = \lambda \mathbf{y_n}.\lambda \mathbf{h_n} : \mathbf{H_n}(\mathbf{p}, \mathbf{y_n}).f\,(\mathcal{C}_n\,\mathbf{y_n}\,\mathbf{h_n})\ \mathtt{in}\ u \end{cases} : G}\ LI{\to}$$

FIG. 3.10 – Construction des termes de preuve pour $LJTI$

$$\frac{}{\Gamma \vdash t{=}t}\ R{=} \qquad \frac{\Gamma, C \vdash G}{\Gamma, (t{=}t){\to}C \vdash G}\ L{=}{\to}$$

$$\frac{\Gamma, s{=}t \vdash P[t]_p}{\Gamma, s{=}t \vdash P[s]_p}\ L{\overset{=}{\leftarrow}}_1 \qquad \frac{\Gamma, s{=}t \vdash P[s]_p}{\Gamma, s{=}t \vdash P[t]_p}\ L{\overset{=}{\rightarrow}}_1$$

$$\frac{\Gamma, s{=}t, P[t]_p{\to}C \vdash G}{\Gamma, s{=}t, P[s]_p{\to}C \vdash G}\ L{\overset{=}{\leftarrow}}_2 \qquad \frac{\Gamma, s{=}t, P[s]_p{\to}C \vdash G}{\Gamma, s{=}t, P[t]_p{\to}C \vdash G}\ L{\overset{=}{\rightarrow}}_2$$

Où P est une formule atomique ou une égalité.

Fig. 3.11 – Règles supplémentaires pour l'égalité

preuve car il construit un terme de preuve contenant une variable libre (par exemple pour prouver $\forall x.P\,x \vdash \exists x.P\,x$). `Firstorder` résout le problème en laissant cette variable comme un sous-but à prouver par l'utilisateur.

L'analyse des résultats permet d'affirmer que la tactique `Firstorder` est plus performante pour les problèmes de nature plus propositionnelle mais souffre d'une stratégie trop naïve d'instanciation de quantificateurs, tandis que `Jprover` a un comportement inverse : par conception, il peut avoir besoin d'augmenter arbitrairement la multiplicité pour résoudre une tautologie propositionnelle ; en revanche, il déstructure tout le problème avant de procéder à l'instanciation et peut ainsi trouver plus rapidement les bonnes instances (ou plutôt éliminer plus rapidement les mauvaises).

3.5 Ajout de l'égalité

On souhaite ajouter à $LJTI$ une égalité primitive permettant la réécriture dans les hypothèses et la conclusion. Pour cela, nous ajoutons à notre langage de formules celles constituées de l'égalité entre deux termes de même sorte. Cela va permettre d'utiliser les égalités également dans les familles inductives.

Comme nous souhaitons conserver un calcul sans contraction, nous allons devoir, comme l'a fait Dyckhoff pour LJT, restreindre le champ d'application de nos règles de réécriture avant de prouver que les règles plus générales sont admissibles dans notre calcul. Nous étendons donc $LJTI$ avec les règles données à la figure 3.11. Le système ainsi obtenu est appelé $LJTI_{=}$.

Comme pour $LJTI$, nous montrons que le calcul $LJTI_{=}$ est stable par substitution.

Lemme 3.21 (substitution)
La règle Subst est fortement admissible dans $LJTI_{=}$.

Démonstration : On ajoute à la preuve pour $LJTI$ les cas suivants :

– $R{=}$:

$$\frac{}{\Gamma \vdash t{=}t}\ R{=} \quad \Longrightarrow \quad \frac{}{\Gamma\sigma \vdash t\sigma{=}t\sigma}\ R{=}$$

$$\frac{\Gamma, C \vdash G}{\Gamma, (t{=}t){\rightarrow}C \vdash G}\, L{=}{\rightarrow} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, C \vdash G}{\Gamma\sigma, C\sigma \vdash G\sigma}\, Ind}{\Gamma\sigma, (t\sigma{=}t\sigma){\rightarrow}C\sigma \vdash G\sigma}\, L{=}{\rightarrow}$$

– $L\underset{\leftarrow 1}{=}$

$$\frac{\Gamma, s{=}t \vdash P[t]_p}{\Gamma, s{=}t \vdash P[s]_p}\, L\underset{\leftarrow}{=} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t \vdash P[t]_p}{\Gamma\sigma, s\sigma{=}t\sigma \vdash P\sigma[t\sigma]_p}\, Ind}{\Gamma\sigma, s\sigma{=}t\sigma \vdash P\sigma[s\sigma]_p}\, L\underset{\leftarrow}{=}$$

– $L\underset{\leftarrow 2}{=}$

$$\frac{\Gamma, s{=}t, P[t]_p{\rightarrow}C \vdash G}{\Gamma, s{=}t, P[s]_p{\rightarrow}C \vdash G}\, L\underset{\leftarrow}{=} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t, P[t]_p{\rightarrow}C \vdash G}{\Gamma\sigma, s\sigma{=}t\sigma, P\sigma[t\sigma]_p{\rightarrow}C\sigma \vdash G\sigma}\, Ind}{\Gamma\sigma, s\sigma{=}t\sigma, P\sigma[s\sigma]_p{\rightarrow}C\sigma \vdash G\sigma}\, L\underset{\leftarrow}{=}$$

car $(P[s]_p)\sigma = P\sigma[s\sigma]_p$ et $(P[t]_p)\sigma = P\sigma[t\sigma]_p$.

– $L\underset{\rightarrow 1}{=}$ et$L\underset{\rightarrow 2}{=}$: On procède de façon similaire à $L\underset{\leftarrow 1}{=}$ et $L\underset{\leftarrow 2}{=}$.

□

À l'aide de ce lemme, on peut prouver que les règles de réécriture non restreintes sont admissibles dans $LJTI_{=}$. Pour effectuer cette démonstration, nous réutilisons la fonction de poids définie en 3.10 en considérant les égalités comme des formules atomiques (de poids 1).

LEMME 3.22 (RÉÉCRITURE GÉNÉRALISÉE)
Les règles de réécriture suivantes sont admissibles dans $LJTI_{=}$.

$$\frac{\Gamma, s{=}t \vdash G[t]_p}{\Gamma, s{=}t \vdash G[s]_p}\, L\underset{\leftarrow 1}{=}^{*} \qquad \frac{\Gamma, s{=}t \vdash G[s]_p}{\Gamma, s{=}t \vdash G[t]_p}\, L\underset{\rightarrow 1}{=}^{*}$$

$$\frac{\Gamma, s{=}t, C[t]_p \vdash G}{\Gamma, s{=}t, C[s]_p \vdash G}\, L\underset{\leftarrow 2}{=}^{*} \qquad \frac{\Gamma, s{=}t, C[s]_p \vdash G}{\Gamma, s{=}t, C[t]_p \vdash G}\, L\underset{\rightarrow 2}{=}^{*}$$

Dans les règles L=, il est bien entendu que s ou t ne peuvent contenir des variables liées au-dessus de la position p de C ou G. En outre, C ne peut pas être la formule $s{=}t$ elle-même.

Démonstration : Pour alléger les notations, nous noterons C' la formule C réécrite par l'égalité $s = t$ et $\mathbf{p}'$ la liste de paramètres $p_1, \ldots, p_n$ dans laquelle p_i a été réécrit par $s = t$.

La démonstration se fait par récurrence sur le poids de la formule réécrite d'une part, et sur la hauteur de la dérivation d'autre part.

Si la formule réécrite n'est pas principale, on raisonne par cas sur la dernière règle de la prémisse :

– Ax : (seulement $L{=}^*_2$)

$$\dfrac{\dfrac{}{\Gamma, s{=}t, C', P \vdash P}\, Ax}{\Gamma, s{=}t, C, P \vdash P}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{}{\Gamma, s{=}t, C, P \vdash P}\, Ax$$

– $R{\to}$: (seulement $L{=}^*_2$)

$$\dfrac{\dfrac{\Gamma, s{=}t, C', A \vdash B}{\Gamma, s{=}t, C' \vdash A{\to}B}\, R{\to}}{\Gamma, s{=}t, C \vdash A{\to}B}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, C', A \vdash B}{\Gamma, s{=}t, C, A \vdash B}\, Ind'}{\Gamma, s{=}t, C \vdash A{\to}B}\, R{\to}$$

– $La{\to}$:

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, P, A \vdash G'}{\Gamma, s{=}t, P, P{\to}A \vdash G'}\, La{\to}}{\Gamma, s{=}t, P, P{\to}A \vdash G}\, L{=}^*_1 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P, A \vdash G}{\Gamma, s{=}t, P, A \vdash G}\, Ind'}{\Gamma, s{=}t, P, P{\to}A \vdash G}\, La{\to}$$

2. Pour $L{=}^*_2$:
   Si $C = P$:

$$\dfrac{\dfrac{\Gamma, s{=}t, P', A \vdash G}{\Gamma, s{=}t, P', P'{\to}A \vdash G}\, La{\to}}{\Gamma, s{=}t, P, P'{\to}A \vdash G}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P', A \vdash G}{\Gamma, s{=}t, P, A \vdash G}\, Ind(P)}{\Gamma, s{=}t, P, P{\to}A \vdash G}\, La{\to}}{\Gamma, s{=}t, P, P'{\to}A \vdash G}\, L{=}_2$$

   Sinon :

$$\dfrac{\dfrac{\Gamma, s{=}t, C', P, A \vdash G}{\Gamma, s{=}t, C', P, P{\to}A \vdash G}\, La{\to}}{\Gamma, s{=}t, C, P, P{\to}A \vdash G}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, C', P, A \vdash G}{\Gamma, s{=}t, C, P, A \vdash G}\, Ind'}{\Gamma, s{=}t, C, P, P{\to}A \vdash G}\, La{\to}$$

– $L{\to}{\to}$:

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, D, B{\to}C, A \vdash B \quad \Gamma, s{=}t, D, C \vdash G'}{\Gamma, s{=}t, D, (A{\to}B){\to}C \vdash G'}\, L{\to}{\to}}{\Gamma, s{=}t, D, (A{\to}B){\to}C \vdash G}\, L{=}^*_1$$

$$\Downarrow$$

$$\dfrac{\Gamma, s{=}t, D, B{\to}C, A \vdash B \quad \dfrac{\Gamma, s{=}t, D, C \vdash G'}{\Gamma, s{=}t, D, C \vdash G}\, Ind'}{\Gamma, s{=}t, D, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

2. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, D', B{\to}C, A \vdash B \quad \Gamma, s{=}t, D', C \vdash G}{\Gamma, s{=}t, D', (A{\to}B){\to}C \vdash G}\, L{\to}{\to}}{\Gamma, s{=}t, D, (A{\to}B){\to}C \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, D', B{\to}C, A \vdash B}{\Gamma, s{=}t, D, B{\to}C, A \vdash B}\, Ind' \quad \dfrac{\Gamma, s{=}t, D', C \vdash G}{\Gamma, s{=}t, D, C \vdash G}\, Ind'}{\Gamma, s{=}t, D, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

– $R\forall$: (seulement $L{=}^*_2$) L'ensemble des variables libres du séquent n'est pas modifié par la réécriture.

$$\dfrac{\dfrac{\Gamma, s{=}t, C' \vdash A}{\Gamma, s{=}t, C' \vdash \forall x.A}\, R\forall}{\Gamma, s{=}t, C \vdash \forall x.A}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, C' \vdash A}{\Gamma, s{=}t, C \vdash A}\, Ind'}{\Gamma, s{=}t, C \vdash \forall x.A}\, R\forall$$

– $L\forall$:

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, \forall x.A, A\{u/x\} \vdash G'}{\Gamma, s{=}t, \forall x.A \vdash G'}\, L\forall}{\Gamma, s{=}t, \forall x.A \vdash G}\, L{=}^*_1 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, \forall x.A, A\{u/x\} \vdash G'}{\Gamma, s{=}t, \forall x.A, A\{u/x\} \vdash G'}\, Ind'}{\Gamma, s{=}t, \forall x.A \vdash G}\, L\forall$$

2. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, C', \forall x.A, A\{u/x\} \vdash G}{\Gamma, s{=}t, C', \forall x.A \vdash G}\, L\forall}{\Gamma, s{=}t, C, \forall x.A \vdash G}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, C', \forall x.A, A\{u/x\} \vdash G}{\Gamma, s{=}t, C, \forall x.A \vdash G}\, Ind'}{\Gamma, s{=}t, C, \forall x.A \vdash G}\, L\forall$$

– $L\forall{\to}$:

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.A){\to}B \vdash \forall x.A \quad \Gamma, s{=}t, B \vdash G'}{\Gamma, s{=}t, (\forall x.A){\to}B \vdash G'}\, L\forall{\to}}{\Gamma, s{=}t, (\forall x.A){\to}B \vdash G}\, L{=}^*_1$$

$$\Downarrow$$

$$\dfrac{\Gamma, s{=}t, (\forall x.A){\to}B \vdash \forall x.A \quad \dfrac{\Gamma, s{=}t, B \vdash G'}{\Gamma, s{=}t, B \vdash G}\, Ind'}{\Gamma, s{=}t, (\forall x.A){\to}B \vdash G}\, L\forall{\to}$$

2. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, C', (\forall x.A){\to}B \vdash \forall x.A \quad \Gamma, s{=}t, C', B \vdash G'}{\Gamma, s{=}t, C', (\forall x.A){\to}B \vdash G}\, L\forall}{\Gamma, s{=}t, C, (\forall x.A){\to}B \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, C', s{=}t, (\forall x.A){\to}B \vdash \forall x.A}{\Gamma, C, s{=}t, (\forall x.A){\to}B \vdash \forall x.A}\, Ind' \quad \dfrac{\Gamma, C', s{=}t, B \vdash G}{\Gamma, C, s{=}t, B \vdash G}\, Ind'}{\Gamma, C, s{=}t, (\forall x.A){\to}B \vdash G}\, L\forall{\to}$$

– RI_i : (seulement $L{=}^*_2$)

$$\dfrac{\dfrac{\ldots \quad \Gamma, s{=}t, C' \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma, s{=}t, C' \vdash I(\mathbf{p})}\, RI_i}{\Gamma, s{=}t, C \vdash I(\mathbf{p})}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, s{=}t, C' \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, s{=}t, C \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind' \quad \ldots}{\Gamma, s{=}t, C \vdash I(\mathbf{p})}\, RI_i$$

– LI :

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\ldots \quad \Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G' \quad \ldots}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G'}\, LI}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G}\, L{=}^*_1$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G'}{\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G}\, Ind' \quad \ldots}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G}\, LI$$

2. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{\ldots \quad \Gamma, s{=}t, C', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G \quad \ldots}{\Gamma, s{=}t, C', I(\mathbf{p}) \vdash G}\, LI}{\Gamma, s{=}t, C, I(\mathbf{p}) \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, s{=}t, C', \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G}{\Gamma, s{=}t, C, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G}\, Ind' \quad \ldots}{\Gamma, s{=}t, C, I(\mathbf{p}) \vdash G}\, LI$$

– $LI{\rightarrow}$:

1. Pour $L{=}^*_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G'}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B \vdash G'}\, LI{\rightarrow}}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B \vdash G}\, L{=}^*_1$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G'}{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\, Ind'}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

2. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, C'\{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}{\Gamma, s{=}t, C', I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}}{\Gamma, s{=}t, C, I(\mathbf{p}){\rightarrow}B \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, s{=}t, C', \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}{\Gamma, s{=}t, C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\, Ind'}{\Gamma, s{=}t, C, I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

Si la formule réécrite est principale :

– Ax : Pour $L{=}^*_1$, la règle est déjà une instance de $L{=}_1$. Pour $L{=}^*_2$:

$$\dfrac{\dfrac{}{\Gamma, s{=}t, P' \vdash P'}\, Ax}{\Gamma, s{=}t, P \vdash P'}\, L{=}^*_2 \quad\Longrightarrow\quad \dfrac{\dfrac{}{\Gamma, s{=}t, P \vdash P}\, Ax}{\Gamma, s{=}t, P \vdash P'}\, L{=}_2$$

– $R{\rightarrow}$: Deux cas sont à traiter, l'un où l'hypothèse de récurrence utilisée est $L{=}^*_1$ et l'autre où celle utilisée est $L{=}^*_2$. Cela nous montre qu'il est impératif de démontrer simultanément l'admissibilité des deux règles.

1. Si la réécriture a lieu dans A :

$$\dfrac{\dfrac{\Gamma, s{=}t, A' \vdash B}{\Gamma, s{=}t \vdash A'{\rightarrow}B}\, R{\rightarrow}}{\Gamma, s{=}t \vdash A{\rightarrow}B}\, L{=}^*_1 \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, s{=}t, A' \vdash B}{\Gamma, s{=}t, A \vdash B}\, Ind(A)}{\Gamma, s{=}t \vdash A{\rightarrow}B}\, R{\rightarrow}$$

2. Si la réécriture a lieu dans B :

$$\dfrac{\dfrac{\Gamma, s{=}t, A \vdash B'}{\Gamma, s{=}t \vdash A{\rightarrow}B'}\, R{\rightarrow}}{\Gamma, s{=}t \vdash A{\rightarrow}B}\, L{=}^*_1 \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, s{=}t, A \vdash B'}{\Gamma, s{=}t, A \vdash B}\, Ind(B)}{\Gamma, s{=}t \vdash A{\rightarrow}B}\, R{\rightarrow}$$

– $La{\to}$: Si la réécriture a lieu dans P alors notre instance de $L{=}^*_2$ est une instance de $L{=}_2$; sinon la réécriture a lieu dans B :

$$\dfrac{\dfrac{\Gamma, s{=}t, P, B' \vdash G}{\Gamma, s{=}t, P, P{\to}B' \vdash G}\, La{\to}}{\Gamma, s{=}t, P, P{\to}B \vdash G}\, L{=}^*_2 \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P, B' \vdash G}{\Gamma, s{=}t, P, B \vdash G}\, Ind(B)}{\Gamma, s{=}t, P, P{\to}B \vdash G}\, La{\to}$$

– $L{\to}{\to}$

1. Si la réécriture se fait dans A :

$$\dfrac{\dfrac{\Gamma, s{=}t, A', B{\to}C \vdash B' \quad \Gamma, s{=}t, C' \vdash G}{\Gamma, s{=}t, (A'{\to}B){\to}C \vdash G}\, L{\to}{\to}}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, s{=}t, A', B{\to}C \vdash B}{\Gamma, s{=}t, A, B{\to}C \vdash B}\, Ind(A) \qquad \Gamma, s{=}t, C \vdash G}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

2. Si la réécriture se fait dans B :

$$\dfrac{\dfrac{\Gamma, s{=}t, A, B'{\to}C \vdash B' \quad \Gamma, s{=}t, C \vdash G}{\Gamma, s{=}t, (A{\to}B'){\to}C \vdash G}\, L{\to}{\to}}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, A, B'{\to}C \vdash B'}{\Gamma, s{=}t, A, B{\to}C \vdash B'}\, Ind(B{\to}C)}{\Gamma, s{=}t, A, B{\to}C \vdash B}\, Ind(B) \qquad \Gamma, s{=}t, C \vdash G}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

3. Si la réécriture se fait dans C :

$$\dfrac{\dfrac{\Gamma, s{=}t, A, B{\to}C' \vdash B \quad \Gamma, s{=}t, C' \vdash G}{\Gamma, s{=}t, (A{\to}B){\to}C' \vdash G}\, L{\to}{\to}}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{=}^*_2$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, s{=}t, A, B{\to}C' \vdash B}{\Gamma, s{=}t, A, B{\to}C \vdash B}\, Ind(B{\to}C) \quad \dfrac{\Gamma, s{=}t, C' \vdash G}{\Gamma, s{=}t, C \vdash G}\, Ind(C)}{\Gamma, s{=}t, (A{\to}B){\to}C \vdash G}\, L{\to}{\to}$$

– $L\underset{\leftarrow_1}{=}$: Nous traitons ici le cas $L\underset{\leftarrow_2}{=}^*$ lorsque la réécriture se produit à la position p dans le membre gauche u de l'égalité ; les autres cas se traitent de façon analogue :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, u[t]_p{=}v \vdash P[v]_{p'}}{\Gamma, s{=}t, u[t]_p{=}v \vdash P[u[t]_p]_{p'}}\, L\underset{\leftarrow_1}{=}}{\Gamma, s{=}t, u[s]_p{=}v \vdash P[u[t]_p]_{p'}}\, L\underset{\leftarrow_2}{=}^*}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, u[t]_p{=}v \vdash P[v]_{p'}}{\Gamma, s{=}t, u[s]_p{=}v \vdash P[v]_{p'}}\, Ind'}{\Gamma, s{=}t, u[s]_p{=}v \vdash P[u[s]_p]_{p'}}\, L\underset{\leftarrow_1}{=}}{\Gamma, s{=}t, u[s]_p{=}v \vdash P[u[t]_p]_{p'}}\, L\underset{\rightarrow_1}{=}}$$

– $L\underset{\rightarrow_1}{=}$,$L\underset{\leftarrow_2}{=}$ et $L\underset{\rightarrow_2}{=}$: de façon similaire au cas $L\underset{\leftarrow_1}{=}$.

– $R{=}$: L'instance de $L{=}_1^*$ est aussi une instance de $L{=}_1$.

– $L{=}{\rightarrow}$:

$$\dfrac{\dfrac{\Gamma, s{=}t, C' \vdash G}{\Gamma, s{=}t, u''{=}u''{\rightarrow}C' \vdash G}\, L{=}{\rightarrow}}{\Gamma, s{=}t, u'{=}u{\rightarrow}C \vdash P}\, L{=}_2^* \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\Gamma, s{=}t, C' \vdash G}{\Gamma, s{=}t, C \vdash G}\, Ind(C)}{\Gamma, s{=}t, u''{=}u''{\rightarrow}C \vdash G}\, L{=}{\rightarrow}}{\Gamma, s{=}t, u'{=}u{\rightarrow}C \vdash P}\, L{=}_2$$

– $R\forall$:

$$\dfrac{\dfrac{\Gamma, s{=}t \vdash C[t]_p}{\Gamma, s{=}t \vdash \forall x.(C[t]_p)}\, R\forall}{\Gamma, s{=}t \vdash \forall x.(C[s]_p)}\, L{=}_1^* \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t \vdash C[t]_p}{\Gamma, s{=}t \vdash (C[s]_p)}\, Ind(C)}{\Gamma, s{=}t \vdash \forall x.(C[s]_p)}\, R\forall$$

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, \forall x.(C[t]_p), (C[t]_p)\{x/u\} \vdash G}{\Gamma, s{=}t, \forall x.(C[t]_p) \vdash G}\, L\forall}{\Gamma, s{=}t, \forall x.(C[s]_p) \vdash G}\, L{=}_2^*}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, \forall x.(C[t]_p), (C\{x/u\}[t]_p) \vdash G}{\Gamma, s{=}t, \forall x.(C[s]_p), (C[t]_p\{x/u\}) \vdash G}\, Ind'}{\Gamma, s{=}t, \forall x.(C[s]_p), (C[s]_p)\{x/u\} \vdash G}\, Ind(C\{x/u\})}{\Gamma, s{=}t, \forall x.(C[s]_p) \vdash G}\, L\forall$$

Les positions où x apparaît sont indépendantes de celles où s apparaît : on peut bien affirmer que :

$$(C[t]_p)\{x/u\} = (C\{x/u\})[t]_p$$

– $L\forall\rightarrow$:

1. Si la réécriture a lieu dans C :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.C')\rightarrow D \vdash \forall x.C' \quad \Gamma, s{=}t, D \vdash G}{\Gamma, s{=}t, (\forall x.C')\rightarrow D \vdash G}\, L\forall\rightarrow}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash G}\, L{=}^*_2}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.C')\rightarrow D \vdash \forall x.C'}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash \forall x.C'}\, Ind'}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash \forall x.C}\, Ind(\forall x.C) \qquad \Gamma, s{=}t, D \vdash G}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash G}\, L\forall\rightarrow$$

2. Si la réécriture a lieu dans D :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.C)\rightarrow D' \vdash \forall x.C \quad \Gamma, s{=}t, D' \vdash G}{\Gamma, s{=}t, (\forall x.C)\rightarrow D' \vdash G}\, L\forall\rightarrow}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash G}\, L{=}^*_2}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.C)\rightarrow D' \vdash \forall x.C}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash \forall x.C}\, Ind' \qquad \dfrac{\Gamma, s{=}t, D' \vdash G}{\Gamma, s{=}t, D \vdash G}\, Ind(D)}{\Gamma, s{=}t, (\forall x.C)\rightarrow D \vdash G}\, L\forall\rightarrow$$

– RI_i : Il faut réécrire à toutes les positions où le paramètre p_i concerné par la réécriture apparaît, et ce dans chacune des hypothèses $H_{i,j}(\mathbf{p}, \mathbf{t_i})$. C'est possible car le sous-terme réécrit dans p_i ne contient aucune variable liée par λ.

$$\dfrac{\dfrac{\dots \quad \Gamma, s{=}t \vdash H_{i,j}(\mathbf{p}', \mathbf{t_i}) \quad \dots}{\Gamma, s{=}t \vdash I(\mathbf{p}')}\, RI_i}{\Gamma, s{=}t \vdash I(\mathbf{p})}\, L{=}^*_1$$

$$\Downarrow$$

$$\dfrac{\dots \quad \begin{array}{l}\Gamma, s{=}t \vdash H_{i,j}(\mathbf{p}', \mathbf{t_i}) \\ \quad \vdots \text{ plusieurs étapes } Ind(H_{i,j}(\mathbf{p}, \mathbf{t_i})) \\ \Gamma, s{=}t \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})\end{array} \quad \dots}{\Gamma, s{=}t \vdash I(\mathbf{p})}\, RI_i$$

– LI : Il faut réécrire à toutes les positions où le paramètre p_i modifié apparaît (comme

au cas précédent).

$$\dfrac{\dfrac{\ldots \quad \Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}', \mathbf{y_i}) \vdash G \quad \ldots}{\dfrac{\Gamma, s{=}t, I(\mathbf{p}') \vdash G}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G}\, L{=}^*_2}\, LI}{\Downarrow}$$

$$\dfrac{\begin{array}{l}\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}', \mathbf{y_i}) \vdash G \\ \quad \vdots \text{ plusieurs étapes } Ind(\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})) \\ \ldots \quad \Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G\end{array} \qquad \ldots}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G}\, LI$$

– $LI{\rightarrow}$: (comme aux cas précédents)

1. Si la réécriture a lieu dans $\mathbf{p}$:

$$\dfrac{\dfrac{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}', \mathbf{y_i}){\rightarrow}B\}_i \vdash G}{\Gamma, s{=}t, I(\mathbf{p}'){\rightarrow}B \vdash G}\, L{=}^*_2}{\Downarrow}\, LI{\rightarrow}$$

$$\dfrac{\begin{array}{l}\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}', \mathbf{y_i}){\rightarrow}B\}_i \vdash G \\ \quad \vdots \text{ plusieurs } Ind(\{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B\}_i) \\ \Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B\}_i \vdash G\end{array}}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

2. Si la réécriture a lieu dans B :

$$\dfrac{\dfrac{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B'\}_i \vdash G}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B' \vdash G}\, L{=}^*_2}{\Downarrow}\, LI{\rightarrow}$$

$$\dfrac{\begin{array}{l}\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B'\}_i \vdash G \\ \quad \vdots \text{ plusieurs } Ind(\{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B\}_i) \\ \Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\rightarrow}B\}_i \vdash G\end{array}}{\Gamma, s{=}t, I(\mathbf{p}){\rightarrow}B \vdash G}\, LI{\rightarrow}$$

Ce qui clôt notre démonstration par récurrence. □

Grâce à ce lemme, nous pouvons affirmer que le système $LJTI_=$ permet de dériver toutes les conséquences possibles d'une égalité, malgré les restrictions imposées aux règles $L{=}_1$ et $L{=}_2$.

Théorème 3.23
La règle d'affaiblissement W ci-dessous est fortement admissible dans $LJTI_{=}$.

$$\frac{\Gamma \vdash G}{\Gamma, \Gamma' \vdash G}\, W$$

Démonstration : On procède de façon identique au cas sans égalité, avec les cas supplémentaires suivants :

$$\frac{}{\Gamma \vdash t{=}t}\, R{=} \quad \Longrightarrow \quad \frac{}{\Gamma, \Gamma' \vdash t{=}t}\, R{=}$$

$$\frac{\Gamma, C \vdash G}{\Gamma, t{=}t{\to}C \vdash G}\, L{=}{\to} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, C \vdash G}{\Gamma, \Gamma', C \vdash G}\, Ind}{\Gamma, \Gamma', t{=}t{\to}C \vdash G}\, L{=}{\to}$$

$$\frac{\Gamma, s{=}t \vdash P'}{\Gamma, s{=}t \vdash P}\, L{=}_1 \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t \vdash P'}{\Gamma, \Gamma', s{=}t \vdash P'}\, Ind}{\Gamma, \Gamma', s{=}t \vdash P}\, L{=}_1$$

$$\frac{\Gamma, s{=}t, P'{\to}C \vdash G}{\Gamma, s{=}t, P{\to}C \vdash G}\, L{=}_2 \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t, P'{\to}C \vdash G}{\Gamma, \Gamma', s{=}t, P'{\to}C \vdash G}\, Ind}{\Gamma, \Gamma', s{=}t, P{\to}C \vdash G}\, L{=}_2$$

□

Lemme 3.24 (règles inversibles)
Les règles du lemme 3.9 (page 74) ainsi que les règles suivantes sont fortement admissibles dans $LJTI_{=}$:

$$\frac{\Gamma, t{=}t \vdash G}{\Gamma \vdash G} \qquad (3.11) \qquad\qquad \frac{\Gamma, s{=}t{\to}C \vdash G}{\Gamma, C \vdash G} \qquad (3.12)$$

Démonstration : On trouvera le détail de ces preuves dans l'annexe A.2. □

Lemme 3.25
Les règles axiome généralisé *et* modus ponens *sont aussi admissibles dans $LJTI_{=}$.*

Démonstration : On ajoute les cas suivants à la preuve du lemme 3.12, règle 1 :

$$\frac{\dfrac{}{\Gamma, s{=}t \vdash s{=}s}\, R{=}}{\Gamma, s{=}t \vdash s{=}t}\, L{=}_{\to 1} \qquad \frac{\dfrac{\dfrac{\dfrac{Ind(C)}{\Gamma, C, s{=}t \vdash C}}{\Gamma, s{=}s{\to}C, s{=}t \vdash C}\, L{=}{\to}}{\Gamma, s{=}t{\to}C, s{=}t \vdash C}\, L{=}_{\to 2}}{\Gamma, s{=}t{\to}C \vdash s{=}t{\to}C}\, R{\to}$$

□

LEMME 3.26
Les règles suivantes sont admissibles dans $LJTI_=$:

$$\frac{\Gamma \vdash D \quad \Gamma, B \vdash E}{\Gamma, D{\to}B \vdash E}\ (1) \qquad \frac{\Gamma, (C{\to}D){\to}B \vdash E}{\Gamma, C, D{\to}B, D{\to}B \vdash E}\ (2) \tag{3.13}$$

Démonstration : Pour la règle (1), on ajoute les cas suivants à la preuve du lemme 3.13 :

- $R{=}$:

$$\dfrac{\dfrac{}{\Gamma \vdash t{=}t}\ R{=} \qquad \Gamma, B \vdash E}{\Gamma, t{=}t{\to}B \vdash E} \implies \frac{\Gamma, B \vdash E}{\Gamma, t{=}t{\to}B \vdash E}\ L{=}{\to}$$

- $L{=}{\to}$:

$$\dfrac{\dfrac{\Gamma, C \vdash D}{\Gamma, t{=}t{\to}C \vdash D}\ L{=}{\to} \qquad \Gamma, t{=}t{\to}C, B \vdash E}{\Gamma, t{=}t{\to}C, D{\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\Gamma, C \vdash D \qquad \dfrac{\Gamma, t{=}t{\to}C, B \vdash E}{\vdots\ \text{règle 3.12}\ \Gamma, C, B \vdash E}}{\dfrac{\Gamma, C, D{\to}B \vdash E}{\Gamma, t{=}t{\to}C, D{\to}B \vdash E}\ L{=}{\to}}\ Ind$$

- $L{=}_1$: D est une formule atomique ou une égalité.

$$\dfrac{\dfrac{\Gamma, s{=}t \vdash D'}{\Gamma, s{=}t \vdash D}\ L{=}_1 \qquad \Gamma, s{=}t, B \vdash E}{\Gamma, s{=}t, D{\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\Gamma, s{=}t \vdash D' \quad \Gamma, s{=}t, B \vdash E}{\dfrac{\Gamma, s{=}t, D'{\to}B \vdash E}{\Gamma, s{=}t, D{\to}B \vdash E}\ L{=}_2}\ Ind$$

– $L{=}_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, C' \vdash D}{\Gamma, s{=}t, C \vdash D}\, L{=}_2 \quad \Gamma, s{=}t, C, B \vdash E}{\Gamma, s{=}t, C, D{\to}B \vdash E}$$

$$\Downarrow$$

$$\dfrac{\Gamma, s{=}t, C' \vdash D \quad \dfrac{\Gamma, s{=}t, C, B \vdash E}{\Gamma, s{=}t, C', B \vdash E}\, L{=}_2}{\dfrac{\Gamma, s{=}t, C', D{\to}B \vdash E}{\Gamma, s{=}t, C, D{\to}B \vdash E}\, L{=}_2}\, Ind$$

Ce qui clôt la preuve d'admissibilité de (1).

Pour la règle (2), on suit la preuve du lemme 3.14. $L{=}_2$ commute sans problème puisque $C{\to}D$ n'est ni une formule atomique, ni une égalité et ne peut donc être réécrite. Les autres règles commuttent de façon immédiate. □

Pour réussir à démontrer l'admissibilité de la règle de contraction, il nous faut en fait montrer l'admissibilité d'une règle plus générale qui permet la contraction de deux formules égales *modulo* réécriture par les égalités du contexte.

Définition 3.27 (congruence *modulo* Γ)
Deux formules A et A' sont congrues modulo Γ, *ce que l'on note $A \overset{\Gamma}{=} A'$ si, et seulement si, pour toute formule C, il existe une dérivation de $\Gamma, A \vdash C$ à partir de $\Gamma, A' \vdash C$ utilisant seulement la règle $L{=}_2$.*

On remarque que le choix de C n'influe pas sur le fait que les dérivations par $L{=}_2$ soient possibles ou non. De plus, par construction, $A \overset{\Gamma}{=} A'$ si, et seulement si $A' = A$ ou $A = P{\to}C$ et $A' = P'{\to}C$ avec P et P' deux égalités ou formules atomiques.

Théorème 3.28 (Admissibilité de la contraction)
La règle de contraction modulo *ci-dessous est admissible dans $LJTI_=$.*

$$\dfrac{\Gamma, A, A' \vdash G}{\Gamma, A \vdash G}\, Contr_= \ (A \overset{\Gamma}{=} A')$$

Démonstration : La preuve est effectuée par récurrence sur le poids de la formule contractée et la hauteur de la dérivation.

Tout d'abord, examinons le cas où la formule contractée n'est pas principale dans la règle précédente. Pour les règles de $LJTI$, on procède de la même façon que pour le théorème 3.15. Nous indiquons ici les cas des nouvelles règles :

– Si la dernière règle est $R{=}$, on a :

$$\dfrac{\dfrac{}{\Gamma, A, A' \vdash s{=}s}\, R{=}}{\Gamma, A \vdash s{=}s}\, Contr_= \quad \Longrightarrow \quad \dfrac{}{\Gamma, A \vdash s{=}s}\, R{=}$$

– Si la dernière règle est $L{=}_1$, on a :

$$\dfrac{\dfrac{\Gamma, s{=}t, A, A' \vdash P'}{\Gamma, s{=}t, A, A' \vdash P}\, L{=}_1}{\Gamma, s{=}t, A \vdash P}\, Contr_= \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, A, A' \vdash P'}{\Gamma, s{=}t, A \vdash P'}\, Ind'}{\Gamma, s{=}t, A \vdash P}\, L{=}_1$$

– Si la dernière règle est $L{=}_2$, trois cas sont possibles :

1. La formule réécrite disparaît par contraction : on utilise la transitivité de la congruence.

$$\dfrac{\dfrac{\Gamma, s{=}t, P{\to}C, P''{\to}C \vdash G}{\Gamma, s{=}t, P{\to}C, P'{\to}C \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\to}C \vdash G}\, Contr_= \quad \Longrightarrow \quad \dfrac{\Gamma, s{=}t, P{\to}C, P''{\to}C \vdash G}{\Gamma, s{=}t, P{\to}C \vdash G}\, Ind$$

2. La formule réécrite est contractée mais ne disparaît pas : on utilise la transitivité de la congruence.

$$\dfrac{\dfrac{\Gamma, s{=}t, P''{\to}C, P'{\to}C \vdash G}{\Gamma, s{=}t, P{\to}C, P'{\to}C \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\to}C \vdash G}\, Contr_= \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P''{\to}C, P'{\to}C \vdash G}{\Gamma, s{=}t, P''{\to}C \vdash G}\, Ind}{\Gamma, s{=}t, P{\to}C \vdash G}\, L{=}_2$$

3. La formule réécrite n'est pas contractée :

$$\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, A, A' \vdash G}{\Gamma, s{=}t, P{\to}C, A, A' \vdash G}\, L{=}_1}{\Gamma, s{=}t, P{\to}C, A \vdash G}\, Contr_= \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, A, A' \vdash G}{\Gamma, s{=}t, P'{\to}C, A \vdash G}\, Ind'}{\Gamma, s{=}t, P{\to}C, A \vdash G}\, L{=}_1$$

– Si la dernière règle est $L{=}{\to}$, on a :

$$\dfrac{\dfrac{\Gamma, C, A, A' \vdash G}{\Gamma, s{=}s{\to}C, A, A' \vdash G}\, L{=}_1}{\Gamma, s{=}s{\to}C, A \vdash G}\, Contr_= \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, C, A, A' \vdash G}{\Gamma, C, A \vdash G}\, Ind'}{\Gamma, s{=}s{\to}C, A \vdash G}\, L{=}_1$$

Traitons maintenant les cas où la formule réécrite est principale, par cas selon la forme de A. Les cas non mentionnés ici se traitent comme dans la preuve du théorème 3.15.

– Le cas où A est de la forme $P{\to}B$ avec P atomique diffère de la preuve pour $LJTI$:

1. Cas où la formule principale est conservée :

$$\dfrac{\dfrac{\Gamma, P, B, P'{\to}B \vdash G}{\Gamma, P, P{\to}B, P'{\to}B \vdash G}\, La{\to}}{\Gamma, P, P{\to}B \vdash G}\, Contr_= \quad \Longrightarrow \quad \dfrac{\begin{array}{c}\Gamma, P, B, P'{\to}B \vdash G \\ \vdots \text{ règle 3.2} \\ \Gamma, P, B, B \vdash G\end{array}}{\Gamma, P, B \vdash G}\, Ind(B)}{\Gamma, P, P{\to}B \vdash G}\, La{\to}$$

2. Cas où la formule principale est effacée : on utilise la définition de la congruence.

$$\dfrac{\dfrac{\dfrac{\Gamma, P', P{\to}B, B \vdash G}{\Gamma, P', P{\to}B, P'{\to}B \vdash G}\, La{\to}}{\Gamma, P', P{\to}B \vdash G}\, Contr_{=}}{} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P', P{\to}B, B \vdash G}{\vdots\ \text{règle 3.2}}}{\dfrac{\dfrac{\Gamma, P', B, B \vdash G}{\Gamma, P', B \vdash G}\, Ind(B)}{\Gamma, P', P'{\to}B \vdash G}\, La{\to}} \quad \vdots\ \text{plusieurs } L{=}_2 \quad \Gamma, P', P{\to}B \vdash G$$

– Si A est de la forme $s{=}t$, on a, avec P atomique ou égalité :

1. Cas $L{=}_1$:

$$\dfrac{\dfrac{\Gamma, s{=}t, s{=}t \vdash P'}{\Gamma, s{=}t, s{=}t \vdash P}\, L{=}_1}{\Gamma, s{=}t \vdash P}\, Contr_{=} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, s{=}t \vdash P'}{\Gamma, s{=}t \vdash P'}\, Ind}{\Gamma, s{=}t \vdash P}\, L{=}_1$$

2. Cas $L{=}_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, s{=}t, P'{\to}C \vdash G}{\Gamma, s{=}t, s{=}t, P{\to}C \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\to}C \vdash G}\, Contr_{=} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, s{=}t, P'{\to}C \vdash G}{\Gamma, s{=}t, P'{\to}C \vdash G}\, Ind}{\Gamma, s{=}t, P{\to}C \vdash G}\, L{=}_2$$

– Si A est de la forme $s{=}t{\to}C$: on utilise la définition de la congruence.

1. Si la formule principale est conservée :

$$\dfrac{\dfrac{\dfrac{\Gamma, C, s'{=}s''{\to}C \vdash G}{\Gamma, s{=}s{\to}C, s'{=}s''{\to}C \vdash G}\, L{=}{\to}}{\Gamma, s{=}s{\to}C \vdash G}\, Contr_{=}}{} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s'{=}s''{\to}C, C \vdash G}{\vdots\ \text{plusieurs } L{=}_2}}{\dfrac{\dfrac{\Gamma, s{=}s{\to}C, C \vdash G}{\vdots\ \text{règle 3.12}}}{\dfrac{\dfrac{\Gamma, C, C \vdash G}{\Gamma, C \vdash G}\, Ind(C)}{\Gamma, s{=}s{\to}C \vdash G}\, L{=}{\to}}}$$

2. Si la formule principale est effacée :

$$\dfrac{\dfrac{\Gamma, C, s'{=}s''{\to}C \vdash G}{\Gamma, s{=}s{\to}C, s'{=}s''{\to}C \vdash G}\, L{=}{\to}}{\Gamma, s'{=}s''{\to}C \vdash G}\, Contr_{=} \quad \Longrightarrow \quad \begin{array}{c} \Gamma, s'{=}s''{\to}C, C \vdash G \\ \vdots \text{ plusieurs } L{=}_2 \\ \Gamma, s{=}s{\to}C, C \vdash G \\ \vdots \text{ règle 3.12} \\ \dfrac{\dfrac{\Gamma, C, C \vdash G}{\Gamma, C \vdash G}\, Ind(C)}{\Gamma, s{=}s{\to}C \vdash G}\, L{=}{\to} \\ \vdots \text{ plusieurs } L{=}_2 \\ \Gamma, s'{=}s''{\to}C \vdash G \end{array}$$

Ceci clôt la preuve par récurrence. □

Comme pour la contraction, on généralise la coupure en utilisant la relation de congruence, afin de prouver son admissibilité.

THÉORÈME 3.29 (ADMISSIBILITÉ DE LA COUPURE)
La règle de coupure modulo *ci-dessous est admissible dans* $LJTI_{=}$.

$$\dfrac{\Gamma \vdash A \quad \Gamma', A' \vdash E}{\Gamma, \Gamma' \vdash E}\, Cut_{=}\ (A \overset{\Gamma'}{=} A')$$

Démonstration : La preuve est réalisée par récurrence, (i) sur le poids de A, (ii) sur la hauteur de la dérivation de la première prémisse, et (iii) sur la hauteur de la dérivation de la seconde prémisse.

Si A n'est pas la formule principale, on fait comme pour le théorème 3.17, et on ajoute les cas suivants :

– $L{=}_1$: A est une formule atomique ou une égalité donc $A = A'$.

$$\dfrac{\dfrac{\Gamma, s{=}t \vdash A''}{\Gamma, s{=}t \vdash A}\, L{=}_1 \quad \Gamma', A \vdash E}{\Gamma, s{=}t, \Gamma' \vdash E}\, Cut_{=} \quad \Longrightarrow \quad \dfrac{\Gamma, s{=}t \vdash A'' \quad \dfrac{\dfrac{\Gamma', A \vdash E}{\Gamma', s{=}t, A \vdash E}\, W}{\Gamma', s{=}t, A'' \vdash E}\, L{=}^{*}_{2}}{\Gamma, s{=}t, \Gamma', s{=}t \vdash E}\, Ind'}{\Gamma, s{=}t, \Gamma' \vdash E}\, Contr_{=}$$

– $L{=}_2$: P est une formule atomique ou une égalité.

$$\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C \vdash A}{\Gamma, s{=}t, P{\to}C \vdash A}\, L{=}_2 \quad \Gamma', A' \vdash E}{\Gamma, s{=}t, P{\to}C, \Gamma' \vdash E}\, Cut_= \qquad \Downarrow \qquad \dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C \vdash A \quad \Gamma', A' \vdash E}{\Gamma, s{=}t, P'{\to}C, \Gamma' \vdash E}\, Ind'}{\Gamma, s{=}t, P{\to}C, \Gamma' \vdash E}\, L{=}_2$$

Si A est une formule principale dans la première prémisse mais pas A' dans la seconde, on raisonne par cas sur la dernière règle de la seconde prémisse.

– $R{=}$:

$$\dfrac{\Gamma \vdash A \quad \dfrac{}{\Gamma', A' \vdash s{=}s}\, R{=}}{\Gamma, \Gamma' \vdash s{=}s}\, Cut_= \quad \Longrightarrow \quad \dfrac{}{\Gamma, \Gamma' \vdash s{=}s}\, R{=}$$

– $L{=}_1$: P est une formule atomique ou une égalité.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma', A', s{=}t \vdash P'}{\Gamma', A', s{=}t \vdash P}\, L{=}_1}{\Gamma, \Gamma', s{=}t \vdash P}\, Cut_= \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma \vdash A \quad \Gamma', A', s{=}t \vdash P'}{\Gamma, \Gamma', s{=}t \vdash P'}\, Ind''}{\Gamma, \Gamma', s{=}t \vdash P}\, L{=}_1$$

– $L{=}_2$: deux cas peuvent se présenter :

1. A' n'est pas la formule réécrite. Soit P une formule atomique ou une égalité.

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Gamma', A', s{=}t, P'{\to}C \vdash E}{\Gamma', A', s{=}t, P{\to}C \vdash E}\, L{=}_2}{\Gamma, \Gamma', s{=}t, P{\to}C \vdash E}\, Cut_= \qquad \Downarrow \qquad \dfrac{\dfrac{\Gamma \vdash A \quad \Gamma', A', s{=}t, P'{\to}C \vdash E}{\Gamma, \Gamma', s{=}t, P'{\to}C \vdash E}\, Ind''}{\Gamma, \Gamma', s{=}t, P{\to}C \vdash E}\, L{=}_2$$

2. $A' = P'{\to}C$ est réécrite, où P' est une formule atomique ou une égalité.

$$\dfrac{\Gamma \vdash P{\to}C \quad \dfrac{\Gamma', s{=}t, P''{\to}C \vdash E}{\Gamma', s{=}t, P'{\to}C \vdash E}\, L{=}_2}{\Gamma, \Gamma', s{=}t \vdash E}\, Cut_= \qquad \Downarrow \qquad \dfrac{\Gamma \vdash P{\to}C \quad \Gamma', s{=}t, P''{\to}C \vdash E}{\Gamma, \Gamma', s{=}t \vdash E}\, Ind''$$

Enfin, si A et A' sont principales dans chacune des prémisses :

- Si $A = P{\rightarrow}B$ et $A' = P'{\rightarrow}B$ (P,P' atomiques) : on utilise la définition de la congruence.

$$\dfrac{\dfrac{\Gamma, P \vdash B}{\Gamma \vdash P{\rightarrow}B}\, R{\rightarrow} \quad \dfrac{\Gamma', P', B \vdash E}{\Gamma', P', P'{\rightarrow}B \vdash E}\, La{\rightarrow}}{\Gamma, \Gamma', P' \vdash E}\, Cut_{=}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, P \vdash B \quad \Gamma', P', B \vdash E}{\Gamma, \Gamma', P, P' \vdash E}\, Ind(B)}{\begin{array}{c}\vdots\ L{=}_2^* \text{ plusieurs fois}\\ \Gamma, \Gamma', P', P' \vdash E\end{array}}\, Contr_{=}$$

- Si $A = s'{=}s''{\rightarrow}B$ et $A' = s{=}s{\rightarrow}B$:

$$\dfrac{\dfrac{\Gamma, s'{=}s'' \vdash B}{\Gamma \vdash s'{=}s''{\rightarrow}B}\, R{\rightarrow} \quad \dfrac{\Gamma', B \vdash E}{\Gamma', s{=}s{\rightarrow}B \vdash E}\, L{=}{\rightarrow}}{\Gamma, \Gamma' \vdash E}\, Cut_{=}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, s'{=}s'' \vdash B \quad \Gamma', B \vdash E}{\Gamma, \Gamma', s'{=}s'' \vdash E}\, Ind(B)}{\begin{array}{c}\vdots\ L{=}_2^* \text{ plusieurs fois}\\ \Gamma, \Gamma', s{=}s \vdash E\end{array}}\ \text{règle 3.11}$$

$$\Gamma, \Gamma' \vdash E$$

- Si $A = A' = s{=}s$:

$$\dfrac{\dfrac{}{\Gamma \vdash s{=}s}\, R{=} \quad \dfrac{\Gamma', s{=}s \vdash E}{\Gamma', s{=}s \vdash E}\, L{=}}{\Gamma, \Gamma' \vdash E}\, Cut_{=}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma', s{=}s \vdash E}{\Gamma' \vdash E}\ \text{règle 3.11}}{\Gamma, \Gamma' \vdash E}\, W$$

Ce qui clôt la preuve de notre théorème. □

Comme dans le cas de $LJTI$, ce théorème permet d'établir l'équivalence entre le système $LJTI_{=} + Cut$ et le système $LJTI_{=}$ pur.

3.6 Conclusion

Dans ce chapitre, nous avons montré qu'il était possible d'intégrer le formalisme des connecteurs inductifs à la logique du premier ordre, et d'obtenir pour ce système formel un calcul de séquents sans contraction satisfaisant la propriété d'élimination des coupures. Ceci en fait un bon candidat pour la mise au point de procédures de recherche de preuves dans le fragment du premier ordre du Calcul des Constructions Inductives. C'est ce que nous avons réalisé avec la tactique `firstorder`. Nous avons montré que l'on pouvait également ajouter à ce formalisme des règles de raisonnement équationnel. La mise au point d'une tactique utilisant cette extension serait l'étape suivante naturelle pour ce travail.

Chapitre 4

Preuves réflexives en logique du premier ordre avec égalité

Dans ce chapitre, nous proposons une alternative à la méthode proposée dans la section 3.4.1 pour transformer une dérivation $LJTI$ en preuve Coq. Celle-ci fait appel à la réflexion calculatoire que nous avons définie dans l'introduction.

4.1 Réflexion en Théorie des Types

4.1.1 Expériences précédentes avec la réflexion

Dimitri Hendriks [Hen03, Hen02] a déjà utilisé ce genre de schéma de réflexion afin de prouver des propriétés des systèmes de preuve, ou pour certifier, avec l'aide de Marc Bezem et Hans de Nivelle [BHdN02], une procédure de mise sous forme clausale, dans le but d'interfacer Coq avec le démonstrateur par résolution Bliksem.

Pierre Crégut [Cré01] a utilisé la réflexion dans le domaine de l'arithmétique linéaire sans quantificateurs. Cuihtlauac Alvarado a automatisé la production de preuves par réécriture au moyen de fonctions transformant des arbres [Alv02].

Nous proposons ici un cadre plus général puisque l'ensemble de notre travail est paramétré par l'ensemble de sortes et la signature utilisée, et qu'un soin particulier est apporté à l'efficacité des fonctions de vérification de preuves écrites dans le langage de Coq.

4.1.2 Conversion et réflexion : un exemple simple

Pour montrer comment la règle de conversion permet d'utiliser la réflexion calculatoire dans le Calcul des Constructions Inductives, nous allons partir d'un petit exemple en Coq : l'égalité des entiers de Peano. Les entiers de Peano sont définis en Coq comme un type inductif `nat` engendré par deux constructeurs : le premier, `O`, représente l'entier 0, et `S` est la fonction successeur, qui permet de construire l'entier $n + 1$ à partir de l'entier n.

Dans Coq, il est possible de définir récursivement une fonction `nat_equal` de type `nat→nat→bool`, qui retourne la valeur booléenne `true` si ses deux arguments sont des valeurs identiques de type `nat`, et `false` s'ils sont distincts.

```
Fixpoint nat_equal (m n:nat) {struct m} : bool :=
  match m,n with
  | O,O => true
  | S m',S n' => nat_equal m' n'
  | _,_ => false
  end.
```

Ensuite on peut construire par une séquence de tactiques une preuve du théorème suivant :

```
Theorem nat_equal_semantics :
  ∀ m n, if nat_equal m n then m = n else m <> n.
```

Celui-ci établit un *principe de réflexion* qui relie le résultat booléen de la fonction `nat_equal` et l'égalité des deux arguments de cette fonction. Toutefois, cette correspondance nécessite l'évaluation totale de l'expression `nat_equal n m` vers une valeur booléenne `true` ou `false` pour que l'application du théorème puisse avoir, selon le cas, le type `m = n` ou `m <> n`. Illustrons cela par quelques exemples :

- $m = n = 1$:
  Le type de `nat_equal_semantics 1 1` est :

  ```
  if nat_equal 1 1 then 1 = 1 else 1 <> 1
  ```

  Or `nat_equal 1 1` se réduit en `nat_equal 0 0` puis en `true`. On en déduit que le type de `nat_equal_semantics 1 1` est convertible en `if true then 1 = 1 else 1 <> 1` et donc en `1 = 1` par ι-réduction, ce qui signifie que `nat_equal_semantics 1 1` est une preuve de `1 = 1`.
- $m = 1$ et $n = 2$:
  Le type de `nat_equal_semantics 1 2` est :

  ```
  if nat_equal 1 2 then 1 = 2 else 1 <> 2
  ```

  Après calcul de `nat_equal 1 2`, ce type devient : `if false then 1 = 2 else 1 <> 2` puis, après une ι-réduction (réduction du filtrage), il devient `1 <> 2` donc `nat_equal_semantics 1 2` est une preuve de `1 <> 2`.
- $m = n =$ `x` (où `x` est une variable de type `nat`) :
  Le type de `nat_equal_semantics x x` est :

  ```
  if nat_equal x x then x = x else x <> x
  ```

Comme ce type est en forme normale, on ne peut rien en faire directement. Toutefois, il est possible de prouver par récurrence que

$$\forall \texttt{n, nat\_equal n n = true}$$

puis d'utiliser ce résultat pour réécrire `nat_equal x x` en `true`, ce qui permet de transformer le type de `nat_equal_semantics x x` en `x = x`.

– $m = 0$ et $n =$ `S x` (où `x` est une variable de type `nat`) :
Notre implantation de `nat_equal` est telle que `nat_equal 0 (S x)` se réduit en `false`, on peut donc par exemple construire une preuve de `∀x, 0 <> S x` à l'aide du terme

$$\lambda \texttt{x.(nat\_equal\_semantics 0 (S x))}$$

De cet exemple on tire les conclusions suivantes :

– La réflexion calculatoire permet d'automatiser la construction de preuves de certaines propriétés.
– Les propriétés concernées sont toujours limitées à une classe restreinte.
– La technique de réflexion, en pratique, est limitée aux objets clos, comme en témoigne le troisième exemple ci-dessus : le quatrième exemple est marginal puisque l'on s'intéressera surtout à des cas où la fonction booléenne retourne `true` (propriété de correction).

4.1.3 Application à la preuve de propositions logiques

L'exemple précédent est de faible portée car la réflexion ne concerne que des égalités ou diségalités d'entiers. La puissance de la Théorie des Types apparaît lorsque l'on construit des types et des propositions suffisamment variés par filtrage et par récurrence sur des types de données : étant donné un type `form` de sorte Set qui nous sert de représentation concrète pour des formules logiques, et une fonction d'interprétation $[\![\_]\!]$: `form`→Prop, il est possible de définir une fonction `decide` : `form`→`bool` qui calcule si un objet de type `form` représente une tautologie, et de prouver dans le système Coq un *principe de réflexion* pour `decide` :

```
Theorem decide_correct :∀ F:form, decide F = true → [[F]].
```

Ceci nous donne une méthode utile pour prouver qu'une proposition A dans l'image de $[\![\_]\!]$ est valide. Pour commencer, il est nécessaire de trouver une réification de A, c'est-à-dire de trouver une représentation $\dot{A}$: `form` telle que $[\![\dot{A}]\!]$ soit convertible en A. Ensuite on construit le terme (`decide_correct` $\dot{A}$ (`refl_equal true`)), où `refl_equal` est le constructeur du prédicat d'égalité, tel que défini dans la section 1.3.1. Trois cas sont possibles :

– Si `decide` $\dot{A}$ se réduit en `true`, alors le terme (`decide_correct` $\dot{A}$) est de type `true = true` → $[\![\dot{A}]\!]$. Or (`refl_equal true`) est de type `true = true`, donc le terme (`decide_correct` $\dot{A}$ (`refl_equal true`)) est un terme de preuve pour $[\![\dot{A}]\!]$, et donc pour A par conversion.

- Si `decide` $\dot{A}$ se réduit en `false`, alors le terme (`decide_correct` $\dot{A}$) est de type `true` = `false` $\rightarrow [\![\dot{A}]\!]$. L'application de (`decide_correct` $\dot{A}$) à (`refl_equal true`) est alors mal typée. Ce terme n'est donc pas *a fortiori* un terme de preuve pour A.
- dans le cas intermédiaire où la forme normale de `decide` $\dot{A}$ est un terme `b` qui n'est ni `true` ni `false`, le type de (`decide_correct` $\dot{A}$) est alors `t` = `true` $\rightarrow [\![\dot{A}]\!]$ et on se retrouve dans le cas précédent : le terme (`decide_correct` $\dot{A}$ (`refl_equal true`)) est mal typé.

L'avantage de cette approche est que la vérification du terme de preuve repose, non pas sur la vérification du type d'un terme de preuve inerte (c'est-à-dire presque en forme normale) et souvent de grande taille, mais sur le calcul (potentiellement long) de la forme normale d'un terme plus petit. En outre, une implantation efficace de la fonction `decide` peut permettre de diminuer les besoins en mémoire et en temps de calcul nécessaires pour vérifier la preuve.

La distinction entre les propriétés logiques (dans Prop) et leur représentation (dans `form`) est indispensable puisqu'il n'y a pas moyen de calculer quoi que ce soit par filtrage et récurrence sur une propriété logique. En particulier, il est impossible de construire une fonction de type Prop→Prop→`bool` qui renvoie `true` si, et seulement si, ses deux arguments sont syntaxiquement égaux. En revanche, il est possible, si le type `form` est bien choisi, de calculer si deux objets F et G de type `form` sont égaux, bien que cette condition soit suffisante, mais non nécessaire, pour que $[\![F]\!]$ et $[\![G]\!]$ soient égaux.

En dépit de l'intérêt que peut susciter cette approche, plusieurs points sont à surveiller afin d'en préserver l'utilité :

1. La classe des propriétés potentiellement concernées est limitée à l'image de la fonction $[\![\_]\!]$: il faudra donc veiller à avoir une représentation suffisamment générale pour les propriétés.
2. Dans l'image de $[\![\_]\!]$, seules sont démontrables à l'aide de `decide_correct` les propriétés ayant une représentation F : `form` telle que (`decide` F) = `true`. On a donc intérêt à avoir une fonction `decide` renvoyant `true` pour un ensemble de valeurs le plus étendu possible.
3. Il faut pouvoir prouver le théorème de réflexion `decide_correct`, ce qui est en opposition avec le point 2 : la fonction ayant la valeur constante `false` a un théorème de réflexion très facile à prouver mais présente un intérêt limité. À l'opposé, la fonction ayant la valeur constante `true` donne un théorème de réflexion incohérent, à moins que l'image de $[\![\_]\!]$ ne contienne que des tautologies, auquel cas le point 1 est rarement respecté.
4. Enfin tout en maintenant l'équilibre entre les contraintes 2 et 3, il faut garder un coût raisonnable pour le calcul de `decide` F.

Si l'on souhaite s'affranchir du terme (`refl_equal true`) dans la preuve de $[\![\dot{A}]\!]$, on peut prouver une version équivalente du théorème de correction :

$$\texttt{decide\_correct} : \forall F : \texttt{form}, \texttt{if decide } F \texttt{ then } [\![F]\!] \texttt{ else True}$$

Enfin, dans le cas de propriétés décidables, on peut être en mesure de prouver un théorème donnant la correction et la complétude de `decide` (voir par exemple 4.1.2) :

$$\texttt{decide\_correct\_complet} : \forall F : \texttt{form}, \texttt{if decide}\ F\ \texttt{then}\ [\![F]\!]\ \texttt{else}\ \neg [\![F]\!]$$

4.1.4 Réflexion des preuves

Comme la décision de la validité en logique intuitionniste propositionnelle est PSPACE-complète [Sta79], il paraît difficile d'écrire en Coq une fonction de décision réaliste du point de vue du temps de calcul, et ce d'autant plus que le critère de terminaison de ce genre de fonction est souvent complexe. De plus, il serait sans doute difficile de prouver qu'une telle fonction est correcte directement.

Pour pallier cette difficulté, nous avons choisi d'ajouter au schéma de réflexion un objet concret de type `proof` : Set représentant la preuve de la proposition considérée. La fonction `decide` devient donc une fonction `check` qui vérifie si l'objet en question est la représentation d'une preuve de cette proposition. Le théorème de correction prend ainsi la forme suivante :

$$\texttt{check\_correct} : \forall F : \texttt{form}.\ \forall \pi : \texttt{proof}.((\texttt{check}\ F\ \pi) = \texttt{true}) \rightarrow [\![F]\!]$$

Le déroulement du calcul d'une trace de preuve est décrit dans la figure 4.1. Cette figure montre comment une procédure de recherche de preuve (aussi bien interne à Coq qu'externe) peut être utilisée par l'interface de Coq pour trouver une trace de preuve de la formule logique donnée en entrée. L'étape de réification se déroule dans le système Coq à l'aide de fonctions Caml et non à l'aide de fonctions de la Théorie des Types.

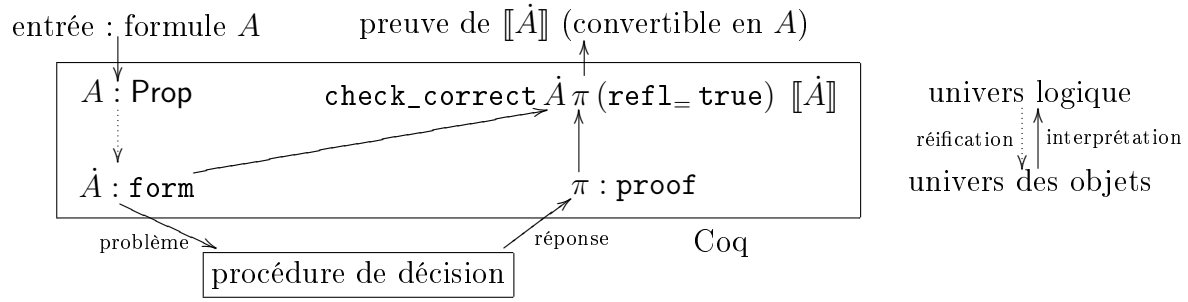


Fig. 4.1 – Schéma de réflexion avec traces de preuves

Si l'on souhaite retrouver une fonction `decide` comme à la section précédente, on pourra désormais écrire une fonction `search` : `form` →`proof` qui essaie de retourner une trace de preuve pour la formule F. La fonction `decide` pourra alors s'écrire :

$$\texttt{decide} := (\lambda F.\texttt{check}\ F\ (\texttt{search}\ F))$$

Son théorème de correction aura pour preuve $\lambda F.$ `check_correct` F (`search` F). Ainsi la correction de cette méthode ne repose que sur la correction de `check` et l'utilisateur peut programmer `search` comme il le souhaite, il devra simplement s'assurer de sa terminaison.

Les traces de preuve peuvent revêtir des formes très diverses, aussi faut-il tenir compte des contraintes qui s'imposent à nous pour ce choix :

1. Tout d'abord, il faut que ces traces soient naturelles pour qu'un outil externe puisse les engendrer sans une épaisse couche de traduction.
2. Mais il faut aussi qu'elles soient plus courtes que ne le serait un terme de preuve classique.
3. Enfin, elles doivent être relativement explicites pour que la fonction de vérification soit peu coûteuse et que sa preuve de correction soit simple. La question cruciale est donc bien la quantité d'informations implicites que la fonction `check` devra retrouver à partir de la trace au moment de la vérification de la preuve par le noyau de Coq : s'il y a trop de données à recalculer, le théorème de correction peut devenir difficile à prouver (par exemple s'il faut recalculer des unificateurs).

La solution que nous avons retenue ici est l'utilisation d'arbres de dérivation dans un calcul de séquents. Les hypothèses utilisées par les règles de déduction sont désignées par un indice : leur forme exacte est contenue dans le séquent à prouver. La description détaillée de cette technique fait l'objet du reste de ce chapitre.

4.2 Réflexion en logique propositionnelle

4.2.1 Structures de données utilisées

Les premières tentatives pour représenter des collections d'objets indexées par des listes dépendantes représentées par des entiers unaires (type `nat`) comme l'ont fait Marc Bezem, Dimitri Hendriks et Hans de Nivelle [BHdN02] se sont révélées inefficaces, gaspillant les ressources en temps et espace. Pour résoudre ce problème, nous avons donc opté pour des arbres binaires, indexés par des entiers binaires strictement positifs. Ce type d'arbre est appelé arbre de préfixes ou *trie* ([dlB59]). Bien que les arbres de Patricia [Mor68] soient plus efficaces en général, nous allons ici travailler avec des arbres quasi-complets, ce qui rend inutiles et coûteuses les optimisations des arbres de Patricia.

Les entiers binaires sont représentés par le type suivant :

```
Inductive positive : Set :=
  | xI : positive → positive
  | xO : positive → positive
  | xH : positive.
```

`xH` représente le bit de poids fort du nombre représenté tandis que les constructeurs `x0` et `xI` représentent l'ajout de 0 et 1 en position de poids faible. Par exemple, le nombre 6 est représenté par la valeur `(x0 (xI xH))`. Pour la suite, nous avons utilisé le mécanisme de section et de décharge de variables de section pour définir tout un ensemble d'objets paramétrés par un type `A` : Type.

```
Variable A:Type.
```

Nous avons alors défini des arbres binaires de la façon suivante :

```
Inductive Tree : Type :=
 | Tempty : Tree A
 | Branch0 : Tree A → Tree A → Tree A
 | Branch1 : A → Tree A → Tree A → Tree A.
```

Le constructeur `Tempty` construit un arbre réduit à une feuille tandis que les constructeurs `Branch0` et `Branch1` en sont des noeuds respectivement vide et contenant un élément de type `A` : Type. Pour définir la recherche d'un élément dans un arbre, il est nécessaire de pouvoir donner une valeur à la recherche d'un élément dans un noeud vide ou inexistant. Pour cela, nous définissons un nouveau type inductif :

```
Inductive Poption : Type:=
| PSome : A → Poption A
| PNone : Poption A.
```

Ce type de définition inductive est la solution la plus naturelle pour représenter le résultat de fonctions partielles, cela correspond à l'interprétation monadique de la partialité dans les langages fonctionnels. Le fait que ces types inductifs soient définis dans la sorte Type est obligatoire pour que ces types puissent contenir un objet de type `A` : Type. En particulier, nous allons les utiliser pour contenir des propositions (de type Prop : Type), mais aussi des objets informatifs dont le type est dans Set, grâce à la règle de cumulativité : tout type de sorte Set est aussi un type de sorte Type.

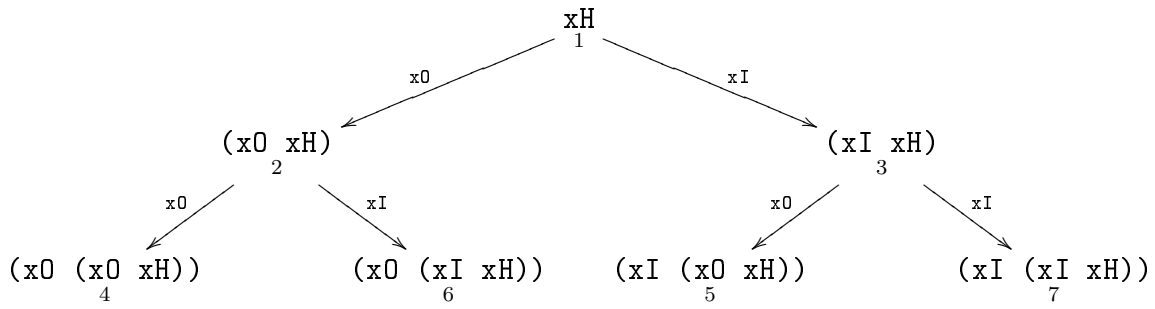


Fig. 4.2 – Indexation des noeuds dans les arbres binaires

L'indexation des noeuds (voir Figure 4.2) d'un arbre par des entiers de type `positive` se fait par la convention suivante : `xH` désigne la racine de l'arbre tandis que `xO n` et `xI n` désignent le noeud indexé par `n` respectivement dans le fils gauche et dans le fils droit de l'arbre[1]. La fonction d'accès à l'élément d'un noeud donné, et celle d'ajout d'un élément

[1]L'ordre des bits utilisé *(little-endian)* fait que la numérotation n'est pas strictement de gauche à droite.

dans un arbre sont ainsi définies par récurrence de la manière suivante :

```
Fixpoint Tget (p:positive) (T:Tree A) {struct p} : Poption :=
  match T with
  | Tempty => PNone
  | Branch0 T1 T2 =>
      match p with
      | xI pp => Tget pp T2
      | xO pp => Tget pp T1
      | xH => PNone
      end
  | Branch1 a T1 T2 =>
      match p with
      | xI pp => Tget pp T2
      | xO pp => Tget pp T1
      | xH => PSome a
      end
  end.

Fixpoint Tadd (p:positive) (a:A) (T:Tree A) {struct p}: Tree :=
  match T with
  | Tempty =>
      match p with
      | xI pp => Branch0 Tempty (Tadd pp a Tempty)
      | xO pp => Branch0 (Tadd pp a Tempty) Tempty
      | xH => Branch1 a Tempty Tempty
      end
  | Branch0 T1 T2 =>
      match p with
      | xI pp => Branch0 T1 (Tadd pp a T2)
      | xO pp => Branch0 (Tadd pp a T1) T2
      | xH => Branch1 a T1 T2 (* l'élément a est inséré *)
      end
  | Branch1 b T1 T2 =>
      match p with
      | xI pp => Branch1 b T1 (Tadd pp a T2)
      | xO pp => Branch1 b (Tadd pp a T1) T2
      | xH => Branch1 a T1 T2 (* l'élément b est remplacé *)
      end
  end.
```

Le nombre d'appels récursifs à `Tadd` et `Tget` est inférieur à la longueur de l'argument `p`, c'est-à-dire à son logarithme en base 2, et non à la taille de l'arbre. Nous allons donc, grâce

à une insertion séquentielle, obtenir une structure dans laquelle l'ajout et la recherche d'un élément se feront avec une durée logarithmique par rapport au nombre d'objets qu'elle contient. Afin de faciliter cette utilisation, nous définissons le type `Store` comme une paire constituée d'un arbre binaire et de la première adresse libre dans cet arbre :

```
Record Store : Type :=
        mkStore {index:positive;contents:Tree A}.

Definition empty := mkStore xH Tempty.
```

Les fonctions de recherche et d'addition sont définies grâce aux projections et à la fonction `Psucc` qui calcule le successeur $n+1$ d'un entier binaire n.

```
Definition get i S := Tget i (contents S).

Definition push a S :=
  mkStore (Psucc (index S)) (Tadd (index S) a (contents S)).
```

Par construction, tout objet de type `Store` construit uniquement à l'aide de `empty` et de `push` contient un arbre complet, c'est-à-dire que tous les nombres inférieurs à son index pointent sur des noeuds pleins, et que l'index pointe sur le premier emplacement libre. Malheureusement, il est possible de construire des objets de type `Store` mal formés en utilisant `mkStore` avec un arbre et un entier qui ne se correspondent pas : par exemple `mkStore (x0 xH) (Branch0 Tempty (Branch1 x Tempty Tempty))` est mal formé.

Pour pouvoir raisonner sur les objets de type `Store`, nous avons défini un prédicat `Full` exprimant le fait d'avoir été construit exclusivement au moyen de `empty` et de `push` :

```
Inductive Full : Store → Type:=
    F_empty : Full empty
  | F_push : ∀ a S, Full S → Full (push a S).
```

Ce prédicat est également utilisable comme une liste chaînée contenant les éléments de notre arbre dans l'ordre inverse de leur insertion. Il permet un accès séquentiel impossible à obtenir structurellement avec les arbres puisque les adresses des éléments sont lues par le bit de poids faible (codage *little endian*), ce qui mélange les adresses.

Cette combinaison arbre-liste chaînée est toutefois peu performante car elle occupe une place mémoire proportionnelle à $n \log_2(n)$, où n est le nombre d'éléments dans le `Store` : chaque ajout dans le `Store` crée $\log_2(n)$ nouveaux noeuds d'arbres, le reste étant partagé avec l'ancien arbre. Nous utiliserons donc cet objet dans les preuves pour spécifier des invariants, et dans quelques fonctions que l'on utilise peu.

Dans la suite, nous noterons `S`; `x` le terme (`push S x`), ou encore `S`; $i \mapsto$ `x` lorsque nous souhaiterons expliciter l'indice pointant sur `x`. Nous noterons parfois $\mathtt{S}_i$ l'élément pointé par l'index i dans `S`, c'est-à-dire tel que `get` i `S` = `PSome` $\mathtt{S}_i$.

4.2.2 Représentation et interprétation des formules propositionnelles

Pour représenter les formules propositionnelles, nous avons défini un type inductif `form`, dans lequel les formules atomiques sont représentées par des entiers de type `positive` :

DÉFINITION 4.1 (FORMULE RÉIFIÉE)
Les formules sont définies récursivement par :

$$\begin{aligned} \texttt{form} := \;& \texttt{Atom positive} \\ \mid\; & \dot{\bot} \\ \mid\; & \texttt{form} \dot{\rightarrow} \texttt{form} \\ \mid\; & \texttt{form} \dot{\wedge} \texttt{form} \\ \mid\; & \texttt{form} \dot{\vee} \texttt{form} \end{aligned}$$

Dans la suite du document, nous noterons ces connecteurs réifiés en les surmontant d'un point ($\dot{\rightarrow}, \dot{\bot}, \dot{\wedge}, \dot{\vee}$) pour les distinguer des connecteurs logiques ($\rightarrow, \bot, \wedge, \vee$).

Pour interpréter ces objets de type `form`, nous utilisons d'abord un environnement de type `(Store Prop)` qui associe une proposition logique à chaque numéro de formule atomique.

```
Variable env:Store Prop.
```

Les formules atomiques dont le numéro n'est pas dans l'environnement sont interprétées par la proposition triviale `True`. L'interprétation des formules se fait par la fonction suivante :

```
Fixpoint interp_form (f:form): Prop :=
  match f with
    Atom n =>
      match get n env with
        PNone => True
      | PSome P => P
      end
  | A ⇀̇ B => (interp_form A) → (interp_form B)
  | ⊥̇ => ⊥
  | A ∧̇ B => (interp_form A) ∧ (interp_form B)
  | A ∨̇ B => (interp_form A) ∨ (interp_form B)
  end.
```

Dans la suite de ce chapitre, l'interprétation réflexive d'un objet x sera dénotée $[\![x]\!]$ et sera faite implicitement dans l'environnement `env`. Lorsqu'une proposition F : Prop est représentable par une formule de type `form`, on notera $\dot{F}$ et on appellera réification de F une telle formule.

Comme nous faisons la réflexion d'un calcul de séquents à la Gentzen [Gen34], il faut représenter la partie gauche du séquent qui est constituée d'une liste de formules appelée contexte. Nous allons utiliser le type `ctx` = `Store form` pour représenter ce contexte. L'interprétation d'un contexte $F_1, \ldots, F_n$ avec le but G est définie comme $[\![F_1]\!] \to \ldots \to [\![F_n]\!] \to G$ à l'aide de la fonction `interp_ctx` ci-après. On notera ici l'utilisation de l'objet `F : Full` Γ comme une liste des objets du contexte.

```
Fixpoint interp_ctx
  (Γ:ctx) (F:Full Γ) (G:Prop) {struct F} : Prop :=
  match F with
    F_empty          => G
  | F_push H Γ0 F0 => interp_ctx Γ0 F0 ([[H]]→G)
  end.
```

Une fois définis les objets logiques et leur interprétation, passons maintenant aux preuves. Plusieurs choix sont possibles pour le calcul des séquents que nous allons utiliser : nous avons choisi le calcul LJT (cf. 3.3) car nous avions des procédures de décision basées sur ce calcul. Les preuves sont représentées par des arbres étiquetés par le nom des règles d'inférence et éventuellement par l'index de l'hypothèse ou des hypothèses utilisées, ce qui donne, dans le cas de LJT :

Définition 4.2 (trace de preuve)
Les traces de preuve sont définies récursivement comme suit :

```
proof :=  Ax    positive
        | Cut   form proof proof
        | →I    proof
        | →E    positive positive proof
        | →D    positive proof proof
        | ⊥E    positive proof proof
        | ∧I    proof proof
        | ∧E    positive proof
        | ∧D    positive proof
        | ∨I1   proof
        | ∨I2   proof
        | ∨E    positive proof proof
        | ∨D    positive proof
```

Dans le type `proof`, les arguments de type `proof` sont les traces de preuve des prémisses tandis que les arguments de type `positive` sont les indices des formules principales des règles utilisées. Dans la figure 4.3, nous définissons la sémantique des traces de preuve. Le jugement formel $\pi : \Gamma \vdash G$ y signifie : *« π est une trace de preuve correcte pour le séquent $\Gamma \vdash G$ »*.

$$\frac{}{(\text{Ax } i) : \Gamma \vdash G}\ \Gamma_i = G \qquad \frac{\pi_1 : \Gamma \vdash A \quad \pi_2 : \Gamma; A \vdash G}{(\text{Cut } A\ \pi_1\ \pi_2) : \Gamma \vdash G}$$

$$\frac{}{(\bot_E\ i) : \Gamma \vdash G}\ \Gamma_i = \dot{\bot}$$

$$\frac{\pi : \Gamma; A \vdash B}{(\rightarrow_I \pi) : \Gamma \vdash A \dot{\rightarrow} B} \qquad \frac{\pi : \Gamma; B \vdash G}{(\rightarrow_E i\ j\ \pi) : \Gamma \vdash G}\ \begin{cases} \Gamma_i = A \dot{\rightarrow} B \\ \Gamma_j = A \end{cases}$$

$$\frac{\pi_1 : \Gamma; A; B \dot{\rightarrow} C \vdash B \quad \pi_2 : \Gamma; C \vdash G}{(\rightarrow_D i\ \pi_1\ \pi_2) : \Gamma \vdash G}\ \Gamma_i = (A \dot{\rightarrow} B) \dot{\rightarrow} C$$

$$\frac{\pi_1 : \Gamma \vdash A \quad \pi_2 : \Gamma \vdash B}{(\wedge_I\ \pi_1\ \pi_2) : \Gamma \vdash A \dot{\wedge} B} \qquad \frac{\pi : \Gamma; A; B \vdash G}{(\wedge_E\ i\ \pi) : \Gamma \vdash G}\ \Gamma_i = A \dot{\wedge} B$$

$$\frac{\pi : \Gamma; A \dot{\rightarrow} B \dot{\rightarrow} C \vdash G}{(\wedge_D\ i\ \pi) : \Gamma \vdash G}\ \Gamma_i = (A \dot{\wedge} B) \dot{\rightarrow} C$$

$$\frac{\pi : \Gamma \vdash A}{(\vee_{I_1}\ \pi) : \Gamma \vdash A \dot{\vee} B} \qquad \frac{\pi : \Gamma \vdash B}{(\vee_{I_2}\ \pi) : \Gamma \vdash A \dot{\vee} B}$$

$$\frac{\pi_1 : \Gamma; A \vdash G \quad \pi_2 : \Gamma; B \vdash G}{(\vee_E\ i\ \pi_1\ \pi_2) : \Gamma \vdash G}\ \Gamma_i = A \dot{\vee} B$$

$$\frac{\pi : \Gamma; A \dot{\rightarrow} C; B \dot{\rightarrow} C \vdash G}{(\vee_D\ i\ \pi) : \Gamma \vdash G}\ \Gamma_i = (A \dot{\vee} B) \dot{\rightarrow} C$$

FIG. 4.3 – Sémantique des traces de preuve

EXEMPLE 4.3 (COMMUTATIVITÉ DE $\dot{\wedge}$)
Pour illustrer l'utilisation des traces de preuve définies plus haut, voici la preuve de $A \wedge B \vdash B \wedge A$ *:*

$$\frac{\dfrac{\overline{(\text{Ax } 3) : A \dot{\wedge} B, A, B \vdash B} \quad \overline{(\text{Ax } 2) : A \dot{\wedge} B, A, B \vdash A}}{(\wedge_I\ (\text{Ax } 3)\ (\text{Ax } 2)) : A \dot{\wedge} B, A, B \vdash B \dot{\wedge} A}}{(\wedge_E\ 1\ (\wedge_I\ (\text{Ax } 3)\ (\text{Ax } 2))) : A \dot{\wedge} B \vdash B \dot{\wedge} A}$$

Pour vérifier si une trace est une preuve valide d'un séquent, on va définir une fonction `check_proof` qui va faire cette vérification par récurrence sur la trace de preuve. Cette fonction prendra un argument supplémentaire par rapport à la fonction `check` citée précédemment : le contexte Γ du séquent.

Pour vérifier la correction de la règle d'axiome, il faut être capable de vérifier si deux formules sont syntaxiquement identiques (dans le cas des règles Ax et $\rightarrow_E$). Cela est réalisé par une fonction très simple `form_eq : form→form→bool` dont on prouve la correction :

```
Lemma form_eq_refl: ∀ p q, form_eq p q = true → p = q.
```

Grâce à cette fonction, il est devenu possible de vérifier la correction d'une trace de preuve à l'aide de la fonction `check_proof` suivante :

```
Fixpoint check_proof Γ G (P:proof) {struct P}: bool :=
match P with
  Ax i => match get i Γ with
    PSome F => form_eq F G
  | _ => false end
| →I => match G with
  (A ⇀ B) => check_proof (Γ; A) B p
  | _ => false end
| →E i j p => match get i Γ,get j Γ with
    PSome A,PSome (B ⇀ C) =>
      form_eq A B && check_proof (Γ; C) G p
  | _,_ => false end
| →D i p1 p2 => match get i Γ with
    PSome ((A ⇀ B) ⇀ C) =>
     check_proof (Γ; B ⇀ C; A) B p1 &&
     check_proof (Γ; C) G p2
    | _ => false end
| ⊥E i => match get i Γ with
    PSome ⊥̇ => true
  | _ => false end
| ∧I p1 p2 => match G with
    (A ∧̇ B) =>
    check_proof Γ A p1 && check_proof Γ B p2
  | _ => false end
| ∧E i p => match get i Γ with
    PSome  (A ∧̇ B) => check_proof (Γ; A; B) G p
  | _ => false end
| ∧D i p => match get i Γ with
    PSome (A ∧̇ B ⇀ C) =>
      check_proof (Γ; A ⇀ B ⇀ C) G p
  | _ => false end
| ∨I1 p => match G with
    (A ∨̇ _) => check_proof Γ A p
  | _ => false end
| ∨I2 p => match G with
    (_ ∨̇ B) => check_proof Γ B p
  | _ => false end                                                    ··/..
```

```
| ∨_E i p1 p2 => match get i Γ with
    PSome (A ∨̇ B) =>
      check_proof (Γ; A) G p1 &&
      check_proof (Γ; B) G p2
  | _ => false end
| ∨_D i p => match get i Γ with
    PSome (A ∨̇ B →̇ C) =>
      check_proof (Γ; A →̇ C; B →̇ C) G p
  | _=> false end
| Cut A p1 p2 =>
    check_proof Γ A p1 && check_proof (Γ; A) G p2
end.
```

Preuve du théorème de réflexion

Le principe de réflexion que nous allons prouver établit la correction de `check_proof` vis-à-vis de l'interprétation des formules. La formulation que nous avons choisie pour le théorème de réflexion est celle écrite ci-dessous, qui affirme que toute formule ayant une trace de preuve avec un contexte vide s'interprète comme une tautologie :

```
Theorem Reflect: ∀ (G:form) (π:proof),
 if check_proof empty G π then ⟦G⟧ else True.
```

On remarquera que ce théorème est implicitement quantifié sur toutes les interprétations possibles pour les formules atomiques. Ceci met en évidence le fait que les tautologies propositionnelles sont valides quelle que soit l'interprétation des formules atomiques.

Pour prouver ce théorème, nous allons d'abord en prouver une généralisation sur tous les contextes car, les prémisses des règles de notre calcul ont parfois des contextes distincts de celui de la conclusion (notamment $\rightarrow_I$).

```
Theorem interp_proof:
  ∀ π Γ (F:Full Γ) G,
  check_proof Γ G π = true → interp_ctx Γ F ⟦G⟧.
```

La démonstration de ce théorème passe par des lemmes techniques de passage au contexte qui permettent de prouver le passage des prémisses à la conclusion :

```
Lemma compose_0 :
  ∀ Γ F (A:Prop),A→
  interp_ctx Γ F A.

Lemma compose_1 :                                              ··/..
```

```
  ∀ Γ F (A B:Prop),(A→B)→
  interp_ctx Γ F A →
  interp_ctx Γ F B.

Lemma compose₂ :
  ∀ Γ F (A B C:Prop),(A→B→C)→
  interp_ctx Γ F A → interp_ctx Γ F B →
  interp_ctx Γ F C.

Lemma compose₃ :
  ∀ Γ F (A B C D:Prop),(A→B→C→D)→
  interp_ctx Γ F A → interp_ctx Γ F B → interp_ctx Γ F C →
  interp_ctx Γ F D.
```

Le choix de s'arrêter à $\texttt{compose}_3$ est ici purement pragmatique : si nous avions des règles utilisant plus d'hypothèses ou ayant plus de prémisses, il serait nécessaire d'utiliser des lemmes $\texttt{compose}_n$ pour $n > 3$.

Ces lemmes sont accompagnés d'un autre qui permet d'utiliser les hypothèses du contexte :

```
Theorem project : ∀ Γ F i G,
  get i Γ = PSome G → interp_ctx Γ F ⟦G⟧.
```

Pour illustrer l'utilisation de ces lemmes, prenons l'exemple de la règle $\rightarrow_D$: après élimination des cas absurdes, le sous-but correspondant à cette règle dans la preuve de `interp_proof` est le suivant :

```
p p₀ : positive
π₁ : proof
Γ : ctx
F : Full Γ
gl f f₁ f₂ : form
IH_π₁ : ∀ (Γ : ctx) (F : Full Γ) (gl : form),
      check_proof Γ gl π₁ = true → interp_ctx Γ F ⟦gl⟧
ef₀ : get p₀ Γ = PSome (f₁ ⇢ f₂)
ef : get p Γ = PSome f
e : form_eq f f₁ = true
check_π₁ : check_proof (Γ; f₂) gl π₁ = true
──────────────────────────────────────────────
interp_ctx Γ F ⟦gl⟧
```

La première étape de raisonnement que nous allons ordonner à Coq est l'utilisation de l'hypothèse de récurrence $\texttt{IH}_{\pi_1}$ à l'aide l'hypothèse $\texttt{check\_}\pi_1$, puis la suppression de ces hypothèses désormais inutiles :

```
generalize (IHπ1 (Γ; f2) (F_push f2 Γ F) gl checkπ1);
clear π1 checkπ1 IHp.
```

Ce qui donne le sous-but suivant :

```
p p0 : positive
Γ : ctx
F : Full Γ
gl f f1 f2 : form
ef0 : get p0 Γ = PSome (f1 ⇀̇ f2)
ef : get p Γ = PSome f
e : form_eq f f1 = true
______________________________________
interp_ctx (Γ; f2) (F_push f2 Γ F) gl →
interp_ctx Γ F ⟦gl⟧
```

Ensuite nous allons extraire du contexte les hypothèses d'indice `p` et $\texttt{p}_0$ à l'aide du lemme `project` :

```
generalize (project F ef) (project F ef0); clear p p0 ef ef0.
```

Nous obtenons alors le sous-but suivant :

```
Γ : ctx
F : Full Γ
gl f f1 f2 : form
e : form_eq f f1 = true
______________________________________
interp_ctx Γ F ⟦f⟧ →
interp_ctx Γ F ⟦f1 ⇀̇ f2⟧ →
interp_ctx (Γ; f2) (F_push f2 Γ F) gl →
interp_ctx Γ F ⟦gl⟧
```

Une étape de réduction par la tactique `simpl` permet de déplier les fonctions d'interprétation et de transformer la conclusion de notre but en :

```
interp_ctx Γ F ⟦f⟧ → interp_ctx Γ F (⟦f1⟧→⟦f2⟧) →
interp_ctx Γ F (⟦f2⟧→⟦gl⟧) → interp_ctx Γ F ⟦gl⟧
```

Nous pouvons alors appliquer le lemme $\texttt{compose}_3$ à l'aide de la commande `apply` $\texttt{compose}_3$ `; clear Γ F` qui donne le but suivant :

```
gl f f1 f2 : form
e : form_eq f f1 = true
______________________________________
⟦f⟧→(⟦f1⟧→⟦f2⟧)→(⟦f2⟧→⟦gl⟧)→⟦gl⟧
```

L'hypothèse `e` nous permet d'affirmer, à l'aide du théorème `form_eq_refl`, que $\texttt{f} = \texttt{f}_1$. Nous allons donc réécrire le but par la commande `rewrite (form_eq_refl e)` et obtenir une tautologie facile à prouver :

```
gl f f1 f2 : form
e : form_eq f f1 = true
______________________________________
⟦f1⟧→(⟦f1⟧→⟦f2⟧)→(⟦f2⟧→⟦gl⟧)→⟦gl⟧
```

Tous les sous-cas correspondant aux autres règles sont traités de façon similaire, c'est-à-dire par l'utilisation de l'hypothèse de récurrence, l'extraction du contexte des éventuelles hypothèses utilisées et l'application d'un lemme de composition afin qui'il ne reste plus à prouver que des tautologies propositionnelles. On peut ajouter, si on le souhaite, des règles supplémentaires pourvu qu'elles se ramènent à des tautologies propositionnelles par cette méthode.

4.2.3 La tactique `rtauto`

Afin de tester cette méthode, une tactique appelée `rtauto` a été implantée dans le système Coq. Cette tactique effectue successivement plusieurs tâches :

- décomposition du but et réification des hypothèses logiques
- procédure de décision par recherche en profondeur
- analyse de dépendances et simplification de la preuve (*pruning*)
- synthèse d'un terme de preuve utilisant le théorème de réflexion et la trace de preuve trouvée.

Le but Coq est analysé pour en extraire les hypothèses et la conclusion. Tout objet de type Prop qui ne commence pas par un connecteur logique $\wedge$, $\vee$, $\rightarrow$ et qui est différent de $\bot$ est considéré comme une formule atomique. La décomposition crée également une table globale des formules atomiques et de leur indice. Pour avoir un minimum de flexibilité, chaque sous-formule est mise en forme normale de tête faible avant d'être décomposée.

La méthode utilisée pour la recherche de preuve est classique : nous effectuons une recherche en profondeur sur les dérivations possibles à l'aide d'états persistants. Chaque état est une forme simplifiée de *zipper* [Hue97] dont les feuilles sont les séquents non prouvés, et dont les arêtes portent les étapes de raisonnement utilisées. Lorsqu'une branche est résolue, elle est transformée directement en arbre de preuve. La stratégie de preuve est similaire à celle décrite pour la tactique `firstorder`, avec en plus une pile des règles $\rightarrow_E$ non appliquées.

Dans l'arbre de dérivation, on note à chaque étape le nom des hypothèses engendrées, ce qui permet lorsque l'on clôt une branche de l'arbre, d'effectuer une analyse de dépendance sur les hypothèses et la conclusion, et d'effacer les étapes de raisonnement inutiles. Cette technique permet parfois déliminer des buts non résolus, lorsqu'une règle ayant plusieurs prémisses est effacée.

Pour engendrer le terme de preuve il faut construire trois sous-termes :
- l'environnement qui contient les formules atomiques
- la forme réifiée des hypothèses et de la conclusion
- la trace de preuve

Pour la trace de preuve, la partie la plus délicate à gérer est la numérotation des hypothèses engendrées.

La tactique `rtauto` est inclue dans la version de développement de Coq.

4.3 Réflexion au premier ordre

Comme les quantificateurs de Coq sont typés, notre représentation des formules du premier ordre sera paramétrée par une signature de domaines, afin que puissent être résolus des problèmes portant sur des objets dont les types sont inconnus au moment de la preuve du principe de réflexion. Cette technique permet la manipulation de problèmes multi-sortés et n'est donc pas limitée aux propositions dont les quantificateurs portent sur un domaine fixé *a priori* — c'est-à-dire codé « en dur » dans les fonctions d'interprétation — comme ce que propose Hendriks [Hen03]. Nos domaines sont représentés par des entiers de type `positive` qui pointent sur un arbre de type `Store Set`, paramètre de notre principe de réflexion. Les entiers non représentés dans cet arbre seront considérés comme pointant sur le type `unit`, qui est l'ensemble à un élément. Dans la suite, nous supposerons définis un environnement de domaines `Denv` et la fonction qui calcule le domaine correspondant à un indice.

```
Definition domain:=positive.

Definition interp_domain Denv d :=
  match get d Denv with
    PSome S => S
  | Pnone => unit
  end.
```

4.3.1 Représentation des termes et des formules

Nous avons choisi de représenter les variables liées par des indices de deBruijn [dB72] de type `nat` et les variables libres par des entiers de type `positive` qui représentent leur position dans un contexte de variables. Les symboles de fonctions et les prédicats sont aussi représentés par des entiers binaires qui pointent sur deux contextes situés dans une signature.

Pour pouvoir interpréter nos termes dans cette signature, il faut pouvoir y stocker les symboles : nous avons donc besoin d'encapsuler des objets de type différent dans des objets d'un même type afin de les stocker dans un `Store` ou une liste. La solution à ce problème est d'utiliser une paire dépendante contenant à la fois le type de l'objet représenté et l'objet lui-même : ainsi, chaque paire contient un élément de type différent mais toutes les paires ont le même type. Notre solution va donc ressembler à ceci :

```
Record entry: Type := mkE { type:Set; obj:type }.
```

Ce choix n'est pas acceptable pour autant car il serait alors impossible d'utiliser de tels objets en établissant une distinction selon leur type, ce que l'on veut précisément faire puisque l'on veut construire des termes applicatifs bien typés pour la théorie des types simples.

Par exemple, nous ne pouvons pas définir une fonction `coerce` ayant pour type $\forall$`(A : Set),entry` $\rightarrow$ `option A` telle que `coerce A (mkE A x)` se réduise en `Some x` pour tout couple (`A`, `x`) et en `None` sinon. Ce résultat est démontré en annexe B.

Pour remédier à ce problème, nous allons remplacer le type de l'objet, qui est non calculatoire, par une représentation de ce type utilisant les domaines de l'environnement `Denv`. Ainsi, pour représenter des variables dans un environnement, nous allons utiliser la paire dépendante suivante :

```
Record obj_entry: Set := mkOE
  { OE_domain : domain ;
    OE_it : interp_domain OE_domain }.
```

Maintenant, il devient possible d'écrire une fonction `f` ayant pour type $\forall$`(d : domain), obj_entry`$\rightarrow$`option (interp_domain d)` telle que `f dom (mkOE d x)` se réduise en `x` pour tout couple (`dom`, `x`), à la condition que x soit une valeur, c'est-à-dire que sa forme normale soit un terme clos :

```
Definition f expected pair :=
  let (dom,it) := pair in
  match pos_eq_dec dom expected with
    right _ => None
  | left e =>
    Some (eq_rec dom (interp_domain Denv) it expected e)
  end.
```

Pour bien montrer comment cela fonctionne, il nous faut d'abord définir les fonctions et types auxiliaires utilisés. Le type dont les constructeurs sont `left` et `right` est le type `sumbool` dont voici la définition :

```
Inductive sumbool (A : Prop) (B : Prop) : Set :=
    left : A → {A} + {B} | right : B → {A} + {B}
```

Ce type peut être vu soit comme le type des booléens annotés par des preuves de propriétés, soit comme la disjonction informative de ces propriétés : étant donné que cette disjonction est dans la sorte Set, nous pouvons construire notre fonction `f` par cas sur un objet de type `{m=n}+{m<>n}`. La fonction `pos_eq_dec` est de type `∀m n :positive,{m=n}+{m<>n}`, et a la propriété fondamentale suivante : pour tout entier binaire clos `n`, `pos_eq_dec n n` se réduit en `left (refl_equal n)`.

La fonction `f` commence ainsi par vérifier si l'indice du domaine `dom` de l'objet `it` est bien le même que celui du type attendu `expected`. Si c'est le cas, nous pouvons construire l'objet `Some it` mais il est du type `option (interp_domain dom)` et non du type `option (interp_domain expected)`, même si nous savons que ces types sont égaux.

Nous allons donc utiliser le principe de remplacement `eq_rec` qui est le principe d'élimination de l'égalité `eq` et agit comme une conversion de types garantie par une preuve d'égalité pour ces types (une version sûre de la fonction Caml `Obj.magic`) .

```
eq_rec : ∀(x : domain) (P : domain→Set), P x → ∀y, x=y → P y
```

Le terme `(eq_rec dom (interp_domain Denv) it expected e)` a donc pour type `option (interp_domain expected)`, et de plus, `eq_rec` a pour propriété que `(eq_rec d P x d (refl_equal d))` se réduit en `x` : or on sait que `e` se réduit en `(refl_equal n)`, d'où on obtient que le terme `Some (eq_rec dom (interp_domain Denv) it expected e)` est bien de type `option (interp_domain expected)` et se réduit bien en `Some it`, ce que l'on voulait.

Signature : fonctions et prédicats

Dans la suite de ce chapitre, nous supposerons fixé un environnement de domaines `Denv` et nous noterons $[\![\mathtt{d}]\!]_{\mathrm{Dom}}$ le type `interp_domain Denv d`. De plus, nous noterons {`x`∈`d`} la paire `(mkOE d x)`.

Pour les symboles de fonctions et les prédicats, il est nécessaire de construire des types du premier ordre à partir des domaines de l'environnement : les symboles de fonction ont des types de la forme $[\![\mathtt{d}_1]\!]_{\mathrm{Dom}} \to \ldots \to [\![\mathtt{d}_n]\!]_{\mathrm{Dom}} \to [\![\mathtt{d}]\!]_{\mathrm{Dom}}$ et les prédicats de la forme $[\![\mathtt{d}_1]\!]_{\mathrm{Dom}} \to \ldots \to [\![\mathtt{d}_n]\!]_{\mathrm{Dom}} \to \mathsf{Prop}$. Pour pouvoir construire ces types à partir de listes de types d'arguments, nous allons utiliser la fonction suivante :

```
Definition arity := list domain.

Fixpoint abstract (rge:arity) (hd:Type) {struct rge} : Type :=
  match rge with
    nil  => hd
  | r::q => ⟦r⟧_Dom → (abstract q hd)
end.
```

Celle-ci nous permet de définir des objets qui encapsulent fonctions et prédicats dans deux types de paire et triplet dépendants.

```
Record term_entry : Type := mkTE
  { TE_arity : arity;
    TE_domain : domain;
    TE_it : abstract TE_arity ⟦TE_domain⟧_Dom}.

Record pred_entry : Type := mkPE
  { PE_arity : arity;
    PE_it : abstract PE_arity Prop}.
```

Dans le type `term_entry`, le premier champ désigne la liste des types des arguments et le second le type du résultat, le troisième contient la fonction elle-même. Pour les prédicats, seul le type des arguments est nécessaire. La donnée d'une signature pour un problème du premier ordre sera alors la donnée de `Denv` et de deux environnements de type respectif `Store term_entry` et `Store pred_entry`.

Dans la suite, nous utiliserons les notations suivantes :

$$\{\mathtt{f} \in [\mathtt{d}_1;\ldots;\mathtt{d}_n]{\to}\mathtt{d}\} = \mathtt{mkTE}\ \ [\mathtt{d}_1;\ldots;\mathtt{d}_n]\ \ \mathtt{d}\ \ \mathtt{f}$$
$$\{\mathtt{P} \in [\mathtt{d}_1;\ldots;\mathtt{d}_n]{\to}*\} = \mathtt{mkPE}\ \ [\mathtt{d}_1;\ldots;\mathtt{d}_n]\quad\ \ \mathtt{P}$$

Exemple 4.4
Pour exprimer les propriétés d'entiers de type $\mathbb{N}$ *et de tableaux d'entiers de type (in* $\mathcal{A}_{\mathbb{N}}$*), nous utiliserons la signature de domaines* $\mathtt{Denv} = 1 \mapsto \mathbb{N}\,;\, 2 \mapsto \mathcal{A}_{\mathbb{N}}$.

Dans la signature, nous allons mettre la constante $\mathbf{0}$ *ainsi que les symboles de relation* $<$ *et* $\geq$ *sur les entiers. Nous mettrons également les fonctions* `select` *et* `store` *sur les tableaux et le prédicat* `sorted` *exprimant le fait que le tableau contient des entiers triés par ordre croissant. Nous aurons aussi besoin du prédicat* `valid` *exprimant le fait qu'un entier est un indice valide (dans les bornes) pour un tableau.*

Si nous faisons le point, les fonctions de notre signature sont :

$$\mathbf{0} : \mathbb{N}$$
$$\mathtt{select} : \mathcal{A}_{\mathbb{N}} {\to} \mathbb{N} {\to} \mathbb{N}$$
$$\mathtt{store} : \mathcal{A}_{\mathbb{N}} {\to} \mathbb{N} {\to} \mathbb{N} {\to} \mathcal{A}_{\mathbb{N}}$$

et les prédicats :

$$
\begin{aligned}
&< : \mathbb{N} \to \mathbb{N} \to \mathsf{Prop}\\
&\geq : \mathbb{N} \to \mathbb{N} \to \mathsf{Prop}\\
&\texttt{sorted} : \mathcal{A}_{\mathbb{N}} \to \mathsf{Prop}\\
&\texttt{valid} : \mathbb{N} \to \mathcal{A}_{\mathbb{N}} \to \mathsf{Prop}
\end{aligned}
$$

Dans notre encodage, les environnements correspondants seront :

$$
\begin{aligned}
\texttt{Tenv} = \quad & 1 \mapsto \{\mathbf{0} \in [] \to 1\}\\
& 2 \mapsto \{\texttt{select} \in [2;1] \to 1\}\\
& 3 \mapsto \{\texttt{store} \in [2;1;1] \to 2\}\\
\texttt{Penv} = \quad & 1 \mapsto \{< \in [1;1] \to *\}\\
& 2 \mapsto \{\geq \in [1;1] \to *\}\\
& 3 \mapsto \{\texttt{sorted} \in [2] \to *\}\\
& 4 \mapsto \{\texttt{valid} \in [1;2] \to *\}
\end{aligned}
$$

4.3.2 Termes et formules

DÉFINITION 4.5 (TERME, FORMULE)
Les termes et les formules sont des types récursifs définis par :

$$
\begin{aligned}
\texttt{term} := \quad & \texttt{Var positive}\\
\mid \quad & \texttt{DB nat}\\
\mid \quad & \texttt{App positive args}\\
\texttt{args} := \quad & \emptyset \mid \texttt{term}, \texttt{args}\\
\texttt{form} := \quad & \texttt{Atom positive args} \mid \dot{\forall}_{\texttt{domain}}\texttt{form} \mid \dot{\exists}_{\texttt{domain}}\texttt{form}\\
\mid \quad & \dot{\bot} \mid \texttt{form} \dot{\to} \texttt{form} \mid \texttt{form} \dot{\wedge} \texttt{form} \mid \texttt{form} \dot{\vee} \texttt{form}
\end{aligned}
$$

Dans les termes, le constructeur `Var` représente les variables libres et le constructeur `DB` représente les variables liées sous la forme d'indices de deBruijn[dB72] : les indices sont des entiers naturels unaires (du type $\texttt{nat} := \texttt{O} \mid \texttt{S nat}$), `O` représente ainsi la variable liée par le quantificateur le plus interne, $(\texttt{S O}) = 1$ la suivante, etc. Les constructeurs `App` et `Atom` représentent respectivement les termes composés à partir de l'application d'un symbole de fonction et les formules atomiques (application d'un prédicat à des termes). Leur premier argument est donc l'indice du symbole de fonction ou de prédicat dans la signature. Les indices des quantificateurs $\dot{\forall}$ et $\dot{\exists}$ représentent le domaine (selon `Denv`) de quantification. Les variables quantifiées ne sont pas représentées au niveau du quantificateur, c'est la notation de deBruijn qui prend en charge leur identification. La négation de la formule F peut être obtenue par la formule $F \dot{\to} \dot{\bot}$.

Un terme est dit clos si, et seulement si, il ne contient aucun constructeur `DB`, et une formule est dite close si tous ses constructeurs ont des indices inférieurs à leur profondeur de quantification.

Exemple 4.6
Dans cet exemple, nous allons utiliser la signature de l'exemple 4.4. Nous allons exprimer le fait que dans un tableau trié d'entiers, il existe un indice à partir duquel tous les entiers du tableau sont positifs :

$$\forall a : \mathcal{A}_{\mathbb{N}}, \texttt{sorted}(a) \rightarrow \exists i : \mathbb{N}, \forall j : \mathbb{N}, \texttt{valid}(j, a) \rightarrow \\ ((j < i) \rightarrow (\texttt{select}(a, j) < \mathbf{0})) \wedge ((j \geq i) \rightarrow (\texttt{select}(a, j) \geq \mathbf{0}))$$

Pour représenter cette proposition, nous utiliserons la représentation suivante :

$$\dot{\forall}_2(\texttt{Atom 3 (DB 0)} \dot{\rightarrow} \dot{\exists}_1 \dot{\forall}_1(\texttt{Atom 4 (DB 0, DB 2)}) \dot{\rightarrow} \\ (\texttt{Atom 1 (DB 0, DB 1)}) \dot{\rightarrow} (\texttt{Atom 1 (App 2 (DB 2, DB 0), App 1 } \emptyset)) \dot{\wedge} \\ (\texttt{Atom 2 (DB 0, DB 1)}) \dot{\rightarrow} (\texttt{Atom 2 (App 2 (DB 2, DB 0), App 1 } \emptyset))$$

Interprétation des termes et des formules

Dans cette partie, nous supposons fixée, outre l'environnement de domaines `Denv`, la signature composée d'un environnement de fonctions `Tenv : Store term_entry` et d'un environnement de prédicats `Penv : Store pred_entry`.

Nous souhaitons maintenant écrire une fonction `interp_term` dont le type sera :

$$\texttt{Store obj\_entry} \rightarrow \texttt{list obj\_entry} \rightarrow \texttt{term} \rightarrow \forall (d : \texttt{domain}), \texttt{option } [\![\texttt{d}]\!]_{\text{Dom}}$$

Les deux premiers arguments seront les environnements d'interprétation pour les variables libres et liées. Pour les termes réduits à une variable libre ou liée, nous allons procéder de la même façon que dans notre fonction `f` précédente, mais pour les termes formés par application, il faut définir une fonction `interp_args` mutuellement récursive avec `interp_term` pour interpréter les arguments des symboles de fonction. Le problème qui se pose alors est complexe : quel doit être le type des objets retournés par `interp_args` ?

La solution la plus simple serait une liste d'objets du type `option obj_entry`, mais le problème de cette représentation est qu'elle nous contraindrait à parcourir à nouveau cette liste pour y appliquer le symbole de tête. Plutôt que de faire cela, nous avons choisi de passer le symbole de tête en argument à la fonction (une sorte de passage par continuation), notre fonction `interp_args` aura alors le type :

$$\texttt{Store obj\_entry} \rightarrow \texttt{list obj\_entry} \rightarrow \texttt{args} \rightarrow \\ \forall (\texttt{rge} : \texttt{arity})(T : \mathsf{Type}), \texttt{abstract rge } T \rightarrow \texttt{Poption } T$$

Cette fonction est très simple puisque les coercitions de types auront toutes lieu dans `interp_term`, il s'agit simplemement d'un combinateur d'application n-aire :

```
Fixpoint interp_args ξ δ a rge T {struct rge}:
abstract Denv rge T → Poption T :=
match a,rge return abstract Denv rge T → Poption T with              ··/..
```

```
  ∅,nil => (λt. PSome t)
| (t;argv),dom::domv =>
  match interp_term ξ δ t dom with
    None => (λf. PNone)
  | Some arg => (λf. interp_args ξ δ argv domv T (f arg))
  end
| _ , _ => (λf. PNone)
end with interp_term ...
```

La bonne formation de cette fonction repose sur le fait que si `x` est de type $[\![\texttt{d}]\!]_{\text{Dom}}$ et `f` de type `abstract (d :: rge) T` — qui peut se convertir en $[\![\texttt{d}]\!]_{\text{Dom}} \to$`(abstract rge T)` — alors `(f x)` est de type `abstract rge T`. Ce dernier élément nous permet d'écrire notre fonction d'interprétation des termes.

```
... interp_term ξ δ t expected {struct t}:
  option [[expected]]_Dom :=
match t with
  App hd a =>
  match get hd Tenv with PNone => None
  | PSome {head ∈ rge → dom} =>
    match pos_eq_dec dom expected with right _ => None
    | left e =>
      match interp_args ξ δ a rge [[dom]]_Dom head with
        PNone => None
      | PSome it =>
        Some (eq_rec dom interp_domain it expected e)
      end
    end
  end
| DB i =>
  match Lget i δ with None => None
  | Some {it∈dom} =>
    match pos_eq_dec dom expected with right _ => None
    | left e => Some (eq_rec _ (interp_domain Denv) it _ e)
    end
  end
| Var n =>
  match get n ξ with PNone => None
  | PSome {it∈dom} =>
    match pos_eq_dec dom expected with right _ => None
    | left e => Some (eq_rec _ (interp_domain Denv) it _ e)
    end                                                    ··/..
```

```
  end
end
```

Dans la suite de ce document, nous utiliserons les notations suivantes :

$$[\![\mathtt{t}]\!]^{\mathtt{d}}_{\xi,\delta} = \mathtt{interp\_term}\ \xi\ \delta\ \mathtt{t}\ \mathtt{d}$$
$$[\![\Phi|\mathtt{a}]\!]^{\mathtt{l}}_{\xi,\delta} = \mathtt{interp\_args}\ \xi\ \delta\ \mathtt{a}\ \mathtt{l}\ \mathtt{T}\ \Phi$$

Interprétation des formules L'interprétation des formules se fait de façon similaire au cas propositionnel :

```
Fixpoint interp_form ξ δ (f:form) {struct f} : Prop :=
match f with
  Atom hd a =>
  match get hd Penv with PNone => True
  | PSome {head ∈ rge → *} =>
    match [[head|a]]^rge_{ξ,δ} with PNone => True
    | PSome P => P
    end
  end
| ⊥̇ => ⊥
| A →̇ B => (interp_form ξ δ A) → (interp_form ξ δ B)
| A ∧̇ B => (interp_form ξ δ A) ∧ (interp_form ξ δ B)
| A ∨̇ B => (interp_form ξ δ A) ∨ (interp_form ξ δ B)
| ∀̇_dom G => ∀ x:[[dom]]_Dom, interp_form ξ ({x∈dom}::δ) G
| ∃̇_dom G => ∃ x:[[dom]]_Dom, interp_form ξ ({x∈dom}::δ) G
end.
```

On notera que toutes les variables liées par des quantificateurs s'interprètent sur une variable nommée `x`. Le mécanisme d'évitement de la capture du système Coq va en fait ajouter un nombre comme suffixe pour garantir qu'il n'y ait pas de collision de noms. Ainsi la formule de l'exemple 4.6 s'interprètera sur la proposition Coq suivante, qui est bien α-équivalente à celle de l'exemple.

$$\forall x : \mathcal{A}_{\mathbb{N}}, \mathtt{sorted}(x) \to \exists x_0 : \mathbb{N}, \forall x_1 : \mathbb{N}, \mathtt{valid}(x_1, x) \to \\ ((x_1 < x_0) \to (\mathtt{select}(x, x_1) < \mathbf{0})) \wedge ((x_1 \geq x_0) \to (\mathtt{select}(x, x_1) \geq \mathbf{0}))$$

L'utilisation de listes pour l'environnement des variables liées est parfaitement adaptée pour la notation de deBruijn : à chaque fois que l'on interprète un quantificateur, la position des anciennes variables liées est décalée de 1 dans la liste de façon identique à l'indice pointant sur la variable en question dans la formule, et comme il est rare de rencontrer des formules dont la profondeur de quantificateurs est élevée, nous avons choisi de conserver cette structure malgré un temps d'accès linéaire.

Dans la suite, nous noterons $[\![\mathtt{G}]\!]_{\xi,\delta}$ le terme `(interp_form ξ δ G)`.

Contextes et séquents

DÉFINITION 4.7 (SÉQUENT)
Un séquent est un triplet noté $\Xi, \Gamma \vdash G$*, où* Ξ *est l'ensemble des domaines des variables libres du séquent, stockés dans un* `Store domain`*, leur position dans le* `Store` *correspond aux indices du constructeur* `Var`*.* Γ *est le contexte de type* `Store form` *contenant les hypothèses, et* G : `form` *est la conclusion du séquent.*

Il nous est nécessaire de maintenir explicitement l'ensemble des variables libres dans le séquent pour pouvoir l'interpréter par une proposition Coq où ces variables sont universellement quantifiées.

Il faut alors procéder en deux temps : d'abord nous définissons une fonction `interp_ctx` similiaire à celle utilisée dans le cas propositionnel qui nous permet de construire une série d'implications des hypothèses vers la conclusion du séquent :

```
Fixpoint interp_hyps Γ (F:Full Γ) (Ψ:var_env → Prop)
  {struct F} : var_env→Prop :=
match F with
  F_empty => Ψ
| F_push G Γ0 F0 =>
      interp_hyps Γ0 F0 (λξ. ⟦G⟧ξ,δ→ Ψ(ξ))
end.
```

Le résultat de cette fonction n'est pas directement une proposition mais une fonction attendant un environnement pour les variables globales et donnant la proposition correspondante.

Ensuite, nous définissons une fonction qui construit la clôture universelle de la proposition représentant le séquent par récurrence sur la liste des domaines des variables libres :

```
Fixpoint interp_vars Ξ (F:Full Ξ) (Ψ:var_env → Prop)
{struct F} : Prop :=
match F with
  F_empty => Ψ empty
| F_push dom Ξ0 F0 =>
      interp_vars Ξ0 F0 (λξ.∀x:⟦dom⟧Dom,Ψ(ξ;{x : dom}))
end.
```

L'ensemble du schéma d'interprétation est résumé dans la figure 4.4

Instanciation des quantificateurs

Pour formaliser le raisonnement au premier ordre avec égalité, il nous manque encore une opération élémentaire sur nos objets : l'instanciation d'une formule par un terme clos (`inst`).

Termes

$$[\![\texttt{DB}\ n]\!]^r_{\xi,\delta} = x \quad \text{si} \begin{cases} \delta_n = \{x \in r'\} \\ r = r' \end{cases}$$

$$[\![\texttt{Var}\ i]\!]^r_{\xi,\delta} = x \quad \text{si} \begin{cases} \xi_i = \{x \in r'\} \\ r = r' \end{cases}$$

$$[\![\texttt{App}\ i\ a]\!]^r_{\xi,\delta} = [\![f|\, a]\!]^d_{\xi,\delta} \quad \text{si} \begin{cases} \texttt{Tenv}_i = \{f \in d \to r'\} \\ r = r' \end{cases}$$

Arguments

$$[\![\Phi|\, \emptyset]\!]^{\emptyset}_{\xi,\delta} = \Phi$$

$$[\![\Phi|t :: a]\!]^{r::d}_{\xi,\delta} = \left[\!\!\left[\Phi([\![t]\!]^r_{\xi,\delta}) \,\middle|\, a \right]\!\!\right]^d_{\xi,\delta}$$

Formules

$$[\![\texttt{Atom}\ i\ a]\!]_{\xi,\delta} = [\![P|\, a]\!]^d_{\xi,\delta} \quad \text{si } \texttt{Penv}_i = \{P \in d \to *\}$$

$$\left[\!\!\left[\dot{\bot} \right]\!\!\right]_{\xi,\delta} = \bot$$

$$\left[\!\!\left[\dot{\forall}_d F \right]\!\!\right]_{\xi,\delta} = \forall x : [\![d]\!]_{\mathrm{Dom}}, [\![F]\!]_{\xi,\{x \in d\}::\delta}$$

$$\left[\!\!\left[\dot{\exists}_d F \right]\!\!\right]_{\xi,\delta} = \exists x : [\![d]\!]_{\mathrm{Dom}}, [\![F]\!]_{\xi,\{x \in d\}::\delta}$$

$$[\![F_1 \dot{\to} F_2]\!]_{\xi,\delta} = [\![F_1]\!]_{\xi,\delta} \to [\![F_2]\!]_{\xi,\delta}$$

$$[\![F_1 \dot{\wedge} F_2]\!]_{\xi,\delta} = [\![F_1]\!]_{\xi,\delta} \wedge [\![F_2]\!]_{\xi,\delta}$$

$$[\![F_1 \dot{\vee} F_2]\!]_{\xi,\delta} = [\![F_1]\!]_{\xi,\delta} \vee [\![F_2]\!]_{\xi,\delta}$$

Hypothèses

$$[\![\emptyset\,|\Psi]\!]_{\xi} = \Psi(\xi)$$

$$[\![\Gamma; F\,|\Psi]\!]_{\xi} = \left[\!\!\left[\Gamma \,\middle|\, \xi' \mapsto \left([\![F]\!]_{\xi',\emptyset} \to \Psi(\xi') \right) \right]\!\!\right]_{\xi}$$

Variables libres

$$[\![\emptyset\,|\Psi]\!] = \Psi(\emptyset)$$

$$[\![\Xi; d\,|\Psi]\!] = [\![\Xi\,|\xi \mapsto \forall x : [\![d]\!]_{\mathrm{Dom}}, \Psi(\xi; \{x \in d\})]\!]$$

Séquents

$$[\![\Xi, \Gamma \vdash G]\!] = \left[\!\!\left[\Xi \,\middle|\, \xi \mapsto \left[\!\!\left[\Gamma \,\middle|\, \xi' \mapsto [\![G]\!]_{\xi',\emptyset} \right]\!\!\right]_{\xi} \right]\!\!\right]$$

Fig. 4.4 – Description synthétique de l'interprétation des séquents

L'opération `inst` se ramène au remplacement de (`DB` n) par le terme à substituer, où n est le nombre de quantificateurs traversés. Il n'y a pas de décalage d'indices à effectuer dans `t` puisque ce terme est clos (il ne contient aucun indice de deBruijn). La fonction qui effectue cette transformation va donc prendre un argument supplémentaire de type `nat` que l'on utilisera pour savoir si un indice de deBruijn doit être remplacé. La fonction `nat_eq` utilisée est une fonction booléenne de comparaison d'entiers de type `nat`.

```
Fixpoint inst_term t n (s:term) {struct s} :term :=
match s with
  App f args => App f (inst_args t n args)
| DB i =>  if nat_eq i n then t else DB i
| Var v => Var v
end
with inst_args t n (a:args) {struct a} :args :=
match a with
  ∅ => ∅
| arg; more => (inst_term t n arg); (inst_args t n more)
end.

Fixpoint inst_form t n (f:form) {struct f} :form :=
 match f with
  Atom p args => Atom p (inst_args t n args)
| ⊥̇ => ⊥̇
| A →̇ B => (inst_form t n A) →̇ (inst_form t n B)
| A ∧̇ B => (inst_form t n A) ∧̇ (inst_form t n B)
| A ∨̇ B => (inst_form t n A) ∨̇ (inst_form t n B)
| ∀̇_dom G => ∀̇_dom (inst_form t (n+1) G)
| ∃̇_dom G => ∃̇_dom (inst_form t (n+1) G)
end.
```

4.3.3 Traces de preuve

Les traces de preuve que nous allons utiliser ici sont une simple extension de celles utilisées précédemment pour la logique propositionnelle.

Définition 4.8 (traces de preuve)
Les traces de preuve sont définies récursivement comme suit :

$$\begin{array}{rcl}
\texttt{proof} := & \ldots & (\textit{idem}\ \text{définition 4.2}) \\
| & \forall_I & \texttt{proof} \\
| & \forall_E & \texttt{positive term proof} \\
| & \forall_D & \texttt{positive proof proof} \\
| & \exists_I & \texttt{term proof} \\
| & \exists_E & \texttt{positive proof} \\
| & \exists_D & \texttt{positive proof}
\end{array}$$

La sémantique des traces de preuves propositionnelles s'étend naturellement au premier ordre en ajoutant le contexte de variables Ξ qui reste invariable par ces règles. La sémantique des règles ajoutées est décrite dans la figure 4.5. Nous avons suivi ici le calcul *G4i* mais il est possible d'utiliser des variantes ou d'ajouter des règles.

$$\frac{\pi : \Xi; j \mapsto d, \Gamma \vdash (\texttt{inst}\ A\ (\texttt{Var}\ j))}{(\forall_I\ \pi) : \Xi, \Gamma \vdash \dot\forall_d A} \qquad \frac{\pi : \Xi, \Gamma; (\texttt{inst}\ A\ t) \vdash G}{(\forall_E\ i\ t\ \pi) : \Xi, \Gamma \vdash G}\ \Gamma_i = \dot\forall_d A$$

$$\frac{\pi_1 : \Xi, \Gamma \vdash \dot\forall_d A \quad \pi_2 : \Xi, \Gamma; B \vdash G}{(\forall_D\ i\ \pi_1\ \pi_2) : \Xi, \Gamma \vdash G}\ \Gamma_i = (\dot\forall_d A) \dot\to B$$

$$\frac{\pi : \Xi, \Gamma \vdash (\texttt{inst}\ A\ t)}{(\exists_I\ t\ \pi) : \Xi, \Gamma \vdash \dot\exists_d A} \qquad \frac{\pi : \Xi; j \mapsto d, \Gamma; (\texttt{inst}\ A\ (\texttt{Var}\ j)) \vdash G}{(\exists_E\ i\ \pi) : \Gamma \vdash G}\ \Gamma_i = \dot\exists_d A$$

$$\frac{\pi : \Xi, \Gamma; \dot\forall_d (A \dot\to B) \vdash G}{(\exists_D\ i\ \pi) : \Xi, \Gamma \vdash G}\ \Gamma_i = (\dot\exists_d A) \dot\to B$$

Fig. 4.5 – Règles d'inférence pour les quantificateurs

Exemple 4.9
Pour illustrer le fonctionnement des règles pour les quantificateurs, nous allons prouver le théorème suivant, exprimé dans la signature de l'exemple 4.4 : « si tous les tableaux sont triés, alors il existe un tableau trié ».

Malheureusement, la validité de cette proposition dépend de l'existence d'un terme clos de type $\mathcal{A}_{\mathbb{N}}$, or la signature fournie ne le permet pas, c'est pourquoi nous allons ajouter un quantificateur universel en tête de formule qui nous donnera une variable dans le type $\mathcal{A}_{\mathbb{N}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{Ax}\ i : 1 \mapsto 2, \dot\forall_2 \dot{\texttt{sorted}}(\texttt{DB}\,0); \dot{\texttt{sorted}}(\texttt{Var}\,1) \vdash \dot{\texttt{sorted}}(\texttt{Var}\,1)}}{\forall_E\,1\,\texttt{Var}\,1\,(\text{Ax}\ i) : 1 \mapsto 2, \dot\forall_2 \dot{\texttt{sorted}}(\texttt{DB}\,0) \vdash \dot{\texttt{sorted}}(\texttt{Var}\,1)}}{\exists_I\,\texttt{Var}\,1\,(\forall_E\,1\,\texttt{Var}\,1\,(\text{Ax}\ i))) : 1 \mapsto 2, \dot\forall_2 \dot{\texttt{sorted}}(\texttt{DB}\,0) \vdash \dot\exists_2 \dot{\texttt{sorted}}(\texttt{DB}\,0)}}{\to_I\,(\exists_I\,\texttt{Var}\,1\,(\forall_E\,1\,\texttt{Var}\,1\,(\text{Ax}\ i)))) : 1 \mapsto 2, \emptyset \vdash (\dot\forall_2 \dot{\texttt{sorted}}(\texttt{DB}\,0)) \dot\to \dot\exists_2 \dot{\texttt{sorted}}(\texttt{DB}\,0)}}{\forall_I(\to_I(\exists_I\,\texttt{Var}\,1\,(\forall_E\,1\,\texttt{Var}\,1\,(\text{Ax}\ i))))) : \emptyset, \emptyset \vdash \dot\forall_2(\dot\forall_2 \dot{\texttt{sorted}}(\texttt{DB}\,0)) \dot\to \dot\exists_2 \dot{\texttt{sorted}}(\texttt{DB}\,0)}$$

Nous pouvons alors étendre notre fonction `check_proof` pour gérer les règles supplémentaires. Les règles propositionnelles sont traitées comme à la section 4.2, à l'exception de la règle de coupure Cut car on souhaite ici vérifier que la bonne formation des formules est préservée. Dans les règles $\forall_E$ et $\exists_I$, ainsi que pour la règle Cut, nous vérifions que les termes et les formules introduits sont bien formés, de sorte que toutes les règles préservent la bonne formation des séquents.

```
Fixpoint check_proof Ξ Γ G π {struct π}: bool :=
match π with
   ⋮
| ∀_I p =>
  match G with ∀̇_d F =>
    check_proof (Ξ; d) Γ (inst_form (Var (index Ξ)) 0 F) p
  | _ => false end
| ∀_E i t p =>
  match get i Γ with PSome (∀̇_d F) =>
    check_ground_term Denv Tenv Ξ t d &&
    check_proof Ξ (Γ; inst_form t 0 F) G p
  | _ => false end
| ∀_D i p_1 p_2 =>
  match get i Γ with PSome ((∀̇_d F) ⇀̇ C) =>
    check_proof Ξ Γ (∀̇_d F) p_1 &&
    check_proof Ξ (Γ; C) G p_2
  | _ => false end
| ∃_I t p =>
  match G with ∃̇_d F =>
    check_ground_term Denv Tenv Ξ t d &&
    check_proof Ξ Γ  (inst_form t 0 F) p
  | _ => false end
| ∃_E i p =>
  match get i Γ with PSome (∃̇_d F) =>
    check_proof (Ξ; d) (Γ; inst_form (Var (index Ξ)) 0 F) G p
  | _ => false end
| ∃_D i p =>
  match get i Γ with PSome ((∃̇_d F) ⇀̇ C) =>
    check_proof Ξ (Γ; (∀̇_d(F ⇀̇ C))) G p
  | _ => false end
| Cut A p_1 p_2 =>
  check_form Denv Tenv Penv Ξ nil A &&
  check_proof Ξ Γ A p_1 &&
  check_proof Ξ (Γ; A) G p_2.
end.
```

4.3.4 Preuve de correction

Bonne formation des formules

Contrairement au cas propositionnel, la validité des formules du premier ordre n'est pas préservée lorsque l'on change la signature. En voici un exemple simple, prenons la formule réifiée suivante :

$$F \equiv \dot{\forall}_1(\texttt{Atom}\,1\,(\texttt{App}\,1\,())\,(\texttt{DB}\,0))\dot{\rightarrow}\dot{\exists}_1\,(\texttt{Atom}\,1\,(\texttt{DB}\,0)\,(\texttt{DB}\,1))$$

Si l'on interprète cette formule avec des signatures de fonctions et de prédicats vides, comme aucune des sous-formules atomiques n'est bien formée, on obtient la formule tautologique suivante :

$$[\![F]\!] \equiv \forall x : \mathbb{N}\,.(\texttt{True}\rightarrow\exists x_0 : \mathbb{N}\,.\texttt{True})$$

Maintenant, si l'on garde une signature vide pour les fonctions mais que l'on reprend la signature de prédicats de l'exemple 4.4, on obtient la formule suivante, qui n'est plus tautologique :

$$[\![F]\!] \equiv \forall x : \mathbb{N}\,.\texttt{True}\rightarrow\exists y : \mathbb{N}\,.(y < x)$$

Enfin, si l'on interprète cette formule avec les deux signatures de l'exemple 4.4, alors on obtient une nouvelle tautologie :

$$[\![F]\!] \equiv \forall x : \mathbb{N}\,.(\mathbf{0} < x)\rightarrow\exists y : \mathbb{N}\,.(y < x)$$

Nous voyons donc que des modifications dans les signatures peuvent rendre l'interprétation des formules valides ou invalides, les deux cas sont possibles.

C'est pourquoi, pour prouver la correction de notre interprétation réflexive, il faudra s'assurer de la bonne formation des formules manipulées, et vérifier que cette propriété est conservée au cours du raisonnement et par l'opération d'instanciation. En particulier cela explique pourquoi, lorsque l'on introduit une formule étrangère (règle Cut) ou des termes étrangers (règles $\exists_I$ et $\forall_E$), on doit s'assurer que ceux-ci sont bien formés (voir l'encadré page 148).

Pour cela, on commence par définir pour chacun des objets définis plus haut (termes, arguments, formules, contextes et séquents) un prédicat `WF` de bonne formation, assorti d'une fonction booléenne `check` de décision de ce prédicat. Les termes sont bien formés par rapport à un domaine s'ils sont bien typés et du type de ce domaine. Les arguments sont bien formés par rapport à une liste de domaines s'ils sont formés d'une liste d'autant de termes bien formés et dont les types correspondent. Les formules sont bien formées si toutes les sous-formules atomiques le sont dans un contexte contenant les variables quantifiées. Les contextes d'hypothèses sont bien formés si toutes les hypothèses sont des formules bien formées et closes. Un séquent est bien formé si son contexte et sa conclusion le sont.

Pour illustrer notre propos, nous donnons ici les définitions utilisées pour les formules :

```
Inductive WF_form (Ξ:Store domain): (list domain)→form→Prop:=
  WF_Atom : ∀ Δ n args doms p,
    get n Penv = PSome {p∈doms→*} →
    WF_args Ξ Δ args doms → WF_form Ξ Δ (Atom n args)
| WF_Absurd : ∀ Δ,WF_form Ξ Δ ⊥̇
| WF_Arrow : ∀ Δ f₁ f₂,
    WF_form Ξ Δ f₁ → WF_form Ξ Δ f₂ → WF_form Ξ Δ (f₁ →̇ f₂)
| WF_And : ∀ Δ f₁ f₂,
    WF_form Ξ Δ f₁ → WF_form Ξ Δ f₂ → WF_form Ξ Δ (f₁ ∧̇ f₂)
| WF_Or : ∀ Δ f₁ f₂,
    WF_form Ξ Δ f₁ → WF_form Ξ Δ f₂ → WF_form Ξ Δ (f₁ ∨̇ f₂)
| WF_Forall : ∀ Δ dom f₀,
    WF_form Ξ (dom::Δ) f₀ → WF_form Ξ Δ (∀̇dom dom f₀)
| WF_Exists : ∀ Δ dom f₀,
    WF_form Ξ (dom::Δ) f₀ → WF_form Ξ Δ (∃̇dom dom f₀).
```

Vérifier la bonne formation des formules consiste principalement à s'assurer du bon typage des formules atomiques en s'appuyant sur le prédicat `WF_args`. Ce dernier vérifie que les types des termes sont bien ceux de la liste `doms`. Afin de pouvoir obtenir simplement une preuve de cette bonne formation, on définit une fonction booléenne `check_form` qui décide le prédicat `WF_form`.

```
Fixpoint check_form (Ξ:Store domain) (Δ:list domain) (f:form)
  {struct f}: bool :=
match f with
  Atom p a =>
  match get p Penv with PNone => false
  | PSome {_∈doms→*} => check_args Ξ Δ a doms
  end
| ⊥̇ => true
| f₁ →̇ f₂ | f₁ ∧̇ f₂ | f₁ ∨̇ f₂ =>
  check_form Ξ Δ f₁ && check_form Ξ Δ f₂
| ∀̇dom f₀ | ∃̇dom f₀ => check_form Ξ (dom::Δ) f₀
end.
```

On prouve ensuite pour chaque objet la correction de `check` par rapport à `WF`. Pour les formules, cela donne le lemme suivant, que l'on démontre par une récurrence immédiate sur `f`.

```
Lemma WF_checked_form : ∀ Ξ Δ f,
    check_form Ξ Δ f = true → WF_form Ξ Δ f.
```

On pourrait se passer du prédicat `WF_form` Ξ Δ `f` en le remplaçant par la proposition `check_form` Ξ Δ `f = true`, mais les prédicats sont plus faciles à manipuler dans les preuves, notamment avec la tactique `inversion`, qui effectue une analyse par cas sur le prédicat en éliminant les cas absurdes automatiquement. L'utilisation de l'égalité booléenne entraînerait en revanche l'apparition de beaucoup de cas absurdes où cette égalité deviendrait `false = true`. En quelque sorte, cette redondance permet de jouer sur les deux tableaux : on obtient ainsi à la fois un prédicat prouvable par réflexion et un prédicat inductif permettant l'inversion et l'induction.

Préservation de l'interprétation par affaiblissement

Au cours du raisonnement, on est amené à introduire de nouvelles hypothèses ou variables (règles $\rightarrow_I$ et $\forall_I$) : il faut alors s'assurer que l'interprétation des formules ne s'en trouve pas changée. Le problème est que lors des preuves par récurrence, on va devoir prouver des propriétés du type $(\forall x.Px = Qx) \rightarrow (\forall x, Px) = (\forall x, Qx)$. Malheureusement, l'égalité de Coq ne permet pas de prouver ce type de théorème puisqu'elle est intensionnelle, c'est-à-dire qu'elle n'identifie pas deux fonctions dont les valeurs sont égales en tout point du domaine. Or une égalité qui permettrait de prouver la propriété ci-dessus devrait être extensionnelle pour rendre égaux deux prédicats P et Q égaux en tout point du domaine.

Pour remédier à ce problème, une solution simple est de remplacer l'égalité par l'équivalence logique $\leftrightarrow$, définie par $(A \leftrightarrow B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$. Nous serons alors en mesure de prouver $(\forall x.Px \leftrightarrow Qx) \rightarrow ((\forall x.Px) \leftrightarrow (\forall x.Qx))$. Bien que l'équivalence logique soit plus large que l'égalité, elle préserve la validité des propositions et s'avère donc suffisante pour nos besoins. Nous pouvons alors prouver quatre lemmes d'affaiblissement sur le type `form`.

```
Lemma Weak_WF_form : ∀ Ξ Δ dom G, Full Ξ →
  WF_form Ξ Δ G → WF_form (Ξ;dom) Δ G.

Lemma Bind_WF_form : ∀ Ξ Δ Δ0 G, Full Ξ →
  WF_form Ξ Δ G → WF_form Ξ (Δ ++ Δ0) G.

Lemma Weak_interp_form: ∀ Ξ F ξ x dom G δ, Instance Ξ F ξ →
  WF_form Denv Ξ (List.map OE_domain δ) G →
  (⟦G⟧ξ,δ ↔ ⟦G⟧(ξ;{ x∈dom}),δ).

Lemma Bind_interp_form : ∀ Ξ F ξ δ0 G δ, Instance Ξ F ξ →
  WF_form Ξ (List.map OE_domain δ) G →
  (⟦G⟧ξ,δ ↔ ⟦G⟧ξ,(δ++δ0)).
```

Le prédicat utilisé dans les deux derniers lemmes, `Instance` Ξ `F` ξ, signifie que la valuation ξ : `Store obj_entry` est une instance du contexte de variables Ξ : `Store domain`, c'est-à-dire que si l'on projette pour chaque paire de ξ le domaine correspondant, on obtient

Ξ. Ce prédicat permet de quantifier sur toutes les valuations possibles d'un contexte de variables Ξ, on le définit de la façon suivante :

```
Inductive Instance : ∀ Ξ, Full Ξ → ∀ ξ : Type :=
  I_empty : Instance empty F_empty empty
| I_var : ∀ Ξ F ξ dom x, Instance Ξ F ξ →
  Instance (Ξ; dom) (F_push dom Ξ F) (ξ; {x∈dom}).
```

Pour le contexte des variables liées, nous n'utilisons pas un prédicat mais la fonction `List.map OE_domain` δ qui construit la liste des domaines contenus dans les paires de la liste δ.

Correction de l'instanciation

La fonction d'instanciation `inst_form` est utilisée dans notre calcul de deux manières différentes :

- Dans les règles $\forall_E$ et $\exists_I$, `inst_form` est utilisée pour calculer $P\{x \mapsto t\}$ à partir de $\forall x.P$.
- Dans les règles $\forall_I$ et $\exists_I$, `inst_form` est utilisée pour calculer $P\{x \mapsto X\}$ à partir de $\forall x.P$, où X est la variable introduite dans le contexte Δ.

Pour chacune de ces utilisations, il faut prouver que l'instanciation préserve la bonne formation des formules :

```
Lemma WF_Forall_instx : ∀ ξ (F:Full ξ) dom G,
WF_form ξ nil (∀̇_dom G) →
WF_form (ξ; dom) nil (inst_form (Var (index ξ)) 0 G).

Lemma WF_Exists_instx : ∀ ξ (F:Full ξ) dom G,
WF_form ξ nil (∃̇_dom G) →
WF_form (ξ; dom) nil (inst_form (Var (index ξ)) 0 G).

Lemma WF_form_inst : ∀ Ξ t dom, WF_ground_term Ξ t dom →
∀ G Δ, WF_form Ξ (Δ ++ (dom::nil)) G →
WF_form Ξ hyps Δ (inst_form t (length Δ) G).
```

Le prédicat `WF_ground_term` est analogue à `WF_term` mais vérifie que le terme est clos, c'est-à-dire sans variable liée `DB` i. Maintenant, pour spécifier le comportement de la fonction, nous énonçons puis prouvons les lemmes suivants :

```
Lemma form_instx : ∀ Ξ F ξ, Instance Ξ F ξ → ∀ dom x G δ,
  WF_form Ξ (List.map OE_domain (δ ++ ({x∈dom} :: nil))) G →
  (⟦inst_form (Var (index Ξ)) (length δ) G⟧_(ξ;{x∈dom}),δ ↔
  ⟦G⟧_ξ,(δ++({x∈dom}::nil)) ·                                    ··/..
```

```
Lemma form_inst : ∀ Ξ F ξ t dom x, Instance Ξ F ξ →
WF_ground_term Ξ t dom → interp_term ξ nil t dom = Some x →
∀ G δ, WF_form Ξ (List.map OE_domain (δ++({x∈dom}::nil))) G →
([[G]]_{ξ,(δ++({x∈dom}::nil))} ↔ [[inst_form t (length δ) G]]_{ξ,δ}
```

Schéma de réflexion

Comme pour les objets, nous avons choisi de définir un prédicat `WF_proof` qui correspond à la notation $\pi : \Xi, \Gamma \vdash G$ de la figure 4.5 et de prouver qu'il est la réflexion de la fonction `check_proof` :

```
Lemma WF_checked_proof : ∀ π Ξ Γ G,
    check_proof Ξ Γ G π = true → WF_proof Ξ Γ G π.
```

et ce prédicat nous permet d'énoncer le théorème fondamental :

Théorème 4.10 (Principe de réflexion)
Les théorèmes suivants peuvent être prouvés dans Coq :

$$\forall \pi : \texttt{proof}. \forall\, \Xi\, \Gamma\, G. (\texttt{WF\_seq}\ \Xi\ \Gamma\ G) \rightarrow (\texttt{WF\_proof}\ \Xi\ \Gamma\ G\ \pi) \rightarrow [\![\Xi, \Gamma \vdash G]\!] \tag{4.1}$$

$$\forall \pi : \texttt{proof}. \forall\, \Xi\, \Gamma\, G.\ \texttt{if}\ (\texttt{check\_seq}\, \Xi\, \Gamma\, G)\ \&\&\ (\texttt{check\_proof}\ \Xi\ \Gamma\ G)\ \texttt{then}\ [\![\Xi, \Gamma \vdash G]\!]\ \texttt{else True} \tag{4.2}$$

Comme dans le cas propositionnel, la démonstration procède par récurrence sur la trace de preuve π. Le théorème (4.2) est une conséquence directe de (4.1), par l'utilisation des lemmes `WF_checked_seq` et `WF_checked_proof`.

Pour conclure cette section, nous allons examiner un exemple qui va illustrer notre propos.

Exemple 4.11
Cet exemple est la preuve que si n est pair, alors $n + 2$ l'est aussi. Nous commençons par spécifier les symboles que nous allons utiliser.

```
Variable N:Set.
Variable S:N → N.
Variables pair impair: N → Prop.
```

Puis nous annonçons à Coq la propriété que nous allons prouver :

```
Lemma pair_plus_deux :
(∀ x, pair x → impair (S x)) →
(∀ x, impair x → pair (S x)) →
(∀ x, pair x → pair (S (S x))).
```

Ensuite, il faut définir la version réifiée de la signature, qui va nous donner les domaines de quantification, les fonctions et les prédicats, puis la version réifiée de la propriété à prouver.

```
pose (Denv := empty; N).
pose (Tenv := empty; {S∈1→1}).
pose (Penv := empty; {impair∈1→*}; {pair∈1→*}).

pose (goal :=
  (∀̇₁, Atom 2 (DB 0) ⇢ Atom 2 (App 1 (App 1 (DB 0))))).
pose (hyp2 := (∀̇₁, Atom 2 (DB 0) ⇢ Atom 2 (App 1 (DB 0)))).
pose (hyp1 := (∀̇₁, Atom 1 (DB 0) ⇢ Atom 2 (App 2 (DB 0)))).
```

Nous donnons ensuite la trace de preuve :

```
pose (prf :=
 (∀_I (→_I (∀_E 1 (Var 1) (∀_E 2 (App 1 (Var 1))
 (→_E 3 4 (→_E 6 5 (Ax 7)))))))).
```

Enfin nous pouvons donner directement à Coq la preuve utilisant le principe de réflexion :

```
exact (Reflect Denv Tenv Penv prf empty
        (empty;hyp1; hyp2; nil) goal) .
Qed.
```

4.4 Ajout de l'égalité

4.4.1 Réification de l'égalité et de la réécriture

Pour ajouter l'égalité dans le cadre que nous avons défini, il faut d'abord étendre le type `form` par la construction $\texttt{term} \underset{\texttt{domain}}{\doteq} \texttt{term}$ qui représente l'égalité de deux termes partageant un certain domaine, et nous définissons l'interprétation de cette formule par l'équation suivante :

$$\left[\!\left[t_1 \underset{d}{\doteq} t_2 \right]\!\right]_{\xi,\delta} \quad = \quad \left([\![t_1]\!]^d_{\xi,\delta} \underset{[\![d]\!]_{\mathrm{Dom}}}{=} [\![t_2]\!]^d_{\xi,\delta} \right)$$

On ajoute alors le cas suivant dans la fonction `interp_form` :

```
| t1 ≐ t2 =>
  match [[t1]]^dom_ξ,δ , [[t2]]^dom_ξ,δ with
    Some x1, Some x2 => x1=x2
  |    _    ,    _    => True
  end ...
```

Pour modéliser la règle de remplacement, il est nécessaire d'implanter cette opération au niveau des objets réifiés, ce que l'on fait à l'aide de l'opération `rewrite`. Cette opération nécessite que l'on spécifie à quelle(s) position(s) d'une formule on souhaite remplacer un terme clos par un autre. Pour ce faire, il faut définir un type `occ` représentant des positions dans les formules et les termes.

Définition 4.12 (Occurrences)
Les ensembles d'occurrences de termes dans les formules sont représentés par les types récursifs suivants :

$$
\begin{array}{rcl}
\mathtt{occ}_{term} & := & \ominus \\
& | & \triangle \\
& | & @\ \mathtt{occ}_{args} \\
\mathtt{occ}_{args} & := & \ominus \\
& | & \mathtt{occ}_{term}; \mathtt{occ}_{args}.
\end{array}
\qquad
\begin{array}{rcl}
\mathtt{occ}_{form} & := & \ominus \\
& | & @\ \mathtt{occ}_{args} \\
& | & \mathtt{occ}_{term} \mathbin{\hat{=}} \mathtt{occ}_{term} \\
& | & \mathtt{occ}_{form} \bowtie \mathtt{occ}_{form} \\
& | & \bigwedge \mathtt{occ}_{form}
\end{array}
$$

Le symbole $\ominus$ désigne l'ensemble vide de positions et le symbole $\triangle$ désigne l'occurrence en tête du terme. Les autres symboles permettent de construire des ensembles de positions dans des sous-termes ou des sous-formules : le symbole @ construit un ensemble de positions dans les arguments d'une fonction ou d'un prédicat, le symbole $\hat{=}$ unit les ensembles de positions dans les deux membres d'une égalité, le symbole $\bowtie$ unit les ensembles de positions dans les deux sous-formules d'un connecteur et le symbole $\bigwedge$ désigne les positions sous un quantificateur.

Les fonctions de réécriture sont relativement simples à écrire ; on notera l'utilisation de la fonction booléenne `term_eq` qui vérifie que le terme que l'on remplace est bien celui que l'on souhaite remplacer.

```
Fixpoint rewrite_term ta tb occ t {struct t} :
 option term :=
  match occ with ⊖ => Some t
  | △ => if term_eq t ta then Some tb else None
  | @ aocc =>
    match t with App f args =>
       match rewrite_args ta tb aocc args with None => None
       | Some rargs => Some (App f rargs) end
    | _ => None
    end end                                                    ../..
```

```
with rewrite_args ta tb aocc a {struct a} :
  option args:=
  match aocc,a with
    ⊖,_ => Some a
  | occ; occq, t; q  =>
     match rewrite_term ta tb occ  t,
           rewrite_args ta tb occq q with
       Some rt,Some rq => Some (rt; rq)
     | _,_ => None end
  | _,_ => None
  end end.

Fixpoint rewrite_form ta tb occ G {struct G} :
 option form :=
  match occ,G with
    ⊖, _ => Some G
  | @ aocc, Atom p args =>
    match rewrite_args ta tb aocc args with None => None
    | Some rargs => Some (Atom p rargs) end
  | occ1 ≟° occ2, t1 ≟[d] t2 =>
    match rewrite_term ta tb occ1 t1,
          rewrite_term ta tb occ2 t2 with
      Some t1',Some t2' => Some (t1' ≟[d] t2') | _,_ => None end
  | occ1 ⋈ occ2, f1 ⇀ f2 =>
    match rewrite_form ta tb occ1 f1,
          rewrite_form ta tb occ2 f2 with
      Some f1',Some f2' => Some (f1 ⇀ f2') | _,_ => None end
  | occ1 ⋈ occ2, f1 ∧̇ f2 =>
    match rewrite_form ta tb occ1 f1,
          rewrite_form ta tb occ2 f2 with
      Some f1',Some f2' => Some (f1' ∧̇ f2') | _,_ => None end
  | occ1 ⋈ occ2, f1 ∨̇ f2 =>
    match rewrite_form ta tb occ1 f1,
          rewrite_form ta tb occ2 f2 with
      Some f1',Some f2' => Some (f1' ∨̇ f2') | _,_ => None end
  | ⋀ occ0, ∀̇[d] f0 =>
    match rewrite_form ta tb occ0 f0 with None => None
    | Some f0'  => Some (∀̇[d] f0' ) end
  | ⋀ occ0, ∃̇[d] f0 =>
    match rewrite_form ta tb occ0 f0 with None => None
    | Some f0'  => Some (∃̇[d] f0' ) end
                                                                    ../..
```

```
  | _,_ => None
end.
```

La réécriture sous les quantificateurs pourrait poser des problèmes de capture, par exemple si l'on remplaçait le terme réifié (App 1 []) par (DB 0), mais nous avons ici supposé que les termes remplacés sont clos, ce qui permet de dire qu'il ne contiennent pas d'indice de deBruijn.

Pour prouver la correction du système étendu avec l'égalité, il faut d'abord étendre les fonctions et les définitions du système de base et ajouter des lemmes prouvant que la réécriture donne des termes et des formules bien formés :

```
Lemma WF_rewrite_form : ∀ Γ dom t1 t2,
  WF_ground_term Γ t1 dom → WF_ground_term Γ t2 dom →
  ∀ G occ G' Δ, rewrite_form t1 t2 occ G = Some G' →
  WF_form Γ Δ G → WF_form Γ Δ G'.
```

La sémantique de la fonction de réécriture est donnée par le lemme suivant : si les termes que l'on remplace sont égaux, alors la formule réécrite est équivalente à la formule originale (et non égale car on peut réécrire sous des quantificateurs, ce qui pose le problème de l'extensionnalité).

```
Lemma form_rewrite : ∀ Γ F ξ, Instance Γ F ξ →
  ∀ dom t1 t2 t1' t2',
  WF_ground_term Γ t1 dom → WF_ground_term Γ t2 dom →
  ⟦t1⟧ξ,nil = Some t1' → ⟦t2⟧ξ,nil = Some t2' → t1' = t2' →
  ∀ G occ δ G', rewrite_form t1 t2 occ G = Some G' →
  WF_form Γ (List.map OE_domain δ) G →
  (⟦G⟧ξ,δ ↔ ⟦G⟧ξ,δ).
```

4.4.2 Preuve par réécriture

Il faut maintenant ajouter dans le type `proof` les règles d'inférence pour l'égalité. On choisit d'introduire un type intermédiaire `dir` := ← | → pour indiquer dans quel sens on souhaite réécrire l'égalité. La sémantique de ces nouvelles règles est donnée dans la figure 4.6.

```
proof := ...                                   (cf. 4.8)
       | =I
       | =E1 positive dir occ proof
       | =E2 positive positive dir occ proof
       | =D positive proof
```

L'ajout des cas correspondants dans la fonction `check_proof` se fait de façon immédiate.

$$\frac{}{=_I : \Xi, \Gamma \vdash t \mathrel{\dot{=}}_{\tau} t} \qquad \frac{\pi : \Xi, \Gamma; A \vdash G}{(=_D\ i\ \pi) : \Xi, \Gamma \vdash G}\ \Gamma_i = (t \mathrel{\dot{=}}_{\tau} t) \dot{\rightarrow} A$$

$$\frac{\pi : \Xi, \Gamma \vdash (\texttt{rewrite}\ s\ t\ p\ G)}{(=_{E_1}\ i\ \rightarrow\ p\ \pi) : \Xi, \Gamma \vdash G}\ \Gamma_i = s \mathrel{\dot{=}}_{d} t$$

$$\frac{\pi : \Xi, \Gamma \vdash (\texttt{rewrite}\ t\ s\ p\ G)}{(=_{E_1}\ i\ \leftarrow\ p\ \pi) : \Xi, \Gamma \vdash G}\ \Gamma_i = s \mathrel{\dot{=}}_{d} t$$

$$\frac{\pi : \Xi, \Gamma; (\texttt{rewrite}\ s\ t\ p\ A) \vdash G}{(=_{E_2}\ i\ j\ \rightarrow\ p\ \pi) : \Xi, \Gamma \vdash G}\ \begin{cases} \Gamma_i = s \mathrel{\dot{=}}_{d} t \\ \Gamma_j = A \end{cases}$$

$$\frac{\pi : \Xi, \Gamma; (\texttt{rewrite}\ t\ s\ p\ A) \vdash G}{(=_{E_2}\ i\ j\ \leftarrow\ p\ \pi) : \Xi, \Gamma \vdash G}\ \begin{cases} \Gamma_i = s \mathrel{\dot{=}}_{d} t \\ \Gamma_j = A \end{cases}$$

FIG. 4.6 – Règles d'inférence pour l'égalité

```
| =_I =>
  match G with t1 ≐_dom t2 => term_eq t1 t2
  | _ => false end
| =_E1 i dir occ p =>
  match get i Γ with PSome (t1 ≐_dom t2) =>
    match
      (if dir (* = ← *) then
         rewrite_form t1 t2 occ G else
         rewrite_form t2 t1 occ G) with None => false
    | Some G' => check_proof Ξ Γ G' p end
  | _ => false end
| =_E2 i j dir occ p =>
  match get i Γ,get j Γ with
    PSome H,PSome (t1 ≐_dom t2) =>
      match
        (if dir (* = ← *) then
          rewrite_form t1 t2 occ H else
          rewrite_form t2 t1 occ H) with None => false
      | Some f0' => check_proof Ξ (Γ; f0' ) G p end
      | _,_ => false end
| =_D i p =>
  match get i Γ with
    PSome ((t1 ≐_dom t2) ⇀ C) =>
      term_eq t1 t2 && check_proof Ξ (Γ; C) G p
  | _ => false end
```

Enfin, grâce aux lemmes intermédiaires, nous pouvons étendre le principe de réflexion à ces nouvelles règles.

4.5 Interprétation de traces de réécriture

Dans cette partie, nous allons décrire une expérience réalisée avec Évelyne Contejean : l'utilisation de notre principe de réflexion pour valider des preuves par réécriture calculées par l'outil CiME. Ces preuves sont obtenues par le calcul de complétion ordonnée à partir de problèmes de la bibliothèque TPTP. Cette expérience a fait l'objet de la publication [CC05].

4.5.1 Complétion ordonnée

Le but de la complétion ordonnée [Pet83, HR87, BDP89] est de construire un système de réécriture confluent et normalisant [Ter03] à partir d'un ensemble d'équations afin de décider le problème du mot, c'est-à-dire si l'égalité de deux termes est une conséquence équationnelle des égalités du système de réécriture. Certains prouveurs automatiques comme CiME [CMU04] ou Waldmeister [HL02] utilisent une version étendue de la complétion ordonnée pour résoudre également des problèmes d'unifiabilité : existe-t-il une instance de l'égalité à prouver qui soit une conséquence du système de réécriture ?

Nous allons utiliser la présentation classique du calcul de complétion comme un ensemble d'applications de règles d'inférence. La donnée d'entrée pour ce calcul est une paire $(E_0, s = t)$, où E_0 est un ensemble d'égalités (universellement closes) qui définissent une théorieéquationnelle, et où $s = t$ est une conjecture (existentiellement close). Le résultat est **True** quand il existe une substitution σ telle que $s\sigma$ et $t\sigma$ soient égaux modulo E_0, et **False** sinon.

Toutefois, cette procédure de semi-décision peut ne pas terminer. L'ensemble du calcul utilise un ordre de réduction $>$. Les règles de complétion sont des transitions entre des triplets (E, R, C), où E et C sont des ensembles d'équations (paires de termes non ordonnées), alors que R est un ensemble de règles (paires ordonnées de termes).

L'ensemble des règles de complétion (voir figure 4.7) se divise en plusieurs groupes :

- La règle **Init**$(E_0 \vdash s = t)$ démarre le calcul en construisant le triplet initial à partir des axiomes de l'ensemble E_0 et de la conjecture $s = t$.
- Les règles **Orient** et **Orient'** ordonnent des égalités $u = v$ de E pour en faire des règles de réécriture dans R. Si u et v sont comparables par $>$, alors on crée une seule règle dans le sens décroissant, sinon on peut avoir $u\sigma > v\sigma$ ou $v\sigma > u\sigma$ suivant le choix de la substitution σ et on doit donc ajouter les règles dans les deux sens.
- Les règles de réécriture sont utilisées par les règles d'inférence **Rewrite**, **Rewrite'**, **Collapse** et **Compose** pour réécrire respectivement un membre d'une équation de E, une conjecture de C, le membre gauche d'une règle de réécriture ou son membre droit.

– De nouvelles conjectures ou égalités sont calculées par calcul de paires critiques entre deux règles (**Critical pair**) ou une règle et une conjecture (**Narrow**). Une application successive de règles d'inférence peut continuer *ad infinitum* ou déboucher sur une application de la règle **Success**.

4.5.2 Annotation des règles et traces de complétion

Pour construire une trace destinée à un logiciel de gestion de preuves formelles comme Coq, les règles d'inférence doivent être annotées par des informations supplémentaires. Les règles se divisent alors en deux catégories : celles qui *simplifient* les équations ou les règles en les réécrivant et celles qui créent de nouvelles équations ou conjectures. L'application des règles de simplification est inscrite dans l'historique du terme simplifié, et les nouvelles équations et conjectures sont annotées par l'étape de raisonnement qui les a créées.

De plus, il faut identifier avec précision les membres gauches et droits des équations, ce qui conduit à dupliquer certaines règles. Nous ne donnons ici que la version gauche de ces règles quand la version droite peut se déduire sans ambiguïté.

Les objets que nous manipulons vont donc être annotés de la façon suivante :

– Une équation est une paire de termes avec une trace, nous la noterons $u = v$, lorsque la trace ne sera pas importante.
– Un terme u a une version actuelle u^*, une version originale u°, et un historique consistant en une suite d'étapes de réécriture permettant de réécrire u° en u^*.
– Une étape de réécriture est donnée par une position, une substitution et une règle de réécriture.
– Une règle de réécriture est une équation $u = v$ orientée ($+$ ou $-$) que nous noterons $u \xrightarrow{+} v$ ou $v \xrightarrow{-} u$ suivant le sens choisi.
– Une trace est, ou un axiome (ω), ou un « pic » correspondant à une paire critique (κ), ou une spécialisation (*narrowing*) (soit dans le membre gauche (ν_L), soit dans le membre droit (ν_R)).
– Un pic est la donnée du terme commun t, de deux règles rl_1 et rl_2, et de la position à laquelle la règle la plus profonde rl_2 doit s'appliquer à t : $\xleftarrow[p]{rl_2} t \xrightarrow[\Lambda]{rl_1}$.

Un axiome (paire de termes nus $s = t$) peut être transformé en une équation $\widehat{s = t}$ annotée par la trace $\omega(s = t)$ et où les termes s et t sont annotés par l'historique vide.

Étant donné un terme u, une règle $l \to r$, une position p et une substitution σ, avec $u^*|_p = l^*\sigma$, le terme u peut être réécrit en un nouveau terme $u' = Rew(u, \longrightarrow_{p,\sigma}^{l \to r})$, où $u'^\circ = u^\circ$, $u'^* = u^*[r^*\sigma]_p$ et l'historique de u' est h; $(p, \sigma, l \to r)$, où h est l'historique de u. Les règles de complétion avec annotations sont résumées dans la Figure 4.8.

4.5.3 Construction d'une trace de preuve à partir des annotations

Quand un triplet (E, R, C) peut faire l'objet d'une application de la règle **Success**, toute l'information nécessaire pour construire une preuve formelle peut être extraite de la trace de la conjecture $s = t$ telle que s^* et t^* soient unifiables. La preuve formelle est

$$\frac{}{E_0, \emptyset, \{s = t\}} \textbf{Init}(E_0 \vdash s = t)$$

$$\frac{\{u = v\} \cup E, R, C}{E, \{u \to v\} \cup R, C} \textbf{Orient} \text{ si } u > v.$$

$$\frac{\{u = v\} \cup E, R, C}{E, \{u \to v; v \to u\} \cup R, C} \textbf{Orient'} \text{ si } u \not< v \text{ et } v \not< u.$$

$$\frac{\{u = v\} \cup E, \{l \to r\} \cup R, C}{\{u[r\sigma]_p = v\} \cup E, \{l \to r\} \cup R, C} \textbf{Rewrite} \text{ si } u|_p = l\sigma \text{ et } l\sigma > r\sigma.$$

$$\frac{E, \{l \to r\} \cup R, \{s = t\} \cup C}{E, \{l \to r\} \cup R, \{s[r\sigma]_p = t\} \cup C} \textbf{Rewrite'} \text{ si } s|_p = l\sigma \text{ et } l\sigma > r\sigma.$$

$$\frac{E, \{l \to r, g \to d\} \cup R, C}{\{l[d\sigma]_p = r\} \cup E, \{g \to d\} \cup R, C} \textbf{Collapse} \text{ si } l|_p = g\sigma.$$

$$\frac{E, \{l \to r, g \to d\} \cup R, C}{E, \{l \to r[d\sigma]_p, g \to d\} \cup R, C} \textbf{Compose} \text{ si } r|_p = g\sigma.$$

$$\frac{E, \{l \to r, g \to d\} \cup R, C}{\{l\rho_1[d\rho_2]_p\sigma = r\rho_1\sigma\} \cup E, \{l \to r, g \to d\} \cup R, C} \textbf{Critical pair}^a \text{ si } l\rho_1|_p\sigma = g\rho_2\sigma.$$

$$\frac{E, \{g \to d\} \cup R, \{s = t\} \cup C}{E, \{g \to d\} \cup R, \{s\rho_1[d\rho_2]_p\sigma = t\rho_1\sigma\} \cup \{s = t\} \cup C} \textbf{Narrow}^b \text{ si } s\rho_1|_p\sigma = g\rho_2\sigma.$$

$$\frac{E, R, \{s = t\} \cup C}{True} \textbf{Success} \text{ si } s \text{ et } t \text{ sont unifiables}$$

[a] Les règles $l \to r$ et $g \to d$ doivent subir les renommages respectifs ρ_1 et ρ_2 de façon à séparer leurs ensembles de variables.

[b] La conjecture $s = t$ et la règle $l \to r$ doivent subir les renommages respectifs ρ_1 et ρ_2 de façon à séparer leurs ensembles de variables.

Fig. 4.7 – Règles du calcul de complétion

$$\frac{}{\widehat{E_0}, \emptyset, \widehat{\{s = t\}}}\ \textbf{Init} \qquad \frac{E, R, \{s = t\} \cup C}{True}\ \textbf{Success}$$

$$\frac{\{u = v\} \cup E, R, C}{E, \{u \xrightarrow{+} v\} \cup R, C}\ \textbf{Orient}_L \qquad \frac{\{u = v\} \cup E, R, C}{E, \{v \xrightarrow{-} u\} \cup R, C}\ \textbf{Orient}_R$$

$$\frac{\{u = v\} \cup E, R, C}{E, \{u \xrightarrow{+} v; v \xrightarrow{-} u\} \cup R, C}\ \textbf{Orient'}$$

$$\frac{\{u = v\} \cup E, \{l \xrightarrow{\pm} r\} \cup R, C}{\{Rew(u, \xrightarrow[p,\sigma]{l \xrightarrow{\pm} r}) = v\} \cup E, \{l \xrightarrow{\pm} r\} \cup R, C}\ \textbf{Rewrite}_L$$

$$\frac{E, \{l \xrightarrow{\pm} r\} \cup R, \{s = t\} \cup C}{E, \{l \xrightarrow{\pm} r\} \cup R, \{Rew(s, \xrightarrow[p,\sigma]{l \xrightarrow{\pm} r}) = t\} \cup C}\ \textbf{Rewrite'}_L$$

$\textbf{Rewrite}_R$ $\qquad$ $\textbf{Rewrite'}_R$

$$\frac{E, \{l \xrightarrow{+} r; g \xrightarrow{\pm} d\} \cup R, C}{\{Rew(l, \xrightarrow[p,\sigma]{g \xrightarrow{\pm} d}) = r\} \cup E, \{g \xrightarrow{\pm} d\} \cup R, C}\ \textbf{Collapse}_+$$

$$\frac{E, \{l \xrightarrow{-} r; g \xrightarrow{\pm} d\} \cup R, C}{\{r = Rew(l, \xrightarrow[p,\sigma]{g \xrightarrow{\pm} d})\} \cup E, \{g \xrightarrow{\pm} d\} \cup R, C}\ \textbf{Collapse}_-$$

$$\frac{E, \{l \xrightarrow{\pm} r; g \xrightarrow{\pm} d\} \cup R, C}{E, \{l \xrightarrow{\pm} Rew(r, \xrightarrow[p,\sigma]{g \xrightarrow{\pm} d}); g \xrightarrow{\pm} d\} \cup R, C}\ \textbf{Compose}$$

$$\frac{E, \{l \xrightarrow{\pm} r; g \xrightarrow{\pm} d\} \cup R, C}{\left\{ \begin{array}{c} l^*\rho_1[d^*\rho_2]_p\sigma = r^*\rho_1\sigma \\ \text{par } \kappa(\xleftarrow[p]{g \xrightarrow{\pm} d} l\rho_1\sigma \xrightarrow[\Lambda]{l \xrightarrow{\pm} r}) \end{array} \right\} \cup E, \{l \xrightarrow{\pm} r; g \xrightarrow{\pm} d\} \cup R, C}\ \textbf{Critical pair}$$

$$\frac{E, \{g \xrightarrow{\pm} d\} \cup R, \{s = t\} \cup C}{E, \{g \xrightarrow{\pm} d\} \cup R, \left\{ \begin{array}{c} s^*\rho_1[d^*\rho_2]_p\sigma = t^*\rho_1\sigma \\ \text{par } \nu_L(\xleftarrow[p]{g \xrightarrow{\pm} d} s\rho_1\sigma \xrightarrow[\Lambda]{s \xrightarrow{+} t}) \end{array} \right\} \cup \{s = t\} \cup C}\ \textbf{Narrow}_L$$

$$\frac{E, \{g \xrightarrow{\pm} d\} \cup R, \{s = t\} \cup C}{E, \{g \xrightarrow{\pm} d\} \cup R, \left\{ \begin{array}{c} s^*\rho_1\sigma = t^*\rho_1[d^*\rho_2]_p\sigma \\ \text{par } \nu_R(\xleftarrow[p]{g \xrightarrow{\pm} d} t\rho_1\sigma \xrightarrow[\Lambda]{t \xrightarrow{-} s}) \end{array} \right\} \cup \{s = t\} \cup C}\ \textbf{Narrow}_R$$

FIG. 4.8 – Règles de complétion annotées

alors construite en deux étapes : les listes d'étapes de réécriture sont d'abord extraites des annotations, puis transformées en trace de preuve réifiée ou en script de tactiques.

Problème du mot

Dans un premier temps, nous allons traiter le cas le plus simple, lorsque la conjecture de départ est une égalité entre termes clos. Les règles de spécialisation ne s'appliquent pas, l'ensemble de conjectures est toujours un singleton et la règle **Success** s'applique seulement lorsque les deux membres de la conjecture sont syntaxiquement égaux dans leur version courante.

Pour construire la preuve, deux possibilités s'offrent à nous : la première est de donner une séquence d'étapes de réécriture entre les deux membres de la conjecture de départ, ce qui revient à donner une preuve sans coupures de cette conjecture. Pour obtenir ce résultat, il est nécessaire de déplier récursivement toutes les paires critiques, ce qui s'avère coûteux et relativement complexe, alors que nous disposons de l'alternative qui consiste à prouver chaque paire critique comme un lemme intermédiaire avant de donner la preuve de la conjecture finale.

Les paires critiques comme coupures Tout d'abord, il est à noter que toutes les applications de règles de complétion ne seront pas présentes dans la preuve finale. Nous procédons donc à la sélection et à l'analyse de dépendances sur des règles réellement utiles, en commençant par les règles des historiques de s et t. L'ensemble des règles utiles est donné par l'ensemble $\mathcal{U}(s) \cup \mathcal{U}(t)$, où $\mathcal{U}$ est défini comme suit :

$$\begin{aligned}
&\mathcal{U}(u) = \textstyle\bigcup_{l \xrightarrow{\pm} r \in \mathrm{His}(u)} \mathcal{T}(l \xrightarrow{\pm} r)\\
&\mathcal{T}(l \xrightarrow{\pm} r) = \{l \xrightarrow{\pm} r\} \cup \mathcal{U}(l) \cup \mathcal{U}(r) \text{ quand } \mathrm{Trc}((l \xrightarrow{\pm} r)) = \omega(u, v)\\
&\mathcal{T}(l \xrightarrow{\pm} r) = \{l \xrightarrow{\pm} r\} \cup \mathcal{U}(l) \cup \mathcal{U}(r) \cup \mathcal{T}(l_1 \xrightarrow{\pm} r_1) \cup \mathcal{T}(l_2 \xrightarrow{\pm} r_2)\\
&\qquad \text{quand } \mathrm{Trc}((l \xrightarrow{\pm} r)) = \kappa(\xleftarrow[p]{l_2 \xrightarrow{\pm} r_2} l_1\sigma \xrightarrow[\Lambda]{l_1 \xrightarrow{\pm} r_1})
\end{aligned}$$

L'ensemble des règles utiles peut être trié topologiquement par rapport à la relation $\prec$ définie par $rl_1 \prec rl_2 \Leftrightarrow rl_1 \in \mathcal{T}(rl_2) \setminus \{rl_2\}$. Pour y arriver, la solution la plus simple est de dater les créations de nouvelles règles.

Chaque paire critique peut être vue comme la clôture universelle d'une égalité entre deux termes dont la preuve n'utilise que les axiomes de E_0 et les lemmes précédents (pour $\prec$).

Quand la trace de $l \xrightarrow{\pm} r$ est $\omega(u = v)$, cela veut dire que la règle est obtenue par l'égalité initiale $u = v \in E_0$ en réécrivant éventuellement les membres de cette égalité. La preuve de $l^* = r^*$ est alors constituée de la concaténation de :

1. l'historique de l retourné.
2. $u = v$ en position de tête avec la substitution identité et le signe $+$ si la règle est dans le même sens que l'équation, $-$ sinon.

3. l'historique de r.

Quand la trace d'une règle $l \stackrel{\pm}{\to} r$ est $\kappa(\frac{l_2 \stackrel{\pm}{\to} r_2}{p} l_1\sigma \frac{l_1 \stackrel{\pm}{\to} r_1}{\Lambda})$, la règle est obtenue par une paire critique entre deux règles et éventuellement un changement d'orientation. On prouve alors que $l^* = r^*$ par la suite d'étapes de réécriture suivante :

1. l'historique de l retourné.
2. une preuve de l'égalité $l^\circ = r^\circ$, suivant l'orientation :
   (a) quand l'orientation est conservée :
      i. $l_2 \stackrel{\pm}{\to} r_2$ appliquée vers l'avant à la position p avec la substitution σ_2 telle que $l_2^*\sigma_2 = l_1^*\sigma|_p$ et $r_2^*\sigma_2 = l^\circ|_p$.
      ii. $l_1 \stackrel{\pm}{\to} r_1$ appliquée vers l'arrière en position de tête avec la substitution σ_1 telle que $l_1^*\sigma_1 = l_1^*\sigma$ and $r_1^*\sigma_1 = r^\circ$,
   (b) quand l'orientation est inversée :
      i. $l_1 \stackrel{\pm}{\to} r_1$ appliquée vers l'arrière en position de tête avec la substitution σ_1 telle que $l_1^*\sigma_1 = l_1^*\sigma$ et $r_1^*\sigma_1 = l^\circ$
      ii. $l_2 \stackrel{\pm}{\to} r_2$ appliquée vers l'avant à la position p avec la substitution σ_2 telle que $l_2^*\sigma_2 = l_1^*\sigma|_p$ et $r_2^*\sigma_2 = r^\circ|_p$
3. l'historique de r.

La preuve de la conjecture $s^\circ = t^\circ$ est alors constituée de la concaténation de l'historique de s avec l'historique inversé de t.

Dépliage de la preuve Pour obtenir une preuve dépliée (sans coupures) de $s^\circ = t^\circ$, il suffit de remplacer chaque règle allant de u à v à la position p avec la substitution σ par sa trace de preuve placée sous le contexte $u[\_]_p$ vers l'avant ou $v[\_]_p$ vers l'arrière, en instanciant la liste par σ.

Problèmes d'unifiabilité

Dans ce cas, les règles de spécialisation (*narrowing*) peuvent être utilisées, et comme dans le cas précédent, on peut construire une preuve dépliée utilisant uniquement les équations de E_0 ou une suite de lemmes.

Il y aura alors deux sortes de lemmes : les lemmes pour les paires critiques et les lemmes pour la spécialisation. De nouveau nous pouvons extraire ceux d'entre eux qui sont utiles de la conjecture $s = t$ qui a déclenché la règle **Success**.

La fonction $\mathcal{U}$ reste celle utilisée pour le problème du mot, et $\mathcal{T}$ est étendue naturellement par :

$$\mathcal{T}(l \stackrel{\pm}{\to} r) = \{l \stackrel{\pm}{\to} r\} \cup \mathcal{U}(l) \cup \mathcal{U}(r) \cup \mathcal{T}(l_1 \stackrel{\pm}{\to} r_1) \cup \mathcal{T}(l_2 \stackrel{\pm}{\to} r_2)$$
$$\text{quand } \mathrm{Trc}((l \stackrel{\pm}{\to} r)) = \nu_{L/R}(\frac{l_2 \stackrel{\pm}{\to} r_2}{p} l_1\sigma \frac{l_1 \stackrel{\pm}{\to} r_1}{\Lambda})$$

Paires critiques et spécialisations comme coupures Le calcul de l'ensemble des lemmes utiles est proche de celui utilisé pour le problème du mot quand la trace de $s = t$ est de la forme $\omega(\_)$, mais l'ensemble de départ doit être étendu par $l_1 \xrightarrow{\pm} r_1$ et $l_2 \xrightarrow{\pm} r_2$ quand la trace est de la forme $\nu_{L/R}(\xleftarrow[p]{l_2 \xrightarrow{\pm} r_2} l_1\sigma \xrightarrow[\Lambda]{l_1 \xrightarrow{\pm} r_1})$.

Cet ensemble peut être trié comme précédemment, et la preuve d'un lemme de paire critique est la même qu'auparavant. La preuve d'un lemme de spécialisation est différente : soit $s' = t'$ la conjecture obtenue par spécialisation de la conjecture $s = t$ par la règle $g \xrightarrow{\pm} d$. Du point de vue de la preuve formelle, cela signifie que l'on va remplacer le but $s = t$ à prouver par $s' = t'$. Il faut donc montrer que ce remplacement est correct, c'est-à-dire que $s' = t' \rightarrow s = t$. La preuve de $s = t$ est en fait une suite de réécritures entre s^* et t^* instanciées par la substitution appropriée utilisant $s' = t'$ et des lemmes de paires critiques antérieurs.

Dans le cas d'une spécialisation du membre gauche, par exemple, la substitution est égale à σ_1 telle que $s^*\sigma_1 = s^*\sigma$ et $t^*\sigma_1 = t'^\circ$, et la liste d'étapes est la concaténation de :

1. l'application vers l'avant de la règle $g \xrightarrow{\pm} d$ à la position p avec la substitution σ_2 telle que $g^*\sigma_2 = s^*\sigma|_p$ et $d^*\sigma_2 = s'^\circ|_p$.
2. l'historique de s'.
3. l'application de la règle $s' = t'$ à la position de tête avec la substitution identité.
4. l'historique de t' à l'envers.

Le cas de la spécialisation droite est similaire et les paires critiques sont traitées comme pour le problème du mot.

La preuve de la dernière conjecture $s = t$ (la plus spécialisée, celle qui déclenche la règle **Success**) est l'application d'une substitution σ telle que $s^*\sigma = t^*\sigma$, et la concaténation de l'historique de s et de l'historique inversé de t, où chaque pas de réécriture est instancié par σ.

Dépliage On procède de façon similaire au cas du problème du mot mais on propage la substitution introduite par chaque spécialisation.

4.5.4 Production de preuves réifiées

Nous commençons par montrer comment nous modélisons les problèmes d'unifiabilité dans Coq, ensuite nous décrivons comment réifier une suite d'étapes de réécriture dans un contexte approprié, et finalement nous expliquons comment obtenir une preuve complète en combinant ces morceaux de preuves.

Modélisation de problèmes d'unifiabilité

Une méthode naturelle pour représenter les problèmes d'unifiabilité est de représenter les termes algébriques par des termes Coq (réflexion profonde). Mais comme Coq a un langage typé, il nous faut supposer l'existence d'un type `Domain` : `Set`.

Cet encodage n'est pas totalement innocent car les types de Coq ne sont pas habités par défaut, ce qui contredit la sémantique des problèmes d'unifiabilité : avec un domaine vide, la conjecture $f(x) = f(x)$ (ou plutôt sa clôture existentielle) n'est pas démontrable si la signature ne contient pas de symbole constant. Il nous faut donc supposer que nous avons une constante `dummy` : `Domain` pour modéliser le fait que `Domain` est habité.

Pour chaque symbole de fonction n-aire f de la signature, nous introduisons une constante Coq de type $\overbrace{\texttt{Domain} \to \cdots \to \texttt{Domain}}^{n \text{ times}} \to \texttt{Domain}$. Pour représenter l'égalité, nous utilisons l'égalité polymorphe habituelle `eq`, qui est l'interprétation de l'égalité réifiée $\doteq$. Comme nous ne quantifions que sur le type `Domain`, notre signature de domaines sera $\{1 \mapsto \texttt{Domain}\}$ et nous omettrons les indices de $\doteq$, $\dot{\forall}$ et $\dot{\exists}$. Pour chaque égalité, nous choisirons un ordre total arbitraire sur les variables libres et nous supposerons les égalités quantifiées dans l'ordre croissant pour cet ordre.

Un problème d'unifiabilité est alors la donnée d'une liste d'égalités universellement closes $R_1, \ldots, R_n$ déclarées dans Coq comme hypothèses et d'une égalité existentiellement close que nous souhaitons démontrer.

Séquence d'étapes de réécriture

Dans un outil comme CiME, les équations et les règles de réécriture sont quantifiées implicitement, dans un ordre qui importe peu, et de nouvelles variables peuvent être créées à volonté. Dans le calcul de séquents de notre principe de réflexion, en revanche, chaque variable doit provenir du contexte. Le principal problème à résoudre ici sera donc de réussir à rapprocher ces deux visions du même problème.

DÉFINITION 4.13 (CONTEXTE ADAPTÉ)
Soient $R_1, \ldots, R_n$ des règles, supposons que CiME nous donne une trace de réécriture entre s_1 et t comme suit :

$$s_1 \to^{\sigma_1,p_1}_{\pm R_1} s_2 \to^{\sigma_2,p_2}_{\pm R_2} \cdots \to^{\sigma_n,p_n}_{\pm R_n} s_{n+1} \;=\; t \tag{4.3}$$

Ξ, Γ sont des contextes adaptés *pour cette trace si Ξ contient une représentation de $y_1, \ldots, y_m$, les variables libres de s_1 et t et une représentation de* `dummy`*, et Γ contient les hypothèses (closes) $\dot{R}_1, \ldots, \dot{R}_n$.*

Dans de tels contextes Ξ, Γ, la réification d'un terme (ouvert) t est définie de la façon suivante :
- si $t = (f\ a_1 \ldots a_p)$ alors $\dot{t} = (\dot{f}\ \dot{a}_1 \ldots \dot{a}_p)$
- si $t = y_i$ alors $\dot{t}$ est la variable correspondante dans Ξ.
- si t est une variable autre, alors $\dot{t} = \dot{\texttt{dummy}}$.

Ce genre de variable inconnue peut apparaître quand nous voulons prouver que $f(a, a) = f(b, b)$ à l'aide de l'hypothèse $\forall xy.f(x, x) = y$.

LEMME 4.14
Si Ξ, Γ sont adaptés à la trace 4.3, alors il existe une trace de preuve pour $\Xi, \Gamma \vdash \dot{s}_1 \doteq \dot{t}$.

Démonstration : Nous prouvons par récurrence descendante sur i que pour tous contextes adaptés Ξ, Γ, il existe une trace de preuve π telle que

$$\pi_i : \Xi, \Gamma \vdash \dot{s}_i \doteq \dot{t}$$

- Si $i = n + 1$ alors $s_i = t$, donc on a $=_I : \Gamma \vdash (t \doteq t)$ pour tous Ξ, Γ adaptés (t est bien formé).
- Sinon, soient Ξ, Γ des contextes adaptés de longueur k_1 et k_2 et h_i l'indice de la règle $\dot{R}_i$ dans Γ, soient $z_1, \ldots, z_l$ les variables libres de R_i. Nous construisons le terme de preuve suivant :

$$\dfrac{\begin{array}{c}\text{Hypothèse de récurrence avec } \Gamma; h+1 : R_i\{z_1 \dot{\mapsto} z_1\sigma_i\} \ldots; h+l : \dot{R_i}\sigma_i \\ \vdots \\ \pi : \Xi, \Gamma; \ldots; h+l : \dot{R_i}\sigma_i \vdash \dot{s_{i+1}} \doteq \dot{t}\end{array}}{\begin{array}{c}(=_{E_1}\ (h+l) \leftrightarrow \dot{p_i}\,\pi) : \Xi, \Gamma; \ldots; h+l : \dot{R_i}\sigma_i \vdash \dot{s}_i \doteq \dot{t} \\ \vdots\ l \text{ étapes } \forall_E \\ (\forall_E\, k\, \dot{z_1}\sigma_i(\forall_E\,(h+1)\,\dot{z_2}\sigma_i \ldots (\forall_E\,(h+l-1)\,\dot{z_l}\sigma_i(=_{E_1}\ (h+l) \leftrightarrow p_i\,\pi))\ldots)) : \\ \Xi, \Gamma \dot{s}_i \doteq \dot{t}\end{array}}$$

L'étape d'induction est acceptable car une extension d'un contexte adapté est adaptée. □

Conjectures closes et paires critiques

THÉORÈME 4.15
Soit G une conjecture close, $O_1, \ldots, O_n$ des hypothèses initiales et soit $C_1, \ldots, C_m$ la liste des paires critiques utiles ordonnées selon $\prec$. Soit $\Xi = \emptyset; \dot{\texttt{dummy}}$, soit Γ_k le contexte $\emptyset; \dot{O}_1; \ldots; \dot{O}_n; \dot{C}_1; \ldots; \dot{C}_k$ avec $0 \leq k \leq m$. Il y a une trace de preuve π pour $\Xi, \Gamma_0 \vdash \dot{G}$.

Démonstration : Nous construisons par récurrence une trace de preuve π pour le séquent $\Xi, \Gamma_i \vdash \dot{G}$, en allant de $i = m$ à $i = 0$.

- On peut construire une trace de preuve π pour $\Xi, \Gamma_m \vdash \dot{G}$ en utilisant le Lemme 4.14 et la trace donnée par CiME pour G.
- Si l'on a une trace π telle que π:$\Xi, \Gamma_i \vdash \dot{G}$, et si $C_i = \forall x_1 \ldots x_p.s = t$, nous construisons l'arbre de dérivation suivant pour avoir une preuve de $\Xi, \Gamma_{i-1} \vdash G$:

$$\dfrac{\begin{array}{cc}\begin{array}{c}\text{Lemme 4.14} \\ \vdots \\ \pi' : \Xi; \dot{x}_1; \ldots; \dot{x}_p, \Gamma_{i-1} \vdash \dot{s} \doteq \dot{t} \\ \vdots\ p \text{ étapes } \forall_I \\ (\forall_I\ \ldots (\forall_I\ \pi') \ldots) : \Xi, \Gamma_{i-1} \vdash \dot{\forall}_1 \ldots \dot{\forall}_1 \dot{s} \doteq \dot{t}\end{array} & \begin{array}{c}\text{Hypothèse de récurrence} \\ \vdots \\ \pi : \Xi, \Gamma_i \vdash \dot{G}\end{array}\end{array}}{(\text{Cut}\ \dot{C}_i\ (\forall_I\ \ldots (\forall_I\ \pi') \ldots)\ \pi) : \Xi, \Gamma_{i-1} \vdash \dot{G}}$$

□

Conjectures existentielles et spécialisation

Conjectures existentielles Quand le but est de la forme $\exists x_1, \ldots, x_n.s = t$, CiME donne une substitution σ et une trace de preuve pour $s\sigma = t\sigma$. Grâce au Lemme 4.14, nous pouvons construire une trace π pour $\dot{s\sigma} \doteq \dot{t\sigma}$. La trace pour le but quantifié est alors : $(\exists_I\ \dot{x_1\sigma} \ldots (\exists_I\ \dot{x_n\sigma}\ \pi) \ldots)$.

Spécialisations Si le but courant est une égalité $\exists x_1, \ldots, x_n.s = t$ et que CiME donne une trace de preuve avec une spécialisation $N = \exists y_1, \ldots, y_m.s' = t'$, il faut commencer par effectuer une coupure sur $\dot{N}$. Dans la première prémisse, le but devient $\dot{N}$ et nous pouvons alors construire la trace pour tous les lemmes suivants. Dans la seconde prémisse, nous appliquons la règle $\exists_E$ m fois pour obtenir les hypothèses $\dot{y}_1, \ldots, \dot{y}_m, \dot{s'} \doteq \dot{t'}$, et alors nous nous retrouvons dans la situation décrite au paragraphe précédent.

4.5.5 Banc d'essai

CiME et Coq coopèrent avec succès sur 230 problèmes de la bibliothèque [SS98] — *Thousands of Problems for Theorem Proving* — qui sert de référence dans la communauté des démonstrateurs automatiques. Ces problèmes sont une partie des 778 problèmes du mot et d'unifiabilité que l'on peut trouver dans TPTP. Certains de ces 778 problèmes sont éliminés car non-résolus par CiME dans le temps imparti (298), d'autres car il donnent une réponse négative (11) et enfin les (239) derniers parce qu'ils utilisent des symboles associatifs-commutatifs, que nous ne savons pas encore traiter.

Les expériences ont été faites grâce à un ordinateur PC avec un microprocesseur Pentium cadencé à 1.8GHz et 1Gigaoctet de mémoire vive. Chaque problème disposait de 600 secondes de calcul. Pour chaque complétion réussie, quatre preuves ont été automatiquement engendrées : deux preuves courtes (avec coupure) dont l'une est réflexive et l'autre par tactiques, et deux preuves longues (sans coupure) de même. Nous avons utilisé les versions CVS courantes de Coq et de CiME3 avec la machine virtuelle de réduction compilée [GL02] dans le cas de Coq.

Coq a tenté de vérifier chacune des 4×230 preuves engendrées, avec toujours une limite de temps à 600 secondes. Nous avons observé que les preuves courtes sont toujours vérifiées en moins d'une seconde, qu'elles soient réflexives ou par tactiques et ce même si la complétion avait duré longtemps. Les preuves réflexives courtes semblent plus rapides que les preuves courtes par tactiques mais sur des durées aussi brèves, cela est peu significatif. Pour les preuves expansées, les preuves réifiées prennent entre 1 et 30 fois moins de temps que les tactiques. Il y a même un exemple (`GRP614-1`) dont la preuve réifiée sans coupure est vérifiée en 2 secondes et la preuve par tactique ne réussit pas par manque de ressources. Certaines des preuves sans coupure sont trop énormes (plusieurs millions de lignes) pour que Coq puisse les traiter (14 preuves par réflexion et 16 par tactiques). Les détails des tests effectués sont disponibles dans l'annexe C.2.

4.6 Extensions

4.6.1 Preuves incomplètes

Lorsque l'on fait une preuve dans un système comme Coq, le fragment du premier ordre peut permettre de simplifier le but à prouver mais nécessiter d'autres types d'inférences pour réussir à prouver certains sous-buts. On peut alors vouloir fournir au cadre réflexif des preuves incomplètes. Une méthode que nous proposons pour y arriver est l'ajout du constructeur suivant dans le type `proof` :

```
Meta: ∀ Ξ Γ G, ⟦Ξ, Γ ⊢ G⟧ → proof.
```

L'inconvénient de ceci est que l'ajout de ce constructeur va entraîner une dépendance du type `proof` envers la signature (`Denv`, `Tenv` et `Penv`). Il faudra de plus s'assurer que le séquent $\Xi, \Gamma \vdash$ `G` correspond bien à celui que l'on veut prouver.

EXEMPLE 4.16
Nous allons illustrer l'utilisation de `Meta` *par un exemple :*

```
Lemma not_true_and_false : ∀ b, ¬(b=true ∧ b=false).
```

La preuve réflexive va créer l'hypothèse `true` $=$ `false` *et l'introduire dans le nouveau but. D'abord, nous réifions la signature :*

```
pose (Denv := (empty; bool)).
pose (Tenv := (empty; {true∈1}; {false∈1})).
pose (goal := ∀̇₁ ((DB 0)≐₁(App 1 ()) ∧̇ (DB 0)≐₁(App 2 ())) ⇀̇ ⊥̇).
```

Ensuite nous définissons une preuve incomplète à l'aide d'une fonction :

```
pose (prf :=λ H.
  ((∀_I (→_I (∧_E 1
    (=_E2 3 2 → (△ ≙ ⊖)
     (Meta (empty;1)
      (empty; (Var 1)≐₁(App 1 ()) ∧̇ (Var 1)≐₁(App 2 ());
       (Var 1)≐₁(App 1 ()); (Var 1)≐₁(App 2 ()))
      ⊥̇ H) : proof Denv Tenv empty)).
```

Enfin nous utilisons la tactique `refine` *pour changer notre sous-but.*

```
refine ((fun H =>
           Reflect Denv Tenv empty (prf H) empty empty goal) _).
```

Le but obtenu n'est pas très lisible, mais après une application de la tactique de réduction `compute`*, nous obtenons :*

```
∀ var : bool, var = true ∧ var = false →
var = true → var = false → true = false → ⊥
```

Nous avons donc obtenu un but simplifié que nous pouvons maintenant prouver par l'utilisation de la tactique `congruence` *(voir chapitre 2).*

4.6.2 Modularité du développement

En vue de la conception de variantes à notre système, il est possible de séparer dans les preuves et les fonctions définies la partie qui concerne les termes et signatures de termes, de toute la mécanique concernant les formules et les preuves. On obtient ainsi un foncteur qui, à partir d'une définition des termes et signatures du premier ordre avec la fonction d'interprétation et la preuve de ses propriétés, construit un type de formules et un principe de réflexion sur cette nouvelle algèbre de termes. On peut ainsi construire l'algèbre vide qui ne contient aucun terme, l'application du foncteur nous donne alors la logique propositionnelle avec des quantificateurs inutiles.

4.7 Conclusion

Nous avons décrit comment utiliser la réflexion calculatoire pour effectuer des preuves dans le système Coq. L'expérience montre que la réflexion, au-delà de son intérêt théorique, peut devenir une solution efficace en pratique. Comme le montre ce travail, la réflexion est un outil utile pour construire une couche d'interprétation entre Coq et d'autres formalismes logiques.

La première chose que nous souhaiterions faire pour étendre ce travail est la gestion des termes modulo associativité commutativité dans nos preuves. Ce type d'algèbre de termes rentre parfaitement dans le cadre de l'approche modulaire décrite précédemment, aussi espérons-nous arriver rapidement à des résultats dans ce domaine. Ainsi nous pourrions effectuer des preuves utilisant des symboles associatifs-commutatifs sans avoir à expliciter les transformations associatives-commutatives dans les traces de preuve.

Nous souhaiterions aussi pouvoir interpréter des traces engendrées pour des calculs de séquents multi-successeurs (en logique classique).

Conclusion

Nous avons proposé des méthodes qui constituent des avancées significatives vers notre objectif initial : la combinaison de l'approche interactive et de l'approche automatique pour la preuve formelle. D'une part, nous avons montré que l'approche basée sur les calculs de séquents sans contraction peut s'intégrer au formalisme logique du Calcul des Constructions Inductives. D'autre part, nous avons mis en place une méthodologie de preuve réflexive au premier ordre qui offre enfin la possibilité d'obtenir de façon sûre (selon l'approche sceptique) des preuves complexes d'énoncés du premier ordre qui sont obtenues entièrement automatiquement et re-vérifiées efficacement.

Le travail effectué dans le cadre de l'outil CiME montre que la production d'une trace de preuve est relativement simple lorsque les bases théoriques sur lesquelles repose le fonctionnement de l'outil externe sont bien comprises. Nous pensons ainsi que le développement de prouveurs automatiques devrait s'accompagner d'un réel effort d'instrumentation des programmes de preuve automatique afin qu'ils puissent justifier leurs réponses. Nous avons montré qu'il est possible d'ajouter, grâce à la réflexion, des couches d'interprétation permettant de lire des traces de preuve dans des formats plus variés que les tactiques Coq ou les termes de preuves. Notre travail fournit ainsi un réel langage intermédiaire de preuve, dont la sémantique est plus simple que celle du Calcul des Constructions Inductives, et qui est ainsi destiné à des utilisateurs ayant des connaissances en logique mais pas en Théorie des Types.

Outre l'élargissement du champ d'action de notre principe de réflexion, il faudra penser à essayer de rendre le langage plus souple (moins explicite), ce qui permettrait d'être moins exigeant envers l'outil qui fournit la trace. En revanche, il faudrait alors ajouter dans le mécanisme de réflexion des fonctions destinées à pallier le manque de précision de la trace de preuve. Ceci est à l'étude pour une extension aux algèbres de termes avec symboles associatifs-commutatifs s'appuyant sur [Con04].

À court terme, une nouvelle tactique Coq sera disponible ; elle extraira le contenu du premier ordre du but courant et des hypothèses, appellera une procédure automatique de recherche de preuve qui en cas de succès retournera une trace interprétable par réflexion. À l'avenir, une telle tactique soulèvera de nouveaux problèmes ; on peut d'ores et déjà penser que l'on aura besoin de gérer les cas d'échec. Quelquefois, les prouveurs automatiques proposent des formes de contre-exemples, auquel cas se pose le problème de réinterpréter ces contre-exemples dans la syntaxe de Coq. Par ailleurs, la procédure externe peut échouer à cause de son incomplétude. On aimerait qu'une telle procédure, loin de se comporter comme

une boîte noire, renvoie à l'utilisateur une forme simplifiée du problème posé, contenant seulement les parties du problème non-résolues automatiquement.

De manière générale, il reste encore à identifier les méthodes appropriées d'interaction entre assistants de preuves et procédures de recherche automatique de preuves.

Annexe A

Preuves d'inversibilité

A.1 Inversibilité des règles pour $LJTI$

Dans cette section se trouvent les preuves d'admissibilité forte (voir définition 1.14) des règles énoncées dans le lemme 3.9.

A.1.1 Inversibilité de la règle $R\rightarrow$

Nous allons montrer que la règle 3.1 ci-dessous est fortement admissible dans $LJTI$.

$$\frac{\Gamma \vdash A \rightarrow B}{\Gamma, A \vdash B} \quad (3.1)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

- Ax : comme $A \rightarrow B$ n'est pas atomique, la dernière règle n'est pas un axiome.
- $R\rightarrow$: La preuve est de la forme :

$$\frac{\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}\, R\rightarrow}{\Gamma, A \vdash B}\ \text{règle 3.1}}{} \quad \Longrightarrow \quad \Gamma, A \vdash B$$

  On peut donc enlever ces deux étapes de la dérivation.
- $La\rightarrow$:

$$\frac{\dfrac{\dfrac{\Gamma, P, C \vdash A \rightarrow B}{\Gamma, P, P \rightarrow C \vdash A \rightarrow B}\, La\rightarrow}{\Gamma, P, P \rightarrow C, A \vdash B}\ \text{règle 3.1}}{} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, P, C \vdash A \rightarrow B}{\Gamma, P, C, A \vdash B}\, Ind}{\Gamma, P, P \rightarrow C, A \vdash B}\, La\rightarrow$$

– $L{\to}{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, F{\to}C, E \vdash F \quad \Gamma, C \vdash A{\to}B}{\Gamma, (E{\to}F){\to}C \vdash A{\to}B}\, L{\to}{\to}}{\Gamma, (E{\to}F){\to}C, A \vdash B}\ \text{règle 3.1}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, F{\to}C, E \vdash F}{\Gamma, F{\to}C, E, A \vdash F}\, W \quad \dfrac{\Gamma, C \vdash A{\to}B}{\Gamma, C, A \vdash B}\, Ind}{\Gamma, (E{\to}F){\to}C, A \vdash B}\, L{\to}{\to}$$

Puisque la règle d'affaiblissement est fortement admissible, la hauteur de la dérivation de la première prémisse n'augmente pas.

– $R\forall$: Impossible puisque la conclusion du séquent est une implication.

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\} \vdash A{\to}B}{\Gamma, \forall x.C \vdash A{\to}B}\, L\forall}{\Gamma, \forall x.C, A \vdash B}\ \text{règle 3.1} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\} \vdash A{\to}B}{\Gamma, \forall x.C, C\{t/x\}, A \vdash B}\, Ind}{\Gamma, \forall x.C, A \vdash B}\, L\forall$$

– $L\forall{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.D){\to}C \vdash \forall x.D \quad \Gamma, C \vdash A{\to}B}{\Gamma, (\forall x.D){\to}C \vdash A{\to}B}\, L\forall{\to}}{\Gamma, (\forall x.D){\to}C, A \vdash B}\ \text{règle 3.1}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, (\forall x.D){\to}C \vdash \forall x.D}{\Gamma, (\forall x.D){\to}C, A \vdash \forall x.D}\, W \quad \dfrac{\Gamma, C \vdash A{\to}B}{\Gamma, C, A \vdash B}\, Ind}{\Gamma, (\forall x.D){\to}C, A \vdash B}\, L{\to}{\to}$$

– RI_i : Impossible puisque la conclusion du séquent est une implication.

– LI :

$$\dfrac{\dfrac{\dfrac{\ldots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash A{\to}B \quad \ldots}{\Gamma, I(\mathbf{p}) \vdash A{\to}B}\, LI}{\Gamma, I(\mathbf{p}), A \vdash B}\ \text{règle 3.1} \quad \Longrightarrow \quad \dfrac{\ldots \quad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash A{\to}B}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), A \vdash B}\, Ind \quad \ldots}{\Gamma, I(\mathbf{p}), A \vdash B}\, LI$$

– $LI\rightarrow$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash A\rightarrow B}{\Gamma, I(\mathbf{p})\rightarrow C \vdash A\rightarrow B}\, LI\rightarrow}{\Gamma, I(\mathbf{p})\rightarrow C, A \vdash B}\ \text{règle 3.1}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i \vdash A\rightarrow B}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow C\}_i, A \vdash B}\, Ind}{\Gamma, I(\mathbf{p})\rightarrow C, A \vdash B}\, LI\rightarrow$$

Ce qui clôt la démonstration par récurrence. □

A.1.2 Inversibilité de la règle $La\rightarrow$

Nous allons montrer que la règle 3.2 ci-dessous est fortement admissible dans $LJTI$.

$$\dfrac{\Gamma, P\rightarrow B \vdash G}{\Gamma, B \vdash G}\ (3.2)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– Ax :

$$\dfrac{\dfrac{}{\Gamma, P\rightarrow B, Q \vdash Q}\, Ax}{\Gamma, B, Q \vdash Q}\ \text{règle 3.2} \quad \Longrightarrow \quad \dfrac{}{\Gamma, B, Q \vdash Q}\, Ax$$

– $R\rightarrow$: La preuve est de la forme :

$$\dfrac{\dfrac{\dfrac{\Gamma, P\rightarrow B, C \vdash D}{\Gamma, P\rightarrow B \vdash C\rightarrow D}\, R\rightarrow}{\Gamma, B \vdash C\rightarrow D}\ \text{règle 3.2} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P\rightarrow B, C \vdash D}{\Gamma, B, C \vdash D}\, Ind}{\Gamma, B \vdash C\rightarrow D}\, R\rightarrow$$

– $La\rightarrow$, deux cas sont possibles :

1. $P\rightarrow B$ est la formule principale :

$$\dfrac{\dfrac{\dfrac{\Gamma, P, B \vdash G}{\Gamma, P, P\rightarrow B \vdash G}\, La\rightarrow}{\Gamma, P, B \vdash G}\ \text{règle 3.2} \quad \Longrightarrow \quad \Gamma, P, B \vdash G$$

2. $P\rightarrow B$ n'est pas la formule principale :

$$\dfrac{\dfrac{\dfrac{\Gamma, Q, C, P\rightarrow B \vdash G}{\Gamma, Q, Q\rightarrow C, P\rightarrow B \vdash G}\, La\rightarrow}{\Gamma, Q, Q\rightarrow C, B \vdash G}\ \text{règle 3.2} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, Q, C, P\rightarrow B \vdash G}{\Gamma, Q, C, B \vdash G}\, Ind}{\Gamma, Q, Q\rightarrow C, B \vdash G}\, La\rightarrow$$

– $L{\to}{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, F{\to}C, E, P{\to}B \vdash F \quad \Gamma, C, P{\to}B \vdash G}{\Gamma, (E{\to}F){\to}C, P{\to}B \vdash G}\, L{\to}{\to}}{\Gamma, (E{\to}F){\to}C, B \vdash G}\ \text{règle 3.2}}{}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, F{\to}C, E, P{\to}B \vdash F}{\Gamma, F{\to}C, E, B \vdash F}\, Ind \quad \dfrac{\Gamma, C, P{\to}B \vdash G}{\Gamma, C, B \vdash G}\, Ind}{\Gamma, (E{\to}F){\to}C, B \vdash G}\, L{\to}{\to}}{}$$

– $R\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, P{\to}B \vdash C}{\Gamma, P{\to}B \vdash \forall x.C}\, R\forall}{\Gamma, B \vdash \forall x.C}\ \text{règle 3.2} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P{\to}B \vdash C}{\Gamma, B \vdash C}\, Ind}{\Gamma, B \vdash \forall x.C}\, L\forall}{}$$

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, P{\to}B \vdash G}{\Gamma, \forall x.C, P{\to}B \vdash G}\, L\forall}{\Gamma, \forall x.C, B \vdash G}\ \text{règle 3.2} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, P{\to}B \vdash G}{\Gamma, \forall x.C, C\{t/x\}, B \vdash G}\, Ind}{\Gamma, \forall x.C, B \vdash G}\, L\forall}{}$$

– $L\forall{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.D){\to}C, P{\to}B \vdash \forall x.D \quad \Gamma, C, P{\to}B \vdash G}{\Gamma, (\forall x.D){\to}C, P{\to}B \vdash G}\, L\forall{\to}}{\Gamma, (\forall x.D){\to}C, B \vdash G}\ \text{règle 3.2}}{}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.D){\to}C, P{\to}B \vdash \forall x.D}{\Gamma, (\forall x.D){\to}C, B \vdash \forall x.D}\, Ind \quad \dfrac{\Gamma, C, P{\to}B \vdash G}{\Gamma, C, B \vdash G}\, Ind}{\Gamma, (\forall x.D){\to}C, B \vdash G}\, L{\to}{\to}}{}$$

– RI_i :

$$\dfrac{\dfrac{\dfrac{\ldots \quad \Gamma, P{\to}B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma, P{\to}B \vdash I(\mathbf{p})}\, RI_i}{\Gamma, B \vdash I(\mathbf{p})}\ \text{règle 3.2}}{}$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, P{\to}B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind \quad \ldots}{\Gamma, B \vdash I(\mathbf{p})}\, RI_i$$

– LI :

$$\dfrac{\dfrac{\dots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), P{\to}B \vdash G \quad \dots}{\Gamma, I(\mathbf{p}), P{\to}B \vdash G}\, LI}{\Gamma, I(\mathbf{p}), B \vdash G}\ \text{règle 3.2}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), P{\to}B \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), B \vdash G}\, Ind \quad \dots}{\Gamma, I(\mathbf{p}), B \vdash G}\, LI$$

– $LI{\to}$:

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}C\}_i, P{\to}B \vdash G}{\Gamma, I(\mathbf{p}){\to}C, P{\to}B \vdash G}\, LI{\to}}{\Gamma, I(\mathbf{p}){\to}C, B \vdash G}\ \text{règle 3.2}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}C\}_i, P{\to}B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}C\}_i, B \vdash G}\, Ind}{\Gamma, I(\mathbf{p}){\to}C, B \vdash G}\, LI{\to}$$

Ce qui clôt la démonstration par récurrence. □

A.1.3 Inversibilité partielle de la règle $L{\to}{\to}$

Nous allons montrer que la règle 3.3 ci-dessous est fortement admissible dans $LJTI$.

$$\dfrac{\Gamma, (C{\to}D){\to}B \vdash G}{\Gamma, B \vdash G}\ (3.3)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– Ax :

$$\dfrac{\dfrac{}{\Gamma, (C{\to}D){\to}B, P \vdash P}\, Ax}{\Gamma, B, P \vdash P}\ \text{règle 3.3} \quad \Longrightarrow \quad \dfrac{}{\Gamma, B, P \vdash P}\, Ax$$

– $R{\to}$: La preuve est de la forme :

$$\dfrac{\dfrac{\dfrac{\Gamma, (C{\to}D){\to}B, E \vdash F}{\Gamma, (C{\to}D){\to}B \vdash E{\to}F}\, R{\to}}{\Gamma, B \vdash E{\to}F}\ \text{règle 3.3} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, (C{\to}D){\to}B, E \vdash F}{\Gamma, B, E \vdash F}\, Ind}{\Gamma, B \vdash E{\to}F}\, R{\to}$$

– $La{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, P, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, P, P{\rightarrow}A, (C{\rightarrow}D){\rightarrow}B \vdash G}\, La{\rightarrow}}{\Gamma, P, P{\rightarrow}A, B \vdash G}\ \text{règle 3.3} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, P, A, B \vdash G}\, Ind}{\Gamma, P, P{\rightarrow}A, B \vdash G}\, La{\rightarrow}$$

– $L{\rightarrow}{\rightarrow}$, deux cas sont possibles :

1. $(C{\rightarrow}D){\rightarrow}B$ est la formule principale :

$$\dfrac{\dfrac{\dfrac{\Gamma, D{\rightarrow}B, C \vdash D \quad \Gamma, B \vdash G}{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash G}\, L{\rightarrow}{\rightarrow}}{\Gamma, B \vdash G}\ \text{règle 3.3} \quad \Longrightarrow \quad \Gamma, B \vdash G$$

2. $(C{\rightarrow}D){\rightarrow}B$ n'est pas la formule principale :

$$\dfrac{\dfrac{\dfrac{\Gamma, F{\rightarrow}A, E, (C{\rightarrow}D){\rightarrow}B \vdash F \quad \Gamma, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, (E{\rightarrow}F){\rightarrow}A, (C{\rightarrow}D){\rightarrow}B \vdash G}\, L{\rightarrow}{\rightarrow}}{\Gamma, (E{\rightarrow}F){\rightarrow}A, B \vdash G}\ \text{règle 3.3}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, F{\rightarrow}A, E, (C{\rightarrow}D){\rightarrow}B \vdash F}{\Gamma, F{\rightarrow}A, E, B \vdash F}\, Ind \quad \dfrac{\Gamma, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, A, B \vdash G}\, Ind}{\Gamma, (E{\rightarrow}F){\rightarrow}A, B \vdash G}\, L{\rightarrow}{\rightarrow}$$

– $R\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash C}{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash \forall x.C}\, R\forall}{\Gamma, B \vdash \forall x.C}\ \text{règle 3.3} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, (C{\rightarrow}D){\rightarrow}B \vdash C}{\Gamma, B \vdash C}\, Ind}{\Gamma, B \vdash \forall x.C}\, L\forall$$

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall x.A, A\{t/x\}, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, \forall x.A, (C{\rightarrow}D){\rightarrow}B \vdash G}\, L\forall}{\Gamma, \forall x.A, B \vdash G}\ \text{règle 3.3}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \forall x.A, A\{t/x\}, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, \forall x.A, A\{t/x\}, B \vdash G}\, Ind}{\Gamma, \forall x.A, B \vdash G}\, L\forall$$

– $L\forall\rightarrow$:

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow E, (C\rightarrow D)\rightarrow B \vdash \forall x.A \quad \Gamma, E, (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, (\forall x.A)\rightarrow E, (C\rightarrow D)\rightarrow B \vdash G}\, L\forall\rightarrow}{\dfrac{\Gamma, (\forall x.A)\rightarrow E, (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, (\forall x.A)\rightarrow E, B \vdash G}\ \text{règle 3.3}}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow E, (C\rightarrow D)\rightarrow B \vdash \forall x.A}{\Gamma, (\forall x.A)\rightarrow E, B \vdash \forall x.A}\, Ind \qquad \dfrac{\Gamma, E, (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, E, B \vdash G}\, Ind}{\Gamma, (\forall x.A)\rightarrow E, B \vdash G}\, L\rightarrow\rightarrow$$

– RI_i :

$$\dfrac{\dfrac{\ldots \quad \Gamma, (C\rightarrow D)\rightarrow B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma, (C\rightarrow D)\rightarrow B \vdash I(\mathbf{p})}\, RI_i}{\Gamma, B \vdash I(\mathbf{p})}\ \text{règle 3.3}$$

$$\Downarrow$$

$$\dfrac{\ldots \qquad \dfrac{\Gamma, (C\rightarrow D)\rightarrow B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind \qquad \ldots}{\Gamma, B \vdash I(\mathbf{p})}\, RI_i$$

– LI :

$$\dfrac{\dfrac{\ldots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), (C\rightarrow D)\rightarrow B \vdash G \quad \ldots}{\Gamma, I(\mathbf{p}), (C\rightarrow D)\rightarrow B \vdash G}\, LI}{\Gamma, I(\mathbf{p}), B \vdash G}\ \text{règle 3.3}$$

$$\Downarrow$$

$$\dfrac{\ldots \qquad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), B \vdash G}\, Ind \qquad \ldots}{\Gamma, I(\mathbf{p}), B \vdash G}\, LI$$

– $LI\rightarrow$:

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow A\}_i, (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, I(\mathbf{p})\rightarrow A, (C\rightarrow D)\rightarrow B \vdash G}\, LI\rightarrow}{\Gamma, I(\mathbf{p})\rightarrow A, B \vdash G}\ \text{règle 3.3}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow A\}_i, (C\rightarrow D)\rightarrow B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow A\}_i, B \vdash G}\, Ind}{\Gamma, I(\mathbf{p})\rightarrow A, B \vdash G}\, LI\rightarrow$$

Ce qui clôt la démonstration par récurrence. □

A.1.4 Inversibilité partielle de la règle $L\forall\rightarrow$

Nous allons montrer que la règle 3.4 ci-dessous est fortement admissible dans $LJTI$.

$$\frac{\Gamma, (\forall x.A)\rightarrow B \vdash G}{\Gamma, B \vdash G} \ (3.4)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

- Ax :

$$\dfrac{\dfrac{}{\Gamma, (\forall x.A)\rightarrow B, P \vdash P}\, Ax}{\Gamma, B, P \vdash P}\ \text{règle 3.4} \quad \Longrightarrow \quad \frac{}{\Gamma, B, P \vdash P}\, Ax$$

- $R\rightarrow$: La preuve est de la forme :

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, C \vdash D}{\Gamma, (\forall x.A)\rightarrow B \vdash C\rightarrow D}\, R\rightarrow}{\Gamma, B \vdash C\rightarrow D}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, C \vdash D}{\Gamma, B, C \vdash D}\, Ind}{\Gamma, B \vdash C\rightarrow D}\, R\rightarrow$$

- $La\rightarrow$:

$$\dfrac{\dfrac{\Gamma, P, C, (\forall x.A)\rightarrow B \vdash G}{\Gamma, P, P\rightarrow C, (\forall x.A)\rightarrow B \vdash G}\, La\rightarrow}{\Gamma, P, P\rightarrow C, B \vdash G}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P, C, (\forall x.A)\rightarrow B \vdash G}{\Gamma, P, C, B \vdash G}\, Ind}{\Gamma, P, P\rightarrow C, B \vdash G}\, La\rightarrow$$

- $L\rightarrow\rightarrow$:

$$\dfrac{\dfrac{\Gamma, D\rightarrow E, C, (\forall x.A)\rightarrow B \vdash D \quad \Gamma, E, (\forall x.A)\rightarrow B \vdash G}{\Gamma, (C\rightarrow D)\rightarrow E, (\forall x.A)\rightarrow B \vdash G}\, L\rightarrow\rightarrow}{\Gamma, (C\rightarrow D)\rightarrow E, B \vdash G}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, D\rightarrow E, C, (\forall x.A)\rightarrow B \vdash D}{\Gamma, D\rightarrow E, C, B \vdash D}\, Ind \quad \dfrac{\Gamma, E, (\forall x.A)\rightarrow B \vdash G}{\Gamma, E, B \vdash G}\, Ind}{\Gamma, (C\rightarrow D)\rightarrow E, B \vdash G}\, L\rightarrow\rightarrow$$

- $R\forall$:

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B \vdash C}{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.C}\, R\forall}{\Gamma, B \vdash \forall x.C}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B \vdash C}{\Gamma, B \vdash C}\, Ind}{\Gamma, B \vdash \forall x.C}\, L\forall$$

– $L\forall$:

$$\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, (\forall x.A)\to B \vdash G}{\Gamma, \forall x.C, (\forall x.A)\to B \vdash G}\, L\forall}{\Gamma, \forall x.C, B \vdash G}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, (\forall x.A)\to B \vdash G}{\Gamma, \forall x.C, C\{t/x\}, B \vdash G}\, Ind}{\Gamma, \forall x.C, B \vdash G}\, L\forall$$

– $L\forall\to$, deux cas sont possibles :

1. $(\forall x.A)\to B$ est la formule principale :

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\to B \vdash \forall x.A \quad \Gamma, B \vdash G}{\Gamma, (\forall x.A)\to B \vdash G}\, L\forall\to}{\Gamma, B \vdash G}\ \text{règle 3.4} \quad \Longrightarrow \quad \Gamma, B \vdash G$$

2. $(\forall x.A)\to B$ n'est pas la formule principale :

$$\dfrac{\dfrac{\Gamma, (\forall x.C)\to E, (\forall x.A)\to B \vdash \forall x.C \quad \Gamma, E, (\forall x.A)\to B \vdash G}{\Gamma, (\forall x.C)\to E, (\forall x.A)\to B \vdash G}\, L\forall\to}{\Gamma, (\forall x.C)\to E, B \vdash G}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, (\forall x.C)\to E, (\forall x.A)\to B \vdash \forall x.C}{\Gamma, (\forall x.C)\to E, B \vdash \forall x.C}\, Ind \quad \dfrac{\Gamma, E, (\forall x.A)\to B \vdash G}{\Gamma, E, B \vdash G}\, Ind}{\Gamma, (\forall x.C)\to E, B \vdash G}\, L\forall\to$$

– RI_i :

$$\dfrac{\dfrac{\ldots \quad \Gamma, (\forall x.A)\to B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma, (\forall x.A)\to B \vdash I(\mathbf{p})}\, RI_i}{\Gamma, B \vdash I(\mathbf{p})}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, (\forall x.A)\to B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, B \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind \quad \ldots}{\Gamma, B \vdash I(\mathbf{p})}\, RI_i$$

– LI :

$$\dfrac{\dfrac{\dots \quad \Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{y_i}), (\forall x.A)\to B \vdash G \quad \dots}{\Gamma, I(\mathbf{p}), (\forall x.A)\to B \vdash G}\, LI}{\Gamma, I(\mathbf{p}), B \vdash G}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\dots \qquad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{y_i}), (\forall x.A)\to B \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{y_i}), B \vdash G}\, Ind \qquad \dots}{\Gamma, I(\mathbf{p}), B \vdash G}\, LI$$

– $LI\to$:

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to C\}_i, (\forall x.A)\to B \vdash G}{\Gamma, I(\mathbf{p})\to C, (\forall x.A)\to B \vdash G}\, LI\to}{\Gamma, I(\mathbf{p})\to C, B \vdash G}\ \text{règle 3.4}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to C\}_i, (\forall x.A)\to B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to C\}_i, B \vdash G}\, Ind}{\Gamma, I(\mathbf{p})\to C, B \vdash G}\, LI\to$$

Ce qui clôt la démonstration par récurrence. □

A.1.5 Inversibilité de la règle LI

Nous allons montrer que la règle 3.5 ci-dessous est fortement admissible dans $LJTI$.

$$\dfrac{\Gamma, I(\mathbf{p}) \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}) \vdash G}\ (3.5)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– Ax :

$$\dfrac{\dfrac{}{\Gamma, I(\mathbf{p}), P \vdash P}\, Ax}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}), P \vdash P}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}), P \vdash P}\, Ax$$

– $R\to$: La preuve est de la forme :

$$\dfrac{\dfrac{\dfrac{\Gamma, I(\mathbf{p}), C \vdash D}{\Gamma, I(\mathbf{p}) \vdash C\to D}\, R\to}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}) \vdash C\to D}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, I(\mathbf{p}), C \vdash D}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}), C \vdash D}\, Ind}{\Gamma, \mathbf{H_i}(\mathbf{p},\mathbf{t}) \vdash C\to D}\, R\to$$

– $La{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, P, C, I(\mathbf{p}) \vdash G}{\Gamma, P, P{\rightarrow}C, I(\mathbf{p}) \vdash G}\, La{\rightarrow}}{\Gamma, P, P{\rightarrow}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}}{} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P, C, I(\mathbf{p}) \vdash G}{\Gamma, P, C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, P, P{\rightarrow}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, La{\rightarrow}$$

– $L{\rightarrow}{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, D{\rightarrow}E, C, I(\mathbf{p}) \vdash D \quad \Gamma, E, I(\mathbf{p}) \vdash G}{\Gamma, (C{\rightarrow}D){\rightarrow}E, I(\mathbf{p}) \vdash G}\, L{\rightarrow}{\rightarrow}}{\Gamma, (C{\rightarrow}D){\rightarrow}E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}}{}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, D{\rightarrow}E, C, I(\mathbf{p}) \vdash D}{\Gamma, D{\rightarrow}E, C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash D}\, Ind \quad \dfrac{\Gamma, E, I(\mathbf{p}) \vdash G}{\Gamma, E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, (C{\rightarrow}D){\rightarrow}E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L{\rightarrow}{\rightarrow}$$

– $R\forall$, soit y une variable n'apparaissant pas libre dans Γ et $\mathbf{H_i}(\mathbf{p}, \mathbf{t})$:

$$\dfrac{\dfrac{\dfrac{\Gamma, I(\mathbf{p}) \vdash C}{\Gamma, I(\mathbf{p}) \vdash \forall x.C}\, R\forall}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash \forall x.C}\ \text{règle 3.5}}{} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\Gamma, I(\mathbf{p}) \vdash C}{\Gamma, I(\mathbf{p}) \vdash C\{y/x\}}\, Subst_{\{y/x\}}}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash C\{y/x\}}\, Ind}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash \forall y.C\{y/x\}}\, L\forall$$

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall x.C, C\{u/x\}, I(\mathbf{p}) \vdash G}{\Gamma, \forall x.C, I(\mathbf{p}) \vdash G}\, L\forall}{\Gamma, \forall x.C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}}{}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \forall x.C, C\{u/x\}, I(\mathbf{p}) \vdash G}{\Gamma, \forall x.C, C\{u/x\}, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, \forall x.C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L\forall$$

– $L\forall{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.C){\rightarrow}E, I(\mathbf{p}) \vdash \forall x.C \quad \Gamma, E, I(\mathbf{p}) \vdash G}{\Gamma, (\forall x.C){\rightarrow}E, I(\mathbf{p}) \vdash G}\, L\forall{\rightarrow}}{\Gamma, (\forall x.C){\rightarrow}E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}}{}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, (\forall x.C){\rightarrow}E, I(\mathbf{p}) \vdash \forall x.C}{\Gamma, (\forall x.C){\rightarrow}E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash \forall x.C}\, Ind \quad \dfrac{\Gamma, E, I(\mathbf{p}) \vdash G}{\Gamma, E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, (\forall x.C){\rightarrow}E, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L{\rightarrow}{\rightarrow}$$

– RI_i :

$$\dfrac{\dfrac{\dots \quad \Gamma, I(\mathbf{p}) \vdash H'_{j,k}(\mathbf{q}, \mathbf{u_i}) \quad \dots}{\Gamma, I(\mathbf{p}) \vdash I'(\mathbf{q})}\, RI'_j}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash I'(\mathbf{q})}\ \text{règle 3.5}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\Gamma, I(\mathbf{p}) \vdash H'_{j,k}(\mathbf{q}, \mathbf{u_j})}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash H'_{j,k}(\mathbf{q}, \mathbf{u_j})}\, Ind \quad \dots}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash I(\mathbf{q})}\, RI'_j$$

– LI, deux cas peuvent se présenter :

1. $I(\mathbf{p})$ est la formule principale :

$$\dfrac{\dfrac{\dots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G \quad \dots}{\Gamma, I(\mathbf{p}) \vdash G}\, LI}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Subst_{\{\mathbf{t}/\mathbf{y_i}\}}$$

2. $I(\mathbf{p})$ n'est pas la formule principale ; pour chaque constructeur $\mathcal{C}'_j$ de l'inductif I', on choisit des variables $\mathbf{z_j}$ n'apparaissant pas libres dans Γ, $\mathbf{q}$, $\mathbf{H_i}(\mathbf{p}, \mathbf{t})$ ni G :

$$\dfrac{\dfrac{\dots \quad \Gamma, \mathbf{H'_j}(\mathbf{q}, \mathbf{y_j}), I(\mathbf{p}) \vdash G \quad \dots}{\Gamma, I'(\mathbf{q}), I(\mathbf{p}) \vdash G}\, LI'}{\Gamma, I'(\mathbf{q}), \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\dfrac{\Gamma, \mathbf{H'_j}(\mathbf{q}, \mathbf{y_j}), I(\mathbf{p}) \vdash G}{\Gamma, \mathbf{H'_j}(\mathbf{q}, \mathbf{z_j}), I(\mathbf{p}) \vdash G}\, Subst_{\{\mathbf{z_j}/\mathbf{y_j}\}}}{\Gamma, \mathbf{H'_j}(\mathbf{q}, \mathbf{z_j}), \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind \quad \dots}{\Gamma, I'(\mathbf{q}), \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, LI'$$

– $LI{\rightarrow}$:

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_j}.\mathbf{H'_j}(\mathbf{q}, \mathbf{y_j}){\rightarrow}C\}_j, I(\mathbf{p}) \vdash G}{\Gamma, I'(\mathbf{q}){\rightarrow}C, I(\mathbf{p}) \vdash G}\, LI'{\rightarrow}}{\Gamma, I'(\mathbf{q}){\rightarrow}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_j}.\mathbf{H'_j}(\mathbf{q}, \mathbf{y_j}){\rightarrow}C\}_j, I(\mathbf{p}) \vdash G}{\Gamma, \{\forall \mathbf{y_j}.\mathbf{H'_j}(\mathbf{q}, \mathbf{y_j}){\rightarrow}C\}_j, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, I'(\mathbf{q}){\rightarrow}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, LI{\rightarrow}$$

Ce qui clôt la démonstration par récurrence. □

A.1.6 Inversibilité de la règle $LI{\rightarrow}$

Nous allons montrer que la règle 3.6 ci-dessous est fortement admissible dans $LJTI$.

$$\frac{\Gamma, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G} \ (3.6)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

- Ax :

$$\dfrac{\dfrac{}{\Gamma, I(\mathbf{p}){\rightarrow}B, P \vdash P}\ Ax}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i, P \vdash P}\ \text{règle 3.6} \quad \Longrightarrow \quad \dfrac{}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i, P \vdash P}\ Ax$$

- $R{\rightarrow}$: La preuve est de la forme :

$$\dfrac{\dfrac{\Gamma, I(\mathbf{p}){\rightarrow}B, C \vdash D}{\Gamma, I(\mathbf{p}){\rightarrow}B \vdash C{\rightarrow}D}\ R{\rightarrow}}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash C{\rightarrow}D}\ \text{règle 3.6} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, I(\mathbf{p}){\rightarrow}B, C \vdash D}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i, C \vdash D}\ Ind}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash C{\rightarrow}D}\ R{\rightarrow}$$

- $La{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, P, C, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, P, P{\rightarrow}C, I(\mathbf{p}){\rightarrow}B \vdash G}\ La{\rightarrow}}{\Gamma, P, P{\rightarrow}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ \text{règle 3.6}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, P, C, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, P, C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ Ind}{\Gamma, P, P{\rightarrow}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ La{\rightarrow}$$

- $L{\rightarrow}{\rightarrow}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, D{\rightarrow}E, C, I(\mathbf{p}){\rightarrow}B \vdash D \quad \Gamma, E, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, (C{\rightarrow}D){\rightarrow}E, I(\mathbf{p}){\rightarrow}B \vdash G}\ L{\rightarrow}{\rightarrow}}{\Gamma, (C{\rightarrow}D){\rightarrow}E, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ \text{règle 3.6}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, D{\rightarrow}E, C, I(\mathbf{p}){\rightarrow}B \vdash D}{\Gamma, D{\rightarrow}E, C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash D}\ Ind \quad \dfrac{\Gamma, E, I(\mathbf{p}){\rightarrow}B \vdash G}{\Gamma, E, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ Ind}{\Gamma, (C{\rightarrow}D){\rightarrow}E, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\rightarrow}B\}_i \vdash G}\ L{\rightarrow}{\rightarrow}$$

– $R\forall$:

$$\dfrac{\dfrac{\Gamma, I(\mathbf{p})\rightarrow B \vdash C}{\Gamma, I(\mathbf{p})\rightarrow B \vdash \forall x.C}\, R\forall}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash \forall x.C}\ \text{règle 3.6} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, I(\mathbf{p})\rightarrow B \vdash C}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash C}\, Ind}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash \forall x.C}\, L\forall$$

La condition de variable fraîche est repectée car les variables libres de $\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B$ le sont aussi dans $I(\mathbf{p})\rightarrow B$.

– $L\forall$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, I(\mathbf{p})\rightarrow B \vdash G}{\Gamma, \forall x.C, I(\mathbf{p})\rightarrow B \vdash G}\, L\forall}{\Gamma, \forall x.C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \forall x.C, C\{t/x\}, I(\mathbf{p})\rightarrow B \vdash G}{\Gamma, \forall x.C, C\{t/x\}, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\, Ind}{\Gamma, \forall x.C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\, L\forall$$

– $L\forall\rightarrow$:

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow C, I(\mathbf{p})\rightarrow B \vdash \forall x.A \quad \Gamma, C, I(\mathbf{p})\rightarrow B \vdash G}{\Gamma, (\forall x.A)\rightarrow C, I(\mathbf{p})\rightarrow B \vdash G}\, L\forall\rightarrow}{\Gamma, (\forall x.A)\rightarrow C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow C, I(\mathbf{p})\rightarrow B \vdash \forall x.A}{\Gamma, (\forall x.A)\rightarrow C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash \forall x.A}\, Ind \quad \dfrac{\Gamma, C, I(\mathbf{p})\rightarrow B \vdash G}{\Gamma, C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\, Ind}{\Gamma, (\forall x.A)\rightarrow C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash G}\, L\forall\rightarrow$$

– RI_i :

$$\dfrac{\dfrac{\dots \quad \Gamma, I(\mathbf{p})\rightarrow B \vdash H'_{j,k}(\mathbf{q},\mathbf{t_j}) \quad \dots}{\Gamma, I(\mathbf{p})\rightarrow B \vdash I'(\mathbf{q})}\, RI'_i}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash I'(\mathbf{q})}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\Gamma, I(\mathbf{p})\rightarrow B \vdash H'_{j,k}(\mathbf{q},\mathbf{t_j})}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash H'_{j,k}(\mathbf{q},\mathbf{t_j})}\, Ind \quad \dots}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\rightarrow B\}_i \vdash I'(\mathbf{q})}\, RI'_i$$

– LI :

$$\dfrac{\dfrac{\ldots \quad \Gamma, \mathbf{H'_j}(\mathbf{q},\mathbf{z_j}), I(\mathbf{p})\to B \vdash G \quad \ldots}{\Gamma, I'(\mathbf{q}), I(\mathbf{p})\to B \vdash G}\, LI'}{\Gamma, I'(\mathbf{q}), \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, \mathbf{H'_j}(\mathbf{q},\mathbf{z_j}), I(\mathbf{p})\to B \vdash G}{\Gamma, \mathbf{H'_j}(\mathbf{q},\mathbf{z_j}), \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\, Ind \quad \ldots}{\Gamma, I'(\mathbf{q}), \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\, LI'$$

– $LI\to$:

1. $I(\mathbf{p})\to B$ est la formule principale :

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}{\Gamma, I(\mathbf{p})\to B \vdash G}\, LI\to}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\ \text{règle 3.6} \quad \Longrightarrow \quad \Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G$$

2. $I(\mathbf{p})\to B$ n'est pas la formule principale :

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{z_j}.\mathbf{H'_j}(\mathbf{q},\mathbf{z_j})\to C\}_j, I(\mathbf{p})\to B \vdash G}{\Gamma, I'(\mathbf{q})\to C, I(\mathbf{p})\to B \vdash G}\, LI'\to}{\Gamma, I'(\mathbf{q})\to C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, \{\forall \mathbf{z_j}.\mathbf{H'_j}(\mathbf{q},\mathbf{z_j})\to C\}_j, I(\mathbf{p})\to B \vdash G}{\Gamma, \{\forall \mathbf{z_j}.\mathbf{H'_j}(\mathbf{q},\mathbf{z_j})\to C\}_j, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\, Ind}{\Gamma, I'(\mathbf{q})\to C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i})\to B\}_i \vdash G}\, LI'\to$$

Ce qui clôt la démonstration par récurrence. □

A.2 Inversibilité des règles pour $LJTI_=$

A.2.1 Inversibilité dans $LJTI_=$ des règles de $LJTI$

Inversibilité de la règle $R\to$

Nous allons montrer que la règle 3.1 ci-dessous est fortement admissible dans $LJTI_=$.

$$\dfrac{\Gamma \vdash A\to B}{\Gamma, A \vdash B}\ (3.1)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– $R{=}$, $L{=}_1$: impossible à cause de la forme de $A{\to}B$.
– $L{=}_2$:

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C \vdash A{\to}B}{\Gamma, s{=}t, P{\to}C \vdash A{\to}B}\, L{=}_2}{\Gamma, s{=}t, P{\to}C, A \vdash B}\ \text{règle 3.1} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C \vdash A{\to}B}{\Gamma, s{=}t, P'{\to}C, A \vdash B}\, Ind}{\Gamma, s{=}t, P{\to}C, A \vdash B}\, L{=}_2$$

– $L{=}{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, C \vdash A{\to}B}{\Gamma, t{=}t{\to}C \vdash A{\to}B}\, L{=}{\to}}{\Gamma, t{=}t{\to}C, A \vdash B}\ \text{règle 3.1} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\Gamma, C \vdash A{\to}B}{\Gamma, C, A \vdash B}\, Ind}{\Gamma, t{=}t{\to}C, A \vdash B}\, L{=}{\to}$$

– Les autres cas sont traités comme pour $LJTI$.
Ceci clôt la démonstration par récurrence. □

Inversibilité de la règle $La{\to}$

Nous allons montrer que la règle 3.2 ci-dessous est fortement admissible dans $LJTI_{=}$.

$$\dfrac{\Gamma, P{\to}B \vdash G}{\Gamma, B \vdash G}\ (3.2)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.
– $R{=}$:

$$\dfrac{\dfrac{}{\Gamma, P{\to}B \vdash t{=}t}\, R{=}}{\Gamma, B \vdash t{=}t}\ \text{règle 3.2} \qquad \Longrightarrow \qquad \dfrac{}{\Gamma, B \vdash t{=}t}\, R{=}$$

– $L{=}_1$:

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P{\to}B \vdash G'}{\Gamma, s{=}t, P{\to}B \vdash G}\, L{=}_1}{\Gamma, s{=}t, B \vdash G}\ \text{règle 3.2} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\Gamma, s{=}t, P{\to}B \vdash G'}{\Gamma, s{=}t, B \vdash G'}\, Ind}{\Gamma, s{=}t, B \vdash G}\, L{=}_1$$

– $L{=}_2$, deux cas sont possibles :

1. La formule $P{\to}B$ est réécrite :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}B \vdash G}{\Gamma, s{=}t, P{\to}B \vdash G}\, L{=}_2}{\Gamma, s{=}t, B \vdash G}\ \text{règle 3.2} \qquad \Longrightarrow \qquad \dfrac{\Gamma, s{=}t, P'{\to}B \vdash G}{\Gamma, s{=}t, B \vdash G}\, Ind$$

2. Une autre formule, $Q{\to}C$, est réécrite :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, Q'{\to}C, P{\to}B \vdash G}{\Gamma, s{=}t, Q{\to}C, P{\to}B \vdash G}\, L{=}_2}{\Gamma, s{=}t, Q{\to}C, B \vdash G}\ \text{règle 3.2} \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, s{=}t, Q'{\to}C, P{\to}B \vdash G}{\Gamma, s{=}t, Q'{\to}C, B \vdash G}\, Ind}{\Gamma, s{=}t, Q{\to}C, B \vdash G}\, L{=}_2$$

– $L{=}{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, C, P{\to}B \vdash G}{\Gamma, t{=}t{\to}C, P{\to}B \vdash G}\, L{=}{\to}}{\Gamma, t{=}t{\to}C, B \vdash G}\ \text{règle 3.2} \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, C, P{\to}B \vdash G}{\Gamma, C, B \vdash G}\, Ind}{\Gamma, t{=}t{\to}C, B \vdash G}\, L{=}{\to}$$

– Les autres cas sont traités comme pour $LJTI$.

Ceci clôt la démonstration par récurrence. □

Inversibilité partielle de la règle $L{\to}{\to}$

Nous allons montrer que la règle 3.3 ci-dessous est fortement admissible dans $LJTI_=$.

$$\dfrac{\Gamma, (C{\to}D){\to}B \vdash G}{\Gamma, B \vdash G}\ (3.3)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– $R{=}$:

$$\dfrac{\dfrac{}{\Gamma, (C{\to}D){\to}B \vdash t{=}t}\, R{=}}{\Gamma, B \vdash t{=}t}\ \text{règle 3.3} \quad\Longrightarrow\quad \dfrac{}{\Gamma, B \vdash t{=}t}\, R{=}$$

– $L{=}_1$, G est une formule atomique ou une égalité :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, (C{\to}D){\to}B \vdash G'}{\Gamma, s{=}t, (C{\to}D){\to}B \vdash G}\, L{=}_1}{\Gamma, s{=}t, B \vdash G}\ \text{règle 3.3} \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, s{=}t, (C{\to}D){\to}B \vdash G'}{\Gamma, s{=}t, B \vdash G'}\, Ind}{\Gamma, s{=}t, B \vdash G}\, L{=}_1$$

– $L{=}_2$:

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}A, (C{\to}D){\to}B \vdash G}{\Gamma, s{=}t, P{\to}A, (C{\to}D){\to}B \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\to}A, B \vdash G}\ \text{règle 3.3} \quad\Longrightarrow\quad \dfrac{\dfrac{\Gamma, s{=}t, P'{\to}A, (C{\to}D){\to}B \vdash G}{\Gamma, s{=}t, P'{\to}A, B \vdash G}\, Ind}{\Gamma, s{=}t, P{\to}A, B \vdash G}\, L{=}_2$$

– $L{=}{\rightarrow}$:

$$\dfrac{\dfrac{\Gamma, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, t{=}t{\rightarrow}A, (C{\rightarrow}D){\rightarrow}B \vdash G}\, L{=}{\rightarrow}}{\Gamma, t{=}t{\rightarrow}A, B \vdash G}\ \text{règle 3.3} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, A, (C{\rightarrow}D){\rightarrow}B \vdash G}{\Gamma, A, B \vdash G}\, Ind}{\Gamma, t{=}t{\rightarrow}A, B \vdash G}\, L{=}{\rightarrow}$$

– Les autres cas sont traités comme pour $LJTI$.

Ceci clôt la démonstration par récurrence. □

Inversibilité partielle de la règle $L\forall{\rightarrow}$

Nous allons montrer que la règle 3.4 ci-dessous est fortement admissible dans $LJTI_{=}$.

$$\dfrac{\Gamma, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, B \vdash G}\ (3.4)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– $R{=}$:

$$\dfrac{\dfrac{}{\Gamma, (\forall x.A){\rightarrow}B \vdash t{=}t}\, R{=}}{\Gamma, B \vdash t{=}t}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{}{\Gamma, B \vdash t{=}t}\, R{=}$$

– $L{=}_1$, G est une formule atomique ou une égalité :

$$\dfrac{\dfrac{\Gamma, s{=}t, (\forall x.A){\rightarrow}B \vdash G'}{\Gamma, s{=}t, (\forall x.A){\rightarrow}B \vdash G}\, L{=}_1}{\Gamma, s{=}t, B \vdash G}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, (\forall x.A){\rightarrow}B \vdash G'}{\Gamma, s{=}t, B \vdash G'}\, Ind}{\Gamma, s{=}t, B \vdash G}\, L{=}_1$$

– $L{=}_2$:

$$\dfrac{\dfrac{\Gamma, s{=}t, P'{\rightarrow}C, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, s{=}t, P{\rightarrow}C, (\forall x.A){\rightarrow}B \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\rightarrow}C, B \vdash G}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P'{\rightarrow}C, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, s{=}t, P'{\rightarrow}C, B \vdash G}\, Ind}{\Gamma, s{=}t, P{\rightarrow}C, B \vdash G}\, L{=}_2$$

– $L{=}{\rightarrow}$:

$$\dfrac{\dfrac{\Gamma, C, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, t{=}t{\rightarrow}C, (\forall x.A){\rightarrow}B \vdash G}\, L{=}{\rightarrow}}{\Gamma, t{=}t{\rightarrow}C, B \vdash G}\ \text{règle 3.4} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, C, (\forall x.A){\rightarrow}B \vdash G}{\Gamma, C, B \vdash G}\, Ind}{\Gamma, t{=}t{\rightarrow}C, B \vdash G}\, L{=}{\rightarrow}$$

Ceci clôt la démonstration par récurrence. □

Inversibilité de la règle LI

Nous allons montrer que la règle 3.5 ci-dessous est fortement admissible dans $LJTI_=$.

$$\frac{\Gamma, I(\mathbf{p}) \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G} \quad (3.5)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– $R=$:

$$\frac{\dfrac{}{\Gamma, I(\mathbf{p}) \vdash t{=}t}\, R=}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash t{=}t}\ \text{règle 3.5} \quad \Longrightarrow \quad \frac{}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash t{=}t}\, R=$$

– $L{=}_1$, G est une formule atomique ou une égalité :

$$\frac{\dfrac{\dfrac{\Gamma, s{=}t, I(\mathbf{p}) \vdash G'}{\Gamma, s{=}t, I(\mathbf{p}) \vdash G}\, L{=}_1}{\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, I(\mathbf{p}) \vdash G'}{\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G'}\, Ind}{\Gamma, s{=}t, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L{=}_1$$

– $L{=}_2$:

$$\frac{\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, I(\mathbf{p}) \vdash G}{\Gamma, s{=}t, P{\to}C, I(\mathbf{p}) \vdash G}\, L{=}_2}{\Gamma, s{=}t, P{\to}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, I(\mathbf{p}) \vdash G}{\Gamma, s{=}t, P'{\to}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, s{=}t, P{\to}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L{=}_2$$

– $L{=}{\to}$:

$$\frac{\dfrac{\dfrac{\Gamma, C, I(\mathbf{p}) \vdash G}{\Gamma, t{=}t{\to}C, I(\mathbf{p}) \vdash G}\, L{=}{\to}}{\Gamma, t{=}t{\to}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\ \text{règle 3.5} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, C, I(\mathbf{p}) \vdash G}{\Gamma, C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, Ind}{\Gamma, t{=}t{\to}C, \mathbf{H_i}(\mathbf{p}, \mathbf{t}) \vdash G}\, L{=}{\to}$$

– Les autres cas sont traités comme pour $LJTI$.

Ceci clôt la démonstration par récurrence. □

Inversibilité de la règle $LI{\to}$

Nous allons montrer que la règle 3.6 ci-dessous est fortement admissible dans $LJTI_=$.

$$\frac{\Gamma, I(\mathbf{p}){\to}B \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i}){\to}B\}_i \vdash G} \quad (3.6)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

– $R{=}$:

$$\dfrac{\dfrac{}{\Gamma, I(\mathbf{p}){\to}B \vdash t{=}t}\ R{=}}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash t{=}t}\ \text{règle 3.6}$$

$$\Downarrow$$

$$\dfrac{}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash t{=}t}\ R{=}$$

– $L{=}_1$, G est une formule atomique ou une égalité :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, I(\mathbf{p}){\to}B \vdash G'}{\Gamma, s{=}t, I(\mathbf{p}){\to}B \vdash G}\ L{=}_1}{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ \text{règle 3.6}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, s{=}t, I(\mathbf{p}){\to}B \vdash G'}{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G'}\ Ind}{\Gamma, s{=}t, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ L{=}_1$$

– $L{=}_2$:

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, I(\mathbf{p}){\to}B \vdash G}{\Gamma, s{=}t, P{\to}C, I(\mathbf{p}){\to}B \vdash G}\ L{=}_2}{\Gamma, s{=}t, P{\to}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ \text{règle 3.6}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, s{=}t, P'{\to}C, I(\mathbf{p}){\to}B \vdash G}{\Gamma, s{=}t, P'{\to}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ Ind}{\Gamma, s{=}t, P{\to}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ L{=}_2$$

– $L{=}{\to}$:

$$\dfrac{\dfrac{\dfrac{\Gamma, C, I(\mathbf{p}){\to}B \vdash G}{\Gamma, t{=}t{\to}C, I(\mathbf{p}){\to}B \vdash G}\ L{=}{\to}}{\Gamma, t{=}t{\to}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ \text{règle 3.6}}{\Downarrow}$$

$$\dfrac{\dfrac{\Gamma, C, I(\mathbf{p}){\to}B \vdash G}{\Gamma, C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ Ind}{\Gamma, t{=}t{\to}C, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p},\mathbf{y_i}){\to}B\}_i \vdash G}\ L{=}{\to}$$

– Les autres cas sont traités comme pour $LJTI$.

Ceci clôt la démonstration par récurrence. □

A.2.2 Élimination des égalités triviales

Nous allons montrer que la règle 3.11 ci-dessous est fortement admissible dans $LJTI_=$.

$$\frac{\Gamma, t{=}t \vdash G}{\Gamma \vdash G} \text{ (3.11)}$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

- Ax :

$$\dfrac{\dfrac{}{\Gamma, t{=}t, P \vdash P}\, Ax}{\Gamma, P \vdash P}\ \text{règle 3.11} \quad \Longrightarrow \quad \frac{}{\Gamma, P \vdash P}\, Ax$$

- $R{\to}$: La preuve est de la forme :

$$\dfrac{\dfrac{\Gamma, t{=}t, C \vdash D}{\Gamma, t{=}t \vdash C{\to}D}\, R{\to}}{\Gamma \vdash C{\to}D}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, t{=}t, C \vdash D}{\Gamma, C \vdash D}\, Ind}{\Gamma \vdash C{\to}D}\, R{\to}$$

- $La{\to}$:

$$\dfrac{\dfrac{\Gamma, P, C, t{=}t \vdash G}{\Gamma, P, P{\to}C, t{=}t \vdash G}\, La{\to}}{\Gamma, P, P{\to}C \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, P, C, t{=}t \vdash G}{\Gamma, P, C \vdash G}\, Ind}{\Gamma, P, P{\to}C \vdash G}\, La{\to}$$

- $L{\to}{\to}$:

$$\dfrac{\dfrac{\Gamma, D{\to}B, C, t{=}t \vdash D \quad \Gamma, B, t{=}t \vdash G}{\Gamma, (C{\to}D){\to}B, t{=}t \vdash G}\, L{\to}{\to}}{\Gamma, (C{\to}D){\to}B \vdash G}\ \text{règle 3.11}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma, D{\to}B, C, t{=}t \vdash D}{\Gamma, D{\to}B, C \vdash D}\, Ind \quad \dfrac{\Gamma, B, t{=}t \vdash G}{\Gamma, B \vdash G}\, Ind}{\Gamma, (C{\to}D){\to}B \vdash G}\, L{\to}{\to}$$

- $R\forall$:

$$\dfrac{\dfrac{\Gamma, t{=}t \vdash A}{\Gamma, t{=}t \vdash \forall x.A}\, R\forall}{\Gamma \vdash \forall x.A}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, t{=}t \vdash A}{\Gamma \vdash A}\, Ind}{\Gamma \vdash \forall x.A}\, L\forall$$

- $L\forall$:

$$\dfrac{\dfrac{\Gamma, \forall x.A, A\{u/x\}, t{=}t \vdash G}{\Gamma, \forall x.A, t{=}t \vdash G}\, L\forall}{\Gamma, \forall x.A \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, \forall x.A, A\{u/x\}, t{=}t \vdash G}{\Gamma, \forall x.A, A\{u/x\} \vdash G}\, Ind}{\Gamma, \forall x.A \vdash G}\, L\forall$$

– $L\forall\rightarrow$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, t{=}t \vdash \forall x.A \quad \Gamma, B, t{=}t \vdash G}{\Gamma, (\forall x.A)\rightarrow B, t{=}t \vdash G}\, L\forall\rightarrow}{\Gamma, (\forall x.A)\rightarrow B \vdash G}\ \text{règle 3.11}}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, t{=}t \vdash \forall x.A}{\Gamma, (\forall x.A)\rightarrow B \vdash \forall x.A}\, Ind \quad \dfrac{\Gamma, B, t{=}t \vdash G}{\Gamma, B \vdash G}\, Ind}{\Gamma, (\forall x.A)\rightarrow B \vdash G}\, L\rightarrow\rightarrow}$$

– RI_i :

$$\dfrac{\dfrac{\dots \quad \Gamma, t{=}t \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \dots}{\Gamma, t{=}t \vdash I(\mathbf{p})}\, RI_i}{\Gamma \vdash I(\mathbf{p})}\ \text{règle 3.11}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\Gamma, t{=}t \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind \quad \dots}{\Gamma \vdash I(\mathbf{p})}\, RI_i$$

– LI :

$$\dfrac{\dfrac{\dots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), t{=}t \vdash G \quad \dots}{\Gamma, I(\mathbf{p}), t{=}t \vdash G}\, LI}{\Gamma, I(\mathbf{p}) \vdash G}\ \text{règle 3.11}$$

$$\Downarrow$$

$$\dfrac{\dots \quad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), t{=}t \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}) \vdash G}\, Ind \quad \dots}{\Gamma, I(\mathbf{p}) \vdash G}\, LI$$

– $LI\rightarrow$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow C\}_i, t{=}t \vdash G}{\Gamma, I(\mathbf{p})\rightarrow C, t{=}t \vdash G}\, LI\rightarrow}{\Gamma, I(\mathbf{p})\rightarrow C \vdash G}\ \text{règle 3.11}}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow C\}_i, t{=}t \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow C\}_i \vdash G}\, Ind}{\Gamma, I(\mathbf{p})\rightarrow C \vdash G}\, LI\rightarrow}$$

– $R=$:

$$\dfrac{\dfrac{}{\Gamma, t{=}t \vdash u{=}u}\, R=}{\Gamma \vdash u{=}u}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{}{\Gamma \vdash u{=}u}\, R=$$

– $L=_1$:

1. Si $t{=}t$ est principale, G est réécrit sur lui-même :

$$\dfrac{\dfrac{\Gamma, t{=}t \vdash G}{\Gamma, t{=}t \vdash G}\, L=_1}{\Gamma \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\Gamma, t{=}t \vdash G}{\Gamma \vdash G}\, Ind$$

2. Sinon :

$$\dfrac{\dfrac{\Gamma, s{=}u, t{=}t \vdash G'}{\Gamma, s{=}u, t{=}t \vdash G}\, L=_1}{\Gamma, s{=}u \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}u, t{=}t \vdash G'}{\Gamma, s{=}u \vdash G'}\, Ind}{\Gamma, s{=}u \vdash G}\, L=_1$$

– $L=_2$, deux cas sont possibles :

1. La formule $t{=}t$ est principale et $P{\to}B$ est réécrit sur lui-même :

$$\dfrac{\dfrac{\Gamma, t{=}t, P{\to}B \vdash G}{\Gamma, t{=}t, P{\to}B \vdash G}\, L=_2}{\Gamma, P{\to}B \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\Gamma, t{=}t, P{\to}B \vdash G}{\Gamma, P{\to}B \vdash G}\, Ind$$

2. Sinon :

$$\dfrac{\dfrac{\Gamma, s{=}u, P'{\to}B, t{=}t \vdash G}{\Gamma, s{=}u, P{\to}B, t{=}t \vdash G}\, L=_2}{\Gamma, s{=}u, P{\to}B \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, s{=}u, P'{\to}B, t{=}t \vdash G}{\Gamma, s{=}u, P'{\to}B \vdash G}\, Ind}{\Gamma, s{=}u, P{\to}B \vdash G}\, L=_2$$

– $L{=}{\to}$:

$$\dfrac{\dfrac{\Gamma, C, t{=}t \vdash G}{\Gamma, u{=}u{\to}C, t{=}t \vdash G}\, L{=}{\to}}{\Gamma, u{=}u{\to}C \vdash G}\ \text{règle 3.11} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, C, t{=}t \vdash G}{\Gamma, C \vdash G}\, Ind}{\Gamma, u{=}u{\to}C \vdash G}\, L{=}{\to}$$

– Les autres cas sont traités comme pour $LJTI$.

Ceci clôt la démonstration par récurrence. □

Inversibilité de $L{=}{\to}$

Nous allons montrer que la règle 3.11 ci-dessous est fortement admissible dans $LJTI_{=}$.

$$\frac{\Gamma, s{=}t{\to}C \vdash G}{\Gamma, C \vdash G} \quad (3.12)$$

Démonstration : La démonstration se fait par récurrence sur la hauteur de la dérivation de la prémisse, et par cas sur la règle précédente.

- Ax :

$$\frac{\dfrac{}{\Gamma, s{=}t{\to}C, P \vdash P}\, Ax}{\Gamma, P \vdash P}\ \text{règle 3.12} \quad \Longrightarrow \quad \frac{}{\Gamma, C, P \vdash P}\, Ax$$

- $R{\to}$: La preuve est de la forme :

$$\frac{\dfrac{\Gamma, s{=}t{\to}C, A \vdash B}{\Gamma, s{=}t{\to}C \vdash A{\to}B}\, R{\to}}{\Gamma, C \vdash A{\to}B}\ \text{règle 3.12} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t{\to}C, A \vdash B}{\Gamma, C, A \vdash B}\, Ind}{\Gamma, C \vdash A{\to}B}\, R{\to}$$

- $La{\to}$:

$$\frac{\dfrac{\Gamma, P, B, s{=}t{\to}C \vdash G}{\Gamma, P, P{\to}B, s{=}t{\to}C \vdash G}\, La{\to}}{\Gamma, P, P{\to}B, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, P, B, s{=}t{\to}C \vdash G}{\Gamma, P, B, C \vdash G}\, Ind}{\Gamma, P, P{\to}B, C \vdash G}\, La{\to}$$

- $L{\to}{\to}$:

$$\frac{\Gamma, D{\to}B, A, s{=}t{\to}C \vdash D \quad \Gamma, B, s{=}t{\to}C \vdash G}{\dfrac{\Gamma, (A{\to}D){\to}B, s{=}t{\to}C \vdash G}{\Gamma, (A{\to}D){\to}B, C \vdash G}\ \text{règle 3.12}}\, L{\to}{\to}$$

$$\Downarrow$$

$$\frac{\dfrac{\Gamma, D{\to}B, A, s{=}t{\to}C \vdash D}{\Gamma, D{\to}B, A, C \vdash D}\, Ind \quad \dfrac{\Gamma, B, s{=}t{\to}C \vdash G}{\Gamma, B, C \vdash G}\, Ind}{\Gamma, (A{\to}D){\to}B, C \vdash G}\, L{\to}{\to}$$

- $R\forall$:

$$\frac{\dfrac{\Gamma, s{=}t{\to}C \vdash A}{\Gamma, s{=}t{\to}C \vdash \forall x.A}\, R\forall}{\Gamma, C \vdash \forall x.A}\ \text{règle 3.12} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, s{=}t{\to}C \vdash A}{\Gamma, C \vdash A}\, Ind}{\Gamma, C \vdash \forall x.A}\, L\forall$$

- $L\forall$:

$$\frac{\dfrac{\Gamma, \forall x.A, A\{u/x\}, s{=}t{\to}C \vdash G}{\Gamma, \forall x.A, s{=}t{\to}C \vdash G}\, L\forall}{\Gamma, \forall x.A, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \frac{\dfrac{\Gamma, \forall x.A, A\{u/x\}, s{=}t{\to}C \vdash G}{\Gamma, \forall x.A, A\{u/x\}, C \vdash G}\, Ind}{\Gamma, \forall x.A, C \vdash G}\, L\forall$$

– $L\forall\rightarrow$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, s{=}t\rightarrow C \vdash \forall x.A \quad \Gamma, B, s{=}t\rightarrow C \vdash G}{\Gamma, (\forall x.A)\rightarrow B, s{=}t\rightarrow C \vdash G}\, L\forall\rightarrow}{\Gamma, (\forall x.A)\rightarrow B, C \vdash G}\ \text{règle 3.12}}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, (\forall x.A)\rightarrow B, s{=}t\rightarrow C \vdash \forall x.A}{\Gamma, (\forall x.A)\rightarrow B, C \vdash \forall x.A}\, Ind \quad \dfrac{\Gamma, B, s{=}t\rightarrow C \vdash G}{\Gamma, B, C \vdash G}\, Ind}{\Gamma, (\forall x.A)\rightarrow B, C \vdash G}\, L\rightarrow\rightarrow}$$

– RI_i :

$$\dfrac{\dfrac{\dfrac{\ldots \quad \Gamma, s{=}t\rightarrow C \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i}) \quad \ldots}{\Gamma, s{=}t\rightarrow C \vdash I(\mathbf{p})}\, RI_i}{\Gamma, C \vdash I(\mathbf{p})}\ \text{règle 3.12}}{\Downarrow}$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, s{=}t\rightarrow C \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}{\Gamma, C \vdash H_{i,j}(\mathbf{p}, \mathbf{t_i})}\, Ind \quad \ldots}{\Gamma, C \vdash I(\mathbf{p})}\, RI_i$$

– LI :

$$\dfrac{\dfrac{\dfrac{\ldots \quad \Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), s{=}t\rightarrow C \vdash G \quad \ldots}{\Gamma, I(\mathbf{p}), s{=}t\rightarrow C \vdash G}\, LI}{\Gamma, I(\mathbf{p}), C \vdash G}\ \text{règle 3.12}}{\Downarrow}$$

$$\dfrac{\ldots \quad \dfrac{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), s{=}t\rightarrow C \vdash G}{\Gamma, \mathbf{H_i}(\mathbf{p}, \mathbf{y_i}), C \vdash G}\, Ind \quad \ldots}{\Gamma, I(\mathbf{p}), C \vdash G}\, LI$$

– $LI\rightarrow$:

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i, s{=}t\rightarrow C \vdash G}{\Gamma, I(\mathbf{p})\rightarrow B, s{=}t\rightarrow C \vdash G}\, LI\rightarrow}{\Gamma, I(\mathbf{p})\rightarrow B, C \vdash G}\ \text{règle 3.12}}{\Downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i, s{=}t\rightarrow C \vdash G}{\Gamma, \{\forall \mathbf{y_i}.\mathbf{H_i}(\mathbf{p}, \mathbf{y_i})\rightarrow B\}_i, C \vdash G}\, Ind}{\Gamma, I(\mathbf{p})\rightarrow B, C \vdash G}\, LI\rightarrow}$$

– $R{=}$:

$$\dfrac{\dfrac{}{\Gamma, s{=}t{\to}C \vdash u{=}u}\, R{=}}{\Gamma, C \vdash u{=}u}\ \text{règle 3.12} \quad \Longrightarrow \quad \dfrac{}{\Gamma, C \vdash u{=}u}\, R{=}$$

– $L{=}_1$:

$$\dfrac{\dfrac{\dfrac{\Gamma, u{=}v, s{=}t{\to}C \vdash G'}{\Gamma, u{=}v, s{=}t{\to}C \vdash G}\, L{=}_1}{\Gamma, u{=}v, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, u{=}v, s{=}t{\to}C \vdash G'}{\Gamma, u{=}v, C \vdash G'}\, Ind}{\Gamma, u{=}v, C \vdash G}\, L{=}_1$$

– $L{=}_2$, deux cas sont possibles :

1. La formule $s{=}t{\to}C$ est réécrite :

$$\dfrac{\dfrac{\dfrac{\Gamma, s{=}t'{\to}C, u{=}v \vdash G}{\Gamma, s{=}t{\to}C, u{=}v \vdash G}\, L{=}_2}{\Gamma, P{\to}B \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \dfrac{\Gamma, s{=}t'{\to}C, u{=}v \vdash G}{\Gamma, P{\to}B \vdash G}\, Ind$$

2. Sinon :

$$\dfrac{\dfrac{\dfrac{\Gamma, u{=}v, P'{\to}B, s{=}t{\to}C \vdash G}{\Gamma, u{=}v, P{\to}B, s{=}t{\to}C \vdash G}\, L{=}_2}{\Gamma, u{=}v, P{\to}B, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, u{=}v, P'{\to}B, s{=}t{\to}C \vdash G}{\Gamma, u{=}v, P'{\to}B, C \vdash G}\, Ind}{\Gamma, u{=}v, P{\to}B, C \vdash G}\, L{=}_2$$

– $L{=}{\to}$, deux cas sont possibles :

1. Si $s{=}t{\to}B$ est la formule principale, alors $s \equiv t$:

$$\dfrac{\dfrac{\dfrac{\Gamma, C \vdash G}{\Gamma, s{=}s{\to}C, \vdash G}\, L{=}{\to}}{\Gamma, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \Gamma, C \vdash G$$

2. Sinon :

$$\dfrac{\dfrac{\dfrac{\Gamma, B, s{=}t{\to}C \vdash G}{\Gamma, u{=}u{\to}B, s{=}t{\to}C \vdash G}\, L{=}{\to}}{\Gamma, u{=}u{\to}B, C \vdash G}\ \text{règle 3.12} \quad \Longrightarrow \quad \dfrac{\dfrac{\Gamma, B, s{=}t{\to}C \vdash G}{\Gamma, B, C \vdash G}\, Ind}{\Gamma, u{=}u{\to}B, C \vdash G}\, L{=}{\to}$$

Ceci clôt la démonstration par récurrence. □

Annexe B

Non-définissabilité de la coercition de types dans le Calcul des Constructions Inductives

Soit la définition de paire dépendante suivante :

```
Record entry: Type := mkE { type:Set; obj:type }.
```

Théorème B.1 (Coercition de types)
Il n'existe pas de fonction `coerce` *ayant pour type* $\forall$`(A : Set),entry` $\rightarrow$ `option A` *dans le Calcul des Constructions Inductives telle que pour tout terme* `A : Set` *et* `x : A`, `coerce A (mkE A x)` *se réduise en* `Some x`.

Démonstration : La démonstration est directement adaptée d'un résultat de Jean-Yves Girard [Gir71]. Supposons l'existence d'une telle fonction et montrons qu'elle permet de construire un terme non-normalisant. Tout d'abord, nous allons nous placer dans la sorte Prop grâce aux définitions suivantes :

```
Record embed (A:Prop):Set  := inject {retract : A}.
Inductive opt (A:Prop):Prop := none : opt A | some : A → opt A.
```

Le type `embed` permet d'encapsuler un objet de sorte Prop dans un objet de sorte Set, et le type `opt` est la version Prop du type `option`. Ainsi, nous pouvons construire une fonction analogue à `coerce` mais dans la sorte Prop.

```
Definition coerce_prop  (A B:Prop) (b:B) : opt A:=
  match coerce (embed A) (mkE (embed B) (inject B b)) with
    Some a => some A (retract A a)
  | None => none A end.
```

Cette fonction est telle que pour tout terme `A` : Prop et `x` : `A`, `coerce A (mkE A x)` se réduise en `some x`.

```
Definition Apply (A B:Prop) of (x:A) : opt B :=
match of with some f => f x | none => (none B) end.
```

Le combinateur `Apply` est un applicateur, il est tel que `Apply A B Some f x` se réduit en `f x`. Comme nous sommes passés dans la sorte Prop qui est imprédicative, le type `D` suivant est de sorte Prop :

```
Definition D := ∀ A : Prop, opt A.
```

On construit ensuite le terme `C` qui est une version typée du λ-terme $\Delta = \lambda z.zz$:

```
Definition C B:= coerce_prop B (D → opt D)
  (λz. Apply D D (z (D → opt D)) z).
```

On peut alors construire une version typée de $\Omega = \Delta\Delta$.

```
Definition Paradox := Apply D D (C (D → opt D)) C.
```

Le terme `Paradox` se réduit alors en :

```
Apply D D
  (coerce_prop (D → opt D) (D → opt D)
    (λz. Apply D D (z (D → opt D)) z))
  (λB. (coerce_prop B (D → opt D)
    (λz. Apply D D (z (D → opt D)) z)))                    (i)
```

Par définition de `coerce_prop`, (i) se réduit en :

```
Apply D D
  (some (D → opt D) (λz. Apply D D (z (D → opt D)) z))
  (λB. (coerce_prop B (D → opt D)
    (λz. Apply D D (z (D → opt D)) z)))                    (ii)
```

Ensuite, par réduction de `Apply` :

```
(λz. Apply D D (z (D → opt D)) z))
  (λB. (coerce_prop B (D → opt D)
    (λz. Apply D D (z (D → opt D)) z)))                    (iii)
```

Ensuite, par une série de β-réductions :

```
Apply D D
  (coerce_prop (D → opt D) (D → opt D)
    (λz. Apply D D (z (D → opt D)) z))
  (λB. (coerce_prop B (D → opt D)
    (λz. Apply D D (z (D → opt D)) z)))                    (iv)
```

On remarque alors que $(iv) = (i)$ et la réduction $(i) \rightarrow (ii) \rightarrow (iii) \rightarrow (i) \rightarrow \ldots$ montre que le terme `Paradox` n'est pas normalisable, ce qui est contradictoire avec la normalisation forte du fragment sans univers du Calcul des Constructions Inductives démontrée par Benjamin Werner [Wer94].

On en conclut qu'une telle fonction `coerce` n'existe pas. □

Annexe C

Résultats des tests

C.1 Comparaison de Firstorder et Jprover sur la librairie de problèmes ILTP

Nous donnons dans cette annexe les résultats annoncés page 97.

Le statut T désigne les problèmes démontrables et N les problèmes non démontrables. Le statut T* désigne les cas où une preuve est trouvée mais produite incorrectement par `Jprover`. Pour chaque mesure, nous donnons le temps utilisateur u et le temps système s en secondes.

Problème	Note	Statut	`Fo v2`	temps	vérif.	`Jp`	temps	vérif.
ILTP								
`con_n.002.p`	0.60	N	N	$0.07u, 0.02s$		-	∞	
`con_n.006.p`	0.60	N	N	$3.85u, 0.08s$		-	∞	
`con_n.010.p`	0.60	N	-	∞		-	∞	
`con_p.002.p`	0.20	T	T	$0.07u, 0.07s$	$0.u, 0.s$	T	$0.11u, 0.05s$	$0.u, 0.s$
`con_p.006.p`	0.60	T	T	$0.14u, 0.06s$	$0.02u, 0.s$	-	∞	
`con_p.010.p`	0.60	T	T	$0.29u, 0.06s$	$0.12u, 0.s$	-	∞	
`debruijn_n.002.p`	0.60	N	N	$26.53u, 0.09s$		-	∞	
`debruijn_n.006.p`	0.80	N	-	∞		-	∞	
`debruijn_n.010.p`	1.00	N	-	∞		-	∞	
`debruijn_p.002.p`	0.60	T	T	$2.38u, 0.04s$	$0.56u, 0.01s$	-	∞	
`debruijn_p.006.p`	0.60	T	-	∞		-	∞	
`debruijn_p.010.p`	0.60	T	-	∞		-	∞	
`equiv_n.002.p`	0.60	N	N	$0.08u, 0.03s$		-	∞	
`equiv_n.006.p`	0.60	N	-	∞		-	∞	
`equiv_n.010.p`	0.60	N	-	∞		-	∞	
`equiv_p.002.p`	0.20	T	T	$0.09u, 0.06s$	$0.u, 0.s$	T	$0.10u, 0.05s$	$0.u, 0.s$
`equiv_p.006.p`	0.60	T	-	∞		-	∞	
`equiv_p.010.p`	0.60	T	-	∞		-	∞	
`ft1.1.p`	0.00	T	-	∞		T*	$0.12u, 0.03s$	
`ft1.2.p`	0.00	T	-	∞		T*	$0.20u, 0.07s$	
`ft1.3.p`	0.25	T	-	∞		T*	$0.59u, 0.07s$	
`ft1.4.p`	0.00	T	-	∞		T*	$0.12u, 0.04s$	
`ft1.5.p`	0.00	T	-	∞		T*	$0.21u, 0.06s$	
`ft1.6.p`	0.00	T	-	∞		T*	$0.43u, 0.03s$	
`ft1.7.p`	0.00	T	-	∞		T*	$0.10u, 0.06s$	

../..

Problème	Note	Statut	Fo v2	temps	vérif.	Jp	temps	vérif.
ft1.8.p	0.25	T	-	∞		T*	$0.17u, 0.06s$	
ft2.1.p	0.50	T	-	∞		-	∞	
ft2.2.p	0.50	T	-	∞		-	∞	
ft2.3.p	0.50	T	-	∞		-	∞	
ft2.4.p	0.50	T	-	∞		-	∞	
ft4.1.p	0.25	T	-	∞		-	∞	
ft4.2.p	0.50	T	-	∞		-	∞	
ft4.3.p	0.50	T	-	∞		-	∞	
ft5.1.p	0.00	T	T	$0.52u, 0.02s$	$0.02u, 0.s$	T	$0.14u, 0.02s$	$0.03u, 0.s$
ft5.2.p	0.00	T	-	∞		T	$20.34u, 0.06s$	$0.83u, 0.s$
ft5.3.p	0.25	T	-	∞		-	∞	
ft6.10.p	0.00	T	T	$205.60u, 0.20s$	$0.u, 0.s$	T	$0.11u, 0.02s$	$0.01u, 0.s$
ft6.11.p	0.00	T	T	$0.93u, 0.03s$	$0.u, 0.s$	T	$0.09u, 0.02s$	$0.u, 0.s$
ft6.12.p	0.25	T	T	$12.60u, 0.03s$	$0.01u, 0.s$	-	∞	
ft6.13.p	0.25	T	-	∞		-	∞	
ft6.14.p	0.00	T	-	∞		T*	$0.19u, 0.03s$	
ft6.15.p	0.00	T	T	$17.47u, 0.04s$	$0.u, 0.s$	T	$0.98u, 0.04s$	$0.01u, 0.s$
ft6.1.p	0.00	T	T	$0.19u, 0.03s$	$0.u, 0.s$	T	$0.09u, 0.02s$	$0.u, 0.s$
ft6.2.p	0.00	T	-	∞		T	$0.20u, 0.03s$	$0.03u, 0.s$
ft6.3.p	0.00	T	T	$7.15u, 0.04s$	$0.u, 0.s$	T	$1.03u, 0.02s$	$0.u, 0.s$
ft6.4.p	0.00	T	T	$0.81u, 0.03s$	$0.u, 0.s$	T	$0.14u, 0.03s$	$0.u, 0.s$
ft6.5.p	0.25	T	T	$2.20u, 0.03s$	$0.u, 0.s$	T	$0.16u, 0.02s$	$0.01u, 0.s$
ft6.6.p	0.00	T	T	$0.10u, 0.02s$	$0.u, 0.s$	T	$0.10u, 0.02s$	$0.01u, 0.s$
ft6.7.p	0.00	T	T	$0.08u, 0.03s$	$0.u, 0.s$	T	$0.09u, 0.02s$	$0.01u, 0.s$
ft6.8.p	0.00	T	-	∞		T	$0.10u, 0.02s$	$0.01u, 0.s$
ft6.9.p	0.00	T	T	$11.80u, 0.04s$	$0.01u, 0.s$	T	$0.09u, 0.03s$	$0.u, 0.s$
ft7a.p	0.00	T	T	$0.18u, 0.02s$	$0.u, 0.s$	T	$0.10u, 0.02s$	$0.u, 0.s$
ft7b.p	0.00	T	T	$42.91u, 0.06s$	$0.01u, 0.s$	T	$0.14u, 0.03s$	$0.01u, 0.s$
ft7c.p	0.00	T	-	∞		T	$2.03u, 0.03s$	$0.02u, 0.s$
ft7d.p	0.25	T	-	∞		T	$182.94u, 0.17s$	$0.02u, 0.s$
ft8.1.p	0.00	T	T	$33.97u, 0.06s$	$0.u, 0.s$	T	$16.25u, 0.05s$	$0.u, 0.s$
ft8.2.p	0.00	T	T	$0.42u, 0.03s$	$0.u, 0.s$	T	$0.41u, 0.03s$	$0.01u, 0.s$
kk_n.002.p	0.40	N	N	$0.08u, 0.03s$		-	∞	
kk_n.006.p	0.60	N	-	∞		-	∞	
kk_n.010.p	0.80	N	-	∞		-	∞	
kk_p.002.p	0.20	T	T	$0.09u, 0.03s$	$0.01u, 0.s$	T	$0.22u, 0.03s$	$0.u, 0.s$
kk_p.006.p	0.60	T	-	∞		-	∞	
kk_p.010.p	0.80	T	-	∞		-	∞	
ph_n.002.p	0.40	N	N	$0.09u, 0.02s$		-	∞	
ph_n.006.p	0.60	N	-	∞		-	∞	
ph_n.010.p	0.80	N	-	∞		-	∞	
ph_p.002.p	0.00	T	T	$0.12u, 0.02s$	$0.01u, 0.s$	T	$0.12u, 0.04s$	$0.01u, 0.s$
ph_p.006.p	0.40	T	-	∞		-	∞	
ph_p.010.p	1.00	T	-	∞		-	∞	
sch_agatha.p	0.25	T	-	∞		T	$1.63u, 0.03s$	$0.01u, 0.s$
sch_all_exst.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T*	$0.07u, 0.03s$	
sch_ax_all.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
sch_ax.p	0.00	T	T	$0.07u, 0.03s$	$0.u, 0.s$	T	$0.07u, 0.03s$	$0.u, 0.s$
sch_deadlock1.p	0.00	T	-	∞		N	$0.07u, 0.03s$	
sch_deadlock2.p	0.00	T	-	∞		N	$0.08u, 0.02s$	
sch_deadlock3.p	0.00	T	-	∞		N	$0.08u, 0.02s$	
sch_deadlock4.p	0.00	T	-	∞		N	$0.09u, 0.02s$	
sch_fo_n.001.p	0.00	T	T	$0.09u, 0.02s$	$0.u, 0.s$	T	$0.10u, 0.02s$	$0.u, 0.s$
sch_fo_n.002.p	0.00	T	T	$80.00u, 0.09s$	$0.u, 0.s$	T	$0.74u, 0.03s$	$0.01u, 0.s$
sch_fo_n.003.p	0.00	T	-	∞		T	$38.92u, 0.06s$	$0.02u, 0.s$
sch_fo_n.004.p	0.00	T	-	∞		-	∞	

../..

Problème	Note	Statut	Fo v2	temps	vérif.	Jp	temps	vérif.
sch_fun1.p	0.00	T	T	$0.25u, 0.03s$	$0.u, 0.s$	T	$0.59u, 0.02s$	$0.u, 0.s$
sch_fun2.p	0.00	T	T	$0.11u, 0.02s$	$0.u, 0.s$	T	$0.10u, 0.03s$	$0.01u, 0.s$
sch_fv2.p	0.00	T	T	$0.07u, 0.02s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
sch_implies.p	0.00	T	T	$0.07u, 0.02s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
sch_jens_fo_fv.p	0.00	T	T	$239.73u, 0.21s$	$0.u, 0.s$	T	$0.27u, 0.03s$	$0.01u, 0.s$
sch_jens_fo.p	0.00	T	T	$229.18u, 0.23s$	$0.u, 0.s$	T	$0.31u, 0.04s$	$0.01u, 0.s$
sch_jens_prop.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T	$0.09u, 0.02s$	$0.01u, 0.s$
sch_mult.002.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T	$0.07u, 0.02s$	$0.u, 0.s$
sch_mult.003.p	0.00	T	T	$0.09u, 0.02s$	$0.u, 0.s$	T	$0.09u, 0.02s$	$0.u, 0.s$
sch_mult.004.p	0.00	T	T	$0.09u, 0.02s$	$0.01u, 0.s$	T	$0.11u, 0.02s$	$0.u, 0.s$
sch_mult_eigen_del.p	0.00	T	-	∞		N	$0.08u, 0.02s$	
sch_mult_no_rename2.p	0.00	T	-	∞		N	$0.08u, 0.03s$	
sch_mult_no_rename.p	0.00	T	-	∞		N	$0.08u, 0.02s$	
sch_mult_rename.p	0.00	T	-	∞		N	$0.08u, 0.02s$	
sch_notnot2.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
sch_notnot.p	0.00	T	T	$0.07u, 0.03s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
sch_prop_n.001.p	0.00	T	N	$0.06u, 0.02s$		N	$0.06u, 0.02s$	
sch_prop_n.002.p	0.00	T	T	$0.09u, 0.02s$	$0.01u, 0.s$	T	$0.10u, 0.02s$	$0.01u, 0.s$
sch_prop_n.003.p	0.00	T	T	$0.14u, 0.02s$	$0.01u, 0.s$	T	$0.18u, 0.02s$	$0.01u, 0.s$
sch_prop_n.004.p	0.00	T	T	$0.28u, 0.03s$	$0.05u, 0.s$	T	$0.37u, 0.03s$	$0.03u, 0.s$
sch_subst.p	0.00	T	T	$1.90u, 0.02s$	$0.u, 0.s$	T*	$0.08u, 0.03s$	
schwicht_n.002.p	0.40	N	N	$0.07u, 0.02s$		-	∞	
schwicht_n.006.p	0.40	N	N	$0.08u, 0.02s$		-	∞	
schwicht_n.010.p	0.40	N	N	$0.08u, 0.03s$		-	∞	
schwicht_p.002.p	0.00	T	T	$0.08u, 0.02s$	$0.u, 0.s$	T	$0.08u, 0.02s$	$0.u, 0.s$
schwicht_p.006.p	0.40	T	T	$0.08u, 0.06s$	$0.u, 0.s$	-	∞	
schwicht_p.010.p	0.40	T	T	$0.13u, 0.03s$	$0.02u, 0.s$	-	∞	

C.2 Résultats des preuves par complétion de la bibliothèque TPTP

Nous donnons dans cette annexe les résultats annoncés page 168.

Les résultats sont donnés en secondes. Le temps donné pour Coq est le temps du typage de la preuve par le noyau. La limite de temps est de 300 secondes pour CiME et 300 secondes pour les quatre preuves Coq.

Problème	Note TPTP	Temps CiME	Temps Coq (court) réflexion	Temps Coq (court) tactiques	Temps Coq (long) réflexion	Temps Coq (long) tactiques
ALG						
ALG006-1.p	0.0	0.33	0.01	0.03	0.02	0.14
ALG007-1.p	0.11	30.51	0.01	0.02	0.	0.05
BOO						
BOO001-1.p	0.0	0.02	0.	0.01	0.	0.01
BOO002-1.p	0.0	14.21	0.01	0.02	0.01	0.04
BOO002-2.p	0.0	9.95	0.02	0.03	0.01	0.06
BOO024-1.p	0.0	0.22	0.01	0.02	0.02	0.06
BOO034-1.p	0.11	0.10	0.01	0.03	0.02	0.08
BOO067-1.p	0.33	12.43	0.02	0.07	31.82	41.12
BOO068-1.p	0.11	9.78	0.02	0.07	18.23	28.46
BOO069-1.p	0.11	2.75	0.02	0.04	0.11	0.67
BOO070-1.p	0.11	7.07	0.02	0.06	5.53	17.17
BOO071-1.p	0.11	3.29	0.02	0.05	0.45	2.18
BOO074-1.p	0.0	44.88	0.01	0.06		

../..

Problème	Note TPTP	Temps CiME	Temps Coq (court) réflexion	Temps Coq (court) tactiques	Temps Coq (long) réflexion	Temps Coq (long) tactiques
COL						
COL001-1.p	0.11	406.45	0.01	0.01	0.01	0.02
COL001-2.p	0.11	0.07	0.01	0.01	0.01	0.01
COL002-1.p	0.0	0.09	0.01	0.01	0.	0.
COL002-4.p	0.11	0.02	0.01	0.	0.01	0.01
COL003-12.p	0.22	1.98	0.01	0.02	0.	0.02
COL003-13.p	0.11	1.98	0.01	0.02	0.	0.01
COL003-14.p	0.11	2.00	0.	0.01	0.	0.01
COL003-16.p	0.11	1.98	0.	0.01	0.01	0.01
COL003-17.p	0.11	7.26	0.01	0.02	0.01	0.02
COL003-19.p	0.11	7.14	0.01	0.02	0.	0.01
COL004-3.p	0.22	0.02	0.01	0.	0.	0.01
COL007-1.p	0.0	0.00	0.01	0.	0.	0.
COL008-1.p	0.0	0.01	0.01	0.01	0.	0.
COL010-1.p	0.11	0.02	0.01	0.01	0.	0.01
COL012-1.p	0.0	0.00	0.01	0.	0.	0.
COL013-1.p	0.0	0.00	0.	0.01	0.	0.
COL014-1.p	0.0	0.01	0.	0.	0.	0.
COL015-1.p	0.0	0.01	0.01	0.01	0.01	0.
COL016-1.p	0.11	0.01	0.	0.	0.	0.
COL017-1.p	0.0	0.04	0.	0.	0.	0.01
COL018-1.p	0.0	0.00	0.01	0.	0.	0.
COL019-1.p	0.0	0.05	0.01	0.01	0.	0.
COL020-1.p	0.11	0.33	0.	0.01	0.	0.01
COL021-1.p	0.0	0.83	0.01	0.01	0.01	0.
COL022-1.p	0.0	0.09	0.01	0.01	0.01	0.
COL023-1.p	0.0	1.36	0.	0.01	0.01	0.01
COL024-1.p	0.0	0.60	0.01	0.	0.	0.
COL025-1.p	0.0	0.01	0.01	0.01	0.01	0.
COL026-1.p	0.11	2.35	0.01	0.02	0.	0.01
COL027-1.p	0.0	3.48	0.01	0.01	0.	0.
COL029-1.p	0.0	0.00	0.	0.	0.	0.01
COL030-1.p	0.11	22.79	0.01	0.01	0.	0.
COL031-1.p	0.0	0.84	0.01	0.01	0.	0.01
COL032-1.p	0.22	0.77	0.	0.02	0.	0.
COL033-1.p	0.11	96.02	0.01	0.01	0.01	0.
COL042-6.p	0.11	2.35	0.	0.02	0.01	0.01
COL042-8.p	0.11	2.33	0.01	0.01	0.	0.01
COL042-9.p	0.11	290.96	0.01	0.02	0.01	0.02
COL045-1.p	0.0	0.04	0.	0.	0.	0.
COL048-1.p	0.0	0.02	0.01	0.01	0.	0.
COL050-1.p	0.0	0.00	0.	0.	0.	0.
COL051-1.p	0.0	0.01	0.	0.01	0.	0.
COL052-1.p	0.0	0.01	0.01	0.01	0.	0.01
COL053-1.p	0.0	0.01	0.01	0.01	0.	0.
COL056-1.p	0.0	0.01	0.	0.01	0.	0.
COL058-1.p	0.0	0.06	0.01	0.01	0.	0.
COL058-2.p	0.0	0.02	0.01	0.01	0.	0.02
COL058-3.p	0.0	0.03	0.01	0.01	0.	0.01
COL059-1.p	0.0	0.06	0.01	0.03	0.	0.04
COL060-1.p	0.33	1.64	0.01	0.08	0.	0.01
COL060-2.p	0.0	0.02	0.01	0.	0.	0.
COL060-3.p	0.0	0.00	0.	0.	0.	0.
COL061-1.p	0.33	1.68	0.01	0.07	0.01	0.01
COL061-2.p	0.0	0.01	0.	0.01	0.01	0.)
COL061-3.p	0.0	0.02	0.	0.01	0.	0.

../..

Problème	Note TPTP	Temps CiME	Temps Coq (court) réflexion	tactiques	Temps Coq (long) réflexion	tactiques
COL062-1.p	0.44	449.84	0.01	0.11	0.	0.02
COL062-2.p	0.0	0.00	0.01	0.01	0.01	0.
COL062-3.p	0.0	0.01	0.01	0.01	0.	0.01
COL063-2.p	0.0	0.01	0.01	0.01	0.01	0.
COL063-3.p	0.0	0.02	0.	0.	0.	0.01
COL063-4.p	0.0	0.02	0.01	0.01	0.	0.01
COL063-5.p	0.0	0.01	0.01	0.	0.	0.01
COL063-6.p	0.0	0.01	0.01	0.01	0.	0.01
COL064-10.p	0.11	0.02	0.	0.01	0.01	0.01
COL064-11.p	0.0	0.01	0.01	0.01	0.	0.
COL064-2.p	0.11	0.01	0.01	0.	0.01	0.01
COL064-3.p	0.0	0.02	0.01	0.01	0.01	0.01
COL064-4.p	0.11	0.02	0.	0.01	0.	0.
COL064-5.p	0.11	0.01	0.01	0.01	0.	0.01
COL064-6.p	0.0	0.01	0.	0.01	0.	0.01
COL064-7.p	0.11	0.01	0.	0.01	0.	0.
COL064-8.p	0.0	0.01	0.01	0.01	0.	0.01
COL064-9.p	0.11	0.02	0.	0.01	0.	0.01
COL066-2.p	0.0	0.13	0.01	0.01	0.01	0.
COL066-3.p	0.11	0.14	0.01	0.01	0.	0.
COL070-1.p	0.11	0.62	0.	0.01	0.	0.01
COL075-2.p	0.0	254.02	0.	0.01	0.01	0.01
COL083-1.p	0.0	0.00	0.01	0.	0.	0.
COL084-1.p	0.0	0.01	0.01	0.	0.	0.
COL085-1.p	0.0	0.00	0.	0.	0.	0.
COL086-1.p	0.0	0.00	0.	0.	0.	0.01
GRP						
GRP001-2.p	0.0	0.01	0.01	0.01	0.	0.01
GRP001-4.p	0.0	0.01	0.	0.	0.	0.
GRP002-2.p	0.0	0.19	0.01	0.05	0.03	0.17
GRP002-3.p	0.11	0.28	0.01	0.06	0.05	0.3
GRP002-4.p	0.11	0.22	0.01	0.07	0.03	0.14
GRP010-4.p	0.0	0.01	0.01	0.	0.01	0.01
GRP011-4.p	0.0	0.03	0.01	0.01	0.01	0.01
GRP012-4.p	0.0	0.03	0.01	0.01	0.	0.01
GRP022-2.p	0.0	0.01	0.	0.	0.	0.
GRP023-2.p	0.0	0.01	0.01	0.01	0.	0.
GRP115-1.p	0.0	0.07	0.01	0.02	0.01	0.09
GRP116-1.p	0.0	1.68	0.01	0.04	2.05	3.82
GRP117-1.p	0.0	0.34	0.01	0.02	0.14	0.56
GRP118-1.p	0.0	1.90	0.	0.03	2.31	4.2
GRP195-1.p	0.0	0.09	0.	0.03	0.01	0.03
GRP200-1.p	0.33	11.80	0.01	0.08	0.51	2.64
GRP201-1.p	0.44	8.94	0.01	0.14	8.56	18.99
GRP202-1.p	0.44	13.53	0.01	0.08	0.49	2.24
GRP203-1.p	0.11	137.11	0.01	0.09		
GRP205-1.p	0.33	13.66	0.01	0.09	0.45	1.99
GRP206-1.p	0.0	0.02	0.01	0.	0.	0.01
GRP403-1.p	0.11	41.96	0.01	0.04	0.03	0.32
GRP406-1.p	0.0	2.03	0.02	0.07	0.12	1.6
GRP409-1.p	0.0	159.33	0.01	0.1	0.16	1.5
GRP412-1.p	0.0	21.52	0.02	0.1	6.74	67.94
GRP415-1.p	0.11	140.77	0.02	0.21		
GRP418-1.p	0.11	8.69	0.02	0.1	0.11	1.17
GRP421-1.p	0.22	23.11	0.02	0.09	0.09	1.24
GRP424-1.p	0.0	0.18	0.01	0.03	0.01	0.06

../..

Problème	Note TPTP	Temps CiME	Temps Coq (court) réflexion	Temps Coq (court) tactiques	Temps Coq (long) réflexion	Temps Coq (long) tactiques
GRP433-1.p	0.0	66.25	0.01	0.11		
GRP434-1.p	0.0	48.73	0.02	0.11		
GRP435-1.p	0.0	84.14	0.02	0.11		
GRP442-1.p	0.0	337.08	0.03	0.21		
GRP445-1.p	0.0	0.05	0.01	0.01	0.01	0.06
GRP446-1.p	0.0	0.44	0.01	0.03	0.11	0.6
GRP447-1.p	0.0	35.68	0.01	0.06		
GRP448-1.p	0.0	0.07	0.01	0.01	0.01	0.06
GRP449-1.p	0.0	0.44	0.01	0.03	0.1	0.66
GRP450-1.p	0.11	37.26	0.02	0.07		
GRP451-1.p	0.0	0.14	0.01	0.02	0.01	0.1
GRP452-1.p	0.11	98.54	0.02	0.14		
GRP454-1.p	0.0	0.01	0.01	0.	0.	0.
GRP455-1.p	0.0	0.02	0.01	0.01	0.01	0.01
GRP456-1.p	0.11	1.74	0.01	0.05	0.59	4.34
GRP457-1.p	0.0	0.01	0.01	0.	0.01	0.
GRP458-1.p	0.0	0.02	0.	0.01	0.01	0.01
GRP459-1.p	0.0	1.67	0.01	0.05	0.62	4.11
GRP460-1.p	0.0	0.00	0.01	0.	0.01	0.
GRP461-1.p	0.0	0.02	0.	0.01	0.01	0.02
GRP462-1.p	0.0	0.54	0.01	0.04	0.17	1.17
GRP463-1.p	0.0	0.01	0.01	0.	0.01	0.
GRP464-1.p	0.0	0.03	0.01	0.01	0.	0.02
GRP465-1.p	0.0	0.54	0.01	0.03	0.17	1.07
GRP466-1.p	0.0	0.04	0.01	0.01	0.	0.02
GRP467-1.p	0.0	0.24	0.01	0.02	0.01	0.08
GRP468-1.p	0.0	0.87	0.01	0.03	0.36	1.58
GRP481-1.p	0.0	0.02	0.	0.01	0.01	0.
GRP482-1.p	0.0	0.06	0.01	0.02	0.01	0.05
GRP483-1.p	0.0	0.63	0.01	0.04	0.15	1.39
GRP484-1.p	0.0	0.08	0.01	0.03	0.01	0.09
GRP485-1.p	0.0	1.47	0.01	0.03	1.47	4.2
GRP486-1.p	0.0	43.25	0.02	0.08		
GRP487-1.p	0.0	0.04	0.01	0.02	0.	0.03
GRP490-1.p	0.0	0.04	0.	0.01	0.01	0.03
GRP491-1.p	0.0	0.49	0.	0.03	0.13	0.88
GRP492-1.p	0.0	34.03	0.02	0.06		
GRP493-1.p	0.0	0.01	0.	0.	0.	0.
GRP494-1.p	0.0	0.06	0.01	0.02	0.01	0.06
GRP495-1.p	0.0	2.53	0.01	0.05	2.09	12.68
GRP496-1.p	0.0	0.01	0.01	0.01	0.	0.01
GRP497-1.p	0.0	0.37	0.01	0.03	0.08	0.52
GRP498-1.p	0.0	6.76	0.01	0.07	18.17	27.7
GRP517-1.p	0.0	0.06	0.01	0.02	0.01	0.07
GRP518-1.p	0.0	0.02	0.01	0.01	0.	0.01
GRP519-1.p	0.0	0.14	0.01	0.03	0.03	0.21
GRP521-1.p	0.0	0.02	0.01	0.02	0.	0.02
GRP522-1.p	0.0	0.06	0.01	0.01	0.01	0.04
GRP525-1.p	0.0	0.03	0.01	0.01	0.	0.01
GRP526-1.p	0.0	0.03	0.01	0.01	0.	0.02
GRP529-1.p	0.0	0.03	0.01	0.01	0.	0.02
GRP530-1.p	0.0	0.02	0.01	0.01	0.01	0.01
GRP533-1.p	0.0	0.01	0.01	0.	0.	0.
GRP534-1.p	0.0	0.01	0.01	0.	0.	0.
GRP537-1.p	0.0	0.01	0.01	0.01	0.	0.01
GRP538-1.p	0.0	0.02	0.01	0.01	0.	0.

../..

Problème	Note TPTP	Temps CiME	Temps Coq (court) réflexion	Temps Coq (court) tactiques	Temps Coq (long) réflexion	Temps Coq (long) tactiques
GRP541-1.p	0.0	0.00	0.	0.01	0.	0.01
GRP542-1.p	0.0	0.02	0.01	0.01	0.	0.
GRP545-1.p	0.0	0.00	0.01	0.	0.	0.
GRP546-1.p	0.0	0.01	0.01	0.01	0.	0.
GRP549-1.p	0.0	0.01	0.01	0.	0.	0.
GRP554-1.p	0.0	0.97	0.01	0.04	0.26	2.2
GRP555-1.p	0.0	0.07	0.01	0.01	0.01	0.07
GRP558-1.p	0.0	0.15	0.	0.02	0.03	0.21
GRP562-1.p	0.0	0.41	0.01	0.03	0.08	0.63
GRP565-1.p	0.0	0.04	0.01	0.02	0.	0.01
GRP566-1.p	0.0	0.06	0.	0.02	0.	0.05
GRP569-1.p	0.0	0.03	0.01	0.01	0.	0.01
GRP570-1.p	0.0	0.05	0.01	0.02	0.01	0.04
GRP573-1.p	0.0	0.03	0.	0.01	0.	0.01
GRP574-1.p	0.0	7.54	0.01	0.05	28.98	
GRP577-1.p	0.0	0.13	0.01	0.03	0.02	0.15
GRP578-1.p	0.0	42.05	0.01	0.1		
GRP581-1.p	0.0	0.04	0.	0.01	0.01	0.
GRP582-1.p	0.0	2.30	0.01	0.06	1.95	12.78
GRP589-1.p	0.0	7.69	0.01	0.08	2.5	29.82
GRP590-1.p	0.0	9.63	0.02	0.09	2.99	35.85
GRP593-1.p	0.0	0.42	0.01	0.04	0.07	0.7
GRP594-1.p	0.0	0.41	0.01	0.03	0.08	0.79
GRP597-1.p	0.0	2.88	0.01	0.05	0.04	0.62
GRP602-1.p	0.11	28.06	0.02	0.1		
GRP603-1.p	0.11	3.95	0.02	0.05	7.01	11.54
GRP610-1.p	0.11	1.60	0.01	0.03	0.51	3.40
GRP611-1.p	0.0	0.57	0.01	0.03	0.12	1.03
GRP613-1.p	0.0	28.67	0.01	0.09	1.91	19.1
GRP614-1.p	0.0	44.93	0.02	0.12		
LAT						
LAT014-1.p	0.0	0.01	0.	0.	0.	0.
LAT088-1.p	0.0	0.03	0.01	0.	0.	0.01
LAT090-1.p	0.0	0.04	0.01	0.01	0.	0.03
LCL						
LCL110-2.p	0.0	0.13	0.	0.03	0.01	0.18
LCL111-2.p	0.22	203.75	0.01	0.04	0.12	1.19
LCL112-2.p	0.0	0.12	0.01	0.03	0.01	0.18
LCL113-2.p	0.0	0.21	0.01	0.03	0.02	0.2
LCL114-2.p	0.0	0.41	0.01	0.04	0.06	0.73
LCL115-2.p	0.0	0.12	0.01	0.03	0.02	0.19
LCL116-2.p	0.0	1.05	0.01	0.03	0.3	2.33
LCL132-1.p	0.0	0.02	0.	0.01	0.	0.
LCL134-1.p	0.0	0.01	0.01	0.	0.	0.01
LCL135-1.p	0.0	0.03	0.	0.	0.	0.
LCL139-1.p	0.0	0.09	0.01	0.03	0.01	0.08
LCL140-1.p	0.0	0.11	0.	0.03	0.02	0.18
LCL141-1.p	0.0	0.40	0.01	0.03	0.06	0.72
LDA						
LDA001-1.p	0.0	0.05	0.01	0.03	0.	0.03
LDA002-1.p	0.0	97.89	0.01	0.09	0.06	0.29
LDA007-3.p	0.0	0.01	0.01	0.01	0.01	0.
SYN						
SYN080-1.p	0.0	0.00	0.01	0.	0.	0.
SYN083-1.p	0.0	0.01	0.	0.	0.01	0.

Bibliographie

[AH76] K. Appel et W. Haken. A proof of the four color theorem. *Discrete Mathematics*, 16 :179–180, 1976.

[Alv02] Cuihtlauac Alvarado. *Réflexion pour la réécriture dans le calcul des constructions inductives*. Thèse de doctorat, Université Paris-Sud, décembre 2002.

[AN00] Cuihtlauac Alvarado et Quang-Huy Nguyen. Elan for equational reasoning in coq. In *Proceeding of the 2nd Workshop on Logical Frameworks and Metalanguages*, Santa Barbara, Californie, 2000.

[Bar84] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North Holland, Amsterdam, 2ème édition, 1984.

[BC04] Yves Bertot et Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer-Verlag, 2004.

[BDP89] Leo Bachmair, Nachum Dershowitz et David A. Plaisted. Completion without failure. In H. Aït-Kaci et M. Nivat, éditeurs, *Resolution of Equations in Algebraic Structures 2 : Rewriting Techniques*, chapitre 1, pages 1–30. Academic Press, New York, 1989.

[BHdN02] Marc Bezem, Dimitri Hendriks et Hans de Nivelle. Automated proof construction in type theory using resolution. *Journal of Automated Reasoning*, 29(3) :253–275, 2002.

[BK92] G. Bellin et J. Ketonen. A decision procedure revisited : Notes on direct logic, linear logic and its implementation. *Theoretical computer science*, 95(1) :115–142, 1992.

[Bou97] Samuel Boutin. Using reflection to build efficient and certified decision procedures. In *Third International Symposium on Theoretical Aspects of Computer Software*, volume 1281 de *Lecture Notes in Computer Science*, pages 515–528, 1997.

[BT00] Leo Bachmair et Ashish Tiwari. Abstract congruence closure and specializations. In D. A. McAllester, éditeur, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 de *Lecture Notes in Artificial Intelligence*, pages 64–78, Pittsburgh, Pennsylvanie, 2000. Springer-Verlag.

[BTV03] L. Bachmair, A. Tiwari et L. Vigneron. Abstract congruence closure. *Journal of Automated Reasoning*, 31(2) :129–168, 2003.

[CAD05] *20th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Tallinn, Estonie, juillet 2005. Springer-Verlag.

[CC05] Evelyne Contejean et Pierre Corbineau. Reflecting proofs in first-order logic with equality. In CAD05 [CAD05].

[CK04] Sylvain Conchon et Sava Krstić. Strategies for combining decision procedures. *Theoretical Computer Science*, 2004. Special Issue of TCS dedicated to a refereed selection of papers presented at TACAS'03.

[CMU04] Evelyne Contejean, Claude Marché et Xavier Urbain. CiME3, 2004. `http://cime.lri.fr/`.

[Con04] Evelyne Contejean. A certified AC matching algorithm. In van Oostrom [vO04], pages 70–84.

[Coq] The Coq Proof Assistant. `http://coq.inria.fr/`.

[Coq85] Thierry Coquand. *Une Théorie des Constructions*. Thèse de doctorat, Université Paris 7, 1985.

[Coq00] Catarina Coquand. Agda, 2000. `http://www.cs.chalmers.se/~catarina/agda/`.

[Coq04] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.0*, avril 2004. `http://coq.inria.fr`.

[Cor01] P. Corbineau. Autrour de la clôture de congruence avec Coq. Mémoire de DEA, Université Paris 7, septembre 2001.

[Cré01] Pierre Crégut. Une procédure de décision réflexive pour l'arithmétique de Presburger en Coq. Deliverable, Projet RNRT Calife, 2001.

[dB72] Nikolas. G. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. of the Koninklijke Nederlands Akademie*, 75(5) :380–392, 1972.

[dB80] Nikolas G. de Bruijn. A survey of the project Automath. In *To H.B. Curry : Essays on Combinatory Logic, Lambda-Calculus and Formalism*. Academic Press, 1980.

[Del00] D. Delahaye. A tactic language for the system Coq. In *Proceedings of Logic for Programming and Automated Reasoning*, volume 1955 de *LNCS/LNAI*, pages 85–95. Springer, novembre 2000.

[dlB59] Renée de la Briandais. File searching using variable length keys. In *Proc. WJCC, AFIPS*, volume 15, Montvale, New Jersey, 1959. AFIPS Press.

[DN00] R. Dyckhoff et S. Negri. Admissibility of structural rules for contraction-free systems of intuitionistic logic. *Journal of Symbolic Logic*, 65 :1499–1518, décembre 2000.

[Dra87] A. G. Dragalin. *Mathematical Intuitionism : Introduction to Proof Theory*, volume 67 de *Translations of Mathematical Monographs*. American Mathematical Society, Providence, Rhode Island, 1987.

[DST80] P. J. Downey, R. Sethi et R. E. Tarjan. Variations on the common subexpressions problem. *Journal of the ACM*, 27(4) :771–785, 1980.

[Dyc92] Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *The Journal of Symbolic Logic*, 57(3) :795–807, 1992.

[Fil95] J.-C. Filliâtre. A decision procedure for direct predicate calculus : study and implementation in the Coq system. Rapport technique 96–25, LIP, ENS Lyon, février 1995.

[Fre03] G. Frege. *Grundgesetze der Arithmetik, Begriffsschriftlich abgeleitet.* Verlag Hermann Pohle, Jena, 1893–1903.

[Gen34] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, (39) :176–210, 405–431, 1934.

[Gie05] Jürgen Giesl, éditeur. *Term Rewriting and Applications*, volume 3467 de *Lecture Notes in Computer Science*, Nara, Japon, avril 2005. Springer-Verlag.

[Gir71] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, éditeur, *Proceedings 2nd Scandinavian Logic Symp., Oslo, Norvège, 18–20 juin 1970*, volume 63 de *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, Amsterdam, 1971.

[GL02] Benjamin Grégoire et Xavier Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.

[Göd34] K. Gödel. On undecidable propositions of formal mathematical systems ; in : Collected Works, 1934.

[Gon] G. Gonthier. A computer-checked proof of the four color theorem. `http://research.microsoft.com/~gonthier/4colproof.pdf`.

[Har95] John Harrison. Metatheory and reflection in theorem proving : A survey and critique. Rapport technique CRC-053, SRI Cambridge, Millers Yard, Cambridge, Royaume-Uni, 1995. `http://www.cl.cam.ac.uk/users/jrh/papers/reflect.dvi.gz`.

[Hen02] Dimitri Hendriks. Proof Reflection in Coq. *Journal of Automated Reasoning*, 29(3-4) :277–307, 2002.

[Hen03] D. Hendriks. *Metamathematics in Coq.* Thèse de doctorat, Department of Philosophy, Utrecht University, 2003.

[Hey56] A. Heyting. *Intuitionism : An Introduction.* North-Holland, Amsterdam, 1956.

[HL02] T. Hillenbrand et B. Löchner. The next waldmeister loop. In A. Voronkov, éditeur, *Proceedings of the 18th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 486–500. Springer-Verlag, 2002.

[How80] William A. Howard. The formulae-as-types notion of construction. In J. Roger Hindley Jonathan P. Seldin, éditeur, *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Londres, 1980.

[HR87] Jieh Hsiang et Michaël Rusinowitch. On word problems in equational theories. In Ottmann [Ott87], pages 54–71.

[Hud93] J. Hudelmaier. An $O(n \log n)$-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1) :63–75, 1993.

[Hue97] Gérard Huet. The Zipper. *Journal of Functional Programming*, 7(5) :549–554, 1997. Functional Pearl.

[Isa] The ISABELLE system. `http://isabelle.in.tum.de/`.

[Kle52] S. C. Kleene. *Introduction to metamathematics*, volume I de *Bibliotheca Mathematica*. North-Holland, Amsterdam, 1952.

[KO99] C. Kreitz et J. Otten. Connection-based theorem proving in classical and non-classical logi cs. *Journal of Universal Computer Science*, 5(3) :88–112, 1999.

[Kre02] C. Kreitz. *The Nuprl Proof Development System, Version 5*, décembre 2002.

[KvE76] Robert A. Kowalski et M. H. van Emden. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23 :733–742, octobre 1976.

[KW84] J. Ketonen et R. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32 :297–307, 1984.

[Leg] The LEGO proof assistant. `http://www.dcs.ed.ac.uk/home/lego/`.

[Mar47] A. A. Markov. On the impossibility of certain algorithms in the theory of associative systems. *Doklady Akademii Nauk SSSR*, 55(7) :587–590, 1947. en russe, traduction en anglais dans C.R. Acad. Sci. URSS, 55, 533–586.

[McB99] Conor McBride. *Dependently Typed Functional Programs and their Proofs*. Thèse de doctorat, University of Edinburgh, 1999. `http://www.lfcs.informatics.ed.ac.uk/reports/00/ECS-LFCS-00-419/`.

[ML84] Per Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

[Mor68] D. R. Morrison. PATRICIA - practical algorithm to retrieve coded in alphanumeric. *Journal of the ACM*, 15(4) :514–534, 1968.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper et David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, mai 1997.

[Muñ95] César Augusto Muñoz Hurtado. Démonstration automatique dans la logique propositionnelle intuitionniste. Mémoire de DEA, Université Paris 7, INRIA Rocquencourt, 1995.

[NO79] G. Nelson et D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming, Languages and Systems*, 1(2) :245–257, octobre 1979.

[NO80] G. Nelson et D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27 :356–364, 1980.

[NO05] Robert Nieuwenhuis et Albert Oliveras. Proof-Producing Congruence Closure. In Jürgen Giesl, éditeur, *Proceedings of the 16th International Conference on Term Rewriting and Applications, RTA'05 (Nara, Japan)*, volume 3467 de *Lecture Notes in Computer Science*, pages 453–468. Springer, 2005.

[NR01] Robert Nieuwenhuis et Albert Rubio. *Handbook of Automated Reasoning*, chapitre Paramodulation-Based Theorem Proving, pages 371–444. Elsevier Science Publishers, 2001.

[Obj] The Objective Caml language. `http://www.ocaml.org/`.

[Opp78] D. C. Oppen. Reasoning about recursively defined data structures. *Journal of the ACM*, 1978. Version préliminaire dans *Proceedings 5th Symposium on Principles of Programming Language*.

[Ott87] Thomas Ottmann, éditeur. *14th International Colloquium on Automata, Languages and Programming*, volume 267 de *Lecture Notes in Computer Science*, Karlsruhe, Allemagne, juillet 1987. Springer-Verlag.

[Pea89] Guiseppe Peano. The principles of arithmetic, presented by a new method. In van Heijenoort [vH67], pages 83–97.

[Pet83] Gerald E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal on Computing*, 12(1) :82–100, février 1983.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard huet et Gordon D. Plotkin, éditeurs, *Logical Frameworks*. Cambridge University Press, 1991.

[Pos47] Emil L. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 13 :1–11, 1947.

[PS00] Frank Pfenning et Carsten Schuermann. *Twelf User's Guide, version 1.3*, 2000. `http://www.cs.cmu.edu/~twelf/guide/twelf_toc.html`.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, 1965.

[ROK05] Thomas Rath, Jens Otten et Christoph Kreitz. The iltp library : Benchmarking theorem provers for intuitionistic logic. Rapport technique, Université de Potsdam, 2005. `http://www.cs.uni-potsdam.de/ti/iltp/`.

[RW69] G. Robinson et L. T. Wos. Paramodulation and first-order theorem proving. In B. Meltzer et D. Mitchie, éditeurs, *Machine Intelligence 4*, pages 135–150. Edinburgh University Press, 1969.

[Sho82] R.E. Shostak. Deciding combinations of theories. Rapport technique CSL 132, SRI International, février 1982.

[Sho84] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31 :1–12, 1984.

[SLKN01] S. Schmitt, L. Lorigo, C. Kreitz et A. Nogin. Integrating connection-based theorem proving into interactive proof assistants. In R. Gore, A. Leitsch et T. Nipkow, éditeurs, *Proceedings of International Joint Conference on Automated Reasoning*, volume 2083 de *LNAI*, pages 421–426. Springer, 2001.

[SS98] G. Sutcliffe et C.B. Suttner. The TPTP Problem Library : CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2) :177–203, 1998.

[Sta79] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9(1) :67–72, juillet 1979.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 de *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003. `http://www.cs.vu.nl/~terese/`.

[TS96] A. S. Troesltra et H. Schwichtenberg. *Basic Proof Theory*, volume 43 de *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1996.

[vH67] Jean van Heijenoort, éditeur. *From Frege to Gödel*. Harvard University Press, 1967.

[vO04] Vincent van Oostrom, éditeur. *15th International Conference on Rewriting Techniques and Applications*, volume 3091 de *Lecture Notes in Computer Science*, Aix-la-Chapelle, Allemagne, juin 2004. Springer-Verlag.

[Vor52] N. N. Vorob'ev. The derivability problem in the constructive propositional calculus with strong negation. 85 :689–92, 1952.

[Vor58] N. Vorobjev. New derivability algorithm in the constructive propositional calculus. (en russe). In *Proceedings of the Steklov institute of Mathematics (Trudy)*, volume 52, pages 193–225, 1958.

[Wer94] Benjamin Werner. *Méta-théorie du Calcul des Constructions Inductives*. Thèse de doctorat, Université Paris 7, 1994.

[WR10] Alfred North Whitehead et Bertrand Russell. *Principia mathematica*. Cambridge University Press, Cambridge, 1910.

Index

Table des figures

Résumé

Les logiciels d'aide à la démonstration se répartissent entre prouveurs automatiques et assistants de preuve interactifs. Les premiers sont des outils spécialisés dont les techniques sont éloignées des méthodes intuitives de raisonnement mais dont la portée est faible. Les seconds ont un champ d'application plus large mais l'utilisateur doit expliciter tous les détails de la démonstration. L'automatisation des preuves y est difficile car la logique utilisée est plus expressive.

La thèse se compose de trois contributions dans ce domaine. D'abord, nous présentons une extension de la clôture de congruence à une théorie des constructeurs avec application partielle. Un algorithme pour résoudre ce problème est décrit et étudié. Ensuite nous construisons un formalisme du premier ordre incluant des connecteurs définis comme des types inductifs non récursifs dans l'esprit du Calcul des Constructions Inductives. Nous présentons un nouveau calcul de séquents sans contraction pour la logique intuitionniste du premier ordre, adapté à ce formalisme, et prouvons ses propriétés fondamentales : élimination des contractions et des coupures. Nous en dérivons une procédure de semi-décision. Ces deux premières contributions sont implantées dans l'assistant de preuve Coq ; la troisième est la description d'une méthode d'interprétation des preuves au premier ordre dans la Théorie des Types grâce à la réflexion calculatoire. Ce paradigme est appliqué à la logique propositionnelle et utilisé par une procédure de décision du système Coq. Cette méthode est aussi étendue au premier ordre avec égalité et permet de vérifier des preuves engendrées par complétion ordonnée dans le système CiME.
Mots-clés : Démonstration automatique, Procédure de décision, Réflexion calculatoire, Système Coq, Théorie des Types, Calcul des Constructions Inductives, Logique intuitionniste, Logique du premier ordre

Abstract

Among software designed to help formal reasoning, there are automated theorem provers and interactive proof assistants. The former are specialized tools using technics that differ from traditionnal reasoning methods, but they have a very limited range. The latter have a wider range but their users must specify explicitly every proof step. Automation of proofs is tricky in these latter tools because they use expressive logical systems.

This thesis is build around three contributions in this field. First, we present an extension of congruence-closure the a theory of constructors with partial application. An algorithm to solve this problem is described and studied. Then, we build a first-order formalism including connectives defined as non-recursive inductive types similar to those in the Calculus of Inductive Constructions. We present a new contraction-free sequent calculus for first-order intuitionistic logic that is adapted to this formalism, and we prove its fundamental properties : contraction- and cut-elimination. We derive a semi-decision procedure from these results. Those two constributions are implemented inside the Coq proof assistant ; the third one is the description of a methode allowing the interpretation of first-order proof tree into Type Theory by computational reflection. This paradigm is applied to propositionnal logic and is used by a decision procedure inside the Coq system. This method is also adapted to first-order logic with equality and allows to check proofs obtained by ordered completion in the CiME system.
Keywords : Automated Reasoning, Decision Procedure, Computational Reflection, Coq proof-assistant, Type Theory, Calculus of Inductive Constructions, Intuitionistic Logic, First-order Logic