UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Modélisation logique de la langue et Grammaires Catégorielles Abstraites

THÈSE

présentée et soutenue publiquement le

pour l'obtention du

Doctorat de l'Université de Lorraine

(Informatique)

par

Florent Pompigne

Composition du jury

Rapporteurs :	Christian Retoré	Pr, LaBRI, Université de Bordeaux 1, Talence, France
	Annie Foret	MCF HDR, Université de Rennes 1, France
Examinateurs :	Michael Moortgat	Pr, Utrecht Institute of Linguistics OTS, Pays-bas
	Laurent Vigneron	Pr, Université de Lorraine, Nancy, France
	Philippe de Groote (Directeur)	DR, INRIA, Nancy, France
	Sylvain Pogodalla	CR, INRIA, Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Sommaire

Introduction

Le traitement automatique de la langue (TAL) est une discipline au carrefour de la linguistique, de l'informatique et des mathématiques, qui a pour ambition d'obtenir des programmes capables de manipuler les langues naturelles. À cet effet, l'approche des méthodes symboliques, par contraste avec les méthodes statistiques, est d'expliciter les règles gouvernant la formation et la compréhension des énoncés de ces langues naturelles. La linguistique étudie les langues à différents niveaux, notamment celui des phonèmes (phonologie), des mots (morphologie), des phrases (syntaxe), de la signification (sémantique) et du contexte d'énonciation (pragmatique), et chacun de ces niveaux engendre son lot de problématiques quant aux façons de capturer son comportement et ses interactions avec les autres niveaux. Dans le présent travail, nous nous intéressons aux niveaux de syntaxe et de sémantique et à l'interface qu'on peut établir entre eux. Il s'agit donc de déterminer, pour une langue naturelle donnée, comment les unités lexicales (nous prendrons parfois le raccourci de parler de mots dans la suite de cette thèse) peuvent se combiner pour former des phrases, et comment ces façons de combiner les mots influent sur le sens des phrases résultantes. Une modélisation fine de la syntaxe, de la sémantique et de leur interface a notamment comme applications possibles l'amélioration de la qualité de la traduction automatique, de la génération de texte, du résumé automatique et des agents conversationnels.

L'ensemble des règles permettant de modéliser un ou plusieurs niveaux d'une langue constitue une grammaire formelle. Ces règles peuvent prendre plusieurs formes, et suivre différentes stratégies, pour capturer les mêmes phénomènes d'une même langue naturelle, et le domaine des méthodes symboliques comprend donc différents formalismes grammaticaux, qui se distinguent entre autres par leur couverture, leur simplicité, la facilité et l'efficacité de leur implémentation, ou la finesse de leur analyse. Notre travail s'inscrit dans la tradition des grammaires catégorielles, qui associent à chaque mot une catégorie et identifient les arbres de dérivation syntaxique à des arbres de preuve logique. Plus précisément, nous nous basons sur les grammaires catégorielles abstraites (ACG, pour Abstract Categorial Grammars), introduites par de Groote (2001).

Les ACG ont pour ambition d'abstraire au maximum ces processus de traitement de la langue, rendus nombreux par la multiplication des niveaux linguistiques à modéliser et des formalismes grammaticaux qui s'y attachent. Les définitions des ACG ne sont ainsi liées à aucun concept linguistique particulier, mais reposent plutôt sur la manipulation de termes de λ-calcul (ou λ-termes). Ces termes sont en effet capables d'encoder différents niveaux linguistiques, tels que les formes syntaxiques de surface (sous forme de chaînes de caractères), les structures de dérivations syntaxiques (les λ-termes pouvant eux-mêmes être identifiés à des arbres) ou les formules sémantiques. Ces différents niveaux linguistiques peuvent alors être mis en relation par le *lexique* dont est constituée toute ACG. L'absence de présupposés linguistiques dans les définitions des ACG leur permet d'encoder de nombreux autres formalismes grammaticaux. L'utilisation du λ-calcul et de la théorie des types, objets mathématiques bien étudiés, comme fondements de ces grammaires facilite l'étude de leurs propriétés, comme la complexité de leur analyse ou la classe des langages qu'elles engendrent, et par conséquent l'étude des propriétés des formalismes encodés.

Nous explorons dans ce travail les possibilités ouvertes par la propriété des ACG de manipuler explicitement les structures de dérivation dans la modélisation de l'interface syntaxe-sémantique. En particulier, nous examinons les problématiques de mouvements linguistiques, explicites (constructions relatives et interrogatives) et implicites (prises de portée de constituants quantificateurs), à travers quelques exemples des langues française et anglaise, et nous proposons une modélisation de ces phénomènes qui détecte les constructions incorrectes au niveau des structures de dérivation, plutôt que des formes de surface. Ce travail ne prétend ni être exhaustif quant à l'étendue des phénomènes de mouvement examinés, ni même être exact dans le fonctionnement précis de ces phénomènes : déterminer leurs frontières précises est une question linguistique pointue, et nous nous intéressons seulement à la question du *comment*.

Afin de contrôler ce mécanisme de mouvements linguistiques, nous explorons la possibilité d'ajouter aux ACG

des structures de traits, construction qui se retrouve dans de nombreux formalismes grammaticaux, permettant d'ajouter à la catégorie syntaxique d'un constituant des informations telles que son genre, son nombre, ou, comme nous le montrerons, la présence de mouvements en cours. La solution naïve pour exprimer ces structures de traits dans le système de types des ACG est de démultiplier le nombre de types pour chaque combinaison d'informations possibles. Les assignations de types aux unités lexicales seraient alors également démultipliées, à l'exemple d'un verbe intransitif comme "mange" qui devrait, ne serait-ce que vis-à-vis du trait dénotant le genre, disposer d'un type lui permettant d'avoir un sujet masculin et d'un type lui permettant d'avoir un sujet féminin. Cette solution n'est pas seulement inélégante, elle va contre l'ambition des ACG d'abstraire et de factoriser les données linguistiques. De plus, en échouant à capturer ces factorisations, elle abandonne toute possibilité d'optimiser l'impact de ces structures de traits sur le problème de l'analyse syntaxique dans les ACG. Une autre solution, proposée par de Groote et Maarek (2007), consiste à exprimer ces structures de traits en étendant le système de type des ACG, avec les constructions de la somme disjointe (permettant les analyses de cas), du produit cartésien (permettant aux structures de traits de contenir plusieurs champs) et du produit dépendant (permettant aux catégories syntaxiques de dépendre d'une structure de traits). Notre travail est une poursuite de leur proposition. Notre contribution est de démontrer la pertinence de cette extension du système de typage pour capturer les phénomènes de mouvements, mais également de donner les preuves mathématiques des propriétés fondamentales du calcul résultant de cette extension, notamment sa normalisation (terminaison) et sa confluence (indifférence du résultat selon l'ordre d'évaluation).

Le λ-calcul, sur lequel se basent les ACG, est un système formel qui manipule l'essence des notions de fonction et d'application. Le λ-calcul peut être muni d'un système de typage, garantissant pour les termes correctement typés des propriétés, appréciables dans le cadre d'une implémentation, telles que la normalisation forte. Dans le cadre des ACG, le λ-calcul utilisé est linéaire, ce qui signifie que les variables des fonctions (les variables λ-abstraites) apparaissent une et une seule fois dans l'expression du corps de la fonction. Cette propriété permet de capturer la sensibilité aux ressources des combinaisons syntaxiques : par exemple, un verbe conjugué ne peut avoir qu'un et un seul sujet. Avec l'extension du système de typage, ce λ-calcul linéaire se voit compléter par des constructions d'analyse de cas, d'enregistrement et de types dépendants de termes. Ces constructions ne sont pas nouvelles, et l'impact de l'ajout de chacune d'entre elles sur les propriétés du calcul a déjà été étudié. Cependant, l'interaction entre la somme disjointe et le produit dépendant pose pour les preuves de confluence et de normalisation forte des problèmes complexes qui n'ont pas encore été entièrement mis à plat. Les preuves que nous donnons dans le présent travail, bien qu'incomplètes et portant sur une version légèrement limitée du produit dépendant, peuvent donc présenter un intérêt allant au-delà du domaine du TAL.

Cette thèse est structurée de la façon suivante :

Dans le chapitre 1, *Grammaires Catégorielles*, nous présentons les idées directrices des grammaires catégorielles, et en particulier des grammaires catégorielles abstraites. Dans un premier temps, nous donnons une brève introduction historique des grammaires catégorielles, en pointant notamment les problématiques liées aux mouvements linguistiques rencontrées par ces formalismes. Nous montrons également comment obtenir une interface syntaxe-sémantique dans cette tradition. Dans une seconde partie, nous donnons les définitions formelles des grammaires catégorielles abstraites, ainsi que des exemples d'utilisation linguistique et une illustration de l'extension du système de typage.

Dans le chapitre 2, *β-réductions et conversions permutatives dans le $\lambda\Pi$-calcul linéaire avec produit et somme*, nous établissons formellement le λ-calcul résultant de l'extension du système de typage, à travers ses notations, ses règles de réécriture et son système de typage. De plus, dans un second temps, nous démontrons des premiers résultats de confluence et de normalisation forte, lorsque les réductions considérées sont restreintes à la β-réduction, et à la γ-conversion (permutations liées aux analyses de cas). Pour obtenir la normalisation forte, nous introduisons un calcul plus simple, pour lequel la propriété est connue, et nous y projetons notre calcul d'une façon qui conserve les étapes de réduction.

Dans le chapitre 3, *Propriétés fondamentales de $\longrightarrow_{\beta\gamma\eta}$ dans le $\lambda\Pi$-calcul linéaire avec produit et somme*, nous nous attachons à étendre ces propriétés de normalisation et de confluence dans le cas où l'η-conversion (qui permet d'identifier les termes extensionnellement équivalents) est considérée dans les règles de réduction. Cette partie de la preuve est séparée de la précédente car elle concentre les difficultés théoriques, et c'est dans ce chapitre

que la preuve proposée devient incomplète, dépendant en l'état d'hypothèses qui, bien que limitées, restent à être démontrées.

Dans le chapitre 4, *Modélisation des dépendances à distance*, nous présentons comment le calcul examiné dans les deux chapitres précédents peut être utilisé dans les ACG pour contrôler les dépendances à distance, c'est à dire les mouvements linguistiques. Nous établissons dans un premier temps les idées générales d'une interface syntaxe-sémantique traitant les dépendances à distance dans les ACG, puis nous examinons différents problèmes réputés difficiles à modéliser : îlots d'extraction pour les mouvements explicites, pour les mouvements implicites, et contraintes de formation des extractions interrogatives multiples, et nous montrons comment éliminer les phrases incorrectes au niveau des structures de dérivation.

Dans le chapitre 5, *Comparaison aux autres approches*, nous confrontons la stratégie détaillée au chapitre précédent, dans son utilisation de termes d'ordre supérieur pour procéder aux extractions et dans son utilisation d'un contrôle sur les structures de dérivation pour éliminer la surgénération, aux approches de formalismes grammaticaux comparables aux ACG.

Chapitre 1

Grammaires Catégorielles

Ce chapitre présente les idées directrices qui ont motivé l'élaboration des grammaires catégorielles, et en particulier des Grammaires Catoégorielles Abstraites (ACG, pour Abstract Categorial Grammars). L'ambition de ces grammaires est de transcrire la notion linguistique de catégorie syntaxique d'un mot (*part of speech*) au sein d'un formalisme mathématique, ce qui permet de disposer d'un calcul formel pour la syntaxe d'une langue naturelle. Nous verrons de plus qu'une telle approche permet également de manipuler la sémantique de la langue.

Dans une première partie nous présenterons les concepts fondateurs des grammaires catégorielles à travers une introduction aux grammaires AB et aux grammaires de Lambek, en observant notamment la problématique des extractions grammaticales dans ces formalismes. Nous montrerons également comment les calculs syntaxiques de ce type de formalisme peuvent être utilisés pour en dériver des calculs sémantiques.

Dans une seconde partie, nous introduirons le formalisme des grammaires catégorielles abstraites à travers leur définition, un exemple d'utilisation, leurs propriétés, une discussion sur les architectures qu'elles peuvent former et une discussion sur les extensions possibles de ces grammaires, extensions qui sont au cœur des discussions des chapitres suivants.

1.1 Grammaires catégorielles classiques

Cette introduction aux grammaires catégorielles suit la présentation de Moot et Rétoré (2012).

1.1.1 Grammaires AB

Les grammaires AB sont définies par Bar-Hillel (1953). Elles reprennent une idée developpée par Adjukiewicz (1935) : attribuer à chaque composant d'un langage donné une fraction indiquant comment ce composant doit être combiné. On peut alors tester si une expression est syntaxiquement correcte en calculant le produit de l'ensemble des fractions impliquées, selon des règles de simplification faisant écho aux règles de simplifications habituelles de l'arithmétique. En ce sens on peut donc rapprocher cette idée de l'analyse dimensionnelle pratiquée en physique, grâce à laquelle on peut vérifier qu'une expression est bien de la dimension attendue par un calcul sur les dimensions de chaque constituant de l'expression.

Ajdukiewicz s'intéressait principalement aux langages formels, dans lesquels les liens entre constituants ne sont en général pas ambigus, grâce notamment à l'usage des parenthèses. Cependant, dans beaucoup de langues naturelles, à l'exemple du français et de l'anglais, les liens entre constituants ne sont pas triviaux, peuvent être ambigus, et sont suggérés par l'ordre dans lequel ces constituants sont agencés. Pour traiter la syntaxe de ces langues naturelles, Bar-Hillel propose donc d'orienter les fractions. On peut ainsi par exemple construire une catégorie pour les verbes intransitifs, qui nécessitent un sujet à leur gauche, et pour les déterminants, qui nécessitent un nom à leur droite.

Plus formellement, l'ensemble C des catégories des grammaires AB se définit comme :

$$C ::= B \mid C/C \mid C\backslash C$$

où B est un ensemble de catégories de base. Dans les exemples qui suivront, nos catégories de base seront S (pour *sentence*), N (pour *noun*) et NP (pour *noun phrase*).

Un lexique associe à des entités lexicales une liste de catégorie. La raison pour laquelle une même entité lexicale peut se voir attribuer plusieurs catégories est qu'un même mot peut être utilisé avec différents rôles syntaxiques.

On peut alors déterminer si une expression, c'est-à-dire une séquence de mots $m_1 \cdots m_n$, a pour catégorie c. Il faut pour celà sélectionner pour chaque mot m_i une catégorie c_i qui lui est associée, et vérifier que $c_1 . \cdots . c_n$ se réduit en c par le moyen des règles suivantes :

$$\begin{array}{ll} a.(a\backslash b) & \rightarrow b \\ (a/b).b & \rightarrow a \end{array}$$

Ces règles peuvent être interprétées comme des simplifications de fractions pour un opérateur . non commutatif. Cette non commutativité traduit bien que l'interversion de deux mots n'est pas anodine : l'ordre des mots est crucial pour la syntaxe des langues naturelles que nous considérons. a/b correspond ainsi à la catégorie syntaxique des expressions qui forment une expression de catégorie a si on lescomplète avec une expression de catégorie b à leur droite, et symétriquement $a\backslash b$ correspond à la catégorie des expressions qui forment un b si on leur ajoute un a à gauche.

Une grammaire AB est alors définie comme le quadruplet constitué d'un ensemble de catégories syntaxiques de base, d'un vocabulaire de mots, d'un lexique qui associe à chaque mot une ou plusieurs catégorie, et d'une catégorie particulière, appelée catégorie distinguée. On peut alors définir le langage d'une telle grammaire comme l'ensemble des séquences de mots auxquelles on peut attribuer la catégorie distinguée, qui correspond usuellement à la catégorie syntaxique des phrases. Bar-Hillel et al. (1960) ont montré que la classe des langages engendrés par les grammaires AB est la classe des langages engendrés par les grammaires hors-contexte.

Illustrons ces définitions par cet exemple d'un fragment jouet du français, avec le lexique de la table 1.1

$Marie, Paul$	: *np*
$chat, garçon$	: *n*
le, un	: *np*/*n*
$voit, aime$	: (*np**s*)/*np*
$marche$	: *np**s*
qui	: (*np**np*)/(*np**s*)

TAB. 1.1 – Un exemple de lexique pour les grammaires AB

Un verbe intransitif, comme *marche*, nécessite un sujet à sa gauche pour constituer une phrase. Ce sujet peut être aussi bien un nom propre comme *Marie* qu'un groupe nominal composé, comme *un chat*. Si on fixe *np* comme la catégorie syntaxique de ces groupes nominaux, il est donc naturel d'attribuer la catégorie *np**s* aux verbes intransitifs. Le même raisonnement guide l'attribution des autres catégories syntaxiques. Ainsi le pronom relatif *qui* nécessite à sa gauche le groupe nominal qu'il complète, et à sa droite sa proposition relative associée, qui est une phrase dans laquelle le sujet est manquant, et donc de catégorie *np**s*. Sémantiquement, le sujet de cette proposition relative correspond au groupe nominal complété par *qui*. Il est commun, en suivant par exemple Chomsky (1977), d'interprêter ce phénomène comme un mouvement grammatical qui laisse dans la proposition relative une trace, un mot vide qui possède néanmoins une fonction grammaticale, groupe nominal sujet dans cet exemple. Dans le même esprit, nous dirons que les pronoms relatifs extraient les groupes nominaux qu'ils complètent de la proposition relative.

Dans cet exemple de lexique, chaque mot ne reçoit qu'une seule catégorie, et il n'y a donc pas d'ambiguité lexicale. Il aurait été possible d'attribuer également à *marche* la catégorie *n*, car c'est également un nom commun en langue française. Ce nom commun étant féminin, on se serait cependant heurté à des problématiques d'accord, dont le lexique ne se préoccupe pas en l'état (de même qu'il ne se préoccupe pas de conjugaison).

Ce lexique permet néanmoins bien de vérifier la correction syntaxique du fragment (minuscule) de la langue française construit sur ces mots, en calculant si les règles de simplification de fraction permettent d'obtenir *s*, qui est ici notre catégorie distinguée. Ainsi, une phrase comme

(1) le garçon qui aime Marie voit un chat

est reconnue comme grammaticalement correcte, car la séquence des catégories associées à chacun des mots se réduit bien vers s :

$$
\begin{aligned}
& np/n.n.(np\backslash np)/(np\backslash s).(np\backslash s)/np.np.(np\backslash s)/np.\underline{np/n.n} \\
\rightarrow\ & np/n.n.(np\backslash np)/(np\backslash s).(np\backslash s)/np.np.\underline{(np\backslash s)/np.np} \\
\rightarrow\ & np/n.n.(np\backslash np)/(np\backslash s).\underline{(np\backslash s)/np.np}.np\backslash s \\
\rightarrow\ & np/n.n.\underline{(np\backslash np)/(np\backslash s).np\backslash s}.np\backslash s \\
\rightarrow\ & \underline{np/n.n.(np\backslash np).np\backslash s} \\
\rightarrow\ & \underline{np.np\backslash np}.np\backslash s \\
\rightarrow\ & \underline{np.np\backslash s} \\
\rightarrow\ & s
\end{aligned}
$$

À l'inverse, une séquence de mots non grammaticale comme

(2) *le garçon Marie marche

sera rejetée par la grammaire, car il n'existe pour $np/n.n.np.np\backslash s$ aucune possibilité de réduction vers s.

Ces grammaires présentent néanmoins une limitation qui peut être illustrée avec une proposition relative. On a vu dans l'exemple (1) que le pronom relatif *qui* était bien traité. Si on s'intéresse à présent au pronom relatif *que*, qui extrait un groupe nominal objet de la proposition relative, on devrait lui attribuer selon le même raisonnement la catégorie syntaxique $(np\backslash np)/(s/np)$. Regardons à présent si la phrase

(3) le garçon que Marie aime marche

est reconnue par la grammaire. La séquence de catégories correspondante est

$$np/n.n.(np\backslash np)/(s/np).np.(np\backslash s)/np.np\backslash s.$$

On observe que pour que cette séquence se réduise vers s, il faudrait qu'on ait $np.(np\backslash s)/np \rightarrow s/np$, ce qui n'est pas une réduction valide selon les règles qui ont été fixées. On peut également observer que (3) serait reconnu si on avait fait plutôt choisi pour les verbes transitifs la catégorie $np\backslash(s/np)$, qui revient à considérer qu'un verbe transitif est associé d'abord à son sujet, puis à son objet. Le problème ne serait néanmoins que déplacé, puisque (1) ne serait plus reconnue pour une raison similaire.

L'existence de ces limitations liées à la rigidité des règles de calcul a motivé de nombreuses propositions de raffinement des règles. Nous allons voir à présent que Lambek a proposé un formalisme qui subsume l'apport de ces extensions des grammaires AB.

1.1.2 Calcul de lambek

Lambek (1958) reprend l'idée des grammaires AB d'attribuer via un lexique une liste de catégories à chaque mot du vocabulaire considéré. L'ensemble C des catégories se définit donc là aussi comme[1] :

$$C ::= B \mid C/C \mid C\backslash C$$

La différence est que les possibilités de combinaison entre ces catégories sont désormais dictées par des règles de dérivation pour une logique formelle. Puisqu'un verbe doit s'assurer qu'il a un et un seul groupe nominal comme sujet, cette logique prend en compte la consommation des ressources disponibles, et anticipe en celà les principes de la logique linéaire de Girard (1987). Cette logique doit également être sensible à l'ordre des catégories, qui reflète l'ordre des mots dans la phrase, et se distingue donc des logiques traditionnelles par l'absence de règle contextuelle permettant d'inverser deux catégories. Les règles du calcul de Lambek, sous forme de calcul des séquents, sont données dans la table 1.2.

Les catégories à gauche du symbole $\vdash$ dans un séquent sont appelées le contexte, et la catégorie à sa droite est la catégorie prouvée sous ce contexte par ce séquent. Montrer qu'une phrase $m_1 \cdots m_n$ est syntaxiquement correcte

[1] Par soucis de simplicité, on omet le produit non-commutatif . dans cette présentation des grammaires de Lambek

$$\frac{\Gamma, \beta, \Gamma' \vdash \gamma \qquad \Delta \vdash \alpha}{\Gamma, \Delta, \alpha\backslash\beta, \Gamma' \vdash \gamma}\ (\backslash\ \text{l}) \qquad \frac{\alpha, \Gamma \vdash \beta \qquad \Gamma \neq \epsilon}{\Gamma \vdash \alpha\backslash\beta}\ (\backslash\ \text{r})$$

$$\frac{\Gamma, \beta, \Gamma' \vdash \gamma \qquad \Delta \vdash \alpha}{\Gamma, \beta/\alpha, \Delta, \Gamma' \vdash \gamma}\ (/\ \text{l}) \qquad \frac{\Gamma, \alpha \vdash \beta \qquad \Gamma \neq \epsilon}{\Gamma \vdash \beta/\alpha}\ (/\ \text{r})$$

$$\frac{}{\alpha \vdash \alpha}\ \text{(Axiome)} \qquad \frac{\Gamma \vdash \alpha \qquad \Delta, \alpha, \Delta' \vdash \beta}{\Delta, \Gamma, \Delta' \vdash \beta}\ \text{(coupure)}$$

TAB. 1.2 – Règles de déduction pour le calcul de Lambek

consiste alors à prouver que le séquent $\alpha_1, \cdots, \alpha_n \vdash s$, où α_i est une catégorie associée à m_i, est dérivable. A l'instar des grammaires AB, l'ensemble des règles de calcul des grammaires de Lambek est fixé et ne dépend pas du langage qu'on considère. Pour spécifier les caractéristiques du langage que l'on souhaite reconnaître, on dispose d'un concept de lexique identique aux grammaires AB. De même, pour pouvoir définir le langage reconnu par une grammaire de Lambek, il convient de fixer une catégorie distinguée, qui restera *s* dans les exemples suivants.

Lambek (1958) démontre que le calcul de Lambek bénéficie de la propriété d'élimination des coupures, qui signifie que la règle (coupure) est redondante, dans le sens où tout séquent dérivable dans le calcul de Lambek peut être dérivé sans utiliser cette règle Lambek (1958). Cette propriété garanti que le problème de savoir si un séquent est dérivable dans le calcul de Lambek est décidable. En effet, si on exclue cette règle (coupure), chaque règle soit est un axiome, soit contient strictement plus de connecteurs dans sa conclusion que dans ses hypothèse. Plus précisément, toute formule impliquée dans la preuve d'un séquent est une sous-formule d'une formule apparaissant dans ce séquent, et on parle donc de propriété de la sous-formule. La hauteur d'une dérivation d'un séquent sans la règle (coupure) est donc bornée, et il n'y a donc qu'un nombre fini de possibilités à tester.

Montrons à présent que

(3) le garçon que Marie aime marche

correspondant à la séquence de catégories :

$np/n, n, (np\backslash np)/(s/np), np, (np\backslash s)/np, np\backslash s$

est reconnue par le calcul de Lambek (par soucis de clarté, lors de l'utilisation des règles ($\backslash$ l) et ($/$ l), le connecteur introduit par la règle est souligné dans le cas où il y a d'autres occurrences de ce connecteur dans le contexte) :

$$\dfrac{\dfrac{\dfrac{\dfrac{np \vdash np \qquad n \vdash n}{np/n, n \vdash np}\ (/\ \text{l}) \qquad np \vdash np}{np/n, n, np\backslash np \vdash np}\ (\backslash\ \text{l}) \qquad s \vdash s}{np/n, n, np\backslash np, \underline{np\backslash s} \vdash s}\ (\backslash\ \text{l}) \qquad \dfrac{\dfrac{\dfrac{s \vdash s \qquad np \vdash np}{np, np\backslash s \vdash s}\ (\backslash\ \text{l}) \qquad np \vdash np}{np, (np\backslash s)/np, np \vdash s}\ (/\ \text{l})}{np, (np\backslash s)/np \vdash s/np}\ (/\ \text{r})}{np/n, n, \underline{(np\backslash np)/(s/np)}, np, (np\backslash s)/np, np\backslash s \vdash s}\ (/\ \text{l})$$

Informellement, on s'aperçoit que, contrairement au calcul correspondant dans les grammaires AB, on peut dériver le séquent $np, (np\backslash s)/np \vdash s/np$ en introduisant à la droite du contexte une hypothèse sous la forme d'une catégorie *np*. Ce *np* introduit parmi la séquence de catégorie ne correspond à aucun mot de la phrase (3), et une façon d'interpréter sa présence est de le voir comme la trace du groupe nominal qui est extrait par le pronom relatif *que*. Cette possibilité d'introduire des hypothèse offre donc un avantage notable aux grammaires de Lambek pour traiter ces phénomènes d'extractions.

L'extraction dans les grammaires de Lambek pose néanmoins problème quand elle se fait sur une position non périphérique. Pour le constater, enrichissons le lexique à notre disposition de l'adverbe *passionnément*. Un adverbe peut apparaître à la droite d'un groupe verbal, pour former un nouveau groupe verbal, ce qui nous incite à associer

à *passionnément* la catégorie $(np\backslash s)\backslash(np\backslash s)$[2]. Considérons à présent la phrase

(4) le garçon que marie aime passionnément marche.

Pour que (4) soit reconnue par notre grammaire de Lambek, il faudrait pouvoir dériver le séquent

$$np/n, n, (np\backslash np)/(s/np), np, (np\backslash s)/np, (np\backslash s)\backslash(np\backslash s), np\backslash s \vdash s$$

Or on constate que prouver ce séquent revient à prouver $np, (np\backslash s)/np, (np\backslash s)\backslash(np\backslash s) \vdash s/np$. Cette fois-ci néanmoins, l'introduction de *np* comme une hypothèse à droite n'est pas suffisante, puisque la trace devrait se trouver à la droite du verbe *aime*, et non à la droite de *passionnément*. De fait, le séquent obtenu,

$$np, (np\backslash s)/np, (np\backslash s)\backslash(np\backslash s), np \vdash s$$

n'est pas dérivable.

Malgré les facilités de traitement de l'extraction des grammaires de Lambek par rapport aux grammaires AB, la classe des langages générés par les grammaires de Lambek reste la même que celle générée par les grammaires hors-contexte, comme prouvé par Cohen (1967) et Pentus (1993).

1.1.3 Interface syntaxe-sémantique

Un intérêt notable de reconnaître des phrases de la langue naturelle en construisant une démonstration en calcul de Lambek est qu'il est possible d'utiliser cet arbre de preuve pour en extraire une sémantique. En effet, Montague (1974) présente notamment comment utiliser des λ-termes pour représenter la sémantique des langues naturelles. Chaque mot de la phrase se voit attribuer un λ-terme et le sens de la phrase est obtenu en combinant et réduisant tous ces termes, selon le principe de compositionnalité attribué à Frege qui postule que le sens d'une expression est fonction du sens de chacun de ses constituants et de la façon dont ils sont combinés ensemble. Or, il existe un mécanisme approprié pour dériver des λ-termes à partir d'un arbre de démonstration : l'isomorphisme de Curry-Howard.

En effet, une démonstration en calcul de Lambek peut aisément être traduite en une démonstration de la logique intuitionniste, puisque le calcul de Lambek correspond à un fragment de la logique intuitionniste dans lequel l'absence de règles structurelles permet la sensibilité aux ressources et la dissociation de l'implication $\rightarrow$ en deux implications orientées $/$ et $\backslash$. Il s'agit donc d'associer à chaque catégorie syntaxique de base a une formule intuitionniste $\tau(a)$, d'utiliser les règles de traduction

$$\begin{array}{ll} \tau(\alpha/\beta) & := \tau(\beta) \rightarrow \tau(\alpha) \\ \tau(\alpha\backslash\beta) & := \tau(\alpha) \rightarrow \tau(\beta) \end{array}$$

et de traduire chaque règle du calcul de Lambek par la règle correspondante en logique intuitionniste.

Ainsi, si on prend comme exemple la phrase

(5) Marie que Jean aime marche

On dispose dans le calcul de Lambek de l'arbre de preuve Π :

$$\dfrac{\dfrac{\dfrac{np \vdash np \qquad np \vdash np}{np, np\backslash np \vdash np}\,(\backslash\,\mathrm{l}) \qquad s \vdash s}{np, np\backslash np, \underline{np\backslash s} \vdash s}\,(\backslash\,\mathrm{l}) \qquad \dfrac{\dfrac{\dfrac{s \vdash s \qquad np \vdash np}{np, np\backslash s \vdash s}\,(\backslash\,\mathrm{l}) \qquad np \vdash np}{np, (np\backslash s)/np, np \vdash s}\,(/\,\mathrm{l})}{np, (np\backslash s)/np \vdash s/np}\,(/\,\mathrm{r})}{np, \underline{(np\backslash np)/(s/np)}, np, (np\backslash s)/np, np\backslash s \vdash s}\,(/\,\mathrm{l})$$

[2]On pourrait également envisager qu'un adverbe apparait à la droite d'une phrase pour former une nouvelle phrase, ce qui correspondrait à la catégorie $s\backslash s$. Ce choix conduirait néanmoins à la même constatation

Si on suppose à présent au niveau sémantique on dispose des types E dénotant les entités[3] et t dénotant les propositions, que E est associé à la catégorie syntaxique *np* et que t est associé à *s*, alors on observe qu'on peut construire une preuve en logique intuitionniste héritant de la structure de Π :

$$\dfrac{\dfrac{\dfrac{E \vdash E \qquad E \vdash E}{E, E \to E \vdash E}\ (\to \text{l}) \qquad t \vdash t}{E, E \to E, \underline{E \to t} \vdash t}\ (\to \text{l}) \qquad \dfrac{\dfrac{\dfrac{t \vdash t \qquad E \vdash E}{E, E \to t \vdash t}\ (\to \text{l}) \qquad E \vdash E}{E, E \to E \to t, E \vdash t}\ (\to \text{l})}{E, E \to E \to t \vdash E \to t}\ (\to \text{r})}{E, \underline{(E \to t) \to (E \to E)}, E, E \to E \to t, E \to t \vdash t}\ (\to \text{l})$$

Par l'isomorphisme de Curry-Howard, cette preuve correspond à la preuve du séquent

$$x_{Marie} : E, x_{que} : (E \to t) \to (E \to E), x_{Jean} : E, x_{aime} : E \to E \to t, x_{marche} : E \to t \\ \vdash x_{marche}\ (x_{que}\ (\lambda x : E.x_{aime}\ x\ x_{Jean})\ x_{marie}) : t$$

Cette étape nous permet donc de déterminer comment la sémantique de chacun de ses constituants doit être combinée pour obtenir la sémantique de la phrase. Il ne reste donc plus qu'à prendre en compte le λ-terme représentant la sémantique de chacun de ces composants, par exemple :

$$\begin{array}{ll} x_{Marie} & : \lambda P.P\ \mathbf{m} \\ x_{Jean} & : \lambda P.P\ \mathbf{j} \\ x_{que} & : \lambda p, x, n.(p\ x) \wedge (x\ n) \\ x_{aime} & : \lambda o, s.o\ (\lambda y.s(\lambda x.\mathbf{aime}\ y\ x)) \\ x_{marche} & : \lambda s.s\ (\lambda x.\mathbf{marche}\ x) \end{array}$$

Après remplacement des variables par leur λ-terme associé, on obtient alors le terme :

$$(\lambda s.s\ (\lambda x.\mathbf{marche}\ x))\ ((\lambda p, x, n.(p\ x) \wedge (x\ n))\ (\lambda x.(\lambda o, s.o\ (\lambda y.s(\lambda x.\mathbf{aime}\ y\ x)))\ x\ (\lambda P.P\ \mathbf{j}))\ (\lambda P.P\ \mathbf{m}))$$

qui se β-réduit en

$$(\mathbf{marche\ m}) \wedge (\mathbf{aime\ m\ j}).$$

1.2 Grammaires Catégorielles Abstraites

On a vu dans la section précédente que les grammaires catégorielles offraient un calcul syntaxique dans lequel les unités lexicales reçoivent une catégorie exprimant leurs possibilités de combinaison au sein d'une phrase. Les arbres de dérivations de ce calcul pouvaient alors être transformés en arbres de dérivation dans un calcul sémantique. L'idée fondamentale des Grammaires Catégorielles Abstraites, définies par de Groote (2001), est de considérer que ces deux calculs sont de même nature, et peuvent être traités par le même ensemble restreint de primitives mathématiques de la théorie des types et du λ-calcul.

En effet, chacun des trois niveaux impliqués – le langage naturel étudié exprimé sous forme de chaînes de caractère, les structures de dérivations de ces chaînes de caractère et les formules logiques exprimant leur sens – peut être définit comme un ensemble de λ-termes. On parlera dans la suite respectivement de niveau syntaxique, abstrait et sémantique, et bien qu'ils jouent des rôles distincts d'un point de vue linguistique, ils ne seront pas distingués du point de vue des définitions formelles. C'est en ce sens que ces grammaires sont abstraites, chacun de ces niveaux concrètement distincts étant traité de manière uniforme. Les relations linguistiques qui existent entre ces différents niveaux seront alors exprimées comme des morphismes entre ensembles de λ-termes. En particulier, cette architecture offre une séparation claire de la structure profonde de la phrase, qui est traitée au niveau abstrait et varie peu d'une langue naturelle à l'autre, et l'agencement des mots qui en résulte, qui est propre à chaque langue naturelle et qui se traduit comme un morphisme du niveau abstrait vers le niveau syntaxique.

Cette abstraction des différents niveaux offre plusieurs avantages. D'une part, cette factorisation vers un formalisme unique muni d'un nombre restreint de règles simplifie la tâche d'implémentation. D'autre part, traiter les

[3] Ce type n'est pas atomique et consiste en réalité en une montée du type atomique des entités $e : E := (e \to t) \to t$. Nous n'entrerons pas dans le détail de ces considérations.

structures de dérivations comme des termes du formalisme perment d'avoir un contrôle direct sur celles-ci. Enfin, cette structure générale en niveaux de même nature reliés par des morphismes assure la flexibilité et la modularité du formalisme, chaque niveau disposant de toutes les propriétés nécessaires pour être l'ensemble de départ ou d'arrivée d'un tel morphisme.

Notons également que, pour la même raison que le calcul de Lambek correspondait par l'absence de règle de contraction à une logique sensible aux ressources, le λ-calcul sur lequel les ACG vont être définies est un λ-calcul linéaire, ce qui signifie que les variables abstraites apparaissent une unique fois dans le corps de l'abstraction, et ne sont donc jamais ni supprimées ni dédoublées.

1.2.1 Définitions

Les définitions suivent de Groote (2001). Le système de type sur lequel repose le formalisme est la logique linéaire implicative intuitionniste, dotée comme unique constructeur de l'implication linéaire $\multimap$. Nous discuterons dans la section 1.2.5 des possibilités d'extension de ce système à d'autres constructeurs.

Définition 1 (Types)**.** *Soit A un ensemble fini de types atomiques. On définit inductivement l'ensemble $\mathscr{T}(A)$ des types linéaires implicatifs construits sur A comme :*

$$\mathscr{T}(A) := A \mid \mathscr{T}(A) \multimap \mathscr{T}(A).$$

On introduit à présent le concept de signature linéaire d'ordre supérieur, qui consiste informellement en une liste de constantes auxquelles on attribue un type.

Définition 2 (Signature)**.** *Une signature linéaire d'ordre supérieur*[4] *Σ est un triplet $\langle A, C, \tau\rangle$, avec*
- *A un ensemble fini de types atomiques ;*
- *C un ensemble fini de constantes ;*
- *τ une fonction de C dans $\mathscr{T}(A)$.*

Les constantes introduites par une telle signature peuvent alors être utilisées pour construire des λ-termes sur cette signature.

Définition 3 (Termes)**.** *On suppose qu'on dispose d'un ensemble infini de variables X. Soit $\Sigma = \langle A, C, \tau\rangle$ une signature linéaire d'ordre supérieur. L'ensemble $T(\Sigma)$ des λ-termes linéaires construits sur Σ est défini inductivement comme :*

$T(\Sigma) := X \mid C \mid \lambda^0 X.T(\Sigma) \mid (T(\Sigma)\ T(\Sigma))$.

On suppose pour la suite qu'on dispose sur ces λ-termes de la notion de subsitution sans capture, et des relations d'α-conversion, de β-réduction, d'η-réduction et de $\beta\eta$-conversion usuelles, comme présentées par Barendregt (1984).

On introduit également des règles de typages pour ces λ-termes

Définition 4 (Contexte)**.** *Soit X un ensemble infini de variables et $\Sigma = \langle A, C, \tau\rangle$.*

On définit un contexte Γ comme un ensemble fini d'assignations de types à des variables, de la forme $x : \alpha$, avec $x \in X$ et $\alpha \in \mathscr{T}(A)$, et tel qu'une même variable n'apparaît pas plus d'une fois.

Définition 5 (Typage)**.** *Soit une signature $\Sigma = \langle A, C, \tau\rangle$, on dit que t est typable de type $\alpha \in \mathscr{T}(A)$ dans Σ sous le contexte Γ si le séquent $\Gamma \vdash_\Sigma t : \alpha$ est dérivable à l'aide des règles de la table 1.3.*

Il est immédiat de vérifier que les λ-termes typables dans une signature Σ sont effectivement linéaires, c'est-à-dire que chaque abstraction correspond à une unique occurrence de la variable liée.

Nous avons vu que nos ensembles de λ-termes s'articulaient ensembles à l'aide de morphismes. Ces morphismes viennent par paires, puisqu'ils doivent traduire les termes mais aussi leur type. Nous appellerons ces paires de morphismes des lexiques, à l'image des lexiques des grammaires de lambek qui décrivent pour chaque mot comment il peut être agencé aux autres dans la phrase.

Définition 6 (Lexique)**.** *Soient deux signatures linéaires d'ordre supérieur $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ et $\Sigma_2 = \langle A_2, C_2, \tau_2\rangle$. Un* lexique *$\mathscr{L}$ de Σ_1 vers Σ_2 est une paire $\langle \eta, \theta\rangle$ où :*

[4] On parlera dans la suite plus simplement de signature

$$\frac{}{\vdash_\Sigma c : \tau(c)}\ \text{(const.)} \qquad \frac{}{x : \alpha \vdash_\Sigma x : \alpha}\ \text{(var.)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \lambda^0 x.t : \alpha \multimap \beta}\ \text{(abs.)} \qquad \frac{\Gamma \vdash_\Sigma t : \alpha \multimap \beta \qquad \Gamma' \vdash_\Sigma u : \alpha}{\Gamma, \Gamma' \vdash_\Sigma t\ u : \beta}\ \text{(app.)}$$

TAB. 1.3 – Règles de typage des ACG

- *η est un morphisme de A_1 vers $\mathscr{T}(A_2)$ (on note également η son unique extension à $\mathscr{T}(A_1)$) ;*
- *θ est un morphisme de C_1 vers $T(\Sigma_2)$ (on note également θ son unique extension à $T()(\Sigma_1)$) ;*
- *pour tout $c \in C_1$, le séquent*
  $\vdash_{\Sigma_2} \theta(c) : \eta(\tau_{\Sigma_1}(c))$
  est dérivable.

On pourra utiliser la notation $\mathscr{L}$ pour η ou θ dans la suite.
On dispose à présent de toutes les définitions préalables pour introduire la notion de grammaire.

Définition 7 (Grammaire catégorielle abstraite)**.** *Une* grammaire catégorielle abstraite *est un quadruplet $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ où :*

1. *$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ et $\Sigma_2 = \langle A_1, C_1, \tau_1 \rangle$ sont deux signatures linéaires d'ordre supérieur : le* vocabulaire abstrait *et le* vocabulaire objet*, respectivement ;*
2. *$\mathscr{L} : \Sigma_1 \to \Sigma_2$ est un lexique du vocabulaire abstrait vers le vocabulaire objet ;*
3. *$s \in \mathscr{T}(A_1)$ (dans le vocabulaire abstrait) est le* type distingué *de la grammaire.*

Le vocabulaire objet spécifie les structures de surface de la grammaire (*e.g.* des chaînes de caractères ou des formules logiques sous forme d'arbres), alors que le vocabulaire abstrait spécifie les structures de dérivation (*e.g.* des arbres, ou plus généralement des arbres de preuve de grammaire catégorielle). Le lexique spécifie comment les structures de surface sont associées aux structures de dérivation.

Une grammaire catégorielle abstraite génère deux langages, le langage abstrait et le langage objet. Le langage abstrait est constitué des termes typables par le type distingué, qui est bien un type du vocabulaire abstrait. le langage objet est dérivé du langage objet par le lexique.

Définition 8 (Langages)**.** *Une ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ défini deux langages :*

- *le* langage abstrait *: $\mathcal{A}(\mathscr{G}) = \{t \in T(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ est dérivable}\}$*
- *le* langage objet*, qui est l'image du langage abstrait par le lexique :*
  $\mathcal{O}(\mathscr{G}) = \{t \in T(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathscr{G}).\, t =_{\beta\eta} \mathscr{L}(u)\}$

1.2.2 Un exemple de modélisation de la syntaxe en ACG

Afin d'illustrer le fonctionnement des ACG, reprenons le fragment du français introduit dans la section1.1.2. et définissons une ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}_{\text{Syn}}, s \rangle$ qui reconnait ce fragment.

$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ est la signature sur laquelle seront construites les structures de dérivation. Ces structures expriment les relations syntaxiques entre les constituants de la phrase, les types atomiques qui doivent être utilisés correspondent donc aux catégories syntaxiques usuelles : $A_1 = \{n, np, s\}$.

Les constantes introduites dans C_1 ne sont pas les mots proprement dits, mais les constantes abstraites qui les représentent dans ce niveau des structures de dérivation. Leur type fait écho aux catégories syntaxiques du calcul de Lambek, où / et \ sont convertis en $\multimap$. Ainsi, un verbe intransitif sera représenté par une constante de fonction qui prend un *np*, son sujet, en argument. L'information syntaxique induite par / et \ sur la direction dans laquelle ces arguments sont attendus est donc totalement écartée ici. Σ_1 est détaillée dans la table 1.4.

La signature $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ correspond aux formes de surface. Il est raisonnable de considérer les formes de surface syntaxiques comme des chaînes de caractères. On peut représenter une chaîne de caractère /abcd/ par le λ-terme linéaire $\lambda^0 x.a(b(c(d\ x)))$. L'opération de concaténation de deux chaînes peut alors être définie comme la composition de fonctions $+ := \lambda^0 f\ g\ x.f(g\ x)$, qui est bien associative (on utilisera dans la suite une notation

$$
\begin{array}{lll}
\Sigma_1 : & C_{\text{Marie}}, C_{\text{Paul}} & : np \\
& C_{\text{le}}, C_{\text{un}} & : n \multimap np \\
& C_{\text{marche}} & : np \multimap s \\
& C_{\text{qui}}, C_{\text{que}} & : (np \multimap s) \multimap np \multimap s \\
& C_{\text{chat}}, C_{\text{garçon}} & : n \\
& C_{\text{voit}}, C_{\text{aime}} & : np \multimap np \multimap s \\
& C_{\text{passionnément}} & : (np \multimap s) \multimap (np \multimap s)
\end{array}
$$

TAB. 1.4 – Exemple de signature abstraite

$$
\begin{array}{ll}
\Sigma_2 : & /\text{Marie}/, /\text{Paul}/, /\text{chat}/, /\text{garçon}/, /\text{le}/, /\text{un}/, /\text{voit}/, /\text{aime}/, \\
& /\text{marche}/, /\text{passionnément}/, /\text{qui}/, /\text{que}/ : \sigma \multimap \sigma
\end{array}
$$

TAB. 1.5 – Exemple de signature objet

infixe par soucis de clarté). Il faut donc introduire dans Σ_2 un type σ de sorte que le type des chaînes de caractères sera $\sigma \multimap \sigma$. Σ_2 est détaillée dans la table 1.5

On peut alors définir le lexique $\mathscr{L}_{\text{Syn}}$ entre les deux signatures. Ce lexique met en relation les constantes abstraites de Σ_1 avec les chaînes de caractères qu'elles représentent. Ce lexique spécifie également comment les combinaisons syntaxiques sont répercutées en agencements dans la phrase. Ce lexique utilise le connecteur $+$ défini précédemment, ainsi que le λ-terme identité $\lambda^0 y.y$ qui correspond à la chaîne de caractères vide dans cet encodage. Il est détaillé dans la table 1.6

$\mathscr{L}_{\text{Syn}}$:

$$
\begin{array}{lll}
s, np, n & :=_{\text{Syn}} \sigma \multimap \sigma & \\
C_{\text{Marie}} & :=_{\text{Syn}} /\text{Marie}/ & C_{\text{Paul}} :=_{\text{Syn}} /\text{Paul}/ \\
C_{\text{chat}} & :=_{\text{Syn}} /\text{chat}/ & C_{\text{garçon}} :=_{\text{Syn}} /\text{garçon}/ \\
C_{\text{le}} & :=_{\text{Syn}} \lambda^0 n.(/\text{le}/ + n) & C_{\text{un}} :=_{\text{Syn}} \lambda^0 n.(/\text{un}/ + n) \\
C_{\text{voit}} & :=_{\text{Syn}} \lambda^0 o, s.(s + /\text{voit}/ + o) & C_{\text{aime}} :=_{\text{Syn}} \lambda^0 o, s.(s + /\text{aime}/ + o) \\
C_{\text{marche}} & :=_{\text{Syn}} \lambda^0 s.(s + /\text{voit}/) & C_{\text{passionnément}} :=_{\text{Syn}} \lambda^0 v.(v + /\text{passionnément}/) \\
C_{\text{qui}} & :=_{\text{Syn}} \lambda^0 p, n.(n + /\text{qui}/ + (p\ (\lambda^0 x.x)) & C_{\text{que}} :=_{\text{Syn}} \lambda^0 p, n.(n + /\text{que}/ + (p\ (\lambda^0 y.y))
\end{array}
$$

TAB. 1.6 – Un exemple de lexique traitant la syntaxe

On peut à présent constater que le terme
$C_{\text{marche}}\ (C_{\text{que}}\ (\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))\ (C_{\text{le}}\ C_{\text{garçon}}))$
appartient au langage abstrait de la grammaire, comme le prouve l'arbre de dérivation de la table 1.7

Par ailleurs, en appliquant le lexique $\mathscr{L}_{\text{Syn}}$ à ce terme et en β-normalisant le résultat, on obtient bien le λ-terme correspondant à la chaîne de caractères $/\text{le}/ + /\text{garçon}/ + /\text{que}/ + /\text{Marie}/ + /\text{aime}/ + /\text{passionnément}/ + /\text{marche}/$. Cette phrase fait donc partie du langage reconnu par notre grammaire.

A l'inverse, une chaîne de caractères grammaticalement incorrecte comme $/\text{le}/+/\text{garçon}/+/\text{que}/+/\text{Marie}/+/\text{marche}/$ n'a aucun antécédent typable par $\mathscr{L}_{\text{Syn}}$, a fortiori de type s. Cette chaîne n'est donc pas reconnue par notre grammaire.

On constate que, le niveau abstrait étant libéré des problématiques d'agencement des mots (phénosyntaxe), le mécanisme pour introduire la trace d'une extraction est plus flexible que dans le calcul de Lambek. En effet, dans le terme abstrait

$$C_{\text{marche}}\ (C_{\text{que}}\ (\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))\ (C_{\text{le}}\ C_{\text{garçon}}))$$

la variable x, introduite par une λ-abstraction, correspond à cette trace, et rien n'interdit d'utiliser cette variable x dans n'importe quelle position du terme gouvernée par cette λ-abstraction. La présence du mot *passionnément* n'entrave donc en rien la possibilité de construire un tel terme.

Il s'avère au contraire que cette possibilité d'utiliser la trace de l'extraction dans n'importe quelle position engendre des problèmes de surgénération : il est possible de construire des termes appartenant au langage abstrait dont l'image par $\mathscr{L}_{\text{Syn}}$ ne correspond pas à une phrase syntaxiquement correcte. Par exemple, comme la phénosyntaxe

$$
\dfrac{\dfrac{\dfrac{\Pi :}{\vdash_{\Sigma_1} C_{\text{passionnément}} : (np \multimap s) \multimap (np \multimap s)}\,(\text{const.}) \quad \dfrac{\dfrac{}{\vdash_{\Sigma_1} C_{\text{aime}} : np \multimap np \multimap s}\,(\text{const.}) \quad \dfrac{}{x : np \vdash_{\Sigma_1} x : np}\,(\text{var})}{x : np \vdash_{\Sigma_1} C_{\text{aime}}\ x : np \multimap s}\,(\text{app.})}{x : np \vdash_{\Sigma_1} C_{\text{passionnément}}\ (C_{\text{aime}}\ x) : np \multimap s}\,(\text{app.}) \quad \dfrac{}{\vdash_{\Sigma_1} C_{\text{Marie}} : np}\,(\text{const.})}{x : np \vdash_{\Sigma_1} C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}} : s}\,(\text{app.})}{\vdash_{\Sigma_1} \lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}) : np \multimap s}\,(\text{abs.})
$$

$$
\dfrac{\dfrac{}{\vdash_{\Sigma_1} C_{\text{marche}} : np \multimap s}\,(\text{const.}) \quad \dfrac{\dfrac{\dfrac{}{\vdash_{\Sigma_1} C_{\text{que}} : (np \multimap s) \multimap np \multimap np}\,(\text{const.}) \quad \Pi}{\vdash_{\Sigma_1} C_{\text{que}}\ (\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}})) : np \multimap np}\,(\text{app.}) \quad \dfrac{\dfrac{}{\vdash_{\Sigma_1} C_{\text{le}} : n \multimap np}\,(\text{const.}) \quad \dfrac{}{\vdash_{\Sigma_1} C_{\text{garçon}} : n}\,(\text{const.})}{\vdash_{\Sigma_1} C_{\text{le}}\ C_{\text{garçon}} : np}\,(\text{app.})}{\vdash_{\Sigma_1} C_{\text{que}}\ (\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))\ (C_{\text{le}}\ C_{\text{garçon}}) : np}\,(\text{app.})}{\vdash_{\Sigma_1} C_{\text{marche}}\ (C_{\text{que}}\ (\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))\ (C_{\text{le}}\ C_{\text{garçon}})) : s}\,(\text{app.})
$$

TAB. 1.7 – Un exemple de typage d'un terme abstrait

n'apparait pas au niveau abstrait, rien n'interdit au pronom relatif *que* d'extraire un groupe nominal qui n'est pas un objet. Afin de pallier ces phénomènes de surgénération, il est donc nécessaire de détecter et d'écarter ces termes incorrects, soit au niveau des structures de dérivation, soit au niveau des chaînes de caractères. Nous reviendrons sur cette problématique dans le chapitre 4.

1.2.3 Propriétés

Tout d'abord, une propriété cruciale du formalisme est la cohérence du lexique et de la relation de typage. Lorsque le lexique correspond à la réalisation sémantiques des structures de dérivation du langage natural, cette propriété correspond alors à la notion de compositionnalité de la sémantique, comme noté par Lebedeva (2012).

Théorème 1 (Compositionnalité)**.**
Si $\mathscr{L} = \langle \eta, \theta \rangle$ est un lexique de Σ_1 dans Σ_2, et si on a une dérivation Π du séquent
$x_0 : \alpha_0, \cdots, x_n : \alpha_n \vdash_{\Sigma_1} t : \alpha$ *;*
alors le séquent
$x_0 : \eta(\alpha_0), \cdots, x_n : \eta(\alpha_n) \vdash_{\Sigma_1} \theta(t) : \eta(\alpha)$
est dérivable.

Démonstration. Immédiat par récurrence sur la hauteur de Π.

□

Le problème de l'analyse syntaxique pour une ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ et un terme $t \in T(\Sigma_2)$ consiste à construire un terme $u \in T(\Sigma_1)$ tel que $\mathscr{L}(u) =_{\beta\eta} t$, c'est-à-dire à calculer un antécédent du terme t par le lexique $\mathscr{L}$. L'analyse syntaxique est un problème crucial, car il permet d'associer à une forme de surface une de ses structures de dérivation, et il a donc été étudié en conséquence. Salvati (2005)) montre que l'analyse syntaxique des ACG se réduit au problème de la prouvabilité dans le fragment multiplicatif exponentiel de la logique linéaire (MELL), qui est toujours ouvert.

Dans les ACG lexicalisées, c'est -à-dire lorsque chaque constante du vocabulaire abstrait est associée par le lexique à un terme contenant une constante du vocabulaire objet, le problème de l'appartenance d'un terme t au langage reconnu est NP-complet Yoshinaka et Kanazawa (2005)

Le pouvoir expressif des ACG englobe notamment celui des grammaires d'abres adjoints (TAG), des grammaires hors-contexte (CFG) et des systèmes de réécriture hors-contexte m-linéaires, puisque chacun de ces formalisme peut être encodé à l'aide d'une ACG de Groote (2002, 2001); de Groote et Pogodalla (2003).

Il est possible d'affiner ces résultats en distinguant plusieurs classes de grammaires, le critère principal étant l'ordre d'une ACG.

Définition 9 (ordre)**.**

1. *L'ordre $ord(\alpha)$ d'un type α est défini inductivement comme suit :*
   - $ord(\alpha) := 1$ *si α est atomique ;*
   - $ord(\alpha) := max(1 + ord(\alpha_1), ord(\alpha_2))$ *si $\alpha = \alpha_1 \multimap \alpha_2$.*
2. *L'ordre $ord(\Sigma)$ d'une signature $\Sigma = \langle A, C, \tau \rangle$ est défini comme :*
   $ord(\Sigma) = max\{ord(\tau(c)) | c \in C\}$.
3. *L'ordre d'une ACG $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ est défini comme l'ordre de son vocabulaire abstrait Σ_1.*

de Groote et Pogodalla (2004); Salvati (2005); Kanazawa (2007) ont notamment prouvé que l'analyse syntaxique des ACG d'ordre 2 est de complexité polynomiale, et les langages générés correspondent aux "mildly context-sensitive languages".

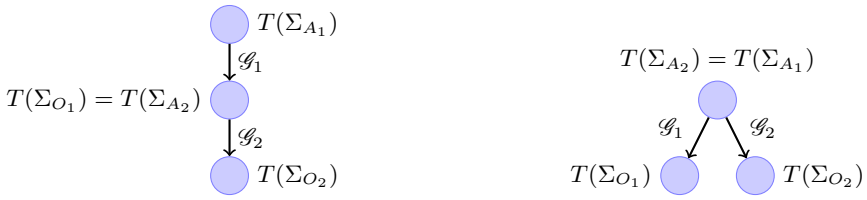
1.2.4 Architecture grammaticale

Comme les deux sont des signatures d'ordre supérieur, le vocabulaire abstrait et le vocabulaire objet ne présentent aucune différence structurelle. Cette propriété permet de composer différentes ACG de façon toute naturelle. La figure 1.1(a) illustre une première manière de composer deux ACG : le vocabulaire objet de la première ACG $\mathscr{G}_1$ est le vocabulaire abstrait de la seconde $\mathscr{G}_2$. L'intérêt est double :

– soit un terme $u \in \mathcal{A}(\mathscr{G}_2)$ a au moins un antécédent par le lexique de $\mathscr{G}_1$ dans $\mathcal{A}(\mathscr{G}_1)$. $\mathscr{G}_2 \circ \mathscr{G}_1$ fournit alors plus d'analyses pour un même terme objet de $\mathcal{O}(\mathscr{G}_2)$ que ne le fait $\mathscr{G}_2$. Pogodalla (2007); de Groote et al. (2009) utilise cette architecture pour modéliser les ambiguités de portée en utilisant pour les groupes nominaux quantifiés des types d'ordre supérieur au niveau de Σ_{A_1} tandis que leurs types restent d'ordre bas au niveau de Σ_{A_2} ;
– Soit un terme $u \in \mathcal{A}(\mathscr{G}_2)$ n'a aucun antécédent par le lexique de $\mathscr{G}_1$ dans $\mathcal{A}(\mathscr{G}_1)$. Cela siginifie que $\mathscr{G}_2 \circ \mathscr{G}_1$ *rejette* d'une certaine manière des analyses fournies par $\mathscr{G}_2$ d'un terme objet de $T(O_2)$. Cette architecture permet donc d'exercer un contrôle sur $\mathscr{G}_2$ pour décider quelles sont les structures de dérivation acceptables. C'est ce mécanisme de contrôle que nous emploierons dans le chapitre 4 pour identifier et écarter les termes abstraits qui correspondent à des mouvements grammaticaux incorrects.

La figure 1.1(b) illustre la seconde façon de composer deux ACG : $\mathscr{G}_1$ et $\mathscr{G}_2$ partagent le même vocabulaire abstrait, et définissent donc le même langage abstrait. Cette architecture est utilisée en particulier lorsque l'une des ACG spécifie les structures syntaxiques, à l'image de l'exemple de la section 1.2.2, et la seconde les structures sémantiques de la langue. Le vocabulaire abstrait partagé spécifie alors l'interface syntaxe-sémantique. Cette perspective est notamment explorée par Pogodalla (2004).

Dans cette architecture, la syntaxe et la sémantique sont donc dérivées chacune d'un niveau intermédiaire qui spécifie les structures profondes de la langue. Elle s'inscrit donc dans la famille des architectures parallèles de Groote et al. (2009), par contraste avec les architectures syntactocentrées dans lesquelles le calcul sémantique est directement dérivé du calcul syntaxique, à l'exemple de l'interface syntaxe-sémantique des grammaires catégorielles de la section 1.1.3. Cette architecture parallèle pour l'interface syntaxe-sémantique correspond notameent à la présentation des TAG synchrones comme architecture bi-morphique, introduite par Shieber (2006).



(a) Premier mode de composition

(b) Second mode de composition

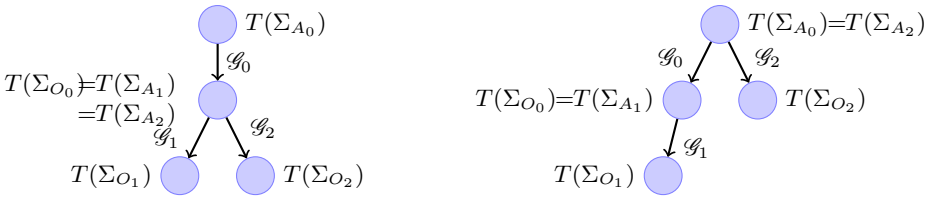
FIG. 1.1 – Différents modes de composition des ACG

Enfin, il est également possible de mélanger les deux modes de composition, comme l'illustre la figure 1.2. La figure 1.2(b) correspond par exemple à une architecture où l'ACG traitant la sémantique est rattachée au niveau abstrait le plus haut. Cette architecture a été utilisée dans Pogodalla (2007) et de Groote et al. (2009) pour modéliser les ambiguités sémantiques tout en gardant à un niveau intermédiaire un type syntaxique non ambigu pour les quantificateurs. En effet la sémantique doit dans cet exemple être rattachée à un endroit où l'ambiguité est déjà apparue.

D'un autre côté, si l'interface syntaxe-sémantique se situe à un niveau intermédiaire comme dans Fig. 1.2(a), l'ACG la plus abstraite peut exercer un contrôle supplémentaire sur les structures acceptables : certaines structures syntaxiques peuvent par exemple se voir attribuer une sémantique bien qu'elles soient interdites dans certaines langues, d'où le besoin de ce niveau de contrôle supplémentaire pour les rejeter. Cette approche permet ainsi de définir dans un premier temps une interface syntaxe-sémantique classique, et dans un second temps d'ajouter ce niveau de contrôle supplémentaire sans rien modifier à l'interface elle-même.

Dans les différents cas, puisque la composition de deux ACG est elle-même une ACG, ces architecture se ramènent à l'une de Fig. 1.1(b). Cependant, conserver une architecture à plusieurs niveaux permet une approche modulaire de la conception de grammaires effectives, soit en réutilisant des composantes comme dans Fig. 1.2(a) (où le niveau syntaxe-sémantique n'est pas affecté par le niveau de contrôle exercé par l'ACG supérieure), soit en fournissant des composantes intermédiaires comme dans Fig. 1.2(b) (comme le type d'ordre bas pour les quantifi-

cateurs, à l'inverse des grammaires catégorielles)[5].



(a) Première combinaison

(b) Seconde combinaison

FIG. 1.2 – Combinaisons des modes de composition

1.2.5 Extension du système de typage

Les définitions des ACG que nous avons données ne sont pas assujetties au système de typage utilisé, et il est donc aisé d'introduire dans le formalisme de nouveaux constructeurs de types et les constructeurs de termes qui leur sont associés. Si le pouvoir expressif des ACG munies uniquement de l'implication linéaire semble suffisant pour traiter les langues naturelles, l'ajout de nouveaux constructeurs peut en effet permettre de rendre l'écriture effective de grammaires plus intuitive et plus pratique.

Les connecteurs de la logique linaire, interprétés comme des constructeurs de type par l'isomorphisme de Curry-Howard, sont des candidats naturels pour cette extension du système de typage, comme noté dans de Groote (2001). Ils garantissent notamment que le λ-calcul ainsi enrichi conserve les propriétés de normalisation et de confluence nécessaires pour calculer la forme normale d'un terme retourné par le lexique.

La conjonction et la disjonction additives $\&$ et $\oplus$ correspondent respectivement au produit cartésien et à l'union disjointe. Disposer du produit cartésien permet par exemple d'encapsuler toutes les informations d'un objet grammatical telles que le genre ou le nombre dans une structure de traits. L'union disjointe permet d'utiliser un constructeur d'analyse de cas, permettant par exemple au lexique de faire varier l'image d'un terme selon la valeur de cette structure de traits.

Une autre extension intéressante consiste à permettre d'utiliser des λ-termes non linéaires. Cette contrainte de linéarité des λ-terme est en effet héritée de la sensibilité aux ressources des calculs logiques des grammaires catégorielles, et est pertinente pour traiter la syntaxe de la langue. Cependant, le traitement de la sémantique appelle naturellement la possibilité d'abstractions non linéaires, ce qui apparait précisément dans l'exemple d'interface syntaxe-sémantique que nous avons présenté dans la section 1.1.3. Le connecteur logique correspondant à l'abstraction non linéaire est l'implication intuitionniste. Ce n'est pas un connecteur de la logique linéaire, mais peut être défini à l'aide de la modalité "‘ !"', comme $A! \multimap B$.

Enfin, le produit dépendant est un constructeur de type permettant à un type de dépendre d'un terme. Cette construction permet donc au type associé à une catégorie syntaxique, comme *np*, de dépendre de la structure de traits que nous avons évoquée précédemment. Attribuer ce type à un terme permet alors non seulement de renseigner la catégorie syntaxique correspondant à ce terme, mais également la valeur de chaque champ de la structure de traits pour ce terme. Une autre application est de faire dépendre les types par un terme de contrôle, calculé de sorte que certains termes syntaxiquement incorrects mais pourtant acceptés par le typage simple soient défaussés. Cette seconde application sera notamment mise en oeuvre au chapitre 4. Enfin, d'un point de vue technique l'implication non linéaire discutée précédemment est un cas particulier du produit dépendant.

Avant de donner formellement le détail du λ-calcul obtenu en procédant à toutes ces extensions, illustrons leur intérêt par l'exemple suivant, tiré de[de Groote et Maarek (2007)], et traitant les problématiques d'accord selon le genre et le nombre à l'aide d'une structure de traits :

[5]Cependant, par simplicité, ce niveau supplémentaire n'apparaîtra pas dans la suite, et les groupes nominaux quantifiés auront directement un type d'ordre supérieur classique.

$\mathscr{G} = \langle \Sigma_A, \Sigma_O, \mathscr{L}, s \rangle$
avec Σ_A, Σ_O et $\mathscr{L}$ donnés par les tables 1.8, 1.9 et 1.10

Σ_A :	
$genre = \{m\|f\}$	: **type**
$nombre = \{s\|p\}$	: **type**
$trait = [g : genre, n : nombre]$	: **type**
n, np	: $(trait)$ **type**
s	: **type**
$C_{\text{Marie}}, C_{\text{Alice}}$	: $np\ [g = f, n = s]$
$C_{\text{Paul}}, C_{\text{Jean}}$	: $np\ [g = m, n = s]$
$C_{\text{être}}$	: $(\Pi x : trait)\,((\Pi y : trait)\,n) \multimap np\ x \multimap s$
C_{et}	: $(\Pi x, y : trait)\,np\ x \multimap np\ y \multimap np\ [g = Accord\ x\ y, n = p]$
$C_{\text{mathématicien}}$	: $(\Pi x : trait)\,n\ x$

TAB. 1.8 – Vocabulaire abstrait avec extensions

Dans la table 1.8, $Accord$ désigne le λ-terme $\lambda^0 x, y.\{case\ x.g\ of\ m \rightarrow m | f \rightarrow y.g\}$, qui spécifie que le genre d'une conjonction est féminin si et seulement si les deux termes conjoints sont féminins.

Σ_O :	
$genre = \{m\|f\}$	: **type**
$nombre = \{s\|p\}$	: **type**
$trait = [g : genre, n : nombre]$	: **type**
σ	: **type**
/Marie/, /Alice/, /Paul/, /Jean/, /être/, /et/, /mathématicien/	: $\sigma \multimap \sigma$

TAB. 1.9 – Vocabulaire objet avec extensions

$\mathscr{L}_{\text{Syn}}$:

$genre$	$:=_{\text{Syn}} genre$
$nombre$	$:=_{\text{Syn}} nombre$
$trait$	$:=_{\text{Syn}} trait$
np, n	$:=_{\text{Syn}} \lambda x.\sigma \multimap \sigma$
s	$:=_{\text{Syn}} \sigma \multimap \sigma$
C_{Marie}	$:=_{\text{Syn}}$ /Marie/
C_{Paul}	$:=_{\text{Syn}}$ /Paul/
$C_{\text{être}}$	$:=_{\text{Syn}} \lambda m.\lambda^0 x, y.y + \{case\ m.n\ of\ s \rightarrow$ /est/$\|p \rightarrow$ /sont/$\} + x\ m$
C_{et}	$:=_{\text{Syn}} \lambda m, n.\lambda^0 x, y.x +$ /et/ $+ y$
$C_{\text{mathématicien}}$	$:=_{\text{Syn}} M$

TAB. 1.10 – Lexique avec extensions

Dans la table 1.10, M désigne le terme $\lambda m.\{case\ m.g\ of\ m \rightarrow \{case\ m.n\ of\ s \rightarrow$ /mathématicien/$|p \rightarrow$ /mathématiciens/$\}|f \rightarrow \{case\ m.n\ of s \rightarrow$ /mathématicienne/$|p \rightarrow$ /mathématiciennes/$\}\}$.

La syntaxe exacte du calcul et les règles qui lui sont associées sont données au chapitre 2. Intuitivement, [ ] correspond aux enregistrements, et { } aux analyses de cas. Ainsi par exemple $[g = m, n = s].g$ se réduit vers m, et $\{case\ m\ of\ m \rightarrow$ /mathématicien/$|f \rightarrow$ /mathématicienne/$\}$ vers /mathématicien/. Les types atomiques se voient attribuer des sortes dans les signatures, pour indiquer leurs possibilités d'applications à des termes. Par exemple, dans Σ_A, s reçoit la sorte **type**, et a donc le même comportement que dans les exemples précédents. En

revanches, les types atomiques *n* et *np* reçoivent la sorte $(trait)$ **type**, ce qui signifie que ces types doivent être appliqués à un terme de type $trait$ pour être attribués à un terme. Ainsi, C_{Paul} est typé par *np* $[g = m, n = s]$. Le constructeur Π correspond au produit dépendant. $(\Pi x : \alpha)\, \beta$ correspond au type des termes qui peuvent s'appliquer à un terme u de type α pour retourner un terme de type $\beta[x := u]$. Le terme $C_{\text{être}}$ $[g = m, n = s]$ est donc de type $((\Pi y : trait)\, n) \multimap np\ [g = m, n = s] \multimap s$ par exemple.

Le terme

$$C_{\text{être}}\ [g = m, n = s]\ C_{\text{mathématicien}}\ C_{\text{Paul}}$$

fait donc partie du langage abstrait de $\mathscr{G}$. Ce terme devient par le lexique et β-réduction la chaine de caractères /Paul/ + /est/ + /mathématicien/. A l'inverse, le terme

$$C_{\text{être}}\ [g = f, n = s]\ C_{\text{mathématicien}}\ C_{\text{Paul}}$$

n'est pas typable, la structure de traits passée en argument à $C_{\text{être}}$ produisant un type incompatible avec celui de C_{Paul}.

De la même façon, le terme

$$C_{\text{être}}\ [g = m, n = p]\ C_{\text{mathématicien}}\ (C_{\text{et}}\ [g = m, n = s]\ [g = f, n = s]\ C_{\text{Paul}}\ C_{\text{Alice}})$$

appartient au langage abstrait, et correspond à la chaîne de caractères /Paul/ + /et/ + /Alice/ + /sont/ + /mathématiciens/.

de Groote et al. (2007) ont prouvé que l'extension du système de type avec la somme disjointe, le produit cartésien ou le produit dépendant produit un formalisme Turing-complet. Le problème d'identifier des fragments intéressants et conservant un problème de l'analyse résoluble est donc essentiel. Cependant, en ce qui concerne le produit dépendant, une signature dont les types ne dépendent que de types habités par un nombre fini de termes (comme dans l'exemple précédent, *np* dépend du type $trait$ qui n'a que quatre représentants) peut être exprimé dans les ACG basiques, et les résultats de complexité peuvent donc être conservées. Dans les cas où le type est habité par un nombre infini de termes, une implémentation pratique peut utiliser une borne supérieure au nombre possible d'extractions concommitantes, dans le même esprit que Johnson (1998); Morrill (2000) relie la charge de calcul au nombre de dépendances non résolues lors du traitement d'une phrase.

Il est également indispensable de conserver comme propriétés la normalisation et la confluence du λ-calcul utilisé, afin notamment de pouvoir calculer la forme normale d'un terme produit par le lexique. Ces propriétés sont connues pour le λ-calcul avec somme disjointe et produit dépendant Lindley (2007), ainsi que pour le λ-calcul linéaire avec produit dépendant Cervesato et Pfenning (1996), mais la combinaison de ces deux possibilités d'extensions du calcul ne dispose pas de preuve établie, et pose pour ces propriétés des problèmes non triviaux. Le chapitre suivant a donc pour objet de présenter ce λ-calcul linéaire avec produit cartésien, somme disjointe et produit dépendant, et d'en proposer une preuve de normalisation et de confluence.

Chapitre 2

β-réductions et conversions permutatives dans le λ-Π-calcul linéaire avec produit et somme

Dans ce chapitre nous définissons le λ-calcul et le système de typage associé permettant l'extension des ACG présentée dans la section 1.2.5, et nous montrons que la relation de β-réduction de ce calcul, complétée par les conversions permutatives pour la somme disjointe, est fortement normalisante et confluente. Ce λ-calcul est construit sur l'implication linéaire $\multimap$, sur une construction de types variants correspondant à la somme disjointe, sur une construction de types enregistrement correspondant au produit cartésien et sur un produit dépendant noté $(\Pi x : \alpha)\, \beta$, nous l'appellerons donc λ-Π-calcul linéaire avec produit et somme.

Le produit dépendant ne correspond pas à un type primitif de la logique linéaire. Cette construction puissante permet aux types du système de dépendre de la valeur d'un terme, à l'image des catégories syntaxiques dépendant de la valeur d'une structure de traits présentées dans la section 1.2.5. Ce mécanisme de dépendance de types envers des termes constitue l'un des trois axes de la figure du λ-cube de Barendregt (1991), classifiant les différents raffinements du système de type depuis le λ-calcul simplement typé vers le calcul des constructions, les deux autres axes correspondant aux termes dépendant de types (polymorphisme) et aux types dépendant de types (opérateurs de types). Barendregt a également démontré les propriétés de normalisation forte et de confluence pour l'intégralité du λ-cube.

La syntaxe du calcul, les règles de β-réduction, de conversions permutatives et de typage suivent ce qui a été introduit dans de Groote et Maarek (2007), à l'exception de quelques modifications permettant de simplifier les démonstrations. Notamment, afin de pouvoir obtenir la normalisation de la relation d'η-expansion (qui sera discutée au chapitre 3), le calcul doit être formulé à la Church, c'est-à-dire que les variables liées par une λ-abstraction apparaissent étiquetées par leur type. On écrira ainsi $\lambda^0 x : \alpha.t$ plutôt que $\lambda^0 x.t$. La syntaxe des analyses de cas a également été modifiée, en incorporant la garde dans le terme. On écrira par exemple $\{case\ x \text{ of } m\ x_m \to m\ x_m \mid f\ x_f \to y\}$ plutôt que $(\{m\ x_m \to m \mid f\ x_f \to y\}\ x)$. Ce changement rend les règles d'η-expansion et de conversions permutatives plus naturelles, sans modifier le calcul en profondeur.

Enfin, de Groote et Maarek (2007) proposent de traiter les types énumérés tels que $gender = \{m \mid f\}$ comme des cas particuliers de la somme disjointe $gender = \{m \text{ of } \alpha_1 | f \text{ of } \alpha_2\}$ dans laquelle les α_i sont identifiés au type unité et omis. Cette optionnalité des types introduits dans une somme disjointe se répercute sur la syntaxe des analyses de cas $\{case\ u \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n$, dans lesquelles les variables x_i peuvent donc être optionnelles. Nous n'intégrerons pas ici ce raffinement du calcul, afin de ne pas alourdir plus les notations et les démonstrations. Cette formulation des types énumérés ne modifierait néanmoins pas la substance de ces démonstrations.

Nous donnerons dans un premier temps les définitions préliminaires du calcul, puis ses règles de typage et de conversion. Les deux dernières parties consisteront à prouver la normalisation forte et la confluence de l'union des relations de β-réduction et de conversion permutative.

2.1 Définitions et notations

Cette partie comprend les règles de syntaxe, les définitions et les notations nécessaires pour introduire les règles de calcul de la partie suivante.

2.1.1 Syntaxe du calcul

Puisque les types du calcul peuvent dépendre de termes, il est nécessaire d'introduire un troisième niveau, celui des sortes (*kind*), permettant d'instaurer un contrôle sur ces combinaisons de types et de termes, de la même façon que les combinaisons de termes sont contrôlées par leur type.

La syntaxe du calcul comprend également les règles de formation d'un contexte Γ, qui est utilisé dans les règles de typage et d'η-expansion pour garder trace du type des variables libres, et d'une signature Σ, qui répertorie les constantes et types atomiques susceptibles d'être utilisés, et leur attribue un type ou une sorte. Ces signatures correspondent donc bien au concept de signature considéré dans la section 1.2.5.

Définition 10 (Syntaxe du calcul)**.**

On suppose qu'on dispose d'un ensemble infini de variables X, d'un ensemble fini de constantes C, d'un ensemble fini d'injections C', d'un ensemble fini de projections L et d'un ensemble fini de types atomiques A. La syntaxe du calcul est définie par induction comme :

Signatures		
$\Sigma ::=$	$.$	*(Signature vide)*
	$\mid \Sigma; A : \mathscr{K}$	*(Déclaration de type atomique)*
	$\mid \Sigma; A = [L : \mathscr{T}, \cdots, L : \mathscr{T}] :$ **`type`**	*(Déclaration de type enregistrement)*
	$\mid \Sigma; A = \{C' \text{ of } \mathscr{T} \mid \ldots \mid C' \text{ of } \mathscr{T}\} :$ **`type`**	*(Déclaration de type variant)*
	$\mid \Sigma; C : \mathscr{T}$	*(Déclaration de constante)*
Sortes		
$\mathscr{K} ::=$	**`type`**	*(Sorte des types)*
	$\mid (\mathscr{T})\mathscr{K}$	*(Sorte des types dépendants)*
Types		
$\mathscr{T} ::=$	A	*(Type atomique)*
	$\mid (\lambda X : \mathscr{T}.\mathscr{T})$	*(Abstraction de type)*
	$\mid (\mathscr{T}\, T)$	*(Application de type)*
	$\mid (\mathscr{T} \multimap \mathscr{T})$	*(Type fonctionnel linéaire)*
	$\mid (\Pi\, X : \mathscr{T})\mathscr{T}$	*(Produit dépendant)*
Termes		
$T ::=$	C	*(Constante)*
	$\mid X$	*(Variable)*
	$\mid (\lambda^0 X : \mathscr{T}.T)$	*(Abstraction linéaire)*
	$\mid (\lambda X : \mathscr{T}.T)$	*(Abstraction non-linéaire)*
	$\mid (T\, T)$	*(Application)*
	$\mid [L = T; \ldots; L = T]$	*(Enregistrement)*
	$\mid T.L$	*(Sélection)*
	$\mid \{case\ T \text{ of } (C'\, X) \to T \mid \ldots \mid (C'\, X) \to T\}$	*(Analyse de cas)*
	$\mid C'$	*(Injection)*
Contextes		
$\Gamma ::=$	$.$	*(Contexte vide)*
	$\mid \Gamma, X : \mathscr{T}$	*(typage de variable)*

Notation 1.

1. *Dans la suite, les variables seront notées $x, x_i, x', \ldots, y, \ldots, z, \ldots$, les constantes $c, c', \ldots, d, \ldots$, les injections $c_i, c_i' \ldots, d_i, \ldots$, les projections $l_i, l_i', \ldots$, les types atomiques $a, a', \ldots, b, \ldots$, les sortes $K, K', \ldots$, les types $\alpha, \alpha_i, \alpha', \ldots, \beta, \ldots, \gamma, \ldots$ et les termes $t, t_i, t', \ldots, u, \ldots, v, \ldots, w, \ldots, k, \ldots$. Un type enregistrement, de la forme $[l_1 : \alpha_1, \ldots, l_n : \alpha_n]$, sera noté R. Un type variant, de la forme $\{c_1 \text{ of } \alpha_1 \mid \ldots \mid c_n \text{ of } \alpha_n\}$, sera noté V.*

2. *Nous appellerons objet ce qui peut être construit sur la syntaxe du calcul, noté* $o, o_i, \ldots$.
3. *le terme* $(\ldots(t_1\ t_2)\ldots t_n)$ *sera noté* $t_1\ t_2 \ldots t_n$, *le type* $\alpha_1 \multimap (\alpha_2 \multimap (\ldots \alpha_n)\ldots)$ *sera noté* $\alpha_1 \multimap \alpha_2 \ldots \multimap \alpha_n$.
4. *Le terme* $\{case\ u\ of\ (c_1\ x_1) \to t_1 | \ldots | (c_n\ x_n) \to t_n\}$ *sera noté* $\{case\ u\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n$ *à chaque fois que les formes respectives des termes* t_i *ne seront pas considérées. De la même façon, le terme* $[l_1 = t_1; \ldots; l_n = t_n]$ *sera noté* $[l_i = t_i]_{i=1}^n$.
5. *Les parenthèses seront omises à chaque fois que leur absence ne crée pas d'ambiguïté.*

2.1.2 Relations de conversion

Les définitions suivantes introduisent l'ensemble des relations entre termes et entre types utilisées dans le calcul.

Définition 11 (Relations de conversion)**.**

1. *La relation de* β*-réduction entre termes ou entre types* $\longrightarrow_\beta$ *est définie par les règles de la section 2.2.2. La relation de* γ*-conversion entre termes ou types* $\longrightarrow_\gamma$ *est définie par les règles de la section 2.2.3. La relation d'*η*-expansion* $\longrightarrow_\eta$ *(et sa relation associée* $\longrightarrow_c$ *) entre termes ou entre types est définie par les règles de la section 2.2.5. Ces relations* $\longrightarrow_\eta$ *et* $\longrightarrow_c$ *ne sont définies que sous une signature, un contexte et un type (si les objets mis en relation sont des termes) ou une sorte (si les objets mis en relation sont des types) donnés, et on les notera sous les formes :*
   – $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta u : \alpha$ *;*
   – $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c u : \alpha$ *;*
   – $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \beta : K$ *;*
   – $\Gamma \vdash_\Sigma \alpha \longrightarrow_c \beta : K$.

   La relation d'équivalence des types des abstractions $\approx$ *est définie par les règles de la section 2.2.6. Là encore, il s'agit d'une relation entre termes ou entre types sous une signature et un contexte non-linéaire (mais pas un type ou une sorte) donnés.*

2. *La relation* $\longrightarrow_{\beta\gamma}$ *est définie comme l'union des relations* $\longrightarrow_\beta$ *et* $\longrightarrow_\gamma$.

3. *Pour un contexte et un type ou une sorte donnés, la relation* $\longrightarrow_{\beta\gamma\eta}$ *est définie comme l'union des relations* $\longrightarrow_{\beta\gamma}$ *et* $\longrightarrow_\eta$*. On a donc :*

   $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_{\beta\gamma\eta} u : \alpha$ *si et seulement si* $t \longrightarrow_{\beta\gamma} u$ *ou* $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta u : \alpha$ *;*

   $\Gamma \vdash_\Sigma \alpha \longrightarrow_{\beta\gamma\eta} \beta : K$ *si et seulement si* $\alpha \longrightarrow_{\beta\gamma} \beta$ *ou* $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \beta : K$.

4. *Les relations* $\longrightarrow_{\beta\gamma\approx}$ *et* $\longrightarrow_{\beta\gamma\eta\approx}$ *sont définies de manière similaire comme l'union des relations apparaissant en indice.*

Définition 12 (Clôture transitive réflexive)**.**

1. *Pour toute relation* R *entre termes ou entre types, on notera de la manière classique* R^n *la relation obtenue par* n *compositions de*R*. Ainsi par exemple* $\longrightarrow_\eta^n$ *est définie comme :*

   $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_{\eta,\alpha}^n u$ *si et seulement si* $\exists v_1, \ldots, v_n$ *tels que*
   $t = v_1, u = v_n$, *et* $\forall i \in \{1, \ldots, n-1\},\ \Gamma\,;\,\Delta \vdash_\Sigma v_i \longrightarrow_{\eta,\alpha} v_{i+1}$.

2. *Pour toute relation* R *entre termes ou entre types, on définit de la manière classique* R^* *comme la clôture réflexive transitive de* R *:*
   $a\ R^*\ b$ *si et seulement si* $\exists n \in \mathbf{N}$ *tel que* $a\ R^n\ b$

Ces relations de conversion peuvent à présent être utilisées pour définir une relation d'équivalence entre termes et entre types, définie de manière classique comme la clôture symétrique réflexive transitive de l'union de ces conversions. Le caractère réflexif de cette clôture est mis entre parenthèse car il est redondant avec le fait que toutes les relations de conversion que l'on considère sont elles-mêmes réflexives.

Définition 13 (Relation de $\beta\gamma\eta \approx$-équivalence)**.**

On définit $\equiv$ *comme la clôture symétrique (réflexive) transitive de* $\longrightarrow_{\beta\gamma\eta\approx}$*, c'est-à-dire :*

$\Gamma\,;\Delta \vdash_\Sigma t \equiv u : \alpha$ *si et seulement si* $\exists v_1, \ldots, v_n$ *tels que*

$t = v_1, u = v_n$, *et* $\forall i \in \{1, \ldots, n-1\}$:

$v_i \longrightarrow_{\beta\gamma} v_{i+1}$ *ou* $v_{i+1} \longrightarrow_{\beta\gamma} v_i$ *ou* $\Gamma\,;\Delta \vdash_\Sigma v_i \longrightarrow_\eta v_{i+1} : \alpha$ *ou* $\Gamma\,;\Delta \vdash_\Sigma v_{i+1} \longrightarrow_\eta v_i : \alpha$ *ou* $\Gamma \vdash_\Sigma v_i \approx v_{i+1}$ *ou* $\Gamma \vdash_\Sigma v_{i+1} \approx v_i$.

De même, $\Gamma \vdash_\Sigma \alpha \equiv_K \beta$ *si et seulement si* $\exists \gamma_1, \ldots, \gamma_n$ *tels que*

$\alpha = \gamma_1, \beta = \gamma_n, \forall i \in \{1, \ldots, n-1\}$:

$\gamma_i \longrightarrow_{\beta\gamma} \gamma_{i+1}$ *ou* $\gamma_{i+1} \longrightarrow_{\beta\gamma} \gamma_i$ *ou* $\Gamma \vdash_\Sigma \gamma_i \longrightarrow_\eta \gamma_{i+1} : K$ *ou* $\Gamma \vdash_\Sigma \gamma_{i+1} \longrightarrow_\eta \gamma_i : K$ *ou* $\Gamma \vdash_\Sigma \gamma_i \approx \gamma_{i+1}$ *ou* $\Gamma \vdash_\Sigma \gamma_{i+1} \approx \gamma_i$.

Les démonstrations des propriétés de $\longrightarrow_\eta$, qui sont données au chapitre 3, s'appuient sur le fait que l'η-expansion ne crée pas de $\beta\gamma$-rédex. Cette propriété n'est néanmoins pas vérifiée lorsque le terme η-expansé est une analyse de cas, comme nous le montrerons au chapitre 3. Pour remédier à cet inconvénient, nous définissons une η-expansion procédant implicitement aux conversions permutatives et β-réductions immédiates qu'elle peut générer lorsqu'elle s'applique à une analyse de cas. Pour ce faire, on introduit la notation $\downarrow$ pour noter les termes dans lesquels les rédex ont été propagés dans les analyses de cas et évalués.

Définition 14 (Réduction propagée)**.**

1. $t \downarrow v$ *est défini inductivement de la façon suivante :*

$$\begin{array}{ll}
(\lambda^0 y.u) \downarrow v & := u[y := v] \\
(\lambda x.u) \downarrow v & := u[y := v] \\
\{case\ u\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \downarrow v & := \{case\ u\ of\ (c_i\ x_i) \to t_i \downarrow v\}_{i=1}^n \\
t \downarrow v & = t\ v\ \textit{si}\ t\ \textit{n'est pas d'une des formes précédentes.}
\end{array}$$

2. $t \downarrow l_j$ *est défini inductivement de la façon suivante :*

$$\begin{array}{ll}
[l_i = t_i]_{i=1}^n \downarrow l_j & := t_j \\
\{case\ u\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \downarrow l_j & := \{case\ u\ of\ (c_i\ x_i) \to t_i \downarrow l_j\}_{i=1}^n \\
t \downarrow l_i & := t.l_i\ \textit{si}\ t\ \textit{n'est pas d'une des formes précédentes.}
\end{array}$$

2.1.3 Constantes et variables

Nous donnons à présent la définition de trois fonctions partielles, $type_\Sigma, kind_\Sigma$ et $binding_\Sigma$, permettant respectivement d'extraire d'une signature le type d'une constante, la sorte d'un type atomique ou le type enregistrement ou variant correspondant à un type atomique.

Définition 15 (Accès à la signature)**.**

Les fonctions partielles $type_\Sigma, kind_\Sigma$ *et* $binding_\Sigma$ *sont définies inductivement comme :*

1. $type_\Sigma$ *:*

$$\begin{array}{ll}
type_{()}(c) & \textit{n'est pas défini} \\
type_{\Sigma;a:K}(c) & := type_\Sigma(c) \\
type_{\Sigma;a=R:type}(c) & := type_\Sigma(c) \\
type_{\Sigma;a=V:type}(c) & := type_\Sigma(c) \\
type_{\Sigma;c:\alpha}(c) & := \alpha \\
type_{\Sigma;d:\alpha}(c) & := type_\Sigma(c)\ \textit{si}\ c \neq d
\end{array}$$

2. $kind_\Sigma$:

$kind_{()}(a)$	*n'est pas défini*
$kind_{\Sigma;a:K}(a)$	$:= K$
$kind_{\Sigma;b:K}(a)$	$:= kind_\Sigma(a)$ *si* $a \neq b$
$kind_{\Sigma;a=R:type}(a)$	$:=$ **`type`**
$kind_{\Sigma;b=R:type}(a)$	$:= kind_\Sigma(a)$ *si* $a \neq b$
$kind_{\Sigma;a=V:type}(a)$	$:=$ **`type`**
$kind_{\Sigma;b=V:type}(a)$	$:= kind_\Sigma(a)$ *si* $a \neq b$
$kind_{\Sigma;c:\alpha}(a)$	$:= kind_\Sigma(a)$

3. $binding_\Sigma$:

$binding_{()}(a)$	*n'est pas défini*
$binding_{\Sigma;a:K}(a)$	*n'est pas défini*
$binding_{\Sigma;b:K}(a)$	$:= binding_\Sigma(a)$ *si* $a \neq b$
$binding_{\Sigma;a=R:type}(a)$	$:= R$
$binding_{\Sigma;b=R:type}(a)$	$:= binding_\Sigma(a)$ *si* $a \neq b$
$binding_{\Sigma;a=V:type}(a)$	$:= V$
$binding_{\Sigma;b=V:type}(a)$	$:= binding_\Sigma(a)$ *si* $a \neq b$
$binding_{\Sigma;c:\alpha}(a)$	$:= binding_\Sigma(a)$

Les variables doivent être distinguées entre variables libres et liées. Les variables libres sont les variables qui apparaissent dans un terme, un type, une sorte, un contexte ou une signature sans être liées par un symbole mutificateur.

Définition 16 (Variables libres)**.**

1. *L'ensemble des variables libres FV d'un terme t ou d'un type α est défini inductivement comme :*

$FV(a)$	$:= \emptyset$
$FV(\lambda x : \alpha.\beta)$	$:= (FV(\beta) \backslash \{x\}) \cup FV(\alpha)$
$FV(\alpha\, t)$	$:= FV(\alpha) \cup FV(t)$
$FV(\alpha \multimap \beta)$	$:= FV(\alpha) \cup FV(\beta)$
$FV((\Pi\, x : \alpha)\beta)$	$:= (FV(\beta) \backslash \{x\}) \cup FV(\alpha)$
$FV(c)$	$:= \emptyset$
$FV(x)$	$:= \{x\}$
$FV(\lambda^0 x : \alpha.t)$	$:= (FV(t) \backslash \{x\}) \cup FV(\alpha)$
$FV(\lambda x : \alpha.t)$	$:= (FV(t) \backslash \{x\}) \cup FV(\alpha)$
$FV(t\, u)$	$:= FV(t) \cup FV(u)$
$FV([l_i = t_i]_{i=1}^n)$	$:= \cup_{i=1}^n FV(t_i)$
$FV(t.l)$	$:= FV(t)$
$FV(\{case\ u\ of\ (c_i\, x_i) \rightarrow t_i\}_{i=1}^n)$	$:= \cup_{i=1}^n (FV(t_i) \backslash \{x_i\})$
$FV(c_i)$	$:= \emptyset$

2. *L'ensemble des variables libres FV d'une sorte, d'une signature ou d'un contexte est défini comme l'union de l'ensemble des variables libres des termes et types qui y apparaissent.*

Dans les jugements de la partie suivante, les types des variables libres présentes dans un terme ou un type seront donnés par un contexte de typage. Nous verrons également que ces mêmes jugements ne permettent pas aux signatures et aux sortes bien formées de contenir des variables libres.

Les variables liées sont toutes les variables qui ne sont pas libres, et qui sont donc introduites par un symbole mutificateur. Nous adopterons l'identification usuelle des objets par α-renommage, c'est-à-dire que le nom donné à une variable liée ne permet pas de distinguer deux objets : ainsi nous traiterons $\lambda x : \alpha.x$ et $\lambda y : \alpha.y$ comme le même terme.

Notation 2 (Substitution sans capture)**.**

Nous noterons $t[x := u]$ le résultat de la substitution sans capture usuelle de x par u dans t, où chaque occurence de la variable libre x dans t est remplacée par le terme u, sans remplacer aucune variable liée. On utilisera de la même façon la notation $\alpha[x := u]$ pour une substitution dans un type.

Le domaine d'un contexte correspond à l'ensemble des variables dont le contexte fournit un type :

Définition 17 (Domaine d'un contexte)**.**
Le domaine $dom(\Gamma)$ *d'un contexte de typage* Γ *est défini inductivement comme :*

$$\begin{array}{ll} dom(.) & := \emptyset \\ dom(\Gamma', x : \alpha) & := \{x\} \cup dom(\Gamma') \end{array}$$

2.2 Règles du calcul

Nous donnons dans cette section les différentes règles utilisées par le calcul : typage, β-réduction, γ-conversion, calcul des formes normales, η-expansion et $\approx$-équivalence. Chacun de ces ensembles de règles sera présenté sous la forme d'un système de preuve formel.

2.2.1 Le système de typage

Le système de typage s'assure que tous les objets construits dans le calcul respectent la discipline imposée par les sortes, les types et les déclarations de la signature. Les termes et types ont notamment besoin d'être reconnus par le système de typage pour que leur normalisation soit garantie.

En plus des règles usuelles de structure, d'introduction et d'élimination, ce système de typage comprend une règle (eq.) permettant de tenir compte du fait que les types attribués aux termes peuvent eux-même être convertis.

Pour garder trace des types affectés aux variables libres, on utilise de façon usuelle des contextes de typage. A cause de la présence dans le calcul d'abstractions linéraires et non linéaires, ces contextes sont également distingués entre contexte linéaires, notés Δ, qui ne peuvent pas être dupliqués ni affaiblis, et les contextes non linéaires, notés Γ, qui peuvent être dupliqués et affaiblis. Le système de typage est constitué de quatre formes de jugements :

- La signature Σ est bien formée :
  $\text{sig}\,(\Sigma)$;
- La sorte K est bien formée dans la signature Σ :
  $\vdash_\Sigma \mathrm{K} : \mathtt{kind}$;
- α est un type de sorte K dans la signature Σ et le contexte non linéaire Γ :
  $\Gamma \vdash_\Sigma \alpha : \mathrm{K}$;
- t est un terme de type α dans la signature Σ, le contexte non linéaire Σ et le contexte linéaire Δ :
  $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$.

Ces jugements sont définis par une induction mutuelle de la façon suivante :

Signatures bien formées

$$\frac{}{\text{sig}\,(.)}$$

$$\frac{\text{sig}\,(\Sigma) \qquad \vdash_\Sigma \mathrm{K} : \mathtt{kind}}{\text{sig}\,(\Sigma;\, a : \mathrm{K}\,)}$$

$$\frac{\text{sig}\,(\Sigma) \qquad .\vdash_\Sigma \alpha_1 : \mathtt{type} \quad \cdots \quad .\vdash_\Sigma \alpha_n : \mathtt{type}}{\text{sig}\,(\Sigma;\, a = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] : \mathtt{type}\,)}$$

$$\frac{\text{sig}\,(\Sigma) \qquad .\vdash_\Sigma \alpha_1 : \mathtt{type} \quad \cdots \quad .\vdash_\Sigma \alpha_n : \mathtt{type}}{\text{sig}\,(\Sigma;\, a = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} : \mathtt{type}\,)}$$

$$\frac{\mathrm{sig}\,(\Sigma) \qquad \vdash_\Sigma \alpha : \mathbf{type}}{\mathrm{sig}\,(\Sigma;\, c : \alpha)}$$

Dans toutes les règles précédentes, les symboles introduits $(a, l_1, \ldots, l_n, c_1, \ldots, c_n, c)$ ne doivent pas déjà apparaître dans Σ.

Ces règles définissent quelles déclarations peut contenir une signature bien formée. Une constante peut être déclarée de type α si le type α est bien formé, de sorte **type**, sous un contexte vide, ce qui implique qu'il ne contient pas de variables libres. Il peut néanmoins contenir des constantes et types atomiques déclarés plus tôt dans la signature. De la même façon, un type atomique peut être déclaré de sorte K, si K est bien formée. Un type atomique peut également être déclaré comme type variant ou enregistrement, et sa sorte est alors **type**. Là encore, les types sur lesquels sont construits ce type variant ou enregistrement doivent être bien formé sous un contexte vide, et ne contiennent pas de variables libres. Cette déclaration en signature est la seule façon de contruire un type variant ou enregistrement.

Sortes bien formées

$$\frac{}{\vdash_\Sigma \mathbf{type} : \mathbf{kind}}$$

$$\frac{. \vdash_\Sigma \alpha : \mathbf{type} \qquad \vdash_\Sigma \mathrm{K} : \mathbf{kind}}{\vdash_\Sigma (\alpha)\,\mathrm{K} : \mathbf{kind}}$$

type et **kind** sont des mots-clés particuliers du calcul : **type** est la sorte attribuée aux types ne dépendant d'aucun terme. **kind** est le mot-clé qualifiant une sorte bien formée dans un jugement. Les règles ci-dessus définissent quelles sont ces sortes bien formées. La seconde permet d'avoir des sortes de la forme $(\alpha)\,\mathrm{K}$, qui sont attribuées aux types qui peuvent être appliqués à un terme de type α pour obtenir un type de sorte K. Dans une telle sorte, le type α doit être de sorte **type** sous un contexte vide, ce qui assure qu'une sorte ne contient jamais de variable libre.

Types bien formés

$$\frac{}{. \vdash_\Sigma a : kind_\Sigma(a)} \text{ (type const.)}$$

$$\frac{. \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta : \mathrm{K}}{\Gamma, x : \alpha \vdash_\Sigma \beta : \mathrm{K}} \text{ (type weak.)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma \beta : \mathrm{K}}{\Gamma \vdash_\Sigma \lambda x : \alpha.\beta : (\alpha)\,\mathrm{K}} \text{ (type abs.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : (\beta)\,\mathrm{K} \qquad \Gamma; . \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \alpha\, t : \mathrm{K}} \text{ (type app.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta : \mathbf{type}}{\Gamma \vdash_\Sigma \alpha \multimap \beta : \mathbf{type}} \text{ (lin. fun.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma, x : \alpha \vdash_\Sigma \beta : \mathbf{type}}{\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta : \mathbf{type}} \text{ (dep. prod.)}$$

Dans la règle (type weak), x ne doit pas déjà apparaître dans Γ.

Ces règles définissent les types bien formés. Les jugements se font sous un contexte non-linéaire Γ, car les types peuvent contenir des variables libres, au sein des termes dont ils dépendent. Ces termes ne peuvent néanmoins pas contenir de variable répertoriée dans le contexte linéaire, ce qui apparait dans la règle (type app.). Comme les types peuvent contenir des variables, le calcul dispose d'une règle de λ-abstraction (type abs.) sur ces variables. Cette règle ne doit pas être confondue avec la règle du produit dépendant, (dep. prod.) : un type de la forme $\lambda x : \alpha.\beta$ peut être appliqué à un terme t de type α pour obtenir après β-réduction le type $\beta[x := t]$. En revanche un type de la forme $(\Pi x : \alpha)\,\beta$ ne peut pas être appliqué à un terme, puisqu'il est de sorte **type**. C'est le terme t auquel on attribuera ce type qui pourra être appliqué à un terme u de type α, pour obtenir un terme $t\ u$ de type $\beta[x := u]$. $(\Pi x : \alpha)\,\beta$ est donc un type fonctionnel, et correspond d'ailleurs exactement à l'implication intuitionniste $\alpha \to \beta$ lorsque x n'apparait pas dans β.

Termes bien typés

$$\frac{}{.\,;\,. \vdash_\Sigma c : type_\Sigma(c)}\ \text{(const.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type}}{\Gamma\,;\, x : \alpha \vdash_\Sigma x : \alpha}\ \text{(lin. var.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type}}{\Gamma, x : \alpha\,;\,. \vdash_\Sigma x : \alpha}\ \text{(var.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t : \beta}{\Gamma, x : \alpha\,;\,\Delta \vdash_\Sigma t : \beta}\ \text{(weak.)}$$

$$\frac{\Gamma\,;\,\Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma\,;\,\Delta \vdash_\Sigma \lambda^0 x : \alpha.t : \alpha \multimap \beta}\ \text{(lin. abs.)}$$

$$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \qquad \Gamma\,;\,\Delta_2 \vdash_\Sigma u : \alpha}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma t\,u : \beta}\ \text{(lin. app.)}$$

$$\frac{\Gamma, x : \alpha\,;\,\Delta \vdash_\Sigma t : \beta}{\Gamma\,;\,\Delta \vdash_\Sigma \lambda x : \alpha.t : (\Pi x : \alpha)\,\beta}\ \text{(abs.)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma t : (\Pi x : \alpha)\,\beta \qquad \Gamma\,;\,\vdash_\Sigma u : \alpha}{\Gamma\,;\,\Delta \vdash_\Sigma t\,u : \beta[x := u]}\ \text{(app.)}$$

$$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \Gamma\,;\,\Delta \vdash_\Sigma t_1 : \alpha_1 \quad \cdots \quad \Gamma\,;\,\Delta \vdash_\Sigma t_n : \alpha_n}{\Gamma\,;\,\Delta \vdash_\Sigma [l_i = t_i]_{i=1}^n : a}\ \text{(rec.)}$$

$$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \Gamma\,;\,\Delta \vdash_\Sigma t : a}{\Gamma\,;\,\Delta \vdash_\Sigma t.l_i : \alpha_i}\ \text{(sel.)}$$

$$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma u : a \qquad binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \qquad \forall i \in \{1, \ldots, n\}, \Gamma\,;\,\Delta_2, x_i : \alpha_i \vdash_\Sigma t_i : \beta}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma \{case\ u \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n : \beta}\ \text{(case)}$$

$$\frac{binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\}}{.\,;\,. \vdash_\Sigma c_i : \alpha_i \multimap a} \text{ (inj.)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha \qquad \Gamma \vdash_\Sigma \alpha \equiv \beta \qquad \Gamma \vdash_\Sigma \beta : \mathtt{type}}{\Gamma\,;\,\Delta \vdash_\Sigma t : \beta} \text{ (eq.)}$$

Dans les règles (lin. var.) et (var.), x ne doit pas déjà apparaître dans Γ. Dans la règle (abs.), x ne doit pas apparaître libre dans Δ. Dans la règle (weak), x ne doit pas déjà apparaître dans Γ.

Ces règles indiquent comment obtenir des termes bien typés. Ce jugement fait intervenir les deux types de contextes : les termes peuvent contenir des abstractions linéaires et non-linéaires, et le type des variables doit être tracé dans chacun de ces cas. Les variables liées par une abstraction linéaire doivent apparaitrent une et une seule fois dans le terme, et cette propriété est vérifiée par le traitement du contexte linéaire dans les règles, le contexte Δ apparaissant dans la conclusion étant réparti sans créer de doublons dans les différentes prémisses. La seule possibilité d'introduire une variable dans ce contexte linéaire est donc d'utiliser la règle (lin. var.). La règle (case) en est une autre, dans laquelle le contexte linéaire est repris par les n prémisses typant les branches d'une analyse de cas. Ceci est dû au fait que ces branches décrivent des possibilités mutuellement exclusives, et donc que la sensibilité aux ressources capturée par cette linéarité doit se manifester par l'apparition d'une variable linéaire donnée dans toutes les branches, et non dans une seule. De plus, dans cette règle (case) apparaît le fait que chacune de ces n branches est liée à une variable x_i par un mécanisme similaire à l'abstraction linéaire, et cette variable x_i apparaitra donc une et une seule fois dans sa branche. Dans la même logique, la règle (rec.) voit également le contexte linéraire dupliqué dans les prémisses.Par ailleurs, dans la règle (app.), le contexte linéaire est entièrement passé du côté du terme de gauche, car le terme de droite est susceptible d'être dupliqué par la substitution entraînée par une β-réduction impliquant une abstraction non-linéaire. Ainsi, dans le terme $(\lambda x : \alpha.y\ x\ x)\ z$, on ne souhaite pas que z puisse être vue comme une variable linéaire, et abstraite linéairement en conséquence, car ce terme se β-réduit vers $y\ z\ z$, où z apparaît plus d'une fois.

Il est important de noter que l'ordre du contexte non-linéaire est important : en effet le type affecté à une variable peut contenir n'importe quelle variable apparaissant précédemment dans le domaine du contexte, mais pas une variable apparaissant après. En revanche l'ordre du contexte non-linéaire n'est pas pertinent. En effet, les types apparaissant dans ce contexte peuvent contenir des variables déclarées dans le contexte non-linéaire, mais pas de variables déclarées dans le contexte linéaire, le jugement d'un type bien formé ne prenant pas de contexte linéaire en considération. Le contexte linéaire doit donc être traité comme un ensemble non ordonné plutôt que comme une liste. L'introduction d'une règle de permutation permettrait au système de règles de rendre compte de ce comportement, mais alourdirait les preuves. On supposera plutôt , dans les règles précédentes ainsi que dans la suite, que l'ordre d'un contexte linéaire peut être modifié arbitrairement. Par contre, la règle (weak) n'est définie que pour le contexte non-linéaire, puisqu'elle permet de supprimer une variable du contexte sans qu'elle apparaisse dans le terme typé, ce qui violerait la condition de linéarité.

Dans la règle (inj.), c_i dénote une injection, liée à un type variant, et non une constante. Une telle injection reçoit un type fonctionnel, pour pouvoir prendre en argument un terme de type variant.

Il faut noter également que les règles s'assurent que seuls les types de sorte **`type`** peuvent être attribués à des termes. Ces types peuvent néanmoins contenir des termes, et donc subir des réécritures. La règle (eq.) permet de s'assurer que la modification d'un type par la relation $\longrightarrow_{\beta\gamma\eta}$ n'altère pas le fait qu'il puisse être attribué à un terme. Supposons par exemple qu'on dispose dans la signature Σ de la déclaration de constante

$$f : (\Pi y : \alpha \multimap \alpha)\ a\ (y\ c)$$

avec c une constante de type α et a un type atomique de sorte (α) **`type`**. Par la règle (app.), le type du terme $t = f\ (\lambda^0 x : \alpha.x)$ sera $a\ ((\lambda^0 x : \alpha.x)\ c)$. On constate alors que ce type contient un β-rédex, et la règle (eq.) permet d'obtenir pour t le type $a\ c$.

2.2.2 Le système de β-réduction

La règle de β-réduction usuelle du λ-calcul consiste à évaluer l'application d'une λ-abtraction $\lambda x.t$ à un argument u en substituant x par u dans t :

$(\lambda x.t)\ u \longrightarrow_\beta t[x := u]$.

Cette règle reste valide si la λ-abstraction considérée est linéaire. Puisque les types peuvent contenir des variables et être appliqués à des termes, cette notion de β-réduction est également étendue aux abstractions de type :

$(\lambda x.\alpha)\ u \longrightarrow_\beta \alpha[x := u]$.

Par extension, nous rangerons sous cette étiquette de β-réduction les règles de réduction correspondant à la somme disjointe et au produit cartésien :

- $[l_i = t_i]_{i=1}^n.l_j \longrightarrow_\beta t_j$;
- $\{case\ c_j\ k \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \longrightarrow_\beta t_j[x_j := k]$.

Dans la suite, et à la manière de Cervesato et Pfenning (1996), nous présentons ces règles de β-réduction sous la forme d'une réduction parallèle, dans laquelle on peut réduire en une étape plusieurs rédex s'ils se trouvent dans des sous-termes distincts. Ainsi par exemple la réduction

$((\lambda x : \alpha.x)\ t)\ ([l_i = u_i]_{i=1}^n.l_j) \longrightarrow_\beta t\ u_j$

est faite en une seule étape.

Cette formulation permet d'obtenir la propriété du diamant, exprimée et prouvée dans le lemme 8 et qui n'est pas vérifiée pour une présentation non parallèle de la β-réduction : ainsi par exemple les deux réductions divergentes

1. $(\lambda x : \alpha.y\ x\ x)\ ((\lambda z : \alpha.z)\ t) \longrightarrow_\beta (\lambda x : \alpha.y\ x\ x)\ t$
2. $(\lambda x : \alpha.y\ x\ x)\ ((\lambda z : \alpha.z)\ t) \longrightarrow_\beta y\ ((\lambda z : \alpha.z)\ t)\ ((\lambda z : \alpha.z)\ t)$

ne convergent pas en une seule étape si les réductions ne peuvent se faire en parallèle. Dans le second cas le terme $((\lambda z : \alpha.z)\ t)$ a été dupliqué, et il faut donc une étape pour évaluer chacune de ses occurrences et obtenir le terme $y\ t\ t$.

Cette formulation parallèle explicite les règles de congruence, ce qui permet de conserver une homogénéité de présentation avec les règles d'η-expansion, où les règles de congruence sont particulières et doivent être explicitées.

Les règles de β-réductions n'utilisent pas d'informations de typage, et les contextes de typage n'y apparaissent donc pas.

Règles de congruence

$$\frac{}{c \longrightarrow_\beta c}$$

$$\frac{}{x \longrightarrow_\beta x}$$

$$\frac{t \longrightarrow_\beta t' \qquad u \longrightarrow_\beta u'}{t\,u \longrightarrow_\beta t'\,u'}$$

$$\frac{t \longrightarrow_\beta t' \qquad \alpha \longrightarrow_\beta \alpha'}{\lambda^0 x : \alpha.t \longrightarrow_\beta \lambda^0 x : \alpha'.t'}$$

$$\frac{t \longrightarrow_\beta t' \qquad \alpha \longrightarrow_\beta \alpha'}{\lambda x : \alpha.t \longrightarrow_\beta \lambda x : \alpha'.t'}$$

$$\frac{t_1 \longrightarrow_\beta t'_1 \ \cdots \ t_n \longrightarrow_\beta t'_n}{[l_i = t_i]_{i=1}^n \longrightarrow_\beta [l_i = t'_i]_{i=1}^n}$$

$$\frac{t \longrightarrow_\beta t'}{t.l_i \longrightarrow_\beta t'.l_i}$$

$$\frac{t_1 \longrightarrow_\beta t'_1 \ \cdots \ t_n \longrightarrow_\beta t'_n \qquad u \longrightarrow_\beta u'}{\{case\ u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n \longrightarrow_\beta \{case\ u' \text{ of } (c_i\, x_i) \to t'_i\}_{i=1}^n}$$

$$a \longrightarrow_\beta a$$

$$\frac{\alpha \longrightarrow_\beta \alpha' \qquad t \longrightarrow_\beta t'}{\alpha\, t \longrightarrow_\beta \alpha'\, t'}$$

$$\frac{\alpha \longrightarrow_\beta \alpha' \qquad \beta \longrightarrow_\beta \beta'}{\alpha \multimap \beta \longrightarrow_\beta \alpha' \multimap \beta'}$$

$$\frac{\alpha \longrightarrow_\beta \alpha' \qquad \beta \longrightarrow_\beta \beta'}{(\Pi x : \alpha)\, \beta \longrightarrow_\beta (\Pi x : \alpha')\, \beta'}$$

$$\frac{\alpha \longrightarrow_\beta \alpha' \qquad \beta \longrightarrow_\beta \beta'}{\lambda x : \beta.\alpha \longrightarrow_\beta \lambda x : \beta'.\alpha'}$$

Règles de β-reduction

$$\frac{\beta \longrightarrow_\beta \beta' \qquad u \longrightarrow_\beta u'}{(\lambda x : \alpha.\beta)\, u \longrightarrow_\beta \beta'[x := u']}$$

$$\frac{t \longrightarrow_\beta t' \qquad u \longrightarrow_\beta u'}{(\lambda^0 x : \alpha.t)\, u \longrightarrow_\beta t'[x := u']}$$

$$\frac{t \longrightarrow_\beta t' \qquad u \longrightarrow_\beta u'}{(\lambda x : \alpha.t)\, u \longrightarrow_\beta t'[x := u']}$$

$$\frac{t_j \longrightarrow_\beta t'_j}{[l_i = t_i]_{i=1}^n.l_j \longrightarrow_\beta t'_j}$$

$$\frac{t_j \longrightarrow_\beta t'_j \qquad u \longrightarrow_\beta u'}{\{case\ c_j\, u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n \longrightarrow_\beta t_j[x_j := u]}$$

2.2.3 Le système de conversions permutatives

Les conversions permutatives permettent d'intervertir l'ordre dans lequel se combinent une analyse de cas et une autre construction. Par exemple :

$$\{case\ u \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \ \longrightarrow_\gamma\ \{case\ u \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n$$

où l'application au terme v est permutée de l'ensemble de l'analyse de cas vers chacune des branches de cette analyse. Par l'isomorphisme de Curry-Howard, ces conversions permutatives correspondent à la permutation de la règle d'élimination de la somme disjointe avec les autres règles, permettant d'identifier des preuves ne se distinguant que par la place arbitraire prise par cette règle d'élimination (Girard et al., 1989).

Les conversions permutatives standards sont celles où l'analyse de cas est en position de subir une application, une projection ou une autre analyse de cas avant la permutation. Ces permutations ont en effet comme caractéristique de pouvoir faire apparaître des β-rédex qui étaient auparavant masqués par la présence de l'analyse de cas. Par exemple :

$$\begin{aligned}&\{case\ u \text{ of } (c_i\ x_i) \to \lambda y_i : \alpha_i.t_i\}_{i=1}^n\ v\\ &\longrightarrow_\gamma\ \{case\ u \text{ of } (c_i\ x_i) \to (\lambda y_i : \alpha_i.t_i)\ v\}_{i=1}^n\\ &\longrightarrow_\beta \{case\ u \text{ of } (c_i\ x_i) \to t_i[y_i := v]\}_{i=1}^n.\end{aligned}$$

Si ces conversions permutatives standards ne sont pas ajoutées au calcul, il est donc possible pour un terme en forme normale de conserver de tels β-rédex masqués. Les arbres de typage de ces formes normales ne vérifient donc pas la propriété de la sous-formule de Groote et Maarek (2007), et ont donc a priori un problème de l'inférence de type indécidable (Dowek, 1993). Pour cette raison, nous nous attacherons dans la suite à prouver les propriétés permettant de calculer les formes $\beta\gamma$-normales, plutôt que simplement β-normales, avec γ regroupant ces conversions permutatives standards.

Les conversions permutatives non standards correspondent aux autres positions possibles pour l'analyse de cas permutée. Ces conversions permutatives permettent d'étendre l'identification des termes dénotant la même sémantique, cependant elles ne peuvent pas activer des β-rédex qui auraient été masqués par une analyse de cas, et ne sont donc pas cruciales pour le problème de l'inférence de type. De plus, la conversion permettant de permuter une analyse de cas dans un enregistrement :

$$[l_i = \{case\ u \text{ of } (c_j\ x_j) \to t_{i,j}\}_{j=1}^m]_{i=1}^n\ \longrightarrow_\gamma\ \{case\ u \text{ of } (c_j\ x_j) \to [l_i = t_{i,j}]_{i=1}^n\}_{j=1}^m$$

demande que la garde des analyses de cas qui constituent les champs de l'enregistement soient identiques. Cette condition pose de sérieuses difficultés concernant les preuves de confluences. Des problèmes de normalisation et de confluence apparaissent également avec la conversion :

$$\begin{aligned}&\{case\ u \text{ of } (c_i\ x_i) \to \{case\ v \text{ of } (d_j\ y_j) \to t_{i,j}\}_{j=1}^m\}_{i=1}^n\\ &\longrightarrow_\gamma\ \{case\ v \text{ of } (d_j\ y_j) \to \{case\ u \text{ of } (c_i\ x_i) \to t_{i,j}\}_{i=1}^n\}_{j=1}^m.\end{aligned}$$

Ces difficultés nécessitent par exemple de définir une partie de ces conversions permutative au sein d'une relation d'équivalence quotientant les relations de réécriture (Lindley, 2007). Les preuves résultantes sont cependant sérieusement complexifiées, et nous choisissons ici de ne pas intégrer ces conversions permutatives dans le calcul.

Même formalisée sous forme de réduction parallèle, $\longrightarrow_\gamma$ ne satisfait pas la propriété du diamant. Afin de simplifier les démonstrations, les règles sont donc écrites dans une formulation plus simple, où les sous-termes du terme converti ne sont pas eux-même convertis en une même étape.

Règles de congruences

Elles suivent exactement le même schéma que les règles de congruence pour $\longrightarrow_\beta$.

Règles de conversions permutatives

$$\frac{}{\{case\ u \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \ \longrightarrow_\gamma\ \{case\ u \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n}$$

$$\frac{}{\{case\ u \text{ of } (c_i\ x_i) \to t\}_{i=1}^n.l \ \longrightarrow_\gamma\ \{case\ u \text{ of } (c_i\ x_i) \to t_i.l\}_{i=1}^n}$$

$$\frac{}{\{case\ \{case\ v \text{ of } (d_j\ y_j) \to u_j\}_{j=1}^m \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \ \longrightarrow_\gamma\ \{case\ v \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\}_{j=1}^m}$$

On suppose pour toutes ces règles qu'on évite toute capture de λ-variable, l'α-renommage éventuellement nécessaire étant implicite.

2.2.4 Définition syntaxique des formes normales de $\longrightarrow_{\beta\gamma\eta}$

Nous donnons dans cette section un ensemble de règles formant un critère syntaxique pour reconnaître les formes $\beta\gamma\eta$-normales du calcul. De la même manière que Ghani (1997) et Barthe (1999a), ce critère est utilisé par les règles d'η-expansion, qui doivent en effet s'assurer de ne pas introduire dans les étiquettes des abstractions créées de nouveaux rédex. La définition directe des formes $\beta\gamma\eta$-normales n'est en effet pas utilisable dans ces règles d'η-expansion, car cela produirait une boucle de définitions mal fondée. Nous détaillerons au chapitre suivant pourquoi la normalisation de $\longrightarrow_\eta$ requiert de contrôler les étiquettes introduites par expansion.

Symétriquement à la décomposition des règles d'expansion en deux relations $\longrightarrow_\eta$ et $\longrightarrow_c$, permettant de ne pas expandre les positions qui créeraient des $\beta\gamma$-rédex, le critère syntaxique distingue les formes $\beta\gamma\eta$-normales et les formes $\beta\gamma c$-normales, ou formes normales intérieures. Cette distinction n'est néanmoins pas essentielle au niveau des types, car les règles peuvent se servir directement du fait que, par examen des cas, les types étant $\beta\gamma c$-normaux sans être $\beta\gamma\eta$-normaux sont forcément de la forme $a\ t_1\ \ldots\ t_n$. Ce critère prend donc la forme de trois jugements :

- Le type α est $\beta\gamma\eta$-normal dans le contexte Γ :
  $\Gamma \vdash_\Sigma A(\alpha)$
- Le terme t est $\beta\gamma c$-normal dans le contexte Γ, Δ :
  $\Gamma\ ;\ \Delta \vdash_\Sigma I(t)$
- le terme t est $\beta\gamma\eta$-normal dans le contexte Γ, Δ :
  $\Gamma\ ;\ \Delta \vdash_\Sigma E(t)$

Nous donnerons au chapitre suivant une preuve de la correction de ce critère (tous les objets reconnus sont $\beta\gamma\eta$-normaux), mais pas de sa complétude (tous les objets $\beta\gamma\eta$-normaux sont reconnus).

Types en forme normale

$$\frac{\Gamma \vdash_\Sigma a\ v_1\ \ldots\ v_n : \mathtt{type} \qquad \Gamma\ ;\ \vdash_\Sigma E(v_1) \cdots \Gamma\ ;\ \vdash_\Sigma E(v_n)}{\Gamma \vdash_\Sigma A(a\ v_1\ \ldots\ v_n)}$$

$$\frac{\Gamma \vdash_\Sigma A(\alpha) \qquad \Gamma, x : \alpha \vdash_\Sigma A(\beta)}{\Gamma \vdash_\Sigma A(\lambda x : \alpha.\beta)}$$

$$\frac{\Gamma \vdash_\Sigma A(\alpha) \qquad \Gamma, x : \alpha \vdash_\Sigma A(\beta)}{\Gamma \vdash_\Sigma A((\Pi x : \alpha)\ \beta)}$$

$$\frac{\Gamma \vdash_\Sigma A(\alpha) \qquad \Gamma \vdash_\Sigma A(\beta)}{\Gamma \vdash_\Sigma A(\alpha \multimap \beta)}$$

Dans la première règle, n peut valoir 0.

Termes en forme normale

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma E(t)}{\Gamma\,;\,\Delta \vdash_\Sigma I(t)}$$

$$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma I(t) \qquad \Gamma\,;\,\Delta_2 \vdash_\Sigma E(u) \qquad t \text{ n'est ni une abstraction, ni une analyse de cas}}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma I(t\,u)}$$

$$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma I(t) \qquad t \text{ n'est ni un enregistrement, ni une analyse de cas}}{\Gamma\,;\,\Delta \vdash_\Sigma I(t.l)}$$

$$\frac{\begin{array}{cc}\Gamma\,;\,\Delta_1 \vdash_\Sigma u : a & binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \\ \Gamma\,;\,\Delta_1 \vdash_\Sigma I(u) & \forall i \in \{1 \ldots n\}, \Gamma\,;\,\Delta_2, x_i : \alpha_i \vdash_\Sigma E(t_i) \\ u \text{ n'est ni une analyse de cas, ni de la forme } c_j\,k & t_i \text{ n'est ni une abstraction, ni un enregistrement}\end{array}}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma E(\{case\ u \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma I(t) \qquad \Gamma\,;\,\Delta \vdash_\Sigma t : a_\emptyset\,v_1\,\cdots v_n \qquad t \text{ n'est pas une analyse de cas}}{\Gamma\,;\,\Delta \vdash_\Sigma E(t)}$$

où $a_\emptyset$ désigne un type atomique qui n'a pas d'image par la fonction $binding_\Sigma$ (ce n'est donc pas un type enregistrement ou un type variant).

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma x : \alpha}{\Gamma\,;\,\Delta \vdash_\Sigma I(x)}$$

$$\frac{}{\Gamma\,;\,\vdash_\Sigma I(c)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma E(t) \qquad \Gamma \vdash_\Sigma A(\alpha)}{\Gamma\,;\,\Delta \vdash_\Sigma E(\lambda x : \alpha.t)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma E(t) \qquad \Gamma \vdash_\Sigma A(\alpha)}{\Gamma\,;\,\Delta \vdash_\Sigma E(\lambda^0 x : \alpha.t)}$$

$$\frac{\forall i \in \{1, \ldots, n\}, \Gamma\,;\,\Delta \vdash_\Sigma E(t_i)}{\Gamma\,;\,\Delta \vdash_\Sigma E([l_i = t_i]_{i=1}^n)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma E(k)}{\Gamma\,;\,\Delta \vdash_\Sigma E(c_i\,k)}$$

2.2.5 Le système d'η-expansion

Les règles d'η-expansion nécessitent de disposer du contexte de typage et du type ou de la sorte de l'objet auquel elles s'appliquent. Elles se présentent donc comme un enrichissement du système de typage. Les règles de typage, marquées $\longrightarrow_c$, se comportent comme des règles de congruence. Les règles d'η-expansion, marquées $\longrightarrow_\eta$, ne contribuent pas au typage mais réalisent une η-expansion au niveau où elles apparaissent. Cette distinction entre la relation $\longrightarrow_\eta$ et la relation auxiliaire $\longrightarrow_c$ empêche donc les règles d'expansion d'être utilisées dans des position où elles créeraient de nouveaux β-rédex : la partie gauche d'une application, le corps d'une projection et la garde d'un cas, ce qui rendrait le calcul non normalisable. Ce système comprend donc quatre jugements :

- Le terme t s'η-expand vers t' sous le contexte $\Gamma;\Delta$ et avec le type α :
  $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \alpha$
- Le terme t s'η-expand, mais pas au niveau le plus extérieur, vers t' sous le contexte $\Gamma;\Delta$ et avec le type α :
  $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \alpha$
- Le type α s'η-expand vers α' sous le contexte Γ et avec la sorte K :
  $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathrm{K}$
- Le type α s'η-expand, sans expansion du type entier, vers α' sous le contexte Γ et avec la sorte K :
  $\Gamma \vdash_\Sigma \alpha \longrightarrow_c = \alpha' : \mathrm{K}$

Les règles de typage peuvent être utilisées sur n'importe quelle position grâce à la règle (c / η), qui exprimer le fait que $\longrightarrow_c$ subsume $\longrightarrow_\eta$.

Le chapitre suivant est consacré à l'étude des propriété de ces relations $\longrightarrow_\eta$ et $\longrightarrow_c$, et nous y discuterons plus en détail des motivations pour ajouter ces relations au calcul et les présenter sous cette forme. Nous introduisons néanmoins les règles de ces relations dès maintenant car elles influent sur le système de typage, et donc sur les démonstrations de ce chapitre, via la règle (eq.).

Règles de subsomption

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \alpha}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \alpha}\ \text{(c / } \eta\text{)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \longrightarrow_c \alpha' : \mathrm{K}}{\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathrm{K}}\ \text{(c / } \eta\text{)}$$

Règles de congruence pour les types

$$\frac{}{\Gamma \vdash_\Sigma a \longrightarrow_c a : kind_\Sigma(a)}\ \text{(type const.)}$$

$$\frac{\vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta \longrightarrow_c \beta' : \mathrm{K}}{\Gamma, x : \alpha \vdash_\Sigma \beta \longrightarrow_c \beta' : \mathrm{K}}\ \text{(type weak.)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma \beta \longrightarrow_\eta \beta' : \mathrm{K} \qquad \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathbf{type}}{\Gamma \vdash_\Sigma \lambda x : \alpha.\beta \longrightarrow_c \lambda x : \alpha'.\beta' : (\alpha)\,\mathrm{K}}\ \text{(type abs.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \longrightarrow_c \alpha' : (\beta)\,\mathrm{K} \qquad \Gamma;\vdash_\Sigma t \longrightarrow_\eta t' : \beta}{\Gamma \vdash_\Sigma \alpha\, t \longrightarrow_c \alpha' t' : \mathrm{K}}\ \text{(type app.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta \longrightarrow_\eta \beta' : \mathbf{type}}{\Gamma \vdash_\Sigma \alpha \multimap \beta \longrightarrow_c \alpha' \multimap \beta' : \mathbf{type}}\ \text{(lin. fun.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathbf{type} \qquad \Gamma, x : \alpha \vdash_\Sigma \beta \longrightarrow_\eta \beta' : \mathbf{type}}{\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta \longrightarrow_c (\Pi x : \alpha')\,\beta' : \mathbf{type}} \text{ (dep. prod.)}$$

Dans la règle (type weak.), x ne doit pas déjà apparaître dans Γ.

Règles de congruence pour les termes

$$\frac{}{\,;\, \vdash_\Sigma c \longrightarrow_c c : type_\Sigma(c)} \text{ (const.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type}}{\Gamma\,;\, x : \alpha \vdash_\Sigma x \longrightarrow_c x : \alpha} \text{ (lin. var.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type}}{\Gamma, x : \alpha\,;\, \vdash_\Sigma x \longrightarrow_c x : \alpha} \text{ (var.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma\,;\, \Delta \vdash_\Sigma t \longrightarrow_c t' : \beta}{\Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t \longrightarrow_c t' : \beta} \text{ (weak}_c\text{.)}$$

$$\frac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma\,;\, \Delta \vdash_\Sigma t \longrightarrow_\eta t' : \beta}{\Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t \longrightarrow_\eta t' : \beta} \text{ (weak}_\eta\text{.)}$$

$$\frac{\Gamma\,;\, \Delta, x : \alpha \vdash_\Sigma t \longrightarrow_\eta t' : \beta \qquad \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathbf{type}}{\Gamma\,;\, \Delta \vdash_\Sigma \lambda^0 x : \alpha.t \longrightarrow_c \lambda^0 x : \alpha'.t' : \alpha \multimap \beta} \text{ (lin. abs.)}$$

$$\frac{\Gamma\,;\, \Delta_1 \vdash_\Sigma t \longrightarrow_c t' : \alpha \multimap \beta \qquad \Gamma\,;\, \Delta_2 \vdash_\Sigma u \longrightarrow_\eta u' : \alpha}{\Gamma\,;\, \Delta_1, \Delta_2 \vdash_\Sigma t\,u \longrightarrow_c t'\,u' : \beta} \text{ (lin. app.)}$$

$$\frac{\Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t \longrightarrow_\eta t' : \beta \qquad \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : \mathbf{type}}{\Gamma\,;\, \Delta \vdash_\Sigma \lambda x : \alpha.t \longrightarrow_c \lambda x : \alpha'.t' : (\Pi x : \alpha)\,\beta} \text{ (abs.)}$$

$$\frac{\Gamma\,;\, \Delta \vdash_\Sigma t \longrightarrow_c t' : (\Pi x : \alpha)\,\beta \qquad \Gamma\,;\, \vdash_\Sigma u \longrightarrow_\eta u' : \alpha}{\Gamma\,;\, \Delta \vdash_\Sigma t\,u \longrightarrow_c t'\,u' : \beta[x := u]} \text{ (app.)}$$

$$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \forall i \in \{1, \ldots, n\}, \Gamma\,;\, \Delta \vdash_\Sigma t_i \longrightarrow_\eta t'_i : \alpha_i}{\Gamma\,;\, \Delta \vdash_\Sigma [l_i = t_i]_{i=1}^n \longrightarrow_c [l_i = t'_i]_{i=1}^n : a} \text{ (rec.)}$$

$$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \Gamma\,;\, \Delta \vdash_\Sigma t \longrightarrow_c t' : a}{\Gamma\,;\, \Delta \vdash_\Sigma t.l_i \longrightarrow_c t'.l_i : \alpha_i} \text{ (sel.)}$$

$$\frac{binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \qquad \Gamma\,;\, \Delta_1 \vdash_\Sigma u \longrightarrow_c u' : a \qquad \forall i \in \{1, \cdots, n\}, \Gamma\,;\, \Delta_2, x_i : \alpha_i \vdash_\Sigma t_i \longrightarrow_\eta t'_i : \beta}{\Gamma\,;\, \Delta_1, \Delta_2 \vdash_\Sigma \{case\ u \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n \longrightarrow_c \{case\ u' \text{ of } (c_i\,x_i) \to t'_i\}_{i=1}^n : \beta} \text{ (}$$

$$\frac{binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\}}{;\ \vdash_\Sigma c_i \ \longrightarrow_c \ c_i : \alpha_i \multimap a} \text{ (inj.)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : \alpha \qquad \Gamma \vdash_\Sigma \alpha \equiv \beta : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta : \mathbf{type}}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : \beta} \text{ (eq}_c\text{.)}$$

$$\frac{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta t' : \alpha \qquad \Gamma \vdash_\Sigma \alpha \equiv \beta : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta : \mathbf{type}}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta t' : \beta} \text{ (eq}_\eta\text{.)}$$

Dans les règles (lin. var.) et (var.), x ne doit pas déjà apparaître dans Γ. Dans la règle (weak.), x ne doit pas déjà apparaître dans Γ ni Δ. Dans la règle (abs.), x ne doit pas apparaître libre dans Δ.

Règles d'η-expansion

$$\frac{\alpha \text{ n'est pas une abstraction} \qquad \Gamma \vdash_\Sigma \alpha \ \longrightarrow_c \ \alpha' : (\beta)\,\mathrm{K} \qquad \Gamma \vdash_\Sigma A(\beta)}{\Gamma \vdash_\Sigma \alpha \ \longrightarrow_\eta \lambda x : \beta.(\alpha' x) : (\beta)\,\mathrm{K}}$$

$$\frac{t \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : \alpha \multimap \beta \qquad \Gamma \vdash_\Sigma A(\alpha \multimap \beta)}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta \lambda^0 x : \alpha.(t' \downarrow \ x) : \alpha \multimap \beta}$$

$$\frac{t \text{ n'est pas une abstraction non-linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : (\Pi x : \alpha)\,\beta \qquad \Gamma \vdash_\Sigma A((\Pi x : \alpha)\,\beta)}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta \lambda x : \alpha.(t' \downarrow \ x) : (\Pi x : \alpha)\,\beta}$$

$$\frac{t \text{ n'est pas un enregistrement} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : a \qquad binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta [l_i = t' \downarrow l_i]_{i=1}^n : a}$$

$$\frac{t \text{ n'est ni une analyse de cas, ni de la forme } c_j\,u \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_c \ t' : a \qquad binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\}}{\Gamma\,;\,\Delta \vdash_\Sigma t \ \longrightarrow_\eta \{case\ t' \text{ of } (c_i\,x_i) \to (c_i\,x_i)\}_{i=1}^n : a}$$

Dans les trois premières règles d'expansion, x ne doit pas déjà apparaitre dans Γ ou Δ. Dans la cinquième règle, $x_1, \cdots, x_n$ ne doivent pas déjà apparaitrent dans Γ ou Δ.

2.2.6 Équivalence des types des abstractions

Suivant une approche similaire à celle de Ghani (1997), nous introduisons dans le calcul une relation d'équivalence entre termes ou entre types qui ne diffèrent que par les types étiquetant leurs abstractions. Ces étiquettes doivent également être $\beta\gamma\eta$ $\approx$-équivalentes. Afin de tester cette équivalence entre étiquettes, $\approx$ doit donc conserver comme information un contexte non-linéaire. L'information du contexte linéaire est en revanche inutile, puisque l'équivalence entre types ne l'utilise pas. De même, contrairement à $\longrightarrow_\eta$, il n'est pas nécessaire de garder trace du type sous lequel s'opère la relation, puisque les types des étiquettes ne peuvent être que de la sorte **type**. Ce système comprend donc deux jugements :

- Le terme t ne diffère du terme u que par des étiquettes $\beta\gamma\eta$ $\approx$-équivalentes sous le contexte Γ : $\Gamma \vdash_\Sigma t \approx u$
- Le type α ne diffère du type β que par des étiquettes $\beta\gamma\eta$ $\approx$-équivalentes sous le contexte Γ : $\Gamma \vdash_\Sigma \alpha \approx \beta$

Cette relation fait écho à la règle (eq.), qui permet d'appliquer au type assigné à un terme un nombre arbitraire de réécritures. En effet les règles d'η-expansion qui produisent des abstractions introduisent comme étiquette une partie du type assigné au terme expansé. Sans cette possibilité de modifier les étiquettes dans les mêmes conditions que les types assignés, la preuve du lemme 38 (commutativité entre $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$) ne fonctionnerait pas.

Congruences sur les types

$$\Gamma \vdash_\Sigma a \approx a$$

$$\frac{\Gamma \vdash_\Sigma \alpha \approx \alpha' \qquad \Gamma \vdash_\Sigma t \approx t'}{\Gamma \vdash_\Sigma \alpha\, t \approx \alpha'\, t'}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \approx \alpha' \qquad \Gamma \vdash_\Sigma \beta \approx \beta'}{\Gamma \vdash_\Sigma \alpha \multimap \beta \approx \alpha' \multimap \beta'}$$

$$\frac{\Gamma \vdash_\Sigma \alpha \approx \alpha' \qquad \Gamma, x : \alpha \vdash_\Sigma \beta \approx \beta'}{\Gamma \vdash_\Sigma (\Pi x : \alpha)\, \beta \approx (\Pi x : \alpha')\, \beta'}$$

$$\frac{\Gamma \vdash_\Sigma \beta \equiv \beta' \qquad \Gamma, x : \beta \vdash_\Sigma \alpha \approx \alpha'}{\Gamma \vdash_\Sigma \lambda x : \beta.\alpha \approx \lambda x : \beta'.\alpha'}$$

Congruences sur les termes

$$\frac{}{\Gamma \vdash_\Sigma c \approx c}$$

$$\frac{}{\Gamma \vdash_\Sigma x \approx x}$$

$$\frac{\Gamma \vdash_\Sigma t \approx t' \qquad \Gamma \vdash_\Sigma u \approx u'}{\Gamma \vdash_\Sigma t\, u \approx t'\, u'}$$

$$\frac{\Gamma \vdash_\Sigma t \approx t' \qquad \Gamma \vdash_\Sigma \alpha \equiv \alpha'}{\Gamma \vdash_\Sigma \lambda^0 x : \alpha.t \approx \lambda^0 x : \alpha'.t'}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t \approx t' \qquad \Gamma \vdash_\Sigma \alpha \equiv \alpha'}{\Gamma \vdash_\Sigma \lambda x : \alpha.t \approx \lambda x : \alpha'.t'}$$

$$\frac{\Gamma \vdash_\Sigma t_1 \approx t'_1 \ \cdots\ \Gamma\,;\, \Delta \vdash_\Sigma t_n \approx t'_n}{\Gamma \vdash_\Sigma [l_i = t_i]_{i=1}^n \approx [l_i = t'_i]_{i=1}^n}$$

$$\frac{\Gamma \vdash_\Sigma t \approx t'}{\Gamma \vdash_\Sigma t.l_i \approx t'.l_i}$$

$$\frac{\mathit{binding}_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \qquad \Gamma \vdash_\Sigma u \approx u' \qquad \forall i \in \{1, \cdots, n\}, \Gamma \vdash_\Sigma t_i \approx t'_i}{\Gamma \vdash_\Sigma \{\mathit{case}\ u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n \approx \{\mathit{case}\ u' \text{ of } (c_i\, x_i) \to t'_i\}_{i=1}^n}$$

$$\Gamma \vdash_\Sigma c_i \approx c_i$$

Le caractère transitif et réflexif de $\approx$ découle immédiatement des règles précédentes. En revanche, il n'est pas évident de prime abord que $\approx$ soit une relation symétrique, car dans les règles comparant des abstractions, le contexte non-linéaire est renseigné par l'étiquette du terme gauche. En réalité cette rupture de symétrie apparente n'empêche pas $\approx$ d'être une relation d'équivalence, puisque l'étiquette du terme de gauche et celle du terme de droite sont tenues d'être $\beta\gamma\eta$ $\approx$-équivalentes, et que nous établirons dans le lemme 24 (équivalence du contexte) qu'une telle différence dans le contexte est anodine.

2.3 Normalisation forte de $\longrightarrow_{\beta\gamma}$

Une relation de réécriture R est fortement normalisante s'il n'existe pas de séquence infinie $o_1 R o_2 R \ldots$. Cette propriété permet donc de s'assurer de la terminaison sans conditions d'un algorithme calculant une forme normale d'un objet en appliquant la relation R autant que possible.

La relation $\longrightarrow_{\beta\gamma}$ étant réflexive, elle est trivialement non fortement normalisante. Dans la suite, c'est donc à la normalisation forte de la partie non réflexive de $\longrightarrow_{\beta\gamma}$ que nous nous intéresserons, en montrant qu'il n'existe pas de séquence infinie $o_1 \longrightarrow_{\beta\gamma} o_2 \ldots$ telle que pour tout indice i entier, o_i soit différent de o_{i+1}. Nous qualifierons dans la suite les étapes de réduction $t \longrightarrow_{\beta\gamma} t'$ avec $t \neq t'$ de non triviales.

A l'instar de la relation $\longrightarrow_{\beta}$ dans le λ-calcul simple, qui correspond d'ailleurs à un fragment de $\longrightarrow_{\beta\gamma}$, cette propriété de normalisation forte n'est vérifiée qu'en se limitant aux termes bien typés et aux types bien formés.

Suivant la même stratégie que dans Cervesato et Pfenning (1996), où le le $\lambda\Pi$-calcul linéaire qui y est défini est envoyé vers le λ-calcul simplement typé, nous donnons une démonstration de cette propriété en montrant que chaque séquence de $\longrightarrow_{\beta\gamma}$ correspond à une séquence de réduction au moins aussi longue dans un calcul plus simple qui possède déjà une preuve de normalisation forte : nous utiliserons ici le λ-calcul avec produit et somme étudié par Lindley (2007).

La preuve suit donc le schéma suivant : dans une première partie nous présentons brièvement ce λ-calcul avec produit et somme $\lambda^{+,\times}$, nous donnons ensuite un ensemble de règles de traduction de notre calcul vers $\lambda^{+,\times}$, et enfin nous montrons que ces règles de traduction préservent les jugements de typage et les séquences de réduction, ce qui entraîne la normalisation forte de $\longrightarrow_{\beta\gamma}$.

2.3.1 Le λ-calcul avec produit et somme

Ce calcul correspond au λ-calcul simplement typé auquel on ajoute les constructions de somme disjointe ($+$) et de produit cartésien ($\times$). A la différence de notre calcul, ces constructions utilisent leur propre connecteur de type, plutôt que des types atomiques. De plus, ce calcul se distingue du nôtre par l'absence de produit dépendant, d'abstractions linéaires et de signatures.

Les définitions suivantes reprennent celles de Lindley (2007). Les définitions usuelles d'α-conversion, de substitution et de variables libres et liées ne sont pas redonnées. La traduction que nous utiliserons ne nécessite qu'un seul type de base, nous le noterons ω.

Syntaxe des types

$$A ::= \omega \mid A_1 \to A_2 \mid A_1 \times A_2 \mid A_1 + A_2$$

Syntaxe des termes

$$\begin{aligned} t ::= \quad & x \mid t_1\ t_2 \mid \pi_1(t) \mid \pi_2(t) \text{ (termes neutres purs)} \\ & \mid \lambda x.t \mid \langle t_1, t_2 \rangle \mid \iota_1(t) \mid \iota_2(t) \mid \delta(t_3, x_1.t_1, x_2.t_2) \end{aligned}$$

Règles de typage

$$\frac{}{\Gamma, x : A \vdash_L x : A}\ (\text{var}_L)$$

$$\frac{\Gamma, x : A \vdash_L t : B}{\Gamma \vdash_L \lambda x.t : A \to B}\ (\text{abs}_L)$$

$$\frac{\Gamma, y : \beta, x : \alpha, \Gamma' \vdash_L t : \gamma}{\Gamma, x : \alpha, y : \beta, \Gamma' \vdash_L t : \gamma}\ (\text{perm}_L)$$

$$\frac{\Gamma \vdash_L u : A \to B \qquad \Gamma \vdash_L v : A}{\Gamma \vdash_L u\ v : B}\ (\text{app}_L)$$

$$\frac{\Gamma \vdash_L u : A \qquad \Gamma \vdash_L v : B}{\Gamma \vdash_L \langle u, v\rangle : A \times B}\ (\text{rec}_L)$$

$$\frac{\Gamma \vdash_L t : A_1 \times A_2 \qquad i \in \{1,2\}}{\Gamma \vdash_L \pi_i(t) : A_i}\ (\text{sel}_L)$$

$$\frac{\Gamma \vdash_L t : A_i \qquad i \in \{1,2\}}{\Gamma \vdash_L \iota_i(t) : A_1 + A_2}\ (\text{inj}_L)$$

$$\frac{\Gamma \vdash_L m : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash_L t_1 : B \qquad \Gamma, x_2 : A_2 \vdash_L t_2 : B}{\Gamma \vdash_L \delta(m, x_1.t_1, x_2.t_2) : B}\ (\text{case}_L)$$

Dans la règle (abs$_L$.), x ne doit pas apparaître dans Γ. Dans la règle (case$_L$), x_1 et x_2 ne doivent pas apparaître dans Γ.

Règles de β-réduction

$$\begin{array}{ll}
\lambda x.t\ u & \longrightarrow_L t[x := u] \\
\pi_1(\langle t_1, t_2\rangle) & \longrightarrow_L t_1 \\
\pi_2(\langle t_1, t_2\rangle) & \longrightarrow_L t_2 \\
\delta(\iota_1(m), x_1.t_1, x_2.t_2) & \longrightarrow_L t_1[x_1 := m] \\
\delta(\iota_2(m), x_1.t_1, x_2.t_2) & \longrightarrow_L t_2[x_2 := m]
\end{array}$$

Conversions permutatives

Comme discuté dans la section 2.2.3, les conversions permutatives peuvent être classées en fonction de la position occupée par l'analyse de cas dans la permutation. Lindley (2007) distingue quatre familles de positions, qu'on peut définir sous la forme de *trames* :

Définition 18 (Trames)**.**

$$\begin{array}{lll}
F[\,] & ::= F_1[\,] \mid F_2[\,] \mid F_3[\,] \mid F_4[\,] & \textit{(Trames)} \\
F_1[\,] & ::= [\,]n \mid \pi_1([\,]) \mid \pi_2([\,]) \mid \delta([\,], x_1.t_1, x_2.t_2) & \textit{(Trames d'élimination)} \\
F_2[\,] & ::= m[\,] \mid \iota_1([\,]) \mid \iota_2([\,]) \mid \langle [\,], n\rangle \mid \langle m, [\,]\rangle & \textit{(Trames neutres)} \\
F_3[\,] & ::= \lambda x.[\,] & \textit{(Trames lambda)} \\
F_4[\,] & ::= \delta(m, x_1.[\,], x_2.t_2) \mid \delta(m, x_1.t_1, x_2.[\,]) & \textit{(Trames de continuation)}
\end{array}$$

$$\begin{array}{ll}
H[\,] & ::= F_1[\,] \\
D[\,] & ::= [\,] \mid \delta(m, x_1.D[\,], x_2.t_2) \mid \delta(m, x_1.t_1, x_2.D[\,]) \\
HD[\,] & ::= H[D[\,]]
\end{array}$$

La règle générale pour les conversions permutatives est alors :

$$HD[\delta(m, x_1.t_1, x_2.t_2)] \longrightarrow_L \delta(m, x_1.HD[t_1], x_2.HD[t_2])$$
$$x_1, x_2 \notin FV(HD), \text{ et } BV(HD) \cap FV(m) = \emptyset$$

L'ensemble de trames H étant restreint à F_1 dans cette définition, seules les conversions permutatives standards sont autorisées. Étendre H aux autres familles de trames permet de définir une relation de conversion permutative plus puissante, mais pour laquelle Lindley (2007) ne donne pas de preuve de normalisation forte.

La relation complète de β-réduction $\longrightarrow_L$ est, de façon usuelle, la plus petite relation vérifiant les règles précédentes et qui passe au contexte.

2.3.2 Règles de traduction

Le principe de cette traduction est d'effacer de notre calcul tous les éléments qui n'apparaissent pas dans $\lambda^{+,\times}$. Ceci comprend donc toutes les informations de linéarité, qui doivent être défaussées, les constantes et les types atomiques, qui sont traduits en variables, et le mécanisme du produit dépendant.

Trois niveaux de traduction sont distingués : τ permet de former des types dans $\lambda^{+,\times}$, $|\ |$ des termes et μ des contextes. Les types de $\lambda^{+,\times}$ sont traduits à partir des types et sortes de notre calcul, et à ce niveau de traduction l'information des termes dont les types peuvent dépendre doit être défaussée. Les termes de $\lambda^{+,\times}$ sont traduits à partir des termes, mais aussi des types de notre calcul. En effet, la relation $\longrightarrow_{\beta\gamma}$ est définie également sur les types, et pour prouver sa normalisation forte, il faut pouvoir traduire des séquences de réduction de types vers une séquence de réduction de termes dans $\lambda^{+,\times}$. Comme des réductions peuvent survenir au sein des termes dont dépendent les types, il ne faut à ce niveau de traduction surtout pas défausser cette information, ou des séquences de réduction dans ces types dépendants n'auraient aucune séquence correspondante dans $\lambda^{+,\times}$, ce qui ferait échouer le transfert de la normalisation forte. Pour la même raison, les types apparaissant dans les étiquettes des λ-abstractions ou du produit dépendant peuvent eux-même permettre des séquences de réductions, et doivent apparaitre dans le terme résultant de la traduction.

Les règles de traduction suivantes sont définies pour une signature Σ donnée.

Traduction des sortes et types en types

Ces règles de traductions se contentent de défausser toute l'information non pertinente dans $\lambda^{+,\times}$. $a_\emptyset$ désigne un type atomique a pour lequel $bindings(a)$ n'est pas défini.

$$
\begin{array}{ll}
\tau(\,\mathtt{type}\,) & ::= \omega \\
\tau((\alpha)\,\mathrm{K}\,) & ::= \tau(\alpha) \to \tau(\,\mathrm{K}\,) \\
 & \\
\tau(a_\emptyset) & ::= \omega \\
\tau(a) & ::= \tau(\alpha_1) \times \cdots \times \tau(\alpha_n) \times \omega \ \text{ si } binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \\
\tau(a) & ::= \tau(\alpha_1) + \cdots + \tau(\alpha_n) + \tau(\alpha_n) \ \text{ si } binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \\
\tau(\alpha \multimap \beta) & ::= \tau(\alpha) \to \tau(\beta) \\
\tau((\Pi x : \alpha)\,\beta) & ::= \tau(\alpha) \to \tau(\beta) \\
\tau(\lambda x : \beta.\alpha) & ::= \tau(\alpha) \\
\tau(\alpha\ t) & ::= \tau(\alpha)
\end{array}
$$

Traduction des types et termes en termes

Ces règles de traductions ont deux contraintes : produire des termes qui conservent toutes les potentialités de réduction du terme ou type de départ, et qui doivent être typable dans $\lambda^{+,\times}$. Pour ce faire nous introduisons des variables $x_{\multimap}$ et x_α, les règles de traduction des contextes s'assurant que ces variables apparaissent dans le contexte avec le type adéquat.

$$
\begin{array}{ll}
|a| & ::= x_a \\
|\alpha\ t| & ::= |\alpha|\ |t| \\
|\alpha \multimap \beta| & ::= x_{\multimap}\ |\alpha|\ |\beta| \\
|(\Pi x : \alpha)\ \beta| & ::= x_\alpha\ |\alpha|\ (\lambda x.|\beta|) \\
|\lambda x : \beta.\alpha| & ::= (\lambda y.(\lambda x.|\alpha|))|\beta|
\end{array}
$$

$$
\begin{array}{ll}
|x| & ::= x \\
|c| & ::= x_c \\
|\lambda x : \alpha.t| & ::= (\lambda y.(\lambda x.|t|))|\alpha| \\
|\lambda^0 x : \alpha.t| & ::= (\lambda y.(\lambda x.|t|))|\alpha| \\
|t_1\ t_2| & ::= |t_1|\ |t_2| \\
|[l_i = t_i]_{i=1}^n| & ::= \langle |t_1|, |[l_i = t_{i+1}]_{i=1}^{n-1}| \rangle \text{ si } n > 2 \\
|[l_1 = t_1]| & ::= \langle |t_1|, x_{stop} \rangle \\
|t.l_i| & ::= \pi_1(\pi_2^{i-1}(|t|)) \\
|\{case\ u \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n| & ::= \delta(|u|, x_1.|t_1|, x_2'.|\{case\ x_2' \text{ of } (c_i\ x_{i+1}) \to t_{i+1}\}_{i=1}^{n-1}|) \text{ si } n > 2 \\
|\{case\ u \text{ of } (c_1\ x_1) \to t_1\}| & ::= \delta(|u|, x_1.|t_1|, x_1.|t_1|) \\
|c_i| & ::= \lambda x.\iota_2^{i-1}(\iota_1(x))
\end{array}
$$

Traduction des contextes, signatures et types en contextes

Ces règles de traduction permettent de former le contexte d'un jugement de typage dans $\lambda^{+,\times}$. Il faut traduire le contexte de typage de départ, mais aussi la signature, car les constantes et les types atomiques ont été traduits en variables, et les types α apparaissant dans les étiquettes du produit dépendant, car ils ont introduit dans les termes des variables x_α.

$$
\begin{array}{ll}
\mu(\Delta, x : \alpha) & ::= \mu(\Delta), x : \tau(\alpha) \\
\mu(\Gamma, x : \alpha) & ::= \mu(\Gamma), x : \tau(\alpha)
\end{array}
$$

$$
\begin{array}{ll}
\mu(\Sigma; c : \alpha) & ::= \mu(\Sigma), x_c : \tau(\alpha) \\
\mu(\Sigma; a : \mathrm{K}) & ::= \mu(\Sigma), x_a : \tau(\mathrm{K}) \\
\mu(\Sigma; a = \mathrm{R} : \texttt{type}) & ::= \mu(\Sigma) \\
\mu(\Sigma; a = \mathrm{V} : \texttt{type}) & ::= \mu(\Sigma) \\
\mu(.) & ::= .
\end{array}
$$

$$
\begin{array}{ll}
\mu(a) & ::= . \\
\mu(\lambda x : \beta.\alpha) & ::= \mu(\alpha), \mu(\beta) \\
\mu(\alpha\ t) & ::= \mu(\alpha) \\
\mu(\alpha \multimap \beta) & ::= \mu(\alpha), \mu(\beta) \\
\mu((\Pi x : \alpha)\ \beta) & ::= x_\alpha : \omega \to (\tau(\alpha) \to \omega) \to \omega, \mu(\alpha), \mu(\beta) \text{ si } x_\alpha \notin \text{dom}(\mu(\alpha), \mu(\beta)) \\
 & \quad \mu(\alpha), \mu(\beta) \text{ sinon}
\end{array}
$$

Enfin, les variables $x_{\multimap}$ et x_{stop} doivent apparaitre systématiquement dans le contexte traduit :

Définition 19 (Contexte initial)**.** $\Gamma_0 ::= x_{\multimap} : \omega \to \omega \to \omega, x_{stop} : \omega$

2.3.3 Propriétés de la traduction

Dans cette partie nous démontrons les propriétés de conservation pour lesquelles les règles de traduction ont été conçues, et nous montrons qu'elles impliquent la normalisation forte de $\longrightarrow_{\beta\gamma}$.

Nous commençons d'abord par redonner la propriété d'affaiblissement pour $\lambda^{+,\times}$, indispensable pour préserver les règles (weak.) dans les jugements de typage.

Lemme 1 (Affaiblissement pour $\vdash_L$)**.**
Si $\Pi : \Gamma \vdash_L t : A$ *et* $x \notin dom(\Gamma)$
Alors $\Gamma, x : B \vdash_L t : A$

Démonstration. Immédiat par récurrence sur la structure de Π.

□

Le lemme suivant établit que τ efface toute l'information provenant des termes dont les types peuvent dépendre.

Lemme 2 (Effacement dans les types)**.**

1. $\tau(\alpha[x := t]) = \tau(\alpha)$.
2. *Si* $\Gamma \vdash_\Sigma \alpha \equiv \beta$
   Alors $\tau(\alpha) = \tau(\beta)$.

Démonstration. Immédiat en observant que τ efface toute occurrence d'un terme ou d'une abstraction dans un type.

□

Le lemme suivant établit le fait que tous les types attribués aux variables du domaine d'un contexte sont de sorte **`type`**. De plus, ce jugement de typage est d'une taille inférieur au jugement de départ, ce qui nous permettra dans le lemme suivant d'en obtenir une hypothèse de récurrence.

Lemme 3 (Formation correcte des types dans les contextes)**.**

1. *Si* $\Pi : \Gamma, x : \alpha \vdash_\Sigma \beta : K$
   alors $\Pi' : . \vdash_\Sigma \alpha :$ **`type`**.
   et Π' *est de hauteur inférieure à* Π
2. *Si* $\Pi : \Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t : \beta$
   alors $\Pi' : \Gamma \vdash_\Sigma \alpha :$ **`type`**.
   et Π' *est de hauteur inférieure à* Π
3. *Si* $\Pi : \Gamma\,;\, \Delta, x : \alpha \vdash_\Sigma t : \beta$
   alors $\Pi' : \Gamma \vdash_\Sigma \alpha :$ **`type`**.
   et Π' *est de hauteur inférieure à* Π

Démonstration. Immédiat par récurrence sur la hauteur de Π

□

Nous pouvons à présent donner les deux propriétés de préservation nécessaire pour notre résultat. La première est la préservation des jugements de typage : pour tout jugement de typage du calcul, sa traduction est un jugement de typage valide dans $\lambda^{+,\times}$.

Lemme 4 (Préservation du typage)**.**

1. *Si* $\Pi : \Gamma\,;\, \Delta \vdash_\Sigma t : \alpha$
   Alors $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\Delta) \vdash_L |t| : \tau(\alpha)$
2. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha : K$
   Alors $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha) \vdash_L |\alpha| : \tau(K)$

Démonstration. Par récurrence sur la hauteur de Π, selon la dernière règle utilisée :

– $\dfrac{}{\vdash_\Sigma a : kind_\Sigma(a)}$ (type const.)

Σ contient donc une déclaration $a : \mathrm{K}$, avec $kind_\Sigma(a) = \mathrm{K}$ et donc $x_a : \tau(\mathrm{K})$ apparaît dans $\mu(\Sigma)$. On peut donc bien utiliser (var$_L$.) pour obtenir :
$\Gamma_0, \mu(\Sigma) \vdash_L x_a : \tau(\mathrm{K})$

– (type weak) : immédiat en utilisant le lemme 1 (affaiblissement pour $\vdash_L$).

– $\dfrac{\Gamma, x : \alpha \vdash_\Sigma \beta : \mathrm{K}}{\Gamma \vdash_\Sigma \lambda x : \alpha.\beta : (\alpha)\,\mathrm{K}}$ (type abs.)

Par hypothèse de récurrence, on a :
$\Gamma_0, \mu(\Sigma), \mu(\Gamma, x : \alpha), \mu(\beta) \vdash_L |\beta| : \tau(\mathrm{K})$,
et donc par définition de μ :
$\Gamma_0, \mu(\Sigma), \mu(\Gamma), x : \tau(\alpha), \mu(\beta) \vdash_L |\beta| : \tau(\mathrm{K})$.

Par la règle (abs_L) on obtient alors :
$\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\beta) \vdash_L \lambda x.|\beta| : \tau(\alpha) \to \tau(\mathrm{K})$.

Par le lemme 3 (formation correcte des types dans les contextes), on a :
$. \vdash_\Sigma \alpha : \mathbf{type}$,
et donc par hypothèse de récurrence :
$\Gamma_0, \mu(\Sigma), \mu(\alpha) \vdash_L |\alpha| : \omega$.

Le jugement à prouver est $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\lambda x : \alpha.\beta) \vdash_L |\lambda x : \alpha.\beta| : \tau((\alpha)\,\mathrm{K})$,
c'est à dire : $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha), \mu(\beta) \vdash_L (\lambda y.(\lambda x.|\beta|))\ |\alpha| : \tau(\alpha) \to \tau(\mathrm{K})$.

On l'obtient par le lemme 1 (affaiblissement pour $\vdash_L$) et l'arbre de preuve suivant, en notant $\mu = \Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha), \mu(\beta)$:

$$\dfrac{\dfrac{\mu, y : \omega \vdash_L \lambda x.|\beta| : \tau(\alpha) \to \tau(\mathrm{K})}{\mu \vdash_L \lambda y.(\lambda x.|\beta|) : \omega \to \tau(\alpha) \to \tau(\mathrm{K})}\ (\mathrm{abs}_L) \qquad \mu \vdash_L |\alpha| : \omega}{\mu \vdash_L (\lambda y.(\lambda x.|\beta|))\ |\alpha| : \tau(\alpha) \to \tau(\mathrm{K})}\ (\mathrm{app}_L)$$

– $\dfrac{\Gamma \vdash_\Sigma \alpha : (\beta)\,\mathrm{K} \qquad \Gamma;\vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \alpha\ t : \mathrm{K}}$ (type app.)

Par hypothèse de récurrence, on a :
(1) $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha) \vdash_L |\alpha| : \tau(\beta) \to \tau(\mathrm{K})$ et
(2) $\Gamma_0, \mu(\Sigma), \mu(\Gamma) \vdash_L |t| : \tau(\beta)$.

On utilise le lemme 1 (affaiblissement de $\vdash_L$) sur (2) pour obtenir
(3) $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha) \vdash_L |t| : \tau(\beta)$
et la règle (app_L) sur (1) et (3) pour obtenir le résultat.

– $\dfrac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta : \mathbf{type}}{\Gamma \vdash_\Sigma \alpha \multimap \beta : \mathbf{type}}$ (lin. fun.)

On peut construire la preuve suivante, avec $\mu = \Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha), \mu(\beta)$:

$$\dfrac{\dfrac{\dfrac{\mathrm{var}_L}{\mu \vdash_L x_{\multimap} : \omega \to \omega \to \omega}\ (\mathrm{perm}_L^*) \qquad \dfrac{\text{H.R. + affaiblissement}}{\mu \vdash_L |\alpha| : \omega}}{\mu \vdash_L x_{\multimap}|\alpha| : \omega \to \omega}\ (\mathrm{app}_L) \qquad \dfrac{\text{H.R. + affaiblissement}}{\mu \vdash_L |\beta| : \omega}}{\mu \vdash_L x_{\multimap}|\alpha||\beta| : \omega}\ (\mathrm{app}_L)$$

– $\dfrac{\Gamma \vdash_\Sigma \alpha : \mathbf{type} \qquad \Gamma, x : \alpha \vdash_\Sigma \beta : \mathbf{type}}{\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta : \mathbf{type}}$ (dep. prod.)

Similairement au cas précédent, avec $\mu = \Gamma_0, \mu(\Sigma), \mu(\Gamma), x_\alpha : \omega \to (\tau(\alpha) \to \omega) \to \omega, \mu(\alpha), \mu(\beta)$, on construit la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{var}_L}{\mu \vdash_L x_\alpha : \omega \to (\tau(\alpha) \to \omega) \to \omega}\ (\mathrm{perm}_L^*) \qquad \dfrac{\text{H.R. + affaiblissement}}{\mu \vdash_L |\alpha| : \omega}}{\mu \vdash_L x_\alpha|\alpha| : (\tau(\alpha) \to \omega) \to \omega}\ (\mathrm{app}_L) \qquad \dfrac{\text{H.R. + affaiblissement}}{\mu, x : \tau(\alpha) \vdash_L |\beta| : \omega}}{\mu \vdash_L \lambda x.|\beta| : \tau(\alpha) \to \omega}\ (\mathrm{abs}_L)}{\mu \vdash_L x_\alpha|\alpha|(\lambda x.|\beta|) : \omega}\ (\mathrm{app}_L)$$

– (const) : similaire à (type const.)

– (lin. var.) , (var.), (weak.) : immédiat.

– (lin. abs.), (abs.) : similaire à (type abs.)

– (lin. app.) : similaire à (type app.)

– (app.) : similaire à (type app.), en utilisant le lemme 2 (effacement).

– $$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \Gamma\,;\,\Delta \vdash_\Sigma t_1 : \alpha_1 \quad \cdots \quad \Gamma\,;\,\Delta \vdash_\Sigma t_n : \alpha_n}{\Gamma\,;\,\Delta \vdash_\Sigma [l_i = t_i]_{i=1}^n : a}\ (\text{rec.})$$

Avec $\mu = \Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\Delta)$, on peut construire la preuve Π_n :

$$\frac{\dfrac{\text{H.R.}}{\mu \vdash_L |t_n| : \tau(\alpha_n)} \qquad \dfrac{\text{var}_L}{\mu \vdash_L x_{stop} : \omega}\ (\text{perm}_L^*)}{\mu \vdash_L \langle |t_n|, x_{stop}\rangle : \tau(\alpha_n) \times \omega}\ (\text{rec}_L)$$

Et, $\forall j \in 1, \cdots, n-1$, la preuve Π_j :

$$\frac{\dfrac{\text{H.R.}}{\mu \vdash_L |t_j| : \tau(\alpha_j)} \qquad \Pi_{j+1}}{\mu \vdash_L \langle |t_j|, |[l_i = t_{i+j}]_{i=1}^{n-j}|\rangle : \tau(\alpha_j) \times \cdots \times \tau(\alpha_n) \times \omega}\ (\text{rec}_L)$$

On a bien le résultat souhaité avec $j = 1$.

– $$\frac{binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n] \qquad \Gamma\,;\,\Delta \vdash_\Sigma t : a}{\Gamma\,;\,\Delta \vdash_\Sigma t.l_i : \alpha_i}\ (\text{sel.})$$

On applique (sel_L) i fois sur l'hypothèse de récurrence pour obtenir le résultat.

– $$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma u : a \qquad binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\} \qquad \forall i \in \{1, \ldots, n\}, \Gamma\,;\,\Delta_2, x_i : \alpha_i \vdash_\Sigma t_i : \beta}{\Gamma\,;\,\Delta_1 \cup \Delta_2 \vdash_\Sigma \{case\ u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n : \beta}\ (\text{case})$$

On note $\mu = \Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\Delta_1, \Delta_2)$,
et $\mu_j = \mu, x'_j : \tau(\alpha_j) + \cdots + \tau(\alpha_n) + \tau(\alpha_n)$.
On note Π^* pour une preuve obtenue à partir de Π en utilisant la règle (perm_L.) et le lemme 1 (affaiblissement de $\vdash_L$).
Si $n > 2$, on peut construire la preuve Π_n :

$$\frac{\dfrac{}{\mu_n \vdash_L x'_n : \tau(\alpha_n) + \tau(\alpha_n)}\ (\text{var}_L) \qquad \dfrac{\text{H.R.*}}{\mu_n, x_n : \tau(\alpha_n) \vdash_L |t_n| : \tau(\beta)}}{\mu_n \vdash_L \delta(x'_n, x_n.|t_n|, x_n.|t_n|) : \tau(\beta)}\ (\text{case}_L)$$

$\forall j \in 2, \cdots, n-1$, la preuve Π_j :

$$\frac{\dfrac{}{\mu_j \vdash_L x'_j : \tau(\alpha_j) + \cdots + \tau(\alpha_n) + \tau(\alpha_n)}\ (\text{var}_L) \qquad \dfrac{\text{H.R.*}}{\mu_j, x_j : \tau(\alpha_j) \vdash_L |t_j| : \tau(\beta)} \qquad \Pi_{j+1}^*}{\mu_j \vdash_L \delta(x'_j, x_j.|t_j|, x'_{j+1}.|\{case\ x'_{j+1} \text{ of } (c_i\, x_{i+j}) \to t_{i+j}\}_{i=1}^{n-j}|) : \tau(\beta)}\ (\text{case}_L)$$

et Π_1 :

$$\frac{\dfrac{\text{H.R.*}}{\mu \vdash_L |u| : \tau(\alpha_1) + \cdots + \tau(\alpha_n) + \tau(\alpha_n)} \qquad \dfrac{\text{H.R.*}}{\mu, x_1 : \tau(\alpha_1) \vdash_L |t_1| : \tau(\beta)} \qquad \Pi_2^*}{\mu \vdash_L \delta(|u|, x_1.|t_1|, x'_2.|\{case\ x'_2 \text{ of } (c_i\, x_{i+1}) \to t_{i+1}\}_{i=1}^{n-1}|) : \tau(\beta)}\ (\text{case}_L)$$

La conclusion de cette preuve Π_1 est bien égale au résultat à démontrer.
La preuve s'adapte directement aux cas $n = 1$ et $n = 2$

– $\dfrac{binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \ldots | c_n \text{ of } \alpha_n\}}{\Gamma\,;\, \vdash_\Sigma c_i : \alpha_i \multimap a}$ (inj.)

Avec $\mu = \Gamma_0, \mu(\Sigma)$, on peut construire la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mu, x : \tau(\alpha_i) \vdash_L x : \tau(\alpha_i)}\ (\text{var}_L)}{\mu, x : \tau(\alpha_i) \vdash_L \iota_1(x) : \tau(\alpha_i) + \cdots + \tau(\alpha_n) + \tau(\alpha_n)}\ (\text{inj}_L)}{\mu, x : \tau(\alpha_i) \vdash_L \iota_2^{i-1}(\iota_1(x)) : \tau(\alpha_1) + \cdots + \tau(\alpha_n) + \tau(\alpha_n)}\ (\text{inj}_L^{i-1})}{\mu \vdash_L \lambda x.\iota_2^{i-1}(\iota_1(x)) : \tau(\alpha_i) \to (\tau(\alpha_1) + \cdots + \tau(\alpha_n) + \tau(\alpha_n))}\ (\text{abs}_L)$$

– (eq.) : immédiat en utilisant le lemme 2 (effacement).

□

La seconde propriété de préservation concerne les séquences de réductions : chaque étape de $\longrightarrow_{\beta\gamma}$ non triviale doit correspondre à une séquence non nulle de $\longrightarrow_L$.

Lemme 5 (Préservation des séquences de réduction)**.**

1. *Si* $\Pi : t \longrightarrow_{\beta\gamma} t'$ *et* $t \neq t'$
   Alors $|t| \longrightarrow_L^+ |t'|$
2. *Si* $\Pi : \alpha \longrightarrow_{\beta\gamma} \alpha'$ *et* $\alpha \neq \alpha'$
   Alors $|\alpha| \longrightarrow_L^+ |\alpha'|$

Démonstration. Par récurrence sur la structure de Π, selon la dernière règle utilisée.

– Règles de congruence : immédiat.

– $\dfrac{t \longrightarrow_\beta t' \qquad u \longrightarrow_\beta u'}{(\lambda^0 x : \alpha.t)\, u \longrightarrow_\beta t'[x := u']}$

Par hypothèse de récurrence, si $t \neq t'$, $|t| \longrightarrow_L^+ |t'|$,
et si $u \neq u'$, $|u| \longrightarrow_L^+ |u'|$.
On a donc $(\lambda x.|t|)\,|u| \longrightarrow_L^+ (\lambda x.|t'|)\,|u'|$ ou $(\lambda x.|t|)\,|u| = (\lambda x.|t'|)\,|u'|$,
et $(\lambda x.|t'|)\,|u'| \longrightarrow_L |t'|[x := |u'|]$,
d'où $(\lambda x.|t|)\,|u| \longrightarrow_L^+ |t'|[x := |u'|]$.

La preuve est identique pour la réduction d'un λ-rédex non linéaire.

– $\dfrac{t_j \longrightarrow_\beta t'_j}{[l_i = t_i]_{i=1}^n.l_j \longrightarrow_\beta t'_j}$

De la même façon, par hypothèse de récurrence $|t_j| \longrightarrow_L^* |t'_j|$, donc
$\langle |t_j|, |[l_i = t_{i+j}]_{i=1}^{n-j}| \rangle \longrightarrow_L^* \langle |t'_j|, |[l_i = t_{i+j}]_{i=1}^{n-j}| \rangle$.
D'où
$|[l_i = t_i]_{i=1}^n| \longrightarrow_L^* |[l_1 = t_1; \ldots; l_j = t'_j; \ldots; l_n = t_n]|$.

On a donc $[l_i = t_i]_{i=1}^n.l_j \longrightarrow_L^* \pi_1(\pi_2^{i-1}(|[l_1 = t_1; \ldots; l_j = t'_j; \ldots; l_n = t_n]|)) \longrightarrow_L^i |t'_j|$

La preuve est similaire pour la réduction d'un case.

– $\dfrac{}{\{case\ u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n\, v \longrightarrow_\gamma \{case\ u \text{ of } (c_i\, x_i) \to t_i\ v\}_{i=1}^n}$

En considérant le contexte $[\,]\ |v|$, comme on a bien $\forall i \in \{1,\cdots,n\}, x_i \notin \mathrm{FV}(v')$, on a par conversion permutative :
$\delta(|u|, x_1.|t_1|, x_2.\delta(x_2, x_2.|t_2|, \delta(\cdots\delta(x_n, x_n.|t_n|, x_n.|t_n|))\cdots)\ |v|$
$\longrightarrow_L \delta(|u|, x_1.|t_1|\ |v|, x_2.\delta(x_2, x_2.|t_2|, \delta(\cdots\delta(x_n, x_n.|t_n|, x_n.|t_n|))\cdots)\ |v|)$.

On peut appliquer à nouveau n autres conversions permutatives pour obtenir :
$\delta(|u|, x_1.|t_1|\ |v|, x_2.\delta(x_2, x_2.|t_2|\ |v|, \delta(\cdots\delta(x_n, x_n.|t_n|\ |v|, x_n.|t_n|\ |v|)))\cdots)$.
qui est bien égal à $|\{case\ u \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n|$.

La preuve pour les autres conversions permutatives suivantes est similaire.

□

Ces propriétés de préservation permettent de transférer le résultat de normalisation forte sur les termes bien typés acquis pour $\lambda^{+,\times}$, que nous redonnons à présent.

Lemme 6 (Normalisation forte de $\longrightarrow_L$ (Lindley, 2007))**.**

Si on a $\Gamma \vdash_L t : A$
Alors il n'existe pas de séquence infinie de $\longrightarrow_L$ *partant de* t.

Nous avons alors tous les éléments suffisants pour énoncer et démontrer le théorème de normalisation forte de $\longrightarrow_{\beta\gamma}$.

Théorème 2 (Normalisation forte de $\longrightarrow_{\beta\gamma}$)**.**

Si on a $\Gamma\ ;\ \Delta \vdash_\Sigma t : \alpha$ *(resp.* $\Gamma \vdash_\Sigma \alpha : K$*),*
alors il n'existe pas de séquence partant de t *(resp. d'*α*) et comprenant un nombre infini de* $\longrightarrow_{\beta\gamma}$ *non triviales.*

Démonstration. Par le lemme 4 (préservation du typage), on a
$\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\Delta) \vdash_L |t| : \tau(\alpha)$ (resp. $\Gamma_0, \mu(\Sigma), \mu(\Gamma), \mu(\alpha) \vdash_L |\alpha| : \tau(\mathrm{K})$)
Pour toute séquence de $\longrightarrow_{\beta\gamma}$ non triviales partant de t (resp. d'α), par le lemme 5 (préservation des séquences de réduction), il existe une séquence d'au moins autant de $\longrightarrow_L$ à partir de $|t|$ (resp. $|\alpha|$). Par le lemme 6 (normalisation forte de $\longrightarrow_L$), on sait que cette dernière séquence ne peut être infinie. La première étant plus courte, elle ne peut l'être non plus.

□

Nous avons donc établi que tout terme et tout type peut être normalisé en lui appliquant suffisamment d'étapes non triviales de $\longrightarrow_{\beta\gamma}$. Ce résultat n'est cependant pas suffisant pour garantir que la forme normale obtenue ne dépend pas de la séquence de réduction appliquée, et est propre à chaque terme et type. Cette unicité des formes $\beta\gamma$-normales nécessite la propriété de confluence, qui fait l'objet de la partie suivante.

La traduction utilisée dans cette partie ne permet pas d'obtenir la normalisation forte de $\longrightarrow_{\beta\gamma\eta}$, car l'η-expansion de notre calcul n'est pas définie d'une manière compatible avec l'η-expansion définie dans le calcul de Lindley (2007), pour lequel les analyses de cas ne peuvent pas être expansées. Cette différence est due à la présence des conversions permutatives non standards, puisque le résultat de l'η-expansion d'une analyse de cas serait de toute façon permutée vers ses branches. A l'inverse, sans conversions permutatives non standards, l'η-expansion doit être autorisée sur les analyses de cas sous peine de briser la confluence du calcul. Ainsi par exemple en prenant le terme $t = (\lambda^0 x : \alpha \multimap \beta.x)\{case\ u \text{ of } (c_i\ x_i) \to f_i\}_{i=1}^n$, de type $\alpha \multimap \beta$ (les f_i étant des constantes de type $\alpha \multimap \beta$), on a :

$t \longrightarrow_\beta \{case\ u \text{ of } (c_i\ x_i) \to f_i\}_{i=1}^n$
et $.\ ;\ . \vdash_\Sigma t \longrightarrow_\eta (\lambda^0 x : \alpha \multimap \beta.\lambda^0 y : \alpha'.x\ y)\{case\ u \text{ of } (c_i\ x_i) \to f_i\}_{i=1}^n$,

et il n'existe aucun moyen de faire converger ces deux termes obtenus à partir de t sans utiliser d'η-expansion des analyses de cas ou de conversion permutative non standard.

La normalisation de $\longrightarrow_{\beta\gamma\eta}$ sera discutée en détail au chapitre 3.

2.4 Confluence de $\longrightarrow_{\beta\gamma}$

La propriété de confluence pour une relation R énonce que tout couple d'objets qui ont un antécédent commun par R^* ont également une image commune par R^*. Cela signifie qu'un objet donné a au plus une forme normale accessible, qu'une étape de réduction donnée ne peut jamais exclure d'atteindre cette forme normale, et donc que le choix entre différentes façons de réduire un objet donné n'est pas crucial. Si la relation R est également fortement normalisante, on est alors assuré que cette forme normale existe, et qu'elle est atteinte en appliquant n'importe quelle séquence de réduction pourvue qu'elle soit assez longue. On peut alors tester la R-équivalence de deux objets en calculant leur forme normale respective et en vérifiant si elles sont égales.

Nous procédons à la démonstration de la confluence de $\longrightarrow_{\beta\gamma}$ de la façon suivante : nous montrons dans les deux prochaines sections que $\longrightarrow_\beta$ et $\longrightarrow_\gamma$ sont chacune des relations confluentes, puis nous montrons dans une troisième section que ces relations possèdent une propriété de commutation qui assure la confluence de leur union.

2.4.1 Confluence de $\longrightarrow_\beta$

La confluence de $\longrightarrow_\beta$ ne pose pas de difficulté particulière par rapport à un calcul plus simple comme $\lambda^{+,\times}$. La présentation sous forme de calcul parallèle permet d'obtenir un résultat plus fort : la propriété du diamant. La normalisation forte de $\longrightarrow_\beta$ n'est donc pas utilisée dans cette preuve de confluence.

De façon usuelle, nous commençons par établir le fait que $\longrightarrow_\beta$ commute avec l'opération de substitution :

Lemme 7 (Substitution pour $\longrightarrow_\beta$)**.**

1. *Si* $t \longrightarrow_\beta t'$ *et* $\Pi : u \longrightarrow_\beta u'$*, alors* $u[x := t] \longrightarrow_\beta u'[x := t']$.
2. *Si* $t \longrightarrow_\beta t'$ *et* $\Pi : \alpha \longrightarrow_\beta \alpha'$*, alors* $\alpha[x := t] \longrightarrow_\beta \alpha'[x := t']$.

Démonstration. par récurrence sur la structure de Π, et selon la dernière règle utilisée dans Π :

- Règles de congruences : immédiat.
- $$\frac{u_1 \longrightarrow_\beta u_1' \qquad u_2 \longrightarrow_\beta u_2'}{(\lambda^0 y.u_1)\, u_2 \longrightarrow_\beta u_1'[y := u_2']}$$

  par hypothèse de récurrence, on a
  $u_1[x := t] \longrightarrow_\beta u_1'[x := t']$
  et
  $u_2[x := t] \longrightarrow_\beta u_2'[x := t']$.

  Par β-réduction, on a donc
  $(\lambda^0 y.u_1[x := t])\,(u_2[x := t]) \longrightarrow_\beta u_1'[x := t'][y := u_2'[x := t']]$
  c'est-à-dire $((\lambda^0 y.u_1)\, u_2)[x := t] \longrightarrow_\beta u_1'[y := u_2'][x := t']$

- Les autres règles de β-réduction donnent une preuve similaire.

□

La propriété du diamant pour une relation R énonce le fait que tout couple d'objets ayant un antécédent commun par R a également une image commune par R. Cette propriété correspond donc à une propriété de confluence dans laquelle on ne considère que des réductions en une étape plutôt que des séquences de réductions.

Lemme 8 (Propriété du diamant pour $\longrightarrow_\beta$)**.**

1. *Si* $\Pi_1 : u \longrightarrow_\beta u_1$ *et* $\Pi_2 : u \longrightarrow_\beta u_2$*,*
   alors $\exists u'$ *tel que* $u_1 \longrightarrow_\beta u'$ *et* $u_2 \longrightarrow_\beta u'$
2. *Si* $\Pi_1 : \alpha \longrightarrow_\beta \alpha_1$ *et* $\Pi_2 : \alpha \longrightarrow_\beta \alpha_2$*,*
   alors $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_\beta \alpha'$ *et* $\alpha_2 \longrightarrow_\beta u'$

Démonstration. Par récurrence sur la taille de u ou de α.

Si Π_1 et Π_2 terminent par des règles de congruence, la récurrence est immédiate. On suppose donc à présent que Π_1 ne termine pas par une règle de congruence (Π_1 et Π_2 jouent un rôle symétrique dans la preuve).

– Si Π_1 termine par une réduction de λ-rédex linéaire :

$$\frac{t \longrightarrow_\beta t_1 \qquad v \longrightarrow_\beta v_1}{u = (\lambda^0 x : \alpha.t)\, v \longrightarrow_\beta t_1[x := v_1]}$$

1. Si Π_2 termine également par une réduction de λ-rédex :

$$\frac{t \longrightarrow_\beta t_2 \qquad v \longrightarrow_\beta v_2}{(\lambda^0 x : \alpha.t)\, v \longrightarrow_\beta t_2[x := v_2]}$$

   En appliquant l'hypothèse de récurrence sur t et v, on sait qu' $\exists t', v'$ tels que $t_1 \longrightarrow_\beta t', t_2 \longrightarrow_\beta t', v_1 \longrightarrow_\beta v'$, et $v_2 \longrightarrow_\beta v'$.

   On obtient donc par le lemme 7 (substitution) :
   $t_1[x := v_1] \longrightarrow_\beta t'[x := v']$
   et $t_2[x := v_2] \longrightarrow_\beta t'[x := v']$.

2. Si Π_2 termine par une règle de congruence :

$$\frac{\dfrac{t \longrightarrow_\beta t_2 \qquad \alpha \longrightarrow_\beta \alpha'}{\lambda^0 x : \alpha.t \longrightarrow_\beta \lambda^0 x : \alpha'.t_2} \qquad v \longrightarrow_\beta v_2}{(\lambda^0 x : \alpha.t)\, v \longrightarrow_\beta (\lambda^0 x : \alpha'.t_2)\, v_2}$$

   De la même manière $\exists t', v'$ tel que $t_1 \longrightarrow_\beta t', t_2 \longrightarrow_\beta t', v_1 \longrightarrow_\beta v'$, et $v_2 \longrightarrow_\beta v'$.

   Par le lemme 7 (substitution), $t_1[x := v_1] \longrightarrow_\beta t'[x := v']$.

   Par une règle de β-reduction $(\lambda^0 x : \alpha'.t_2)\, v_2 \longrightarrow_\beta t'[x := v']$

– Si Π_1 termine par une réduction de λ-rédex non linéaire, la preuve est similaire à celle du cas précédent.

– Si Π_1 termine par la réduction d'une projection, que Π_2 soit une congruence ou une réduction, la récurrence est immédiate.

– Si Π_1 termine par la réduction du rédex d'un cas, la preuve est également immédiate.

□

La propriété du diamant est plus forte que la confluence. Nous en profitons pour énoncer une version plus fine de la confluence, liant la longueur des séquences de réduction convergentes à celle des séquences divergentes.

Lemme 9 (Confluence de $\longrightarrow_\beta$)**.**

1. *Si* $\Pi_1 : u \longrightarrow_\beta^n u_1$ *et* $\Pi_2 : u \longrightarrow_\beta^m u_2$*, alors* $\exists u'$ *tel que* $u_1 \longrightarrow_\beta^m u'$ *et* $u_2 \longrightarrow_\beta^n u'$
2. *Si* $\Pi_1 : \alpha \longrightarrow_\beta^n \alpha_1$ *et* $\Pi_2 : \alpha \longrightarrow_\beta^m \alpha_2$*, alors* $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_\beta^m \alpha'$ *et* $\alpha_2 \longrightarrow_\beta^n \alpha'$

Démonstration. Immédiat par récurrence sur $n + m$, en utilisant le lemme 8 (propriété du diamant pour $\longrightarrow_\beta$).
□

2.4.2 Confluence de $\longrightarrow_\gamma$

$\longrightarrow_\gamma$ ne vérifie pas la propriété du diamant, et à défaut nous établissons sa confluence faible : pour tout couple d'étapes divergentes de $\longrightarrow_\gamma$, il existe des séquences de $\longrightarrow_\gamma$ convergentes.

Lemme 10 (Confluence faible pour $\longrightarrow_\gamma$)**.**

1. *Si* $\Pi_1 : u \longrightarrow_\gamma u_1$ *et* $\Pi_2 : u \longrightarrow_\gamma u_2$,
   alors $\exists u'$ *tel que* $u_1 \longrightarrow_\gamma^* u'$ *et* $u_2 \longrightarrow_\gamma^* u'$
2. *Si* $\Pi_1 : \alpha \longrightarrow_\gamma \alpha_1$ *et* $\Pi_2 : \alpha \longrightarrow_\gamma \alpha_2$,
   alors $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_\gamma^* \alpha'$ *et* $\alpha_2 \longrightarrow_\gamma^* \alpha'$

Démonstration. Par récurrence sur la taille de u ou de α.

– Si Π_1 et Π_2 terminent par des règles de congruence, la preuve est immédiate.

– Si Π_1 termine par la règle de conversion permutative suivante :

$$\frac{}{\{case\ s \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \ \longrightarrow_\gamma\ \{case\ s \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n}$$

Si Π_2 termine par la même règle, la preuve est triviale.

Si Π_2 termine par une règle de congruence, on distingue deux cas :

1.
$$\frac{\dfrac{t_1 \longrightarrow_\gamma t'_1 \cdots t_n \longrightarrow_\gamma t'_n \qquad s \longrightarrow_\gamma s'}{\{case\ s \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \longrightarrow_\gamma \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n} \qquad v \longrightarrow_\gamma v'}{\{case\ s \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \longrightarrow_\gamma \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n\ v'}$$

On a :
$\{case\ s \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n \longrightarrow_\gamma \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\ v'\}_{i=1}^n$ par congruence,
et $\{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n\ v' \longrightarrow_\gamma \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\ v'\}_{i=1}^n$ par conversion permutative.

2.
$$\frac{\begin{array}{c}\{case\ \{case\ w \text{ of } (d_j\ y_j) \to u_j\}_{j=1}^m \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \\ \longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\}_{j=1}^m \qquad v \longrightarrow_\gamma v'\end{array}}{\begin{array}{c}\{case\ \{case\ w \text{ of } (d_j\ y_j) \to u_j\}_{j=1}^m \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \\ \longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\}_{j=1}^m\ v'\end{array}}$$

avec $s = \{case\ w \text{ of } (d_j\ y_j) \to u_j\}_{j=1}^m$.
On a :
$\{case\ s \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n$
$\longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n\}_{j=1}^m$ (conversion permutative)
$\longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\ v'\}_{i=1}^n\}_{j=1}^m$ (congruence)
et
$\{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\}_{j=1}^m\ v'$
$\longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v'\}_{j=1}^m$ (conversion permutative)
$\longrightarrow_\gamma \{case\ w \text{ of } (d_j\ y_j) \to \{case\ u_j \text{ of } (c_i\ x_i) \to t_i\ v'\}_{i=1}^n\}_{j=1}^m$ (conversion permutative)

– Si Π_1 termine par une autre règle de conversion permutative, la preuve est similaire au cas précédent.

□

La conjonction de la confluence faible et de la normalisation forte implique la confluence, puisqu'on peut établir une récurrence sur la longueur maximale des séquences de réductions partant de l'objet considéré. $\longrightarrow_\gamma$ étant une sous-relation de $\longrightarrow_{\beta\gamma}$, elle est bien fortement normalisante, et donc confluente. Nous n'en donnerons cependant pas d'écriture formelle, car la confluence faible est suffisante pour la suite.

2.4.3 Commutativité entre $\longrightarrow_\beta$ et $\longrightarrow_\gamma$

La commutativité exprime le fait qu'une convergence a également lieu lorsque la divergence implique une $\longrightarrow_\gamma$ et une $\longrightarrow_\beta$.

La β-réduction introduisant des substitutions, il est nécessaire dans un premier temps de montrer que $\longrightarrow_\gamma$ commute avec cet opérateur de substitution.

Lemme 11 (Substitution pour $\longrightarrow_\gamma$)**.**

1. *Si* $t \longrightarrow_\gamma^* t'$ *et* $\Pi : u \longrightarrow_\gamma u'$*, alors* $u[x := t] \longrightarrow_\gamma^* u'[x := t']$.
2. *Si* $t \longrightarrow_\gamma^* t'$ *et* $\Pi : \alpha \longrightarrow_\gamma \alpha'$*, alors* $\alpha[x := t] \longrightarrow_\gamma^* \alpha'[x := t']$.

Démonstration. Immédiat par récurrence sur Π.

□

La propriété de commutativité que nous établissons à présent est faible, dans un sens similaire à la confluence faible : les divergences se font en une étape, mais pas toutes les convergences. Néanmoins, c'est le cas de la convergence impliquant $\longrightarrow_\beta$. Nous qualifions cette forme de commutativité de locale.

Lemme 12 (Commutativité locale de $\longrightarrow_\beta$ et $\longrightarrow_\gamma$)**.**

1. *Si* $\Pi_1 : t \longrightarrow_\beta t_1$ *et* $\Pi_2 : t \longrightarrow_\gamma t_2$*, alors* $\exists t'$ *tel que* $t_1 \longrightarrow_\gamma^* t'$ *et* $t_2 \longrightarrow_\beta t'$
2. *Si* $\Pi_1 : \alpha \longrightarrow_\beta \alpha_1$ *et* $\Pi_2 : \alpha \longrightarrow_\gamma \alpha_2$*, alors* $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_\gamma^* \alpha'$ *et* $\alpha_2 \longrightarrow_\beta \alpha'$

Démonstration. Par récurrence sur la taille de t, et selon les règles qui terminent Π_1 et Π_2

– Si Π_1 et Π_2 terminent par des règles de congruences, la récurrence est immédiate.

– Si Π_2 termine par la règle de conversion permutative suivante :

$$\frac{}{\{case\ u \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \longrightarrow_\gamma \{case\ u \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n}$$

Si Π_1 termine par une règle de congruence, on distingue deux cas :

1.
$$\frac{\dfrac{t_1 \longrightarrow_\beta t'_1 \cdots t_n \longrightarrow_\beta t'_n \qquad s \longrightarrow_\beta s'}{\{case\ s \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \longrightarrow_\beta \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n} \qquad v \longrightarrow_\beta v'}{\{case\ s \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \longrightarrow_\beta \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n\ v'}$$

On a :
$\{case\ s \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n \longrightarrow_\beta \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\ v'\}_{i=1}^n$ par congruence,
et $\{case\ s' \text{ of } (c_i\ x_i) \to t'_i\}_{i=1}^n\ v' \longrightarrow_\gamma \{case\ s' \text{ of } (c_i\ x_i) \to t'_i\ v'\}_{i=1}^n$ par conversion permutative.

2.
$$\frac{\dfrac{t_j \longrightarrow_\beta t'_j \qquad k \longrightarrow_\beta k'}{\{case\ c_j\ k \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n \longrightarrow_\beta t'_j[x_j := k']} \qquad v \longrightarrow_\beta v'}{\{case\ c_j\ k \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n\ v \longrightarrow_\beta t'_j[x_j := k']\ v'}$$

avec $s = c_j\ k$.

On a :
$\{case\ s \text{ of } (c_i\ x_i) \to t_i\ v\}_{i=1}^n \longrightarrow_\beta (t'_j\ v')[x_j := k']$ par β-réduction,
et $t'_j[x_j := k']\ v' = (t'_j\ v')[x_j := k']$ car x_j n'apparait pas dans v, et n'apparait donc pas dans v'.

– Si Π_2 termine par une autre règle de conversion permutative, la preuve est similaire au cas précédent.

– Si Π_1 termine par la règle de β-réduction :

$$\frac{v \longrightarrow_\beta v_1 \qquad u \longrightarrow_\beta u_1}{(\lambda^0 x : \alpha.v)\, u \longrightarrow_\beta v_1[x := u_1]}$$

Π_2 ne peut que terminer par une règle de congruence :

$$\frac{\dfrac{\alpha \longrightarrow_\gamma \alpha' \qquad v \longrightarrow_\gamma v_2}{\lambda^0 x : \alpha.v \longrightarrow_\gamma \lambda^0 x : \alpha'.v_2} \qquad u \longrightarrow_\gamma u_2}{(\lambda^0 x : \alpha.v)\, u \longrightarrow_\gamma (\lambda^0 x : \alpha'.v_2)\, u_2}$$

Par hypothèse de récurrence, $\exists u', v'$ tels que $u_1 \longrightarrow_\gamma^* u'$, $u_2 \longrightarrow_\beta u'$, $v_1 \longrightarrow_\gamma^* v'$ et $v_2 \longrightarrow_\beta v'$.

$v_1[x := u_1] \longrightarrow_\gamma^* v'[x := u']$ par le lemme 11 (substitution pour $\longrightarrow_\gamma$).
$(\lambda^0 x : \alpha'.v_2)\, u_2 \longrightarrow_\beta v'[x := u']$ par β-réduction.

Si Π_1 termine par une autre règle de β-réduction, la preuve est similaire au cas précédent.

□

On dispose à présent de tous les éléments pour obtenir la confluence de $\longrightarrow_{\beta\gamma}$. Là encore, on commence par énoncer la confluence faible.

Lemme 13 (Confluence faible de $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $t \longrightarrow_{\beta\gamma} t_1$ *et* $t \longrightarrow_{\beta\gamma} t_2$,
   alors $\exists t'$ *tel que* $t_1 \longrightarrow_{\beta\gamma}^* t'$ *et* $t_2 \longrightarrow_{\beta\gamma}^* t'$.
2. *Si* $\alpha \longrightarrow_{\beta\gamma} \alpha_1$ *et* $\alpha \longrightarrow_{\beta\gamma} \alpha_2$,
   alors $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_{\beta\gamma}^* \alpha'$ *et* $\alpha_2 \longrightarrow_{\beta\gamma}^* \alpha'$.

Démonstration. Les différents cas sont tous couverts par les lemmes 8 (propriété du diamant pour $\longrightarrow_\beta$), 10 (confluence faible pour $\longrightarrow_\gamma$) et 12 (commutativité locale de $\longrightarrow_\beta$ et $\longrightarrow_\gamma$).

□

$\longrightarrow_{\beta\gamma}$ étant fortement normalisante, on obtient la propriété de confluence.

Théorème 3 (Confluence de $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $t \longrightarrow_{\beta\gamma}^* t_1$ *et* $t \longrightarrow_{\beta\gamma}^* t_2$,
   alors $\exists t'$ *tel que* : $t_1 \longrightarrow_{\beta\gamma}^* t'$ *et* $t_2 \longrightarrow_{\beta\gamma}^* t'$.
2. *Si* $\alpha \longrightarrow_{\beta\gamma}^* \alpha_1$ *et* $\alpha \longrightarrow_{\beta\gamma}^* \alpha_2$,
   alors $\exists \alpha'$ *tel que* $\alpha_1 \longrightarrow_{\beta\gamma}^* \alpha'$ *et* $\alpha_2 \longrightarrow_{\beta\gamma}^* \alpha'$.

Démonstration. Immédiat par récurrence sur la longueur maximale des séquences de $\longrightarrow_{\beta\gamma}$ partant de t, en utilisant le lemme 14 (confluence faible de $\longrightarrow_{\beta\gamma}$).

□

On dispose à présent de tous les éléments pour obtenir la confluence de $\longrightarrow_{\beta\gamma}$. La version donnée établit de plus la conservation du nombre d'étapes $\longrightarrow_\beta$ impliquées entre les séquences divergentes et les séquences convergentes. On introduit donc d'abord une notation pour compter le nombre de $\longrightarrow_\beta$ dans une séquence de $\longrightarrow_{\beta\gamma}$.

Notation 3. *Soit* S *une séquence d'étapes de* $\longrightarrow_{\beta\gamma}$. S *est donc composée d'étapes* $\longrightarrow_\beta$ *et d'étapes* $\longrightarrow_\gamma$ *, et on notera* $\beta(S)$ *le nombre d'étapes de* S *qui sont des* $\longrightarrow_\beta$.

Là encore, on commence par énoncer la confluence faible.

Lemme 14 (Confluence faible de $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $S_1 : t \longrightarrow_{\beta\gamma} t_1$ *et* $S_2 : t \longrightarrow_{\beta\gamma} t_2$,
   alors $\exists t'$ *tel que* $S'_1 : t_1 \longrightarrow_{\beta\gamma} t'$ *et* $S'_2 : t_2 \longrightarrow_{\beta\gamma} t'$, $\beta(S_1) = \beta(S'_2)$ *et* $\beta(S_2) = \beta(S'_1)$.
2. *Si* $S_1 : \alpha \longrightarrow_{\beta\gamma} \alpha_1$ *et* $S_2 : \alpha \longrightarrow^*_{\beta\gamma} \alpha_2$,
   alors $\exists \alpha'$ *tel que* $S'_1 : \alpha_1 \longrightarrow_{\beta\gamma} \alpha'$ *et* $S'_2 : \alpha_2 \longrightarrow_{\beta\gamma} \alpha'$, $\beta(S_1) = \beta(S'_2)$ *et* $\beta(S_2) = \beta(S'_1)$.

Démonstration. Les différents cas sont tous couverts par les lemmes 8 (propriété du diamant pour $\longrightarrow_\beta$), 10 (confluence faible pour $\longrightarrow_\gamma$) et 12 (commutativité locale de $\longrightarrow_\beta$ et $\longrightarrow_\gamma$).

□

$\longrightarrow_{\beta\gamma}$ étant fortement normalisante, on obtient la propriété de confluence.

Théorème 4 (Confluence de $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $S_1 : t \longrightarrow^*_{\beta\gamma} t_1$ *et* $S_2 : t \longrightarrow^*_{\beta\gamma} t_2$,
   alors $\exists t'$ *tel que* $S'_1 : t_1 \longrightarrow^*_{\beta\gamma} t'$ *et* $S'_2 : t_2 \longrightarrow^*_{\beta\gamma} t'$, $\beta(S_1) = \beta(S'_2)$ *et* $\beta(S_2) = \beta(S'_1)$
2. *Si* $\Pi_1 : \alpha \longrightarrow^*_{\beta\gamma} \alpha_1$ *et* $\Pi_2 : \alpha \longrightarrow^*_{\beta\gamma} \alpha_2$,
   alors $\exists \alpha'$ *tel que* $\Pi'_1 : \alpha_1 \longrightarrow^*_{\beta\gamma} \alpha'$ *et* $\Pi'_2 : \alpha_2 \longrightarrow^*_{\beta\gamma} \alpha'$, $\beta(S_1) = \beta(S'_2)$ *et* $\beta(S_2) = \beta(S'_1)$

Démonstration. Immédiat par récurrence sur la longueur maximale des séquences de $\longrightarrow_{\beta\gamma}$ partant de t, en utilisant le lemme 14 (confluence faible de $\longrightarrow_{\beta\gamma}$).

□

La relation de réécriture $\longrightarrow_{\beta\gamma}$ est donc fortement normalisante et confluente, ce qui signifie que chaque objet possède une unique forme $\beta\gamma$-normale. Dans la suite, nous noterons pour un objet o donné cette forme normale $NF_{\beta\gamma}(o)$.

Il est immédiat de vérifier que deux objets sont $\beta\gamma$-équivalents si et seulement si ils ont la même forme normale. Comme il suffit pour calculer la forme normale d'appliquer suffisamment d'étapes arbitraires de $\longrightarrow_{\beta\gamma}$, la $\beta\gamma$-équivalence entre deux objets est donc décidable.

Le calcul restreint à la relation $\longrightarrow_{\beta\gamma}$, où la relation d'équivalence utilisée dans la règle (eq.) est définie comme la $\beta\gamma$-équivalence, dispose donc de propriétés satisfaisantes pour être utilisé comme support des ACG étendues : les formes normales sont calculables, et leur problème d'inférence de type est rendu possible par la propriété de la sous-formule.

Un tel calcul présente néanmoins comme inconvénient de ne pas disposer d'η-conversion, empêchant d'identifier des termes qu'on souhaiterait l'être. A titre d'exemple, si un vocabulaire abstrait Σ contient une constante f de type fonctionel, qui est envoyée par un lexique $\mathscr{L}$ vers une λ-abstraction $u = \lambda x : \alpha.t$, alors ce terme u aura au moins deux antécédents en forme normale par $\mathscr{L}$: f et $\lambda x : \alpha.(f\ x)$. L'absence de relation d'η-conversion engendre donc des ambiguités lors de l'inversion du lexique qui n'ont aucune pertinence linguistique.

Le chapitre suivant se propose de supprimer ces ambiguités non désirées en discutant de l'introduction d'une η-conversion dans le calcul.

Chapitre 3

Propriétés fondamentales de $\longrightarrow_{\beta\gamma\eta}$ dans le λ-Π-calcul linéaire avec produit et somme

En λ-calcul, l'η-conversion est la relation permettant d'identifier un terme de type fonctionnel f avec le terme $\lambda x.(f\ x)$, où x n'apparait pas dans f. Les deux termes génèrent en effet le même résultat quel que soit l'argument auquel ils sont appliqués. Ce principe d'η-conversion s'étend aux autres constructeurs de types qui ont été introduits dans notre calcul. On souhaite ainsi avoir les relations suivantes :

- $t =_\eta \lambda x.(t\ x)$ si t a pour type un produit dépendant ;
- $t =_\eta \lambda^0 x.(t\ x)$ si t a pour type une implication linéaire ;
- $t =_\eta [l_i = t.l_i]_{i=1}^n$ si t a un type enregistrement ;
- $t =_\eta \{case\ t\ \text{of}\ (c_i\ x_i) \to (c_i\ x_i)\}_{i=1}^n$ si t a un type variant.

L'enjeu est alors de traduire ces relations par des règles de réécriture préservant la confluence et la normalisation du calcul.

Dans ce chapitre, nous nous intéressons aux propriétés de confluence et de normalisation lorsque l'η-conversion est ajoutée au calcul. Les preuves que nous en proposons restent cependant tributaires de plusieurs hypothèses que nous explicitons. La combinaison des présences du produit dépendant et des analyses de cas entraîne en effet de sérieuses complications dans le traitement de l'η-conversion : comme noté par Balat et al. (2004), les analyses de cas nécessitent d'utiliser une η-conversion orientée dans le sens de l'expansion. En effet, en orientant l'η-conversion dans le sens de la réduction, la confluence est perdue, comme l'illustre l'exemple suivant :

$$\lambda x.(\{case\ t\ \text{of}\ (c_i\ x_i) \to t_i\}_{i=1}^n\ x) \longrightarrow_\eta (\{case\ t\ \text{of}\ (c_i\ x_i) \to t_i)\}_{i=1}^n$$
$$\lambda x.(\{case\ t\ \text{of}\ (c_i\ x_i) \to t_i\}_{i=1}^n\ x) \longrightarrow_\gamma \lambda x.\{case\ t\ \text{of}\ (c_i\ x_i) \to t_i\ x\}_{i=1}^n.$$

Sans conversion permutative non-standard permettant de déplacer l'abstraction à l'intérieur de l'analyse de cas, les deux termes résultants sont impossibles à faire converger. Par ailleurs, même si l'on disposait des conversions permutatives non-standards, la confluence resterait perdue, comme l'illustre l'exemple :

$$\lambda x.\{case\ t\ \text{of}\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n \longrightarrow_\eta \lambda x.t$$
$$\lambda x.\{case\ t\ \text{of}\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n \longrightarrow_\gamma \{case\ t\ \text{of}\ (c_i\ x_i) \to \lambda x.c_i\ x_i\}_{i=1}^n.$$

En rentrant dans l'analyse de cas par conversion permutative, l'abstraction modifie la forme de cette analyse de cas de sorte qu'il n'est plus possible de l'η-réduire. Inverser le sens des conversions permutatives n'est évidemment pas une solution, comme le montre l'exemple :

$$\{case\ u\ of\ (c_1\ x_1) \to t\ v | (c_2\ x_2) \to (\lambda x.c)\ v\} \longrightarrow_\gamma \{case\ u\ of\ (c_1\ x_1) \to t | (c_2\ x_2) \to (\lambda x.c)\}\ v$$

$$\{case\ u\ of\ (c_1\ x_1) \to t\ v | (c_2\ x_2) \to (\lambda x.c)\ v\} \longrightarrow_\beta \{case\ u\ of\ (c_1\ x_1) \to t\ v | (c_2\ x_2) \to c\},$$

où aucune règle ne permet de faire converger les termes résultants.

Il est donc nécessaire d'orienter l'η-conversion comme une expansion. Ce sens demande de s'assurer que certaines positions ne puissent pas être expansées, lorsqu'un β-rédex serait créé dans l'opération. En effet sinon la normalisation du calcul n'est plus assurée, comme le montre l'exemple suivant :

$$f\ x \longrightarrow_\eta (\lambda y.(f\ y))\ x \longrightarrow_\beta f\ x \longrightarrow_\eta \dots$$

C'est pour intégrer cette contrainte dans le calcul que nous avons introduitk, de façon classique, la relation associée $\longrightarrow_c$, qui ne permet pas d'expanser un terme au niveau courant. Dans les règles de congruence, les prémisses utilisent alors $\longrightarrow_\eta$ dans les positions où une expansion est valide, et $\longrightarrow_c$ dans une position où une expansion violerait la normalisation du calcul.

Le sens de l'expansion nécessite également de formuler le système à la Church, c'est-à-dire en indiquant lors de chaque abstration le type de la variable abstraite. Sans ces étiquettes de types, lη-expansion ne serait en effet pas normalisable, comme l'illustre l'exemple suivant :

$$(\lambda x.c)\ (\lambda y.y) \longrightarrow_\eta (\lambda x.c)\ (\lambda y.\lambda z.(y\ z)) \longrightarrow_\eta (\lambda x.c)\ (\lambda y.\lambda z.(y\ \lambda z'.(z\ z'))) \longrightarrow_\eta \dots$$

Le terme global expansé dans cet exemple peut toujours être typé par le type de la constante c. Par contre, à chaque itération de $\longrightarrow_\eta$, les types correspondant aux variables abstraites $x, y, z, \dots$ sont de plus en plus complexes. Cet exemple montre donc que le typage d'un terme n'est pas suffisant pour être assuré du bon comportement de l'η-expansion. Le type des variables abstraites doit également être fixé, d'où les étiquettes de types. On peut également montrer que l'absence de ces étiquettes compromet la confluence de la relation, avec l'exemple :

$$\begin{array}{l}(\lambda x.c)\ (\lambda y.y) \longrightarrow_\eta (\lambda x.c)\ (\lambda y.\lambda z.(y\ z)) \\ (\lambda x.c)\ (\lambda y.y) \longrightarrow_\eta (\lambda x.c)\ (\lambda y.[l_i = y.l_i]_{i=1}^n\end{array}$$

qui fonctionne sur le même principe : le type du terme de départ est fixé, mais ce n'est pas suffisant pour fixer le type de la variable y, qui peut donc être expansé comme une fonction dans le premier cas et comme un enregistrement dans le second cas. Les deux termes résultants ne pourront donc plus confluer.

La problématique induite par l'introduction des étiquettes de type est que lorsqu'une telle étiquette apparait à la suite de l'η-expansion d'un terme fonctionnel, il faut pouvoir s'assurer que cette étiquette ne contient pas elle-même de nouveaux rédexs arbitraires, à cause des types dépendants. Ainsi, comme noté par Ghani (1997), la normalisation de l'η-expansion est perdue si les étiquettes sont introduites sans contrôle, comme l'illustre l'exemple :

$$\begin{array}{l}x : \alpha \multimap \alpha \vdash_\Sigma x \longrightarrow_\eta \lambda^0 z : ((\lambda y : \alpha \multimap \alpha.\alpha)\ x).(x\ z) \\ \longrightarrow_\eta \lambda^0 z : ((\lambda y : \alpha \multimap \alpha.\alpha)\ \lambda^0 z' : ((\lambda y' : \alpha \multimap \alpha.\alpha)\ x).(x\ z')).(x\ z) \longrightarrow_\eta \dots\end{array}$$

Dans cet exemple, la variable x apparait dans l'étiquette introduite par sa propre expansion, car on a bien $(\lambda y : \alpha \multimap \alpha.\alpha)\ x \longrightarrow_\beta \alpha$, donc rien n'interdit d'attribuer à x le type $((\lambda y : \alpha \multimap \alpha.\alpha)\ x) \multimap \alpha$. De plus, la position dans laquelle x apparait est valide pour une nouvelle η-expansion (x apparait dans une position d'argument et non d'appliquant). Le processus peut donc être répété un nombre arbitraire de fois.

Comme noté par Barthe (1999a), il est suffisant pour prévenir ce phénomène de demander aux étiquettes créées par η-expansion d'être en forme β-normale. Cependant, forcer ces étiquettes à être en forme $\beta\gamma\eta$-normale facilite la preuve de la normalisation faible d' $\longrightarrow_\eta$, car l'on n'a alors pas à se soucier de prouver que ces étiquettes sont elles-mêmes normalisables. C'est cette stratégie que nous choisissons d'appliquer dans ce chapitre. Afin de pouvoir faire référence aux formes $\beta\gamma\eta$-normales au sein même de la définition de $\longrightarrow_\eta$, nous passons, suivant Barthe et Ghani, par une définition syntaxique de ces formes, donnée dans la section 2.2.4.

La preuve que nous établissons dans ce chapitre nécessite que $\longrightarrow_\eta$ conserve la $\beta\gamma$-normalité. Cette propriété est cependant fausse a priori, à chaque fois qu'une analyse de cas subit une η-expansion :

$\{case\ t \text{ of } (c_i\ x_i) \rightarrow f_i\}_{i=1}^n \longrightarrow_\eta \lambda x : \alpha.(\{case\ t \text{ of } (c_i\ x_i) \rightarrow f_i\}_{i=1}^n\ x) \longrightarrow_\gamma \lambda x : \alpha.\{case\ t \text{ of } (c_i\ x_i) \rightarrow f_i\ x\}_{i=1}^n$

Même si le terme de départ est une forme $\beta\gamma$-normale, son expansion permet forcément une conversion permutative, qui peut elle-même former de nouveaux β ou γ-rédex. Afin de bénéficier tout de même de cette propriété de conservation de la $\beta\eta$-normalité, nous avons introduit la notation $\downarrow$, qui permet de réduire implicitement tous ces rédex simples formés par une expansion. Un terme fonctionnel f s'expansera donc vers $\lambda x : \alpha.(f \downarrow x)$ plutôt que $\lambda x : \alpha.(f\ x)$. Ainsi on a :

$$\{case\ t \text{ of } (c_i\ x_i) \rightarrow \lambda y : \alpha.(f_i\ y)\}_{i=1}^n \longrightarrow_\eta \lambda x : \alpha.\{case\ t \text{ of } (c_i\ x_i) \rightarrow f_i\ x\}_{i=1}^n,$$

où la conversion permutative et la β-réduction qui auraient dû résulter de cette expansion ont été faites dans cette même étape. Cette modification ne modifie pas le comportement global de $\longrightarrow_\eta$, et permet simplement de pouvoir exprimer le lemme d'ajournement de $\longrightarrow_\eta$, et la stratégie de normalisation faible de $\longrightarrow_{\beta\gamma\eta}$, où les expansions sont faites en dernier.

Comme la plupart des lemmes formulés dans ce chapitre ne seraient de toute façon vérifiés que sur des termes typables, ou des types bien formés, nous avons formulé les règles d'expansion, comme elles apparaissent dans la section 2.2.5, de sorte à ce qu'elles procèdent au passage à un typage du terme. Ainsi les preuves de ces lemmes ne considère qu'un seul arbre de dérivation plutôt que deux, ce qui simplifie les récurrences.

Cervesato et Pfenning (1996) évitent les problématiques liées à la combinaison de l'η-conversion et du produit dépendant en n'intégrant pas de règle d'η-expansion parmi les règles de réécritures, et en utilisant un système de typage tel que seules les formes normales pour l'η-expansion, ou formes η-longues, sont typables. Cette solution part donc du principe que tous les termes et types manipulés par le calcul sont d'emblée en forme η-longue. Elle repose de plus sur le fait que cet ensemble des formes η-longues est stable par β-réduction. Cette propriété, classique pour le lambda calcul simplement typé, reste bien vérifiée dans le $\lambda\Pi$-calcul de Cervesato. Cependant, elle n'est pas conservée lorsque le calcul intègre des analyses de cas sans disposer des conversions permutatives non standards, ce qui est notre cas. En effet, dans l'exemple suivant :

$$\{case\ t \text{ of } (c_i\ x_i) \rightarrow \lambda x : \alpha.(f_i\ x)\}_{i=1}^n\ u \longrightarrow_\gamma \{case\ t \text{ of } (c_i\ x_i) \rightarrow (\lambda x : \alpha.(f_i\ x))\ u\}_{i=1}^n$$

le terme de départ est bien en forme η-longue, si u l'est lui-même, mais après conversion permutative, le terme résultant ne l'est plus. En effet le terme entier est de type fonctionnel et peut donc encore être expansé. Lorsque le calcul dispose, comme celui de Lindley (2007), des conversions permutatives non-standards, il est possible de se passer de l'expansion des analyses de cas, puisque la forme expansée se ramène à la forme de départ par conversions permutatives et β-réductions, comme l'illustre l'exemple :

$$\begin{aligned}
&\lambda y : \alpha.(\{case\ t \text{ of } (c_i\ x_i) \rightarrow \lambda x : \alpha.(f_i\ x)\}_{i=1}^n\ y)\\
&\longrightarrow_\gamma^2 \{case\ t \text{ of } (c_i\ x_i) \rightarrow \lambda y : \alpha.((\lambda x : \alpha.(f_i\ x))\ y)\}_{i=1}^n\\
&\longrightarrow_\beta \{case\ t \text{ of } (c_i\ x_i) \rightarrow \lambda x : \alpha.(f_i\ x)\}_{i=1}^n.
\end{aligned}$$

Cette identification n'est cependant pas possible sans les conversions permutatives non-standards (permettant dans l'exemple précédent de déplacer l'abstraction de y à l'intérieur de l'analyse de cas), et notre choix de ne pas traiter ce type de conversion empêche donc cette possibilité de travailler sur les formes η-longues.

Afin de démontrer la confluence et la normalisation de $\longrightarrow_{\beta\gamma\eta}$, nous procédons comme suit : dans un premier temps nous établissons une propriété de commutativité entre $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_\eta$, modulo la relation $\approx$. Dans un second temps nous utilisons ce résultat pour démontrer que les formes $\beta\gamma\eta$-normales sont uniques pour chaque classe d'équivalence de $\equiv$. Enfin, nous établissons la normalisation faible de $\longrightarrow_{\beta\gamma\eta}$, qui implique alors la confluence, puisque deux termes équivalents se réduisent vers une forme normale unique.

3.1 Commutativité de $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_{\eta}$

Dans cette section, nous démontrons une propriété de commutativité entre $\longrightarrow_{\eta}$ et $\longrightarrow_{\beta\gamma}$ cruciale pour établir l'unicité des formes normales. Nous commençons d'abord par établir des lemmes préliminaires posant des propriétés basiques de $\longrightarrow_{\eta}$ ou de $\downarrow$. Dans une seconde étape, nous établissons une propriété de substitution pour $\longrightarrow_{\eta}$. Enfin, nous en déduisons la propriété de commutativité recherchée.

La preuve de la propriété de substitution pour $\longrightarrow_{\eta}$ utilise néanmoins deux hypothèses pour lesquelles nous ne disposons pas de démonstration. Ces hypothèses semblent toutefois suffisamment raisonnables pour laisser espérer qu'une preuve complète puisse reprendre les lignes que nous explorons ici.

3.1.1 Lemmes préliminaires

Le premier lemme de ce chapitre énonce les contraintes qui gouvernent la présence des variables libres dans les jugements de typage. Comme l'η-expansion se comporte notamment comme un jugement de typage, ce lemme pourra également, à l'aide du lemme suivant, être invoqué pour décrire les contraintes sur les variables libres présentes dans un séquent de $\longrightarrow_{\eta}$.

Lemme 15 (Variables libres)**.**

1. *Si* $\Pi :\ \Gamma \vdash_\Sigma K :$ **`kind`** *alors FV(* K *)*$= \emptyset$

2. *Si* $\Pi :\ \Gamma \vdash_\Sigma \alpha : K$ *alors*
   - *FV(*α*)* $\subseteq$ *dom(*Γ*)*
   - *FV(* K *)*$= \emptyset$
   - *FV(*Γ*)*$= \emptyset$

3. *Si* $\Pi :\ \Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$ *alors*
   - *FV(*t*)* $\subseteq$ *dom(*$\Gamma;\Delta$*)*
   - *FV(*α*)* $\subseteq$ *dom(*Γ*)*

4. *Si* $\Pi :\ \Gamma, x : \alpha, \Gamma'\,;\,\Delta \vdash_\Sigma t : \beta$ *alors FV(*α*)* $\cup$ *FV(*Γ*)* $\subseteq$ *dom(*Γ*)*

5. *Si* $\Pi :\ \Gamma\,;\,\Delta, x : \alpha, \Delta' \vdash_\Sigma t : \beta$ *alors FV(*α*)* $\cup$ *FV(*$\Gamma;\Delta$*)* $\subseteq$ *dom(*Γ*)*

6. *Si* $\Pi :$ *sig* (Σ) *alors FV(*Σ*)*$= \emptyset$

7. *Si* $\Pi :\ \Gamma\,;\,\Delta \vdash_\Sigma t : \beta$*, alors* $x \in$ *dom(*Δ*) si et seulement si (*x *apparaît libre une et une seule fois dans* t *et* $x \notin$ *dom(*Γ*)).*

Démonstration. Par récurrence sur la structure de Π, selon la dernière règle utilisée :

- $\dfrac{\text{sig}\,(\Sigma) \qquad \vdash_\Sigma \alpha :\ \texttt{type}}{\text{sig}\,(\Sigma;\ c : \alpha)}$

  par l'hypothèse de récurrence, on a FV(Σ)$= \emptyset$ et FV(α) $\subseteq\ \emptyset$
  D'où FV($\Sigma;\ c : \alpha$)$= \emptyset$
  Les autres règles de signatures sont tout aussi immédiates, ainsi que les règles de formation des kinds et la règle (type const.).

- $\dfrac{\vdash_\Sigma \alpha :\ \texttt{type} \qquad \Gamma \vdash_\Sigma \beta : \text{K}}{\Gamma, x : \alpha \vdash_\Sigma \beta : \text{K}}$ (type weak.)

  par l'hypothèse de récurrence, FV(α) $=\ \emptyset$, FV(β) $\subseteq$ dom(Γ), FV(K) $=\ \emptyset$ et FV(Γ) $=\ \emptyset$.
  On en déduit que FV($\Gamma, x : \alpha$) $=\ \emptyset$ et que FV(β) $\subseteq$ dom($\Gamma, x : \alpha$).

– $\dfrac{\Gamma, x : \alpha \vdash_\Sigma \beta : \mathrm{K}}{\Gamma \vdash_\Sigma \lambda x.\beta : (\alpha)\,\mathrm{K}}$ (type abs.)

par l'hypothèse de récurrence, FV(α) = $\emptyset$, FV(β) $\subseteq$ dom(Γ)$\cup\{x\}$, FV(K) = $\emptyset$ et FV($\Gamma, x : \alpha$) = $\emptyset$.
On en déduit que FV(Γ) = $\emptyset$, FV($\lambda x.\beta$) $\subseteq$ dom(Γ) et FV($(\alpha)\,\mathrm{K}$) = $\emptyset$.

– $\dfrac{\Gamma \vdash_\Sigma \alpha : (\beta)\,\mathrm{K} \qquad \Gamma ; \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma \alpha\, t : \mathrm{K}}$ (type app.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV($(\beta)\,\mathrm{K}$) = $\emptyset$, FV(t) $\subseteq$dom(Γ) et FV(Γ) = $\emptyset$.
On en déduit que FV($\alpha\, t$) $\subseteq$ dom(Γ) et FV(K) = $\emptyset$.

– (lin. fun.) immédiat.

– $\dfrac{\Gamma \vdash_\Sigma \alpha : \mathtt{type} \qquad \Gamma, x : \alpha \vdash_\Sigma \beta : \mathtt{type}}{\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta : \mathtt{type}}$ (dep. prod.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV(β) $\subseteq$ dom(Γ)$\cup\{x\}$, et FV(Γ) = $\emptyset$.
On en déduit que FV($(\Pi x : \alpha)\,\beta$) $\subseteq$ dom(Γ).

– (const.) immédiat.

– $\dfrac{\Gamma \vdash_\Sigma \alpha : \mathtt{type}}{\Gamma\,;\, x : \alpha \vdash_\Sigma x : \alpha}$ (lin. var.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ) et FV(Γ) = $\emptyset$.
On en déduit que $\Gamma\,;\, x : \alpha$ vérifie les propriétés 4 et 5.

– (var.) semblable.

– $\dfrac{\Gamma \vdash_\Sigma \alpha : \mathtt{type} \qquad \Gamma\,;\, \Delta \vdash_\Sigma t : \beta}{\Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t : \beta}$ (weak.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV(β) $\subseteq$ dom(Γ), FV(t) $\subseteq$ dom($\Gamma\,;\,\Delta$), FV(Γ) = $\emptyset$ et $\Gamma\,;\,\Delta$ vérifie les propriétés 4 et 5.
On en déduit que $\Gamma, x : \alpha\,;\,\Delta$ vérifie les propriétés 4 et 5, que FV(t) $\subseteq$ dom($\Gamma, x : \alpha\,;\,\Delta$), que FV($\beta$) $\subseteq$ dom($\Gamma, x : \alpha$).

– $\dfrac{\Gamma\,;\, \Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma\,;\, \Delta \vdash_\Sigma \lambda^0 x : \alpha.t : \alpha \multimap \beta}$ (lin. abs.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV(β) $\subseteq$ dom(Γ), FV(t) $\subseteq$ dom($\Gamma\,;\,\Delta, x : \alpha$) et $\Gamma\,;\,\Delta, x : \alpha$ vérifie les propriétés 4 et 5.
On en déduit que $\Gamma\,;\,\Delta$ vérifie les propriétés 4 et 5, que FV($\lambda^0 x : \alpha.t$) $\subseteq$ dom($\Gamma\,;\,\Delta$), que FV($\alpha \multimap \beta$) $\subseteq$ dom(Γ).

– $\dfrac{\Gamma\,;\, \Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \qquad \Gamma\,;\, \Delta_2 \vdash_\Sigma u : \alpha}{\Gamma\,;\, \Delta_1, \Delta_2 \vdash_\Sigma t\, u : \beta}$ (lin. app.)

immédiat.

– $\dfrac{\Gamma, x : \alpha\,;\, \Delta \vdash_\Sigma t : \beta}{\Gamma\,;\, \Delta \vdash_\Sigma \lambda x.t : (\Pi x : \alpha)\,\beta}$ (abs.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV(β) $\subseteq$ dom($\Gamma, x : \alpha$), FV(t) $\subseteq$ dom($\Gamma, x : \alpha\,;\,\Delta$) et $\Gamma, x : \alpha\,;\,\Delta$ vérifie les propriétés 4 et 5. Par ailleurs, x ne peut pas apparaître libre dans Δ
On en déduit que $\Gamma\,;\,\Delta$ vérifie les propriétés 4 et 5, que FV($\lambda x.t$) $\subseteq$ dom($\Gamma\,;\,\Delta$), que FV($(\Pi x : \alpha)\,\beta$) $\subseteq$ dom(Γ) et que FV($(\Pi x : \alpha)\,\beta$) $\subseteq$FV(Γ)$\cup$FV(Δ).

– $\dfrac{\Gamma\,;\, \Delta \vdash_\Sigma t : (\Pi x : \alpha)\,\beta \qquad \Gamma\,;\, \vdash_\Sigma u : \alpha}{\Gamma\,;\, \Delta \vdash_\Sigma t\, u : \beta[x := u]}$ (app.)

par l'hypothèse de récurrence, FV(α) $\subseteq$ dom(Γ), FV(u) $\subseteq$ dom(Γ), FV($(\Pi x : \alpha)\,\beta$) $\subseteq$ dom(Γ), FV(t) $\subseteq$ dom(Γ ; Δ) et Γ ; Δ vérifie les propriétés 4 et 5.
On en déduit que FV($t\,u$) $\subseteq$ dom(Γ ; Δ) et que FV($\beta[x := u]$) $\subseteq$ dom(Γ).

– (rec.), (sel.), (inj.), (case.) et (eq.) : immédiat.

□

Le lemme suivant formalise la propriété d' $\longrightarrow_\eta$ de procéder à un jugement de typage : si l'on dispose d'une preuve d'expansion, on peut en extraire un jugement de typage de l'objet expansé. Á l'inverse, si l'on dispose d'un jugement de typage, on peut en extraire une preuve d'expansion de l'objet typé vers lui-même. De plus on peut établir un lien entre la hauteur de la dérivation de départ et la dérivation extraite, qui sera utile pour justifier des mesures de récurrence dans la suite. Pour exprimer ce lien, nous donnons tout d'abord la notation suivante :

Notation 4 (Poids d'une dérivation)**.**
Soit une dérivation Π. *On notera* $p(\Pi)$ *le couple* $\langle a - b, b\rangle$,
où a *est le nombre total d'occurrences de règles apparaissant dans* Π
et b *est le nombre d'occurrences des règles de subsomption et d'expansion apparaissant dans* Π.

Dans la suite, on manipulera cette mesure p *avec l'addition usuelle sur les couples d'entiers, et l'ordre lexicographique usuel sur les couples d'entiers.*

Lemme 16 (Typage et $\longrightarrow_\eta$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$
   Alors $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t : \alpha$
   avec $p(\Pi') < p(\Pi) + \langle 1, 0\rangle$.

2. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha : K$
   Alors $\Pi' : \Gamma \vdash_\Sigma \alpha \longrightarrow_c \alpha' : K$
   avec $p(\Pi') < p(\Pi) + \langle 1, 0\rangle$.

3. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \alpha$
   Alors $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$
   avec $p(\Pi') \leq p(\Pi)$.

4. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta \alpha' : K$
   Alors $\Pi' : \Gamma \vdash_\Sigma \alpha : K$
   avec $p(\Pi') \leq p(\Pi)$.

Démonstration. Immédiat par récurrence sur la hauteur de Π.

□

Le lemme suivant énonce que l'introduction d'une nouvelle variable dans le contexte non linéaire peut se faire à n'importe quelle position, en remontant suffisamment haut dans l'arbre de dérivation pour y insérer une règle (weak).

Lemme 17 (Affaiblissement)**.**

1. *Si* $\Pi : \Gamma_1, \Gamma_2\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (resp. \longrightarrow_c) t' : \alpha$ *et* $\Gamma_1 \vdash_\Sigma \beta :$ **`type`**
   alors $\Gamma_1, x : \beta, \Gamma_2\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (resp. \longrightarrow_c) t' : \alpha$

2. *Si* $\Pi : \Gamma_1, \Gamma_2 \vdash_\Sigma \alpha \longrightarrow_\eta (resp. \longrightarrow_c) \alpha' : K$ *et* $\Gamma_1 \vdash_\Sigma \beta :$ **`type`**
   alors $\Gamma_1, x : \beta, \Gamma_2 \vdash_\Sigma \alpha \longrightarrow_\eta (resp. \longrightarrow_c) \alpha' : K$

3. *Si* $\Pi : \Gamma_1, \Gamma_2 \vdash_\Sigma t \approx t'$ *et* $\Gamma_1 \vdash_\Sigma \beta :$ **`type`**
   alors $\Gamma_1, x : \beta, \Gamma_2 \vdash_\Sigma t \approx t'$.

4. *Si* $\Pi : \Gamma_1, \Gamma_2 \vdash_\Sigma \alpha \approx \alpha'$ *et* $\Gamma_1 \vdash_\Sigma \beta :$ **`type`**
   alors $\Gamma_1, x : \beta, \Gamma_2 \vdash_\Sigma \alpha \approx \alpha'$

Démonstration. Immédiat par récurrence sur $p(\Pi)$, en utilisant le lemme 16 (typage et $\longrightarrow_\eta$).

□

Le lemme suivant établit que deux types équivalents ne peuvent pas être de nature différente.

Lemme 18 (Homogénéité des types)**.**

1. *Si* $\Gamma \vdash_\Sigma \alpha \multimap \beta \equiv \gamma :$ **`type`**
   Alors γ *n'est pas un type atomique, ni de la forme* $(\Pi x : \gamma_1)\,\gamma_2$

2. *Si* $\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta \equiv \gamma :$ **`type`**
   Alors γ *n'est pas un type atomique.*

3. *Si* $\Gamma \vdash_\Sigma a \equiv \gamma :$ **`type`**
   Alors γ *n'est pas un type atomique différent de* a*.*

Démonstration. Par examen des règles de $\longrightarrow_\beta$ et $\longrightarrow_\eta$. Aucune règle ne peut ajouter ou supprimer une implication linéaire, un produit dépendant ou un type atomique.

□

On peut alors en déduire qu'un terme ne peut pas avoir une forme incohérente avec le type qu'il se voit attribuer.

Lemme 19 (Cohérence du typage)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha \multimap \beta$
   Alors t *n'est pas une abstraction non-linéaire, ni un enregistrement.*

2. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : (\Pi x : \alpha)\,\beta$
   Alors t *n'est pas une abstraction linéaire, ni un enregistrement.*

3. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : a$ *et* $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$
   Alors t *n'est pas une abstraction linéaire, ni une abstraction non-linéaire.*

4. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : a$ *et* $binding_\Sigma(a) = \{c_1 \textit{ of } \alpha_1 | \ldots | c_n \textit{ of } \alpha_n\}$
   Alors t *n'est pas une abstraction linéaire, ni une abstraction non-linéaire, ni un enregistrement.*

Démonstration. Immédiat, par récurrence sur la hauteur de la dérivation, en utilisant le lemme 18 (Homogénéité des types).

□

Le lemme suivant est un lemme d'inversion : en considérant un séquent donné, on énumère toutes les possibilités de prémisses ayant permis d'obtenir ce séquent comme conclusion d'une règle de $\longrightarrow_\eta$. Ces prémisses sont également d'un poids inférieur à celui de la dérivation contenant la conclusion, ce qui permettra par la suite d'obtenir d'eux une hypothèse de récurrence.

Lemme 20 (Inversion)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma \lambda^0 x : \alpha.t \;\longrightarrow_\eta (ou \;\longrightarrow_c)\; u : \gamma$
   alors $\exists \alpha', v, \beta$ *tels que :*
   $\Gamma \vdash_\Sigma \gamma \equiv \alpha \multimap \beta :$ **type**,
   $u = \lambda^0 x : \alpha'.v$,
   $\Gamma \vdash_\Sigma \alpha \;\longrightarrow_\eta \alpha' :$ **type** *et*
   $\Pi' : \Gamma\,;\,\Delta, x : \alpha \vdash_\Sigma t \;\longrightarrow_\eta v : \beta$
   avec $p(\Pi') < p(\Pi)$

2. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma \lambda x : \alpha.t \;\longrightarrow_\eta (ou \;\longrightarrow_c)\; u : \gamma$
   alors $\exists \alpha', v, \beta$ *tels que :*
   $\Gamma \vdash_\Sigma \gamma \equiv (\Pi x : \alpha)\,\beta :$ **type**,
   $u = \lambda x : \alpha'.v$,
   $\Gamma \vdash_\Sigma \alpha \;\longrightarrow_\eta \alpha' :$ **type** *et*
   $\Pi' : \Gamma, x : \alpha\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_\eta v : \beta$
   avec $p(\Pi') < p(\Pi)$

3. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma [l_i = t_i]_{i=1}^n \;\longrightarrow_\eta (ou \;\longrightarrow_c)\; u : \gamma$
   alors $\exists t'_1, \cdots, t'_n, a, \alpha_1, \ldots, \alpha_n$ *tels que :*
   $\Gamma \vdash_\Sigma \gamma \equiv a :$ **type**,
   $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$,
   $u = [l_i = t'_i]_{i=1}^n$,
   $\forall i \in \{1, \ldots, n\}, \Pi_i : \Gamma\,;\,\Delta \vdash_\Sigma t_i \;\longrightarrow_\eta t'_i : \alpha_i$
   avec $p(\Pi_i) < p(\Pi)$

4. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma \{case\ v\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \;\longrightarrow_\eta u : \gamma$
   alors soit $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma \{case\ v\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \;\longrightarrow_c\; u : \gamma$
   avec $p(\Pi') < p(\Pi)$
   soit $\exists \gamma', t'_1, \cdots, t'_n, v'$ *tels que :*
   $\Gamma \vdash_\Sigma \gamma \equiv \gamma' :$ **type**,
   $\Gamma \vdash_\Sigma A(\gamma)$,
   – $u = \lambda^0 x : \gamma_1.\{case\ v'\ of\ (c_i\ x_i) \to t'_i \downarrow\ x\}_{i=1}^n$ *avec* $\gamma' = \gamma_1 \multimap \gamma_2$
   – *ou* $u = \lambda x : \gamma_1.\{case\ v'\ of\ (c_i\ x_i) \to t'_i \downarrow\ x\}_{i=1}^n$ *avec* $\gamma' = (\Pi x : \gamma_1)\,\gamma_2$
   – *ou* $u = [l_j = \{case\ v'\ of\ (c_i\ x_i) \to t'_i \downarrow\ l_j\}_{i=1}^n]_{j=1}^m$ *avec* $\gamma' = a$ *et* $binding_\Sigma(a) = [l_1 : \beta_1; \ldots; l_m : \beta_m]$
   et
   $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma \{case\ v\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \;\longrightarrow_c\; \{case\ v'\ of\ (c_i\ x_i) \to t'_i\}_{i=1}^n$
   avec $p(\Pi') < p(\Pi)$.

5. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma \{case\ v\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \;\longrightarrow_c\; u : \gamma$
   alors $\exists a, \alpha_1, \ldots, \alpha_n, t'_1, \cdots, t'_n, v', \Delta_1, \Delta_2$ *tels que :*
   $binding_\Sigma(a) = \{c_1\ of\ \alpha_1 | \ldots | c_n\ of\ \alpha_n\}$,
   $u = \{case\ v'\ of\ (c_i\ x_i) \to t'_i\}_{i=1}^n$,
   $\Delta = \Delta_1, \Delta_2$,
   $\Gamma\,;\,\Delta_1 \vdash_\Sigma v \;\longrightarrow_c\; v' : a$ *et*
   $\forall i \in \{1, \ldots, n\}, \Pi_i : \Gamma\,;\,\Delta_2, x_i : \alpha_i \vdash_\Sigma t_i \;\longrightarrow_\eta t'_i : \gamma$
   avec $p(\Pi_i) < p(\Pi)$.

6. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_\eta u : \gamma$
   et t *est une application, une projection, une constante ou une variable,*
   alors soit $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_c\; u : \gamma$
   avec $p(\Pi') < p(\Pi)$
   soit $\exists \gamma', t'$ *tels que :*
   $\Gamma \vdash_\Sigma \gamma \equiv \gamma' :$ ***type***,
   $\Gamma \vdash_\Sigma A(\gamma')$,
   – $u = \lambda^0 x : \beta_1.(t'\ x)$ *avec* $\gamma' = \gamma_1 \multimap \gamma_2$
   – *ou* $u = \lambda x : \beta_1.(t'\ x)$ *avec* $\gamma' = (\Pi x : \gamma_1)\,\gamma_2$
   – *ou* $u = [l_i = t'.l_i]_{i=1}^n$ *avec* $\gamma' = a$ *et* $binding_\Sigma(a) = [l_1 : \beta_1; \ldots; l_m : \beta_m]$
   – *ou* $u = \{case\ t'\ of\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n$ *avec* $\gamma' = a$ *et* $binding_\Sigma(a) = \{c_1\ of\ \alpha_1 | \ldots | c_n\ of\ \alpha_n\}$
   et
   $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_c\; t' : \gamma'$
   avec $p(\Pi') < p(\Pi)$

7. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t\,u \;\longrightarrow_c\; v : \gamma$
   alors
   (a) *soit* $\exists \Delta_1, \Delta_2, t', u', \alpha$ *tels que*
   $\Delta = \Delta_1, \Delta_2$
   $v = t'\ u'$
   $\Pi_1 : \Gamma\,;\,\Delta_1 \vdash_\Sigma t \;\longrightarrow_c\; t' : \alpha \multimap \gamma$
   $\Pi_2 : \Gamma\,;\,\Delta_2 \vdash_\Sigma u \;\longrightarrow_\eta u' : \alpha$
   avec $p(\Pi_1) < p(\Pi)$ *et* $p(\Pi_2) < p(\Pi)$

   (b) *soit* $\exists t', u', \alpha, \gamma'$ *tels que*
   $\gamma = \gamma'[x := u]$,
   $v = t'\ u'$
   $\Pi_1 : \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_c\; t' : (\Pi x : \alpha)\,gamma'$
   $\Pi_2 : \Gamma\,;\,. \vdash_\Sigma u \;\longrightarrow_\eta u' : \alpha$
   avec $p(\Pi_1) < p(\Pi)$ *et* $p(\Pi_2) < p(\Pi)$

8. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma x \;\longrightarrow_c\; t : \gamma$
   alors $t = x$.

9. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma c \;\longrightarrow_c\; t : \gamma$
   alors $t = c$.

10. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t.l_i \;\longrightarrow_c\; u : \gamma$
    alors $\exists a, \alpha_1, \ldots, \alpha_n, t'$ *tels que :*
    $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$,
    $\Gamma \vdash_\Sigma \gamma \equiv \alpha_i :$ ***type***,
    $u = t'.l_i$ *et*
    $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_c\; t' : a$
    avec $p(\Pi') < p(\Pi)$.

Démonstration. Par récurrence sur la structure de Π, et selon la dernière règle utilisée.

– Si Π termine par une règle de congruence ou (c/η), la récurrence est facile.

– $$\frac{t \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_c\; t' : \alpha \multimap \beta \qquad \Gamma \vdash_\Sigma A(\alpha \multimap \beta)}{\Gamma\,;\,\Delta \vdash_\Sigma t \;\longrightarrow_\eta\; \lambda^0 x : \alpha.(t' \downarrow\; x) : \alpha \multimap \beta}$$

Par le lemme 19 (Cohérence du typage), t ne peut pas être une abstraction non-linéaire ou un enregistrement.
On est donc dans le cas 4 ou 6.
En utilisant l'hypothèse de récurrence, on obtient bien une des possibilités à prouver.

– Les cas des autres règles d'η-expansion sont similaires.

□

3.1.2 Propriétés de substitution

Les lemmes suivants décrivent le comportement de $\downarrow$, $\longrightarrow_\eta$ et $\approx$ lorsqu'une variable libre se voit substituée par un terme. Une propriété de substitution pour $\longrightarrow_\eta$ est en effet un prérequis à la preuve de la commutativité entre $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$, puisqu'une étape de β-réduction provoque une substitution dans le terme considéré.

Le lemme suivant établit que $\downarrow$ est transparent pour $\longrightarrow_{\beta\gamma}$, ce qui n'est pas surprenant puisque $\downarrow$ correspond à des étapes masquées de $\longrightarrow_{\beta\gamma}$.

Lemme 21 (Congruence pour $\downarrow$)**.**
Si $t \;\longrightarrow_{\beta\gamma}\; u$
Alors $t \downarrow\; v \;\longrightarrow^*_{\beta\gamma}\; u \downarrow\; v$
et $t \downarrow l \;\longrightarrow^*_{\beta\gamma}\; u \downarrow l$

Démonstration. Par récurrence sur la taille de t. On traite la première partie du lemme, la preuve de la seconde étant similaire.

– Si $t = \lambda z : \alpha.w$, on a $u = \lambda z : \alpha'.w'$ avec $w \;\longrightarrow_{\beta\gamma}\; w'$.
$t \downarrow\; v = w[z := v]$, $u \downarrow\; v = w'[z := v]$,
on a bien $w[z := v] \;\longrightarrow_\beta\; w'[z := v]$ par substitution.

– Si t est une abstraction linéaire, la preuve est similaire au cas précédent.

– Si t est une analyse de cas, la récurrence est immédiate.

– Sinon, $t \downarrow\; v = t\,v \;\longrightarrow_{\beta\gamma}\; u\,v \;\longrightarrow^*_{\beta\gamma}\; u \downarrow\; v$.

□

Dans le lemme suivant, on établit la propriété de substitution pour $\downarrow$.

Lemme 22 (Substitution pour $\downarrow$)**.**

1. $(t \downarrow\; w)[y := u] \;\longrightarrow^*_{\beta\gamma}\; (t[y := u]) \downarrow\; w[y := u]$

2. $(t \downarrow l)[y := u] \;\longrightarrow^*_{\beta\gamma}\; (t[y := u]) \downarrow l$

Démonstration. Par récurrence sur la taille de t. On traite la première partie du lemme, la preuve de la seconde étant similaire.

– Si $t = \lambda z : \alpha.v$ (z n'apparaissant pas dans u),
$(t \downarrow\; x)[y := u] = (v[z := x])[y := u]$.
$(t[y := u]) \downarrow\; w[y := u] = (v[y := u])[z := w[y := u]]$.
Les deux expressions sont égales.

– Si t est une abstraction linéaire, la preuve est similaire au cas précédent.

– Si $t = \{case\ v \text{ of } (c_i\,x_i) \to t\}_{i=1}^n$, la récurrence est immédiate.

– Sinon $(t \downarrow\; w)[y := u] = t[y := u]\; w[y := u]$,
et on a bien $t[y := u]\; w[y := u] \;\longrightarrow^*_{\beta\gamma}\; (t[y := u]) \downarrow\; (w[y := u])$.

□

Le lemme suivant permet d'inverser les étapes d'expansion. De plus il exprime que l'expansion d'un terme ne modifie pas le résultat en cas de réduction, dans le sens où $\lambda x.(f\ x)$ et f renvoient tous deux $f\ u$ lorsqu'ils sont appliqués à un terme u.

Lemme 23 (Décomposition de $\longrightarrow_\eta$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma$ *et* $\Gamma \vdash_\Sigma \gamma \equiv \alpha \multimap \beta :$ **`type`**

   alors $\exists t''$ *tel que* $\forall u.t' \downarrow u = t'' \downarrow u$ *et* $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t'' : \gamma$
   avec $p(\Pi') < p(\Pi)$

2. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma$ *et* $\Gamma \vdash_\Sigma \gamma \equiv (\Pi x : \alpha)\,\beta :$ **`type`**

   alors $\exists t''$ *tel que* $\forall u.t' \downarrow u = t'' \downarrow u$ *et* $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t'' : \gamma$
   avec $p(\Pi') < p(\Pi)$

3. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma\,,\ \Gamma \vdash_\Sigma \gamma \equiv a :$ **`type`** *et* $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$

   alors $\exists t''$ *tel que* $\forall l.t' \downarrow l = t'' \downarrow l$ *et* $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t'' : \gamma$
   avec $p(\Pi') < p(\Pi)$

4. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma\,,\ \Gamma \vdash_\Sigma \gamma \equiv a :$ **`type`** *et* $binding_\Sigma(a) = \{c_1\ of\ \alpha_1 | \ldots | c_n\ of\ \alpha_n\}$

   alors $\exists t''$ *tel que :*
   $\forall u_1, \ldots, u_n.\{case\ t'\ of\ (c_i\ x) \to u_i\}_{i=1}^n \longrightarrow_{\beta\gamma}^* \{case\ t''\ of\ (c_i\ x) \to u_i\}_{i=1}^n$ *et*
   $\Pi' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \gamma'$
   avec $p(\Pi') < p(\Pi)$

Démonstration. Par récurrence sur la structure de Π. On traite le cas 1, les autres cas étant similaires.
On distingue les différents cas :

– $\dfrac{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \gamma}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma}$ (c / η)

$t'' = t'$ convient.

– $\dfrac{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \alpha \qquad \Gamma \vdash_\Sigma \alpha \equiv \gamma :$ **`type`** $\qquad \Gamma \vdash_\Sigma \gamma :$ **`type`**$}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta t' : \gamma}$ (eq$_\eta$.)

Par hypothèse de récurrence, on a
$\Pi'' : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t'' : \alpha$,
$p(\Pi'') < p(\Pi) - 1$,
$\forall u.t' \downarrow u = t'' \downarrow u$

On peut donc appliquer (eq$_c$.) sur Π'' pour obtenir le résultat.

– Si Π se termine par (weak$_c$), la preuve est similaire au cas précédent.

– La dernière règle de Π est une règle d'η-expansion.
D'après le lemme 18 (homogénéité des types), la seule possibilité est :
t' est donc de la forme $\lambda^0 z : \alpha'.(t'' \downarrow z)$, avec z ni dans Γ ni dans Δ, donc pas dans t'', et
$\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t'' : \gamma$.

On a bien $\forall u$:
$t' \downarrow\ u = (\lambda^0 z : \alpha'.(t'' \downarrow\ z))\ u = t'' \downarrow\ u$

□

Le lemme suivant énonce que le type d'une variable déclarée dans un contexte peut être remplacé par n'importe quel autre type équivalent. Ce lemme entraîne donc notamment que $\approx$ est une relation symétrique, donc une relation d'équivalence.

Lemme 24 (Équivalence du contexte)**.**

1. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2\ ;\ \Delta \vdash_\Sigma t\ \longrightarrow_\eta (resp.\ \longrightarrow_c)\ t' : \beta$
   et $\Gamma_1 \vdash_\Sigma \alpha \equiv \alpha' :$ **type**
   alors $\Gamma_1, x : \alpha', \Gamma_2\ ;\ \Delta \vdash_\Sigma t\ \longrightarrow_\eta (resp.\ \longrightarrow_c)\ t' : \beta$

2. *Si* $\Pi : \Gamma\ ;\ \Delta_1, x : \alpha, \Delta_2 \vdash_\Sigma t\ \longrightarrow_\eta (resp.\ \longrightarrow_c)\ t' : \beta$
   et $\Gamma_1 \vdash_\Sigma \alpha \equiv \alpha' :$ **type**
   alors $\Gamma\ ;\ \Delta_1, x : \alpha', \Delta_2 \vdash_\Sigma t\ \longrightarrow_\eta (resp.\ \longrightarrow_c)\ t' : \beta$

3. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \vdash_\Sigma \beta\ \longrightarrow_\eta (resp.\ \longrightarrow_c) \beta' :$ **K**
   et $\Gamma_1 \vdash_\Sigma \alpha \equiv \alpha' :$ **type**
   alors $\Gamma_1, x : \alpha', \Gamma_2 \vdash_\Sigma \beta\ \longrightarrow_\eta (resp.\ \longrightarrow_c) \beta' :$ **K**

4. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \vdash_\Sigma t \approx t'$
   et $\Gamma_1 \vdash_\Sigma \alpha \equiv \alpha' :$ **type**
   alors $\Gamma_1, x : \alpha', \Gamma_2 \vdash_\Sigma t \approx t'$

5. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \vdash_\Sigma \beta \approx \beta'$
   et $\Gamma_1 \vdash_\Sigma \alpha \equiv \alpha' :$ **type**
   alors $\Gamma_1, x : \alpha', \Gamma_2 \vdash_\Sigma \beta \approx \beta'$

Démonstration. Immédiat par récurrence sur $p(\Pi)$.

□

Le lemme suivant énonce qu'on peut toujours extraire d'un jugement du critère syntaxique un jugement de typage de poids inférieur ou égal.

Lemme 25 (Typage et critère syntaxique)**.**
Si $\Pi : \Gamma \vdash_\Sigma A(\alpha)$
Alors $\Pi' : \Gamma \vdash_\Sigma \alpha :$ **type**
avec $h(\Pi') < h(\Pi)$.

Démonstration. Immédiat par récurrence sur la hauteur de Π.

□

Les deux propriétés suivantes ne sont pas des lemmes, mais des hypothèses. La première énonce que tout type de sorte **type** est dans la classe d'équivalence, pour $\equiv$, d'une forme normale syntaxique.

Hypothèse 1 (Existence des formes normales)**.**
Si $\Gamma \vdash_\Sigma \alpha :$ **type**
Alors $\exists \alpha'$ *tel que* $\Gamma \vdash_\Sigma \alpha \equiv \alpha' :$ **type** *et* $\Gamma \vdash_\Sigma A(\alpha')$.

On admet ce résultat. les résultats suivants sont donc prouvés sous la condition que cette hypothèse est fondée.

La seconde énonce que l'équivalence entre deux types fonctionnels peut être décomposée.

Hypothèse 2 (Décomposition des types fonctionnels)**.**
Si $\Gamma \vdash_\Sigma \alpha \multimap \beta \equiv \alpha' \multimap \beta' :$ **`type`**
ou $\Gamma \vdash_\Sigma (\Pi x : \alpha)\,\beta \equiv (\Pi x : \alpha')\,\beta' :$ **`type`**
Alors $\Gamma \vdash_\Sigma \alpha \equiv \alpha' :$ **`type`**
et $\Gamma \vdash_\Sigma \beta \equiv \beta' :$ **`type`**.
On admet ce résultat. les résultats suivants sont donc prouvés sous la condition que cette hypothèse est fondée.

Ces deux propriétés apparaissent notamment dans la preuve de Barthe (1999a), mais leur preuve est considérablement simplifiée par le fait que $\equiv$ soit à ce stade de la preuve confluente, puisqu'elle peut se reposer, en l'absence de conversions permutatives, sur une η-conversion orientée dans le sens de la réduction.

Le lemme suivant est un lemme intermédiaire permettant d'obtenir la propriété de substitution pour $\longrightarrow_\eta$. Il utilise notamment l'hypothèse de récurrence des lemmes de substitution, mais nous choisissons néanmoins de le séparer du lemme principal pour des raisons de lisibilité. Il énonce basiquement dans quelles conditions un terme $t \downarrow u$ peut être expansé. Ce lemme permet ainsi aux lemmes de substitution suivants de ne pas faire suivre $\longrightarrow_\eta$ par des occurrences de $\longrightarrow_{\beta\gamma}$, ce qui est nécessaire ensuite pour la preuve de l'ajournement de $\longrightarrow_\eta$.

Lemme 26 (Expansions des termes propagés)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta_1 \vdash_\Sigma t \longrightarrow_c t' : \alpha \multimap \beta\,,\ \Gamma\,;\,\Delta_2 \vdash_\Sigma u \longrightarrow_\eta u' : \alpha$ *et si la propriété de substitution linéaire est vérifié pour toute preuve de poids inférieur à celui de* Π
   Alors $\exists w, w'$ *tels que*
   $t \downarrow u \longrightarrow^*_{\beta\gamma} w'$
   $\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma w'' \longrightarrow_\eta w : \beta$
   $\Gamma \vdash_\Sigma t' \downarrow u' \longrightarrow^*_{\beta\gamma\approx} w$

2. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : (\Pi x : \alpha)\,\beta\,,\ \Gamma\,;\,\vdash_\Sigma u \longrightarrow_\eta u' : \alpha$ *et si la propriété de substitution linéaire est vérifié pour toute preuve de poids inférieur à celui de* Π
   Alors $\exists w, w'$ *tels que*
   $t \downarrow u \longrightarrow^*_{\beta\gamma} w'$
   $\Gamma\,;\,\Delta \vdash_\Sigma w' \longrightarrow_\eta w : \beta[x := t]$
   $\Gamma \vdash_\Sigma t' \downarrow u' \longrightarrow^*_{\beta\gamma\approx} w$

3. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : a$ *et* $binding_\Sigma(a) = [l_1 : alpha_1; \ldots; l_n : alpha_n]$
   Alors $\Gamma\,;\,\Delta \vdash_\Sigma t \downarrow l_i \longrightarrow_c t' \downarrow l_i : \alpha_i$

Démonstration. Par récurrence sur la taille de t. On prouve la première partie du lemme, les deux autres étant similaires.

– Si $t = \lambda^0 x : \gamma.v$:

  Par le lemme 20 (inversion), $t' = \lambda^0 x : \gamma'.v'$ et $\Gamma\,;\,\Delta_1, x : \gamma \vdash_\Sigma v \longrightarrow_\eta v' : \beta$

  Par le lemme de substitution linéaire, $\exists w, w'$ tels que :
  $v[x := u] \longrightarrow^*_{\beta\gamma} w'$
  $\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma w' \longrightarrow_\eta w : \beta$
  $\Gamma \vdash_\Sigma v'[x := u'] \longrightarrow^*_{\beta\gamma\approx} w.$

  On a par ailleurs $t \downarrow u = v[x := u]$ et $t' \downarrow u' = v'[x := u']$.

– Par le lemme 19 (cohérence du typage), t ne peut pas être une abstraction non linéaire.

– Si $t = \{case\ v \text{ of } (c_i\ x_i) \to t\}_{i=1}^n$:

  Par le lemme 20 (inversion),

$t' = \{case\ v' \text{ of } (c_i\ x_i) \to t'\}_{i=1}^n$,
$\Gamma\ ;\ \Delta_1^1 \vdash_\Sigma v \ \longrightarrow_c \ v' : a$,
$binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \dots | c_n \text{ of } \alpha_n\}$,
$\forall i \in \{1, \cdots, n\}, \Gamma\ ;\ \Delta_1^2, x_i : \alpha_i \vdash_\Sigma t_i \ \longrightarrow_\eta t_i' : \alpha \multimap \beta$,
$\Delta_1 = \Delta_1^1, \Delta_1^2$.

Par le lemme 23 (décomposition de $\longrightarrow_\eta$), on a :
$\forall i \in \{1, \cdots, n\}, \exists t_i''$ tel que $\Gamma\ ;\ \Delta_1^2, x_i : \alpha_i \vdash_\Sigma t_i \ \longrightarrow_c \ t_i'' : \alpha \multimap \beta$ et $t_i' \downarrow \ u' = t_i'' \downarrow \ u'$.

$\forall i \in \{1, \dots, n\}$, par hypothèse de récurrence, $\exists w_i, w_i'$ tels que :
$t_i \downarrow \ u \ \longrightarrow_{\beta\gamma}^* \ w_i'$
$\Gamma\ ;\ \Delta_1^2, x_i : \alpha_i, \Delta_2 \vdash_\Sigma w_i' \ \longrightarrow_\eta w_i : \alpha \multimap \beta$
$\Gamma \vdash_\Sigma t_i'' \downarrow \ u' \ \longrightarrow_{\beta\gamma\approx}^* \ w_i$.

On a bien :
$t \downarrow \ u = \{case\ v \text{ of } (c_i\ x) \to t_i \downarrow \ u\}_{i=1}^n \ \longrightarrow_{\beta\gamma}^* \ \{case\ v \text{ of } (c_i\ x) \to w_i'\}_{i=1}^n$
$\Gamma\ ;\ \Delta_1, \Delta_2 \vdash_\Sigma \{case\ v \text{ of } (c_i\ x_i) \to w_i'\}_{i=1}^n \ \longrightarrow_\eta \{case\ v' \text{ of } (c_i\ x_i) \to w_i\}_{i=1}^n : \beta$,
$\Gamma \vdash_\Sigma t' \downarrow \ u' = \{case\ v' \text{ of } (c_i\ x) \to t_i'' \downarrow \ u\}_{i=1}^n \ \longrightarrow_{\beta\gamma\approx}^* \ \{case\ v' \text{ of } (c_i\ x) \to w_i\}_{i=1}^n$.

– Sinon, t n'est ni une abstraction, ni une analyse de cas, et par le lemme 20 (inversion), c'est également le cas pour t'.
On a bien $t \downarrow \ u = t\ u$ et $t' \downarrow \ u' = t'\ u'$,
et $\Gamma\ ;\ \Delta_1, \Delta_2 \vdash_\Sigma t\ u \ \longrightarrow_\eta t'\ u' : \beta$.

□

Le lemme suivant constitue le coeur de cette section, et énonce la propriété de substitution pour $\longrightarrow_\eta$ lorsque la variable substituée apparait dans le contexte non-linéaire. Comme les règles (eq.) nécessitent d'opérer une substitution dans un jugement d'équivalence, il est également nécessaire de prouver en parallèle une propriété de substitution pour $\approx$.

Lemme 27 (Substitution d'une variable non-linéaire pour $\longrightarrow_\eta$)**.**

1. *Si* $\Pi : \Gamma, x : \alpha, \Gamma'\ ;\ \Delta \vdash_\Sigma u \ \longrightarrow_\eta (ou \ \longrightarrow_c) \ u' : \beta$ *et* $\Gamma\ ;\ . \vdash_\Sigma t \ \longrightarrow_\eta t' : \alpha$
alors $\exists u^\beta, u^\eta$ *tels que :*
$u[x := t] \ \longrightarrow_{\beta\gamma}^* \ u^\beta$,
$\Gamma, \Gamma'[x := t]\ ;\ \Delta[x := t] \vdash_\Sigma u^\beta \ \longrightarrow_\eta u^\eta : \beta[x := t]$,
$\Gamma, \Gamma'[x := t] \vdash_\Sigma u'[x := t'] \ \longrightarrow_{\beta\gamma\approx}^* \ u^\eta$

2. *Si* $\Pi : \Gamma, x : \alpha, \Gamma' \vdash_\Sigma \beta \ \longrightarrow_\eta (resp. \ \longrightarrow_c) \ \beta' : K$ *et* $\Gamma\ ;\ . \vdash_\Sigma t \ \longrightarrow_\eta t' : \alpha$
alors $\exists \beta^\beta, \beta^\eta$ *tels que :*
$\beta[x := t] \ \longrightarrow_{\beta\gamma}^* \ \beta^\beta$,
$\Gamma, \Gamma' \vdash_\Sigma \beta''' \ \longrightarrow_\eta (resp. \ \longrightarrow_c) \ \beta^\eta : K$,
$\Gamma, \Gamma' \vdash_\Sigma \beta'[x := t'] \ \longrightarrow_{\beta\gamma\approx}^* \ \beta^\eta$

3. *Si* $\Pi : \Gamma, x : \alpha, \Gamma'\ ;\ \Delta \vdash_\Sigma u \approx u'$ *et* $\Gamma\ ;\ . \vdash_\Sigma t : \alpha$
Alors $\Gamma, \Gamma'[x := t] \vdash_\Sigma u[x := t] \approx u'[x := t]$

4. *Si* $\Pi : \Gamma, x : \alpha, \Gamma' \vdash_\Sigma \beta \approx \beta'$ *et* $\Gamma\ ;\ . \vdash_\Sigma t : \alpha$
alors $\Gamma, \Gamma' \vdash_\Sigma \beta[x := t] \approx \beta'[x := t]$.

Démonstration. Par récurrence sur $p(\Pi)$, en distinguant selon la dernière règle utilisée.

– Si c'est la règle (c / η),la récurrence est immédiate.

– $\dfrac{\vdash_\Sigma \gamma : \mathbf{type} \qquad \Gamma \vdash_\Sigma \beta \ \longrightarrow_c \ \beta' : \mathrm{K}}{\Gamma, y : \gamma \vdash_\Sigma \beta \ \longrightarrow_c \ \beta' : \mathrm{K}}$ (type weak.)

Si la variable à remplacer est dans Γ, la récurrence est immédiate. Si la variable à remplacer est y, on sait par le lemme 15 (variables libres) que ni β ni β' ne contiennent y libre, et donc que $\beta[y := t] = \beta$ et $\beta'[y := t'] = \beta'$. On tire donc de la prémisse droite et de (c / η) ($\Gamma \vdash_\Sigma \beta[y := t] \ \longrightarrow_\eta \beta' : \mathrm{K}$.

– (type abs.), (lin. fun.), (dep. prod.) : immédiat

– $\dfrac{\Gamma, x : \alpha, \Gamma' \vdash_\Sigma \gamma \ \longrightarrow_c \ \gamma' : \beta) \, \mathrm{K} \qquad \Gamma, x : \alpha, \Gamma'; \vdash_\Sigma u \ \longrightarrow_\eta u' : \beta}{\Gamma, x : \alpha, \Gamma' \vdash_\Sigma \gamma \, u \ \longrightarrow_c \ \gamma' u' : \mathrm{K}}$ (type app.)

Par le lemme 15 (variables libres), on a $x \notin \mathrm{FV}(\Gamma')$, donc $\Gamma'[x := t] = \Gamma'$, et $x \notin \mathrm{FV}(\beta)$, donc $\beta[x := t] = \beta$. La récurrence est donc immédiate.

– $\dfrac{\Gamma, x : \alpha, \Gamma' \vdash_\Sigma \gamma : \mathbf{type}}{\Gamma, x : \alpha, \Gamma' \, ; \, y : \gamma \vdash_\Sigma y \ \longrightarrow_c \ y : \gamma}$ (lin. var.)

par le lemme 15 (variables libres), on sait que $\Gamma, x : \alpha, \Gamma'$ ne contient aucune variable libre, en particulier x.
Par récurrence, on a donc :
$\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma[x := t] : \mathbf{type}$
Par (lin. var.) et (c / η), on a donc :
$\Gamma, \Gamma'[x := t] \, ; \, y : \gamma[x := t] \vdash_\Sigma y \ \longrightarrow_\eta y : \gamma[x := t]$

– $\dfrac{\Gamma \vdash_\Sigma \gamma : \mathbf{type}}{\Gamma, x : \gamma \, ; \vdash_\Sigma x \ \longrightarrow_c \ x : \gamma}$ (var.)

Si x n'est pas la variable à remplacer, la preuve est similaire au cas précédent. Sinon, par le lemme 15 (variables libres), $x \notin \mathrm{FV}(\alpha)$). Par hypothèse, on a donc bien $\Gamma \, ; \, . \vdash_\Sigma t \ \longrightarrow_\eta t' : \alpha[x := t]$.

– $\dfrac{\Gamma \vdash_\Sigma \gamma : \mathbf{type} \qquad \Gamma \, ; \, \Delta \vdash_\Sigma u \ \longrightarrow_c \ u' : \beta}{\Gamma, x : \gamma \, ; \, \Delta \vdash_\Sigma u \ \longrightarrow_c \ u' : \beta}$ (weak$_c$.)

Si la variable à remplacer est dans Γ, la récurrence est immédiate.
Si la variable à remplacer est x, par le lemme 15 (variables libres) appliqué à la prémisse droite, x n'apparait ni dans Δ, ni dans u, ni dans u', ni dans β. Par application de (c / η) sur la prémisse droite, on a donc bien $\Gamma \, ; \, \Delta[x := t] \vdash_\Sigma u[x := t] \ \longrightarrow_c \ u'[x := t'] : \beta[x := t]$

– (weak$_\eta$.) : Similaire au cas précédent.

– $\dfrac{\Gamma, x : \alpha, \Gamma' \, ; \, \Delta, y : \gamma \vdash_\Sigma u \ \longrightarrow_\eta u' : \beta \qquad \mathit{Gamma}, x : \alpha, \Gamma' \vdash_\Sigma \gamma \ \longrightarrow_\eta \gamma' : \mathbf{type}}{\Gamma, x : \alpha, \Gamma' \, ; \, \Delta \vdash_\Sigma \lambda^0 y : \gamma . u \ \longrightarrow_c \ \lambda^0 y : \gamma' . u' : \gamma \multimap \beta}$ (lin. abs.)

Par hypothèse de récurrence, on a :
$u[x := t] \ \longrightarrow^*_{\beta\gamma} \ u^\beta$
$\Gamma, \Gamma'[x := t] \, ; \, \Delta[x := t], y : \gamma[x := t] \vdash_\Sigma u^\beta \ \longrightarrow_\eta u^\eta : \beta[x := t]$
$\Gamma, \Gamma'[x := t] \vdash_\Sigma u'[x := t'] \ \longrightarrow^*_{\beta\gamma\approx} \ u^\eta$

$\gamma[x := t] \ \longrightarrow^*_{\beta\gamma} \ \gamma^\beta$
$\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma^\beta \ \longrightarrow_\eta \gamma^\eta : \mathbf{type}$
$\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma'[x := t'] \ \longrightarrow^*_{\beta\gamma\approx} \ \gamma^\eta$

On a bien
$(\lambda^0 y : \gamma . u)[x := t] \ \longrightarrow^*_{\beta\gamma} \ \lambda^0 y : \gamma^\beta . u^\beta$
$\Gamma, \Gamma'[x := t] \, ; \, \Delta[x := t] \vdash_\Sigma \lambda^0 y : \gamma^\beta . u^\beta \ \longrightarrow_\eta \lambda^0 y : \gamma^\eta . u^\eta : (\gamma \multimap \beta)[x := t]$ (en utilisant le lemme 24 (équivalence du contexte) et les règles (lin. var.), (eq$_c$) et (c / η))
$\Gamma, \Gamma'[x := t] \vdash_\Sigma \lambda^0 y : \gamma[x := t'] . u'[x := t'] \ \longrightarrow^*_{\beta\gamma\approx} \ \lambda^0 y : \gamma[x := t'] . u^\eta \approx \lambda^0 y : \gamma^\eta . u^\eta$.

– (abs.) : similaire à (lin. abs).

– $\dfrac{\Gamma, x:\alpha, \Gamma' ;\ \Delta \vdash_\Sigma v \ \longrightarrow_c \ v' : (\Pi y:\gamma)\,\beta \qquad \Gamma, x:\alpha, \Gamma' ;\ \vdash_\Sigma u \ \longrightarrow_{\eta,\gamma} u'}{\Gamma, x:\alpha, \Gamma' ;\ \Delta \vdash_\Sigma v\,u \ \longrightarrow_c \ v'\,u' : \beta[y:=u]}$ (app.)

Par hypothèse de récurrence, on a :

$v[x:=t] \ \longrightarrow^*_{\beta\gamma} \ v^\beta$
$\Gamma, \Gamma'[x:=t] ;\ \Delta[x:=t] \vdash_\Sigma v^\beta \ \longrightarrow_\eta v^\eta : (\Pi y:\gamma[x:=t])\,\beta[x:=t]$
$\Gamma, \Gamma'[x:=t] \vdash_\Sigma v'[x:=t'] \ \longrightarrow^*_{\beta\gamma\approx} \ v^\eta$

$u[x:=t] \ \longrightarrow^*_{\beta\gamma} \ u^\beta$
$\Gamma, \Gamma'[x:=t] ;\ \vdash_\Sigma u^\beta \ \longrightarrow_\eta u^\eta : \gamma[x:=t]$
$\Gamma, \Gamma'[x:=t] \vdash_\Sigma u'[x:=t'] \ \longrightarrow^*_{\beta\gamma\approx} \ u^\eta$

On applique le lemme 23 (décomposition de $\longrightarrow_\eta$) pour obtenir :
$\Gamma, \Gamma'[x:=t] ;\ \Delta[x:=t] \vdash_\Sigma v^\beta \ \longrightarrow_c \ v^c : (\Pi y:\gamma[x:=t])\,\beta[x:=t]$
$v^\eta \downarrow \ u^\eta = v^c \downarrow \ u^\eta$

Par le lemme 26 (expansion des termes propagés) et la règle (c / η), on a :
$v^\beta \downarrow \ u^\beta \ \longrightarrow^*_{\beta\gamma} \ w'$
$\Gamma, \Gamma'[x:=t] ;\ \Delta[x:=t] \vdash_\Sigma w' \ \longrightarrow_\eta w : \beta[x:=t][y:=u^\beta]$
$\Gamma, \Gamma'[x:=t] \vdash_\Sigma v^c \downarrow \ u^\eta \ \longrightarrow^*_{\beta\gamma\approx} \ w$

Par les lemmes 7 et 11 (substitution pour $\longrightarrow_\beta$ et $\longrightarrow_\gamma$), on a :
$\beta[y:=u][x:=t] = \beta[x:=t][y:=u[x:=t]] \ \longrightarrow^*_{\beta\gamma} \ \beta[x:=t][y:=u^\beta]$

On a donc :
$(v\,u)[x:=t] \ \longrightarrow^*_{\beta\gamma} \ v^\beta\,u^\beta \ \longrightarrow^*_{\beta\gamma} \ v^\beta \downarrow \ u^\beta \ \longrightarrow^*_{\beta\gamma} \ w'$,
$\Gamma, \Gamma'[x:=t] ;\ \Delta[x:=t] \vdash_\Sigma w' \ \longrightarrow_\eta w : \beta[y:=u][x:=t]$ (par la règle (eq$_\eta$.),
et $\Gamma, \Gamma'[x:=t] \vdash_\Sigma (v'\,u')[x:=t'] \ \longrightarrow^*_{\beta\gamma\approx} \ v^\eta\,u^\eta \ \longrightarrow^*_{\beta\gamma} \ v^\eta \downarrow \ u^\eta = v^c \downarrow \ u^\eta \ \longrightarrow^*_{\beta\gamma\approx} \ w$

– (lin. app.), (sel.), (case) : similaire à (app.)

– (rec.) : immédiat

– $\dfrac{\Gamma, x:\alpha, \Gamma' ;\ \Delta \vdash_\Sigma u \ \longrightarrow_c \ u' : \gamma \qquad \Gamma, x:\alpha, \Gamma' \vdash_\Sigma \gamma \equiv \beta : \mathbf{type} \qquad \Gamma, x:\alpha, \Gamma' \vdash_\Sigma \beta : \mathbf{type}}{\Gamma, x:\alpha, \Gamma' ;\ \Delta \vdash_\Sigma u \ \longrightarrow_c \ u' : \beta}$ (eq$_c$.)

$\exists \gamma_1, \ldots, \gamma_n$ tels que $\gamma_1 = \gamma, \gamma_n = \beta$ et $\forall i \in \{1, \ldots, n-1\}$, on a :
$\gamma_i \ \longrightarrow_{\beta\gamma} \ \gamma_{i+1}$ ou $\gamma_{i+1} \ \longrightarrow_{\beta\gamma} \ \gamma_i$
ou $\Gamma, x:\alpha, \Gamma' \vdash_\Sigma \gamma_i \ \longrightarrow_\eta \gamma_{i+1} : \mathbf{type}$ ou $\Gamma, x:\alpha, \Gamma' \vdash_\Sigma \gamma_{i+1} \ \longrightarrow_\eta \gamma_i : \mathbf{type}$ ou $\Gamma, x:\alpha, \Gamma' \vdash_\Sigma \gamma_i \approx \gamma_{i+1}$
ou $\Gamma, x:\alpha, \Gamma' \vdash_\Sigma \gamma_{i+1} \approx \gamma_i$.

Par les lemme 7 et 11 (substitution pour $\longrightarrow_\beta$ et $\longrightarrow_\gamma$) et par l'hypothèse de récurrence, on a donc $\Gamma, \Gamma' \vdash_\Sigma \gamma[x:=t] \equiv \beta[x:=t] : \mathbf{type}$

Par hypothèse de récurrence, on a par ailleurs :
$u'[x:=t'] \ \longrightarrow^*_{\beta\gamma} \ u^\beta$,
$\Gamma, \Gamma'[x:=t] ;\ \Delta[x:=t] \vdash_\Sigma u^\beta \ \longrightarrow_\eta u^\eta : \gamma[x:=t]$
et $\Gamma, \Gamma'[x:=t] \vdash_\Sigma u[x:=t] \ \longrightarrow^*_{\beta\gamma\approx} \ u^\eta$.

On peut utiliser la règle (eq$_\eta$.) pour obtenir :

$\Gamma, \Gamma'[x := t]\,;\, \Delta[x := t] \vdash_\Sigma u^\beta \;\longrightarrow_\eta u^\eta : \beta[x := t]$

– (eq_η.) : identique à (eq_c.).

– $$\frac{u \text{ n'est pas une abstraction linéaire} \qquad \Gamma, x : \alpha, \Gamma'\,;\, \Delta \vdash_\Sigma u \;\longrightarrow_c u' : \gamma \multimap \beta \qquad \Gamma, x : \alpha, \Gamma' \vdash_\Sigma A(\gamma \multimap \beta)}{\Gamma, x : \alpha, \Gamma'\,;\, \Delta \vdash_\Sigma u \;\longrightarrow_\eta \lambda^0 y : \gamma.(u' \downarrow y) : \gamma \multimap \beta}$$

Par hypothèse de récurrence :
$u[x := t] \;\longrightarrow^*_{\beta\gamma} u^\beta$
$\Gamma, \Gamma'[x := t]\,;\, \Delta[x := t] \vdash_\Sigma u^\beta \;\longrightarrow_\eta u^\eta : (\gamma \multimap \beta)[x := t]$
$\Gamma, \Gamma'[x := t] \vdash_\Sigma u'[x := t'] \;\longrightarrow^*_{\beta\gamma\approx} u^\eta$

1. Si $u^\beta = \lambda^0 z : \gamma'.v$:
   Par le lemme 20 (inversion), on a
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma' \multimap \beta' \equiv (\gamma \multimap \beta)[x := t] : \mathbf{type}$
   $u^\eta = \lambda^0 z : \gamma''.v'$
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma' \;\longrightarrow_\eta \gamma'' : \mathbf{type}$.

   Par les lemmes 25 (typage et critère syntaxique), 16 (typage et $\longrightarrow_\eta$) et l'hypothèse de récurrence, on a :
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma (\gamma \multimap \beta)[x := t] \equiv (\gamma \multimap \beta)[x := t'] : \mathbf{type}$.

   Par l'hypothèse 2 (décomposition des types fonctionnels), on a :
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma'' \equiv \gamma[x := t'] : \mathbf{type}$

   On a bien :
   $u[x := t] \;\longrightarrow^*_{\beta\gamma} u^\beta$
   $\Gamma, \Gamma'[x := t]\,;\, \Delta[x := t] \vdash_\Sigma u^\beta \;\longrightarrow_\eta u^\eta : (\gamma \multimap \beta)[x := t]$
   et, notamment par le lemme 21 (congruence de $\downarrow$) :
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma (\lambda^0 y : \gamma.(u' \downarrow y))[x := t'] \;\longrightarrow^*_{\beta\gamma} \lambda^0 y : \gamma[x := t'].(u'[x := t'] \downarrow y) \;\longrightarrow^*_{\beta\gamma\approx}$
   $\lambda^0 y : \gamma[x := t'].(u^\eta \downarrow y) = \lambda^0 z : \gamma[x := t'].v' \approx u^\eta$.

2. Si u^β n'est pas une abstraction linéaire, d'après l'hypothèse 1 (existence des formes normales), $\exists \gamma', \beta'$ tels que
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma' \equiv_{type} \gamma[x := t]$
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma \beta' \equiv_{type} \beta[x := t]$
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma A(\gamma')$
   $\Gamma, \Gamma'[x := t] \vdash_\Sigma A(\beta')$

   On a donc notamment $\Gamma, \Gamma'[x := t] \vdash_\Sigma A(\gamma' \multimap \beta')$

   Par le lemme 23 (décomposition de $\longrightarrow_\eta$), on a :
   $\Gamma, \Gamma'[x := t]\,;\, \Delta[x := t] \vdash_\Sigma u^\beta \;\longrightarrow_c u^c : (\gamma \multimap \beta)[x := t]$
   $u^\eta \downarrow y = u^c \downarrow y$

   On a donc par (eq_c .) et η-expansion :
   $\Gamma, \Gamma'[x := t]\,;\, \Delta[x := t] \vdash_\Sigma u^\beta \;\longrightarrow_\eta \lambda^0 y : \gamma'(u^c \downarrow y) : \gamma' \multimap \beta'$.

   Par les lemmes 25 (typage et critère syntaxique), 16 (typage et $\longrightarrow_\eta$) et l'hypothèse de récurrence, on a :

$\Gamma, \Gamma'[x := t] \vdash_\Sigma (\gamma \multimap \beta)[x := t] \equiv (\gamma \multimap \beta)[x := t'] : \mathtt{type}$.

Par l'hypothèse 2 (décomposition des types fonctionnels), on a :
$\Gamma, \Gamma'[x := t] \vdash_\Sigma \gamma' \equiv \gamma[x := t'] : \mathtt{type}$

On a bien :
$u[x := t] \longrightarrow^*_{\beta\gamma} u^\beta$
et, notamment par le lemme 21 (congruence de $\downarrow$) :
$\Gamma, \Gamma'[x := t] \vdash_\Sigma (\lambda^0 y : \gamma.(u'\ y))[x := t'] \longrightarrow^*_{\beta\gamma\approx} \lambda^0 y : \gamma[x := t'].(u^\eta \downarrow\ y) \approx \lambda^0 y : \gamma'.(u^c \downarrow\ y)$.

– Si Π se termine par une autre règle d'expansion, la preuve est similaire au cas précédent.

– Si Π se termine par une règle de $\approx$, la récurrence est immédiate.

□

Le lemme suivant énonce une propriété similaire, mais pour les variables apparaissant dans le contexte linéaire.

Lemme 28 (Substitution d'une variable linéaire)**.**
Si $\Pi : \Gamma\ ;\ \Delta, x : \alpha, \Delta' \vdash_\Sigma u \longrightarrow_\eta (ou \longrightarrow_c) u' : \beta$
et $\Gamma\ ;\ \Delta'' \vdash_\Sigma t \longrightarrow_\eta t' : \alpha$
alors $\exists u^\beta, u^\eta$ *tels que*
$u[x := t] \longrightarrow^*_{\beta\gamma} u^\beta$,
$\Gamma\ ;\ \Delta, \Delta', \Delta'' \vdash_\Sigma u^\beta \longrightarrow_{\eta,\beta} u^\eta$,
et $\Gamma \vdash_\Sigma u'[x := t'] \longrightarrow^*_{\beta\gamma\approx} u^\eta$.

Démonstration. similaire à la substitution non-linéaire.

□

Le lemme suivant complète la propriété de substitution pour $\approx$.

Lemme 29 (Substitution pour $\approx$)**.**

1. *Si* $\Pi : \Gamma, x : \alpha, \Gamma' \vdash_\Sigma u : \beta$, $\Gamma\ ;\ . \vdash_\Sigma t : \alpha$ *et* $\Gamma \vdash_\Sigma t \approx t'$
   alors $\Gamma, \Gamma'[x := t] \vdash_\Sigma u[x := t] \approx u[x := t']$.

2. *Si* $\Pi : \Gamma, x : \alpha, \Gamma' \vdash_\Sigma \beta : K$, $\Gamma\ ;\ . \vdash_\Sigma t : \alpha$ *et* $\Gamma\ ;\ . \vdash_\Sigma t \approx t'$
   alors $\Gamma, \Gamma' \vdash_\Sigma \beta[x := t] \approx \beta[x := t']$.

3. *Si* $\Gamma \vdash_\Sigma t \approx t'$, $\Gamma \vdash_\Sigma u \approx u'$ *et* $x \notin dom(\Gamma)$
   alors $\Gamma \vdash_\Sigma t[x := u] \approx t'[x := u']$.

Démonstration. Immédiat par récurrence sur la structure de Π.

□

3.1.3 Propriétés de commutation

À présent que nous disposons d'une propriété de substitution sur $\longrightarrow_\eta$, nous sommes en mesure de démontrer la propriété de commutativité de $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$. Nous ncommençons d'abord par établir quelques lemmes nécessaires, notamment les propriété de réduction du sujet assurant que chaque étape de réécriture conserve le typage, et la cohérence locale forte de $\approx$ avec $\longrightarrow_\eta$ ou $\longrightarrow_{\beta\gamma}$, nous assurant que cette relation d'équivalence introduite dans les lemmes de substitution ne perturbe pas la commutativité de $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$.

Le lemme suivant énonce que seuls les types d'une même classe d'équivalence pour $\equiv$ peuvent être attribués à un terme donné dans un contexte donné. Il s'agit donc du pendant de la propriété d'unicité du type du lambda calcul simplement typé à la Church, tenant compte de la présence dans notre système de la règle (eq.).

Lemme 30 (Equivalence du typage)**.**
Si $\Pi_1 : \Gamma \vdash_\Sigma t : \alpha$ *et* $\Pi_1 : \Gamma \vdash_\Sigma t : \beta$*,*
Alors $\Gamma \vdash_\Sigma \alpha \equiv \beta$

Démonstration. Immédiat par récurrence sur la somme des hauteurs de Π_1 et Π_2.

□

Le lemme suivant correspond à l'inverse du lemme d'affaiblissement précédemment établit. Il énonce que si une variable du contexte non-linéaire n'est jamais utilisée dans une dérivation, alors elle peut en être supprimée.

Lemme 31 (Renforcement)**.**

1. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \,;\, \Delta \vdash_\Sigma t \longrightarrow_\eta (resp. \longrightarrow_c) t' : \beta$ *et* x *n'apparait ni dans* Δ*, ni dans* Γ_2*, ni dans* t*, ni dans* β*,*
   alors $\Gamma_1, \Gamma_2 \,;\, \Delta \vdash_\Sigma t \longrightarrow_\eta (resp. \longrightarrow_c) t' : \beta$*.*

2. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \vdash_\Sigma \beta \longrightarrow_\eta (resp. \longrightarrow_c) \beta' : K$ *et* x *n'apparait pas dans* β*,*
   alors $\Pi : \Gamma_1, \Gamma_2 \vdash_\Sigma \beta \longrightarrow_\eta (resp. \longrightarrow_c) \beta' : K$

3. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \,;\, \Delta \vdash_\Sigma t \approx t'$ *et* x *n'apparait ni dans* Δ*, ni dans* Γ_2*, ni dans* t*,*
   alors $\Gamma_1, \Gamma_2 \,;\, \Delta \vdash_\Sigma t \approx t'$*.*

4. *Si* $\Pi : \Gamma_1, x : \alpha, \Gamma_2 \vdash_\Sigma \beta \approx \beta'$ *et* x *n'apparait pas dans* β*,*
   alors $\Pi : \Gamma_1, \Gamma_2 \vdash_\Sigma \beta \approx \beta'$

Démonstration. Similaire à la preuve du lemme 27 (subtitution non-linéaire).

□

Les deux lemmes suivants énoncent les propriétés de réduction du sujet, c'est-à-dire de conservation du type par les relations du calcul. Nous commençons par établir la réduction du sujet de $\longrightarrow_{\beta\gamma}$, qui permet de récupérer la préservation du type par l'occurrence de $\downarrow$ présente dans les règles d'expansion.

Lemme 32 (Réduction du sujet pour $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $t \longrightarrow_{\beta\gamma} t'$ *et* $\Pi : \Gamma \,;\, \Delta \vdash_\Sigma t : \alpha$*,*
   alors $\Gamma \,;\, \Delta \vdash_\Sigma t' : \alpha$*.*

2. *Si* $\alpha \longrightarrow_{\beta\gamma} \alpha'$ *et* $\Pi : \Gamma \vdash_\Sigma \alpha : K$*,*
   alors $\Gamma \vdash_\Sigma \alpha' : K$*.*

Démonstration. Par récurrence sur la structure de Π, selon la dernière règle utilisée :

– (const.), (lin. var.), (var.), (weak.) : immédiat

– $\dfrac{\Gamma \,;\, \Delta, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \,;\, \Delta \vdash_\Sigma \lambda^0 x : \alpha . t : \alpha \multimap \beta}$ (lin. abs.)

La règle de réduction ne peut être que la congruence :

$$\frac{t \longrightarrow_\beta t' \qquad \alpha \longrightarrow_\beta \alpha'}{\lambda^0 x : \alpha . t \longrightarrow_\beta \lambda^0 x : \alpha' . t'}$$

Par hypothèse de récurrence on a donc $\Gamma \,;\, \Delta, x : \alpha \vdash_\Sigma t' : \beta$.

Par le lemme 24 (équivalence du contexte), on a $\Gamma \,;\, \Delta, x : \alpha' \vdash_\Sigma t' : \beta$.

Il suffit donc d'une application de (lin. abs.) et de (eq.) pour obtenir :
$\Gamma \,;\, \Delta \vdash_\Sigma \lambda^0 x : \alpha' . t' : \alpha \multimap \beta$

– $\dfrac{\Gamma\,;\,\Delta_1 \vdash_\Sigma t : \alpha \multimap \beta \qquad \Gamma\,;\,\Delta_2 \vdash_\Sigma u : \alpha}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma t\,u : \beta}$ (lin. app.)

1. la règle de réduction est une congruence. la preuve est alors similaire au premier cas de (lin. abs.).

2. la règle de réduction est une β-réduction :

$$\frac{v \longrightarrow_\beta v' \qquad u \longrightarrow_\beta u'}{(\lambda^0 x : \gamma.v)\,u \longrightarrow_\beta v'[x := u']}$$

avec $t = \lambda^0 x : \gamma.v$.

Par hypothèse de récurrence, on a :
$\Gamma\,;\,\Delta_2 \vdash_\Sigma u' : \alpha$,
et, en inférant $\lambda^0 x : \gamma.v \longrightarrow_\beta \lambda^0 x : \gamma.v'$ à partir de $v \longrightarrow_\beta v'$ (congruence), on a également :
$\Gamma\,;\,\Delta_1 \vdash_\Sigma \lambda^0 x : \gamma.v' : \alpha \multimap \beta$.

Le lemme 20 (inversion) sur ce dernier séquent permet d'obtenir :
$\exists \gamma'$ tel que $\Gamma \vdash_\Sigma \alpha \multimap \beta \equiv_{type} \gamma \multimap \gamma'$ et $\Gamma\,;\,\Delta_1, x : \gamma \vdash_\Sigma v' : \beta$.

Par l'hypothèse 2 (lemme clé), on a $\Gamma \vdash_\Sigma \alpha \equiv_{type} \gamma$.

Par le lemme 24 (equivalence du contexte), on a donc $\Gamma\,;\,\Delta_1, x : \alpha \vdash_\Sigma v' : \beta$.

On peut alors utiliser le lemme 28 (substitution d'une variable linéaire) pour obtenir :
$\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma v'[x := u'] : \beta$.

3. la règle de réduction est une β-réduction :

$$\frac{v \longrightarrow_\beta v' \qquad u \longrightarrow_\beta u'}{(\lambda x : \gamma.v)\,u \longrightarrow_\beta v'[x := u']}$$

avec $t = \lambda x.v$.

Par hypothèse de récurrence, en inférant $\lambda x.v \longrightarrow_\beta \lambda x : \gamma.v'$ à partir de $v \longrightarrow_\beta v'$ (congruence), on a :
$\Gamma\,;\,\Delta_1 \vdash_\Sigma \lambda x : \gamma.v' : \alpha \multimap \beta$.

Le lemme 20 (inversion) sur ce dernier séquent permet d'obtenir :
$\exists \gamma'$ tel que $\Gamma \vdash_\Sigma \alpha \multimap \beta \equiv_{type} (\Pi x : \gamma)\,\gamma'$.

Le lemme 19 (cohérence du typage) permet donc d'exclure ce cas.

4. la règle de réduction est une conversion permutative :

$$\frac{}{\{case\ v \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n\,u \longrightarrow_\gamma \{case\ v \text{ of } (c_i\,x_i) \to t_i\,u\}_{i=1}^n}$$

avec $t = \{case\ v \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n$.

De $\Gamma\,;\,\Delta_1 \vdash_\Sigma \{case\ v \text{ of } (c_i\,x_i) \to t_i\}_{i=1}^n : \alpha \multimap \beta$,
on tire par le lemme 20 (inversion) :

$\exists a, \Delta_1', \Delta_1''$ tels que :
$\Delta_1', \Delta_1'' = \Delta_1$,
$binding_\Sigma(a) = \{c_1 \text{ of } \alpha_1 | \dots | c_n \text{ of } \alpha_n\}$,
$\Gamma\,;\,\Delta_1'' \vdash_\Sigma v : a$ et
$\forall i \in \{1, \cdots, n\}, \Gamma\,;\,\Delta_1', x_i : \alpha_i \vdash_\Sigma t_i : \alpha \multimap \beta$.

On peut donc utiliser (lin. app.) pour obtenir :
$\forall i \in \{1, \cdots, n\}, \Gamma\,;\,\Delta_1', x_i : \alpha_i, \Delta_2 \vdash_\Sigma t_i\, u : \beta$.

Par (case), on obtient bien :
$\Gamma\,;\,\Delta \vdash_\Sigma \{case\ v \text{ of } (c_i\, x) \to t_i\, u\}_{i=1} : \beta$.

– (abs), (app.), (rec.), (sel.), (case), (inj) : similaire aux cas précédents.

– (eq.) : immédiat

□

Lemme 33 (Réduction du sujet pour $\longrightarrow_\eta$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (ou \longrightarrow_c) t' : \alpha$,
   alors $\Gamma\,;\,\Delta \vdash_\Sigma t' : \alpha$.

2. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta (ou \longrightarrow_c) \alpha' : K$,
   alors $\Gamma \vdash_\Sigma \alpha' : K$.

Démonstration. Par récurrence sur la structure de Π, selon la dernière règle utilisée :

– Si la dernière règle est une congruence, la preuve est similaire à celle du lemme précédent.

– $$\frac{t \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \alpha \multimap \beta \qquad \Gamma \vdash_\Sigma A(\alpha \multimap \beta)}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta \lambda^0 x : \alpha.(t' \downarrow x) : \alpha \multimap \beta}$$

  Par hypothèse de récurrence, on a $\Gamma\,;\,\Delta \vdash_\Sigma t' : \alpha \multimap \beta$.

  On a $\Gamma\,;\,\Delta, x : \alpha \vdash_\Sigma t'\ x : \beta$ et $t'\ x \longrightarrow^*_{\beta\gamma} t' \downarrow x$, donc par le lemme 32 (réduction du sujet pour $\longrightarrow_{\beta\gamma}$) :
  $\Gamma\,;\,\Delta, x : \alpha \vdash_\Sigma t' \downarrow x : \beta$.

  Par la règle (lin. abs.), on obtient bien : $\Gamma\,;\,\Delta \vdash_\Sigma \lambda x : \alpha.(t' \downarrow x) : \alpha \multimap \beta$.
  Les autres cas d'η-expansion sont tout aussi immédiats.

□

Lemme 34 (Réduction du sujet pour $\approx$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$ *et* $\Gamma \vdash_\Sigma t \approx t'$
   alors $\Gamma\,;\,\Delta \vdash_\Sigma t' : \alpha$.

2. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha : K$ *et* $\Gamma \vdash_\Sigma \alpha \approx \alpha'$
   alors $\Gamma \vdash_\Sigma \alpha' : K$.

Démonstration. Immédiat par récurrence sur la structure de Π.

□

Les deux lemmes suivants établissent la propriété de cohérence locale forte pour $\approx$ avec $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_{\eta}$. Ces propriété assurent que $\approx$ commute avec ces deux relations, de la manière la plus simple possible. Elles impliquent notamment que les étapes $\approx$ peuvent toujours petre réléguées à la fin d'une séquence de dérivation, et rassemblées grâce à la transitivité de cette relation. Ainsi, si on a une séquence $\Gamma \vdash_\Sigma t \longrightarrow^*_{\beta\gamma\approx} u$, on peut en déduire $\Gamma \vdash_\Sigma t \longrightarrow^*_{\beta\gamma} v \approx u$.

Lemme 35 (Cohérence locale forte pour $\longrightarrow_{\beta\gamma}$ et $\approx$)**.**

1. *Si* $t \longrightarrow_\beta (resp. \longrightarrow_\gamma) t'$, $\Gamma\,;\,\Delta \vdash_\Sigma t : \beta$ *et* $\Gamma \vdash_\Sigma t \approx u$
   alors $\exists u'$ *tel que* $\Gamma \vdash_\Sigma t' \approx u'$ *et* $u \longrightarrow_\beta (resp. \longrightarrow_\gamma) u'$

2. *Si* $\alpha \longrightarrow_\beta (resp. \longrightarrow_\gamma) \alpha'$, $\Gamma \vdash_\Sigma \alpha : K$ *et* $\Gamma \vdash_\Sigma \alpha \approx \beta$
   alors $\exists \beta'$ *tel que* $\Gamma \vdash_\Sigma \alpha' \approx \beta'$ *et* $\beta \longrightarrow_\beta (resp; \longrightarrow_\gamma) \beta'$

Démonstration. Par récurrence sur la taille de t ou de α, selon la règle de réduction utilisée :

– $\dfrac{v \longrightarrow_\beta v' \qquad \alpha \longrightarrow_\beta \alpha'}{\lambda^0 x : \alpha.v \longrightarrow_\beta \lambda^0 x : \alpha'.v'}$

On a par ailleurs $u = \lambda^0 x : \alpha''.w$
avec $\Gamma \vdash_\Sigma \alpha \equiv \alpha'' : \mathbf{type}$ et $\Gamma \vdash_\Sigma v \approx w$.

Par hypothèse de récurrence, $\exists w'$ tel que $w \longrightarrow_\beta w'$ et $\Gamma \vdash_\Sigma v' \approx w'$.

On a bien $\Gamma \vdash_\Sigma \alpha' \equiv \alpha'' : \mathbf{type}$.
d'où : $\Gamma \vdash_\Sigma \lambda^0 x : \alpha'.v' \approx \lambda^0 x : \alpha''.w'$.

Par ailleurs on a bien par congruence $\lambda x : \alpha''.w \longrightarrow_\beta \lambda x : \alpha''.w'$.

– Les autres règles de congruences donnent une preuve similaire.

– $\dfrac{v \longrightarrow_\beta v' \qquad w \longrightarrow_\beta w'}{(\lambda^0 x : \alpha.v)\, w \longrightarrow_\beta v'[x := w']}$

On a par ailleurs $u = (\lambda^0 x : \alpha'.v'')\, w''$
avec $\Gamma \vdash_\Sigma \alpha \equiv \alpha' : \mathbf{type}$, $\Gamma \vdash_\Sigma \alpha' : \mathbf{type}$, $\Gamma \vdash_\Sigma v \approx v''$, $\Gamma \vdash_\Sigma w \approx w''$.

Par hypothèse de récurrence, on a $\exists v''', w'''$ tels que
$v'' \longrightarrow_\beta v'''$, $\Gamma \vdash_\Sigma v' \approx v'''$
$w'' \longrightarrow_\beta w'''$, $\Gamma \vdash_\Sigma w' \approx w'''$.

Par β-réduction, on a bien $(\lambda^0 x : \alpha'.v'')\, w'' \longrightarrow_\beta v'''[x := w''']$.

Par les lemmes 20 (inversion), 15 (variables libres) et 19 (cohérence du typage), on a
$\Gamma\,;\,\Delta', x : \alpha \vdash_\Sigma v : \beta$ et $\Gamma\,;\,\Delta'' \vdash_\Sigma w : \alpha$.

Par le lemme 32 (réduction du sujet pour $\longrightarrow_{\beta\gamma}$), on a donc :
$\Gamma\,;\,\Delta', x : \alpha \vdash_\Sigma v' : \beta$ et $\Gamma\,;\,\Delta'' \vdash_\Sigma w' : \alpha$.

Par le lemme 29 (substitution pour $\approx$), on a $\Gamma \vdash_\Sigma v'[x := w'] \approx v'''[x := w''']$.

– Si la dernière règle est une autre règle de β-réduction, la preuve est similaire.

– Si la dernière règle est une règle de conversion permutative, la preuve est immédiate.

□

Lemme 36 (Cohérence locale forte pour $\longrightarrow_{\eta}$ et $\approx$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_{\Sigma} t \longrightarrow_{\eta} (resp. \longrightarrow_{c}) t' : \beta$ *et*
   $Gamma \vdash_{\Sigma} t \approx u$
   alors $\exists u'$ *tel que* $\Gamma \vdash_{\Sigma} t' \approx u'$ *et* $\Gamma\,;\,\Delta \vdash_{\Sigma} u \longrightarrow_{\eta} (resp. \longrightarrow_{c}) u' : \beta$.

2. *Si* $\Pi : \Gamma \vdash_{\Sigma} \alpha \longrightarrow_{\eta} (resp. \longrightarrow_{c}) \alpha' : K$ *et* $\Gamma \vdash_{\Sigma} \alpha \approx \beta$
   alors $\exists \beta'$ *tel que* $\Gamma \vdash_{\Sigma} \alpha' \approx \beta'$ *et* $\Gamma \vdash_{\Sigma} \beta \longrightarrow_{\eta} (resp; \longrightarrow_{c}) \beta' : K$.

Démonstration. Par récurrence sur la structure de Π, selon la dernière règle utilisée.

– Si Π se termine par une règle de congruence, la récurrence est immédiate.

– $$\frac{u \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_{\Sigma} t \longrightarrow_{c} t' : \alpha \multimap \beta \qquad \Gamma, x : \alpha, \Gamma' \vdash_{\Sigma} A(\alpha \multimap \beta)}{\Gamma\,;\,\Delta \vdash_{\Sigma} t \longrightarrow_{\eta} \lambda^0 x : \gamma.(t' \downarrow x) : \alpha \multimap \beta}$$

  Par hypothèse de récurrence, $\exists u'$ tel que
  $\Gamma \vdash_{\Sigma} t' \approx u'$
  et $\Gamma\,;\,\Delta \vdash_{\Sigma} u \longrightarrow_{c} u' : \alpha \multimap \beta$.

  Par examen des règles de $\approx$, u est de la même forme que t, et n'est donc pas une abstraction linéaire. On peut donc appliquer une η-expansion pour obtenir :
  $\Gamma\,;\,\Delta \vdash_{\Sigma} u \longrightarrow_{\eta} \lambda^0 x : \gamma.(u' \downarrow x) : \alpha \multimap \beta$.

  Par ailleurs, on a bien
  $\Gamma \vdash_{\Sigma} \lambda^0 x : \gamma.(t' \downarrow x) \approx \lambda^0 x : \gamma.(u' \downarrow x)$.

□

Le lemme suivant établit enfin la propriété de commutativité entre $\longrightarrow_{\eta}$ et $\longrightarrow_{\beta\gamma}$, à un niveau local, c'est-à-dire lorsqu'on opère sur un objet une seule étape de $\longrightarrow_{\beta\gamma}$ et une seule étape de $\longrightarrow_{\eta}$. Il est alors possible de faire converger les deux termes résultants, bien que cette convergence hérite de la complexité des lemmes de substitution pour $\longrightarrow_{\eta}$.

Lemme 37 (Commutativité locale pour $\longrightarrow_{\eta}$ et $\longrightarrow_{\beta\gamma}$)**.**

1. *Si* $\Pi_1 : \Gamma\,;\,\Delta \vdash_{\Sigma} t \longrightarrow_{\eta} (ou \longrightarrow_{c}) t_1 : \alpha$ *et* $\Pi_2 : t \longrightarrow_{\beta\gamma} t_2$,
   alors $\exists t_1^{\beta}, t_2^{\beta}, t_2^{\eta}$ *tels que*
   $t_1 \longrightarrow_{\beta\gamma}^{*} t_1^{\beta}$,
   $t_2 \longrightarrow_{\beta\gamma}^{*} t_2^{\beta}$,
   $\Gamma\,;\,\Delta \vdash_{\Sigma} t_2^{\beta} \longrightarrow_{\eta} t_2^{\eta} : \alpha$
   et $\Gamma \vdash_{\Sigma} t_1^{\beta} \approx t_2^{\eta}$.

2. *Si* $\Pi_1 : \Gamma \vdash_{\Sigma} \alpha \longrightarrow_{\eta} (resp. \longrightarrow_{c}) \alpha_1 : K$ *et* $\Pi_2 : \alpha \longrightarrow_{\beta\gamma} \alpha_2$,
   alors $\exists \alpha_1^{\beta}, \alpha_2^{\beta}, \alpha_2^{\eta}$ *tels que*
   $\alpha_1 \longrightarrow_{\beta\gamma}^{*} \alpha_1^{\beta}$,
   $\alpha_2 \longrightarrow_{\beta\gamma}^{*} \alpha_2^{\beta}$,
   $\Gamma \vdash_{\Sigma} \alpha_2^{\beta} \longrightarrow_{\eta} (resp. \longrightarrow_{c}) \alpha_2^{\eta} : K$
   et $\Gamma \vdash_{\Sigma} \alpha_1^{\beta} \approx \alpha_2^{\eta}$.

Démonstration. Par récurrence sur la taille de t ou de α, en distinguant les différentes possibilités syntaxiques.

– Si $t = \lambda^0 x : \beta.u$

Par le lemme 20 (inversion), on a :
$\exists \beta_1, \gamma, u_1$ tels que
$\Gamma \vdash_\Sigma \alpha \equiv \beta \multimap \gamma : \mathbf{type}$
$t_1 = \lambda^0 x : \beta_1.u_1$
$\Gamma \vdash_\Sigma \beta \longrightarrow_\eta \beta_1 : \mathbf{type}$
$\Gamma\,;\, \Delta, x : \beta \vdash_\Sigma u \longrightarrow_\eta u_1 : \gamma.$

Par ailleurs, de $\lambda^0 x : \beta.u \longrightarrow_{\beta\gamma} u_2$, on a comme seule possibilité $t_2 = \lambda^0 x : \beta_2.u_2$ avec $\beta \longrightarrow_{\beta\gamma} \beta_2$ et $u \longrightarrow_{\beta\gamma} u_2$.

Par hypothèse de récurrence, on a $\exists u_1^\beta, u_2^\beta, u_2^\eta, \beta_1^\beta, \beta_2^\beta, \beta_2^\eta$ tels que
$u_1 \longrightarrow^*_{\beta\gamma} u_1^\beta$
$u_2 \longrightarrow^*_{\beta\gamma} u_2^\beta$
$\Gamma\,;\, \Delta, x : \beta \vdash_\Sigma u_2^\beta \longrightarrow_\eta u_2^\eta : \gamma$
$\Gamma \vdash_\Sigma u_1^\beta \approx u_2^\eta$
$\beta_1 \longrightarrow^*_{\beta\gamma} \beta_1^\beta$
$\beta_2 \longrightarrow^*_{\beta\gamma} \beta_2^\beta$
$\Gamma \vdash_\Sigma \beta_2^\beta \longrightarrow_\eta \beta_2^\eta : \mathbf{type}$
$\Gamma \vdash_\Sigma \beta_1^\beta \approx \beta_2^\eta.$

Par le lemme 24 (équivalence du contexte), on a
$\Gamma\,;\, \Delta, x : \beta_2^\beta \vdash_\Sigma u_2^\beta \longrightarrow_\eta u_2^\eta : \gamma.$
On a bien
$\lambda^0 x : \beta_1.u_1 \longrightarrow^*_{\beta\gamma} \lambda^0 x : \beta_1^\beta.u_1^\beta,$
$\lambda^0 x : \beta_2.u_2 \longrightarrow^*_{\beta\gamma} \lambda^0 x : \beta_2^\beta.u_2^\beta,$
$\Gamma\,;\, \Delta \vdash_\Sigma \lambda^0 x : \beta_2^\beta.u_2^\beta \longrightarrow_\eta \lambda^0 x : \beta_2^\eta.u_2^\eta : \alpha$ par congruence,
et $\Gamma \vdash_\Sigma \lambda^0 x : \beta_1^\beta.u_1^\beta \approx \lambda^0 x : \beta_2^\eta.u_2^\eta$.

– Si t est une abstraction non linéaire ou un enregistrement, la preuve est similaire au cas précédent.

– Si $\Pi_1 : \Gamma\,;\, \Delta \vdash_\Sigma u\, v \longrightarrow_c t_1 : \alpha$

Par le lemme 20 (inversion), on a deux possibilités. Les preuves étant similaires, supposons qu'on est dans le premier cas :
$\exists \Delta_1, \Delta_2, \beta, u_1, v_1$ tels que
$\Delta = \Delta_1, \Delta_2$
$t_1 = u_1\, v_1$
$\Gamma\,;\, \Delta_1 \vdash_\Sigma u \longrightarrow_c u_1 : \beta \multimap \alpha$
$\Gamma\,;\, \Delta_2 \vdash_\Sigma v \longrightarrow_\eta v_1 : \beta.$

On distingue les différents cas pour la dernière règle de Π_2 :

1. $\dfrac{u \longrightarrow_\beta u_2 \qquad v \longrightarrow_\beta v_2}{u\, v \longrightarrow_\beta u_2\, v_2}$

   Par hypothèse de récurrence, on a :
   $\exists u_1^\beta, u_2^\beta, u_2^\eta, v_1^\beta, v_2^\beta, v_2^\eta$ tels que :
   $u_2 \longrightarrow^*_{\beta\gamma} u_2^\beta$
   $\Gamma\,;\, \Delta_1 \vdash_\Sigma u_2^\beta \longrightarrow_\eta u_2^\eta : \beta \multimap \alpha$
   $\Gamma \vdash_\Sigma u_1^\beta \approx u_2^\eta$

$v_2 \longrightarrow^*_{\beta\gamma} v_2^\beta$
$\Gamma\,;\,\Delta_2 \vdash_\Sigma v_2^\beta \longrightarrow_\eta v_2^\eta : \beta$
$\Gamma \vdash_\Sigma v_1^\beta \approx v_2^\eta$
$u_1 \longrightarrow^*_{\beta\gamma} u_1^\beta$
$v_1 \longrightarrow^*_{\beta\gamma} v_1^\beta$

Par le lemme 23 (décomposition de $\longrightarrow_\eta$), on a :
$\exists u_2^c$ tel que
$\Gamma\,;\,\Delta_1 \vdash_\Sigma u_2^\beta \longrightarrow_c u_2^c : \beta \multimap \alpha$
et $u_2^c \downarrow v_2^\eta = u_2^\eta \downarrow v_2^\eta$

Par le lemme 26 (expansion des termes propagés), $\exists w, w'$ tels que :
$u_2^\beta \downarrow v_2^\beta \longrightarrow^*_{\beta\gamma} w'$
$\Gamma\,;\,\Delta \vdash_\Sigma w' \longrightarrow_\eta w : \alpha$
$\Gamma \vdash_\Sigma u_2^c \downarrow v_2^\eta \longrightarrow^*_{\beta\gamma\approx} w.$

On a $u_2\ v_2 \longrightarrow^*_{\beta\gamma} u_2^\beta\ v_2^\beta \longrightarrow^*_{\beta\gamma} u_2^\beta \downarrow v_2^\beta \longrightarrow^*_{\beta\gamma} w'$,
$\Gamma\,;\,\Delta \vdash_\Sigma w' \longrightarrow_\eta w : \alpha$,
et $\Gamma \vdash_\Sigma u_1\ v_1 \longrightarrow^*_{\beta\gamma} u_1^\beta\ v_1^\beta \approx u_2^\eta\ v_2^\eta \longrightarrow^*_{\beta\gamma} u_2^\eta \downarrow v_2^\eta = u_2^c \downarrow v_2^\eta \longrightarrow^*_{\beta\gamma} w$,
ce qui permet bien d'obtenir le résultat grâce au lemme 35 (cohérence locale forte de $\longrightarrow_{\beta\gamma}$ et $\approx$).

2. $$\frac{w \longrightarrow_\beta w_2 \qquad v \longrightarrow_\beta v_2}{(\lambda^0 x : \beta'.w)\ v \longrightarrow_\beta w_2[x := v_2]}$$

En appliquant le lemme 20 (inversion) à $\Gamma\,;\,\Delta_1 \vdash_\Sigma \lambda^0 x : \beta'.w \longrightarrow_c u_1 : \beta \multimap \alpha$ on a :
$\exists \alpha', \beta_1, w_1$ tels que :
$\Gamma \vdash_\Sigma \beta \multimap \alpha \equiv \beta' \multimap \alpha' :$ **type**
$\Gamma \vdash_\Sigma \beta' \longrightarrow_\eta \beta_1 :$ **type**
$u_1 = \lambda^0 x : \beta_1.w_1$
$\Gamma\,;\,\Delta_1, x : \beta' \vdash_\Sigma w \longrightarrow_\eta w_1 : \alpha'.$

Par hypothèse de récurrence, $\exists w_1^\beta, w_2^\beta, w_2^\eta, v_1^\beta, v_2^\beta, v_2^\eta$ tels que :
$w_1 \longrightarrow^*_{\beta\gamma} w_1^\beta$
$w_2 \longrightarrow^*_{\beta\gamma} w_2^\beta$
$\Gamma\,;\,\Delta_1, x : \beta' \vdash_\Sigma w_2^\beta \longrightarrow_\eta w_2^\eta : \alpha'$
$\Gamma \vdash_\Sigma w_1^\beta \approx w_2^\eta$
$v_1 \longrightarrow^*_{\beta\gamma} v_1^\beta$
$v_2 \longrightarrow^*_{\beta\gamma} v_2^\beta$
$\Gamma\,;\,\Delta_2 \vdash_\Sigma v_2^\beta \longrightarrow_\eta v_2^\eta : \beta$
$\Gamma \vdash_\Sigma v_1^\beta \approx v_2^\eta.$

Par l'hypothèse 2 (lemme clé), on a $\Gamma \vdash_\Sigma \alpha \equiv \alpha' : Ctype$ et $\Gamma \vdash_\Sigma \beta \equiv \beta' :$ **type**
On peut donc appliquer le lemme 28 (substitution linéaire pour $\longrightarrow_\eta$) pour obtenir :
$\exists t^\beta, t^\eta$ tels que
$w_2^\beta[x := v_2^\beta] \longrightarrow^*_{\beta\gamma} t^\beta$
$\Gamma\,;\,\Delta \vdash_\Sigma t^\beta \longrightarrow_\eta t^\eta : \alpha',$

$\Gamma \vdash_\Sigma w_2^\eta[x := v_2^\eta] \longrightarrow^*_{\beta\gamma\approx} t^\eta$.

On a :
$\Gamma \vdash_\Sigma (\lambda^0 x : \beta_1.w_1)\ v_1 \longrightarrow^*_{\beta\gamma} (\lambda^0 x : \beta_1.w_1^\beta)\ v_1^\beta \approx (\lambda^0 x : \beta_1.w_2^\eta)\ v_2^\eta \longrightarrow_\beta w_2^\eta[x := v_2^\eta] \longrightarrow^*_{\beta\gamma\approx} t^\eta$,
$w_2[x := v_2] \longrightarrow^*_{\beta\gamma} w_2^\beta[x := v_2^\beta] \longrightarrow^*_{\beta\gamma} t^\beta$,
et $\Gamma\,;\,\Delta \vdash_\Sigma t^\beta \longrightarrow_\eta t^\eta : \alpha$ par (eq$_\eta$.).

ce qui permet bien d'obtenir le résultat grâce au lemme 35 (cohérence locale forte de $\longrightarrow_{\beta\gamma}$ et $\approx$).

3. $\dfrac{w \longrightarrow_\beta w_2 \qquad v \longrightarrow_\beta v_2}{(\lambda x : \beta'.w)\ v \longrightarrow_\beta w_2[x := v_2]}$

En appliquant le lemme 20 (inversion) à $\Gamma\,;\,\Delta_1 \vdash_\Sigma \lambda x : \beta'.w \longrightarrow_c u_1 : \beta \multimap \alpha$ on a :
$\exists \alpha', \beta_1$ tels que :
$\Gamma \vdash_\Sigma \beta \multimap \alpha \equiv (\Pi x : \beta')\,\alpha' :$ **`type`**, ce qui est impossible d'après le lemme 18 (homogénéité des types). Ce cas est donc exclu.

4. $\dfrac{}{\{case\ w \text{ of } (c_i\ x_i) \to u_i\}_{i=1}^n\ v \longrightarrow_\gamma \{case\ w \text{ of } (c_i\ x_i) \to u_i\ v\}_{i=1}^n}$

En appliquant le lemme 20 (inversion) à $\Gamma\,;\,\Delta_1 \vdash_\Sigma \{case\ w \text{ of } (c_i\ x_i) \to u_i\}_{i=1}^n \longrightarrow_c u_1 : \beta \multimap \alpha$ on a :
$\exists a, \beta_1, \cdots, \beta_n, u_1', \cdots, u_n', w', \Delta_1, \Delta 2$ tels que :
$u_1 = \{case\ w' \text{ of } (c_i\ x_i) \to u_i'\}_{i=1}^n$
$\Delta = \Delta_1, \Delta_2$
$binding_\Sigma(a) = \{c_1 \text{ of } \beta_1 | \ldots | c_n \text{ of } \beta_n\}$
$\Gamma\,;\,\Delta_2 \vdash_\Sigma w \longrightarrow_c w' : a$
$\forall i \in \{1, \cdots, n\}, \Gamma\,;\,\Delta_1, x_i : \beta_i \vdash_\Sigma u_i \longrightarrow_\eta u_i' : \beta \multimap \alpha$.

Par le lemme 23 (décomposition de $\longrightarrow_\eta$), on a donc
$\forall i \in \{1, \cdots, n\}, \exists u_i''$ tel que
$\Gamma\,;\,\Delta_1, x_i : \beta_i \vdash_\Sigma u_i \longrightarrow_c u_i'' : \beta \multimap \alpha$
et $u_i' \downarrow v_1 = u_i'' \downarrow v_1$.

Par le lemme 26 (expansion des termes propagés), $\exists t_i, t_{i'}$ tels que :
$u_i \downarrow v \longrightarrow^*_{\beta\gamma} t_i'$
$\Gamma\,;\,\Delta_1, x_i : \beta_i, \Delta_2 \vdash_\Sigma t_i' \longrightarrow_c t_i : \alpha$
$\Gamma \vdash_\Sigma u_i'' \downarrow v_1 \longrightarrow^*_{\beta\gamma\approx} t_i$.

On a :
$\{case\ w \text{ of } (c_i\ x_i) \to u_i\ v\}_{i=1}^n \longrightarrow^*_{\beta\gamma} \{case\ w \text{ of } (c_i\ x_i) \to u_i \downarrow v\}_{i=1}^n \longrightarrow^*_{\beta\gamma} \{case\ w \text{ of } (c_i\ x_i) \to t_i'\}_{i=1}^n$
$\Gamma\,;\,\Delta \vdash_\Sigma \{case\ w \text{ of } (c_i\ x_i) \to t_i'\}_{i=1}^n \longrightarrow_c \{case\ w' \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n : \alpha$ par congruence
et $\Gamma \vdash_\Sigma \{case\ w' \text{ of } (c_i\ x_i) \to u_i'\}_{i=1}^n\ v_1 \longrightarrow_\gamma \{case\ w' \text{ of } (c_i\ x_i) \to u_i'\ v_1\}_{i=1}^n \longrightarrow^*_{\beta\gamma} \{case\ w' \text{ of } (c_i\ x_i) \to u_i' \downarrow v_1\}_{i=1}^n = \{case\ w' \text{ of } (c_i\ x_i) \to u_i'' \downarrow v_1\}_{i=1}^n \longrightarrow^*_{\beta\gamma\approx} \{case\ w' \text{ of } (c_i\ x_i) \to t_i\}_{i=1}^n$.
ce qui permet bien d'obtenir le résultat grâce au lemme 35 (cohérence locale forte de $\longrightarrow_{\beta\gamma}$ et $\approx$).

– Si $\Pi_1 : \Gamma\,;\,\Delta \vdash_\Sigma u\ v \longrightarrow_\eta t_1 : \alpha$

Par le lemme 20 (inversion), en supposant là encore que l'application est linéaire, on a plusieurs possibilités. Si t_1 est une application, la preuve est identique au cas précédent. Les autres cas étant similaires, supposons que t_1 est une abstraction linéaire :
$\exists \Delta_1, \Delta_2, \beta, u_1, v_1, \alpha_1, \alpha_2$ tels que
$\Delta = \Delta_1, \Delta_2$
$t_1 = \lambda^0 x : \beta_1.((u_1\ v_1)\ x)$
$\Gamma \vdash_\Sigma \alpha \equiv \alpha_1 \multimap \alpha 2 :$ **type**
$\Gamma \vdash_\Sigma A(\alpha_1 \multimap \alpha_2)$
$\Gamma\,;\,\Delta_1 \vdash_\Sigma u \longrightarrow_c u_1 : \beta \multimap \alpha$
$\Gamma\,;\,\Delta_2 \vdash_\Sigma v \longrightarrow_\eta v_1 : \beta.$

De la même façon qu'au cas précédent, quelle que soit la dernière règle de Π_2, on a :
$\exists t^\beta, t^\eta$ tels que :
$\Gamma \vdash_\Sigma u_1\ v_1 \longrightarrow^*_{\beta\gamma\approx} t^\eta$,
$t_2 \longrightarrow^*_{\beta\gamma} t^\beta$,
et $\Gamma\,;\,\Delta \vdash_\Sigma t^\beta \longrightarrow_\eta t^\eta : \alpha.$

Par le lemme 23 (décomposition de $\longrightarrow_\eta$), $\exists t^c$ tel que :
$\Gamma\,;\,\Delta \vdash_\Sigma t^\beta \longrightarrow_c t^c : \alpha$
et $t^c \downarrow x = t^\eta \downarrow x.$

On a alors :
$\Gamma \vdash_\Sigma \lambda^0 x : \beta_1.((u_1\ v_1)\ x) \longrightarrow^*_{\beta\gamma\approx} \lambda^0 x : \beta_1.(t^\eta\ x) \longrightarrow^*_{\beta\gamma} \lambda^0 x : \beta_1.(t^\eta \downarrow x)$,
$t_2 \longrightarrow^*_{\beta\gamma} t^\beta$,
$\Gamma\,;\,\Delta \vdash_\Sigma t^\beta \longrightarrow_\eta \lambda^0 x : \beta_1.(t^c \downarrow x) : \alpha$ par (eq$_c$.), expansion et (eq$_\eta$.)
ce qui permet bien d'obtenir le résultat grâce au lemme 35 (cohérence locale forte de $\longrightarrow_{\beta\gamma}$ et $\approx$).

– Si t est d'une autre forme syntaxique, la preuve est similaire aux cas précédents.

□

On peut alors étendre cette propriété de commutativité au cas où une séquence de $\longrightarrow_{\beta\gamma}$ de longueur arbitraire part de l'objet initial. En revanche l'étape de $\longrightarrow_\eta$ partant également du terme initial reste, elle, unique. Cette formulation est en effet suffisante pour assurer la preuve de l'unicité des formes normales, et la normalisation forte de $\longrightarrow_{\beta\gamma}$ permet facilement de la prouver par récurrence, à partir de la propriété locale établit précédemment.

Lemme 38 (Commutativité entre $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_\eta$)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (ou \longrightarrow_c) t_1 : \alpha$ *et* $t \longrightarrow^*_{\beta\gamma} t_2$,
   alors $\exists t', t''$ *tels que* $\Gamma \vdash_\Sigma t_1 \longrightarrow^*_{\beta\gamma\approx} t'$, $t_2 \longrightarrow^*_{\beta\gamma} t''$ *et* $\Gamma\,;\,\Delta \vdash_\Sigma t'' \longrightarrow_\eta t' : \alpha.$

2. *Si* $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta (resp. \longrightarrow_c) \alpha_1 : K$ *et* $\alpha \longrightarrow^*_{\beta\gamma} \alpha_2$,
   alors $\exists \alpha', \alpha''$ *tels que* $\alpha_1 \longrightarrow^*_{\beta\gamma} \alpha'$, $\alpha_2 \longrightarrow^*_{\beta\gamma} \alpha''$ *et* $\Gamma \vdash_\Sigma \alpha'' \longrightarrow_\eta (resp. \longrightarrow_c) \alpha' : K.$

Démonstration. Par récurrence sur le maximum des longueurs des séquences de $\longrightarrow_{\beta\gamma}$ partant de t ou de $alpha$, en utilisant les lemmes 35 (cohérence locale forte pour $\longrightarrow_{\beta\gamma}$ et $\approx$), 37 (commutativité locale pour $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$) et le théorème 4 (confluence de $\longrightarrow_{\beta\gamma}$).

Ce maximum existe bien, de par le théorème 2 (normalisation forte de $\longrightarrow_{\beta\gamma}$) et le fait que chaque objet n'a qu'un nombre fini de possibilités de $\beta\gamma$-réductions.

□

Cette propriété de commutativité est notamment conçue pour placer l'étape de $\longrightarrow_\eta$ à la fin de la séquence de réécritures dans laquelle elle apparait. Ce détail a son importance car, s'il est difficile d'énumérer les antécédents possibles d'une étape de $\longrightarrow_{\beta\gamma}$, il est en revanche aisé de le faire pour $\longrightarrow_\eta$.

3.2 Confluence de $\longrightarrow_{\beta\gamma\eta}$

Dans cette section, nous utilisons la propriété de commutativité prédécemment établie pour en déduire la confluence de $\longrightarrow_{\beta\gamma\eta}$. Le schéma de la preuve est le suivant : dans un premier temps nous démontrons que deux formes $\beta\gamma\eta$-normales équivalentes sont forcément syntaxiquement égales. Dans un second temps nous démontrons que $\longrightarrow_{\beta\gamma\eta}$ est faiblement normalisant : tout objet peut donc être réduit vers une forme $\beta\gamma\eta$-normale. En conséquence, $\longrightarrow_{\beta\gamma\eta}$ est bien confluente.

3.2.1 Unicité des formes normales

Afin de démontrer que deux formes $\beta\gamma\eta$ -normales équivalentes sont égales, nous démontrons des propriétés de compatibilité, énonçant que deux objets équivalents peuvent toujours être réduits vers des objets $\beta\gamma$-normaux de la même forme. Cette propriété permet alors d'obtenir l'unicité des formes normales par une récurrence simple sur la taille des objets considérés. La compatibilité est prouvée en examinant toutes les configurations possibles entre les séquences de $\longrightarrow_{\beta\gamma}$ menant à une forme $\beta\gamma$-normale et le jugement d'équivalence dont on dispose entre les deux objets considérés. La seule configuration qui ne soit pas couverte par un lemme précédemment établi (confluence de $\longrightarrow_{\beta\gamma}$, cohérence locale forte entre $\longrightarrow_{\beta\gamma}$ et $\approx$, commutativité entre $\longrightarrow_{\eta}$ et $\approx$) est le cas où une expansion précède cette séquence de $\longrightarrow_{\beta\gamma}$. C'est pourquoi nous prouvons dans un premier temps un lemme d'ajournement de $\longrightarrow_{\eta}$ permettant de déplacer une étape $\longrightarrow_{\eta}$ après une séquence de $\longrightarrow_{\beta\gamma}$.

La définition formelle des formes normales de $\longrightarrow_{\eta}$ et $\longrightarrow_{\beta\gamma\eta}$ est la suivante :

Définition 20 (Formes normales)**.**

1. *Un terme t est en forme η-normale (resp. c-normale) pour Σ, Γ, Δ et α si et seulement si $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (resp. \longrightarrow_c) t' : \alpha$ implique $t = t'$.*

2. *Un type α est en forme η-normale (resp. c-normale) pour Σ, Γ et K si et seulement si $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta (resp. \longrightarrow_c) \alpha' : K$ implique $\alpha = \alpha'$.*

3. *Un terme t est en forme $\beta\gamma\eta$-normale (resp. $\beta\gamma$c-normale) pour Σ, Γ, Δ et α si et seulement si t est en forme η-normale (resp. c-normale) pour Σ, Γ, Δ et α, et t est en forme $\beta\gamma$-normale.*

4. *Un type α est en forme $\beta\gamma\eta$-normale (resp. $\beta\gamma$c-normale) pour Σ, Γ et K si et seulement si α est en forme η-normale (resp. c-normale) pour Σ, Γ et K, et α est en forme $\beta\gamma$-normale.*

Tout d'abord, nous établissons le lemme assurant que les termes produits par le critère syntaxique de la section 2.2.4 sont bien des formes $\beta\gamma\eta$-normales.

Lemme 39 (Validité du critère syntaxiques)**.**

1. *Si* $\Pi : \Gamma \vdash_\Sigma A(\alpha)$
   alors α est en forme $\beta\gamma\eta$-normale pour Σ, Γ, K.

2. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma E(t)$ *et* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$
   alors t est en forme $\beta\gamma\eta$-normale pour $\Sigma, \Gamma, \Delta, \alpha$

3. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma I(t)$ *et* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$
   alors t est en forme $\beta\gamma$c-normale pour $\Sigma, \Gamma, \Delta, \alpha$.

Démonstration. Par récurrence sur la structure de Π.

□

Les deux lemmes suivants assurent que la $\beta\gamma$-normalité est préservée par $\downarrow$ et surtout par $\longrightarrow_\eta$. Cette propriété est utile à la preuve de normalisation faible de $\longrightarrow_{\beta\gamma\eta}$, puisqu'il suffit alors de normaliser pour $\longrightarrow_{\beta\gamma}$, puis pour $\longrightarrow_\eta$. De plus, cette propriété est indispensable aux lemmes de compatibilité qui vont suivre. Elle permet en effet de s'assurer que la $\beta\gamma$-normalité est transportée lorsqu'on traite une étape de $\equiv$ en utilisant le lemme d'ajournement ou le lemme de commutativité.

Lemme 40 (Conservation de la $\beta\gamma$-normalité par $\downarrow$)**.**

1. *Si* $t \downarrow x \longrightarrow_{\beta\gamma} u$ *et* $u \neq t \downarrow x$
   Alors $\exists t'$ *tel que* $\longrightarrow_{\beta\gamma} t'$ *et* $t \neq t'$.

2. *Si* $t \downarrow l \longrightarrow_{\beta\gamma} u$ *et* $u \neq t \downarrow l$
   Alors $\exists t'$ *tel que* $\longrightarrow_{\beta\gamma} t'$ *et* $t \neq t'$.

Démonstration. Immédiat par récurrence sur la taille de t.

□

Lemme 41 (Conservation de la $\beta\gamma$-normalité par $\longrightarrow_\eta$)**.**

1. *Si* $\Pi : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (ou \longrightarrow_c) t' : \alpha$
   et $t' \longrightarrow_{\beta\gamma} t''$ *avec* $t' \neq t''$
   Alors $\exists t'''$ *tel que* $t \longrightarrow_{\beta\gamma} t'''$ *avec* $t \neq t'''$.

2. *Si* $\Pi : \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta (ou \longrightarrow_c) \alpha' : K$
   et $\alpha' \longrightarrow_{\beta\gamma} \alpha''$ *avec* $\alpha' \neq \alpha''$
   Alors $\exists \alpha'''$ *tel que* $\alpha \longrightarrow_{\beta\gamma} \alpha'''$ *avec* $\alpha \neq \alpha'''$.

Démonstration. Par récurrence sur la hauteur de Π.

– Si Π termine par (c / η), la récurrence est immédiate.

– Si Π termine par :

$$\frac{\Gamma\,;\,\Delta_1 \vdash_\Sigma u \longrightarrow_c u' : \alpha \multimap \beta \qquad \Gamma\,;\,\Delta_2 \vdash_\Sigma v \longrightarrow_\eta v' : \alpha}{\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma u\,v \longrightarrow_c u'\,v' : \beta}\ \text{(lin. app.)}$$

On a donc $u'\,v' \longrightarrow_{\beta\gamma} t''$.

1. Si $t'' = u''\,v''$ avec $u' \longrightarrow_{\beta\gamma} u''$ et $v' \longrightarrow_{\beta\gamma} v''$, alors on a soit $u' \neq u''$ soit $v' \neq v''$. Les 2 cas étant similaires, supposons $u' \neq u''$.
   On a donc par hypothèse de récurrence $\exists u'''$ tel que $u \longrightarrow_{\beta\gamma} u'''$ et $u \neq u'''$.
   On a donc bien $t = u\,v \longrightarrow_{\beta\gamma} u'''\,v$ avec $u'''\,v \neq t$.

2. Si $u' = \lambda^0 x : \gamma'.w'$, $t'' = w''[x := v'']$ avec $w' \longrightarrow_\beta w''$ et $v' \longrightarrow_\beta v''$,
   par examen des règles de $\longrightarrow_c$, u est de la forme $\lambda^0 x : \gamma.w$.
   On a donc $u\,v \longrightarrow_\beta w[x := v] \neq u\,v$.

3. Si t'' est obtenu par une autre réduction ou une conversion permutative, la preuve est similaire au cas précédent.

– Si Π termine par une autre règle de congruence, la preuve est smilaire au cas précédent.

– Si Π termine par :

$$\frac{t \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \alpha \multimap \beta \qquad \Gamma \vdash_\Sigma A(\alpha \multimap \beta)}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta \lambda^0 x : \alpha.(t' \downarrow x) : \alpha \multimap \beta}$$

Par le lemme 39 (validité du critère syntaxique), α est en forme β-γ-normale.
On a $t'' = \lambda^0 x : \alpha.u''$ avec $t' \downarrow x \longrightarrow_{\beta\gamma} u''$ et $t' \downarrow x \neq u''$.
Par le lemme 40, $\exists t'''$ tel que $t \longrightarrow_{\beta\gamma} t'''$ et $t \neq t'''$.

– Si Π termine par une autre η-expansion la preuve est similaire au cas précédent.

□

Cette propriété de conservation de la $\beta\gamma$-normalité permet également de déduire le lemme d'ajournement du lemme de commutativité, plutôt que de le prouver en énumérant tous les cas possibles.

Lemme 42 (Ajournement de $\longrightarrow_\eta$)**.**

1. *Si* $\Pi_1 : \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta (ou \longrightarrow_c) u : \alpha$ *et* $\Pi_2 : u \longrightarrow^*_{\beta\gamma} v$
   Alors $\exists u', v'$ *tels que*
   $t \longrightarrow^*_{\beta\gamma} u'$,
   $\Gamma\,;\,\Delta \vdash_\Sigma u' \longrightarrow_\eta v' : \alpha$,
   $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} v'$ *et*
   u', v' *sont en forme* $\beta\gamma$*-normale.*

2. *Si* $\Pi_1 : \Gamma \vdash_\Sigma \alpha \longrightarrow_\eta (resp. \longrightarrow_c) \beta : K$ *et* $\Pi_2 : \beta \longrightarrow^*_{\beta\gamma} \gamma$
   Alors $\exists \beta', \gamma'$ *tels que*
   $\alpha \longrightarrow^*_{\beta\gamma} \beta'$,
   $\Gamma \vdash_\Sigma \beta' \longrightarrow_\eta (resp. \longrightarrow_c) \gamma' : K$,
   $\Gamma \vdash_\Sigma \gamma \longrightarrow^*_{\beta\gamma\approx} \gamma'$ *et*
   β', γ' *sont en forme normale.*

Démonstration. On traite le premier cas, le second étant similaire.
Par le théorème 2 (normalisation forte de $\longrightarrow_{\beta\gamma}$), $\exists u'$ forme $\beta\gamma$-normale telle que t $\longrightarrow^*_{\beta\gamma} u'$.

Par le lemme 38 (commutativité entre $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_\eta$), $\exists v'$ tel que
$\Gamma\,;\,\Delta \vdash_\Sigma u' \longrightarrow_\eta v' : \alpha$ et
$\Gamma \vdash_\Sigma u \longrightarrow^*_{\beta\gamma\approx} v'$.

Par le lemme 41 (conservation de la $\beta\gamma$-normalité par $\longrightarrow_\eta$), v' est une forme $\beta\gamma$-normale.

Par le théorème 4 (confluence de $\longrightarrow_{\beta\gamma}$ et le lemme 35 (cohérence locale forte de $\approx$ et $\longrightarrow_{\beta\gamma}$), on a donc
$\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} v'$.

□

Le lemme suivant énonce une propriété basique de $\equiv$ nécessaire pour la preuve de la compatibilité. $\equiv$ est en effet bien congruente. La seule difficulté est posée par le mode de congruence particulier de $\longrightarrow_\eta$, mais les cas concernés sont facilement résolus en utilisant le lemme 23 (décomposition de $\longrightarrow_\eta$). Ainsi, si $\Gamma\,;\,\Delta_1 \vdash_\Sigma f \longrightarrow_\eta \lambda x : \alpha(f\ x) : \alpha \multimap \beta$ et $\Gamma\,;\,\Delta_2 \vdash_\Sigma u \longrightarrow_\eta u'$, on n'a certes pas $\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma f\ u \longrightarrow_\eta (\lambda x : \alpha(f\ x))\ u : \beta$, mais on a bien $(\lambda x : \alpha(f\ x))\ u \longrightarrow_\beta f\ u$, d'où $\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma f\ u \equiv (\lambda x : \alpha(f\ x))\ u$.

Lemme 43 (congruence pour $\equiv$)**.**
Si $\Gamma\,;\,\Delta_1 \vdash_\Sigma t \equiv t' : \alpha \multimap \beta$ *et* $\Gamma\,;\,\Delta_2 \vdash_\Sigma u \equiv u' : \alpha$
Alors $\Gamma\,;\,\Delta_1, \Delta_2 \vdash_\Sigma t\ u \equiv t'\ u' : \alpha$
Si $\Gamma\,;\,\Delta \vdash_\Sigma t \equiv t' : a$ *et* $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$
Alors $\forall i \in \{1, \cdots, n\}, \Gamma\,;\,\Delta \vdash_\Sigma t.l_i \equiv t'.l_i : \alpha$
Si $\Gamma \vdash_\Sigma \alpha \equiv \alpha' : (\beta)\,K$ *et* $\Gamma\,;\,. \vdash_\Sigma t \equiv t' : \beta$
Alors $\Gamma \vdash_\Sigma \alpha\ t \equiv \alpha'\ t' : K$

Démonstration. Immédiat par récurrence sur le nombre de réécritures impliquées dans les $\equiv$.

□

Les deux lemmes suivants établissent la propriété de compatibilité, qui est au coeur de cette section. L'essence de cette propriété est d'assurer que deux formes $\beta\gamma\eta$-normales équivalentes sont forcément de la même forme syntaxique, et que leurs sous-termes sont également équivalents. Cependant, pour les besoins de la démonstration par récurrence sur le nombre d'étapes dans le jugement d'équivalence, cette propriété est étendu au cas où l'un des deux objets comparés n'est pas en forme $\beta\gamma\eta$-normale. En ce cas, il suffit de considérer la forme $\beta\gamma$-normale de cet objet.

Lemme 44 (Compatibilité)**.**

1. *Si* $t\ u$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ*,* t *n'est pas de la forme* c_i *et* $\Gamma\ ;\ \Delta \vdash_\Sigma t\ u \equiv v : \beta$
   Alors
   - *Soit* $\exists t', u', \alpha, \Delta_1, \Delta_2$ *tels que*
     $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} t'\ u'$,
     $\Gamma\ ;\ \Delta_1 \vdash_\Sigma t \equiv t' : \alpha \multimap \beta$,
     $\Gamma\ ;\ \Delta_2 \vdash_\Sigma u \equiv u' : \alpha$,
     $\Delta = \Delta_1, \Delta_2$,
     t' *n'est pas de la forme* c_i *et* $t'\ u'$ *est en forme* $\beta\gamma$*-normale.*

   - *Soit* $\exists t', u', \alpha$ *tels que*
     $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} t'\ u'$,
     $\Gamma\ ;\ \Delta \vdash_\Sigma t \equiv t' : (\Pi x : \alpha)\, \beta'$,
     $\Gamma\ ;\ .\ \vdash_\Sigma u \equiv u' : \alpha$,
     $\beta = \beta'[x := u]$,
     t' *n'est pas de la forme* c_i *et* $t'\ u'$ *est en forme* $\beta\gamma$*-normale.*

2. *Si* $c_i\ u$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ*, et* $\Gamma\ ;\ \Delta \vdash_\Sigma c_i\ u \equiv v : \beta$
   Alors $\exists u'$ *tel que*
   $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} c_i\ u'$,
   $\Gamma\ ;\ \Delta \vdash_\Sigma u \equiv u' : \alpha$ *et*
   $c_i\ u'$ *est en forme* $\beta\gamma$*-normale.*

3. *Si* $t.l$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma\ ;\ \Delta \vdash_\Sigma t.l \equiv v : \beta$
   Alors $\exists t', a$ *tels que*
   $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} t'.l$,
   $\Gamma\ ;\ \Delta \vdash_\Sigma t \equiv t' : a$ *et*
   $t'.l$ *est en forme* $\beta\gamma$*-normale.*

4. *Si* $\{case\ w\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *,* $\exists i$ *tel que* $t_i \neq c_i\ x_i$ *et* $\Gamma\ ;\ \Delta \vdash_\Sigma$ $\{case\ w\ of\ (c_i\ x_i) \to t_i\}_{i=1}^n \equiv v : \beta$
   Alors $\exists w', t'_1, \cdots, t'_n, a, \Delta_1, \Delta_2$ *tels que*
   $\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} \{case\ w'\ of\ (c_i\ x_i) \to t'_i\}_{i=1}^n$,
   $\Gamma\ ;\ \Delta_1 \vdash_\Sigma w \equiv w' : a$,
   $binding_\Sigma(a) = \{c_1\ of\ \alpha_1 | \ldots | c_n\ of\ \alpha_n\}$, $\forall i \in \{1, \cdots, n\}$,
   $\Gamma\ ;\ \Delta_2, x_i \vdash_\Sigma t_i \equiv t'_i : \alpha_i$,
   $\Delta = \Delta_1, \Delta_2$,
   $\exists i$ *tel que* $t'_i \neq c_i\ x_i$ *et*
   $\{case\ w'\ of\ (c_i\ x_i) \to t'_i\}_{i=1}^n$ *est en forme* $\beta\gamma$*-normale.*

5. *Si* $\{case\ w\ of\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma\ ;\ \Delta \vdash_\Sigma \{case\ w\ of\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n \equiv v : a$
   Alors

– *Soit* $\exists w'$ *tel que*
w' *n'est pas une analyse de cas,*
w' *n'est pas de la forme* $c_j\ k$,
$\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} w'$ *et*
w' *est en forme* $\beta\gamma$*-normale.*

– *Soit* $\exists w'$ *tel que*
$\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} \{case\ w'\ of\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n$,
$\Gamma\,;\,\Delta \vdash_\Sigma w \equiv w' : a$ *et*
$\{case\ w'\ of\ (c_i\ x_i) \to c_i\ x_i\}_{i=1}^n$ *est en forme* $\beta\gamma$*-normale.*

6. *Si* x *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma\,;\,\Delta \vdash_\Sigma x \equiv v : \beta$
Alors $v \longrightarrow^*_{\beta\gamma} x$.

7. *Si* c *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma\,;\,\Delta \vdash_\Sigma c \equiv v : \beta$
Alors $v \longrightarrow^*_{\beta\gamma} c$.

8. *Si* $\alpha\ t$ *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma \vdash_\Sigma \alpha\ t \equiv \beta : K$
Alors $\exists \alpha', t', \gamma$ *tels que*
$\Gamma \vdash_\Sigma \beta \longrightarrow^*_{\beta\gamma\approx} \alpha'\ t'$,
$\Gamma \vdash_\Sigma \alpha \equiv \alpha' : (\gamma)\,K$,
$\Gamma\,;\,\vdash_\Sigma t \equiv t' : \gamma$ *et*
$\alpha'\ t'$ *est en forme* $\beta\gamma$*-normale.*

9. *Si* a *est une forme* $\beta\gamma\eta$*-normale dans* Σ, Γ, Δ *et* $\Gamma \vdash_\Sigma a \equiv \alpha : K$
Alors $\alpha \longrightarrow^*_{\beta\gamma} a$.

Démonstration. Par récurrence sur le nombre de réécritures impliquées dans $\equiv$. On traite le premier cas, les autres étant similaires.

On suppose qu'on a un séquent $\Gamma\,;\,\Delta \vdash_\Sigma t\ u \equiv v\ :\ \beta$ sur lequel on dispose de l'hypothèse de récurrence : $\exists t', u', \alpha, \Delta_1, \Delta_2$ tels que
$\Gamma \vdash_\Sigma v \longrightarrow^*_{\beta\gamma\approx} t'\ u'$,
$\Gamma\,;\,\Delta_1 \vdash_\Sigma t \equiv t' : \alpha \multimap \beta$,
$\Gamma\,;\,\Delta_2 \vdash_\Sigma u \equiv u' : \alpha$,
$\Delta = \Delta_1, \Delta_2$,
t' n'est pas de la forme c_i et
$t'\ u'$ est en forme $\beta\gamma$-normale.

On distingue à présent les différentes possibilités d'ajouter une étape de réécriture supplémentaire vers un terme v' :

– Si $v' \longrightarrow_{\beta\gamma} v$, la preuve est immédiate.

– Si $v \longrightarrow_{\beta\gamma} v'$,
par les lemmes 4 (confluence de $\longrightarrow_{\beta\gamma}$) et 35 (cohérence locale forte pour $\approx$ et $\longrightarrow_{\beta\gamma}$), et le fait que $t'\ u'$ est en forme $\beta\gamma$-normale, on a $v' \longrightarrow^*_{\beta\gamma} t'\ u'$.

– Si $\Gamma\,;\,\Delta \vdash_\Sigma v \longrightarrow_\eta v' : \beta$,
par le lemme 38 (commutativité de $\longrightarrow_\eta$ et $\longrightarrow_{\beta\gamma}$), $\exists v''$ tel que $\Gamma \vdash_\Sigma v' \longrightarrow^*_{\beta\gamma\approx} v''$ et $\Gamma\,;\,\Delta \vdash_\Sigma t'\ u' \longrightarrow_\eta v'' : \beta$. Cette dernière règle est une congruence, car sinon on pourrait de la même façon expandre $t\ u$, qui ne serait donc pas une forme normale. On a donc par le lemme 20 (inversion) $v'' = t''\ u''$, avec

$\Gamma\,;\,\Delta_1' \vdash_\Sigma t' \;\longrightarrow_c\; t'' : \alpha' \multimap \beta$ et $\Gamma\,;\,\Delta_2' \vdash_\Sigma u' \;\longrightarrow_\eta\; u'' : \alpha'$.

Par le lemme 15 (variables libres), on a $\Delta_1 = \Delta_1'$ et $\Delta_2 = \Delta_1'$.

On a $\Gamma\,;\,\Delta_2 \vdash_\Sigma u' : \alpha$ et $\Gamma\,;\,\Delta_2 \vdash_\Sigma u' : \alpha'$, donc par le lemme 30 (equivalence du typage),
$\Gamma \vdash_\Sigma \alpha \equiv \alpha' :$ **type**.
On a donc bien $\Gamma\,;\,\Delta_1 \vdash_\Sigma t \equiv t'' : \alpha \multimap \beta$ et $\Gamma\,;\,\Delta_2' \vdash_\Sigma u \equiv u'' : \alpha$.

Par le lemme 41 (conservation de la $\beta\gamma$-normalité par $\longrightarrow_\eta$), $t''\ u''$ est en forme $\beta\gamma$-normale.

Par le lemme 20 (inversion), t'' est de la même forme que t', donc pas de la forme c_i.

– Si $\Gamma\,;\,\Delta \vdash_\Sigma v' \;\longrightarrow_\eta\; v : \beta$,
par les lemmes 42 (ajournement de $\longrightarrow_\eta$), et 35 (cohérence locale forte pour $\approx$ et $\longrightarrow_{\beta\gamma}$), et le fait que $t'\ u'$ est en forme $\beta\gamma$-normale, $\exists w, w'$ formes $\beta\gamma$-normales telles que
$v' \;\longrightarrow^*_{\beta\gamma}\; w$,
$\Gamma\,;\,\Delta \vdash_\Sigma w \;\longrightarrow_\eta\; w' : \beta$ et
$\Gamma \vdash_\Sigma t'\ u' \approx w'$.

Par examen des règles de $\longrightarrow_\eta$ et $\approx$, on a $w' = t'''\ u'''$ et $w = t''\ u''$. La preuve est alors similaire au cas précédent.

□

Le lemme de compatibilité faible décrit un mécanisme similaire lorsque l'on s'intéresse à un terme en forme $\beta\gamma c$-normale plutôt que $\beta\gamma\eta$-normale.

Lemme 45 (Compatibilité faible)**.**

1. *Si $t.l$ est une forme $\beta\gamma c$-normale dans Σ, Γ, Δ et $\Gamma\,;\,\Delta \vdash_\Sigma t.l \equiv v : \gamma \multimap \beta$*
   Alors
   – *Soit $\exists t', a$ tels que*
   $\Gamma \vdash_\Sigma v \;\longrightarrow^*_{\beta\gamma\approx}\; t'.l$,
   $\Gamma\,;\,\Delta \vdash_\Sigma t \equiv t' : a$ *et*
   $t'.l$*est en forme $\beta\gamma$-normale.*

   – *Soit $\exists t', a$ tels que*
   $\Gamma \vdash_\Sigma v \;\longrightarrow^*_{\beta\gamma\approx}\; \lambda^0 x : \gamma.((t'.l)\ x)$,
   $\Gamma\,;\,\Delta \vdash_\Sigma t \equiv t' : a$ *et*
   $\lambda^0 x : \gamma.((t'.l)\ x)$*est en forme $\beta\gamma$-normale.*

2. *On a une propriété similaire pour les 7 autres combinaisons entre la forme du terme faiblement normal (application et projection) et son type (type fonctionnel linéaire, produit dépendant, type somme et type produit).*

Démonstration. Similaire à la preuve du lemme précédent. □

Puisque les lemmes de compatibilités impliquent que deux formes normales équivalentes ont forcément la même forme syntaxique, et que leurs sous-termes sont eux-mêmes équivalents, il est aisé d'établir une récurrence pour montrer que ces deux formes normales équivalentes sont en réalité syntaxiquement égales. C'est ce que nous formalisons dans le lemme suivant :

Lemme 46 (unicité des formes normales)**.**

1. *Si t et t' sont $\beta\gamma\eta$-normaux pour $\Sigma, \Gamma, \Delta, \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma t' : \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$ et $\Gamma\,;\,\Delta \vdash_\Sigma t \equiv t' : \alpha$*
   Alors $t = t'$.

2. *Si* α *et* α' *sont* $\beta\gamma\eta$*-normaux pour* Σ, Γ, K , $\Gamma \vdash_\Sigma \alpha : K$, $\Gamma \vdash_\Sigma \alpha' : K$ *et* $\Gamma \vdash_\Sigma \alpha \equiv \alpha' : K$
   Alors $\alpha = \alpha'$.

3. *Si* t *et* t' *sont* $\beta\gamma c$*-normaux pour* $\Sigma, \Gamma, \Delta, \alpha$ *et* t *et* t' *ne sont pas* $\beta\gamma\eta$*-normaux pour* $\Sigma, \Gamma, \Delta, \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$ *et* t *est une application ou une projection et* $\Gamma\,;\,\Delta \vdash_\Sigma t \equiv t' : \alpha$
   Alors $t = t'$.

4. *Si* α *et* α' *sont* $\beta\gamma c$*-normaux pour* Σ, Γ, K *et* α *et* α' *ne sont pas* $\beta\gamma\eta$*-normaux pour* Σ, Γ, K , $\Gamma \vdash_\Sigma \alpha : K$, $\Gamma \vdash_\Sigma \alpha' : K$ *et* $\Gamma \vdash_\Sigma \alpha \equiv \alpha' : K$
   Alors $\alpha = \alpha'$.

Démonstration. Par récurrence sur la taille de t ou α. On traite le premier cas, les autres étant similaires.

– Si $t = \lambda^0 x : \beta.u$, alors par le lemme 20 (inversion), $\Gamma \vdash_\Sigma \alpha \equiv \beta \multimap \gamma :$ **type**.
  Si t' n'était pas une abstraction, par l'hypothèse 1 (existence des formes normales), on pourrait lui appliquer une expansion, or t' est une forme normale dans Σ, Γ, Δ. On a donc $t' = \lambda^0 x : \beta'.u'$, et $\Gamma \vdash_\Sigma \beta \equiv \beta' :$ **type**.

  β et β' sont normaux dans Σ, Γ, donc par hypothèse de récurrence, $\beta = \beta'$.

  Par le lemme 43 (congruence pour $\equiv$), on a $\Gamma\,;\,\Delta, x : \beta \vdash_\Sigma (\lambda^0 x : \beta.u)\ x \equiv (\lambda^0 x : \beta.u')\ x : \gamma$, or $(\lambda^0 x : \beta.u)\ x \longrightarrow_\beta u$ et $(\lambda^0 x : \beta.u')\ x \longrightarrow_\beta u'$, donc $\Gamma\,;\,\Delta, x : \beta \vdash_\Sigma u \equiv u' : \gamma$. u et u' sont normaux dans $\Sigma, \Gamma, \Delta, x : \beta$ donc par hypothèse de récurrence, $u = u'$. On a donc bien $t = t'$.

– Si t est une abstraction non linéaire ou un enregistrement, la preuve est similaire au cas précédent.

– Si $t = u\ v$, alors par le lemme 44 (compatibilité), $t' = u'\ v'$ avec $\Gamma\,;\,\Delta_1 \vdash_\Sigma u \equiv u' : \beta$ et $\Gamma\,;\,\Delta_2 \vdash_\Sigma v \equiv v' : \beta$.

  v et v' sont normaux pour Σ, Γ, Δ_2, et u et u' sont $\beta\gamma c$-normaux, mais pas $\beta\gamma\eta$-normaux pour Σ, Γ, Δ_1. Par l'hypothèse de récurrence, on a donc $u = u'$ et $v = v'$, donc $t = t'$.

– Les autres cas sont similaires.

□

3.2.2 Normalisation faible de $\longrightarrow_\eta$

À présent que nous disposons de l'unicité des formes normales, la technique la plus simple pour obtenir la confluence du calcul est de démontrer sa normalisation faible. Comme on dispose déjà de la normalisation de $\longrightarrow_{\beta\gamma}$, et que $\longrightarrow_\eta$ préserve la $\beta\gamma$-normalité, il suffit donc de prouver la normalisation faible de $\longrightarrow_\eta$, qui fait donc l'objet de cette section.

La définition suivante formalise la notion de normalisabilité.

Définition 21 (Normalisation faible de $\longrightarrow_\eta$)**.**

1. *Un terme* t *est* η*-normalisable (resp. c-normalisable) pour* Σ, Γ, Δ *et* α *si et seulement si* $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta^* (resp. \longrightarrow_c{}^*) t' : \alpha$ *et* t' η*-normal (resp. c-normal) pour* Γ, Δ *et* α.

2. *Un type* α *est* η*-normalisable (resp. c-normalisable) pour* Σ, Γ *et* K *si et seulement si* $\Gamma \vdash_\Sigma \alpha \longrightarrow_\eta^* (resp. \longrightarrow_c{}^*) \alpha' : K$ *et* α' η*-normal (resp. c-normal) pour* Γ, Δ *et* α.

L'objectif des preuves suivantes est donc de démontrer que tout terme typable est η-normalisable. Une première étape est de démontrer que tout terme c-normalisable est η-normalisable. La preuve de cette propriété utilise une récurrence sur la structure du type sous lequel se fait la réécriture. Cependant, cet ordre structurel doit bien être compris comme se prolongeant à travers la déclaration des types enregistrements et variants dans la signature. C'est ce que formalise la définition suivante :

Définition 22 (Ordre structurel sur les types)**.**
On étend l'ordre structurel classique sur les types avec :
Si $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$ *ou* $binding_\Sigma(a) = \{c_1 \textit{ of } \alpha_1 | \ldots | c_n \textit{ of } \alpha_n\}$
Alors $\forall i \in \{1, \cdots, n\}, \alpha_i < a$*.*
Par définition des signatures bien formées, $<$ *reste un ordre bien fondé.*

Le lemme suivant est un lemme intermédiaire utilisant l'hypothèse de récurrence du lemme de passage de la c-normalisabilité à l'η-normalisabilité.

Lemme 47 (Passage de l'η-normalisabilité à $\downarrow$)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha \multimap \beta$*,* t *est c-normalisable pour* $\Sigma, \Gamma, \Delta, \alpha \multimap \beta$ *et la propriété du lemme suivant est vérifiée pour tous les types inférieurs à* $\alpha \multimap \beta$
   Alors $t \downarrow x$ *est* η*-normalisable pour* $\Sigma, \Gamma, \Delta, x : \alpha, \beta$*.*

2. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : (\Pi x : \alpha)\,\beta$*,* t *est c-normalisable pour* $\Sigma, \Gamma, \Delta, (\Pi x : \alpha)\,\beta$ *et la propriété du lemme suivant est vérifiée pour tous les types inférieurs à* $(\Pi x : \alpha)\,\beta$
   Alors $t \downarrow x$ *est* η*-normalisable pour* $\Sigma, \Gamma, , x : \alpha, \Delta, \beta$*.*

3. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : a$*,* $binding_\Sigma(a) = [l_1 : \alpha_1; \ldots; l_n : \alpha_n]$ *et* t *est c-normalisable pour* $\Sigma, \Gamma, \Delta, a$ *et la propriété du lemme suivant est vérifiée pour tous les types inférieurs à* a
   Alors $t \downarrow l_i$ *est* η*-normalisable pour* $\Sigma, \Gamma, \Delta, \alpha_i$*.*

Démonstration. Par récurrence sur la taille de t. On traite le premier cas, les autres étant similaires.

– Si $t = \lambda^0 x : \alpha_1'.u$, $t \downarrow x = u$ et u est η-normalisable pour $\Sigma, \Gamma, \Delta, x : \alpha, \beta$.

– par les lemme 20 (inversion) et 18 (homogénéité des types), t ne peut pas être une abstraction non linéaire.

– Si $t = \{case\ u \text{ of } (c_i\, x_i) \to t_i\}_{i=1}^n$, $t \downarrow x = \{case\ u \text{ of } (c_i\, x_i) \to t_i \downarrow x\}_{i=1}^n$.

$\forall i \in \{1, \cdots, n\}, t_i$ est η-normalisable pour $\Sigma, \Gamma, \Delta, x_i : \alpha_i, \alpha \multimap \beta$, donc par hypothèse de récurrence, $t_i \downarrow x$ est η-normalisable pour $\Sigma, \Gamma, \Delta, x_i : \alpha_i, x : \alpha, \beta$. $\{case\ u \text{ of } (c_i\, x_i) \to t_i \downarrow x\}_{i=1}^n$ est donc c-normalisable pour $\Sigma, \Gamma, \Delta, x : \alpha, \beta$. Par la propriété du lemme suivant, $\{case\ u \text{ of } (c_i\, x_i) \to t_i \downarrow x\}_{i=1}^n$ est donc η-normalisable.

– Si t n'est pas d'une des formes précédentes, alors $t \downarrow x = t\, x$.

x est c-normalisable pour $\Sigma, \Gamma, x : \alpha, \alpha$, donc par la propriété du lemme suivant, x est η-normalisable pour le même environnement.
$t\, x$ est donc c-normalisable pour $\Sigma, \Gamma, \Delta, x : \alpha, \beta$. Par la propriété du lemme suivant, $t\, x$ est η-normalisable pour le même environnement.

□

Lemme 48 (Passage de la c-normalisabilité à l'η-normalisabilité)**.**

1. *Si* t *est c-normalisable pour* Σ, Γ, Δ *et* α *et* $\Gamma \vdash_\Sigma A(\alpha)$
   Alors t *est* η*-normalisable pour* Σ, Γ, Δ *et* α*.*

2. *Si* α *est c-normalisable pour* Σ, Γ *et* $(\beta)\,K$ *et* $\Gamma \vdash_\Sigma A(\beta)$
   Alors α *est* η*-normalisable pour* Σ, Γ *et* $(\beta)\,K$*.*

Démonstration. Par récurrence structurelle sur α dans le premier cas et (β) K dans le second. On traite le premier cas, le second étant similaire.

t est c-normalisable, donc $\exists t'$ c-normal pour Σ, Γ, Δ et α et $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c {}^* t' : \alpha$.

Par le lemme 46 (unicité des formes normales), α est le seul type en forme normale qui peut être assigné à t' dans cet environnement, donc si α n'est pas un type fonctionnel linéaire, un produit dépendant, un type produit ou un type somme, t' est η-normal et t est donc η-normalisable.

Sinon, les différents cas étant similaires, supposons $\alpha = \alpha_1 \multimap \alpha_2$. Si t' est une abstraction, alors de même t' est η-normal et t est η-normalisable. Si t' n'est pas une abstraction, alors on a
$\Gamma\,;\,\Delta \vdash_\Sigma t' \longrightarrow_\eta \lambda^0 x : \alpha_1.(t' \downarrow x) : \alpha$.

Par le lemme 39 (validité des formes syntaxiques), α_1 est η-normal.

Par le lemme précédent et l'hypothèse de récurrence, on sait que $t' \downarrow x$ est η-normalisable.

t est donc bien η-normalisable.

□

Une fois ce lemme établi, la normalisation faible de $\longrightarrow_\eta$ est facilement démontrée par une simple récurrence sur la taille de l'objet :

Lemme 49 (Normalisation faible de $\longrightarrow_\eta$)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$,
   Alors t *est* η*-normalisable dans* $\Sigma, \Gamma, \Delta, \alpha$.
2. *Si* $\Gamma \vdash_\Sigma \alpha : K$,
   alors α *est* η*-normalisable dans* Σ, Γ, K.

Démonstration. Par récurrence sur la taille de t ou de α, et selon la forme syntaxique. Les cas étant similaires, supposons $t = v.l$. Par le lemme 20 (inversion), on a $\Gamma\,;\,\Delta \vdash_\Sigma v : a$, avec $\Gamma \vdash_\Sigma A(a)$. Par hypothèse de récurrence, v est η-normalisable pour $\Sigma, \Gamma, \Delta, a$, donc t est c-normalisable dans $\Sigma, \Gamma, \Delta, \alpha$. Par le lemme précédent, t est donc η-normalisable dans $\Sigma, \Gamma, \Delta, a$.

□

$\longrightarrow_\eta$ est donc faiblement normalisante, et on peut extraire des preuves précédentes une stratégie de normalisation, consistant à expanser toujours au plus profond du terme. De plus, à ce stade de la preuve, la confluence de $\longrightarrow_\eta$ serait aisément prouvable. Il suffirait alors de prouver que chaque étape d' $\longrightarrow_\eta$ fait augmenter une mesure entière pour en déduire la normalisation forte de $\longrightarrow_\eta$, Klop (1992) ayant prouvé que la confluence faible, la normalisation faible et l'augmentation impliquaient la confluence et la normalisation forte. Le caractère augmentant de l'η-expansion est en général immédiat, ce n'est cependant pas le cas ici, à cause de la présence d'$\downarrow$ et des étiquettes de type. La normalisation faible de $\longrightarrow_\eta$ est néanmoins suffisante pour pouvoir tester si deux objets sont bien équivalents.

3.2.3 Normalisation et confluence du calcul

Nous disposons à présent de tous les éléments pour établir les propriétés fondamentales du calcul.
La normalisation faible du calcul est aisément déduite des normalisations de $\longrightarrow_{\beta\gamma}$ et de $\longrightarrow_\eta$, et de la conservation de la $\beta\gamma$-normalité par $\longrightarrow_\eta$:

Théorème 5 (Normalisation faible de $\longrightarrow_{\beta\gamma\eta}$)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma t : \alpha$,
   Alors $\exists t'$ *forme* $\beta\gamma\eta$*-normale pour* $\Sigma, \Gamma, \Delta, \alpha$ *telle que* $\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow^*_{\beta\gamma\eta} t' : \alpha$.

2. *Si* $\Gamma) \vdash_\Sigma \alpha : K$,
   Alors $\exists \alpha'$ *forme* $\beta\gamma\eta$*-normale pour* Σ, Γ, K *telle que* $\Gamma \vdash_\Sigma \alpha \longrightarrow^*_{\beta\gamma\eta} \alpha' : K$.

Démonstration. On traite le premier cas, le second étant similaire :

Par le théorème 2, t est $\beta\gamma$ normalisable vers un terme t'.
Par le lemme 32 (réduction du sujet pour $\longrightarrow_{\beta\gamma}$), $\Gamma\,;\,\Delta \vdash_\Sigma t' : \alpha$.
Par le lemme précédent (normalisation faible de $\longrightarrow_\eta$), t' est η-normalisable dans $\Sigma, \Gamma, \Delta, \alpha$ vers un terme t''.
Par le lemme 41 (conservation de la $\beta\gamma$-normalité par $\longrightarrow_\eta$), t'' est également $\beta\gamma$-normal.

□

Notons que, même dans le cas où la normalisation forte de $\longrightarrow_\eta$ serait acquise, la normalisation forte de $\longrightarrow_{\beta\gamma\eta}$ serait difficile à obtenir, comme en attestent les preuves de Barthe (1999b) et Joachimski (2003).

La normalistion faible suffit néanmoins à prouver directement un résultat plus fort que la confluence de $\longrightarrow_{\beta\gamma\eta}$ ou de $\longrightarrow_{\beta\gamma\eta\approx}$: tous objets équivalents se réduisent vers leur forme normale commune :

Théorème 6 (Propriété de Church-Rosser pour $\equiv$)**.**

1. *Si* $\Gamma\,;\,\Delta \vdash_\Sigma u_1 : \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma u_2 : \alpha$ *et* $\Gamma\,;\,\Delta \vdash_\Sigma u_1 \equiv u_2 : \alpha$*, alors* $\exists u'$ *tel que* $\Gamma\,;\,\Delta \vdash_\Sigma u_1 \longrightarrow^*_{\beta\gamma\eta} u' : \alpha$ *et* $\Gamma\,;\,\Delta \vdash_\Sigma u_2 \longrightarrow^*_{\beta\gamma\eta} u' : \alpha$.
2. *Si* $\Gamma \vdash_\Sigma \alpha_1 : K$, $\Gamma \vdash_\Sigma \alpha_2 : K$ *et* $\Gamma \vdash_\Sigma \alpha_1 \equiv \alpha_2 : K$*, alors* $\exists \alpha'$ *tel que* $\Gamma \vdash_\Sigma u_1 \longrightarrow^*_{\beta\gamma\eta} u' : \alpha$ *et* $\Gamma \vdash_\Sigma u_2 \longrightarrow^*_{\beta\gamma\eta} u' : \alpha$.

Démonstration. On traite le premier cas, le second étant similaire :

Par le théorème5 (normalisation faible pour $\longrightarrow_{\beta\gamma\eta}$), $\exists u'_1, u'_2$ tels que $\Gamma\,;\,\Delta \vdash_\Sigma u_1 \longrightarrow^*_{\beta\gamma\eta} u'_1 : \alpha$, $\Gamma\,;\,\Delta \vdash_\Sigma u_2 \longrightarrow^*_{\beta\gamma\eta} u'_2 : \alpha$ et u'_1, u'_2 sont normaux pour $\Sigma, \Gamma, \Delta, \alpha$.

Par les lemmes 32 et 33 (réduction du sujet pour $\longrightarrow_{\beta\gamma}$ et $\longrightarrow_\eta$), on a $\Gamma\,;\,\Delta \vdash_\Sigma u'_1 : \alpha$ et $\Gamma\,;\,\Delta \vdash_\Sigma u'_2 : \alpha$.

Par le lemme 46 (unicité des formes normales), on a $u'_1 = u'_2$.

□

Ces deux résultats offrent donc une stratégie pour décider si deux objets typés sont équivalents : on normalise chacun des objets, en utilisant la stratégie de normalisation induite par la preuve de normalisation, en appliquant dans un premier temps suffisamment d'étapes $\longrightarrow_{\beta\gamma}$ pour obtenir une forme $\beta\gamma$-normale, puis en appliquant des $\longrightarrow_\eta$ au plus profond du terme jusqu'à obtenir une forme η-normale. Une fois la normalisation effectuée, il suffit de vérifier si les formes normales sont identiques. Cette décidabilité de $\equiv$, cruciale pour utiliser le calcul comme support pour les ACG, reste cependant conditionnée par les deux hypothèses qui sont admises pour pouvoir prouver le lemme de substitution de $\longrightarrow_\eta$. La décomposition des types fonctionnel correspond au *key lemma* que l'on trouve dans les preuves de normalisation et de confluence du λ-cube avec η-expansions données par Ghani (1997) et Barthe (1999a). L'existence des formes normales syntaxiques correspond à une forme de normalisation faible, mais nécessite également de montrer que le critère syntaxique reconnait toute forme normale. Un tel résultat apprait dans la preuve de Barthe. Dans tous ces cas, les preuves de ces propriétés exploitent le fait que la propriété de church-Rosser soit acquise pour $\equiv$, qui est définie en fonction de l'η-réduction et non de l'η-expansion. Une telle approche nous est cependant impossible, puisque l'η-réduction n'engendre pas un calcul confluent en présence d'analyses de cas.

Une solution pour ne pas nécessiter l'hypothèse d'existence des formes normales syntaxiques serait de suivre la proposition de Barthe de demander aux étiquettes introduites par η-expansion d'être en forme $\beta(\gamma)$-normale seulement, plutôt qu'en forme normale syntaxique, en utilisant une règle comme :

$$\frac{t \text{ n'est pas une abstraction linéaire} \qquad \Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_c t' : \alpha \multimap \beta \qquad \alpha \text{ est en forme } \beta\gamma\text{-normale}}{\Gamma\,;\,\Delta \vdash_\Sigma t \longrightarrow_\eta \lambda^0 x : \alpha.(t' \downarrow x) : \alpha \multimap \beta}$$

En effet l'existence des formes $\beta\gamma$-normales est acquise, puisque la normalisation forte de $\longrightarrow_{\beta\gamma}$ a été prouvée au chapitre précédent. Cependant, il faut alors pouvoir modifier la preuve de normalisation faible de $\longrightarrow_\eta$, qui utilise en état le fait que les étiquettes introduites ne contiennent pas d'η-rédex. Si une telle étiquette est seulement en forme $\beta\gamma$-normale, et qu'elle peut donc contenir des η-rédex, il faut disposer d'une hypothèse de récurrence garantissant que cette étiquette reste η-normalisable. Un ordre bien fondé entre les objets du λ-cube

permettant cette récurrence est présenté par Dowek et Werner (1993). Cependant, là encore, le système de type considéré s'appuie sur une relation d'équivalence héritant des propriétés de l'η-réduction, et il n'est pas immédiat de déterminer comment adapter cette preuve de bon fondement. Cette direction semble néanmoins prometteuse pour l'objectif d'obtenir une preuve complète des propriétés de $\longrightarrow_{\beta\gamma\eta}$.

Chapitre 4

Modélisation des dépendances à distance

Dans ce chapitre, nous présentons une possibilité d'application des extensions des ACG présentées dans la section 1.2.5 reposant sur le calcul discuté dans les chapitres 2 et 3, et cette application concerne la modélisation de phénomènes liés aux dépendances à distance dans les langues naturelles. Ces dépendances à distance correspondent aux cas où le comportement d'un constituant de la phrase implique une position se trouvant à une distance arbitraire de ce constituant. Les constructions relatives, que nous avons déjà considérées dans le chapitre 1, en sont un exemple : le pronom relatif peut se trouver à une distance arbitraire de la position laissée vide dans la proposition relative, comme dans une phrase de la forme

L'homme que$_1$ Marie croit que Jean croit que Marie croit que ... que Jean croit que Marie connaît t_1 arrive.

La notation t_1 indique la position où un élément est manquant pour que la proposition relative constitue une phrase complète. On appelle cette position la *trace* associée au pronom relatif, et on indice les deux par le même entier pour figurer ce lien. Dans les grammaires décrivant un niveau syntaxique abstrait de formes logiques, comme le programme minimaliste de Chomsky (1995), une construction relative est interprétée comme un mouvement grammatical : au niveau de la forme logique, le constituant complété, ici "l'homme", apparaît dans l'interprétation de la proposition relative, alors qu'il en a disparu au niveau de la forme de surface. Comme ce mouvement est visible dans la forme de surface syntaxique, on le qualifie d'*explicite*.

Une dépendance à distance peut également se manifester au niveau de surface sémantique, à l'exemple du phénomène de prise de portée des quantificateurs. Ainsi la phrase

toute femme aime un homme

est habituellement considérée comme possédant les deux lectures suivantes, selon que *toute femme* prenne portée au-dessus de *un homme* (lecture 1) ou non (lecture 2) :

1. $\forall x \exists y.homme(y) \wedge (femme(x) \Rightarrow aime(x, y))$;
2. $\exists y.homme(y) \wedge \forall x(femme(x) \Rightarrow aime(x, y))$.

Une telle prise de portée peut également se faire à une distance arbitraire, et on la qualifie de mouvement grammatical implicite, car elle n'est pas visible au niveau de surface syntaxique.

Les Grammaires Catégorielles Abstraites peuvent aisément rendre compte de ces dépendances à distance grâce à la λ-abstraction, permettant, au niveau abstrait, de traiter la trace d'un quantificateur ou la position syntaxique d'un quantificateur comme une variable liée gouvernée par le terme abstrait d'ordre supérieur correspondant au constituant. Cependant, ces dépendances à distance sont sujettes à de nombreuses contraintes, et ce mécanisme d'extraction doit donc être contrôlé d'une manière ou d'une autre. Comme synthétisé par Ruys et Winter (2011), les contraintes s'appliquant aux mouvements explicites et implicites sont étroitement liées, et il semble donc intéressant de les traiter de façon homogène, au niveau abstrait. C'est cette approche que nous nous proposons d'explorer ici.

Dans une première partie nous détaillerons l'approche des ACG pour les mouvements grammaticaux, et nous donneront un exemple d'interface syntaxe-sémantique permettant de tels mouvements. Dans une seconde partie,

nous discuterons quelques contraintes s'appliquant à ces mouvements grammaticaux, et nous montrerons comment elles peuvent être capturées par une ACG contrôlant le niveau abstrait de cette interface syntaxe-sémantique, et utilisant les extensions définies dans les chapitres précédents.

4.1 Les mouvements grammaticaux dans les ACG

4.1.1 Mouvements explicites

Comme nous l'avons vu dans la section 1.2.2, les ACG peuvent traiter un pronom relatif en lui affectant un type lui permettant d'être combiné d'une part au groupe nominal qu'il complète et d'autre part à sa proposition relative, qui est une phrase dans laquelle manque un constituant. Comme le calcul des ACG dispose d'une λ-abstraction linéaire, une proposition relative peut toujours être représenté au niveau abstrait comme une fonction retournant un terme de type *s*, quelle que soit la position dans laquelle manque un constituant. Ainsi, à la proposition relative *Marie aime passionnément* correspond le terme abstrait $(\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))$, de type $np \multimap s$, qui peut donc être passé en argument au terme abstrait C_{que}, de type $(np \multimap s) \multimap np \multimap np$. La trace du mouvement apparaît donc au niveau abstrait comme une variable liée. Dans la suite, on désignera cette variable comme la variable *extraite*, ou comme la variable *susceptible d'être extraite*. La phrase

Jean que Marie aime passionnément marche

correspond donc au terme abstrait

$C_{\text{marche}}(C_{\text{que}}(\lambda^0 x.(C_{\text{passionnément}}\ (C_{\text{aime}}\ x)\ C_{\text{Marie}}))C_{\text{Jean}})$.

On observe qu'au niveau de surface syntaxique, la trace reste vide. Ce la signifie donc que dans le lexique syntaxique, l'image du terme abstrait C_{que} doit passer le mot vide ϵ en argument de la proposition relative, notée p dans :

$C_{\text{que}} :=_{\text{Syn}} \lambda^0 pn.(n + /\text{que}/ + (p\ \epsilon))$.

En revanche, au niveau de surface sémantique la position de la trace doit être occupée par la sémantique du groupe nominal complété, qui doit donc être passé en argument à la proposition relative. La sémantique du groupe nominal complété est également utilisée par la proposition principale, et ne peut pas être dupliquée car elle est argument d'une implication linéaire. Pour dépasser cette limitation, nous utilisons une variable existentielle non-linéaire, passée à la proposition principale, à la proposition relative, et également posée comme égale à la sémantique du groupe nominal complété :

$C_{\text{que}} :=_{\text{Sem}} \lambda^0 pnx.\exists y.(p\ (\lambda^0 r.r\ y)) \wedge (x\ y) \wedge (n\ (\lambda^0 z.z = y))$.

Cette image sémantique suppose, pour pouvoir accéder à la proposition principale, ici la variable x, que le type sémantique correspondant à *np* soit $(e \multimap t) \multimap t$.

Les pronoms interrogatifs peuvent être traités de manière similaire. Nous n'en donnons pas les détails pour ne pas alourdir l'interface syntaxe-sémantique avec les structures interrogatives.

4.1.2 Mouvements implicites

Les opérateurs in situ comme les quantificateurs ont la propriété de prendre sémantiquement portée sur des expressions complexes de surface auxquelles ils appartiennent. Dans (1) par exemple, le groupe nominal quantifié (QNP) *quelqu'un*, bien que partie de la phrase complète, voit la quantification existentielle de sa contribution sémantique prendre portée sur la proposition entière, comme apparait dans (1-a).

(1) Marie aime quelqu'un
   a. $\exists x.\textbf{aime m}\ x$

b. $C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{aime}}\, x\, C_{\text{Marie}})$

La façon dont ce phénomène est traité dans les grammaires catégorielles est d'attribuer aux QNP le type d'ordre supérieur $(np \multimap s) \multimap s$, dont l'argument est une phrase à laquelle il manque un NP. Similairement au cas des mouvements explicites, un tel argument peut correspondre à un λ-terme commençant par une abstraction $\lambda^0 x.u$ avec x apparaissant (libre) dans u jouant le rôle de n'importe quel NP non quantifié ayant la position de surface du QNP. Ainsi, dans le premier exemple, u représenterait l'expression *Marie aime* x, et la représentation de (1) est (1-b).

Ce traitement rend bien compte des ambiguités de portée entre QNP, puisque ces termes d'ordres supérieurs peuvent être appliqués à la phrase dans un ordre décorrélé de leur position de surface. Ainsi la phrase (2) bénéficie des deux lectures (2-a) et (2-b).

(2) Tout le monde aime quelqu'un

a. $C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{tout le monde}}(\lambda^0 y.C_{\text{aime}} x\ y))$
$\exists x.\forall y.\textbf{aime}\, x\, y$

b. $C_{\text{tout le monde}}(\lambda^0 y.C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{aime}} x\ y))$
$\forall y.\exists x.\textbf{aime}\, x\, y$

Le propre des mouvements implicites est qu'ils n'apparaissent pas au niveau de surface syntaxique, la trace du mouvement étant occupée par la forme de surface du QNP. Dans le lexique syntaxique un QNP doit donc donner sa réalisation syntaxique en argument à la phrase :

$C_{\text{quelqu'un}} :=_{\text{Syn}} \lambda^0 s.s$ /quelqu'un/.

Au niveau de surface sémantique, la position de la trace est occupée par une variable liée par la quantification du QNP :

$C_{\text{quelqu'un}} :=_{\text{Sem}} \lambda^0 s.\exists x.s\ (\lambda^0 r.r\ x)$.

4.1.3 Une interface syntaxe-sémantique traitant les mouvements

Nous sommes à présent en mesure de définir une interface syntaxe-sémantique d'un fragment de la langue française utilisant les traitements des mouvements grammaticaux présentés précédemment. Comme nous l'avons vu dans la section 1.2.4, cela consiste à définir deux ACG, l'une traitant la syntaxe et l'autre la sémantique, et les deux partageant le même niveau abstrait. Ces ACG utilisent les définitions de la section 1.2.1, à l'exception de la signature du niveau sémantique, dans laquelle le type des constantes correspondant aux quantificateurs $\exists$ et $\forall$ nécessitent l'implication non-linéaire de l'extension de la section 1.2.5.

Nous n'intégrons pas d'articles, définis ou indéfinis, à cette grammaire jouet, car les problématiques de sémantique qui y sont rattachées dépassent l'objet de notre propos. L'article indéfini un peut former des groupes nominaux quantifiés, mais ceux-ci semblent subir moins de contrainte de portée que les autres QNP, et constituent donc des exemples moins probants pour les phénomènes que nous entendons traiter. Nous utiliserons à la place les QNP *quelqu'un* et *tout le monde*, chacun considéré comme une unité lexicale par soucis de simplicité. De la même façon, nous traiterons *dit que* comme une unité lexicale.

On définit $\mathscr{G}_{\text{Syn}} = \langle \Sigma_{\text{Syn}}, \Sigma_{\text{String}}, \mathscr{L}_{\text{Syn}}, s\rangle$ l'ACG qui relie les structures syntaxiques à leur réalisation de surface. La table 4.1 presente Σ_{Syn} la signature des structures syntaxiques, Σ_{String} la signature des réalisations de surface, et $\mathscr{L}_{\text{Syn}}$ le lexique qui les relie. Les détails de l'encodage des chaînes de caractères ont été donnés dans la section 1.2.2, et nous utiliserons ici directement $string$ pour le type des chaînes de caractères, $+$ pour leur opération de concaténation et ϵ pour la chaîne vide.

On définit également $\mathscr{G}_{\text{Sem}} = \langle \Sigma_{\text{Syn}}, \Sigma_{\text{Sem}}, \mathscr{L}_{\text{Sem}}, s\rangle$ l'ACG qui relie les structures syntaxiques à leur interprétation *sémantique*. Comme annoncé, $\mathscr{G}_{\text{Syn}}$ et $\mathscr{G}_{\text{Sem}}$ partagent le même vocabulaire abstrait Σ_{Syn} présenté dans la table 4.1. La table 4.2 presente Σ_{Sem} la signature logique des formules logiques et $\mathscr{L}_{\text{Sem}}$ le lexique qui relie Σ_{Syn} à Σ_{Sem}. Ce lexique associe (1-b) avec sa sémantique (1-a). Afin d'alléger les notations, nous écrivons $\forall x.t$ pour $\forall(\lambda x.t)$.

De la même façon que dans la section 1.1.3, le comportement sémantique des pronoms relatifs nécessite de

Σ_{Syn} :

np, s	: **type**		
$C_{\text{Marie}}, C_{\text{Jean}}$	: np	C_{aime}	: $np \multimap np \multimap s$
C_{marche}	: $np \multimap s$	$C_{\text{passionnément}}$	: $(np \multimap s) \multimap (np \multimap s)$
$C_{\text{qui}}, C_{\text{que}}$	: $(np \multimap s) \multimap np \multimap np$	$C_{\text{quelqu'un}}, C_{\text{tout le monde}}$	: $(np \multimap s) \multimap s$
$C_{\text{dit que}}$	: $s \multimap np \multimap s$		

Σ_{String} :

/Marie/, /Jean/, /aime/, /marche/, /passionnément/,
/qui/, /que/, /quelqu'un/, /tout le monde/, /dit que/ : $string$

$\mathscr{L}_{\text{Syn}}$:

s, np	$:=_{\text{Syn}}\ string$		
C_{Marie}	$:=_{\text{Syn}}$ /Marie/	C_{Jean}	$:=_{\text{Syn}}$ /Jean/
$C_{\text{quelqu'un}}$	$:=_{\text{Syn}}\ \lambda^0 p.p$ /quelqu'un/	$C_{\text{tout le monde}}$	$:=_{\text{Syn}}\ \lambda^0 p.p$ /tout le monde/
C_{aime}	$:=_{\text{Syn}}\ \lambda^0 os.s +$ /aime/ $+ o$	C_{marche}	$:=_{\text{Syn}}\ \lambda^0 s.s +$ /marche/
C_{qui}	$:=_{\text{Syn}}\ \lambda^0 pn.n +$ /qui/ $+ (p\,\epsilon)$	C_{que}	$:=_{\text{Syn}}\ \lambda^0 pn.n +$ /que/ $+ (p\,\epsilon)$
$C_{\text{dit que}}$	$:=_{\text{Syn}}\ \lambda^0 cs.s +$ /says/ $+ c$	$C_{\text{passionnément}}$	$:=_{\text{Syn}}\ \lambda^0 v.v +$ /passionnément/

TAB. 4.1 – Σ_{Syn}, Σ_{String} et $\mathscr{L}_{\text{Syn}}$

faire correspondre le type *np* au type complexe $(e \multimap t) \multimap t$, afin qu'ils puissent accéder au prédicat s'appliquant au groupe nominal qu'ils complètent.

Σ_{Sem} :

e, t	: **type**	$\mathbf{m}, \mathbf{j}$	$: e$	$\forall, \exists$	$: (e \to t) \multimap t$
$\wedge, \Rightarrow$	$: t \multimap t \multimap t$	**aime**	$: e \multimap e \multimap t$	**dit que**	$: t \multimap e \multimap t$
marche	$: e \multimap t$	**passionnément**	$: (e \multimap t) \multimap (e \multimap t)$	$=$	$: e \multimap e \multimap t$

$\mathscr{L}_{\text{Sem}}$:

s	$:=_{\text{Sem}}\ t$
np	$:=_{\text{Sem}}\ (e \multimap t) \multimap t$
C_{Jean}	$:=_{\text{Sem}}\ \lambda^0 p.p\ \mathbf{j}$
C_{Marie}	$:=_{\text{Sem}}\ \lambda^0 p.p\ \mathbf{m}$
$C_{\text{quelqu'un}}$	$:=_{\text{Sem}}\ \lambda^0 p.\exists x.p\ (\lambda^0 r.r\ x)$
$C_{\text{tout le monde}}$	$:=_{\text{Sem}}\ \lambda^0 p.\forall x.p\ (\lambda^0 r.r\ x)$
C_{marche}	$:=_{\text{Sem}}\ \lambda^0 s.s(\lambda^0 y.\mathbf{marche}\ y)$
C_{aime}	$:=_{\text{Sem}}\ \lambda^0 os.o(\lambda^0 x.s(\lambda^0 y.\mathbf{aime}\ x\ y))$
$C_{\text{qui}}, C_{\text{que}}$	$:=_{\text{Sem}}\ \lambda^0 pnx.\exists y.(p\ (\lambda^0 r.r\ y)) \wedge (x\ y) \wedge (n\ (\lambda^0 z.z = y))$
$C_{\text{dit que}}$	$:=_{\text{Sem}}\ \lambda^0 cs.s(\lambda^0 y.\mathbf{dit\ que}\ c\ y)$
$C_{\text{passionnément}}$	$:=_{\text{Sem}}\ \lambda^0 vs.s(\lambda^0 y.\mathbf{passionnément}\ (v\ (\lambda^0 p.p\ y)))$

TAB. 4.2 – Σ_{Sem} et $\mathscr{L}_{\text{Sem}}$

Le traitement de la quantification et de la relativisation effectué par cette interface syntaxe-sémantique présente un défaut de surgénération, dans le sens où des termes abstrait de type *s* correspondent à des chaînes de caractères qui ne sont pas des phrases grammaticalement correctes. En effet, en construisant un terme contenant des variables libres, ces variables peuvent se trouver à une profondeur arbitraire à l'intérieur du terme, et peuvent être abstraites dans n'importe quel ordre (ce qui provoque notamment les ambiguités de portée). À l'inverse, les langues naturelles ne sont pas aussi libre de ce point de vue, et se pose ainsi la question d'observer ces contraintes et d'introduire un contrôle pour en tenir compte, ce que nous examinons dans la partie suivante.

4.2 Modélisation de contraintes de mouvements

Dans cette section nous considérons plusieurs problématiques de contraintes des dépendances à distance, et pour chacune d'entre elles nous proposons une ACG de contrôle permettant de la modéliser. Suivant les remarques de la section 1.2.4, une telle ACG de contrôle a pour niveau objet le niveau abstrait de l'interface syntaxe-sémantique décrite à la section précédente. Cette architecture permet d'appliquer ces mécanismes de contrôle des extraction de façon modulaire, sans avoir à remodifier à chaque fois l'interface syntaxe-sémantique. Ces ACG de contrôle utilisent toutes les extensions présentées dans la section 1.2.5 : types enregistrement, variants et produit dépendant. Mise à part cette modification du système de typage, les définitions de ces ACG sont similaires à celles présentées dans la section 1.2.1.

Le principe général de notre approche est le suivant : les opérateurs qui déclenchent des extractions ont un type suivant le modèle $(\alpha \multimap \beta) \multimap \gamma$. Cependant, tous les éléments de type α ne peuvent pas être extraits. par exemple, si α est *np*, il peut être nécessaire que ce soit un mominatif, ou un accusatif. ces contraintes sont prises en compte en utilisant des structures de traits (qui correspondent ici au produit dépendant) sur les types syntaxiques.

Ce n'est cependant pas suffisant, car β peut lui-même introduire des contraintes supplémentaires. Par exemple, si β est *s*, l'extraction peut parfois être contrainte par le fait qu'il ne doit pas y avoir une autre extraction en cours. Cette contrainte peut s'exprimer en utilisant des traits sur *s*.

Enfin, il peut arriver que certaines combinaisons sur les contraintes de α et β ne soient pas toutes possibles, ce qui signifie que les contraintes d'extraction sont exprimées par une *relation*, qui ne se réduit pas au produit cartésien, entre les différents traits. Par exemple, l'extraction d'un sujet par un pronom relatif n'est possible que s'il s'agit du sujet de la proposition relative. Les types dépendant permettent d'implémenter de telles relations. Cette approche partage de nombreuses similarités avec l'usage fait dans Moot et Piazza (2001) de la logique linéaire du premier ordre où les variables du premier ordre implémentent également des types de relations entre constituants.

4.2.1 Contraintes des cas

En langue française, l'extraction par les pronoms relatifs "qui" et "que" (ainsi que leur équivalent interrogatif) dépend du rôle grammatical de la position extraite. Ainsi, "qui" peut seulement extraire un groupe nominal en position de sujet, et non d'objet ou de complément, comme le montrent les exemples (3) et (4).

(3) Marie qui$_1$ aime Jean t_1 marche

(4) *Marie qui$_1$ jean aime t_1 marche

Pour bloquer ce type d'extraction, il suffit de reprendre le principe des structures de traits de la section 1.2.5, en indiquant dans le type *np* le rôle (ou cas) du constituant en question, plutôt que son genre ou son nombre. Un cas *nom* (pour nominatif) est alors requis par le type des verbes transitifs pour leur second argument, qui correspond bien d'après le lexique syntaxique au sujet d'un tel verbe. Ce cas apparaît également dans le type de la constante C_{qui}, affecté à la variable extraite par le pronom relatif. Cette variable ne peut donc être utilisée que dans une position de sujet. Le niveau abstrait de cette ACG de contrôle reprend donc les types et constantes de Σ_{Syn}, complétés par ces structures de traits. Le niveau objet correspondant à Σ_{Syn}, le lexique de l'ACG de contrôle se contente de faire disparaître ces structures de traits. Cette ACG de contrôle $\mathscr{G}_{\text{Cont}_1} = \langle \Sigma_{\text{Cont}_1}, \Sigma_{\text{Syn}}, \mathscr{L}_{\text{Cont}_1}, s \rangle$ est définie dans la table 4.3.

(3) a pour antécédent par $\mathscr{L}_{\text{Syn}}$ le terme

$C_{\text{marche}}\ (C_{\text{qui}}\ (C_{\text{aime}}\ C_{\text{jean}})\ C_{\text{Marie}})$.

Ce terme a un antécédent par $\mathscr{L}_{\text{Cont}_1}$, le terme :

$D_{\text{marche}}\ ((D_{\text{qui}}\ nom\ (D_{\text{aime}}\ (D_{\text{jean}}\ acc))\ (D_{\text{Marie}}\ nom))\ nom)$

de type *s*. (3) est donc accepté par notre grammaire.
À l'inverse, l'antécédent de (4) :

Σ_{Cont_1} :

cas, s	: **type**
np	: (cas) **type**
nom, acc	: cas
$D_{\text{Marie}}, D_{\text{Jean}}$	: $(\Pi x : cas)\, \mathit{np}\ x$
D_{aime}	: $\mathit{np}\ acc \multimap \mathit{np}\ nom \multimap s$
D_{marche}	: $\mathit{np}\ nom \multimap s$
$D_{\text{passionnément}}$	: $(\mathit{np}\ nom \multimap s) \multimap (\mathit{np}\ nom \multimap s)$
D_{qui}	: $(\Pi x : cas)\, (\mathit{np}\ nom \multimap s) \multimap \mathit{np}\ x \multimap \mathit{np}\ x$
D_{que}	: $(\Pi x : cas)\, (\mathit{np}\ acc \multimap s) \multimap \mathit{np}\ x \multimap \mathit{np}\ x$
$D_{\text{dit que}}$	: $s \multimap \mathit{np}\ nom \multimap s$
$D_{\text{quelqu'un}}, D_{\text{tout le monde}}$	: $(\Pi x : cas)\, (\mathit{np}\ x \multimap s) \multimap s$

$\mathscr{L}_{\mathrm{Cont}_1}$:

s	$:=_{\mathrm{Cont}_1}$ *s*
np	$:=_{\mathrm{Cont}_1}$ $\lambda x.\, \mathit{np}$
D_{T}	$:=_{\mathrm{Cont}_1}$ $\lambda x.C_{\mathrm{T}}$ si T correspond à Marie, Jean, qui, que, quelqu'un ou tout le monde
D_{T}	$:=_{\mathrm{Cont}_1}$ C_{T} sinon

TAB. 4.3 – Σ_{Cont_1} et $\mathscr{L}_{\mathrm{Cont}_1}$

$C_{\text{marche}}\ (C_{\text{qui}}\ (\lambda^0 x.C_{\text{aime}}\ x C_{\text{jean}})\ C_{\text{Marie}})$

n'a lui-même pas d'antécédent typable par $\mathscr{L}_{\mathrm{Cont}_1}$. En effet D_{qui} impose à la variable x d'être de type $\mathit{np}\ nom$, et D_{aime} impose à cette même variable d'être de type $\mathit{np}\ acc$. (4) est donc correctement rejeté par notre grammaire.

4.2.2 Les propositions comme îlots d'extraction

(5) est un deuxième exemple de contrainte sur les dépendances à distance. Il est en effet classique de considérer que dans une telle phrase, le groupe nominal quantifieur "tout le monde" ne devrait pas prendre portée sur "quelqu'un", ou même sur "dit que", comme dans (5-b) et (5-c) : le groupe nominal quantifieur "tout le monde" ne peut pas prendre portée au-delà de sa proposition.

(5) Quelqu'un dit que tout le monde aime Marie
- a. $C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{dit que}}\ (C_{\text{tout le monde}}(\lambda^0 y.C_{\text{aime}}\ C_{\text{Marie}}\ y))\ x)$
  $\exists x.$**dit que** $x\ (\forall y.$**aime** y **m**$)$
- b. $*C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{tout le monde}}(\lambda^0 y.C_{\text{dit que}}\ (C_{\text{aime}}\ C_{\text{Marie}}\ y)\ x))$
  $*\exists x.\forall y.$**dit que** $x\ ($**aime** y **m**$)$
- c. $*C_{\text{tout le monde}}(\lambda^0 y.C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{dit que}}\ (C_{\text{aime}}\ C_{\text{Marie}}\ y)\ x))$
  $*\forall y.\exists x.$**dit que** $x\ ($**aime** y **m**$)$

D'un point de vue linguistique, le caractère strict d'une telle contrainte est discutable, et nous n'avons pas pour objectif de juger de sa pertinence, mais seulement de montrer comment ce type d'extraction peut être contrôlé dans les ACG.

Le fait qu'un groupe nominal quantificateur ne puisse pas prendre portée au-delà de sa proposition signifie qu'à chaque fois qu'une proposition est passée en argument à un verbe comme "dit que", elle ne devrait plus contenir aucune variable libre susceptible d'être extraite par un QNP. Pour modéliser cela, on étiquette les types *s* et *np* avec pour trait un entier qui indique le nombre courant de variables libres correspondant à une extraction par un QNP. Il faut donc que chaque constante de type *np* introduite par la signature soit décoré par 0. Également, il faut que les variables correspondant à une extraction par un pronom relatif soit aussi décorée par 0. Pour assurer cette propriété, il suffit que le type du premier argument d'un pronom relatif soit de la forme $\mathit{np}\ 0 \multimap \ldots$. À l'inverse, les variables correspondant à l'extraction par un QNP doivent apparaître étiquetée par 1, raison pour laquelle le type de l'argument d'un QNP doit être de la forme $\mathit{np}\ 1 \multimap \ldots$. Si ces entiers sont sommés à chaque fois que plusieurs constituants sont combinés, et que les QNP font décroître le total lorsque l'extraction est réalisée, par leur type

$(\mathit{np}\ 1 \multimap s\ i+1) \multimap \mathit{si}$, une proposition ne véhicule pas d'extractions non résolues par un QNP si et seulement si elle est de type $s0$. C'est donc le type qui doit être requis pour le premier argument du verbe "dit que", de type $(\Pi i : int)\ (s\,0 \multimap \mathit{np}\,i \multimap s\,i)$, ou du pronom relatif "qui", de type $(\Pi i : int)\ (\mathit{np}\ 0 \multimap s\ 0) \multimap \mathit{np}\ i \multimap \mathit{np}\ i$.

Pour un type α fixé, les entiers naturels peuvent être représentés en λ-calcul par l'encodage associant à n le terme $\lambda f : \alpha \multimap \alpha.\lambda^0 x : \alpha.f^n\ x$. Dans la suite, nous utiliserons les notations suivantes :

$$\begin{array}{ll} 0 & := \lambda f : \alpha \multimap \alpha.\lambda^0 x : \alpha.x \\ 1 & := \lambda f : \alpha \multimap \alpha.\lambda^0 x : \alpha.f\ x \\ int & := (\alpha \multimap \alpha) \to \alpha \multimap \alpha \\ + & := \lambda^0 mn : int.\lambda f : \alpha \multimap \alpha.\lambda^0 x : \alpha.m\ f\ (n\ f\ x) \end{array}$$

et nous noterons $+$ en position infixe par soucis de clarté. Il est suffisant de noter que cet encodage capture les propriétés usuelles de l'addition sur les entiers naturels, l'encodage de $+$ appliqué aux encodages de n et de m se β-réduisant vers l'encodage de $n+m$.

Comme dans l'exemple précédent, ce contrôle utilisant une structure de trait est implémenté à un niveau abstrait supérieur par la signature Σ_{Cont_2}, traitant uniquement cette problématique des îlots d'extraction (les cas de la section précédente n'y apparaissent donc pas), et décrite dans la table 4.4. On définit également $\mathscr{G}_{\mathrm{Cont}_2} = \langle \Sigma_{\mathrm{Cont}_2}, \Sigma_{\mathrm{Syn}}, \mathscr{L}_{\mathrm{Cont}_2}, s\,0 \rangle$, l'ACG qui réalise le contrôle sur les structures syntaxiques. $\mathscr{L}_{\mathrm{Cont}_2}$ (Table 4.4) ne fait que supprimer le produit dépendant et transforme Σ_{Cont_2} en Σ_{Syn}.

Σ_{Cont_2} :

α	: **type**
s, *np*	: (int) **type**
D_{Marie}, D_{Jean}	: *np* 0
D_{aime}	: $(\Pi i, j : int)\ (\mathit{np}\,i \multimap \mathit{np}\,j \multimap s\,(i+j))$
D_{marche}	: $(\Pi i : int)\ \mathit{np}\ i \multimap s\ i$
$D_{\text{passionnément}}$	:: $(\Pi i : int)\ (\mathit{np}\ i \multimap s\ i) \multimap (\mathit{np}\ i \multimap s\ i)$
D_{qui}, D_{que}	: $(\Pi i : int)\ (\mathit{np}\ 0 \multimap s\ 0) \multimap \mathit{np}\ i \multimap \mathit{np}\ i$
$D_{\text{dit que}}$	: $(\Pi i : int)\ (s\,0 \multimap \mathit{np}\,i \multimap s\,i)$
$D_{\text{quelqu'un}}$, $D_{\text{tout le monde}}$	: $(\Pi i : int)\ ((\mathit{np}\,1 \multimap s\,(i+1)) \multimap s\,i)$

$\mathscr{L}_{\mathrm{Cont}_2}$:

s	$:=_{\mathrm{Cont}_2} \lambda x.\, s$
np	$:=_{\mathrm{Cont}_2} \lambda x.\, \mathit{np}$
D_{aime}	$:=_{\mathrm{Cont}_2} \lambda xy.C_{\text{aime}}$
D_{T}	$:=_{\mathrm{Cont}_2} C_{\text{T}}$ si T correspond à Marie ou Jean
D_{T}	$:=_{\mathrm{Cont}_2} \lambda x.C_{\text{T}}$ sinon

TAB. 4.4 – Σ_{Cont_2} et $\mathscr{L}_{\mathrm{Cont}_2}$

Le nombre d'extractions par QNP est donc inscrit dans le type de chaque constituant. Les types attribués dans la signature permettent de calculer ce nombre d'extractions lorsque des constituants se combinent. Ainsi un verbe transitif comme D_{aime} a comme trait la somme des traits de ses arguments. Puisque la proposition argument de $D_{\text{dit que}}$ doit être étiquetée par 0, toutes les variables quantifiées doivent avoir déjà retrouvé leur opérateur prenant portée avant que le terme ne soit passé en argument, ce qui les empêche effectivement de franchir l'îlot d'extraction.

(6) est un terme bien typé (de type $s\,0$) de $T(\Sigma_{\mathrm{Cont}_2})$. Il a la même structure que (5-a) qui est en effet son image par $\mathscr{L}_{\mathrm{Cont}_2}$. A l'inverse, dans (7) la variable y ne peut être que de type *np* 0, à cause de la contrainte induite par $D_{\text{dit que}}$. Le type $\mathit{np}\,0 \multimap s\,0$ de (7) (qui correspondrait au sous-terme de (5-c)) l'empêche alors d'être l'argument de $D_{\text{tout le monde}}$. (5-c) est donc rejeté par la grammaire.

(6) $D_{\text{quelqu'un}}\ 0\ (\lambda^0 x : np\ 1.\ D_{\text{dit que}}\ 1\ (D_{\text{tout le monde}}\ 0\ \overbrace{(\lambda^0 y : np\ 1.\ D_{\text{aime}}\ 0\ 1\ D_{\text{Marie}}\ \overbrace{y}^{np\ 1})}^{s\ 1}\ \overbrace{x}^{np\ 1})$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{np\ 1\multimap s\ 1}$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{s\ 0}$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{np\ 1\multimap s\ 1}$

(7) $\lambda^0 y : np\ 0.\ D_{\text{quelqu'un}}\ 0\ \overbrace{(\lambda^0 x : np\ 1.\ D_{\text{dit que}}\ 1\ (D_{\text{aime}}\ 0\ 0\ D_{\text{Marie}}\ \overbrace{y}^{np\ 0})\ \overbrace{x}^{np\ 1})}^{s\ 0}$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{np\ 1\multimap s\ 1}$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{s\ 0}$

Par ailleurs, ce niveau de contrôle n'empêche pas un pronom relatif d'opérer une extraction à travers la portée d'un QNP. Ainsi le terme (8) , qui correspond à la phrase "Marie qui aime quelqu'un marche", est-il bien typable.

(8) $D_{\text{marche}}\ 0\ (D_{\text{qui}}\ 0\ \overbrace{(\lambda^0 x : np\ 0.D_{\text{quelqu'un}}\ 0\ \underbrace{(\lambda^0 y : np\ 1.D_{\text{aime}}\ 1\ 0\ y\ x)}_{np\ 1\multimap s\ 1})}^{np\ 0\multimap s\ 0}\ D_{\text{aime}})$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{s\ 0}$

Ce traitement s'étend aisément aux autres possibilités de former des propositions, comme les propositions conditionnelles, et de manière générale aux autres possibilités de former des îlots d'extraction. Il suffit que dans la signature la constante prenant cet îlot en argument spécifie que le nombre d'extractions de QNP qui y est rattaché soit 0.

Les exemples suivants fonctionnent sur le même principe : chaque type dépend d'une structure de trait qui exprime quelles sont les variables libres présentes dans le terme susceptibles de correspondre à une extraction. Chaque phénomène est traité par une ACG de contrôle indépendante des autres, ce qui assure que le traitement est entièrement modualaire et que les différents mécanismes n'interfèrent pas entre eux.

4.2.3 Îlots d'extraction pour les pronoms relatifs

Nous nous intéressons à présent aux extractions dans les propositions relatives, pour lesquelles il faut faire une distinction entre les extraction de sujet et les autres. Ces deux types d'extraction partagent certains îlots d'extraction, comme les propositions relatives formées par d'autres pronoms, comme l'illustre les exemples suivants :

(9) *Marie qui$_1$ Jean que$_2$ t_1 aime t_2 marche dort

(10)*Marie que$_1$ Jean qui$_2$ t_2 aime t_1 marche dort

En revanche, les extractions de sujet sont les seules à être bloquées par une construction de la forme *dit que*, comme l'illustre les exemples suivants :

(11) *Paul qui$_1$ Jean dit que t$_1$ aime Marie marche
*$C_{\text{marche}}\ (C_{\text{qui}}(\lambda^0 x.C_{\text{dit que}}\ (C_{\text{aime}}\ C_{\text{Marie}}\ x)\ C_{\text{Jean}})\ C_{\text{Paul}})$

(12) Paul que$_1$ Jean dit que Marie aime t$_1$ marche
$C_{\text{marche}}\ (C_{\text{que}}(\lambda^0 x.C_{\text{dit que}}\ (C_{\text{aime}}\ x\ C_{\text{Marie}})\ C_{\text{Jean}})\ C_{\text{paul}})$

Les propositions relatives étant des îlots dans tous les cas, il n'est pas possible pour un terme acceptable de posséder plus d'une variable libre correspondant à l'extraction par un pronom relatif. Il n'est donc pas nécessaire d'utiliser en compteur un entier non borné pour garder trace des extractions, et on utilisera plutôt un type possédant 3 valeurs qui distingue :

- l'absence d'extraction : non,
- l'existence d'une extraction dans la proposition courante : pro,
- l'existence d'une extraction distante (dépassant un *dit que*) : dis.

La nouvelle signature abstraite est donnée dans Table 4.5. L'ACG correspondante $\mathscr{G}_{\mathrm{Cont}_3} = \langle \Sigma_{\mathrm{Cont}_3}, \Sigma_{\mathrm{Syn}}, \mathscr{L}_{\mathrm{Cont}_3}, s\ \mathsf{non}\rangle$ est construite d'une façon similaire à l'exemple précédent.

Σ_{Cont_3} :

$\mathit{extraction} = \{\mathsf{non} \mid \mathsf{pro} \mid \mathsf{dis}\}$	: **type**
s, np	: $(\mathit{extraction})$ **type**
$D_{\text{Marie}}, D_{\text{Jean}}, D_{\text{Paul}}$	: $np\ \mathsf{non}$
D_{aime}	: $(\Pi x, y : \mathit{extraction})\ (np\, x \multimap np\, y \multimap s\, (f\ x\ y))$
D_{marche}	: $(\Pi x : \mathit{extraction})\ np\ x \multimap s\ x$
$D_{\text{passionnément}}$	:: $(\Pi x : \mathit{extraction})\ (np\ x \multimap s\ (f\ x\ \mathsf{non})) \multimap (np\ x \multimap s\ (f\ x\ \mathsf{non}))$
D_{qui}	: $(\Pi x : \mathit{extraction})\ (np\ \mathsf{pro} \multimap s\ \mathsf{pro}) \multimap np\ x \multimap np\ x$
D_{que}	: $(\Pi x : \mathit{extraction})\ (np\ \mathsf{pro} \multimap s\ \mathsf{pro}) \multimap np\ x \multimap np\ x$
D_{que}	: $(\Pi x : \mathit{extraction})\ (np\ \mathsf{pro} \multimap s\ \mathsf{dis}) \multimap np\ x \multimap np\ x$
$D_{\text{dit que}}$	: $(\Pi x, y : \mathit{extraction})\ (s\, x \multimap np\, y \multimap s\, (f\ (g\ x)\ y)$
$D_{\text{quelqu'un}}, D_{\text{tout le monde}}$	: $(\Pi x : \mathit{extraction})\ ((np\ \mathsf{non} \multimap s\, x) \multimap s\, x)$

où f et g sont des analyses de cas correspondant aux fonctions suivantes :

$$f : \begin{cases} \mathsf{pro} & \mathsf{non} \longrightarrow \mathsf{pro} \\ \mathsf{non} & \mathsf{pro} \longrightarrow \mathsf{pro} \\ \mathsf{non} & \mathsf{non} \longrightarrow \mathsf{non} \\ \mathsf{dis} & \mathsf{non} \longrightarrow \mathsf{dis} \\ \mathsf{non} & \mathsf{dis} \longrightarrow \mathsf{dis} \\ * & * \longrightarrow * \end{cases} \qquad g : \begin{cases} \mathsf{non} \longrightarrow \mathsf{non} \\ \mathsf{pro} \longrightarrow \mathsf{dis} \\ \mathsf{dis} \longrightarrow \mathsf{dis} \end{cases}$$

TAB. 4.5 – Σ_{Cont_3}

Le comportement d'un verbe transitif tel que D_{aime} est de vérifier si une variable extraite par un pronom relatif se trouve à l'intérieur d'un de ses paramètres. Le type résultant dépend donc de non si et seulement si le sujet et l'objet dépendent du terme non. C'est ce qu'implémente la fonction f dans Table 4.5. La dernière ligne de cette fonction indique que tous les cas restants sont laissés indéfinis : ce sont ceux où deux extractions ont lieu en même temps, et le type des pronoms relatifs assure que le terme ne sera quoi qu'il arrive pas acceptable. Le résultat de la fonction f dans ces cas n'est donc pas pertinent.

Les verbes requérant une proposition subordonnée, comme $D_{\text{dit que}}$ doivent également propager l'information de la présence d'une variable libre dans la proposition principale ou la proposition subordonnée (et dans ce second cas, l'extraction correspondante devient donc distante). Une extraction éventuellement présente dans la proposition subordonnée doit être étiquetée comme distante. C'est ce qu'implémente la fonction g dans Table 4.5.

Enfin, les pronoms relatifs doivent vérifier le type de leur argument. En particulier, ceux qui extraient un groupe nominal sujet ne peuvent pas accepter en argument une proposition de type $(np\ \mathsf{pro} \multimap s\ \mathsf{dis})$, au contraire des autres pronoms, car ce type implique que l'extraction initialement étiquetée par pro est passée à dis en dépassant un *dit que*. Dans la signature abstraite, D_{qui} n'a donc qu'une seule entrée, autorisant une extraction proche, alors que D_{que} possède deux entrées, l'une permettant les extractions proches et l'autre les extractions distantes. De cette façon l'extraction de sujets distants ne peut plus être générée par la grammaire, là où l'extraction d'objets distants est toujours possible, comme le montre le terme abstrait (13) de type $s\ \mathsf{non}$ associé à (12).

(13) $D_{\text{marche}}\ \mathsf{non}\ (D_{\text{que}}\ \mathsf{non}\ (\lambda^0 x : np\ \mathsf{pro}.\ D_{\text{dit que}}\ \mathsf{pro}\ \mathsf{non}\ \overbrace{(D_{\text{aime}}\ \mathsf{pro}\ \mathsf{non}\ \overbrace{x}^{np\ \mathsf{pro}}\ D_{\text{Marie}})}^{s\ \mathsf{pro}}\ D_{\text{Jean}})\ D_{\text{Paul}})$

$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{s\ \mathsf{dis}}$

$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{np\ \mathsf{pro} \multimap s\ \mathsf{dis}}$

De son côté, $\lambda^0 x.C_{\text{dit que}}\ (C_{\text{aime}}\ C_{\text{Marie}}\ x)\ C_{\text{Jean}}$ ne peut avoir pour antécédent qu'un terme de type $np\ \mathsf{non} \multimap s\ \mathsf{non}$, $np\ \mathsf{pro} \multimap s\ \mathsf{dis}$ ou $np\ \mathsf{dis} \multimap s\ \mathsf{dis}$, qui ne peut être argument de D_{qui}. (11) n'a donc pas d'antécédent par $\mathscr{L}_{\mathrm{Cont}_3}$.

Enfin, si un pronom relatif opère une extraction dans la proposition relative d'un autre pronom relatif, le terme résultant ne sera pas typable. En effet, un pronom relatif renvoie un terme possédant la même étiquette que le

groupe nominal qu'il complète. Comme les deux traces se trouvent dans la proposition relative la plus profonde, le pronom relatif le plus profond produira un terme de type *np* non, et le pronom relatif supérieur recevra donc une proposition relative de type $np_{\text{pro}} \multimap s_{\text{non}}$ comme argument, ce qui n'est pas compatible avec son type. Ainsi le terme

$D_{\text{qui non}}\,(\lambda^0 y : np_{\text{pro}}.D_{\text{aime pro pro}}\ x\ y)D_{\text{Paul}}$,

correspondant à la proposition "Paul qui$_2$ t_1 aime t_2" , ne peut être typé que par *np* non. Le terme

$\lambda^0 x : np_{\text{pro}}.D_{\text{marche non}}\,(D_{\text{qui non}}(\lambda^0 y : np_{\text{pro}}.D_{\text{aime pro pro}}\ x\ y)D_{\text{Paul}})$

ne peut donc être que de type $np_{\text{pro}} \multimap s_{\text{non}}$, et ne peut donc pas être donné en argument à un second pronom relatif.

Ce mécanisme de contrainte s'adapte aisément aux différentes manifestations d'îlots d'extraction pouvant être rencontrées. Par exemple, en langue anglaise la contrainte semble être la même pour les propositions subordonnées introduites par *says that*. En revanche, les propositions introduites par le seul *says* ne semblent pas bloquer les extractions sujet, comme le montre l'exemple suivant (Pollard, 2010)

The man who John said loves Mary sleeps

Cela suggère d'attribuer à D_{says} le type $(\Pi x, y : extraction)\ (s\,x \multimap np\,y \multimap s\,(f\ x\ y)$, qui n'utilise donc pas la fonction g pour changer un pro en dis comme le fait le type de $D_{\text{dit que}}$.

Par ailleurs, la même technique peut être utilisée pour modéliser le fait qu'un pronom interrogatif nominatif peut former une question avec une proposition sans sujet au niveau principal, comme dans in (14) mais pas avec une proposition sans sujet distant, comme dans (15).

(14) Qui marche ?

(15) *Qui$_1$ Marie dit que t_1 marche ?

4.2.4 Extractions interrogatives multiples

Nous traitons enfin le phénomène des contraintes d'imbrication, présentes en langue anglaise lorsque plusieurs pronoms interrogatifs opèrent une extraction dans la même question. Comme le montrent (16) et (17), elles spécifient que les traces doivent apparaître dans l'ordre inverse des pronoms interrogatifs auxquels elles correspondent (par soucis de clarté on ne tient pas compte ici du fait que *know* soit un verbe de contrôle).

(16) Which$_1$ problems does John know whom$_2$ to talk to t$_2$ about t$_1$?

a. $C_{\text{which ?}}\,C_{\text{problems}}\,(\lambda^0 x.\,C_{\text{know}}\,(C_{\text{whom ?}}\,(\lambda^0 y.\,C_{\text{to talk to about}}\,y\,x))\,C_{\text{John}})$

(17) *Whom$_1$ does John know which$_2$ problems to talk to t$_1$ about t$_2$?

a. $*C_{\text{whom ?}}\,(\lambda^0 y.\,C_{\text{know}}\,(C_{\text{which ?}}\,C_{\text{problems}}\,(\lambda^0 x.\,C_{\text{to talk to about}}\,y\,x))\,C_{\text{John}})$

Les extractions interrogatives suivent ainsi un schéma de non-croisement. Bien que ce schéma soit relié à l'ordre linéaire de la phrase réalisée, nous en proposons à nouveau une implantation au niveau abstrait. Le principe est là encore d'étiqueter les types par une information supplémentaire. Ici, cette information indique si le terme typé contient une extraction ou non. S'il en contient une, il est également donné un couple d'entiers, dénotant les bornes de l'intervalle des numéros des traces qui sont présentes dans le terme. Nous reprenons donc l'encodage des entiers présenté dans la section 4.2.2. Table 4.6 décrit la signature abstraite correspondant à cette idée.

Étant donné son type, un pronom interrogatif prend donc en argument un entier i qu'on peut se représenter comme le numéro de son extraction, au sens des numéros apparaissant dans (16) et (17). Puisqu'il prend également en argument un terme de type $(np_{\text{oui}}[g = i+1, d = i+1] \multimap q_{\text{oui}}[g = i+1, d = 1])$, et renvoie une question de type $q_{\text{oui}}[g = i, d = 1]$, le pronom fait décroître de un la borne supérieure de l'intervalle associé à la phrase. Plusieurs pronoms interrogatifs en cascade doivent donc être associés à des numéros d'extraction consécutifs, et croissants pour l'ordre de surface de la phrase.

α	: **type**
$couple = [g : int, d : int]$	: **type**
$ext = \{\mathsf{non}\|\bot\|\mathsf{oui}\ of\ couple\}$	: **type**
np, n, s, q	: (ext) **type**
D_{John}	: $np\ \mathsf{non}$
$D_{\text{to talk to about}}$	: $(\Pi i, j : ext)\ (np\, i \multimap np\, j \multimap q\, (h\, i\, j))$
D_{problems}	: $n\ \mathsf{non}$
D_{know}	: $(\Pi i, j : ext)\ (q\, i \multimap np\, j \multimap q\, (h\, i\, j))$
$D_{\text{whom ?}}$	: $(\Pi i : int)\ ((np\ \mathsf{oui}[g = i+1, d = i+1] \multimap q\ \mathsf{oui}[g = i+1, d = 1])$ $\multimap q\ \mathsf{oui}[g = i, d = 1])$
$D_{\text{which ?}}$	: $(\Pi i : int)\ (n\ \mathsf{non} \multimap (np\ \mathsf{oui}[g = i+1, d = i+1] \multimap q\ \mathsf{oui}[g = i+1, d = 1])$ $\multimap q\ \mathsf{oui}[g = i, d = 1])$

où h est le λ-terme qui implémente la fonction suivante :

$$h : \begin{cases} \mathsf{non} & x & \longrightarrow x \\ x & \mathsf{non} & \longrightarrow x \\ \mathsf{oui}[g = n, d = m] & \mathsf{oui}[g = n', d = m'] & \longrightarrow \mathsf{oui}[g = n, d = m'] \text{ si } m = n' + 1 \\ * & * & \longrightarrow \bot \text{ dans tous les autres cas} \end{cases}$$

TAB. 4.6 – Σ_{Cont_4}

La trace associée au pronom numéro i reçoit le type $np\ \mathsf{oui}[g = i+1, d = i+1]$. Il reste encore à vérifier que ces traces apparaissent dans la phrase avec leur numéros consécutifs et décroissants pour l'ordre de surface. Ceci est assuré par le type des constituants combinant plusieurs termes, qui donne en argument à la fonction h les traits des termes combinés dans l'ordre de surface de cette combinaison.

Ainsi le terme $t = D_{\text{to talk to about}}\ i\ j\ u\ v$ (à lire comme *to talk to* u *about* v), avec u de type $np\, i$ et v de type $np\, j$, est de type $q\ h\ i\ j$. Si ni u ni v ne contiennent d'extraction, t non plus, et on a bien $h\ i\ j = \mathsf{non}$. Si seul u contient des traces, alors $h = i = \mathsf{oui}[n, m]$, où n et m sont le maximum et le minimum des numéros des traces qui apparaissent dans u et donc dans t. Si seul v contient des traces, le cas est similaire. Enfin, si des traces apparaissent dans u et v, alors $i = \mathsf{oui}[n, m]$ et $j = \mathsf{oui}[n', m']$, le typage de u affirme que ses traces sont en position consécutive décroissante de la numéro n à la numéro m, le typage de v que ses traces sont en position consécutives décroissantes du numéro n' au numéro m'. L'ensemble des traces dans le terme t sont donc en position consécutives décroissantes si $m = n' + 1$, et le trait représentant l'intervalle complet est alors $h\ i\ j = \mathsf{oui}[n, m']$. C'est bien ce qu'exprime la fonction h. Si cette condition n'est pas vérifiée, le terme est typé par $\bot$, qui est un état poubelle ganrantissant que la phrase ne sera pas reconnue par la grammaire, car aucun terme du lexique ne peut le faire disparaître.

Le type des questions reconnue par cette signature abstraite correspond donc au type renvoyé par le premier pronom interrogatif dans l'ordre de surface, qui est donc numéroté par 0. Ce type distingué est donc $q\ \mathsf{oui}[0, 1]$. Montrons qu'un antécédent du terme (16) peut se voir attribuer ce type :

$t = D_{\text{which ?}}\ 0\ D_{\text{problems}}$ est un terme de type
$(n\ \mathsf{non} \multimap (np\ \mathsf{oui}[g = 1, d = 1] \multimap q\ \mathsf{oui}[g = 1, d = 1]) \multimap q\ \mathsf{oui}[g = 0, d = 1])$.

$$u = (\lambda^0 x : np\ \mathsf{oui}[1,1].\ \underbrace{D_{\text{know}}\ \mathsf{oui}[1,1]\ \mathsf{non}\ (\overbrace{D_{\text{whom ?}}\ 1\ (\lambda^0 y : np\ \mathsf{oui}[2,2].\ \underbrace{D_{\text{to talk to about}}\ \mathsf{oui}[2,2]\ \mathsf{oui}[1,1]\ y\ x}_{q\ \mathsf{oui}[2,1]})}^{q\ \mathsf{oui}[1,1]}\ D_{\text{John}})}_{q\ \mathsf{oui}[1,1]}$$

est quant à lui un terme de type $n\ \mathsf{non} \multimap (np\ \mathsf{oui}[g = 1, d = 1] \multimap q\ \mathsf{oui}[g = 1, d = 1]$. On a donc bien $t\ u$ de type $q\ \mathsf{oui}[g = 0, d = 1]$, et par ailleurs l'image de $t\ u$ par le lexique correspond bien au terme (16).

À l'inverse, le terme

$C_{\text{whom ?}}\,(\lambda^0 y.\, C_{\text{know}}\,(C_{\text{which ?}}\,C_{\text{problems}}\,(\lambda^0 x.\, C_{\text{to talk to about}}\,y\,x))\,C_{\text{John}})$

ne peut pas avoir d'antécédent de type $q\,\mathsf{oui}[g = 0, d = 1]$. Il faudrait en effet que $D_{\text{whom ?}}$ reçoive le numéro 0 et $D_{\text{which ?}}$ le numéro 1. La variable y serait alors de type étiqueté par $\mathsf{oui}[1, 1]$, et x par $\mathsf{oui}[2, 2]$. La fonction h appliquée à ces deux intervalles dans cet ordre produirait alors le terme $\bot$, et D_{which}recevrait en argument un terme avec le type $np\,\mathsf{oui}[2, 2] \multimap q\,\bot]$ duquel ce pronom est incompatible. Le terme (17) sera donc rejeté par une grammaire utilisant la signature abstraite présentée ici.

Nous avons présenté dans ce chapitre comment traiter un échantillon de phénomènes liés aux mouvements grammaticaux à l'aide d'une ACG de contrôle s'appliquant sur les structures de dérivation de l'interface syntaxe-sémantique. Ces ACG de contrôle utilisent l'extension du système de type que nous avons introduite dans les chapitres précédents.

Comme indiqué dans la section 1.2.5, cette extension entraîne un problème de l'appartenance indécidable dans le cas général, et il est donc crucial de déterminer si son emploi permet ici de rester dans un fragment de la grammaire décidable. Lorsque les termes dont peuvent dépendre les types sont en nombre fini, il est possible d'écrire une signature équivalente sans extension en dupliquant les entrées lexicales pour chaque combinaison de type dépendant possible. C'est le cas des signatures des sections 4.2.1 et 4.2.3. En revanche, lorsque les types des traits contiennent une infinité de termes, comme c'est le cas dans les sections 4.2.2 et 4.2.4, il n'est pas possible de réduire les signatures à des signatures sans extensions. La question se pose alors de savoir s'il existe pour ces ACG étendues utilisant des traits habités par une infinité de valeurs un fragment décidable. Par ailleurs il est possible pour une implémentation pratique d'utiliser une borne sur les entiers utilisés pour numéroter les extractions. Une telle hypothèse est d'autant plus raisonnable qu'une phrase comprenant un nombre trop élevé d'extractions est impossible à comprendre pour un locuteur.

Enfin, il reste à savoir si le niveau des structures de dérivation est le plus indiqué pour traiter les phénomènes d'extraction, ou s'il serait plus judicieux d'exercer un contrôle sur les formes de surface, syntaxiques ou sémantiques. De manière intéressante, la réponse semble dépendre de la facette des phénomènes d'extraction considérée. Ainsi le traitement des cas se fait naturellement au niveau abstrait dans les ACG, alors que le traitement des contraintes d'imbrication, qui repose sur l'ordre linéaire de surface des traces, demande un contrôle complexe pour être appliqué à ce niveau abstrait. Pour apporter quelques éléments de réponse, nous décrivons dans le chapitre suivant plusieurs mécanismes de traitement des extractions proposés pour d'autres formalismes grammaticaux.

Chapitre 5

Comparaison aux autres approches

Dans ce chapitre, nous nous livrons à un tour d'horizon de formalismes grammaticaux connectés aux ACG, et de la façon dont ils s'y comparent, notamment du point de vue des problématiques de mouvements linguistiques. Nous considérons dans un premier temps sur les formalismes calculant de façon parallèle les niveaux tectogrammaticaux, phénogrammaticaux et sémantiques, et contrôlant les mouvements au niveau phénogrammatical. Nous passons ensuite aux formalismes modélisant les mouvements implicites en s'appuyant sur une sémantique avec continuations, puis les formalismes dénotant les mouvements par un type spécial, les formalismes utilisant également le produit dépendant, et enfin les formalismes dont l'encodage dans les ACG utilise un mécanisme de contrôle similaire à celui du chapitre précédent.

5.1 Contrôle au niveau phénogrammatical

Nous discutons ici de la comparaison entre notre approche qui agit au niveau abstrait et les approches dans lesquelles le contrôle vient d'un calcul spécifique au niveau objet. Muskens (2007) propose une approche de ce type pour les Lambda Grammars (LG), en introduisant au niveau phénogrammatical une analyse inspirée des Grammaires Catégorielles Multimodales (MMCG) de Moortgat (1996). Kubota et Pollard (2009) proposent également une approche qui repose sur une analyse en MMCG. Elles correspondent en pratique à un formalisme parallèle où chacun des niveaux tectogrammatical et phénogrammatical sont des MMCG. De façon intéressante, cette approche permet des modifications phonologiques au niveau phénogrammatical sans pour autant modifier le niveau tectogrammatical.

Afin de comparer les trois approches, nous introduisons les notations suivantes :

Définition 23 (Signes et langages)**.** *Un* signe $s = \langle a, o, m\rangle$ *est un triplet tel que :*

- a *est un terme du niveau tectogrammatical*
- o *est un terme du niveau phénogrammatical décrivant la forme de* surface *associée à* a
- m *est un terme du niveau phénogrammatical décrivant la forme* logique *associée à* a

Dans le cas des LG et des ACG, a *est un* λ*-terme linéaire tandis que c'est un terme de preuve de MMCG pour Kubota et Pollard (2009).*

Dans tous les formalismes, un signe $s = \langle a, o, m\rangle$ *appartient au langage si et seulement si* a *est du type principal s. D'après Muskens (2007), on parlera d'un signe* généré*.*

Dans les ACG avec notre approche, o est un λ-terme, utilisant l'opération de concaténation des chaînes de caractère.

De son côté, Muskens (2007) fait de o une formule de logique multimodale construite à partir de constantes et de connecteurs logiques (unaires et binaires), comprenant non seulement un connecteur binaire spécial $\circ$ représentant la concaténation, mais également d'autres connecteurs, en particulier une famille d'opérateurs $\Diamond_i$ et $\Box_i$. Le niveau phénogrammatical peut alors être muni d'une relation de conséquence $\sqsubseteq$ et, de la façon standard pour les MMCG, des axiomes, ou *postulats*, décrivant comment ces modalités peuvent interagir. Il est ensuite possible d'hériter de tous les modèles de ce formalisme, comme celui de Morrill (1992) pour contrôler l'extraction. Pour tout signe $s = \langle a, o, m\rangle$, il est alors possible de définir une notion de dérivabilité :

Définition 24 (Signes dérivables et chaînes)**.** *Soit $s = \langle a, o, m\rangle$ un signe généré et o' une formule logique telle que $o \sqsubseteq o'$. Alors $s' = \langle a, o', m\rangle$ est appelé un signe* dérivable.

Soit $s = \langle a, o, m\rangle$ un signe tel que o n'est composé que de constantes et de $\circ$. Alors on dira que o est lisible[6] *et on dira que s est un signe* chaîne.

Avec cette perspective, le centre d'intérêt n'est plus le signe généré mais bien plutôt les signes chaînes. En particulier, si $s = \langle a, o, m\rangle$ est un signe généré, la question importante est de savoir s'il existe un o' tel que $o \sqsubseteq o'$ et o' *lisible*. Si un tel o' existe, alors s peut être exprimé, et dans le cas contraire il ne peut pas.

(Muskens, 2007, example (35)) est très proche de notre exemple

(5) Quelqu'un dit que tout le monde aime Marie

a. $C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{dit que}}\,(C_{\text{tout le monde}}(\lambda^0 y.C_{\text{aime}}\,C_{\text{Marie}}\,y))\,x)$
$\exists x.\textbf{dit que}\,x\,(\forall y.\textbf{aime}\,y\,\textbf{m})$

b. $*C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{tout le monde}}(\lambda^0 y.C_{\text{dit que}}\,(C_{\text{aime}}\,C_{\text{Marie}}\,y)\,x))$
$*\exists x.\forall y.\textbf{dit que}\,x\,(\textbf{aime}\,y\,\textbf{m})$

c. $*C_{\text{tout le monde}}(\lambda^0 y.C_{\text{quelqu'un}}(\lambda^0 x.C_{\text{dit que}}\,(C_{\text{aime}}\,C_{\text{Marie}}\,y)\,x))$
$*\forall y.\exists x.\textbf{dit que}\,x\,(\textbf{aime}\,y\,\textbf{m})$.

Son analyse est la suivante : (5-a), (5-b) et (5-c) sont deux termes abstraits possibles tels que $s_a = \langle$(5-a)$, o_a, m_a\rangle$, $s_b = \langle$(5-b)$, o_b, m_b\rangle$ et $s_c = \langle$(5-c)$, o_c, m_c\rangle$ sont tous des signes générés. Cependant, il n'existe aucun signe lisible o tel que $o_b \sqsubseteq o$ ou $o_c \sqsubseteq o$ car o_b et o_c utilisent différentes sortes de modalités qui n'intéragissent pas via les postulats : synthétiquement, un QNP introduit au niveau phénogrammatical deux modalités appairées, l'une sur le site syntaxique et l'autre sur le site de prise de portée, et un îlot d'extraction introduit une autre modalité qui se révèle bloquante si elle se trouve entre les deux modalités précédentes. s_b et s_c peuvent donc être générés mais n'ont pas de forme lisible (ou prononçable), et seul s_a aboutit à un signe chaîne et peut donc être exprimé. L'approche de Kubota et Pollard (2009) est similaire à la différence que le niveau phénogrammatical est une algèbre munie d'un préordre dont les éléments maximaux sont les seuls éléments prononçables.

5.2 Sémantique avec continuations

5.2.1 *Continuation Passing Style*

L'usage de continuations au niveau du calcul sémantique peut permettre de capturer les phénomènes de prise de portée des quantificateurs dans les langues naturelles. Ainsi, Barker (2002) propose de différencier les groupes nominaux quantifieurs (QNP) des NP non quantifieurs selon leur faculté à modifier l'environnement dans lequel leur contribution sémantique est calculée, c'est-à-dire à avoir un effet de bord. Dans ce cas, l'effet en question est d'introduire une quantification en amont de la position syntaxique du QNP.

Le passage de continuations permet de modéliser ce type d'effet. Une continuation est en effet un terme dénotant un environnement. Ainsi, pour un terme t de type α, et si le type de retour attendu est β, une continuation pour t est une fonction de α dans β. Si le type du terme t est modifié pour qu'il puisse prendre en argument une continuation, ce terme pourra appliquer des effets sur son environnement, avant de produire un résultat de type β. Dans ce style de passage de continuations (CPS, pour Continuation Passing Style), le type du terme t doit donc devenir $(\alpha \to \beta) \to \beta$.

Habituellement, les constituants syntaxiques de type *np* sont associés à des termes sémantiques de type e, dénotant des entités, et les constituants syntaxiques de type *s* sont associés à des termes sémantiques de type t, dénotant des propositions. Pour obtenir une sémantique en CPS, il faut donc que *np* soit associé à $(e \to t) \to t$, et *s* à $(t \to t) \to t$. La plupart des termes n'ont pas d'effet de bord, et ne modifient donc pas la continuation qu'ils prennent en argument, ils se contentent de l'appliquer à leur contribution sémantique. Par exemple, supposons que le nom propre "Jean" soit dénoté sémantiquement par la constante **j** de type e. En CPS, la sémantique de "Jean" est de type $(e \to t) \to t$, c'est le terme $\lambda c.c\,\textbf{j}$. Par contre, un constituant ayant un effet de bord peut à présent

[6] Kubota et Pollard (2009) parle plutôt de *prononçable* car cette notion serait plus proche de la phonologie que des chaînes de caractères.

accéder à sa continuation pour la modifier. Ainsi, un QNP tel que "tout le monde" sera associé au terme sémantique $\lambda c.\forall x.c\ x$.

Dans la proposition de Barker, les phénomènes d'ambiguité de portée entre quantificateurs, ainsi que les phénomènes de blocage de portée, par exemple l'impossibilité de dépasser une proposition comme discuté dans la section 4.2.2, sont alors pris en compte par la façon dont est combinée la sémantique des différents constituants. Lorsque des effets de bord sont présent, l'ordre d'évaluation a en effet une incidence sur le résultat. Barker prend pour exemple une grammaire hors-contexte, munie notamment d'une règle $s \rightarrow$ *np* vp. Lorsque la grammaire n'est pas en CPS, le terme sémantique associé à un tel s est obtenu en appliquant le terme sémantique du vp à celui du *np*, ce que l'on note $[vp]$ [*np*]. Pour obtenir une sémantique de continuations, ces règles de combinaison sémantique doivent également être modifiées. Une première possibilité est $\lambda c.[vp](\lambda c'.[\textit{np}](\lambda x.c\ (c'\ x)))$. Cependant, Barker avance qu'il est tout aussi légitime d'adopter l'ordre d'évaluation symétrique, ce qui modifie l'ordre dans lequel sont appliqués les effets de bord. Cet ordre d'évaluation permuté correspond à la règle $\lambda c.[np](\lambda x.[vp](\lambda c'.c\ (c'\ x)))$. Les deux façons de combiner génèrent le même resultat lorsqu'il n'y pas de concurrence entre effets de bord dans le *np* et le vp. En revanche, lorsque le np et le vp contiennent chacun des QNP, les deux règles de combinaison produisent des résultats différents. Ainsi, considérons la phrase

(1) Tout le monde aime quelqu'un.

$[aime\ quelqu'un]$ ne contient qu'un QNP, l'ordre d'évaluation n'a donc pas d'importance. On a

$$[aime\ quelqu'un] = \lambda p'.(\lambda r.r\ aime)(\lambda p''.(\lambda p'''.\exists x.p'''\ x)(\lambda x'.p'\ (p''\ x')))$$
$$\rightarrow^*_\beta \lambda p'.\exists x.p'\ (aime\ x)\ .$$

$[tout\ le\ monde\ aime\ quelqu'un]$ contient deux QNP, et peut donc être calculé dans deux ordres différents. La première règle conduit au calcul :

$$\lambda c.[aime\ quelqu'un](\lambda c'.(\lambda p.\forall y.p\ y)(\lambda z.c\ (c'\ z)))$$
$$\rightarrow^*_\beta\ \lambda c.\exists x.\forall y.c\ (aime\ x\ y),$$

qui peut être appliquée à la continuation triviale, le terme identité $\lambda q.q$, pour achever l'évaluation sémantique, ce qui produit le terme $\exists x.\forall y.aime\ x\ y$.

L'usage de la seconde règle de combinaison conduit au calcul :

$$\lambda c.\lambda p.\forall y.p\ y)(\lambda z.[aime\ quelqu'un](\lambda c'.c\ (c'\ z)))$$
$$\rightarrow^*_\beta\ \lambda c.\forall y.\exists x.c\ (aime\ x\ y),$$

qu'on peut appliquer à l'identité pour obtenir $\forall y.\exists x.aime\ x\ y$. L'ambiguité de prise de portée entre quantificateurs est donc bien capturée par cette variation des ordres d'évaluation.

Par ailleurs, Barker montre que pour empêcher un quantificateur de prendre portée au-delà d'une proposition, il suffit que les règles de production de s soient associées à des règles de combinaison sémantique ne donnant pas accès à la continuation associée à la proposition. Par exemple, plutôt que d'associer les règles sémantiques

$\lambda c.[vp](\lambda c'.[\textit{np}](\lambda x.c\ (c'\ x)))$ et
$\lambda c.[np](\lambda x.[vp](\lambda c'.c\ (c'\ x)))$

à la règle de production $s \rightarrow$ *np* vp, comme on l'a vu précédemment, on peut lui associer les règles

$\lambda c.c\ ([vp](\lambda c'.[\textit{np}](\lambda x.c'\ x))$ et
$\lambda c.c([np](\lambda x.[vp](\lambda c'.c'\ x))$.

La variable c, qui correspond au futur du calcul, n'est plus donné en argument aux termes sémantiques du np et du vp, qui ne peuvent donc plus y accéder pour y produire des effets de bord. De la sorte, une phrase comme

(2) Quelqu'un dit que tout le monde aime Marie

ne peut pas être associée aux termes sémantiques $\exists x.\forall y.$**dit que** x (**aime** y **m**) ou $\forall y.\exists x.$**dit que** x (**aime** y **m**). Une telle modification des règles de combinaison sémantique accomplit donc bien le même objectif que le contrôle que nous avons présenté dans la section 4.2.2.

Le principe d'introduire des continuations au niveau sémantique a été exploré dans le cadre des ACG. Il est notamment au coeur du traitement du discours proposé par Lebedeva (2012), dans lequel un contexte pragmatique est transmis de phrases en phrases. Par ailleurs, le calcul sémantique que nous avons introduit dans la section 4.1.3 utilise lui-même une forme de passage de continuations en associant à la catégorie syntaxique *np* le type sémantique d'ordre supérieur $(e \multimap t) \multimap t$. La grammaire décrite n'est cependant pas continuisée au sens de Barker, puisque *s* est associé au type simple t, plutôt qu'à $(t \multimap t) \multimap t$. La raison de cette montée du type sémantique associé aux *np* est de permettre à un pronom relatif d'accéder à la valeur sémantique de la proposition du *np* qu'il complète afin de la conjoindre à la valeur sémantique de la proposition relative. En effet, le lexique sémantique associe à "qui" le terme sémantique :

$$\lambda^0 pnx.\exists y.(p\ (\lambda^0 r.r\ y)) \wedge (x\ y) \wedge (n\ (\lambda^0 z.z = y))$$

L'argument x qui y apparaît est une continuation, qui correspond au futur du calcul sémantique, donc à la proposition principale. Ainsi, si l'on calcule le terme sémantique associé au *np* "Marie qui$_1$ t_1 marche", on obtient :

$$(\lambda^0 pnx.\exists y.(p\ (\lambda^0 r.r\ y)) \wedge (x\ y) \wedge (n\ (\lambda^0 z.z = y)))\ (\lambda^0 t_1.\lambda^0 s.s(\lambda^0 y'.\textbf{marche}\ y')\ t_1)\ (\lambda^0 p'.p'\ \textbf{m})$$
$$\rightarrow^*_\beta \lambda^0 x.\exists y.(\textbf{marche}\ y) \wedge (x\ y) \wedge (\textbf{m} = y).$$

Si on applique le terme sémantique du verbe "chante" à ce résultat, on constate que ce passage de continuation permet au pronom relatif d'opérer une conjonction de la sémantique des propositions relative et principale :

$$(\lambda^0 s.s(\lambda^0 y'.\textbf{chante}\ y'))\ \lambda^0 x.\exists y.(\textbf{marche}\ y) \wedge (x\ y) \wedge (\textbf{m} = y)$$
$$\rightarrow^*_\beta \exists y.(\textbf{marche}\ y) \wedge (\textbf{chante}\ y) \wedge (\textbf{m} = y).$$

Cet usage de continuations n'est par contre pas motivé par le traitement des QNP. En effet, on a vu dans le chapitre 4 que les QNP recevaient un type d'ordre supérieur $(np \multimap s) \multimap np$ dès le niveau abstrait. Une phrase ambigüe comme

(1) Tout le monde aime quelqu'un.

est donc associée à deux structures de dérivation, et les QNP apparaîssent dans ces structures à la position à laquelle ils prennent sémantiquement portée. À l'inverse, Barker souligne que sa méthode ne nécessite pas de montée du type syntaxique des QNP, et n'associe qu'un seul arbre de dérivation à (1). L'ambiguité est alors manifestée par le non-déterminisme du calcul sémantique à partir de cette structure de dérivation. Cela illustre une différence importante entre les deux propositions. Dans les ACG, la relation entre le niveau abstrait (les structures de dérivation) et le niveau sémantique est une fonction du premier vers le second : le lexique sémantique. Le passage des structures de dérivation aux structures de surface est donc déterministe par construction, et les ambiguités linguistiques doivent se manifester comme des structures de dérivation distinctes, et ayant pour image la même forme de surface syntaxique. La différence entre les deux approches s'inscrit donc dans le contraste entre les architectures syntacto-centrées, où la sémantique est fonction de la syntaxe, et les architectures parallèles, où structures syntaxiques et sémantiques sont associées par une relation. Il est néanmoins possible d'encoder une architecture parallèle dans les ACG, comme l'ont montré De Groote et al. (2011) avec les *Convergent Grammars* (CVG).

Barker propose un argument en faveur de la pertinence de l'approche par continuations en considérant les cas où certains ordres de prises de portée par les QNP ne sont pas admissibles. Ainsi, pour la phrase

(3) La plupart des sujets a mis un objet dans toutes les boîtes

il est commun de considérer que le QNP "la plupart des sujets" ne peut pas prendre portée entre "un objet" et "toutes les boîtes". Une interprétation sémantique présentant par exemple l'ordre suivant concernant la prise de portée :

(4) $*toutes\ les\ boîtes > la\ plupart\ des\ sujets > un\ objet$

doit donc être écartée. Le traitement des QNP par passage de continuation est cohérent avec cette observation : ces sémantiques non admissibles ne peuvent pas être obtenues par le calcul sémantique. Cette propriété peut se concevoir en termes d'ordre d'évaluation. L'ordre de prise de portée des QNP correspond à l'ordre dans lequel ils sont évalués. Comme on l'a vu, les règles sémantiques sont non-déterministes, et il est possible d'évaluer d'abord le sujet de la phrase, ou le groupe verbal. Ce non-déterminisme ne permet toutefois pas à une *partie* du groupe verbal d'être évalué, avant d'évaluer le sujet puis le reste du groupe verbal. Si le sujet est évalué en premier, "la plupart des sujets" a la portée maximale. Si le sujet est évalué en second, "la plupart des sujets" a la portée minimale. Le QNP ne peut donc pas voir sa portée prendre une position intermédiaire entre celle des deux autres QNP.

À l'inverse, rien n'empêche dans les ACG que nous avons présentées d'attribuer le type *s* à un terme abstrait (simplifié à l'extrême) de la forme :

$$C_{\text{toutes les boîtes}}(\lambda^0 x.C_{\text{la plupart des sujets}}(\lambda^0 y.C_{\text{un objet}}(\lambda^0 z.C_{\text{mettre}}\ z\ (C_{\text{dans}}\ x)\ y))).$$

Ce terme abstrait correspond bien à la phrase (3) et à l'ordre incorrect (4). Il ne viole pas non plus les contraintes qui sont contrôlées par les grammaires du chapitre 4. Là encore, le mécanisme d'abstraction permettant de construire des termes abstraits se révèle surgénérant. Il semble possible de contrôler ce type de phénomène d'une façon similaire au contrôle apparaissant dans la section 4.2.4 pour les mouvements explicites multiples. Une telle modélisation est une possibilité de travaux futurs. Elle s'annonce complexe en termes de calcul, et ne possédant pas l'élégante économie de moyens de la solution de Barker. Elle permet néanmoins de conserver la manifestation des ambiguités au niveau des structures de dérivation.

5.2.2 Continuations délimitées

Une variante de la méthode des continuations est proposée par Shan (2004) en utilisant les opérateurs de contrôle **shift** et **reset** dans le calcul sémantique. L'usage de ces opérateurs permet notamment de ne pas expliciter le passage des continuations dans tous les termes sémantiques. Seuls les constituants produisant ou limitant des effets de bord ont une contribution sémantique manipulant des continuations, sous la forme de ces opérateurs de contrôle.

L'opérateur **shift** est un symbole mutificateur, noté ξ, qui capture la continuation correspondant au futur du calcul et la déplace là où apparaît la variable liée, comme l'illustre la séquence de réductions suivante :

$$\begin{aligned}
&10 \times (\xi f.1 + (f\ 2)) \\
&\rightarrow 1 + ((\lambda y.10 \times y)\ 2) \\
&\rightarrow 1 + (10 \times 2) \rightarrow 1 + 20 \rightarrow 21
\end{aligned}$$

L'opérateur **reset**, noté à l'aide des parenthèses [ ] , restreint la continuation qu'un opérateur **shift** va capturer. On a ainsi la séquence de réductions :

$$\begin{aligned}
&3 \times [10 \times (\xi f.1 + (f\ 2))] \\
&\rightarrow 3 \times [1 + ((\lambda y.10 \times y)\ 2)] \\
&\rightarrow 3 \times [1 + (10 \times 2)] \rightarrow 3 \times [1 + 20] \rightarrow 3 \times [21] \rightarrow 3 \times 21 \rightarrow 63
\end{aligned}$$

dans laquelle $3\times$ n'est pas capturé par l'opérateur **shift** car il est au-delà de l'expression délimitée par [ ].

L'opérateur **shift** permet d'attribuer aux QNP des termes sémantiques rendant compte du phénomène de prise de portée, par exemple :

$$\begin{aligned}
&\textit{tout le monde} &&: \xi c.\forall x.c(x) \\
&\textit{quelqu'un} &&: \xi c.\exists x.c(x)
\end{aligned}$$

L'opérateur **reset** permet de délimiter ces prises de portées. Il peut donc notamment être utilisé pour modéliser des îlots d'extraction, comme celui des propositions. Ainsi, on peut attribuer au verbe "aime" le terme sémantique $\lambda xy.[\textbf{aime}\ x\ y]$.

Comme on l'a vu précédemment, un tel calcul doit rendre compte des ambiguités de portée entre quantificateurs. La solution de Barker de laisser le calcul sémantique non déterministe, selon l'ordre d'évaluation qui est employé, est tout aussi envisageable ici. Cependant, Shan avance que se restreindre à une évaluation dans l'ordre de gauche à droite donne aux calculs de bonnes propriétés. comme le fait de traiter correctement les phénomènes de polarité, dans lesquels l'ordre linéaire des QNP dans la phrase joue un rôle crucial pour l'acceptabilité de cette phrase, à l'exemple de :

(5) No student liked any course

(6) *Any student liked no course

Comme alternative à ce non déterminisme dans l'ordre d'évaluation, Shan propose d'étendre les opérateurs **shift** et **reset** dans une hiérarchie où chaque opérateur est associé à un entier naturel, dénotant la priorité de l'opérateur. L'ambiguité se manifeste alors par le choix des entiers attribués aux différents QNP : si deux QNP sont susceptibles de prendre portée l'un sur l'autre, c'est celui étiqueté par le plus grand entier qui prendre la portée la plus large.

Cette hiérarchie d'opérateurs peut être traduite en λ-calcul en appliquant une transformation en CPS. Il est donc possible d'intégrer cette technique dans les ACG. Ces continuations peuvent apparaître au niveau du calcul syntaxique, exprimant explicitement les prises de portées des QNP dans les structures de dérivation. Cependant la transformation en CPS implique une augmentation importante de l'ordre des types manipulés, qui va donc avoir un impact conséquent sur la complexité du problème de l'analyse. Une solution alternative pourrait être d'appliquer cette transformation au niveau sémantique, et d'anoter au niveau syntaxique chaque QNP par sa priorité à prendre portée. Une telle solution ne violerait pas le principe d'avoir la sémantique comme une fonction de la syntaxe, et permettrait de bénéficier des bonnes propriétés des continuations sur les exemples comme

(3) La plupart des sujets a mis un objet dans toutes les boîtes

5.3 Types dénotant un mouvement

Le type attribué dans la proposition de Shan à un QNP s'écrit e_t^t, et correspond par la transformation en CPS au type $(e \to t) \to t$. De manière générale, le calcul de Shan permet de construire le type α_β^γ, correspondant par transformation CPS à $(\alpha \to \beta) \to \gamma$, et qui peut être interprété comme le type des constituants se comportant localement comme des élements de types α, mais prenant contrôle sur un contexte de type β, et produisant un nouveau contexte de type γ.

5.3.1 Constructeur q dans les grammaires catégorielles

Ce concept renvoie au constructeur de type ternaire q introduit par Moortgat (1992) pour modéliser le comportement des QNP dans les grammaires catégorielles. Moortgat propose de capturer l'information relative à l'ordre des mots dans un champ de chaîne de caractère associé à chaque constituant, plutôt que par un contexte privé de règles structurelles, comme dans le calcul de Lambek présenté dans la section 1.1.2. Les constituants sont donc représentés par des triplets contenant leur type syntaxique, leur terme sémantique et leur chaîne de caractère de surface, ce sont des signes comme nous l'avons présenté dans la section 5.1. Les règles logiques du calcul peuvent alors décrire explicitement comment les chaînes de caractères sont combinées. Cette propriété permet de donner les règles associées à un connecteur d'extraction $\uparrow$ et un connecteur d'infixation $\downarrow$. $\alpha \uparrow \beta$ correspond au type d'un constituant pouvant entourer un constituant de type β pour constituer un α (une proposition relative est ainsi de type $s \uparrow np$). Un constituant de type $\alpha \downarrow \beta$ peut lui être entouré par un β pour former un α. Dans cette perspective, un QNP correspondrait à un constituant de type $s \downarrow (s \uparrow np)$: l'extraction et l'infixation successives permettent au QNP de prendre portée sur une proposition qui le contient. Cependant, cette solution exige que les équations sur les chaînes de caractères engendrées par les règles de $\downarrow$ et $\uparrow$ soient liées, afin que les positions où le QNP est extrait puis réinséré coïncident. C'est pour exprimer cette condition que Moortgat propose d'intégrer un constructeur ternaire q au calcul, tel que $q(\alpha, \beta, \gamma)$ corresponde à $\gamma \downarrow (\beta \uparrow \alpha)$ avec coïncidence des positions associées à $\uparrow$ et $\downarrow$. Les règles de ce constructeur q en calcul des séquents sont alors les suivantes :

$$\frac{\langle \gamma, t\ (\lambda x.u), \chi\rangle \cup \Delta \vdash T \qquad \langle \alpha, x, \tau\rangle \cup \Gamma \vdash \langle \beta, u, \sigma\rangle}{\langle q(\alpha,\beta,\gamma), t, \psi\rangle \cup \Gamma \cup \Delta \vdash T}\ \text{(QL)}$$

$$\frac{\langle \beta, u, \phi\rangle \cup \Delta \vdash \langle \gamma, t\ (\lambda x.u), \chi\rangle \qquad \Gamma \vdash \langle \alpha, x, \tau\rangle}{\Gamma \cup \Delta \vdash \langle q(\alpha,\beta,\gamma), t, \psi\rangle}\ \text{(QR)}$$

ces deux règles étant associées à l'équation de chaînes de caractères :

$$\begin{cases} \sigma &= \phi_1 + \tau + \phi_2 \\ \chi &= \phi_1 + \psi + \phi_2 \end{cases}$$

Bernardi (2009) a par ailleurs montré comment capturer ces méchanismes d'extraction et d'infixation dans un système de preuve sans recourir à ces équations sur les chaînes de caractères, en introduisant les connecteurs duaux de ".", "/" et "\".

L'ambiguïté de prise de portée entre QNP provient de la possibilité de prouver différents séquents dont la conclusion a la même forme de surface, selon l'ordre de traitement des QNP, comme l'illustrent les dérivations de la table 5.1.

Les QNP se voient donc attribuer le type $q(np, s, s)$. Comme exemple de la pertinence du caractère trinaire du constructeur q, Moortgat propose le cas d'un comparatif "plus ... que", comme apparaît dans la phrase

(7) Jean a acheté plus de livres que Marie a vendu de livres

Sommairement, si on note s_{que} le type correspondant au constituant "que Marie a vendu de livres", alors le type de "plus de livres" doit correspondre à l'extraction depuis une phrase privée d'un *np* (la phrase "Jean a acheté t"), et la réinsertion dans un constituant qui attend un s_{que} a sa droite pour former une phrase ("Jean a acheté plus de livres"). Ce type s'écrit donc $q(np, s, s/s_{que})$, ce qui est bien une instance de $q(\alpha, \beta, \gamma)$ avec $\beta \neq \gamma$.

Le comportement de ce type $q(\alpha, \beta, \gamma)$ est capturé dans les ACG par un type $(\alpha \multimap \beta) \multimap \gamma$ au niveau abstrait. Le type $q(np, s, s)$ des QNP correspond donc au type $(np \multimap s) \multimap s$ que nous avons utilisé dans le chapitre 4, et entraîne les mêmes problématiques de surgénération sémantique. Ces règles autorisent en effet les QNP à prendre des portées arbitraires, et dans l'exemple (3) (La plupart des sujets a mis un objet dans toutes les boîtes), rien n'empêche "la plupart" de prendre portée entre les deux autres QNP, à l'inverse du traitement par continuations discuté précédemment. Cette approche nécessite donc tout autant d'établir un contrôle sur les structures générées.

5.3.2 *Convergent Grammar*

On retrouve un concept similaire, reprenant l'écriture de Shan, dans les Convergent Grammars (CVG) de Pollard (2008). Ces grammaires présentent un certain nombre de similarités avec les ACG. Elles sont composées d'un niveau syntaxique et d'un niveau sémantique, mis en relation par un niveau d'interface. Dans chacun de ces niveaux, les termes acceptables sont ientifiées par un système de typage linéaire. Cependant, à la différence des ACG, il n'y a pas de λ-abstraction, bien que des variables (appelées traces au niveau syntaxique) puissent apparaître dans les termes. Pollard propose deux méchanismes pour lier ces variables et capturer les phénomènes de mouvements. Le premier transcrit la méthode de Gazdar (1981) pour capturer une trace. Un terme a de type α^{γ}_{β} peut être appliqué à un terme de type β contenant une variable t de type α, pour retourner un terme de type γ dans lequel t est liée par a. C'est ce qu'exprime, dans l'exemple du niveau syntaxique, la règle :

$$\frac{\Gamma \vdash a : \alpha^{\gamma}_{\beta} \qquad t : \alpha, \Gamma' \vdash b : \beta}{\Gamma, \Gamma' \vdash a_t\ b : \gamma}\ \text{(G)}$$

L'indice t apparaissant sur le a indique que a est devenu un symbole mutificateur pour la variable t. Cette règle correspond en ACG à l'application d'un terme de type $(\alpha \multimap \beta) \multimap \gamma$ sur un terme dans lequel la variable t est λ-abstraite. $a_t\ b$ correspond alors à $a\ (\lambda^0 t.b)$. Cette règle G correspond donc à la méthode sur laquelle nous nous sommes basés dans le chapitre 4.

La seconde méthode proposée par Pollard correspond à la technique de stockage des QNP introduite par Cooper (1975). Cette technique consiste à traiter un QNP comme une variable de type *np*, tout en stockant l'information relative à ce QNP dans un contexte dédié. Les QNP peuvent ensuite être déstockés plus loin dans la dérivation,

$$\dfrac{\dfrac{}{T \vdash T}\,(\text{Ax}) \quad \dfrac{\dfrac{}{U \vdash U}\,(\text{Ax}) \quad \dfrac{\vdots}{\langle np, x, /\text{tout le monde}/\rangle, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \langle np, y, /\text{quelqu'un}/\rangle \vdash \langle s, \textbf{aime}\ y\ x, /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle}}{\underline{\langle q(np, s, s), \forall, /\text{tout le monde}/\rangle}, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \langle np, y, /\text{quelqu'un}/\rangle \vdash U}\,(\text{QL})}{\langle q(np, s, s), \forall, /\text{tout le monde}/\rangle, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \underline{\langle q(np, s, s), \exists, /\text{quelqu'un}/\rangle} \vdash T}\,(\text{QL})$$

avec $T = \langle s, \exists\ (\lambda y.\forall\ (\lambda x.\textbf{aime}\ y\ x)), /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle$
et $U = \langle s, \forall\ (\lambda x.\textbf{aime}\ y\ x), /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle$.

$$\dfrac{\dfrac{}{T' \vdash T'}\,(\text{Ax}) \quad \dfrac{\dfrac{}{U' \vdash U'}\,(\text{Ax}) \quad \dfrac{\vdots}{\langle np, x, /\text{tout le monde}/\rangle, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \langle np, y, /\text{quelqu'un}/\rangle \vdash \langle s, \textbf{aime}\ y\ x, /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle}}{\langle np, x, /\text{tout le monde}/\rangle, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \underline{\langle q(np, s, s), \exists, /\text{quelqu'un}/\rangle} \vdash U'}\,(\text{QL})}{\underline{\langle q(np, s, s), \forall, /\text{tout le monde}/\rangle}, \langle np\backslash s/np, \textbf{aime}, /\text{aime}/\rangle, \langle q(np, s, s), \exists, /\text{quelqu'un}/\rangle \vdash T'}\,(\text{QL})$$

avec $T' = \langle s, \forall\ (\lambda x.\exists\ (\lambda y.\textbf{aime}\ y\ x)), /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle$
et $U' = \langle s, \exists\ (\lambda y.\textbf{aime}\ y\ x), /\text{tout le monde}/ + /\text{aime}/ + /\text{quelqu'un}/\rangle$.

TAB. 5.1 – Dérivations engeandrant les différentes prises de portée entre QNP

lorsque le type *s* est formé, ce qui leur permet de prendre portée à une distance arbitraire. Les ambiguïtés de prise de portée viennent alors de l'ordre dans lequel les QNP sont déstockés. Pollard capture cette technique en introduit dans le niveau sémantique un contexte droit, dans lequel les QNP sont stockés. Les règles associées sont alors :

$$\frac{\vdash a : \alpha^{\gamma}_{\beta} \dashv \Delta}{\vdash x : \alpha \dashv a_x : \alpha^{\gamma}_{\beta}, \Delta}\ (\mathrm{C}) \qquad \frac{\Gamma \vdash b : \beta \dashv a_x : \alpha^{\gamma}_{\beta}, \Delta}{\Gamma \vdash a_x\ b : \gamma \dashv \Delta}\ (\mathrm{R})$$

C (pour Cooper) correspond au stockage, et R (pour Retrieval) au déstockage.

De Groote et al. (2011) ont montré que les règles (C) et (R) étaient redondantes avec la règle (G) au niveau sémantique : toute dérivation utlisant ces règles (C) et (R) peut être transformée en une dérivation équivalente n'utilisant pas ces règles, mais plutôt la règle (G). Ce résultat permet de mettre en correspondance cette technique de stockage de Cooper avec les approches dans lesquelles les QNP disposent d'un type d'ordre supérieur, comme la nôtre.

5.4 Utilisations du produit dépendant

Le produit dépendant, sur lequel repose le mécanisme de contrôle que nous avons présenté au chapitre 4, n'est pas une construction de type fréquemment utilisée dans le domaine du traitement de la langue. On peut tout de même en noter deux exemples d'utilisation.

La construction apparaît dans le *Grammatical Framework* de Ranta (2004), où elle permet notamment de capturer des phénomènes sémantiques, tels que les *donkey sentences*. Elle permet également de factoriser les règles d'extraction en utilisant des termes syntaxiques d'ordre supérieur, de façon similaire à notre approche, bien que Ranta garde une préférence pour une approche moins permissive de l'extraction, dans l'esprit de Gazdar (1981).

Mihalicek (2012) propose une modélisation de la langue serbo-croate à l'aide d'un formalisme proche des ACG et des Lambda Grammars, qui manipule des signes au sens de la section 5.1. Cependant, plutôt que d'établir le contrôle des mouvements linguistiques aux niveaux de surface à l'aide de modalités, Mihalicek reprend notre approche en associant aux types atomiques du niveau tectogrammatical un trait de type entier dénotant la présence de traces dans le constituant typé. Ces marqueurs de mouvements sont conjoints lorsque des constituants sont assemblés, et contrôlés par les types d'ordre supérieur des constituants qui initient le mouvement, comme par exemple un pronom relatif, d'une façon similaire à la stratégie que nous avons présenté dans le chapitre 4. La différence principale est que nous avons utilisé pour chaque phénomène de limitation de mouvement un niveau de contrôle dédié, dans un esprit modulaire, tandis que le trait de type entier utilisé par Mihalicek encode en même temps tous ces mécanismes de contrôle. Pour celà, chaque type de contrôle est associé à un nombre premier, et la propagation de cette information se fait par multiplication. Il est donc toujours possible de déterminer le nombre et na nature des traces présentes dans un constituant en procédant à une décomposition en facteurs premiers de l'entier qui lui est associé. De la sorte, les constituants qui doivent se comporter comme des îlots d'extraction pour un certain type d'extraction peuvent imposer au type de leur argument de ne pas contenir la marque de cette extraction, sans bloquer pour autant les autres formes. Ainsi, le verbe "misli" (pense) se voit attribuer le type

$$(\Pi ijk : int)\ s\ 3^i 5^k \multimap np\ j \multimap s\ 3^i 5^k j,$$

modélisant le fait que la phrase passée en argument peut contenir un nombre arbitraire de traces associée aux nombres premiers 3 et 5 (de façon simplifiée, traces de pronoms relatifs ou interrogatifs), mais pas de trace associée au nombre 2 (clitiques).

De son côté, le pronom interrogatif "ko" (qui) reçoit le type :

$$(\Pi i : int)\ (np\ 3 \multimap s\ 3n) \multimap s\ n$$

qui suit la même stratégie que les types attribués aux pronoms relatifs dans la section 4.2.3 : le type du pronom contraint la trace présente dans la proposition prise en argument a être étiquetée par un mouvement, et cette marque doit être retrouvée dans le type final de la proposition ($s\ 3n$), et éliminée du constituant retourné ($s\ n$).

Cette proposition montre que notre stratégie de contrôle des structures de dérivation grâce au produit dépendant est pertinente pour décrire finement les contraintes de mouvement d'une langue naturelle, qui plus est différente de l'anglais et du français, sur lesquelles reposent l'essentiel de nos intuitions linguistiques.

5.5 TAG et grammaires de lambek en ACG

Il est également intéressant de relier notre approche aux architectures comparables qui ont été utilisées pour modéliser d'autres formalismes grammaticaux, à savoir les grammaires minimalistes (MG) (Stabler, 1997), les grammaires d'adjonction d'arbres (TAG) (Joshi et Schabes, 1997), et les grammaires de Lambek non-associatives (NL) (Lambek, 1961) .

Afin d'étudier les MG d'un point de vue logique, Salvati (to appear) considère les dérivations des MG dans le formalisme des ACG. Les dérivations sont décrites à un niveau abstrait (en utilisant les opérations **move** et **merge**) et sont ensuite interprétées pour obtenir la représentation syntaxique et la représentation sémantique aux niveaux objets. Cependant, plutôt que de donner une traduction directe, il est possible d'ajouter un niveau intermédiaire qui correspond à ce qui est partagé par la syntaxe et la sémantique, mais qui contient plus que les seules dérivations des MG, ce qui renvoie à l'architecture de la figure Fig. 1.2(a) de la section 1.2.4.

Un autre exemple où une telle architecture apparaît est donné par Kanazawa et Pogodalla (2009)où un premier niveau abstrait spécifie une interface syntaxe-sémantique pour les TAG. Cette interface n'est cependant pas totalement contrainte et n'accepte pas seulement les dérivations TAG. Un niveau abstrait supplémentaire est ensuite ajouté pour contrôler ces dérivations et n'accepter que les TAG, les MCTAG locaux et les MCTAG non-locaux.

L'encodage des NL en ACG par Retoré et Salvati (2010) utilise également ce type d'architecture. Une interface syntaxe-sémantique y est définie dans le même esprit, et un niveau abstrait supérieur contrôle ensuite cette interface afin de défausser les dérivations qui ne sont pas des dérivations NL. Ce dernier résultat donne dans le cas de l'extraction un lien intéressant avec le niveau tectogrammatical des MMCG, en plus de la comparaison au niveau phénogrammatical qui est donnée dans la section 5.1, à cause de la relation entre NL et le calcul avec l'opérateur crochet de Morrill (1992) qui permet de traiter les îlots d'extraction.

Conclusion

Nous nous sommes intéressés dans ce travail à l'extension du système de typage des Grammaires catégorielles Abstraites proposée par de Groote et Maarek (2007). Nos contributions sont les suivantes :

– nous avons d'une part étudié les propriétés formelles, notamment la confluence et la normalisation, du calcul résultant. Nous en avons donné une preuve complète pour un système comprenant la β-réduction et les conversions permutatives, et nous avons présenté les problèmes posés par l'introduction de l'η-conversion, et ramené les preuves de confluence et de normalisation à la preuve de résultats plus simples ;
– nous avons d'autre part présenté une application linguistique de ces extensions, en montrant comment elles permettent de contrôler le niveau des structures de dérivation dans l'objectif de modéliser les phénomènes de contraintes sur les dépendances à distance. Nous avons à cet effet capturé les contraintes de cas pour l'extraction par un pronom relatif ou interrogatif, les contraintes d'îlots d'extraction pour ces mêmes extractions ainsi que pour les prises de portée des constituants quantificateurs, et les contraintes de séquençage inverse des pronoms interrogatifs et de leur trace.

La méthode de contrôle des mouvements que nous proposons permet de tirer parti de la faculté des ACG à contrôler leurs structures de dérivation. En outre, elle permet de rendre de façon homogène la modélisation de phénomènes de mouvements qui se manifestent en surface à des niveaux distincts (formes syntaxiques de surface et sémantique), mais qui révèlent de nombreuses similarités lorsqu'ils sont observés au niveau des structures de dérivation. Certaines de ces modélisations, à l'exemple des contraintes de cas et des modélisation d'îlots d'extraction, se révèlent traitées de façon pertinente au niveau abstrait, et les travaux de Mihalicek (2012) qui appliquent ces méthodes à la langue serbo-croate vont dans ce sens. D'autres modélisations semblent moins pertinentes au niveau abstrait, notamment lorsqu'elles sont impactées par l'ordre des mots dans la phrase, qui dans les ACG est une information écartée du niveau abstrait. Ces phénomènes, tels que les contraintes sur les extractions interrogatives multiples, semblent donc plus indiqués pour être traités au niveau des formes de surface. De façon intéressante, les techniques de contrôle sur les formes de surface, telles que l'introduction de modalités, sont orthogonales aux solutions que nous avons explorées, et il est donc possible d'appliquer la solution la plus indiquée pour chaque phénomène, selon qu'il soit lié à l'ordre des mots ou à la syntaxe profonde. Les sémantiques avec continuations constituent une autre alternative intéressante à nos travaux, car elles permettent de façon économe de capturer les contraintes de prises de portée lorsqu'une phrase comprend plus de trois constituants quantificateurs. L'introduction de ce type de sémantique dans les ACG est bien étudiée pour capturer les mécanismes du discours, et une première poursuite de notre travail serait donc d'étendre notre modélisation à ces phénomènes de prises de portée multiples, pour obtenir une comparaison plus précise à ces méthodes de continuations.

Une autre perspective de poursuite de ce travail réside dans l'identification précise des fragments de l'extension de type nécessaires pour permettre la modélisation que nous proposons, et l'étude précise des propriétés calculatoires de ce fragment. En effet, dans leur généralité, ces extensions ont un impact lourd sur le problème de l'analyse, comme montré par de Groote et al. (2007). La modélisation des langues naturelles nécessite ainsi de trouver les bons compromis entre expressivité et calculabilité.

Du côté des preuves formelles sur le λ-calcul étendu que nous avons introduit, une première perspective de prolongation de cet travail est naturellement de rendre complète la preuve de normalisation et de confluence de $\longrightarrow_{\beta\gamma\eta}$, en explorant la possibilité d'introduire un ordre bien fondé entre les termes et la forme β-normales de leur type. Il reste également, pour compléter les propriétés requises par l'utilisation du calcul étendu dans les ACG, à prouver la décidabilité de son problème d'inférence de type, et à en obtenir un algorithme.

Bibliographie

Kazimierz Adjukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1 :1–27, 1935. English translation “Syntactic Connexion" by H. Weber in McCall, S. (Ed.) *Polish Logic*, pp. 207–231, Oxford University Press, Oxford, 1967.

Vincent Balat, Roberto Di Cosmo, et Marcelo Fiore. Extensional normalisation and type-directed partial evaluation for typed lamda calculus with sums. In *31st Ann. ACM Symp. on Principles of Programming Languages (POPL)*, pages 64–76. ACM, 2004. doi : http://dx.doi.org/10.1145/982962.964007.

Y. Bar-Hillel, C. Caifman, et E. Shamir. *On Categorial and Phrase-structure Grammars*. Weizmann Science Press, 1960.

Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29 :47–58, 1953. Reprinted in Y. Bar-Hillel. (1964). *Language and Information : Selected Essays on their Theory and Application*, Addison-Wesley 1964, 61–74.

Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

Henk Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2) :125–154, 1991.

Chris Barker. Continuations and the nature of quantification. *Natural Language Semantics*, 10(3) :211–242, 2002. ISSN 0925-854X. doi : 10.1023/A:1022183511876.

Gilles Barthe. Existence and uniqueness of normal forms in pure type systems with $\beta\eta$-conversion. In *Proceedings of CSL'98, volume 1584 of Lecture Notes in Computer Science*, pages 24125–9. Springer-Verlag, 1999a.

Gilles Barthe. Expanding the cube. In *Proc. nd FOSSACS 1999 (Amsterdam), volume 1578 of lncs*, pages 90–103. Springer-Verlag, 1999b.

Raffaella Bernardi. Scope ambiguities through the mirror. In H. De Mulder, M. Everaert, T. Lentz, O. Nilsen, et A. Zondervan, editors, *The Linguistics Enterprise : from knowledge of language to knowledge in linguistics.* "Benjamins", 2009.

Iliano Cervesato et Frank Pfenning. A linear logical framework. In *LICS*, pages 264–275, 1996.

Noam Chomsky. On wh-movement. In *Formal Syntax*, pages 71–132, 1977.

Noam Chomsky. *The Minimalist Program*. MIT Press, 1995.

Joel M. Cohen. The equivalence of two concepts of categorial grammar. 10(5) :475–484, 1967.

R. Cooper. *Montague's Semantic Theory and Transformational Syntax*. University of Massachusetts, 1975.

Philippe de Groote. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155, 2001.

Philippe de Groote. Tree-adjoining grammars as abstract categorial grammars. In *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia, 2002.

Philippe de Groote et Sarah Maarek. Type-theoretic extensions of abstract categorial grammars. In *New Directions in Type-Theoretic Grammars, proceedings of the workshop*, pages 18–30, 2007. http://let.uvt.nl/general/people/rmuskens/ndttg/ndttg2007.pdf.

Philippe de Groote et Sylvain Pogodalla. On the expressive power of abstract categorial grammars : Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4) :421–438, 2004. http://hal.inria.fr/inria-00112956/fr/.

Philippe de Groote et Sylvain Pogodalla. m-linear context-free rewriting systems as abstract categorial grammars. In R. T. Oehrle et J. Rogers, editor, *Proceedings of Mathematics of Language - MOL-8, Bloomington, Indiana, États-Unis*, pages 71–80, Jun 2003. URL http://www.loria.fr/publications/2003/A03-R-243/A03-R-243.ps.

Philippe de Groote, Ryo Yoshinaka, et Sarah Maarek. On two extensions of abstract categorial grammars. In Nachum Dershowitz et Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2007. ISBN 978-3-540-75558-6.

Philippe de Groote, Sylvain Pogodalla, et Carl Pollard. On the syntax-semantics interface : From convergent grammar to abstract categorial grammar. In Makoto Kanazawa, Hiroakira Ono, et Ruy de Queiroz, editors, *16th Workshop on Logic, Language, Information and Computation*, volume 5514, pages 182–196, Japon Tokyo, 2009. Springer. http://hal.inria.fr/inria-00390490/en/.

Philippe De Groote, Sylvain Pogodalla, et Carl Pollard. About Parallel and Syntactocentric Formalisms : A Perspective from the Encoding of Convergent Grammar into Abstract Categorial Grammar. *Fundamenta Informaticae*, 106(2-4) :211–231, 2011. doi : 10.3233/FI-2011-384. URL http://hal.inria.fr/inria-00565598.

Gilles Dowek. The undecidability of typability in the lambda-pi-calculus. In *TLCA*, pages 139–145, 1993.

Gilles Dowek et Benjamin Werner. On the definition of the eta-long normal form in type systems of the cube. In *Informal Proceedings of the Workshop on Types for Proofs and Programs*, 1993.

G. Gazdar. Unbounded dependencies and coordinate structure. 1981.

Neil Ghani. Eta-expansions in dependent type theory - the calculus of constructions. In Philippe Groote et J. Roger Hindley, editors, *Typed Lambda Calculi and Applications*, volume 1210 of *Lecture Notes in Computer Science*, pages 164–180. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-62688-6. doi : 10.1007/3-540-62688-3_35. URL http://dx.doi.org/10.1007/3-540-62688-3_35.

J. Girard, Y. Lafont, et P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

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

Felix Joachimski. Syntactic analysis of eta-expansions in pure type systems. *Inf. Comput.*, 182(1) :53–71, 2003. URL http://dblp.uni-trier.de/db/journals/iandc/iandc182.html#Joachimski03.

Mark Johnson. Proof nets and the complexity of processing center embedded constructions. *Journal of Logic, Language and Information*, 7(4), 1998.

Aravin K. Joshi et Yves Schabes. Tree-adjoining grammars. In G. Rozenberg et A. Salomaa, editors, *Handbook of formal languages*, chapter 2. Springer, 1997.

Makoto Kanazawa. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 176–183, Prague, Czech Republic, June 2007. Association for Computational Linguistics. http://www.aclweb.org/anthology/P/P07/P07-1023.

Makoto Kanazawa et Sylvain Pogodalla. Advances in abstract categorial grammars : Language theory and linguistic modelling. ESSLLI 2009 Lecture Notes, 2009. URL http://www.loria.fr/equipes/calligramme/acg/publications/esslli-09/2009-esslli-acg-week-2-part-2.pdf.

J. W. Klop. Term rewriting systems, 1992.

Yusuke Kubota et Carl Pollard. Phonological interpretation into preordered algebras. In *Proceedings of the 11th Meeting of the Association for Mathematics of Language (MOL'11)*, 2009. http://www.ling.ohio-state.edu/~kubota/papers/mol11_proc.pdf.

Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3) :154–170, 1958.

Joachim Lambek. On the calculus of syntactic types. In R. Jacobsen, editor, *Structure of Language and its Mathematical Aspects*, Proceedings of Symposia in Applied Mathematics, XII. American Mathematical Society, 1961.

Ekaterina Lebedeva. *Expression de la dynamique du discours à l'aide de continuations*. PhD thesis, Université de Lorraine, 2012.

Sam Lindley. Extensional rewriting with sums. In *Proceeding TLCA'07 Proceedings of the 8th international conference on Typed lambda calculi and applications*, pages 255–271. Springer-Verlag Berlin, 2007.

V. Mihalicek. *Serbo-croatian Word Order : A Logical Approach*. PhD thesis, 2012.

Richard Montague. *Formal Philosophy ; Selected Papers of Richard Montague*. New Haven,Yale University Press, 1974.

Michael Moortgat. Generalized quantifiers and discontinuous type constructors. In *In Arthur Horck and Wietske Sijtsma, editors, Discontinuous Constituency, Berlin. Mouton de Gruyter*, 1992.

Michael Moortgat. Categorial type logics. In Johan van Benthem et Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier Science Publishers, Amsterdam, 1996.

Richard Moot et Mario Piazza. Linguistic applications of first order intuitionistic linear logic. *Journal of Logic, Language and Information*, 10 :211–232, 2001.

Richard Moot et Christian Rétoré. *The logic of categorial grammars*. Springer, 2012.

Glyn Morrill. Categorial formalisation of relativisation : Islands, extraction sites and pied piping. Technical Report LSI-92-23-R,, Departament de Llenguatges i Sistemes Informtics, Universitat Politcnica de Catalunya, 1992.

Glyn V. Morrill. Incremental processing and acceptability. *Computational Linguistics*, 26(3) :319–338, September 2000.

Reinhard Muskens. Separating syntax and combinatorics in categorial grammar. *Research on Language and Computation*, 5(3) :267–285, September 2007.

Mati Pentus. Lambek grammars are context free. In *LICS*, pages 429–433, 1993.

Sylvain Pogodalla. Computing semantic representation : Towards ACG abstract terms as derivation trees. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 64–71, May 2004. http://www.cs.rutgers.edu/TAG+7/papers/pogodalla.pdf.

Sylvain Pogodalla. Generalizing a proof-theoretic account of scope ambiguity. In Jeroen Geertzen, Elias Thijsse, Harry Bunt, et Amanda Schiffrin, editors, *Proceedings of the 7th International Workshop on Computational Semantics - IWCS-7*, pages 154–165. Tilburg University, Deparment of Communication and Information Sciences, 2007. http://hal.inria.fr/inria-00112898.

Carl Pollard. An introduction to convergent grammar, 2008.

Carl Pollard. communication privée, 2010.

Aarne Ranta. Computational semantics in type theory. *MATHEMATICS AND SOCIAL SCIENCES*, 165 :31–57, 2004.

Christian Retoré et Sylvain Salvati. A faithful representation of non-associative lambek grammars in abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2) :185–200, 2010. `http://www.springerlink.com/content/f48544n414594gw4/`.

E.G. Ruys et Yoad Winter. Quantifier scope in formal linguistics. In Dov M. Gabbay et Franz Guenthner, editors, *Handbook of Philosophical Logic*, volume 16 of *Handbook of Philosophical Logic*, pages 159–225. Springer Netherlands, 2011. ISBN 978-94-007-0478-7. doi : 10.1007/978-94-007-0479-4_3. URL `http://dx.doi.org/10.1007/978-94-007-0479-4_3`.

Sylvain Salvati. *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*. PhD thesis, Institut National Polytechnique de Lorraine, 2005.

Sylvain Salvati. Minimalist grammars in the light of logic. to appear.

Chung-Chieh Shan. Delimited continuations in natural language : Quantification and polarity sensitivity. In Hayo Thielecke, editor, *Proceedings of the 4th continuations workshop*, pages 55–64. School of Computer Science, University of Birmingham, 2004.

Stuart M. Shieber. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, Trento, Italy, 3–7 April 2006. `http://www.aclweb.org/anthology-new/E/E06/E06-1048.pdf`.

Edward Stabler. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *LNCS/LNAI*, pages 68–95. Springer-Verlag, 1997. ISBN 3-540-63700-1.

Ryo Yoshinaka et Makoto Kanazawa. The complexity and generative capacity of lexicalized abstract categorial grammars. In *LACL*, pages 330–346, 2005.

Résumé

Cette thèse s'intéresse à la modélisation de la syntaxe et de l'interface syntaxe-sémantique de la phrase, et explore la possibilité de contrôler au niveau des structures de dérivation la surgénération que produit le traitement des dépendances à distance par des types d'ordre supérieur. À cet effet, nous étudions la possibilité d'étendre le système de typage des Grammaires Catégorielles Abstraites avec les constructions de la somme disjointe, du produit cartésien et du produit dépendant, permettant d'étiqueter les catégories syntaxiques par des structures de traits. Nous prouvons dans un premier temps que le calcul résultant de cette extension bénéficie des propriétés de confluence et de normalisation, permettant d'identifier les termes β-équivalents dans le formalisme grammatical. Nous réduisons de plus le même problème pour la $\beta\eta$-équivalence à un ensemble d'hypothèse de départ. Dans un second temps, nous montrons comment cette introduction de structures de traits peut être appliquée au contrôle des dépendances à distances, à travers les exemples des contraintes de cas, des îlots d'extraction pour les mouvements explicites et implicites, et des extractions interrogatives multiples, et nous discutons de la pertinence de placer ces contrôles sur les structures de dérivation.

Mots-clés : Traitement Automatique de la Langue, méthodes symboliques, grammaires catégorielles, systèmes de typage, dépendances à distance, lambda-calcul.

Abstract

This thesis focuses on the modelisation of syntax and syntax-semantics interface of sentences, and investigate how the control of the surgeneration caused by the treatment of linguistics movements with higher order types can take place at the level of derivation structures. For this purpose, we look at the possibility to extend the type system of Abstract Categorial Grammars with the constructions of disjoint sum, cartesian product and dependent product, which enable syntactic categories to be labeled by feature structures. At first, we demonstrate that the calculus associated with this extension enjoy the properties of confluence and normalization, by which β-equivalence can be computed in the grammatical formalism. We also reduce the same problem for $\beta\eta$-equivalence to a few hypothesis. Then, we show how this feature structures can be used to control linguistics movements, through the examples of case constraints, extraction islands for overt and covert movements and multiples interrogative extractions, and we discuss the relevancy of operating these controls on the derivation structures.

Keywords : Natural Language Processing, symbolic methods, categorial grammars, type systems, linguistics movements, lambda-calculus.