Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Problèmes de Filtrage et Problèmes d'analyse pour les Grammaires Catégorielles Abstraites

THÈSE

présentée et soutenue publiquement le 13 juin 2005

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Sylvain Salvati

Composition du jury

Rapporteurs :	Eric de la Clergerie	
	Alexandre Dikovsky	
Examinateurs :	Philippe de Groote	(directeur de thèse)
	Dale Miller	
	Glyn Morrill	
	Karl Tombre	

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

Mis en page avec la classe thloria.

à Juliette, à Arthur.

Résumé

Ce travail de thèse contient une étude détaillée de la complexité de l'analyse des grammaires catégorielles abstraites et il présente des algorithmes d'analyse pour ce formalisme. Les grammaires catégorielles abstraites ont été introduites par Philippe de Groote [dG01] comme un cadre formel pour l'étude du langage et des liens entre ses diverses dimensions. Les grammaires catégorielles abstraites utilisent comme bases formelles la logique linéaire et le λ-calcul linéaire.

Une partie substantielle de ce travail est consacrée à l'étude du filtrage dans le λ-calcul linéaire ; la conception des algorithmes d'analyse proposés est issue pour une bonne part de cette étude.

Mots-clés: logique linéaire, λ-calcul, filtrage, grammaires catégorielles, analyse, linguistique computationnelle.

Abstract

This thesis presents detailed results about complexity of parsing in abstract catégorial grammars and parsing algorithms for those grammars. Abstract categorial grammars were introduced by Philippe de groote [dG01] as a framework for modeling natural language and the connections between its dimensions. Abstract categorial grammars are based on linear logic and linear λ-calculus.

An important part of this work is dedicated to the study of matching in the linear λ-calculus. Most of the ideas involved in the design of the parsing algorithms we present come from that study.

Keywords: linear logic, λ-calculus, matching, categorial grammars, parsing, computationnal linguistics.

Remerciements

Je tiens à remercier ici Philippe de Groote pour avoir encadré mon travail de thèse. Les explications impromptues et de grandes qualités qu'il a pu me fournir sur de nombreux sujets ont été pour moi un gain de temps considérable. Les discussions que nous avons eues sur la plupart des problèmes que j'ai rencontrés dans cette thèse ont souvent marqué des étapes importantes dans l'avancement de mon activité de recherche. Grâce à ses conseils et à sa rigueur scientifique j'ai pu concrétiser les intuitions à la base de ce travail.

Je tiens également à remercier Luigi Liquori pour avoir encadré mon stage de DEA et pour m'avoir donné le goût de la recherche. Il est probable que sans lui je n'aurais pas entrepris d'études doctorales.

Je remercie Eric de la Clergerie et Alexandre Dikovsky pour avoir accepté d'être les rapporteurs de cette thèse et pour les remarques qu'ils ont faites sur ce travail. Je tiens à remercier les autres membres du jury, Dale Miller, Glyn Morrill et Karl Tombre pour avoir manifesté leur intérêt pour mon travail en acceptant d'en être les examinateurs.

L'équipe Calligramme, pour son accueil, la sympathie de ses membres et la qualité des échanges scientifiques que j'ai eus avec eux a été pour moi une source de soutiens et de conseils sur laquelle j'ai pu compter à chaque instant. Aussi, je remercie chaudement Bruno, Denys, François, Guillaume, Guy, Jean-Yves et Sylvain pour m'avoir offert un cadre scientifique de haut niveau ainsi qu'une ambiance chaleureuse.

J'exprime également toute ma gratitude à ceux qui ont peuplés mon bureau, ses environs et le bâtiment de Saint Fiacre pendant que j'y étais. Je m'adresse tout d'abord à Jérômes avec qui j'ai partagé mon bureau presque tout au long de ma thèse. Lui et son humour caustique envolés, Saint-Fiacre n'a plus été le même pour moi. À Joseph je tiens à exprimer mon admiration pour avoir supporté avec le sourire mes bavardages à tout crin et pour y avoir plus qu'à l'envie apporté la réplique. Je remercie Benjamin pour sa joie de vivre qui transparaît même à travers le brouillard de ses nuits trop courtes. Je remercie Laïka pour avoir brisé la monotonie des pluies et des cieux gris de Lorraine avec sa bonne humeur. Je remercie Clara pour m'avoir rappelé un sud quitté depuis si longtemps. Il me faut maintenant saluer tous les compagnons de thèse Prothéo qui comme moi oscillent entre doute et doute à chaque fois qu'ils croient approcher de la vérité.

Finalement, je remercie Juliette et Arthur de me rendre heureux.

Table des matières

Introduction

Les langues parlées par les êtres humains à travers la planète sont d'une grande diversité. Les linguistes répertorient, catégorisent et comparent les diverses formes qu'ils rencontrent dans cette multitude et par delà cette masse hétérogène en apparence, ils étudient la faculté de langage de l'être humain. Dans chaque langue, les phrases se construisent en respectant une certaine régularité, une certaine structure. Pour modéliser ces régularités et ces structures, les linguistes édictent un ensemble de règles à partir desquelles on doit pouvoir construire toute phrase dans une langue et analyser chaque énoncé formulé dans cette langue. L'analyse de ces règles fait apparaître des similitudes d'une langue à une autre. Intégrer l'ensemble de ces similitudes, trouver un moyen suffisamment général pour pouvoir décrire tous les phénomènes que l'on peut rencontrer à travers toutes les langues, c'est ce à quoi s'attachent les linguistes. Le foisonnement et l'évolution permanente des langues rendent difficile la validation de tel ou tel modèle du langage. Pour faciliter ses investigations, la tradition linguistique sépare divers aspects du langage, elle distingue les phénomènes phonologiques, morphologiques, syntagmatiques, syntaxiques, sémantiques, discursifs... Les règles énoncées par les linguistes se répartissent dans les diverses dimensions linguistiques. Un modèle exhaustif de la langue se définit par la mise en rapport de différents modèles pour chacune des dimensions du langage. Confronter un tel modèle linguistique à la réalité de la langue est d'autant plus ardu qu'il faut instancier pour chaque dimension un modèle complexe et préciser les relations entre les dimensions. Il ne s'agit donc pas de vérifier la pertinence d'un modèle syntaxique ou d'un modèle sémantique de la langue indépendamment l'un de l'autre, mais de voir si l'interaction de ces deux modèles est pertinente par rapport à la réalité du langage.

Tout comme pour la physique, les mathématiques ont permis à la linguistique d'objectiver un ensemble de concepts en leur donnant une définition rigoureuse et qui autorise un débat rationnel sur les modèles possibles du langage. La formalisation des divers domaines du langage étudiés par les linguistes a conduit à un nombre important de formulations mathématiques pour chacun d'entre eux. Dans chacun des formalismes qui ont émergé, les modélisations de la langue sont également d'une grande variété. Les développements de l'informatique ont permis de créer des logiciels qui permettent dans une certaine mesure la validation de certains modèles. Parfois même l'expérimentation est poussée jusqu'à la construction de logiciels qui intègrent plusieurs dimensions linguistiques. Mais les choix effectués sont souvent plus des conséquences des formalismes choisis que celles de réelles motivations linguistiques. De plus, ces approches ne permettent pas de comparer les divers formalismes et d'expérimenter le rapport entre les modèles de chaque dimension du langage. Doter les linguistes d'outils qui permettent l'intégration des nombreux modèles qu'ils proposent, leur donner une flexibilité suffisante pour pouvoir faire interagir différents modèles de différentes dimensions linguistiques, voilà ce que se proposent de faire les mathématiciens et les informaticiens. En retour, les linguistes, en explorant les limites de ces systèmes, leur proposent de nouveaux problèmes mathématiques à résoudre. Cette interaction entre ces disciplines fait émerger des outils d'exploration du langage de plus en plus sophistiqués et permet la création d'applications basées sur la langue.

Les grammaires catégorielles abstraites [dG01] sont une formalisation mathématique dont l'ambition est d'embrasser l'ensemble des dimensions du langage. Ces grammaires se veulent un laboratoire d'expérimentation pour une gamme étendue de formalismes qui modélisent la langue afin d'accomplir le projet d'unification des diverses dimensions du langage et de permettre l'étude précise de leurs interactions. Les grammaires catégorielles abstraites utilisent le λ-calcul comme moyen pour manipuler les divers objets employés pour modéliser la langue, qu'il s'agisse de phonèmes, de mots, de prédicats, de DRS... La théorie des types apporte une structuration abstraite qui sert de support aux concepts linguistiques. Elle contrôle

également l'agencement des λ-termes plongeant ainsi les concepts linguistiques dans la syntaxe des langages. Finalement, et c'est là un point central des grammaires catégorielles abstraites, elles utilisent une notion de lexique qui permet de passer d'une dimension linguistique à une autre. Par là elles étendent à l'ensemble du langage une notion que Montague [Mon74] réservait à la sémantique. Cette notion de lexique permet de voir les dimensions du langage dans un rapport d'abstraction les unes par rapport aux autres. Ainsi, on peut voir la dimension syntagmatique comme une vision abstraite à la fois des structures de surface et de la sémantique. On peut également voir les structures de surface comme des abstractions de la dimension phonologique. Mais ce rapport d'abstraction n'est nullement figé, il peut tout à fait être défini différemment. Par ailleurs, la correspondance de Curry-Howard, le rapport qu'elle établit entre logique et calcul, et la connaissance déjà accumulée autour de la théorie de types font des grammaires catégorielles abstraites un système ouvert susceptible d'évoluer suivant les nécessités rencontrées au cours de la modélisation du langage.

Le présent travail s'intéresse à l'étude mathématique des grammaires catégorielles abstraites. Les questions de la décidabilité des formalismes, de la difficulté intrinsèque des problèmes d'analyse et des façons dont on peut automatiser différentes tâches inhérentes au travail sur le langage... sont essentielles pour valider mathématiquement un formalisme dédié à la modélisation de la langue. Comment par exemple considérer un tel formalisme où la plupart des tâches que peut effectuer un être humain sans trop d'efforts s'avèrent être indécidables ? On attend des propositions formelles de modélisation du langage qu'elles possèdent des caractéristiques qui rendent compte de la faculté de langage des êtres humains. Cette faculté va de l'apprentissage, à la compréhension relativement aisée en passant le cas échéant par une perception convenable des phonèmes. Il faut donc que les formalismes qui cherchent à modéliser cette faculté aient des instanciations qui puissent être apprises (au moins les instanciations qui correspondent à des langues existantes), il faut également que le processus de compréhension (*i.e.* l'ensemble des tâches qui permettent d'obtenir un sens à partir d'une phrase ou d'un texte) soit raisonnablement simple.

Afin de montrer que les grammaires catégorielles abstraites sont un outil pertinent pour le traitement automatique des langues, cette thèse s'intéresse au problème de l'analyse des grammaires catégorielles abstraites. Pour tout formalisme grammatical ce problème est central. Tout d'abord, il permet de validité sa pertinence, car de la complexité de ce problème découle celle de la compréhension des énoncés, si cette complexité s'avère trop élevée, il est difficile d'imaginer que le formalisme proposé permette de modéliser le langage. Aussi ce mémoire présente-t-il une étude détaillée de la complexité de l'analyse des grammaires catégorielles abstraites.

Afin de mener à bien cette étude de complexité et obtenir quelques intuitions sur les résultats que nous pouvions obtenir, il nous a fallu voir le processus d'analyse comme la combinaison de deux processus distinct.

Le premier de ces processus est la recherche de démonstration dans la logique linéaire. La complexité de la recherche de démonstration dans les divers fragments de la logique linéaire est en général connu. Dans le cas des grammaires catégorielles abstraites, les fragments qui nous ont intéressés sont la logique linéaire implicative et intuitionniste et la logique linéaire implicative exponentielle et intuitionniste. Pour la première de ces logiques la recherche de démonstration est NP-complète. Quant à la décidabilité de la seconde elle est liée à la décidabilité de la logique linéaire multiplicative et exponentielle (**MELL**) qui est inconnue. On sait en revanche que sa complexité est très élevée.

Le second processus utile à l'analyse est la résolution d'équations de filtrage dans le λ-calcul linéaire. Lorsque l'on analyse une phrase, on découpe celle-ci en constituants auxquels on associe une catégorie syntaxique. Par exemple, si l'on considère la règle hors contexte $S \to NPVNP$ qui modélise la phrase simple comme la juxtaposition d'un groupe nominal sujet, d'un groupe verbal puis d'un groupe nominal complément, alors chercher à analyser une phrase simple ω revient implicitement à résoudre l'équation $\mathbf{X}\,\mathbf{Y}\,\mathbf{Z} = \omega$ où $\mathbf{X}$ représente le premier groupe nominal, $\mathbf{Y}$ représente le groupe verbal et $\mathbf{Z}$ représente le second groupe nominal. Pour les grammaires catégorielles abstraites, les équations à résoudre sont plus complexes et l'étude de ces équations nous a été d'un grand secours pour résoudre la plupart des autres problèmes que nous avons abordés par la suite.

Suite à l'étude consacrée à la complexité de l'analyse des grammaires catégorielles abstraites, il est apparu qu'une large classe de grammaires catégorielles abstraites ne possède pas les propriétés de complexité nécessaires à en faire un bon candidat pour modéliser la syntaxe du langage. Cependant nous avons défini un semi-algorithme qui permet d'analyser n'importe quelle grammaire catégorielle abstraite.

En nous basant sur les résultats de complexité que nous avons obtenus, nous proposons deux autres algorithmes d'analyse particuliers à deux classes distinctes de grammaires. Le premier de ces algorithmes s'appuie essentiellement sur un algorithme de recherche de démonstration dans la logique linéaire implicative intuitionniste. Il s'inpire d'une tradition linguistique qui s'articule autour du calcul de Lambek [Lam58]. Le second de ces algorithmes s'inscrit dans une tradition plus ancienne, issue des travaux concernant l'un des premiers formalismes étudiés pour modéliser le langage, les grammaires hors contexte.

Pour ce type de grammaires le problème de l'analyse a été étudié depuis leur apparition [Cho56]. Le premier algorithme efficace qui permette de résoudre le problème de l'analyse de grammaires hors contexte a été proposé par Cocke, Kasami et Younger ([Kas65], [You67]). Cet algorithme s'exécute en temps cubique pour les grammaires hors contexte sous forme normale de Chomsky. Bien qu'une grammaire hors contexte puisse être mise sous forme normale de Chomsky en temps linéaire, effectuer l'analyse d'une grammaire transformée ne permet pas un accés directe aux structures d'analyse de la grammaire initiale. De plus il a également pour défaut de s'exécuter systématiquement en temps cubique. On ne peut espérer qu'il se comporte mieux dans des cas relativement simples.

Plus tard, Earley [Ear70] a apporté une solution au problème de l'analyse qui permet d'effectuer une analyse au plus en temps cubique, et peut dans certains cas s'exécuter en temps linéaire. Cet algorithme essaie de parcourir la phrase à analyser de gauche à droite et s'arrête lorsque son parcours ne peut plus poursuivre.

En s'appuyant sur l'idée qui consiste à arrêter une analyse dès que l'on trouve une erreur, Tomita [Tom87] propose de généraliser les techniques jusques-là réservées à l'analyse des langages de programmation construit à partir de grammaire hors contexte déterministe. Il définit un algorithme LR (de gauche à droite) généralisé. En théorie cet algorithme ne s'exécute pas en temps cubique, cependant, dans la pratique, il s'avère qu'il fontionne plus rapidement que l'algorithme de Earley. Cet algorithme a été amélioré par Kipps [Kip91] pour qu'il s'exécute en temps cubique, mais ce nouvel algorithme se révèle être moins performant que celui de Tomita. Satta et Nerderhof en s'inspirant d'idées de Kipps [NS96] ont réussi à améliorer tant en espace qu'en temps les performances de l'algorithme de Tomita, sans pour autant que ce nouvel algorithme n'ait une complexité théorique qui soit cubique en temps.

Une dernière approche est proposée par Nerderhof [Ned93] et consiste en une analyse LC (Left-Corner). Ce nouvel lgorithme peut être moins performant que celui de Tomita, mais il permet de générer des analyseurs plus petits et plus rapidement. De plus, il permet également d'obtenir des forêts de dérivations plus compactes.

Depuis leur création, voici près de cinquante ans, les grammaires hors contexte ont suscité une forte activité de recherche autour du problème de leur analyse. Ces recherches ont permis de concevoir des analyseurs toujours plus performants. Il faut noter que ces grammaires ne permettent de modéliser convenablement la langue naturelle [Shi85]. Cependant, l'étude de ces grammaires relativement simples a apporté un ensemble d'éléments conceptuels qui on permis de concevoir des analyseurs efficaces pour d'autres types de grammaires.

Cela est particulièrement le cas pour les grammaires d'arbres adjoints [AKJ75]. Les premiers algorithmes proposés pour ces grammaires étaient dans le style de l'algorithme de Cocke-Kasami-Younger pour les grammaires hors contexte [VSJ85]. Par la suite, il a été possible de définir un algorithme similaire à l'algorithme de par Earley pour les grammaires hors contexte [SJ88]. Cependant cet algorithme ne présente pas la propriété de validité de préfixe, propriété que l'algorithme de Earley pour les grammaires hors contexte possède. Cet propriété permet de contraindre le fait que toute portion de phrase analysée par l'algorithme puisse effectivement être produite par la grammaire du moment qu'on la complète par son préfixe. Les premiers algorithmes connus qui possèdent cette propriété avaient à ce moment une complexité en $\mathcal{O}(n^9)$. Nederhof est finalement parvenu à proposer un algorithme en $\mathcal{O}(n^6)$ et qui possède cette propriété [Ned99].

Nous proposons pour une certaine classe de grammaire catégorielle abstraite un algorithme *de type Earley*. Cet algorithme est *de type Earley* en ce sens qu'il se comporte exactement comme l'algorithme de Earley pour les grammaires catégorielles abstraites qui codent des grammaires hors contexte. De plus il permet d'analyser les grammaires catégorielles abstraites qui codent les grammaires d'arbres adjoints en $\mathcal{O}(n^6)$. Tout ce travail s'inscrit pour une bonne part dans cette tradition de conception d'algorithmes d'analyse. Pour y parvenir, il nous a cependant fallu définir des concepts relativement complexes comme les descriptions syntaxiques ou les types abstraits pointés qui permettent de faire passer à l'ordre supérieur

des idées qui jusques-là s'exprimaient simplement, comme les indices utilisées pour délimités les portions de phrase analysées ou encore la correspondance entre le type d'un pied d'adjonction et l'arbre qu'on lui substitue pour les grammaires d'arbres adjoints...

Plan de la thèse

Logique linéaire intuitionniste implicative et λ-calcul

Dans ce premier chapitre, nous présentons rapidement la logique linéaire intuitionniste implicative. Nous insistons en particulier sur la décidabilité et la complexité de cette logique. Les résultats concernant ces problèmes seront utiles par la suite lors de l'étude de la décidabilité et de la complexité des grammaires catégorielles abstraites.

Ce chapitre présente également le λ-calcul non typé, puis le λ-calcul linéaire associés par la correspondance de Curry-Howard à la logique linéaire intuitionniste implicative. Finalement, ce chapitre présente un algorithme original de recherche de démonstration pour la logique linéaire intuitionniste implicative. Bien que cet algorithme se base sur le λ-calcul, il utilise implicitement la notion de réseau de démonstration. L'algorithme d'analyse incrémental présenté par la suite se base sur cet algorithme de recherche de démonstration.

Les grammaires catégorielles abstraites

Partant des grammaires de type logique et de la façon dont elles permettent de relier la syntaxe et la sémantique, ce chapitre montre comment, en généralisant les idées de Montague dévolues à la sémantique, les grammaires catégorielles abstraites s'articulent par rapport à la tradition des grammaires catégorielles. Par la suite, nous définissons formellement la notion de grammaire catégorielle abstraite ; puis nous montrons comment les grammaires catégorielles abstraites permettent d'encoder divers formalismes syntaxiques classiques.

Algorithme général d'analyse

Ce court chapitre présente un algorithme d'analyse général pour les grammaires catégorielles abstraites. Les problèmes posés par cet algorithme sont à la base de l'étude mathématique qui suit.

Filtrage dans le λ-calcul linéaire

Ce chapitre est consacré à l'étude du problème de filtrage dans le λ-calcul linéaire. Le point de départ de cette étude est le résultat de NP-complétude du filtrage dans le λ-calcul linéaire montré par Philippe de Groote [dG00a]. Ce résultat est affiné pour montrer que même dans des cas excessivement contraints, le filtrage reste NP-complet.

Algorithmes de filtrage dans le λ-calcul linéaire

Après l'étude de la difficulté du problème de filtrage dans le λ-calcul linéaire, ce chapitre présente des algorithmes de résolution d'équation. Le premier de ces algorithmes est une adaptation de l'algorithme d'unification d'ordre supérieur de Huet [Hue76]. Afin de réduire l'indéterminisme de cet algorithme dans le cas où les inconnues ont des occurrences linéaires, nous proposons une condition nécessaire à l'existence d'une solution. Enfin, ce chapitre présente un système de typage qui permet de représenter des égalités dans le λ-calcul linéaire. À partir de ce système de typage, nous présentons un algorithme de résolution d'équations de filtrage qui utilise des techniques de tabulation.

Décidabilité et complexité de l'analyse

Ce chapitre montre qu'en général, la décidabilité de l'analyse dans les grammaires catégorielles abstraites est au moins aussi difficile que celle du fragment multiplicatif et exponentiel de la logique linéaire.

Ensuite, nous donnons des résultats de complexité pour des ensembles de grammaires catégorielles abstraites plus ou moins contraints. Nous nous intéressons plus particulièrement aux grammaires lexicalisées pour lesquelles nous montrons qu'en général l'analyse est NP-complète. Nous montrons également que si on considère des grammaires catégorielles abstraites dont les langages abstraits sont du second ordre alors l'analyse est polynômiale.

Algorithmes pratiques d'analyse

Au début de ce chapitre, nous revenons sur l'algorithme que nous avions présenté au troisième chapitre pour étudier son comportement à la lumière des résultats obtenus au chapitre 6. Nous montrons qu'en utilisant des techniques de tabulation il termine dans les cas où on sait l'analyse décidable. Cependant, cet algorithme demeure peu efficace. Aussi par la suite, ce chapitre est consacré à l'étude d'algorithmes dédiés à des cas particuliers. Le premier est destiné à l'analyse de grammaires lexicalisées. L'intérêt principal de cet algorithme réside dans la relation étroite qu'il entretient avec des propriétés linguistiques intéressantes. Le second algorithme permet d'analyser en temps polynômial les grammaires catégorielles abstraites dont les langages abstraits sont du second ordre. Pour cela il utilise le système de typage que nous avons développé pour résoudre des équations de filtrage et utilise des techniques de tabulation. Nous verrons que cet algorithme est une généralisation pour les grammaires catégorielles abstraites de l'algorithme d'analyse proposé par Earley pour les grammaires hors contexte et qu'il permet également d'analyser les grammaires catégorielles abstraites qui représentent des grammaires d'arbres adjoints en $\mathcal{O}(n^6)$.

Chapitre 1

Logique linéaire implicative et λ-calcul

Dans ce chapitre, nous allons présenter les fragments de la logique linéaire qui nous seront nécessaires pour définir et étudier les grammaires catégorielles abstraites. Nous allons également présenter le λ-calcul et plus particulièrement le λ-calcul linéaire simplement typé utilisé par les grammaires catégorielles abstraites. Enfin, nous allons présenter un algorithme de recherche de démonstration dans la logique linéaire intuitionniste implicative.

1.1 Logique linéaire intuitionniste implicative

La logique linéaire a été définie par Jean-Yves Girard [Gir87] comme une logique qui permet de raisonner en termes de ressources. Dans le cadre de la langue naturelle ce type de logique permet de modéliser la façon dont des ressources grammaticales peuvent se combiner pour en produire d'autres. Pour définir les grammaires catégorielles abstraites le fragment intuitionniste et implicatif de cette logique est suffisant. Afin de préciser certaines propriétés combinatoires des grammaires catégorielles abstraites, il nous est cependant nécessaire d'introduire également le fragment exponentiel de la logique linéaire.

La logique linéaire peut se voir comme une logique obtenue à partir de la logique intuitionniste par l'abandon des règles structurelles d'affaiblissement et de contraction. Elle conserve cependant la règle d'échange. Afin de simplifier les notations ainsi que les démonstrations, nous allons procéder de la même façon que Troelstra [Tro92] en codant la règle d'échange dans la structure avec laquelle nous allons modéliser les contextes.

Aussi, avant de définir la logique linéaire, nous allons nous doter de quelques outils et notations.

Définition 1.1 *Un multi-ensemble d'éléments d'un ensemble E est une fonction M de E dans $\mathbb{N}$ telle que l'ensemble des éléments e de E tel que $M(e) \neq 0$ est fini.*

Souvent nous confondrons les ensembles finis avec les multi-ensembles M tel que pour tout e, $M(e) \leq 1$.

Définition 1.2 *On dit qu'un multi-ensemble M_1 est inclus dans un multi-ensemble M_2, ce que nous notons $M_1 \subseteq M_2$, si pour tout e, $M_1(e) \leq M_2(e)$.*

Définition 1.3 *Une liste sur un ensemble E est une suite finie $(e_k)_{k\in[1,n]}$ d'éléments de E. On dit que N est le multi-ensemble est associé à la liste $(e_k)_{k\in[1,n]}$ si $N(e) = p$ si et seulement si il existe exactement p éléments k de $[1, n]$ tels que $e_k = e$.*

Définition 1.4 *On appelle cardinal d'un multi-ensemble M la quantité $|M|$ définie par :*

$$|M| = \sum_{e\in E} M(e)$$

Notation 1.1

1. *Si $(e_k)_{k\in[1,n]}$ est une liste, alors nous la noterons $e_1, \ldots, e_n$.*

2. *Si* $L_1 = e_1, \ldots, e_n$, *et* $L_2 = d_1, \ldots, d_p$, *nous noterons* L_1, L_2 *la liste* $e_1, \ldots, e_n, d_1, \ldots, d_p$.
3. *Étant donné une liste d'indices* $S = (i_k)_{i \in [1,n]}$, *nous noterons* $\overline{e_S}$ *la liste* $(e_{i_k})_{k \in [1,n]}$.
4. $[m, n]$ *représentera, si il n'y a pas ambiguïté, la liste des entiers de* m *à* n, $(k)_{k \in [m,n]}$ *(et la liste vide si* $n < m$*).*
5. *Par abus de notation, lorsqu'il n'y aura pas d'ambiguïté, nous noterons* $e_1, \ldots, e_n$ *(resp.* $\overline{e_S}$*) tout multi-ensemble associé à cette liste.*
6. *Nous noterons* $\cdot$ *ou* $\emptyset$ *le multi-ensemble vide, i.e.* $\cdot(e) = 0$ *pour tout* e.
7. *Nous noterons* M_1, M_2 *tout multi-ensemble* N *tel que* $N(e) = M_1(e) + M_2(e)$.
8. *Nous noterons* $M_1 \cap M_2$ *tout multi-ensemble* N *tel que* $N(e) = \min(M_1(e), M_2(e))$.
9. *Si* $M_2 \subseteq M_1$ *alors nous noterons* $M_1 \backslash M_2$ *tout multi-ensemble* N *tel que* $N(e) = M_1(e) - M_2(e)$.

Définition 1.5 *On dit d'une famille de multi-ensembles* $(\Gamma_k)_{k \in [1,n]}$ *qu'elle est une partition d'un multi-ensemble* Γ *si* $\Gamma = \Gamma_1, \ldots, \Gamma_n$.

Nous pouvons maintenant définir la logique linéaire intuitionniste implicative et exponentielle. Nous allons commencer par définir les formules que cette logique manipule.

Définition 1.6 *Étant donné un ensemble fini* $\mathcal{A}$, *appelé ensemble de formules ou de types atomiques, l'ensemble des formules ou types de la logique linéaire intuitionniste implicative et exponentielle construites à partir de* $\mathcal{A}$ *est donné par la grammaire suivante :*

$$\mathscr{L}_{\mathcal{A}} ::= \mathcal{A} \mid (\mathscr{L}_{\mathcal{A}} \multimap \mathscr{L}_{\mathcal{A}}) \mid !\mathscr{L}_{\mathcal{A}}$$

Notation 1.2

1. *Nous noterons les formules de la logique linéaire par* α, α_i, ..., α',..., β, ..., γ, ...
2. *Nous noterons* $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap \beta$ *la formule* $(\alpha_1 \multimap (\ldots \multimap (\alpha_n \multimap \beta) \ldots))$
3. *Étant donnée une liste d'indices,* $S = (i_k)_{k \in [1,n]}$, *nous noterons* $\overline{\alpha_S} \multimap \beta$ *la formule* $\alpha_{i_1} \multimap \ldots \multimap \alpha_{i_n} \multimap \beta$, *si pour tout* $k, k' \in [1, n]$, $\alpha_{i_k} = \alpha_{i_{k'}} = \alpha$ *alors nous noterons ce type* $\alpha^n \multimap \beta$.

Définition 1.7 *Un contexte* Γ *de la logique linéaire est un multi-ensemble de formules de* $\mathscr{L}_{\mathcal{A}}$.

Notation 1.3

1. *Nous noterons les contextes par* Γ, Γ_i, Γ',..., Δ, Δ_i, Δ', ...
2. *Si* Γ *représente le contexte* $\alpha_1, \ldots, \alpha_n$, *nous noterons* $!\Gamma$ *le contexte* $!\alpha_1, \ldots, !\alpha_n$.

Définition 1.8 *Un séquent est une paire consituée d'un contexte* Γ *et d'une formule* α. *Nous noterons* $\Gamma \vdash \alpha$ *un séquent constitué d'un contexte* Γ *et d'une formule* α.

Définition 1.9 *Une démonstration de la logique linéaire intuitionniste implicative et exponentielle est un arbre de dérivation obtenu à partir des règles de la figure 1.1. On dit qu'un séquent est dérivable ou démontrable si il existe une dérivation dont il est la racine.*

Définition 1.10 *On dit qu'une formule* α *est une tautologie si le séquent* $\cdot \vdash \alpha$ *est dérivable.*

Les formules de la logique linéaire intuitionniste implicative et exponentielle peuvent s'interpréter en terme de ressources : les formules atomiques représentent des ressources élémentaires ; les formules de la forme $\alpha \multimap \beta$ représentent une ressource permettant de transformer une ressource de type α en ressource de type β ; et les formules de la forme $!\alpha$ représentent un nombre arbitraire de ressources de type α. Un séquent $\alpha_1, \ldots, \alpha_n \vdash \alpha$ représente le fait d'utiliser toutes les ressources de type α_1, ..., α_n que nous apporte le contexte afin de produire une ressource de type α. Démontrer un séquent dans la logique linéaire intuitionniste implicative et exponentielle revient à montrer que cela est possible.

La logique linéaire intuitionniste implicative et exponentielle possède une propriété importante pour une logique *l'élimination des coupures.*

Règles d'identité

$$\frac{}{\alpha \vdash \alpha}\ \text{Axiome} \qquad \frac{\Gamma_1 \vdash \alpha \quad \alpha, \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash \beta}\ \text{Coupure}$$

Règles logiques

Règles multiplicatives

$$\frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \multimap \beta} \multimap_D \qquad \frac{\Gamma_1 \vdash \alpha \quad \Gamma_2, \beta \vdash \gamma}{\Gamma_1, \Gamma_2, \alpha \multimap \beta \vdash \gamma} \multimap_G$$

Règles exponentielles

$$\frac{!\Gamma \vdash \alpha}{!\Gamma \vdash !\alpha}\ \text{Promotion} \qquad \frac{\Gamma, !\alpha, !\alpha \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ \text{Contraction}$$

$$\frac{\Gamma \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ \text{Affaiblissement} \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma, !\alpha \vdash \beta}\ \text{Déréliction}$$

FIG. 1.1 – Logique linéaire intuitionniste implicative et exponentielle

Théorème 1.1 ***Élimination des coupures*** *Si $\Gamma \vdash \alpha$ est un séquent dérivable dans la logique linéaire intuitionniste implicative et exponentielle alors il existe une démonstration de $\Gamma \vdash \alpha$ qui n'utilise pas la règle de coupure [Gir87].*

La propriété d'élimination des coupures est désirable pour les logiques parce qu'elle implique que si un séquent est dérivable, alors les séquents qui interviennent dans sa démonstration n'utilisent que des sous-formules du séquent initial. Cela permet entre autre d'automatiser la recherche de démonstration d'un séquent. De plus cette propriété permet de raisonner à propos de la logique et de démontrer ses propriétés théoriques.

Enfin, la propriété de la sous-formule nous permet d'isoler des fragments de la logique linéaire intuitionniste implicative et exponentielle en imposant des contraintes syntaxiques sur les formules qui interviennent dans les séquents.

Définition 1.11 *Étant donné un ensemble fini de formules atomiques $\mathcal{A}$, la logique définie par la grammaire de formules suivante :*

$$\mathcal{I}_\mathcal{A} ::= \mathcal{A} \mid \mathcal{I}_\mathcal{A} \multimap \mathcal{I}_\mathcal{A}$$

est la logique linéaire intuitionniste implicative, notée ***IILL*** *(Intuitionnistic Implicative Linear Logic).*

Définition 1.12 *Étant donnée une formule α de $\mathcal{I}_\mathcal{A}$, la complexité de α est donnée par la fonction ρ définie par :*

1. $\rho(\alpha) = 0$ *si* $\alpha \in \mathcal{A}$
2. $\rho(\alpha \multimap \beta) = \rho(\alpha) + \rho(\beta) + 1$

Si $S = \alpha_1, \ldots, \alpha_n \vdash \alpha$ est un séquent construit à partir des formules atomique de $\mathcal{A}$, on note :

$$\rho(S) = \rho(\alpha) + \sum_{i=1}^{n} \rho(\alpha_i)$$

Définition 1.13 *Étant donnée une formule α de $\mathcal{I}_\mathcal{A}$, l'ordre de α est donnée par la fonction ord définie par :*

1. $ord(\alpha) = 1$ *si* $\alpha \in \mathcal{A}$
2. $ord(\alpha \multimap \beta) = \max(ord(\alpha) + 1, ord(\beta))$

Définition 1.14 *La logique qui ne manipule que des séquents de la forme* $!\Gamma, \Delta \vdash \alpha$ *où les formules de* Γ*, celles de* Δ *et* α *sont des formules de* $\mathcal{I}_{\mathcal{A}}$ *est la logique linéaire intuitionniste implicative et exponentielle simple notée* ***sIIELL*** *(simple Intuitionnistic Implicative and exponential Linear Logic).*

Théorème 1.2 *Le problème de savoir si un séquent est dérivable dans* ***IILL*** *est NP-complet [Kan92].*

La décidabilité de la recherche de démonstration de **IIELL** est encore inconnue, elle est équivalente à la décidabilité de la recherche de démonstration de **MELL**, un fragment plus large de la logique linéaire. Cependant nous avons montré, avec Philippe de Groote et Bruno Guillaume, qu'elle pouvait être reliée à celle de **sIIELL**.

Théorème 1.3 ***IIELL*** *est décidable si et seulement si la dérivabilité dans* ***sIIELL*** *des séquents de la forme* $!\Gamma \vdash \beta$ *avec* β *atomique est décidable [PdG04b].*

1.2 λ-calcul et typage

Le λ-calcul est un formalisme de représentation des fonctions. Il a été inventé par Alonso Church dans le but d'avoir un langage simple de description et de manipulation de fonctions [Chu41]. Étant une théorie sur les fonctions, le λ-calcul a des implications fortes en théorie de la calculabilité. Il s'avère que toutes les fonctions calculables sont représentables à l'aide du λ-calcul. Par ailleurs, ces liens avec diverses logiques en ont fait un formalisme largement étudié.

Après avoir présenté le λ-calcul, nous allons voir comment il est relié à la logique linéaire intuitionniste et implicative. Pour cela nous introduirons le λ-calcul linéaire simplement typé et nous verrons comment les λ-termes permettent de représenter des démonstrations de la logique linéaire intuitionniste implicative à travers la correspondance de Curry-Howard.

1.2.1 Le λ-calcul

Définition 1.15 *Étant donnés un ensemble infini dénombrable de variables* $\mathcal{V}$*, un ensemble infini dénombrable d'inconnues* $\mathcal{X}$ *et un ensemble fini de constantes* $\mathcal{C}$*, les termes du λ-calcul, les* λ-termes*, sont donnés par la grammaires suivantes :*

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{X} \mid \mathcal{C} \mid \lambda\mathcal{V}.\mathcal{T} \mid (\mathcal{T}\mathcal{T})$$

Un λ-terme est dit atomique si il est égal à une variable, une inconnue ou une constante.

Notation 1.4

1. *Nous noterons les variables* x, x_i, x', ..., y, ..., z, ..., f, ..., *les inconnues* $\mathbf{X}$, $\mathbf{Y}$ ..., *les constantes* a, a_i, a', ..., b, ..., c, ..., *et les termes seront notés* t, t_i, t', ..., u, ..., v, ..., w,...
2. *Nous noterons* $tu_1 \dots u_n$ *le terme* $(\dots(tu_1)\dots u_n)$ *et étant donnée une liste d'indices* $S = i_1, \dots, i_n$ *nous noterons* $t\overline{u_S}$ *le terme* $tu_{i_1} \dots u_{i_n}$*. Si* $n = 0$ *ou si la liste* S *est vide, les notations* $tu_1 \dots u_n$ *et* $t\overline{u_S}$ *représenteront le terme* t*.*
3. *Nous noterons* $\lambda x_1 \dots x_n.t$ *le terme* $\lambda x_1 \dots \lambda x_n.t$ *et étant donnée un liste d'indices* $S = i_1, \dots, i_n$ *nous noterons* $\lambda\overline{x_S}.t$ *le terme* $\lambda x_{i_1} \dots x_{i_n}.t$*. Si* $n = 0$ *ou si la liste* S *est vide, les notations* $\lambda x_1 \dots x_n.t$ *et* $\lambda\overline{x_S}.t$ *représenteront le terme* t*.*

Définition 1.16 *Un contexte est un λ-terme contenant un trou. Les contextes sont définis suivant la grammaire suivante :*

$$C[] ::= [] \mid \lambda x.C[] \mid C[]\mathcal{T} \mid \mathcal{T}C[]$$

Étant donné un contexte $C[]$ *et un terme* t*, on note* $C[t]$ *le terme obtenu en insérant* t *dans le trou de* $C[]$*.*

Nous noterons les contextes $C[]$, $C_i[]$, $C'[]$, ...

Définition 1.17 *On dit qu'un terme u est sous-terme d'un terme t si $t = C[u]$. On dit que deux termes sont indépendants si ils ne sont pas sous-terme l'un de l'autre.*

Définition 1.18 *La longueur $|t|$ d'un terme t est définie inductivement par :*

1. $|x| = 1$
2. $|c| = 1$
3. $|\lambda x.u| = |u| + 1$
4. $|u_1 u_2| = |u_1| + |u_2|$

Définition 1.19 *Étant donné un λ-terme atomique h, on note $|t|_h$ le nombre d'occurrences de h dans t. On définit $|t|_h$ inductivement par :*

1. $|h|_h = 1$
2. $|h'|_h = 0$ *si* $h' \neq h$
3. $|\lambda x.u|_h = \begin{cases} |u|_h = & \text{si } h \neq x \\ 0 & \text{sinon} \end{cases}$
4. $|u_1 u_2|_h = |u_1|_h + |u_2|_h$

Définition 1.20 *Un terme t est dit pur si il ne contient pas d'inconnues, i.e. si pour tout $\mathbf{X} \in \mathcal{X}$, $|t|_{\mathbf{X}} = 0$. Si un terme contient des inconnues, alors ce terme est dit impur.*

Lorsque nous ne préciserons pas si un terme est pur ou impur, c'est qu'il est pur. Les inconnues nous serviront à représenter des termes indéfinis. Si nous sommes amenés à les distinguer des variables, c'est que dans le cadre du λ-calcul linéaire, les λ-variables sont soumises à des restrictions syntaxiques qui ne sont pas forcément souhaitables pour les cas où on voudrait qu'elles représentent des termes indéterminés.

Définition 1.21 *On dit d'un terme t qu'il est lexicalisé si il contient des constantes, i.e. il existe $c \in \mathcal{C}$, $|t|_c > 0$.Si un terme n'est pas lexicalisé, nous dirons qu'il est non-lexicalisé.*

Définition 1.22 *Le multi-ensemble $At(t)$ des atomes d'un λ-terme t est le multi-ensemble tel que pour tout terme atomique h, $At(t)(h) = |t|_h$.*

Définition 1.23 *Étant donné un λ-terme t, on note $FV(t)$ l'ensemble des variables libres dans t. Cet ensemble est défini inductivement sur la structure de t :*

1. *si* $t = c$ *et* $c \in \mathcal{C}$ *alors* $FV(t) = \emptyset$
2. *si* $t = x$ *et* $x \in \mathcal{V}$ $FV(t) = \{x\}$
3. *si* $t = \lambda x.u$ *alors* $FV(t) = FV(u) \backslash \{x\}$
4. *si* $t = (u_1 u_2)$ *alors* $FV(t) = FV(u_1) \cup FV(u_2)$

Définition 1.24 *L'ensemble de λ-termes linéaires est inductivement défini comme il suit :*

1. *si h est un terme atomique alors h est un λ-terme linéaire*
2. *si t est un λ-terme linéaire et $x \in FV(t)$ alors $\lambda x.t$ est un λ-terme linéaire*
3. *si t_1 et t_2 sont des λ-termes linéaires et $FV(t_1) \cap FV(t_2) = \emptyset$ alors $t_1 t_2$ est un λ-terme linéaire.*

Les λ-termes linéaires sont linéaires dans le sens que si $x \in FV(t)$ alors $|t|_x = 1$.

Définition 1.25 *On dit qu'un terme t est clos si $FV(t) = \emptyset$.*

Nous allons considérer que les termes sont équivalents modulo α-conversion (*i.e.* modulo renommage des variables liées). Cependant l'α-conversion est une relation d'équivalence difficile à mettre en œuvre. Entrer dans les détails des substitutions explicites [dB72] nous éloignerait de beaucoup du sujet de ce mémoire. On ne peut pourtant pas écarter les difficultés posées par l'α-conversion, surtout lorsqu'il est question de décrire des algorithmes précis. Aussi nous adopterons la convention usuelle que les variables liées d'un terme sont deux à deux distinctes, propriété que l'α-conversion permet toujours d'obtenir. Plus formellement, cette propriété s'exprime de la façon suivante :

Définition 1.26 *Étant donné un λ-terme t, on note $BV(t)$, l'ensemble des variables liées de t. Cet ensemble est défini inductivement sur la structure de t :*

1. *si $t = h$ et h est un terme atomique alors $BV(t) = \emptyset$*
2. *si $t = \lambda x.u$ alors $BV(t) = BV(u) \cup \{x\}$*
3. *si $t = (u_1 u_2)$ alors $BV(t) = BV(u_1) \cup FV(u_2)$*

Définition 1.27 *On dit d'un terme t que ces variables liées sont deux à deux distinctes si :*

1. *t est un terme atomique*
2. *$t = t_1 t_2$, si les variables liées de t_1 sont deux à deux distinctes, si il en est de même pour les variables liées de t_2 et si $BV(t_1) \cap BV(t_2) = \emptyset$*
3. *$t = \lambda x.t'$, si les variables liées de t' sont deux à deux distinctes et si $x \notin BV(t')$.*

Nous pouvons maintenant définir une notion centrale pour le λ-calcul, la substitution.

Définition 1.28 *Une substitution est une fonction de $\mathcal{V} \cup \mathcal{X}$ dans $\mathcal{T}$ telle que l'ensemble des variables x et des inconnues $\mathbf{X}$ pour lesquelles $\sigma(x) \neq x$ et $\sigma(\mathbf{X}) \neq \mathbf{X}$ est fini; cet ensemble est appelé le support de la substitution. Étant donnée une substitution σ, $t.\sigma$, l'application de σ au terme t, est définie par :*

1. *$x.\sigma = \sigma(x)$*
2. *$\mathbf{X}.\sigma = \sigma(\mathbf{X})$*
3. *$c.\sigma = c$*
4. *$(\lambda x.t).\sigma = \lambda x.t.\sigma$ si pour tout y dans le support de σ, $y \neq x$, $x \notin FV(\sigma(y))$ et si pour tout $\mathbf{X}$ dans le support de σ, $x \notin FV(\sigma(\mathbf{X}))$ (grâce à l'α-conversion, il est toujours possible de renommer x pour remplir cette condition).*
5. *$(t_1 t_2).\sigma = t_1.\sigma t_2.\sigma$*

Notation 1.5

1. *nous noterons les substitutions par σ, σ_i, ..., σ' ...*
2. *étant donnée une substitution σ, nous noterons $\mathscr{D}(\sigma)$ le support de σ.*
3. *si $\mathscr{D}(\sigma) = \{x_1; \ldots; x_n; \mathbf{X}_1; \ldots; \mathbf{X}_p\}$, nous noterons σ par $[x_1 := \sigma(x_1); \ldots; x_n := \sigma(x_n); \mathbf{X}_1 := \sigma(\mathbf{X}_1); \ldots; \mathbf{X}_n := \sigma(\mathbf{X}_n)]$.*
4. *si $\sigma = [x_1 := t_1; \ldots x_n := t_n]$, nous noterons σ par $[x_k := t_k]_{k=1}^n$ ou si, étant donné un ensemble d'indices $S = \{i_1; \ldots; i_n\}$, $\sigma = [x_{i_1} := t_{i_1}; \ldots x_{i_n} := t_{i_n}]$, nous noterons σ par $[x_k := t_k]_{k \in S}$.*
5. *si $\sigma = [\mathbf{X}_1 := t_1; \ldots \mathbf{X}_n := t_n]$, nous noterons σ par $[\mathbf{X}_k := t_k]_{k=1}^n$ ou, si étant donné un ensemble d'indices $S = \{i_1; \ldots; i_n\}$, $\sigma = [\mathbf{X}_{i_1} := t_{i_1}; \ldots \mathbf{X}_{i_n} := t_{i_n}]$, nous noterons σ par $[\mathbf{X}_k := t_k]_{k \in S}$.*

Définition 1.29 *On appelle renommage, une substitution de la forme $[x_k := x'_k]_{k \in [1,n]}$ et telle que $x'_i = x'_j$ implique $i = j$.*

Définition 1.30 *On dit qu'un terme t_1 se β-contracte en t_2, ce que l'on note $t_1 \to_\beta t_2$, si :*

$$t_1 = C[(\lambda x.t)u] \text{ et } t_2 = C[t[x := u]]$$

Les termes de la forme $(\lambda x.t)u$ sont appelés des β-redex (ou simplement redex lorsqu'il n'y a pas d'ambiguïté), et $t[x := u]$ est appelé le contracté de $(\lambda x.t)u$.

Définition 1.31 *On dit que t_1 se β-réduit en n pas à t_2, ce que l'on note $t_1 \xrightarrow{n}_\beta t_2$, si il existe une famille de $n+1$ termes $(u_k)_{k \in [0,n]}$ tels que $u_i \to_\beta u_{i+1}$ pour $i \in [0, n-1]$, $t_1 = u_0$ et $t_2 = u_n$.*

Définition 1.32 *La relation de β-réduction, $\xrightarrow{*}_\beta$ est la clôture réflexive et transitive de la relation de β-contraction.*

Définition 1.33 *La relation de β-conversion, $=_\beta$ est la clôture réflexive, symétrique et transitive de la relation de β-contraction.*

Définition 1.34 *On dit qu'un terme t_1 se η-contracte en t_2, ce que l'on note $t_1 \rightarrow_\eta t_2$, si :*

$$t_1 = C[\lambda x.(tx)],\ t_2 = C[t] \text{ et } x \notin FV(t)$$

Les termes de la forme $\lambda x.(tx)$ sont appelés des η-redex (ou simplement redex lorsqu'il n'y a pas d'ambiguïté).

Définition 1.35 *On dit que t_1 se η-réduit en n pas à t_2, ce que l'on note $t_1 \overset{n}{\rightarrow}_\eta t_2$, si il existe une famille de $n+1$ termes $(u_k)_{k\in[0,n]}$ tels que $u_i \rightarrow_\eta u_{i+1}$ pour $i \in [1, n-1]$, $t_1 = u_0$ et $t_2 = u_n$.*

Définition 1.36 *La relation de η-réduction, $\overset{*}{\rightarrow}_\eta$ est la clôture réflexive et transitive de la relation de η-contraction.*

Définition 1.37 *La relation de η-conversion, $=_\eta$ est la clôture réflexive, symétrique et transitive de la relation de η-contraction.*

Définition 1.38 *On note $\rightarrow_{\beta\eta}$ l'union des relations de β-contraction, et de η-contraction. On note respectivement $\overset{n}{\rightarrow}_{\beta\eta}$, $\overset{*}{\rightarrow}_{\beta\eta}$ et $=_{\beta\eta}$, la relation de n pas de contraction, la clôture réflexive et transitive, la clôture réflexive, symétrique et transitive de $\rightarrow_{\beta\eta}$*

Définition 1.39 *On dit qu'un terme t est sous forme normale pour une relation $\rightarrow_\mathcal{R}$ si il n'existe pas de terme u tel que $t \rightarrow_\mathcal{R} u$. Le terme t est alors sous forme $\mathcal{R}$-normale.*

Définition 1.40 *On dit qu'un terme t est fortement normalisable pour la relation $\rightarrow_\mathcal{R}$ si il n'existe pas de suite $(u_k)_{k\in\mathbb{N}}$ telle que $u_i \rightarrow_\mathcal{R} u_{i+1}$ pour tout $i \in \mathbb{N}$ et $t = u_0$.*

Les relations $\rightarrow_\beta$ et $\rightarrow_{\beta\eta}$ lorsqu'elles sont choisies comme sémantique opérationnelle du λ-calcul, font du λ-calcul une théorie des fonctions en ce sens que l'on peut considérer que les λ-termes définissent des résultats uniques. La notion d'unicité du résultat n'est pas triviale puisqu'elle tient compte du fait qu'il peut y avoir des réductions infinies. Elle est exprimée par la propriété de Church-Rosser.

Théorème 1.4 ***Propriété de Church-Rosser*** *Quelque soit la relation $\rightarrow_\mathcal{R} \in \{\rightarrow_\beta; \rightarrow_{\beta\eta}\}$, si $t_1 \overset{*}{\rightarrow}_\mathcal{R} t_2$ et $t_1 \overset{*}{\rightarrow}_\mathcal{R} t_3$ alors il existe t_4 tel que $t_2 \overset{*}{\rightarrow}_\mathcal{R} t_4$ et $t_3 \overset{*}{\rightarrow}_\mathcal{R} t_4$*

1.2.2 La correspondance de Curry-Howard et le λ-calcul linéaire

Un des intérêts fondamentaux du λ-calcul est sa capacité à représenter les démonstrations des logiques intuitionnistes. Sa sémantique opérationnelle permet de rendre compte des transformations que l'on peut faire subir aux démonstrations, notamment lorsque l'on élimine la règle de coupure. Dans ce qui va suivre, nous supposons donnés un ensemble infini et dénombrable de λ-variables $\mathcal{V}$ ainsi qu'un ensemble infini dénombrable d'inconnues $\mathcal{X}$.

Définition 1.41 *Une signature linéaire d'ordre supérieur Σ est un triplet $(\mathcal{C}, \mathcal{A}, \tau)$ où $\mathcal{C}$ est un ensemble fini de constantes, $\mathcal{A}$ est un ensemble fini de types atomiques et τ est une fonction de $\mathcal{C}$ dans $\mathcal{I}_\mathcal{A}$.*

Définition 1.42 *On dit qu'une signature Σ est d'ordre n si pour tout $c \in \mathcal{C}_\Sigma$ $ord(\tau_\Sigma(c)) \leq n$.*

Notation 1.6

1. *Nous noterons les signatures par Σ, Σ_i, ..., Σ', ...*
2. *Étant donnée une signature $\Sigma = (\mathcal{C}, \mathcal{A}, \tau)$, nous noterons $\mathcal{C}$ par $\mathcal{C}_\Sigma$, $\mathcal{A}$ par $\mathcal{A}_\Sigma$, τ par τ_Σ et $\mathcal{I}_\Sigma$ l'ensemble des formules de $\mathcal{I}_\mathcal{A}$.*

Définition 1.43 *Étant donnée une signature Σ, un contexte Γ d'assignation de types à des variables sur Σ est un ensemble fini $x_1 : \alpha_1, \ldots, x_n : \alpha_n$ où les x_k sont des λ-variables deux à deux distinctes et les α_k sont des formules de $\mathcal{I}_\Sigma$.*

Définition 1.44 *Étant donnée une signature Σ, un contexte Γ d'assignation de types à des inconnues sur Σ est un ensemble fini $\mathbf{X}_1 : \alpha_1, \ldots, \mathbf{X}_n : \alpha_n$ où les $\mathbf{X}_k$ sont des inconnues deux à deux distinctes et les α_k sont des formules de $\mathcal{I}_\Sigma$.*

Notation 1.7

1. *Étant donnés un ensemble fini d'indices $S = i_1, \ldots, i_n$, un ensemble de variables $(x_k)_{k\in S}$, et un ensemble de type $(\alpha_k)_{k\in S}$, nous noterons $\overline{x_S : \alpha_S}$ le contexte $x_{i_1} : \alpha_{i_1}, \ldots, x_{i_n} : \alpha_{i_n}$.*
2. *Étant donnés un ensemble fini d'indices $S = i_1, \ldots, i_n$, un ensemble d'inconnues $(\mathbf{X}_k)_{k\in S}$, et un ensemble de type $(\alpha_k)_{k\in S}$, nous noterons $\overline{\mathbf{X}_S : \alpha_S}$ le contexte $\mathbf{X}_{i_1} : \alpha_{i_1}, \ldots, \mathbf{X}_{i_n} : \alpha_{i_n}$.*
3. *Étant donné un contexte d'assignation de type à des variables $\Gamma = \overline{x_{[1,n]} : \alpha_{[1,n]}}$, on note $\overline{\Gamma}$ l'ensemble $\overline{x_{[1,n]}}$ et on note $\widetilde{\Gamma}$ l'ensemble $\overline{\alpha_{[1,n]}}$.*
4. *Étant donné un contexte d'assignation de type à des inconnues $\Gamma = \overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}$ on note $\overline{\Gamma}$ l'ensemble $\overline{\mathbf{X}_{[1,n]}}$ et on note $\widetilde{\Gamma}$ l'ensemble $\overline{\alpha_{[1,n]}}$.*
5. *Étant donnés deux contextes, lorsque nous écrirons Γ_1, Γ_2, nous supposerons que $\overline{\Gamma_1} \cap \overline{\Gamma_2} = \emptyset$.*

Définition 1.45 *On dit qu'un λ-terme t est typable dans une signature Σ si il existe un contexte d'assignation d'inconnues Γ, un contexte d'assignation de variables Γ' et $\alpha \in \mathcal{I}_\Sigma$ tels que $\Gamma'; \Gamma \vdash_\Sigma t : \alpha$ est dérivable à l'aide des règles de la figure 1.2. Nous noterons $\mathcal{T}_\Sigma$ l'ensemble des termes typables sur Σ.*

Notation 1.8 *Lorsque nous typerons des termes purs, dans une signature Σ qui sera claire d'après le contexte, nous noterons $\Gamma \vdash t : \alpha$ les séquents de typage.*

Le lemme suivant montre que les termes typables sont des termes linéaires.

Lemme 1.1 *Si $\Gamma'; \Gamma \vdash_\Sigma t : \alpha$ est dérivable alors t est un λ-terme linéaire.*

Le système de typage des λ-termes défini par la figure 1.2 permet de voir que les λ-termes codent bel et bien les démonstrations de la logique linéaire implicative.

Théorème 1.5 ***Correspondance de Curry-Howard***
*Il existe un terme linéaire t pur et non lexicalisé tel que $\Gamma \vdash_\Sigma t : \alpha$ est dérivable si et seulement si $\widetilde{\Gamma} \vdash \alpha$ est dérivable dans **IILL**.*
*Il existe un terme linéaire t pur tel que $\Gamma \vdash_\Sigma t : \alpha$ est dérivable si et seulement si $!\Gamma_\Sigma, \widetilde{\Gamma} \vdash \alpha$ est dérivable dans **sIIELL** avec $\Gamma_\Sigma = \overline{\tau_\Sigma(c_{[1,n]})}$ si $\mathcal{C}_\Sigma = \overline{c_{[1,n]}}$.*

Ce théorème et le théorème 1.3 permettent de conclure qu'il est aussi difficile de chercher un terme ayant le type α dans un contexte Γ et une signature Σ que de rechercher des démonstrations dans **MELL**.

Lemme 1.2 *Si $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable alors $FV(t) = \overline{\Delta}$.*

Ainsi, si t est un terme clos et pur, on pourra parler d'un terme t de type α dans la signature Σ. Si Σ est claire d'après le contexte, on parlera seulement d'un terme clos de type α.

Une des conséquences de la correspondance de Curry-Howard est que, tout comme pour **sIIELL**, on peut dériver tout séquent sans utiliser la règle de coupure.

Théorème 1.6 ***Elimination des coupures*** *Si $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable alors $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable sans utiliser la règle de coupure.*

De plus l'ensemble des termes de type α dans un contexte Γ et sur une signature Σ est clos pour les notions de réduction.

Règles d'identité

$$\frac{\alpha \in \mathcal{I}_\Sigma}{\Gamma; x : \alpha \vdash_\Sigma x : \alpha}\ \text{Axiome} \qquad \frac{\Gamma; \Gamma_1 \vdash_\Sigma u : \alpha \quad \Gamma; x : \alpha, \Gamma_2 \vdash_\Sigma t : \beta}{\Gamma; \Gamma_1, \Gamma_2 \vdash_\Sigma t[x := u] : \beta}\ \text{Coupure}$$

$$\frac{\tau_\Sigma(c) = \alpha}{\Gamma; \cdot \vdash_\Sigma c : \alpha}\ \text{Const} \qquad \frac{\alpha \in \mathcal{I}_\Sigma}{\Gamma, \mathbf{X} : \alpha; \cdot \vdash_\Sigma \mathbf{X} : \alpha}\ \text{Inc}$$

Règles logiques

$$\frac{\Gamma; \Gamma', x : \alpha \vdash_\Sigma t : \beta}{\Gamma; \Gamma' \vdash_\Sigma \lambda x.t : \alpha \multimap \beta}\ \multimap_I \qquad \frac{\Gamma; \Gamma_1 \vdash_\Sigma t : \alpha \multimap \beta \quad \Gamma; \Gamma_2 \vdash_\Sigma u : \alpha}{\Gamma; \Gamma_1, \Gamma_2 \vdash_\Sigma tu : \beta}\ \multimap_E$$

FIG. 1.2 – Typage des λ-termes linéaires

Théorème 1.7 ***Réduction du sujet*** *Étant donnés deux λ-termes t_1 et t_2 et un contexte Γ tels que $\Gamma; \Gamma' \vdash_\Sigma t_1 : \alpha$ est dérivable et $t_1 \to_{\beta\eta} t_2$, alors $\Gamma; \Gamma' \vdash_\Sigma t_2 : \alpha$ est dérivable.*

Enfin, une propriété fondamentale des calculs associés à des logiques cohérentes est que tous les termes typables sont fortement normalisables.

Théorème 1.8 *Les λ-termes linéaires sont fortement normalisables pour la relation $\to_{\beta\eta}$.*

Enfin, un terme t possède un unique type dans un contexte donné.

Théorème 1.9 ***Unicité du type*** *Si $\Gamma; \Delta \vdash_\Sigma t : \alpha$ et $\Gamma; \Delta \vdash_\Sigma t : \beta$ sont dérivables, alors $\alpha = \beta$.*

Théorème 1.10 ***Unicité de typage des sous-termes*** *Soit t un terme tel que $\Gamma; \Delta \vdash_\Sigma t : \alpha$ soit dérivable (les variables liées de t sont deux à deux distinctes), si $t = C[u]$, alors il existe un unique contexte Δ' et un unique type β tels que $\Gamma; \Delta' \vdash_\Sigma u : \beta$ soit un séquent qui intervienne dans la dérivation de $\Gamma; \Delta \vdash_\Sigma t : \alpha$. Ainsi si u est un sous-terme de t et que $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable, nous pourrons parler du type de u ainsi que de son contexte de typage relativement à t.*

Une dernière notion importante et dont nous nous servirons beaucoup est la notion de forme η-longue d'un terme.

Définition 1.46 *Soit t un terme tel que $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable, on dit que t est sous forme η-longue si pour tout contexte $C[]$ et tout terme u tel que $t = C[u]$, si u est un sous-terme de t de type $\beta \multimap \gamma$, alors $u = \lambda x.u'$ ou $C[] = C'[[]v]$.*

Théorème 1.11 *Si t est un terme tel que $\Gamma; \Delta \vdash_\Sigma t : \alpha$ est dérivable et que t est sous forme η-longue, alors si $t \to_\beta t'$, alors t' est sous forme η-longue [Hue76].*

Notation 1.9 *Étant donné un terme t, nous noterons $t\uparrow^\eta$ le terme t' tel que $t =_\eta t'$ et t' est sous forme η-longue.*

L'intérêt de formes η-longues est qu'elles permettent d'utiliser seulement la relation de β-conversion pour comparer des termes modulo $\beta\eta$. Pour cela, il suffit de comparer modulo β des termes sous forme η-longue.

Les λ-termes linéaires nous permettent de représenter des structures de données simples comme les arbres ou les chaînes de caractères. Comme les grammaires catégorielles abstraites ont pour vocation d'analyser les textes, elles utilisent le λ-calcul pour modéliser les chaînes de caractères. Nous allons voir rapidement comment cela se fait.

Définition 1.47 *Étant donné un ensemble fini de caractères L, on définit la signature $\Sigma_L = (L, \{*\}, \tau_L)$ des chaînes sur L avec pour tout $c \in L$, $\tau_L(c) = * \multimap *$. Une chaîne sur Σ_L est un terme clos de type $* \multimap *$. L'opérateur de concaténation de chaîne est le terme de type $(* \multimap *) \multimap (* \multimap *) + = \lambda xyz.x(y(z))$. La chaîne vide est le terme de type $* \multimap * \lambda x.x$.*

Les lemmes suivants montrent que Σ_L modélise bien le monoïde libre construit sur L puisque la concaténation est associative et que la chaîne vide est l'élément neutre de la concaténation.

Lemme 1.3 *Étant donné un ensemble fini de caractères L, si t_1, t_2 et t_3 sont des chaînes sur Σ_L alors on a $+t_1(+t_2t_3) =_\beta +(+t_1t_2)t_3$.*

Lemme 1.4 *Étant donné un ensemble fini de caractères L, si t est une chaîne sur Σ_L alors $+t(\lambda x.x) =_\beta +(\lambda x.x)t =_{\beta\eta} t$.*

Notation 1.10

1. *Nous noterons le terme $+t_1(+\ldots(+t_{n-1}t_n)\ldots)$ par $t_1 + \ldots + t_n$.*
2. *Nous noterons ϵ la chaîne vide sur toute signature de chaîne.*
3. *Nous noterons a^n la chaîne $\underbrace{a + \ldots + a}_{n}$.*

1.3 Algorithme de recherche de démonstration dans IILL

Dans ce qui suit, nous supposerons donnée la signature $\Sigma = (\mathcal{C}, \mathcal{A}, \tau)$ et nous n'utiliserons que des termes non-lexicalisés.

Nous allons ici présenter une technique de recherche de démonstration dans **IILL** qui utilise la correspondance de Curry-Howard. En effet, si on cherche à démontrer le séquent $\Gamma \vdash \alpha$, cela est équivalent à chercher un terme pur et non-lexicalisé t tel que $\Delta \vdash t : \alpha$ avec $\widetilde{\Delta} = \Gamma$. On peut restreindre cette recherche au cas où t est un terme clos et sous forme β-normale et η-longue. Ainsi, étant donné une formule α, l'algorithme que nous allons présenter cherche à construire un terme clos, β-normal et η-long de type α. Cet algorithme s'inspire d'idées provenant de la théorie des réseaux de démonstration de la logique linéaire notamment de [dG00b]. Mais alors que dans [dG00b], l'algorithme de recherche de démonstration construit le λ-terme associé à la démonstration de haut en bas, ici, l'algorithme de recherche de démonstration le construit de bas en haut.

Lorsque l'on cherche à démontrer un séquent, $\Gamma \vdash \alpha$ dans **IILL**, on peut savoir à l'avance, et ce dans toute démonstration éventuelle, quelles formules pourront intervenir à droite du séquent et celles qui pourront intervenir à gauche. Cette discrimination entre la gauche et la droite du séquent correspond en logique classique à la distinction entre hypothèses et conclusions et en logique linéaire entre ressources disponibles et ressources à produire. Formellement, pour traduire ce fait on introduit la notion de polarité. Les formules qui se trouvent toujours à gauche du séquent sont considérées comme négatives alors que celles qui se trouvent toujours à droite du séquent sont considérées comme positives.

Définition 1.48 *L'ensemble des formules polarisées est $\mathcal{I}_\mathcal{A} \times \{+; -\}$. On notera α^ϵ la paire (α, ϵ). La formule polarisée α^+ (resp. α^-) est appelée occurrence positive (resp. négative) de α.*

Nous allons superposer aux termes cette notion de polarité ; les termes que l'on cherche à produire seront des inconnues et les termes dont nous disposons seront des termes purs.

Définition 1.49 *Un terme positif est un terme β-normal de la forme $\mathbf{X}\overline{x_{[1,n]}}$. L'inconnue $\mathbf{X}$ est la tête de ce terme.*

Définition 1.50 *Un terme négatif est un terme β-normal de la forme $x\overline{t_{[1,n]}}$. La variable x est la tête de ce terme. On dit qu'un terme négatif n'est pas instancié si il est de la forme $x\overline{\mathbf{X}_{[1,n]}}$ et on dit qu'il est totalement instancié si il est pur.*

Définition 1.51 *Si u est un terme positif et t est un terme négatif, on dit que u domine t (resp. t domine u), ce que l'on note $t \prec u$ (resp. $u \prec t$), si $\mathbf{X}$ (resp. x) est la tête de u (resp. t) et $|t|_{\mathbf{X}} = 1$ (resp. $|u|_x = 1$). On note $\prec^*$ la clôture réflexive et transitive de $\prec$.*

Définition 1.52 *Un triplet $(\Gamma; \Delta; S)$ est une* pré-structure de démonstration *si :*

1. *Γ est un contexte d'assignation de types à des inconnues*
2. *Δ est un contexte d'assignation de types à des variables*
3. *S un ensemble de paires $t : \alpha^\epsilon$*
4. *pour tout $t : \alpha^\epsilon \in S$, il existe Δ_t tel que $\Delta_t \subseteq \Delta$ et $\Gamma; \Delta_t \vdash t : \alpha$ est dérivable.*
5. *si $t : \alpha^+ \in S$ alors t est un terme positif et si $t : \alpha^- \in S$ alors t est un terme négatif.*

Définition 1.53 *Étant donnée une pré-structure de démonstration $(\Gamma; \Delta; S)$, on dit qu'elle est* arborescente *si :*

1. *il existe $t : \alpha^\epsilon \in S$ la racine de S, tel que pour tout $u : \beta^{\epsilon'} \in S$, $t \prec^* u$ et $u \not\prec t$.*
2. *pour tout $u : \beta^{\epsilon'} \in S$ qui n'est pas la racine de S, il existe un unique $v : \gamma^{\epsilon''}$ tel que $v \prec u$.*
3. *si $\mathbf{X} \in \overline{\Gamma}$ n'est pas la tête de la racine alors il existe uniquement deux paires $u : \alpha_1^{\epsilon_1}$ et $v : \alpha_2^{\epsilon_2}$ dans S telles que $|u|_{\mathbf{X}} = |v|_{\mathbf{X}} = 1$ et de plus $\epsilon_1 = +$ si et seulement si $\epsilon_2 = -$.*
4. *si $x \in \overline{\Delta}$ n'est pas la tête de la racine alors il existe uniquement deux paires $u : \alpha_1^{\epsilon_1}$ et $v : \alpha_2^{\epsilon_2}$ dans S telles que $|u|_x = |v|_x = 1$ et de plus $\epsilon_1 = +$ si et seulement si $\epsilon_2 = -$.*

Une pré-structure de démonstration arborescente est appelée une structure de démonstration.

Définition 1.54 *Si $t : \alpha^+$ (resp. $t : \alpha^-$) est la racine d'une structure de démonstration T, on dit que T a une racine positive (resp. négative).*

Définition 1.55 *On dit qu'une structure de démonstration $T = (\Gamma; \Delta; S)$ est un* pré-réseau de démonstration *si pour tout $t : \alpha^- \in S$, t est un terme négatif non-instancié.*

Définition 1.56 *On dit qu'un pré-réseau de démonstration $(\Gamma; \Delta; S)$ est un* un problème de recherche de démonstration *si il vérifie les propriétés suivantes :*

1. *sa racine est positive.*
2. *si $t : \alpha^\epsilon \in S$ alors α est une formule atomique.*

Définition 1.57 *Étant donné un type α, on associe à α le problème de recherche de démonstration $P_\alpha = It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$ où It est définie par :*

1. *$It(\Gamma; \Delta; t : (\alpha_1 \multimap \alpha_2)^+, S) \rightarrow_{It} It(\Gamma; \Delta, x : \alpha_1; x : \alpha_1^-, tx : \alpha_2^+, S)$ avec $x \notin \overline{\Delta}$*
2. *$It(\Gamma; \Delta; t : (\alpha_1 \multimap \alpha_2)^-, S) \rightarrow_{It} It(\Gamma, \mathbf{X} : \alpha_1; \Delta; \mathbf{X} : \alpha_1^+, t\mathbf{X} : \alpha_2^-, S)$ avec $\mathbf{X} \notin \overline{\Gamma}$*
3. *$It(\Gamma; \Delta; S) \rightarrow_{It} (\Gamma; \Delta; S)$ si pour tout $t : \alpha^\epsilon \in S$, α est atomique.*

La fonction It n'est pas déterministe et le résultat dépend du choix que l'on fait des inconnues ou des variables pour construire la structure de démonstration.

Nous allons voir que $(\Gamma; \Delta; S)$ est égal à $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$ si et seulement si ce triplet est un problème de recherche de démonstration.

Lemme 1.5 *Étant donné un pré-réseau de démonstration $T = (\Gamma; \Delta; S)$, l'une des propriétés suivantes est vérifiée :*

1. *$T = (\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$*
2. *$T = (\cdot; x : \alpha; x : \alpha^+)$*
3. *$S = tx : \alpha^+, x : \beta^-, S'$*
4. *$S = t\mathbf{X} : \alpha^-; \mathbf{X} : \beta^+, S'$*

Démonstration. On procède par induction sur le nombre n des paires qui constituent S. Si $n = 1$ alors évidemment, ou bien $T = (\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$ ou alors $T = (\cdot; x : \alpha; x : \alpha^-)$. Si $n = n + 1$, supposons que la racine r de T soit positive (si elle était négative, on procéderait de façon similaire), alors $r = \mathbf{X}x_1 \dots x_n : \alpha^+$, comme T est un pré-réseau de démonstration, il s'ensuit qu'il existe $v : \beta^- \in S$ tel que la tête de v est x_n. Si $v = x_n$, on peut conclure, sinon soient $S'' = \{u : \gamma^\epsilon \in S | v \prec^* u\}$, $\Gamma'' = \{\mathbf{Y} : \gamma \in \Gamma | \exists u : \gamma \in S''.|u|_{\mathbf{Y}} > 0\}$ et $\Delta'' = \{y : \gamma \in \Delta | \exists u : \gamma \in S''.|u|_y > 0\}$. Nous allons voir que $(\Gamma''; \Delta''; S'')$ est un pré-réseau de démonstration. Bien évidemment, les éléments de S'' constituent un sous-arbre de T, nous avons donc simplement à vérifier que pour tout $y \in \overline{\Delta''}$ (*resp.* $\mathbf{Y} \in \overline{\Gamma''}$) si $y \neq x_n$ il existe exactement deux éléments $u : \gamma^+$, $w : \gamma'^-$ de S'' tels que $|u|_y = |w|_y = 1$ (*resp.* $|u|_{\mathbf{Y}} = |w|_{\mathbf{Y}} = 1$). Si $y \in \overline{\Delta''}$ (*resp.* $\mathbf{Y} \in \overline{\Gamma''}$), c'est qu'il existe $u : \alpha^\epsilon$ tel que $|u|_y = 1$ (*resp.* $|u|_{\mathbf{Y}} = 1$), si $\epsilon = +$, alors il existe $w : \gamma^- \in S$ tel que $|w|_y = 1$. Il s'ensuit que $u \prec w$, et comme $v \prec^* u$, on a que $v \prec^* w$ et donc $w : \gamma^- \in S''$. Si $\epsilon = -$, alors il $w : \gamma^+ \in S$ tel que $|w|_y = 1$ et il vient que $w \prec u$. Comme $y \neq x_n$, $u \neq v$ on a $v \prec^* w \prec u$ et $w : \gamma^- \in S''$. □

Lemme 1.6 *Si T est un pré-réseau de démonstration dont la racine est positive alors $It(T)$ est un problème de recherche de démonstration et il existe $\mathbf{X}$ et α tels que $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_{It} T$.*

Démonstration. Nous allons démontrer ce lemme en deux temps. Tout d'abord nous allons voir que $It(\Gamma; \Delta; S)$ est un problème de recherche de démonstration. Pour cela on procède par induction sur

$$\rho(S) = \sum_{t:\alpha^\epsilon \in S)} \rho(\alpha)$$

Si $\rho(S) = 0$ c'est que pour tout $t : \alpha^\epsilon \in S$, α est atomique et donc $(\Gamma; \Delta; S)$ est un problème de recherche de démonstration et comme dans ce cas $It(\Gamma; \Delta; S) \rightarrow_{It} (\Gamma; \Delta; S)$, on peut conclure.

Si $\rho(S) = n + 1$ alors $S = t : \alpha \multimap \beta^\epsilon \in S'$. Si $\epsilon = +$, on obtient que $It(\Gamma; \Delta; t : \alpha \multimap \beta^\epsilon, S') \rightarrow_{It} It(\Gamma; \Delta, x : \alpha; x : \alpha^-, tx : \beta^+, S')$, or il est évident que $(\Gamma; \Delta; x : \alpha^-, tx : \beta^+, S')$ est un pré-réseau de démonstration dont la racine est positive. De plus $\rho(x : \alpha^-, tx : \beta^+, S') = n$, on peut donc utiliser l'hypothèse d'induction et conclure. Si $\epsilon = -$, on conclut de la même façon.

Ensuite, nous démontrons que si $(\Gamma; \Delta; S)$ est un pré-réseau de démonstration, alors il existe α et $\mathbf{X}$ tels que $It(\Gamma; \Delta; S) = It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$. Pour cela, on procède par induction sur le nombre d'éléments de S. Si $|S| = 1$ alors $T = (\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$ et on peut donc conclure. Si $|S| > 1$, alors d'après le lemme précédent, $S = tx : \beta^+, x : \gamma^- S'$ et $\Delta = x : \gamma, \Delta'$ (*resp.* $S = t\mathbf{Y} : \beta^-, \mathbf{Y} : \gamma^+, S'$) dans S et donc $It(\Gamma; \Delta'; S', t : \gamma \multimap \beta^+) = It(\Gamma; \Delta; S)$. Or par hypothèse d'induction, il existe $\mathbf{X}$ et α tels que $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \rightarrow_{It} It(\Gamma; \Delta'; S')$ et donc $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) = It(\Gamma; \Delta; S)$. □

Ainsi, nous avons démontré qu'une structure T de démonstration était un problème de recherche de démonstration si et seulement si $T = It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+)$.

Nous allons maintenant voir comment utiliser les structures de démonstration pour construire des termes clos de type α.

Définition 1.58 *Étant donnés un terme positif $u = \mathbf{X}\overline{x_P}$, et un terme négatif totalement instancié t tel que $\overline{x_P} \subseteq FV(t)$, alors on définit la relation $\rightarrow_P$ sur les pré-structures de démonstration :*

$$(\Gamma, \mathbf{X} : \overline{\alpha_P} \multimap \alpha; \Delta, \overline{x_P : \alpha_P}; u : \alpha^+, t : \alpha^-, S) \rightarrow_P (\Gamma; \Delta; S.[\mathbf{X} := \lambda\overline{x_P}.t])$$

Nous allons voir que l'ensemble des structures de démonstration est invariant par la relation $\rightarrow_P$.

Lemme 1.7 *Si $T = (\Gamma; \Delta; S)$ est une structure de démonstration et si $T \rightarrow_P T'$ avec $T' = (\Gamma'; \Delta'; S')$ et $S' \neq \emptyset$ alors T' est une structure de démonstration.*

Démonstration. Si $T \rightarrow_P T'$ c'est que $T = (\Gamma; \Delta; u : \alpha^+, t : \alpha^-, S'')$ et que t est un terme totalement instancié et $FV(u) \subseteq FV(t)$. Comme T est une structure de démonstration et puisque t est totalement instancié il s'ensuit qu'il n'existe pas $v : \gamma^+ \in S$ tel que $t \prec v$, si c'était le cas, cela signifierait que t contient la tête de v ce qui contredirait le fait que t est totalement instancié. De plus si il existe $v : \gamma^- \in S$

tel que $u \prec v$ alors $v : \gamma^- = t : \alpha^-$. En effet, comme T est une structure de démonstration, pour tout $x \in FV(u)$, il existe un unique $v : \gamma^-$ tel que $x \in FV(v)$. Or pour tout $x \in FV(u)$, $x \in FV(t)$, et donc si il existe $v : \gamma^- \in S$ tel que $u \prec v$, c'est que u contient la tête de v et donc $v : \gamma^- = t : \alpha^-$.

Si $FV(u) = \overline{x_{[1,n]}}$, il s'ensuit que $S' = S''[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$. Si S' n'est pas vide, c'est donc que S'' ne l'est pas et donc, d'après ce qui précède, $u : \alpha^+$ ne peut être la racine de T et comme T est une structure de démonstration, il existe un unique $v : \gamma^- \in S''$ tel que $|v|_{\mathbf{X}} = 1$. Ainsi, $S' = S^3, v : \gamma^-$ et $S'' = S^3, v.[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$. Soit $w : \gamma'^+ \in S^3$ alors $v \prec w$ si et seulement si $v.[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t] \prec w$. En effet, comme t est totalement instancié, à l'exception de $\mathbf{X}$, v contient exactement les mêmes inconnues que $v.[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$ et donc si v contient la tête de w, comme celle-ci doit être différente de $\mathbf{X}$, il vient que $v[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$ contient la tête de w. Réciproquement, si v contient l'inconnue $\mathbf{Y}$, alors il existe $w : \gamma^+$ tel que $|w|_{\mathbf{Y}} = 1$. Donc $\mathbf{Y}$ est la tête de w et $v \prec w$; de la même manière que précédemment on obtient que $v[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t] \prec w$. De plus, comme v est négatif, sa tête reste inchangée par la substitution $[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$ donc si il existe $w : \gamma^+$ tel que $w \prec v$, $w \prec v.[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$.

Pour conclure, il reste à montrer que $v[\mathbf{X} := \lambda\overline{x_{[1,n]}}.t]$ est un terme linéaire, c'est-à-dire que $FV(v) \cap FV(t) = \emptyset$. Or comme T est une structure de démonstration, pour tout $x \in FV(t)$, il existe un unique $w : \gamma'^{\epsilon}$ tel que $x \in FV(w)$, et $\epsilon = +$.

Ainsi, T' est une structure de démonstration. □

Ce lemme nous permet d'obtenir le fait que si $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_P (\cdot; \cdot; \emptyset)$ alors il existe un terme clos de type α.

Nous allons maintenant voir comme les problèmes de recherche de démonstration sont reliés aux λ-termes.

Lemme 1.8 *Soit t un terme clos sous forme η-longue, de type α et dont les variables liées sont deux à deux distinctes, alors $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_{It} (\Gamma; \Delta; S)$ avec $\overline{\Delta} = BV(t)$ et pour tout sous-terme de t $u = x\overline{v_{[1,n]}}$ de type atomique γ, si $v_i = \lambda\overline{x_{S_i}}.w_i$ avec w_i de type atomique γ_i :*

1. $x\overline{\mathbf{X}_{[1,n]}} : \gamma^- \in S$
2. $\mathbf{X}_i\overline{x_{S_i}} : \gamma_i^+ \in S$
3. $x : \overline{\alpha_n} \multimap \alpha \in \Delta$
4. $\mathbf{X}_i : \alpha_i \in \Gamma$

Démonstration. Nous allons simplement montrer comment l'on dirige l'évaluation de la fonction It. Pour cela, on utilise un système de réécriture sur les triplets $(\Gamma; \Delta; S)$ où Γ et Δ sont les contextes habituels et S est un ensemble de triplets $(u, v : \alpha^\epsilon)$ où u est un sous-terme de t et v et le terme que l'on construit pour le problème de recherche de démonstration. On note $\rightarrow^t_{It}$ ce système de réécriture et il est défini par :

1. $(\Gamma; \Delta; (\lambda x.u, v : \alpha \multimap \beta^+), S) \rightarrow^t_{It} (\Gamma; \Delta, x : \alpha; (x, x : \alpha^-), (u, vx : \beta^+), S)$
2. $(\Gamma; \Delta; (u, v : \alpha \multimap \beta^-), S) \rightarrow^t_{It} (\Gamma; \Delta; (u', \mathbf{Y} : \alpha^+), (uu', v\mathbf{X} : \beta^-), S)$ si $t = C[uu']$

□

Avec ce lemme on montre comment on peut choisir les noms de variables pour qu'un problème de recherche de démonstration utilise les mêmes variables qu'un terme donné. On aurait tout aussi bien pu changer les variables du terme pour qu'elles correspondent avec celles du problème.

Lemme 1.9 *Soit t un terme clos β-normal et η-long de type α (les variables liées de t sont deux à deux distinctes), soit $\mathcal{T} = \overline{v_{[1,n]}}$ un ensemble de sous-termes négatifs(i.e. de type atomique et dont la tête est une variable) de t qui sont deux à deux indépendants et respectivement de type $\gamma_1, \ldots, \gamma_n$, alors $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_P (\Gamma; \Delta; S)$, $\overline{v_{[1,n]} : \gamma^-_{[1,n]}} \subseteq S$ et pour tout sous-terme $u = x\overline{v_{[1,n]}}$ de type atomique γ tels que $u \notin \mathcal{T}$:*

1. $x\overline{\mathbf{X}_{[1,n]}} : \gamma^- \in S$
2. *si $v_i = \lambda\overline{x_{S_i}}.w_i$ et w_i est de type atomique γ_i alors $\mathbf{X}_i\overline{x_{S_i}} : \gamma_i^+ \in S$*

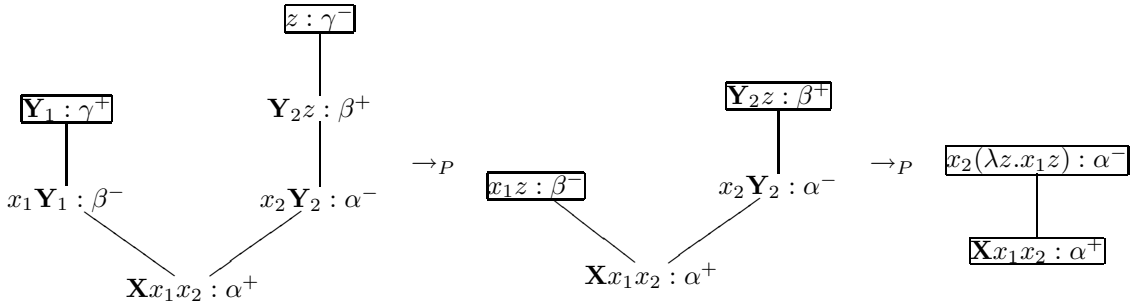


FIG. 1.3 – Exemple : construction d'un terme de type $(\gamma \multimap \beta) \multimap ((\gamma \multimap \beta) \multimap \alpha) \multimap \alpha$

Démonstration. On procède par induction sur la longueur cumulée des termes de $\mathcal{T}$. Si cette longueur est nulle alors $\mathcal{T} = \emptyset$ et on peut conclure avec le lemme précédent. Sinon soit $u = x\overline{v_{[1,n]}}$ un élément de T de type γ avec $v_i = \lambda\overline{x_{S_i}}.w_i$ et w_i de type atomique γ_i. On pose $T' = (T \backslash \{u\}) \cup \overline{w_{[1,n]}}$, la taille cumulée des termes qui composent T' est strictement inférieure à celle de T et on peut appliquer l'hypothèse d'induction et $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_P (\Gamma; \Delta; S)$ et $(\Gamma; \Delta; S)$ remplit les propriétés voulues. En particulier on a :

$$S = x\overline{\mathbf{X}_{[1,n]}} : \gamma^-, \mathbf{X}_1\overline{x_{S_1}} : \gamma_1^+, \ldots, \mathbf{X}_n\overline{x_{S_n}} : \gamma_n^+, , w_1 : \gamma_1^-, \ldots, w_n : \gamma_n^-, S''$$

comme pour tout $i \in [1,n]$ $\overline{x_{S_i}} \subseteq FV(w_i)$, on a :

$$(\Gamma; \Delta; S) \xrightarrow{n}_P (\Gamma'; \Delta'; x\overline{v_{[1,n]}} : \gamma^-, S'')$$

Ce qui nous permet de conclure. □

À partir de ce lemme, on peut démontrer que si il existe un terme t de type α alors $It(\mathbf{X} : \alpha^+; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_P (\cdot; \cdot; \emptyset)$. En effet, si $\alpha = \overline{\alpha_{[1,n]}} \multimap \beta$ où β est un type atomique, $t = \lambda\overline{x_{[1,n]}}.t'$ et t' est un terme négatif. Or du lemme précédent on obtient que $It(\alpha^+; \mathbf{X}) \xrightarrow{*}_P \{\mathbf{X} : \alpha; \overline{x_{[1,n]} : \alpha[1,n]}; \mathbf{X}\overline{x_{[1,n]}} : \beta^+; t' : \beta^-\} \rightarrow_P (\cdot; \cdot; \emptyset)$.

On pourra voir un exemple de construction de terme à la figure 1.3. Nous avons encadré les termes positifs et négatifs qui sont associés à chaque étape par l'algorithme.

Théorème 1.12 *Le problème de recherche de démonstration* $It(\mathbf{X} : \alpha; \cdot; \mathbf{X} : \alpha^+) \xrightarrow{*}_P (\cdot; \cdot; \emptyset)$ *si et seulement si il existe un terme clos de type* α.

La technique mise en œuvre par cet algorithme peut être réemployée pour effectuer des recherches de démonstration et des constructions de λ-termes dans d'autres cas. En effet, cet algorithme permet par exemple de construire des termes bien typés dans le λ-calcul linéaire avec types dépendants et permet dans ce cas de faire l'économie de la résolution d'équations de filtrage difficiles ; il s'adapte également au cas où on utilise les quantificateurs du premier ordre ou encore le tenseur. Dans le cadre des grammaires catégorielles abstraites, il nous fournira un moyen de mener des analyses.

Chapitre 2

Les grammaires catégorielles abstraites

Nous avons défini l'ensemble des concepts (logique linéaire et λ-calcul linéaire) nécessaires à la définition des grammaires catégorielles abstraites. Mais avant de définir ces grammaires, nous allons présenter le cadre dans lequel elles ont émergé. Dans ce chapitre nous verrons comment les grammaires catégorielles abstraites s'articulent par rapport à la tradition des grammaires catégorielles, en particulier par rapport à leur formulation la plus moderne, les grammaires de type logique. Les grammaires de type logique permettent de présenter à travers le prisme de la théorie de la démonstration les idées de Montague concernant la sémantique des langues. Radicalisant encore l'approche des grammaires de type logique, les grammaires catégorielles abstraites essaient d'utiliser les idées de Montague pour décrire tant la syntaxe que la sémantique. Nous verrons ensuite une présentation mathématique des grammaires catégorielles abstraites. Pour finir, nous verrons comment certains formalismes grammaticaux sont codés naturellement dans les grammaires catégorielles abstraites.

2.1 Des grammaires de type logique aux grammaires catégorielles abstraites

Les grammaires de type logique sont des grammaires où la notion d'analyse lexicale est interprétée en terme de démonstration dans une logique bien définie. Ces grammaires prennent leurs origines dans les travaux d'Adjukiewicz [Ajd36] qui furent repris et améliorés par Bar-Hillel [BH50] sous la forme de grammaires catégorielles. Le principe de ces grammaires était d'associer aux mots et aux groupes de mots des *catégories* qui modélisent leurs propriétés grammaticales. Ensuite on vérifie en ne travaillant plus que sur ces catégories si ces mots mis bout à bout forment une phrase. Ce premier travail est essentiel, puisqu'il marque la prise en compte des notions abstraites développées par les grammairiens et qu'il s'attache à développer une théorie mathématique sur l'agencement de ces notions abstraites.

Un pas supplémentaire est franchi avec la contribution de Lambek [Lam58] qui pour palier aux insuffisances du calcul de Bar-Hillel propose de manipuler les catégories à l'aide d'une logique qui désormais porte son nom : le calcul de Lambek. Dans ce système percent déjà, et parfois de façon plus aigüe, les idées qui conduisirent à l'invention de la logique linéaire. Bien avant la logique linéaire, Lambek avait pensé à ôter les règles structurelles de la logique traditionnelle afin d'y rendre sensible tant la notion de ressource que celle d'ordre. Ces deux notions sont cruciales pour pouvoir modéliser la syntaxe de langue naturelle, l'une pour modéliser la satisfaction de dépendances syntaxiques, l'autre pour modéliser la morphologie des phrases. Enfin, des extensions du système de Lambek ont été proposées et étudiées par Moortgat et Morrill. Ces extensions peuvent augmenter encore les restrictions structurelles en rendant la virgule qui sépare les formule ',' non associative, et ajoutent des modalités d'associativité et de commutativité sur les formules qui sont le pendant des exponentielles de la logique linéaire vis-à-vis de la contraction et de l'affaiblissement.

Les grammaires de type logique se fondent donc sur une logique pour définir la notion de correction syntaxique dans une phrase. Plus précisément leur principe consiste à donner à chaque mot un ensemble de *types*, c'est-à-dire de formules de la logique considérée, qui représentent l'ensemble des comportements

syntaxiques recensés pour ce mot. Cette assignation multiple permet en particulier de rendre en compte des phénomènes d'homonymie ; par exemple *orange* en tant que fruit n'a pas le même comportement syntaxique que *orange* en tant que couleur. Vérifier la correction d'une phrase revient à assigner à chaque mot un des types que lui associe le lexique et à démontrer qu'avec ces types pris dans l'ordre des mots, on peut démontrer le type associé à la notion de phrase. Pour vérifier la correction syntaxique de *le chat boit le lait* dans une grammaire de type logique donnée, il s'agit de choisir un type pour *le*, *chat*, *boit*, *le* et *lait*, disons α_1, α_2, α_3, α_4 et α_5, puis de démontrer, dans la logique utilisée, le séquent $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 \vdash S$ si S est le type associé à la notion syntaxique de phrase.

Une propriété remarquable des grammaires de type logique est leur adéquation avec la notion de sémantique développée par Montague [Mon74]. Les travaux de Montague tentent d'associer à une phrase une formule logique exprimant son sens. Pour cela Montague reprend un principe déjà proposé par Frege, principe qui stipule que le sens d'une phrase est une composition du sens des mots qui la constituent et que la composition de ces sens s'opère en rapport étroit avec la structure syntaxique de la phrase. Montague a proposé des approches concrètes pour mettre en œuvre ce principe et a ouvert la voie à la sémantique moderne. Pour cela, il associe à chaque nœud d'une structure d'analyse un λ-terme qui est appliqué aux λ-termes associés aux nœuds fils *etc...* Le terme sous forme normal obtenu après β-réduction représente le sens de la phrase. Cette approche présente de nombreux problèmes techniques qui obligent à *élever* les types des termes employés, à utiliser des logiques qui permettent des mouvements de quantificateurs...

Comme les λ-termes utilisés par Montague sont simplement typés ils représentent des démonstrations dans la logique intuitionniste. De plus comme ces termes sont appliqués les uns aux autres en respectant la discipline des types, en construisant un terme sémantique selon les idées de Montague, on transforme un arbre syntaxique en une démonstration de la logique intuitionniste. Les grammaires de type logique ont pour dérivation des démonstrations, or quoi de plus naturel que de transformer une démonstration en une autre démonstration. Cela est d'autant plus simple que les logiques employées pour construire des grammaires de type logique sont des logiques plus faibles que la logique intuitionniste et il y a une méthode canonique pour transformer les démonstrations de ces logiques en des démonstrations de la logique intuitionniste. Cette méthode consiste à transformer chaque opérateur de la logique utilisée par la grammaire par l'opérateur de la logique intuitionniste qui lui correspond (la flèche intuitionniste pour les implications à gauche et à droite de Lambek par exemple) et d'utiliser l'opération structurelle qui correspond à l'usage de chaque modalité. Cela revient à transformer la démonstration qui tient lieu d'analyse en une démonstration de la logique intuitionniste en utilisant un homomorphisme.

Pour illustrer quelque peu notre propos, nous allons introduire très brièvement le calcul de Lambek et la façon dont la sémantique de Montague tire parti de l'analyse syntaxique pour associer une formule logique à la phrase analysée.

Les formules du calcul de Lambek se définissent à partir d'un ensemble fini de types atomiques $\mathcal{A}$ suivant la grammaire :

$$\mathcal{L} ::= \mathcal{A} \mid \mathcal{L}/\mathcal{L} \mid \mathcal{L}\backslash\mathcal{L}$$

Les démonstrations du calcul de Lambek se construisent à partir de règles du système formel suivant :

$$\frac{}{\alpha \vdash_L \alpha}\, Axiome \qquad \frac{\Gamma \vdash_L \alpha \quad \Delta, \alpha, \Delta' \vdash_L \gamma}{\Delta, \Gamma, \Delta' \vdash_L C}\, Coupure$$

$$\frac{\Gamma, \beta, \Gamma' \vdash_L \gamma \quad \Delta \vdash_L \alpha}{\Gamma, \beta/\alpha, \Delta, \Gamma' \vdash_L \gamma}\, /_G \qquad \frac{\Delta \vdash_L \alpha \quad \Gamma, \beta, \Gamma' \vdash_L \gamma}{\Gamma, \Delta, \alpha\backslash\beta, \Gamma' \vdash_L \gamma}\, \backslash_G$$

$$\frac{\Gamma, \alpha \vdash_L \beta}{\Gamma \vdash_L \beta/\alpha}\, /_D \qquad \frac{\alpha, \Gamma \vdash_L \beta}{\Gamma \vdash_L \alpha\backslash\beta}\, \backslash_D$$

Il faut remarquer dans les règles précédentes que Γ et Δ sont des listes de formules.

Si on considère le lexique qui à *Jean* associe np, à et à *Marie* associe $(np\backslash S)/np$ à *aimer* , alors montrer que la phrase *Jean aime Marie* est syntaxiquement correcte dans le calcul de Lambek revient à démontrer le séquent $np, (np\backslash S)/np, np \vdash_L S$:

$$\dfrac{\dfrac{\dfrac{}{np \vdash np}\,Axiome \qquad \dfrac{}{S \vdash S}\,Axiome}{np, np\backslash S \vdash S}\backslash_G \qquad \dfrac{}{np \vdash np}\,Axiome}{np, (np\backslash S)/np, np \vdash S}/_G$$

La sémantique de Montague profite de la structure des démonstrations associées aux analyses pour les transformer en des démonstrations de la logique intuitionniste. Pour cela, il suffit de remarquer que les opérateurs logiques / et \ du calcul de Lambek se comportent d'une manière similaire à la flèche $\rightarrow$ de la logique intuitionniste. Techniquement, ces deux opérateurs sont des flèches linéaires à *gauche* et à *droite*. Ainsi, en transformant à l'aide d'un homomorphisme $\mathcal{H}$ les formules qui interviennent dans une démonstration du calcul de Lambek d'un séquent $\Gamma \vdash_L C$, on obtient une démonstration en logique intuitionniste de $\mathcal{H}(\Gamma) \vdash \mathcal{H}(C)$. Un homomorphisme tel que $\mathcal{H}$ est défini par induction sur la structure des formules :

1. si α est une formule atomique, $\mathcal{H}(\alpha)$ peut être une formule quelconque de la logique intuitionniste.
2. $\mathcal{H}(\beta/\alpha) = \mathcal{H}(\alpha) \rightarrow \mathcal{H}(\beta)$
3. $\mathcal{H}(\alpha\backslash\beta) = \mathcal{H}(\alpha) \rightarrow \mathcal{H}(\beta)$

Pour définir un tel homomorphisme, il suffit de donner les formules qu'il associe aux formules atomiques. Par exemple si on définit $\mathcal{H}$ comme l'homomorphisme qui fait correspondre e à np et t à S , on obtient que $\mathcal{H}((np\backslash S)/np) = e \rightarrow e \rightarrow t$. L'homomorphisme $\mathcal{H}$ permet de démontrer le séquent $e, e \rightarrow e \rightarrow t, e \vdash t$:

$$\dfrac{\dfrac{\dfrac{}{e \vdash e}\,Axiome \qquad \dfrac{}{t \vdash t}\,Axiome}{e, e \rightarrow t \vdash t}\rightarrow_G \qquad \dfrac{}{e \vdash e}\,Axiome}{e, e \rightarrow e \rightarrow t, e \vdash t}\rightarrow_G$$

à cette démonstration, l'isomorphisme de Curry-Howard associe un terme :

$$\dfrac{\dfrac{\dfrac{}{x : e \vdash x : e}\,Axiome \qquad \dfrac{}{f'' : t \vdash f'' : t}\,Axiome}{xe, f' : e \rightarrow t \vdash f'x : t}\rightarrow_G \qquad \dfrac{}{y : e \vdash y : e}\,Axiome}{x : e, f : e \rightarrow e \rightarrow t, y : e \vdash fyx : t}\rightarrow_G$$

Jusqu'ici, nous avons simplement transformé une démonstration du calcul de Lambek représentant une analyse grammaticale en une démonstration de la logique intuitionniste. Cela ne nous donne pas pour autant une formule logique pour représenter le sens de la phrase. Les grammaires de type logique associent en fait à chaque entrée lexicale (*i.e.* à chaque couple formé d'un mot et d'une formule du calcul de Lambek) un terme sémantique ayant un type en accord avec celui associé à cette entrée par l'homomorphisme. Pour notre exemple, on associerait la constante de type e J comme terme sémantique de *Jean*, la constante de type e M comme terme sémantique de *Marie* et le terme $\lambda xy.love\, y\, x$ comme terme sémantique de *aime*. Obtenir la sémantique de Montague consiste finalement à remplacer dans le λ-terme obtenu avec l'homomorphisme les variables par la sémantique qui leur correspond. Après l'analyse de *Jean aime Marie* on obtient le terme $(\lambda xy.love\, y\, x)MJ$, qui après β-réduction donne pour sémantique à la phrase le terme $love\, J\, M$.

La sémantique de Montague ne se borne pas à obtenir des formules logiques à partir d'analyses grammaticales, elle incorpore une notion de modèle qui prend en compte les mondes possibles ainsi que la temporalité et définit ainsi une notion de vérité pour cette phrase analysée. Dans un tel modèle, le prédicat *love* serait représenté par un ensemble de paires telles que la première composante *aimerait* la seconde ; il nous serait alors possible de vérifier la valeur de vérité de la phrase *Jean aime Marie* en vérifiant que la paire formée par les éléments associés aux constantes J et M dans le modèle est bien une paire de cet ensemble. Bien qu'essentielle, voire centrale, dans la sémantique de Montague, nous ne nous attarderons pas sur cette dimension.

Revenons maintenant sur les éléments nécessaires à la construction d'une grammaire de type logique : il faut choisir une logique pour modéliser la syntaxe, un moyen de transformer les démonstrations de cette logique, c'est-à-dire les analyses grammaticales, sous une forme sémantique. Au niveau syntaxique, il est nécessaire de manipuler des chaînes de caractères, au niveau des analyses on est conduit à manipuler des démonstrations dans la logique que l'on a choisie et enfin au niveau de la sémantique on manipule des λ-termes. Nous avons déjà vu que les chaînes de caractères peuvent être modélisées avec le λ-calcul. L'isomorphisme de Curry-Howard rend possible la représentation des démonstrations, donc des analyses, sous forme de λ-termes. On peut voir les grammaires de type logique comme un formalisme où on utilise uniformément le λ-calcul tant pour représenter la morphologie, les dérivations syntaxiques et la sémantique. Le passage de l'une à l'autre de ces dimensions est assuré par des homomorphismes. Si tout peut être modélisé par le λ-calcul, autant ne plus distinguer formellement les traitements de la syntaxe et de la sémantique, mais voir ces traitements comme des instances particulières d'un problème plus général. L'idée sous-jacente aux grammaires catégorielles abstraites part de cette constatation.

Les grammaires catégorielles abstraites utilisent la théorie des types et le λ-calcul pour représenter à la fois la syntaxe à proprement parler, les analyses grammaticales et enfin les structures sémantiques. La théorie des types fournit une discipline qui permet de structurer mathématiquement les problèmes grammaticaux ; c'est-à-dire un monde abstrait qui s'intéresse avant tout au problème abstrait des catégories grammaticales. À l'inverse, le λ-calcul s'intéresse à tous les détails morphologiques et permet de transposer concrètement l'agencement des mots ou des prédicats sémantiques en s'appuyant sur les dépendances grammaticales. Le couple type et λ-calcul fonctionne donc comme celui d'une théorie grammaticale et de sa réalisation prosodique. Une autre métaphore, peut-être plus adéquate, nous est donnée par la théorie de la démonstration. Dans cette théorie un type est vu comme une spécification et le λ-terme comme un programme qui implémente cette spécification. Ici, les types nous donnent une définition formelle de ce qu'est une phrase, un groupe nominal... et les λ-termes l'agencement des mots qui les forment.

Une théorie grammaticale que les types pourraient peut-être implémenter serait celle de Tesnière [Tes59] où le verbe est central et est décrit comme ayant un certain nombre d'*actants* (prime actant, second actant, tiers actant et parfois quatrième actant) et de *circonstants* (ensemble des circonstances qui encadrent l'action du verbe : adverbes, compléments circonstanciels...). Dans cette théorie, un verbe transitif est représenté comme un mot nécessitant deux actants : un prime actant, le sujet, et un second actant, le complément d'objet direct. Pour simplifier les choses, nous considérerons que ces actants sont des groupes nominaux et qu'il n'y a pas de circonstants. Un type possible pour modéliser les verbes transitifs serait $np \multimap np \multimap S$ si les types atomiques np et S modélisent respectivement les notions syntaxiques *abstraites* de groupe nominal et de phrase. Dans une telle théorie simplifiée, tous les verbes transitifs auraient pour type $np \multimap np \multimap S$. Ce type très général pourrait s'appliquer sans difficulté à de nombreuses langues, ou pourtant la syntaxe des verbes est très différente comme par exemple le français et l'allemand. C'est en ce sens que les grammaires catégorielles abstraites sont effectivement abstraites. Les types contrôlent seulement les dépendances syntaxiques sans prendre en compte la façon dont les mots s'agencent les uns par rapport aux autres. Modéliser, même dans notre vision très simplifiée des verbes, les verbes transitifs en français et en allemand à l'aide du calcul de Lambek représente deux tâches différentes. Bien entendu, le fait qu'un tel verbe ait besoin de deux groupes nominaux pour construire une phrase apparaîtrait à la fois dans le cas du français et dans celui de l'allemand. Cependant les types seraient fortement marqués par les nécessités morphologiques. Il faudrait qu'en plus des informations grammaticales, ces types contiennent des informations sur la façon dont les mots sont ordonnés. Dans le cadre des grammaires catégorielles abstraites, tout cela est relégué au niveau du λ-calcul.

Le λ-calcul nous offre une grande souplesse pour exprimer les propriétés d'ordre sur les mots. Il faut garder à l'esprit que l'isomorphisme de Curry-Howard induit que cette puissance d'expressivité suit la discipline des types. Par conséquent, le choix du système de typage influe fortement les formes syntaxiques qu'une grammaire pourra produire. Pour les grammaires catégorielles abstraites, le système de type choisi est celui de la logique linéaire implicative intuitionniste, le λ-calcul associé est donc le λ-calcul linéaire. Ce choix n'est bien entendu pas limitatif, le système de type peut être enrichi et de son choix dépendra l'équilibre entre l'expressivité de la grammaire et la difficulté de l'analyse. Arne Ranta a créé un tel système, *the grammatical framework* [Ran04], pour la langue naturelle en se fondant sur la logique intuitionniste d'ordre 2.

Pour ce qui est de notre exemple des verbes français et allemands, si on choisit de représenter le verbe

français *boire* et le verbe allemand *trinken* au passé composé et à la troisième personne du singulier on obtiendra alors pour le premier le groupe verbal *a bu* qui attend un sujet en préfixe et un complément d'objet en suffixe, nous le représenterons par le λ-terme linéaire $\lambda xy.x + \ a \ bu \ + y$; pour le second on obtiendra *hat getranken* qui attend un sujet en préfixe et un complément d'objet entre *hat* et *getranken* on utilisera le λ-terme linéaire $\lambda xy.x + \ hat \ + y + \ getranken$.

Les grammaires catégorielles abstraites permettent de la sorte de modéliser des propriétés grammaticales de façon simple et abstraite en les séparant bien de la morphologie. De plus, comme la prosodie, l'analyse et la sémantique sont toutes deux représentées à l'aide de λ-termes linéaires, et chacune peut faire à nouveau l'objet d'une analyse. Ces grammaires permettent donc d'effectuer des analyses en cascade ou de définir des chaînes d'analyses qui pourraient à chaque fois s'intéresser à une propriété particulière que doit vérifier une phrase. Elles offrent ainsi le possibilité de construire des architectures complexes qui dans d'autres formalismes nécessiteraient un nouvel effort d'abstraction.

Maintenant que nous avons vu dans quelle tradition linguistique s'inséraient les grammaires catégorielles abstraites, nous allons présenter mathématiquement ce formalisme et montrer sur quelques exemples son pouvoir d'expressivité en donnant divers exemples de grammaires qu'il permet de coder.

2.2 Définition mathématique

Comme nous venons de le voir, les grammaires catégorielles abstraites se fondent sur le λ-calcul linéaire simplement typé pour modéliser les propriétés syntaxiques et morphologiques des langages et elles utilisent des homomorphismes pour passer d'une dimension à une autre. Dans le cadre des grammaires catégorielles abstraites, ces homomorphismes sont appelés des lexiques.

Définition 2.1 *Étant données deux signatures Σ_1 et Σ_2, un lexique de Σ_1 vers Σ_2 est une paire (f, g) telle que f est une fonction de $\mathcal{C}_{\Sigma_1}$ dans l'ensemble des termes clos typables dans Σ_2 et g est une fonction de $\mathcal{A}_{\Sigma_1}$ dans $\mathcal{I}_{\Sigma_2}$. De plus, pour tout $c \in \mathcal{C}$, $f(c)$ est un terme de type $g(\tau_{\Sigma_1}(c))$ dans Σ_2.*

Définition 2.2 *Étant donné un lexique $\mathcal{L} = (f, g)$ de Σ_1 dans Σ_2, l'extension homomorphique de $\mathcal{L}$, notée $\hat{\mathcal{L}}$ est la paire $(\hat{f}, \hat{g})$ telle que :*

1. $\hat{f}(c) = f(c)$ *pour tout* $c \in \mathcal{C}_{\Sigma_1}$
2. $\hat{f}(x) = x$
3. $\hat{f}(\lambda x.t) = \lambda x.\hat{f}(t)$
4. $\hat{f}(t_1 t_2) = \hat{f}(t_1)\hat{f}(t_2)$

et

1. $\hat{g}(\alpha) = g(\alpha)$ *pour tout* $\alpha \in \mathcal{A}_{\Sigma_1}$
2. $\hat{g}(\alpha \multimap \beta) = \hat{g}(\alpha) \multimap \hat{g}(\beta)$

Notation 2.1 *Étant donné un lexique $\mathcal{L} = (f, g)$, par abus de notation on notera $\mathcal{L}(t)$ le terme $\hat{f}(t)$ et $\mathcal{L}(\alpha)$ le type $\hat{g}(\alpha)$.*

Une première chose à vérifier est qu'un lexique de Σ_1 vers Σ_2 transforme les termes bien typés sur Σ_1 en des termes bien typés sur Σ_2.

Lemme 2.1 *Soient Σ_1 et Σ_2 deux signatures et $\mathcal{L}$ un lexique de Σ_1 dans Σ_2, si t est un terme clos de type α sur Σ_1 alors $\mathcal{L}(t)$ est un terme clos de type $\mathcal{L}(\alpha)$ sur Σ_2.*

Définition 2.3 *Une grammaire catégorielle abstraite est un quadruplet $(\Sigma_1, \Sigma_2, \mathcal{L}, S)$ où Σ_1 est une signature appelée la signature abstraite, Σ_2 est une signature appelée la signature objet, $\mathcal{L}$ est un lexique de Σ_1 dans Σ_2 et S est un type atomique $\mathcal{A}_{\Sigma_1}$.*

Une grammaire catégorielle abstraite définit plusieurs langages, un langage abstrait qui est le langage des analyses, et un langage objet qui est le langage concret de la grammaire.

Définition 2.4 *Le langage abstrait d'une grammaire catégorielle abstraite* $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ *est l'ensemble des termes clos de* Σ_1 *dont le type est* S*. On note ce langage* $\mathcal{A}bs(\mathcal{G})$*.*

Définition 2.5 *On appelle langage objet (ou plus simplement langage) d'une grammaire catégorielle abstraite* $\mathcal{G}$ *l'ensemble :*

$$\mathcal{O}bj(\mathcal{G}) = \{u | \exists t \in \mathcal{A}bs(\mathcal{G}).\mathcal{L}(t) =_{\beta\eta} u\}$$

Définition 2.6 *Une grammaire catégorielle abstraite* $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ *est dite lexicalisée si pour tout* $c \in \mathcal{C}_{\Sigma_1}$*,* $\mathcal{L}(c)$ *est un terme lexicalisé.*

Définition 2.7 *Une grammaire catégorielle abstraite* $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ *est dite semi-lexicalisée si* c *étant une constante de* $\mathcal{C}_{\Sigma_1}$*, le fait que* $\mathcal{L}(c)$ *soit non-lexicalisé implique que* $ord(\tau_{\Sigma_1}(c)) \leq 2$*.*

Analyser un terme u du langage objet revient à trouver un terme du langage abstrait t dont la transformation par le lexique $\mathcal{L}(t)$ est égale à u, modulo $\beta\eta$-conversion.

Pour définir une grammaire catégorielle abstraite, il faut donner la signature abstraite, la signature objet, et définir le lexique. Dans la suite, ou bien nous donnerons une définition exhaustive des signatures et des lexiques ou bien nous donnerons une série de notations abrégées de la forme :

$$c := t : \alpha$$

pour définir que c est une constante abstraite de type α à laquelle le lexique associe le terme objet t.

Comme les lexiques sont des homomorphismes, ils peuvent être manipulés formellement. Si nous avons défini deux grammaires catégorielles abstraites $(\Sigma_1, \Sigma_2, \mathcal{L}_1, S)$ et $(\Sigma_2, \Sigma_3, \mathcal{L}_2, S)$ telles que la signature objet de la première est la signature abstraite de la seconde, il nous est alors possible de définir, par composition des lexiques une grammaire catégorielle abstraite $(\Sigma_1, \Sigma_3, \mathcal{L}_1 \circ \mathcal{L}_2, S)$ ayant pour signature abstraite la signature abstraite de la première grammaire et pour signature objet celle de la seconde. D'un point de vue pratique, cela ouvre de nombreuses perspectives. Imaginons que l'on veuille élaborer un système de commande par la parole. Il nous est nécessaire d'analyser des phrases constituées de phonèmes. Il est assez facile de faire correspondre aux mots de la langue que l'on veut analyser les phonèmes qui rendent compte de leur prononciation. On peut donc facilement définir une grammaire catégorielle abstraite dont la signature abstraite est constituée de l'ensemble des mots de la langue considérée et la signature objet de l'ensemble des phonèmes de cette langue, le lexique faisant correspondre à chaque mot la chaîne de phonèmes qui représente sa prononciation. D'un autre côté, on peut définir une grammaire catégorielle abstraite qui analyse les phrases de la langue considérée à partir des mot qui les composent c'est-à-dire une grammaire catégorielle abstraite qui a pour langage abstrait l'ensemble des analyses grammaticales possibles et pour langage objet l'ensemble des phrases. Pour constituer une grammaire catégorielle abstraite qui analyse directement les suites de phonèmes, il suffit de générer automatiquement le lexique qui est la composition des lexiques qui permettent de passer des analyses aux phrases et des phrases à leur prosodie. Cette architecture est avantageuse car elle permet de traiter séparément les problèmes grammaticaux de la langue et ceux de sa prosodie, puis de créer automatiquement la grammaire qui permet d'analyser directement une suite de phonèmes. La maintenance d'un tel système de dialogue en est facilitée.

Avoir les lexiques comme éléments de base des grammaires ouvre d'autres possibilités techniques. La conception de systèmes multilingues s'avère facilitée. On peut par exemple utiliser un langage abstrait pivot qui puisse représenter les analyses grammaticales des différentes langues envisagées. Cette technique fait la supposition que les langues considérées sont décrites par les mêmes structures grammaticales. Cela est en général vrai pour des langues proches, issues d'une même famille, par exemples les langues indo-européennes, ou parmi ces langues, les langues latines, saxonnes ou slaves. Ensuite, il faut définir une signature objet pour chaque langue que l'on veut inclure dans le système, ainsi qu'un lexique qui permette de passer du langage abstrait à chacun des langages particuliers. La tâche de traduire d'une première langue vers une seconde revient à analyser une phrase dans la première pour obtenir un terme du langage abstrait et transformer ce terme en une phrase de la seconde langue à l'aide du lexique approprié. Ce genre d'architecture a été employé avec succès par Arne Ranta et son *Grammatical Framework* [KNR03].

Afin d'illustrer ce type d'architecture, revenons à la modélisation en français et en allemand du verbe boire au passé composé. Nous donnons une première grammaire qui illustre une modélisation pour le français :

1. $J := Jean : np$
2. $A := \lambda x.une + x : n \multimap np$
3. $Beer :=$ *bière* $: n$
4. $Drank := \lambda xyz.x + a + bu + y : np \multimap np \multimap s$

Dans cette grammaire, le terme abstrait $Drank\, J(A\, Bier)$ est réalisé par le lexique par le terme $Jean + a + bu + une +$ *bière*.

Maintenant, on peut réutiliser la signature abstraite de la grammaire précédente pour obtenir une modélisation en allemand :

1. $J := Jean : np$
2. $A := \lambda x.ein + x : n \multimap np$
3. $Beer := Bier : n$
4. $Drank := \lambda xyz.x + hat + y + getranken : np \multimap np \multimap s$

Dans cette nouvelle grammaire, le terme abstrait $Drank\, J(A\, Bier)$ est réalisé par $Jean + hat + ein + Bier + getranken$. Passer d'une phrase en français à une phrase en allemand et réciproquement revient à analyser la première dans une grammaire puis à transformer l'analyse obtenue avec le lexique de la seconde.

Une autre possibilité, toujours pour des applications multilingues, serait de baser une architecture sur la sémantique de Montague. Pour chaque langue L que l'on veut employer, on définit une grammaire catégorielle abstraite $(\Sigma_{1L}, \Sigma_{2L}, \mathcal{L}_L, S_L)$ qui traite ses phénomènes linguistiques propres. Les formules logiques qui modélisent la sémantique des phrases analysées sont définies à l'aide d'une signature Σ_{Log}. Pour chaque langue L, on définit une grammaire catégorielle abstraite $Sem_L = (\Sigma_{1L}, \Sigma_{Log}, \mathcal{L}_{LogL}, S_L)$ qui permet de passer des analyses à la sémantique de Montague qui leur correspond. Dans ce cadre, la traduction d'une phrase d'une langue L_1 vers une langue L_2 consiste à analyser la phrase dans la langue L_1, lui associer sa sémantique de Montague, analyser le terme de cette sémantique dans la grammaire Sem_{L_2} ; cette analyse correspond à l'analyse d'une phrase dans L_2 qui a la même sémantique de Montague et l'on obtient cette phrase en utilisant le lexique $\mathcal{L}_{L_2}$.

Outre le fait qu'il semble illusoire d'avoir un algorithme qui permette de transformer simplement des formes logiques en des phrase, ce genre de modèle se heurte aux limites des grammaires catégorielles abstraites telles que nous venons de les définir. En effet, les termes employés pour la sémantique de Montague sont rarement des termes linéaires. Il faudrait que les lexiques puissent transformer des types linéaires en types de la logique intuitionniste. Pour l'exemple que nous venons de prendre, il faudrait un lexique de la forme suivante :

1. $J := \lambda P.P(j) : np$
2. $A := \lambda PQ.\exists x.P(x) \wedge Q(x) : n \multimap np$
3. $Beer := \lambda x.bier(x) : n$
4. $Drank := \lambda so.s(o(\lambda xy.drank\, y\, x)) : np \multimap np \multimap s$

Et ce lexique ferait correspondre $e \to t$ à n, $(e \to t) \to t$ à np et t à s. En revanche, il ne semble pas nécessaire sur cet exemple que le lexique transforme la flèche linéaire $\multimap$ en flèche intuitionniste $\to$. En effet, le terme : $\lambda PQ.\exists x.P(x) \wedge Q(x)$ peut être typé avec $(e \to t) \multimap (e \to t) \to t$ et le terme $\lambda so.s(o(\lambda xy.drank\, y\, x))$ peut être typé avec $((e \to t) \to t) \multimap ((e \to t) \to t) \multimap t$. Nous n'avons pas vérifié si cela restait vrai pour une sémantique de Montague plus complète.

2.3 Expressivité

Un problème que nous n'avons pas abordé au cours de cette thèse, mais qui semble réellement difficile est l'évaluation de l'expressivité des grammaires catégorielles abstraites. Contrairement à l'intuition que l'on pourrait avoir, les langages reconnus par les grammaires catégorielles abstraites ne sont pas forcément semi-linéaires [BG01]. Nous n'allons donner ici qu'une évaluation qualitative de cette expressivité, en montrant comment les grammaires catégorielles abstraites peuvent encoder des formalismes grammaticaux connus. Les formalismes grammaticaux dont nous donnons l'encodage ont été choisis parce

qu'ils ont été largement étudiés et qu'ils nous permettront d'évaluer la complexité de l'analyse des grammaires catégorielles abstraites et le cas échéant de mesurer l'efficacité d'algorithmes d'analyse. Sinon, pour une étude plus détaillée de l'expressivité des grammaires catégorielles abstraites, on pourra se référer à [dGP03] qui démontre que les grammaires catégorielles abstraites peuvent coder les systèmes de réécriture hors contexte m-linéaires, systèmes de réécriture dans lesquels peuvent se coder de nombreux formalismes grammaticaux (grammaires d'arbres adjoints multi-composantes, grammaires hors contexte multiples, grammaires minimalistes...).

2.3.1 Hiérarchie des langages

Avant d'aborder les problèmes d'expressivité des grammaires catégorielles abstraites, nous allons définir une hiérarchie assez naturelle des langages qu'elles définissent.

Définition 2.8 *Nous noterons $\mathcal{L}(n,m)$ l'ensemble des grammaires catégorielles abstraites $(\Sigma_1, \Sigma_2, \mathcal{L}, S)$ qui vérifient les propriétés suivantes :*

1. *Σ_1 est une signature d'ordre n*
2. *pour tout $\alpha \in \mathcal{A}_{\Sigma_1}$ $(ord(\mathcal{L}(\alpha))) \leq m$*

Intuitivement, l'emploi de cette classification des langages se justifie par le fait que l'ordre d'un terme est un indicateur de la complexité des opérations qu'il peut mener. Une simple substitution dans une chaîne de caractères est une opération du second ordre, une adjonction est une opération d'ordre 3... Cette classification servira également de support pour l'évaluation de la complexité de l'analyse des grammaires catégorielles abstraites.

2.3.2 Automates d'arbres et transducteurs linéaires

Dans cette partie, nous allons voir que les grammaires de type $\mathcal{L}(2,1)$ correspondent aux langages réguliers d'arbres. Les langages réguliers d'arbres sont les langages reconnus par les automates d'arbres.

Afin de ne pas introduire de nouvelles notations, nous utilisons les signatures pour définir les ensembles d'arbres.

Définition 2.9 *Une signature d'arbres est une signature Σ telle que $\mathcal{A}_\Sigma = \{*\}$ et que pour tout $c \in \mathcal{C}_\Sigma$, $ord(\tau_\Sigma(c)) = 2$. Un arbre sur Σ est un terme clos de type $*$.*

Définition 2.10 *On dit qu'une signature d'arbres Σ est une signature d'états si pour tout $c \in \mathcal{C}_\Sigma$, $\tau_\Sigma(c) = * \multimap *$.*

Définition 2.11 *Un automate d'arbres est un quadruplet $A = (\Sigma_S, \Sigma_T, R, q)$ telle que Σ_S est une signature d'état, Σ_T est une signature d'arbres, R est un ensemble de règles de la forme $q_0(f\overline{x_{[1,n]}}) \rightarrow_T f(\overline{q_{[1,n]}x_{[1,n]}})$ (avec $\rho(\tau_{\Sigma_T}(f)) = n$) et q est un élément de $\mathcal{C}_{\Sigma_S}$.*

On dit qu'un arbre t sur Σ_T est reconnu par A si $qt \overset{*}{\rightarrow}_T t$.

Étant donnés un automate d'arbres $A = (\Sigma_S, \Sigma_T, R, q)$ et un arbre t sur Σ_T, nous allons construire une grammaire catégorielle abstraite de $\mathcal{L}(2,1)$ $\mathcal{G}_A$ dont la signature abstraite est Σ_T et telle que t appartient au langage de $\mathcal{G}_A$ si et seulement si t est reconnu par A. Nous nous bornerons ici à donner $\mathcal{G}_A$, les démonstrations de ces propriétés sont faciles à établir.

La grammaire $\mathcal{G}_A$ est définie par le quadruplet $(\Sigma_A, \Sigma_T, \mathcal{L}_A, q)$ avec :

1. $\mathcal{C}_{\Sigma_A} = R$
2. $\mathcal{A}_{\Sigma_A} = \mathcal{C}_{\Sigma_S}$
3. si $r \in R$ alors $\tau_{\Sigma_A}(r) = \overline{q_{[1,n]}} \multimap q_0$ si $r = q_0(f\overline{x_{[1,n]}}) \rightarrow_T f(\overline{q_{[1,n]}x_{[1,n]}})$
4. $\mathcal{L}_A(r) = \lambda\overline{x_{[1,n]}}.f\overline{x_{[1,n]}}$ si $r = q_0(f\overline{x_{[1,n]}}) \rightarrow_T f(\overline{q_{[1,n]}x_{[1,n]}})$
5. pour tout $q' \in \mathcal{A}_{\Sigma_A}$, $\mathcal{L}_A(q') = *$.

Nous allons voir sur un exemple comment ce codage fonctionne avec l'automate d'arbres qui reconnaît les formules booléennes dont la valeur de vérité est 1. Soit $A = (\Sigma_S, \Sigma_T, R, q_1)$ l'automate défini par :

1. $\mathcal{C}_{\Sigma_T} = \{\wedge; \neg; 1; 0\}$ avec $\tau_{\Sigma_T}(\wedge) = * \multimap * \multimap *$, $\tau_{\Sigma_T}(\neg) = * \multimap *$ et $\tau_{\Sigma_T}(1) = \tau_{\Sigma_T}(0) = *$
2. $\mathcal{C}_{\Sigma_S} = \{q_0; q_1\}$
3. $R = \left\{ \begin{array}{lcl} r_1 & = & q_1(\wedge x\, y) \rightarrow_A \wedge(q_1(x))(q_1(y)) \\ r_2 & = & q_0(\wedge x\, y) \rightarrow_A \wedge(q_0(x))(q_1(y)) \\ r_3 & = & q_0(\wedge x\, y) \rightarrow_A \wedge(q_1(x))(q_0(y)) \\ r_4 & = & q_0(\wedge x\, y) \rightarrow_A \wedge(q_0(x))(q_0(y)) \\ r_5 & = & q_0(\neg x) \rightarrow_A \neg(q_1\, x) \\ r_6 & = & q_1(\neg x) \rightarrow_A \neg(q_0\, x) \\ r_7 & = & q_1(1) \rightarrow_A 1 \\ r_8 & = & q_0(0) \rightarrow_A 0 \end{array} \right\}$

La grammaire catégorielle abstraite associée à A est $\mathcal{G}_A = (\Sigma_A, \Sigma_T, \mathcal{L}_A, q)$ définie par :

1. $\mathcal{C}_{\Sigma_A} = \{r_1; \ldots; r_8\}$
2. $\mathcal{A}_{\Sigma_A} = \mathcal{C}_{\Sigma_S}$
3. $\tau_{\Sigma_A}(r_1) = \tau_{\Sigma_A}(r_2) = \tau_{\Sigma_A}(r_3) = \tau_{\Sigma_A}(r_4) = * \multimap * \multimap *$, $\tau_{\Sigma_A}(r_5) = \tau_{\Sigma_A}(r_6) = * \multimap *$ et $\tau_{\Sigma_A}(r_7) = \tau_{\Sigma_A}(r_8) = *$
4. $\mathcal{L}_A(r_1) = \mathcal{L}_A(r_2) = \mathcal{L}_A(r_3) = \mathcal{L}_A(r_4) = \lambda xy. \wedge\, x\, y$, $\mathcal{L}_A(r_5) = \mathcal{L}_A(r_6) = \lambda x. \neg x$ et $\mathcal{L}(r_7) = 1$ et $\mathcal{L}(r_8) = 0$

Le terme $\wedge(\neg(\wedge\, 0\, 1))1$ est reconnu par l'automate en utilisant la réduction suivante (nous indiquons au-dessus de la flèche la règle utilisée pour transformer le terme et nous soulignons où la règle est appliquée dans le terme) :

$$\begin{array}{lll} \underline{q_1(\wedge(\neg(\wedge\, 0\, 1))1)} & \overset{r_1}{\rightarrow}_A & \wedge(q_1(\neg(\wedge\, 0\, 1))\underline{(q_1(1))}) \\ & \overset{r_7}{\rightarrow}_A & \wedge\underline{(q_1(\neg(\wedge\, 0\, 1)))}1 \\ & \overset{r_6}{\rightarrow}_A & \wedge(\neg\underline{(q_0(\wedge\, 0\, 1))})1 \\ & \overset{r_2}{\rightarrow}_A & \wedge(\neg(\wedge\, \underline{q_0(0)}\, q_1(1)))1 \\ & \overset{r_8}{\rightarrow}_A & \wedge(\neg(\wedge\, 0\, \underline{q_1(1)}))1 \\ & \overset{r_7}{\rightarrow}_A & \wedge(\neg(\wedge\, 0\, 1))1 \end{array}$$

Pour composer le terme abstrait qui correspond à cette réduction, il suffit de construire un terme à partir des règles de la réduction conformément à l'endroit où elles ont pris place au cours de la réduction. Le terme qui correspond à cette réduction est $r_1(r_6(r_2\, r_8\, r_7))r_7$, on peut vérifier que le lexique réalise ce terme par $\wedge(\neg(\wedge\, 0\, 1))1$.

Pour montrer que les grammaires catégorielles abstraites de $\mathcal{L}(2,1)$ ont pour langage des langages réguliers d'arbres, la réduction est un peu plus technique. En effet, rien n'impose que la signature objet soit une signature du second ordre et il est possible que les termes associés aux constantes abstraites par le lexique contiennent des λ-abstractions. En fait la transformation d'un terme sous forme normale par le lexique ne crée pas de redex et ces λ-abstractions sont figées. En voyant les abstractions et les variables comme des nœuds de l'arbre syntaxique on parvient à un automate d'arbres qui reconnaît le même langage.

Ainsi, les grammaires catégorielles abstraites de $\mathcal{L}(2,1)$ correspondent exactement aux automates d'arbres. Et comme l'appartenance d'un arbre à un langage régulier d'arbres se décide en temps polynômial, l'analyse des grammaires catégorielles abstraites de $\mathcal{L}(2,1)$ s'effectue également en temps polynômial.

De plus les termes de type atomique d'une signature d'ordre 2 sont les éléments d'un langage régulier d'arbres. Les lexiques qui relient de telles signatures en faisant correspondre à chaque type atomique un type atomique correspondent à des morphismes linéaires entre langage régulier d'arbres. Tout transducteur d'arbres peut se voir comme un bimorphisme et en particulier tout transducteur linéaire peut être vu comme un bimorphisme linéaire. Si deux signatures d'ordre 2 sont les signatures objets de deux grammaires catégorielles abstraites de $\mathcal{L}(2,1)$, alors les lexiques de ces deux grammaires forment un bimorphisme linéaire entre ces deux signatures. Il s'ensuit donc que ces deux grammaires catégorielles abstraites de $\mathcal{L}(2,1)$ ayant une même signature abstraite constituent un transducteur linéaire d'arbres.

2.3.3 Les grammaires hors contextes

Nous allons voir ici la façon dont les grammaires catégorielles abstraites permettent de représenter les grammaires hors contexte. Plus précisément, la classe des grammaires catégorielles abstraites de $\mathcal{L}(2,2)$ dont les signatures objets représentent des chaînes de caractères correspond exactement aux langages hors contexte.

Définition 2.12 *Une grammaire hors contexte est un quadruplet $H = (N, V, R, S)$ où N est un ensemble fini, les non-terminaux, V est un ensemble fini, l'alphabet, R est un ensemble fini de règles de réécriture de la forme $\beta \rightarrow_H \omega$ où $\beta \in N$ et w est un mot construit sur $N \cup V$ et S est un élément de N.*

On dit qu'un mot ω construit sur V fait partie du langage de H si $S \overset{*}{\rightarrow}_H \omega$.

Étant donnée une grammaire hors contexte $H = (N, V, R, S)$, on construit une grammaire catégorielle abstraite $\mathcal{G}_H = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ dont le langage est identique à celui de H de la façon suivante :

1. $\mathcal{C}_{\Sigma_1} = R$, $\mathcal{A}_{\Sigma_1} = N$ et si $r = \beta \rightarrow_H \alpha_1 \dots \alpha_n$ et si $I = \{i | \alpha_i \in N\}$ alors $\tau_{\Sigma_1}(r) = \overline{\alpha_I} \multimap \beta$.
2. Σ_2 est la signature des chaînes sur V
3. pour tout $\alpha \in N$, $\mathcal{L}(\alpha) = * \multimap *$ et si $r = \beta \rightarrow_H \alpha_1 \dots \alpha_n$ et alors $\mathcal{L}(r) = \lambda \overline{x_I}.\alpha'_1 + \dots + \alpha'_n$ avec $I = \{i | \alpha_i \in N\}$ et $\alpha'_i = \begin{cases} \alpha_i \text{ si } \alpha_i \in V \\ x_i \text{ sinon} \end{cases}$

Si par exemple on prend la grammaire hors contexte suivante $H = (\{S\}, \{a, b\}, \{S \rightarrow_H aSb; S \rightarrow_H \epsilon\}, S)$, alors la grammaire catégorielle abstraite qui encode ce langage est :

1. $r_1 = \lambda x.a + x + b : S \multimap S$
2. $r_2 = \lambda z.z : S$

La réalisation du terme $r_1(\dots(r_1 r_2)\dots)$ est bien le terme $a^n + b^n$.

2.3.4 Les grammaires d'arbres adjoints

Les grammaires d'arbres adjoints sont un formalisme grammatical proposé par Joshi [AKJ75]. Ce formalisme est défini par :

Définition 2.13 *Une grammaire d'arbres adjoints est définie par un quintuplet (V_n, V_t, S, I, A) avec :*

1. *V_n est l'ensemble des non-terminaux*
2. *V_t est l'ensemble des terminaux*
3. *S est non terminal distingué, le non-terminal de départ*
4. *I est l'ensemble des arbres initiaux*
5. *A est l'ensemble des arbres d'adjonction*

Les arbres d'adjonctions sont des contextes d'arbres qui viennent s'insérer à la place d'un nœud d'adjonction dans un autre arbre de la grammaire. Ces grammaires utilisent également la substitution pour mener à bien une analyse, on substitue un arbre initial à une feuille de substitution portant la même catégorie. Les grammaires d'arbres adjoints permettent de construire des arbres en utilisant les opérations d'adjonction et de substitution. Un tel arbre est considéré comme l'analyse d'une phrase si les feuilles de cet arbre, prises dans leur ordre de gauche à droite forment cette phrase. Nous considérons le cas où l'adjonction ne peut s'effectuer qu'au plus une fois par nœud. Pour un codage des grammaires d'arbres adjoints qui tienne compte des diverses modalités supplémentaires du formalisme, on pourra se référer à l'article [dG02].

Nous allons encoder une grammaire d'arbres adjoints (V_n, V_t, S, I, A) avec une grammaire catégorielle abstraite $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \tilde{S})$ de $\mathcal{L}(2,3)$ définie par

1. pour tout non-terminal β de V_n, il existe deux types atomiques β et $\tilde{\beta}$ dans $\mathcal{C}_{\Sigma_1}$. Ces deux types seront respectivement les représentations abstraites des nœuds d'adjonction et des feuilles de substitution.

2. pour tout arbre $t \in I \cup A$, il existe une constante abstraite $\tilde{t} \in \mathcal{C}_{\Sigma_1}$ et si t contient des nœuds d'adjonction de type $\beta_1, \ldots, \beta_n$ et des feuilles de substitution de type $\gamma_1, \ldots, \gamma_m$ alors :
   (a) si t est un arbre initial de type β, alors
   $$\tau_{\Sigma_1}(\tilde{t}) = \overline{\beta_{[1,n]}} \multimap \overline{\tilde{\gamma}_{[1,m]}} \multimap \tilde{\beta}$$
   (b) si t est un arbre d'adjonction de type β alors
   $$\tau_{\Sigma_1}(\tilde{t}) = \overline{\beta_{[1,n]}} \multimap \overline{\tilde{\gamma}_{[1,m]}} \multimap \beta$$

Finalement pour tout type abstrait β, on associe une constante abstraite c_β telle que $\tau_{\Sigma_1}(c_\beta) = \beta$.

Pour ce qui est de la signature objet Σ_2, elle est définie par :

1. $\mathcal{A}_{\Sigma_2} = \{*\}$
2. $\mathcal{C}_{\Sigma_2} = V_t$ et pour tout $a \in C_2$, $\tau_{\Sigma_2}(a) = * \multimap *$

Avant de donner la définition du lexique, nous allons donner une traduction, ϕ_{TAG}, des arbres d'une grammaire d'arbres adjoints sous forme de λ-termes de la signature objet :

1. $\phi_{TAG}(\beta(t_1, \ldots, t_n)) = x(\phi_{TAG}(t_1) + \ldots + \phi_{TAG}(t_n))$ où x est une nouvelle variable de type $(* \multimap *) \multimap (* \multimap *)$. On dit que x correspond au nœud d'adjonction β.
2. $\phi_{TAG}(\beta_{na}(t_1, \ldots, t_n)) = \phi_{TAG}(t_1) + \ldots + \phi_{TAG}(t_n)$.
3. $\phi_{TAG}(a) = a$
4. $\phi_{TAG}(\epsilon) = \lambda x.x$
5. $\phi_{TAG}(\beta) = x$ où β est un nœud de substitution et x est une nouvelle variable de type $* \multimap *$. On dit que x correspond à la feuille de substitution β.
6. $\phi_{TAG}(\beta^*) = s$ où β^* est le pied d'un arbre d'adjonction et s est une nouvelle variable de type $* \multimap *$. On dit que s correspond au pied d'adjonction β^*.

Pour compléter la définition il nous faut donner la définition du lexique :

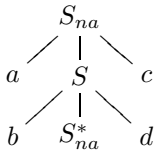
1. pour $\beta \in V_n$, $\mathcal{L}(\beta) = (* \multimap *) \multimap (* \multimap *)$ et $\mathcal{L}(\tilde{\beta}) = * \multimap *$
2. si t est un arbre initial dont les nœuds d'adjonction sont $\beta_1, \ldots \beta_n$ et les feuilles de substitution sont $\gamma_1, \ldots, \gamma_m$ si bien que
   $$\tau_1(\tilde{t}) = \overline{\beta_{[1,n]}} \multimap \overline{\tilde{\gamma}_{[1,m]}} \multimap \tilde{\beta}$$
   alors
   $$\mathcal{L}(\tilde{t}) = \lambda \overline{x_{[1,n]} y_{[1,m]}}.\phi_{TAG}(t)$$
   avec x_i la variable qui correspond au nœud d'adjonction β_i et y_j la variable qui correspond à la feuille de substitution γ_j dans la transformation $\phi_{TAG}(t)$.
3. si t est un arbre d'adjonction dont les nœuds d'adjonction sont $\beta_1, \ldots \beta_n$ et les feuilles de substitution sont $\gamma_1, \ldots, \gamma_m$ si bien que
   $$\tau_1(\tilde{t}) = \overline{\beta_{[1,n]}} \multimap \overline{\gamma_{[1,m]}} \multimap \beta$$
   alors
   $$\mathcal{L}(\tilde{t}) = \lambda \overline{x_{[1,n]} y_{[1,m]}} s.\phi_{TAG}(t)$$
   avec x_i la variable qui correspond au nœud d'adjonction β_i et y_j la variable qui correspond à la feuille de substitution γ_j et s au pied de t dans la transformation $\phi_{TAG}(t)$.
4. enfin si β est une catégorie d'adjonction, $\mathcal{L}(c_\beta) = \lambda xy.(xy)$.

Ce codage des grammaires d'arbres adjoints dans les grammaires catégorielles abstraites est correct et complet. Mais plutôt que de faire une longue preuve technique pour le prouver, nous allons montrer un exemple. Nous allons donner la grammaire catégorielle abstraite qui permet de coder la grammaire d'arbres adjoints dont le langage est $L = \{a^n b^n c^n d^n | n \in \mathbb{N}^*\}$.

Voyons tout d'abord la grammaire d'arbres adjoints qui permet de générer ce langage :

Arbre initial

Arbre d'adjonction



La grammaire catégorielle abstraite $(\Sigma_1, \Sigma_2, \mathcal{L}, \tilde{S})$ qui permet de coder cette grammaire d'arbres adjoints est définie par :

1. $\mathcal{A}_{\Sigma_1} = \{S; \tilde{S}\}$
2. $\mathcal{C}_{\Sigma_1} = \{e; f; c_S\}$
3. $\tau_{\Sigma_1}(e) = S \multimap \tilde{S}$, $\tau_1(f) = S \multimap S$, $\tau_{\Sigma_1}(c_S) = S$
4. $\mathcal{A}_{\Sigma_2} = \{*\}$
5. $\mathcal{C}_{\Sigma_2} = \{a; b; c; d\}$
6. $\tau_{\Sigma_2}(a) = \tau_{\Sigma_2}(b) = \tau_{\Sigma_2}(c) = \tau_{\Sigma_2}(d) = * \multimap *$
7. $\mathcal{L}(S) = (* \multimap *) \multimap (* \multimap *)$ et $\mathcal{L}(\tilde{S}) = * \multimap *$
8. $\mathcal{L}(e) = \lambda yx.y(\lambda x.x)x$, $\mathcal{L}(f) = \lambda ys.a + y(b + s + c) + d$ et $\mathcal{L}(c_S) = \lambda xy.(xy)$.

Dans cette grammaire, les constantes abstraites e et f représentent respectivement l'arbre initial et l'arbre d'adjonction. Quant à la constante c_B, elle permet d'éliminer une variable qui représente un nœud d'adjonction de catégorie S si on décide que ce nœud ne doit pas subir d'adjonction.

Pour se convaincre que ce langage code bel et bien la grammaire d'arbres adjoints donnée plus haut, il suffit de remarquer que tout terme clos du langage abstrait qui a pour type $\tilde{S}$ est de la forme $e(f^n(c_S))$ et que $\mathcal{L}(e(f^n(c_B))) = a^n + b^n + c^n + d^n$.

Ce codage des grammaires d'arbres adjoints dans les grammaires catégorielles abstraites utilise comme structure d'analyse les arbres de dérivation des grammaires d'arbres adjoints. Alors qu'habituellement le passage aux termes sémantiques dans les TAG s'effectue par l'intermédiaire de l'arbre dérivé, Sylvain Pogodalla [Pog04] a résolu élégamment des problèmes de sémantique réputés difficiles pour les grammaires d'arbres adjoints en utilisant les ACG et donc en construisant les termes sémantiques à partir de l'arbre de dérivation. Une telle approche démontre la pertinence de l'utilisation des grammaires catégorielles abstraites comme interface entre syntaxe et sémantique.

Chapitre 3

Algorithme général d'analyse

Ce chapitre présente brièvement un algorithme qui permet de déterminer si un terme u appartient au langage d'une grammaire catégorielle abstraite $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$. Bien que simple, cet algorithme est le point de départ de l'étude des grammaires catégorielles abstraites et des questions que nous allons chercher à résoudre dans la suite de cette thèse. Cet algorithme cherche à construire incrémentalement, du haut vers le bas, un terme abstrait t de type S tel que $\mathcal{L}(t) =_{\beta\eta} u$.

3.1 Algorithmes descendants

Dans ce qui suit, nous allons travailler dans le cadre d'une grammaire catégorielle abstraite quelconque $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$.

Définition 3.1 *On appelle* problème d'analyse *tout triplet* $\langle \Gamma; t : \alpha \rangle$ *où :*
- Γ *est un contexte d'assignation de variables sur* Σ_1
- t *est un* λ*-terme linéaire de* Σ_2 *sous forme* β*-normale et* η*-longue.*
- α *est une formule de* $\mathcal{I}_{\Sigma_1}$

Afin de mener à son terme une analyse, nous transformons les problèmes d'analyse en plusieurs autres problèmes d'analyse.

Définition 3.2 *Un problème d'analyse* e *se réduit à l'ensemble des problèmes d'analyse* $\{e_1; \ldots; e_n\}$*, ce que l'on note* $e \rightarrow_a \{e_1; \ldots; e_n\}$ *si l'on se peut utiliser l'une des règles de la figure 3.1.*

Définition 3.3 *Étant donné un ensemble de problèmes d'analyse* $\mathcal{S}$*, on dit qu'il se réduit à l'ensemble de problèmes* $\mathcal{S}'$*, ce que l'on note* $\mathcal{S} \rightarrow_{\mathcal{A}} \mathcal{S}'$*, si :*

$$e \in \mathcal{S},\ e \rightarrow_a \{e_1; \ldots; e_n\} \text{ et } \mathcal{S}' = (\mathcal{S} \backslash \{e\}) \cup \{e_1; \ldots; e_n\}$$

On note $\xrightarrow{p}_{\mathcal{A}}$ p *itérations de la relation* $\rightarrow_{\mathcal{A}}$ *et* $\xrightarrow{*}_{\mathcal{A}}$ *sa clôture réflexive et transitive.*

La relation $\rightarrow_{\mathcal{A}}$ définit un algorithme qui permet d'effectuer des analyses. En effet, $\{\langle \cdot; u : S \rangle\} \xrightarrow{*}_{\mathcal{A}} \emptyset$ si et seulement si il existe un terme abstrait t tel que $\mathcal{L}(t) =_{\beta\eta} u$.

3.1.1 Correction et complétude

Nous allons montrer que cet algorithme permet d'analyser la grammaire catégorielle abstraite $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$.

Proposition 3.1 *Si* $\mathcal{S} \xrightarrow{p}_{\mathcal{A}} \emptyset$ *alors pour tout* $\langle \Gamma_v; v : A_v \rangle \in \mathcal{S}$ *il existe un terme abstrait* t_v *de type* A_v *tel que* $\mathcal{L}(t_v) =_{\beta\eta} v$.

Élimination des abstractions :

$$\langle \Gamma; \lambda\overline{x_{[1,n]}}.t : \overline{\alpha_{[1,n]}} \multimap \beta\rangle \rightarrow_a \{\langle \Gamma, \overline{x_{[1,n]} : \alpha_{[1,n]}}; t : \beta\rangle\}$$

Utilisation de constante :

$$\langle \Gamma; t : \beta\rangle \rightarrow_a \{\langle \Gamma_1; t_1 : \alpha_1\rangle; \ldots; \langle \Gamma_n; t_n : \alpha_n\rangle\}$$

Cette règle n'est applicable que si il existe une constante a de la signature abstraite ayant pour type $\overline{\alpha_{[1,n]}} \multimap \beta$ telle que $\mathcal{L}(a)\overline{t_{[1,n]}} =_{\beta\eta} t$. Par ailleurs,on a $\Gamma = \Gamma_1, \ldots, \Gamma_n$ et pour tout $i \in \{1; \ldots; n\}$, $\overline{\Gamma_i} = \mathrm{FV}(t_i)$.

Élimination de variable :

$$\langle \Gamma; t : \beta\rangle \rightarrow_a \{\langle \Gamma_1; t_1 : \alpha_1\rangle; \ldots; \langle \Gamma_n; t_n : \alpha_n\rangle\}$$

Cette règle est applicable si $x : \overline{\alpha_{[1,n]}} \multimap \beta \in \Gamma$ et si $x\overline{t_{[1,n]}} =_{\beta\eta} t$. Enfin, on a $\Gamma = \Gamma_1, \ldots, \Gamma_n, x : \overline{\alpha_{[1,n]}} \multimap \beta$ et pour tout $i \in \{1; \ldots; n\}$, $\overline{\Gamma_i} = \mathrm{FV}(t_i)$.

N.B. : dans toutes ces règles on suppose que β est une formule atomique.

FIG. 3.1 – Réduction des problèmes d'analyse

Démonstration. Nous allons procéder par induction sur p. Si $p = 0$ alors $\mathcal{S} = \emptyset$ et on peut donc conclure. Sinon on a $e \in \mathcal{S}$, $e \rightarrow_a \mathcal{S}''$, $\mathcal{S}' = (\mathcal{S}\backslash\{e\}) \cup \mathcal{S}''$ et $\mathcal{S} \rightarrow_{\mathcal{A}} \mathcal{S}' \overset{p-1}{\rightarrow}_{\mathcal{A}} \emptyset$. L'hypothèse d'induction a pour conséquence que pour tout $\langle \Gamma_v; v : \alpha_v\rangle \in \mathcal{S}'$ il existe t_v de type α_v tel que $\mathcal{L}(t_v) =_{\beta\eta} v$. En supposant que $e = \langle \Gamma; u : \alpha\rangle$, pour montrer que cette propriété est vraie également pour $\mathcal{S}$, il nous suffit de montrer qu'il existe t_u de type α tel que $\mathcal{L}(t_u) =_{\beta\eta} u$. Pour le prouver, nous allons vérifier que cela est vrai quelle que soit la règle employée pour réduire e en $\mathcal{S}''$:

1. dans le cas de la règle d'*élimination des abstractions* $u = \lambda\overline{x_{[1,n]}}.u'$ et comme $S'' = \{\langle \Gamma, \overline{x_{[1,n]} : \alpha_{[1,n]}}; u' : \alpha\rangle\}$, il existe un terme t tel que $\mathcal{L}(t) = u'$. En posant $t_u = \lambda\overline{x_{[1,n]}}.t$, on obtient un terme qui vérifie les propriétés souhaitées.
2. dans celui de la règle d'*utilisation de constante* il existe a de type $\overline{\alpha_{[1,n]}} \multimap \beta$, u_1,... et u_n tels que $u = \mathcal{L}(a)\overline{u_{[1,n]}}$ et que pour tout $i \in \{1; \ldots; n\}$ $\langle \Gamma_i, u_i : \alpha_i\rangle \in \mathcal{S}''$. Donc pour tout $i \in [1, n]$ il existe t_i de type α_i tel que $\mathcal{L}(t_i) =_{\beta\eta} u_i$. En posant $t_u = a\overline{t_{[1,n]}}$, on obtient un terme de type β avec les bonnes propriétés.
3. si la règle est l'*élimination de variable* alors on obtient l'existence de t_u de la même manière que dans le cas précédent.

□

Proposition 3.2 *Si pour tout $\langle \Gamma; u : \alpha\rangle \in \mathcal{S}$ il existe un terme abstrait t de type α dans le contexte Γ et tel que $\mathcal{L}(t) =_{\beta\eta} u$ alors $\mathcal{S} \overset{*}{\rightarrow}_{\mathcal{A}} \emptyset$.*

Démonstration. Si $\mathcal{S} = \{\langle \Gamma_1; u_1 : \alpha_1\rangle; \ldots \langle \Gamma_n; u_n : \alpha_n\rangle\}$ on note $\mathcal{T}_{\mathcal{S}} = \{t_1; \ldots; t_n\}$ l'ensemble de termes abstraits tels que $\Gamma_i \vdash t_i : \alpha_i$ est dérivable et $\mathcal{L}(t_i) =_{\beta\eta} u_i$. Sans perte de généralité, on peut supposer que les éléments de $\mathcal{T}_S$ sont sous forme β-normale η-longue. Nous allons démontrer ce lemme par induction sur $|\mathcal{T}_{\mathcal{S}}|$ où :

$$|\mathcal{T}_{\mathcal{S}}| = \sum_{t \in \mathcal{T}_{\mathcal{S}}} |t|$$

D'après la définition même de $\mathcal{T}_{\mathcal{S}}$, il vient que si $|\mathcal{T}_{\mathcal{S}}| = 0$ alors $\mathcal{S} = \emptyset$ et la conclusion est triviale.

Si en revanche $|\mathcal{T}_{\mathcal{S}}| \neq 0$, alors il existe $\langle \Gamma; u : \alpha\rangle$ appartenant à $\mathcal{S}$. Par hypothèse il existe t un terme abstrait de type α et tel que $\mathcal{L}(t) =_{\beta\eta} u$. On distingue plusieurs cas suivant la structure syntaxique de t :

Cas 1: $t = c$

dans ce cas, en utilisant la règle d'*utilisation d'une constante*, on obtient que $\mathcal{S} \rightarrow_{\mathcal{A}} \mathcal{S}'$ où $\mathcal{S}' = \mathcal{S} \backslash \{\langle \Gamma_u; u : \alpha \rangle\}$. On pose $\mathcal{T}_{\mathcal{S}'} = \mathcal{T} \backslash \{t\}$ et ainsi $|\mathcal{T}_{\mathcal{S}'}| < |\mathcal{T}_{\mathcal{S}}|$; on peut donc conclure en utilisant l'hypothèse d'induction.

Cas 2: $t = x$

ce cas se traite comme le cas précédent mais en utilisant la règle d'*élimination d'une variable.*

Cas 3: $t = \lambda \overline{x_{[1,n]}}.t'$

dans ce cas, comme t et u sont sous forme η-longue, $\alpha = \overline{\alpha_{[1,n]}} \multimap \beta$ et $u = \lambda \overline{x_{[1,n]}}.v$. La règle d'élimination des abstractions nous donne que $\langle \Gamma_u; u : \alpha_u \rangle \rightarrow_a S'$ avec $S' = \{\langle \Gamma_u, \overline{x_{[1,n]} : \alpha_{[1,n]}}; v : \beta \rangle\}$, $\mathcal{L}(t') =_{\beta\eta} v$ et $|t'| < |t|$. Ainsi, $S \rightarrow_{\mathcal{A}} S''$ où $S'' = (S \backslash \{\langle \Gamma_u, u, \alpha_u \rangle\}) \cup S'$. Si on pose $\mathcal{T}_{\mathcal{S}''} = (\mathcal{T}_{\mathcal{S}} \backslash \{t\}) \cup \{t'\}$ et $|\mathcal{T}_{\mathcal{S}''}| < |\mathcal{T}_{\mathcal{S}}|$. On peut conclure puisque par hypothèse d'induction $\mathcal{S}'' \xrightarrow{*}_{\mathcal{A}} \emptyset$.

Cas 4: $t = a\overline{t_{[1,n]}}$

dans ce cas, pour tout $i \in [1, n]$, on pose $u_i = \mathcal{L}(t_i)$ et on a donc que $\mathcal{L}(a)\overline{u_{[1,n]}} =_{\beta\eta} u$. Ainsi, $\langle \Gamma; u : \alpha \rangle \rightarrow_a S'$ avec $\mathcal{S}' = \{\langle \Gamma_1; u_1 : \alpha_1 \rangle; \ldots; \langle \Gamma_n; u_n : \alpha_n \rangle\}$ et donc $\mathcal{S} \rightarrow_{\mathcal{A}} \mathcal{S}''$ avec $S'' = (\mathcal{S} \backslash \{\langle \Gamma; u : \alpha \rangle\}) \cup \mathcal{S}'$. Si on pose $\mathcal{T}_{\mathcal{S}''} = (\mathcal{T}_{\mathcal{S}} \backslash \{t\}) \cup \{t_1; \ldots; t_n\}$, alors on a $|\mathcal{T}_{\mathcal{S}''}| < |\mathcal{T}_{\mathcal{S}}|$. Par hypothèse d'induction, on a $\mathcal{S}'' \xrightarrow{*}_{\mathcal{A}} \emptyset$ et donc $\mathcal{S} \xrightarrow{*}_{\mathcal{A}} \emptyset$.

Cas 5: $t_u = x\overline{u_{[1,n]}}q$

pour ce cas on procède de la même façon que pour le cas précédent. □

3.2 Conclusion

Ce premier algorithme, permet d'analyser les grammaires catégorielles abstraites. Il reste cependant de nombreux points que nous n'avons pas encore éclaircis. En particulier, l'opération d'*utilisation d'une constante* s'appuie sur le présupposé qu'étant donné une contante a et un terme u, il est possible de déterminer si il existe des termes t_1, ..., t_n tels que $\mathcal{L}(a)t_1 \ldots t_n =_{\beta\eta} u$. Cela revient à vérifier que l'équation de filtrage $\mathcal{L}(a)\mathbf{X}_1 \ldots \mathbf{X}_n \stackrel{?}{=} u$ possède une solution. Enfin, la terminaison de cet algorithme n'a rien d'évidente, nous verrons d'ailleurs que celle-ci est loin d'être acquise. De plus, dans les cas où cet algorithme termine, il reste à voir si il est efficace ou non.

Aussi dans un premier temps, allons-nous étudier en détail le problème de filtrage dans le λ-calcul linéaire. Puis nous chercherons à dégager des éléments concernant la décidabilité et la complexité de l'analyse des grammaires catégorielles abstraites. Enfin, nous verrons dans quelle mesure l'algorithme que nous venons de décrire est efficace pour effectuer l'analyse.

Chapitre 4

Filtrage dans le λ-calcul linéaire

Comme nous l'avons vu au chapitre précédent, l'analyse des grammaires catégorielles abstraites met en jeu des équations dans le λ-calcul linéaire. Ce chapitre a pour but de montrer que la résolution de ces équations est décidable et d'étudier sa complexité.

Plus précisément, les équations que nous allons étudier sont appelées *équations de filtrage* ; ces équations ne contiennent des inconnues que dans un seul des membres qui les compose. Nous avons étudié ces équations dans plusieurs calculs [PdG04a], mais nous ne présenterons dans ce chapitre que les résultats obtenus pour le λ-calcul linéaire simplement typé.

Par ailleurs, le filtrage et plus généralement l'unification (équations où les inconnues peuvent avoir des occurrences dans les deux membres) sont des problèmes étudiés pour de nombreuses formes du λ-calcul. La résolution d'équations syntaxiques a de nombreuses applications dans des domaines variés allant de la recherche de démonstration, à la programmation en passant par la fouille de documents. Dans le λ-calcul simplement typé, l'unification est décidable au premier ordre [Hue76], mais devient indécidable à partir du second ordre [Gol81]. Quant au filtrage dans ce calcul, la décidabilité a été démontrée pour le premier, le second [Hue76], le troisième [Dow94] et le quatrième ordre [Pad00]. Cependant, au sixième ordre [Loa03], le filtrage dans le λ-calcul (modulo β) est indécidable (modulo $\beta\eta$, le problème est toujours ouvert). Comparé à tous ces problèmes, le filtrage dans le λ-calcul linéaire est un problème de filtrage très contraint, il se rapproche plus du filtrage au niveau des contextes d'arbre que de celui dans le λ-calcul simplement typé. Cependant, il a le mérite de mieux dessiner les contours entre problèmes de filtrages polynômiaux et problèmes de filtrage que l'on ne sait pas résoudre en temps polynômial.

Nous allons en effet démontrer que de manière générale le filtrage dans le λ-calcul linéaire simplement typé est décidable et NP-complet, et que même dans des cas drastiquement simplifiés ce problème reste NP-complet. En poursuivant poursuivi l'étude commencée dans [PdG04a], nous avons obtenu que pour le λ-calcul associé à la logique linéaire intuitionniste additive et multiplicative, le filtrage demeurait décidable, mais surtout que ce problème était un problème de NP pourvu que le membre de droite de l'équation soit sous forme normale, condition qui semble assez naturelle et peu contraignante.

Quant au sujet qui nous occupe, les grammaires catégorielles abstraites, l'étude du filtrage a un impact profond sur la décidabilité et la complexité de leur analyse, mais également sur les méthodes que nous allons utiliser pour mettre en œuvre cette analyse.

4.1 λ-calcul linéaire simplement typé

Dans cette première partie, nous allons montrer que les équations de filtrage dans le λ-calcul linéaire simplement typé est décidable et plus précisément NP-complet. Ce résultat a été montré par Philippe de Groote dans [dG00a].

Mais au préalable, nous allons fixer quelques conventions qui nous permettrons de simplifier les notations. Les termes que nous étudierons seront construits sur une signature Σ donnée et, afin d'éviter d'avoir à écrire systématiquement les contextes, nous allons considérer qu'à chaque inconnue et à chaque variable on associe un type de telle sorte qu'il y a un nombre infini et dénombrable d'inconnues et de

variables pour chaque type $\mathcal{I}_\Sigma$. En utilisant cette convention, nous pouvons, même si un terme n'est pas clos, considérer qu'il a un type unique. Tout au cours de ce chapitre, nous écrirons $\rho(t)$ pour désigner $\rho(\alpha)$ si t est de type α.

Définition 4.1 *Une équation de filtrage est une paire de termes de type α notée $t \stackrel{?}{=} u$ et telle que u est pur. On dit d'une telle équation qu'elle a pour solution σ si le support de σ ne contient que des inconnues et si $t.\sigma =_\beta u$.*

Les résultats qui suivent s'étendent sans difficulté aux équations de filtrage modulo $\beta\eta$-conversion.

Afin de prouver que le problème du filtrage dans le λ-calcul linéaire simplement typé est décidable et même NP-complet, nous allons procéder en deux temps :

1. dans un premier temps nous allons borner le nombre de pas de réduction nécessaires à la normalisation d'un terme en fonction des types des redex qu'il contient ;
2. dans un second temps nous allons borner la taille d'un terme t en fonction de u et de n si on sait par ailleurs que $t \stackrel{n}{\rightarrow}_\beta u$

Comme les bornes que nous allons établir sont polynômiales, nous pourrons en déduire le résultat. Si nous insistons sur cette démarche, c'est qu'elle se transpose au problème du filtrage dans le λ-calcul associé à la logique linéaire sans exponentielles et qu'elle permet de montrer qu'il est dans NP.

Nous allons tout d'abord vérifier que l'on peut majorer le nombre de pas de réduction nécessaires à la normalisation d'un terme en fonction des types des redex qu'il contient. Pour cela on définit une norme pour quantifier la complexité d'une application.

Définition 4.2 *Étant donnés deux termes t_1 et t_2 tels que t_1 a pour type $\alpha \multimap \beta$ et t_2 a pour type α, on note $\varrho(t_1 t_2)$ la complexité de l'application de t_1 à t_2 qui est définie par :*

$$\varrho(t_1 t_2) = \begin{cases} \rho(\alpha \multimap \beta) \text{ si } t_1 = \lambda x.t' \\ 0 \text{ sinon} \end{cases}$$

La fonction $\varrho(.)$ nous permet de définir une quantité qui mesure le nombre de pas nécessaires pour réduire un terme.

Définition 4.3 *On note $\mu(t)$ la complexité de réduction du terme t définie inductivement par :*

1. $\mu(h) = 0$ *si h est un terme atomique*
2. $\mu(\lambda x.t) = \mu(t)$
3. $\mu(t_1 t_2) = \varrho(t_1 t_2) + \mu(t_1) + \mu(t_2)$

Un terme t sur lequel on applique une substitution σ voit sa norme μ croître (*i.e.* $\mu(t) \leq \mu(t.\sigma)$) du simple fait qu'une substitution peut créer de nouveaux redex. Mais cette croissance peut être majorée par le type des termes qui participent à la substitution et le nombre d'occurrence qu'ont dans t les inconnues ou les variables du domaine de σ. Les lemmes suivants vont nous donner une majoration précise de l'accroissement entre $\mu(t)$ et $\mu(t.\sigma)$. Une première application de ce lemme sera de nous permettre de borner le nombre de pas de réduction nécessaires à la normalisation d'un terme. Une seconde sera de majorer la taille d'un terme qui, ayant subi une substitution, doit se réduire à un autre (*i.e.* majorer la taille d'une solution d'une équation de filtrage et donc obtenir la décidabilité du problème de filtrage). Cette seconde application de ce lemme ne se fera bien entendu qu'une fois que nous aurons mené à bien la deuxième étape de notre démarche.

Lemme 4.1 *Soient t un λ-terme linéaire et σ une substitution on a alors*

$$\mu(t.\sigma) \leq \mu(t) + \sum_{x \in \mathscr{D}(\sigma) \cap FV(t)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X} \in \mathscr{D}(\sigma)} |t|_{\mathbf{X}} (\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))$$

Démonstration. On procède par induction sur la structure de t :

Cas 1: $t = \mathbf{Y}$

si $\mathbf{Y} \in \mathscr{D}(\sigma)$, alors $t.\sigma = \sigma(\mathbf{Y})$ et donc :

$$\begin{aligned}\mu(t.\sigma) &= \mu(\sigma(\mathbf{Y}) \\ &\leq \mu(t) + \underbrace{\sum_{x\in\mathscr{D}(\sigma)\cap FV(t)} \rho(x) + \mu(\sigma(x))}_{=0} + \underbrace{\sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |t|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))}_{=\rho(\mathbf{Y})+\mu(\sigma(\mathbf{Y})}\end{aligned}$$

Cas 2: $t = y$

ce cas se traite comme le cas précédent.

Cas 3: $t = \mathbf{Y}u$ et $\sigma(\mathbf{Y}) = \lambda y.v$

dans ce cas, on a :

$$\begin{aligned}\mu(t.\sigma) &= \mu((\lambda y.v)(u.\sigma)) \\ &= \rho(\mathbf{Y}) + \mu(\sigma(\mathbf{Y})) + \mu(u.\sigma) \\ &\leq \rho(\mathbf{Y}) + \mu(\sigma(\mathbf{Y})) + \mu(u) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(u)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |u|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X}))) \\ &= \mu(t) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(t)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |t|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))\end{aligned}$$

Cas 4: $t = \mathbf{Y}u$ et $\sigma(\mathbf{Y}) \neq \lambda y.v$

on a alors :

$$\begin{aligned}\mu(t.\sigma) &= \mu((\lambda y.v)(u.\sigma)) \\ &= \mu(\sigma(\mathbf{Y})) + \mu(u.\sigma) \\ &\leq \mu(\sigma(\mathbf{Y})) + \mu(u) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(u)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |u|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X}))) \\ &\leq \mu(t) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(t)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |t|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))\end{aligned}$$

Cas 5: $t = yu$

ce cas se traite comme les deux précédents.

Cas 6: $t = u_1u_2$ et $u_1 \neq \mathbf{Y}$ et $u_1 \neq y$

on suppose que u_1 n'est pas une abstraction (l'autre cas se démontre d'une manière similaire) dans ce cas on obtient le résultat comme suit :

$$\begin{aligned}\mu(t.\sigma) &= \mu((u_1u_2).\sigma) \\ &= \mu(u_1.\sigma) + \mu(u_2.\sigma) \\ &\leq \mu(u_1) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(u_1)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |u_1|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X}))) + \\ &\quad \mu(u_2) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(u_2)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |u_2|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X}))) \\ &= \mu(t) + \sum_{x\in\mathscr{D}(\sigma)\cap FV(t)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X}\in\mathscr{D}(\sigma)} |t|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))\end{aligned}$$

Cas 7: $t = \lambda x.u$

on a alors :

$$\begin{aligned}
\mu(t.\sigma) &= \mu((\lambda x.u).\sigma) \\
&= \mu(\lambda x.(u.\sigma)) \\
&= \mu(u.\sigma) \\
&\leq \mu(u) + \sum_{x \in \mathscr{D}(\sigma) \cap FV(u)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X} \in \mathscr{D}(\sigma)} |u|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X}))) \\
&= \mu(t) + \sum_{x \in \mathscr{D}(\sigma) \cap FV(t)} \rho(x) + \mu(\sigma(x)) + \sum_{\mathbf{X} \in \mathscr{D}(\sigma)} |t|_{\mathbf{X}}(\rho(\mathbf{X}) + \mu(\sigma(\mathbf{X})))
\end{aligned}$$

□

Le lemme suivant démontre que la norme $\mu(.)$ décroît strictement avec la β-réduction, ce qui fait de cette norme une majoration du nombre de pas de réduction nécessaires à la normalisation d'un terme.

Lemme 4.2 *Si $t_1 \to_\beta t_2$ alors $\mu(t_2) < \mu(t_1)$.*

Démonstration. Ce lemme se prouve par induction sur la structure syntaxique de t_1 :

Cas 1: $t_1 = \lambda x.v$

dans ce cas, $v \to_\beta v'$ et $t_2 = \lambda x.v'$ et :

$$\begin{aligned}
\mu(t_1) &= \mu(v) \\
&< \mu(v') \text{ par hypothèse d'induction} \\
&= \mu(t_2)
\end{aligned}$$

Cas 2: $t_1 = v_1 v_2$

ce cas se divise lui-même en trois sous-cas (on notera à chaque fois $\alpha \multimap \beta$ et α les types respectifs de v_1 et v_2) :

1. $v_1 = \lambda x.w_1$ et $t_2 = w_1[x := v_2]$ et alors $\mu(t_1) = \mu(v_1)+\mu(v_2)+\rho(\alpha \multimap \beta)$; mais $\mu(t_2) = \mu(v_1[x := v_2])$ et d'après le lemme 4.1 $\mu(v_1[x := v_2]) \leq \mu(v_1) + \mu(v_2) + \rho(\alpha)$ et donc $\mu(t_2) < \mu(t_1)$
2. $v_2 \to_\beta w_2$ et $t_2 = v_1 w_2$ alors :

$$\begin{aligned}
\mu(t_2) &= \mu(v_1) + \mu(w_2) + \rho(v_1 w_2) \\
&< \mu(v_1) + \mu(v_2) + \rho(v_1 w_2) \text{ par hypothèse d'induction} \\
&= \mu(v_1) + \mu(v_2) + \rho(v_1 v_2) \\
&= \mu(t_1)
\end{aligned}$$

3. $v_1 \to_\beta w_1$, v_1 n'est pas une λ-abstraction et $t_2 = w_1 v_2$. Il y a alors deux possibilités : ou bien w_1 est une λ-abstraction ou bien w_1 n'est pas une λ-abstraction. Dans le premier cas $v_1 = (\lambda xy.w_1')w_2'$, $w_1 = (\lambda y.w_1')[x := w_2']$, on suppose que $\lambda xy.w_1'$ a pour type $\beta_1 \multimap \beta_2$ et :

$$\begin{aligned}
\mu(t_1) &= \mu(v_1 v_2) \\
&= \mu(v_1) + \mu(v_2) \text{ (puisque } v_1 \text{ n'est pas une } \lambda\text{-abstraction)} \\
&= \mu((\lambda xy.w_1')w_2') + \mu(v_2) \\
&= \mu(\lambda xy.w_1') + \mu(w_2') + \rho(\beta_1 \multimap \beta_2) + \mu(v_2)
\end{aligned}$$

mais on a également que :

$$\begin{aligned}
\mu(t_2) &= \mu(((\lambda y.w'_1)[x := w'_2])v_2) \\
&= \mu((\lambda y.w'_1)[x := w'_2]) + \mu(v_2) + \rho(\beta_2) \\
&\leq \mu(\lambda y.w'_1) + \mu(w'_2) + \rho(w'_2) + \mu(v_2) + \rho(\beta_2) \text{ lemme 4.1} \\
&= \mu(\lambda xy.w'_1) + \mu(w'_2) + \rho(\beta_1) + \mu(v_2) + \rho(\beta_2) \\
&< \mu(\lambda xy.w'_1) + \mu(w'_2) + \rho(\beta_1 \multimap \beta_2) + \mu(v_2) \\
&= \mu(t_1)
\end{aligned}$$

Si w_1 n'est pas une λ-abstraction, alors on procède de la même façon que dans le cas qui précède.

□

Nous venons de borner par un polynôme le nombre de pas de réduction nécessaires à la normalisation d'un terme et par là nous achevons la première partie de notre étude du filtrage dans le λ-calcul linéaire simplement typé. Nous allons maintenant borner en fontion de n et de la taille de u la taille de tout terme t tel que $t \xrightarrow{n}_\beta u$. Nous allons commencer par voir comment la taille d'un terme varie sous l'effet d'une substitution.

Lemme 4.3 *Si t et u sont des λ-termes linéaires et si $x \in FV(t)$ alors $|t[x := u]| = |t| + |u| - 1$.*

Démonstration. On procède par induction sur la structure syntaxique de t :

Cas 1: $t = x$

alors $t[x := u] = u$ et donc $|t[x := u]| = |u|$, or $|t| + |u| - 1 = 1 + |u| - 1 = |u|$. On a donc bien l'égalité $|t[x := u]| = |t| + |u| - 1$

Cas 2: $t = t_1 t_2$

comme t est un λ-terme linéaire, soit x a une occurrence dans t_1 et n'en a pas dans t_2, soit x a une occurrence dans t_2 et pas dans t_1. Ces deux cas étant symétriques, nous allons nous contenter de traiter le cas où x a une occurrence dans t_1 et pas dans t_2. Dans ce cas

$$\begin{aligned}
|t[x := u]| &= |(t_1 t_2)[x := u]| \\
&= |t_1[x := u]| + |t_2[x := u]| \\
&= |t_1[x := u]| + |t_2| \\
&= |t_1| + |u| - 1 + |t_2| \text{ par hypothèse d'induction} \\
&= |t| + |u| - 1
\end{aligned}$$

Cas 3: $t = \lambda y.t'$

dans ce cas on a :

$$\begin{aligned}
|t[x := u]| &= |(\lambda y.t')[x := u]| \\
&= |\lambda y.(t'[x := u])| \\
&= |t'[x := u]| \\
&= |t'| + |u| - 1 \text{ par hypothèse d'induction} \\
&= |\lambda y.t'| + |u| - 1 \\
&= |t| + |u| - 1
\end{aligned}$$

□

Nous pouvons maintenant voir quelles sont les variations de taille d'un terme qui a subi une étape de β-réduction.

Lemme 4.4 *Si* $v_1 \rightarrow_\beta v_2$ *alors* $|v_1| = |v_2| + 2$

Démonstration. Ce lemme se démontre par induction sur la structure de v_1. À l'exception du cas où $v_1 = (\lambda x.t)u$ et $v_2 = t[x := u]$, tous les cas sont des conséquences directes de l'hypothèse d'induction. Dans ce cas particulier, la conclusion est une conséquence directe du lemme précédent. □

Lemme 4.5 *Si* $v_1 \stackrel{n}{\rightarrow}_\beta v_2$ *alors* $|v_1| = |v_2| + 2n$.

Démonstration. Le résultat de ce lemme se démontre en itérant le lemme précédent. □

Lemme 4.6 *Étant donnés deux λ-termes linéaires* t_1 *et* t_2 *tels que* $t_1 \stackrel{*}{\rightarrow}_\beta t_2$ *on a :*

$$|t_1| - |t_2| \leq 2(\mu(t_1) - \mu(t_2))$$

Démonstration. Il suffit, pour montrer ce lemme, de remarquer que si $t_1 \stackrel{n}{\rightarrow}_\beta t_2$ alors $\mu(t_2) \leq \mu(t_1) - 2n$ puisque, à chaque pas de réduction, la mesure $\mu(.)$ décroît strictement (lemme 4.2). Et le lemme 4.5 permet de conclure que si $t_1 \stackrel{n}{\rightarrow}_\beta t_2$ alors $|t_1| = |t_2| + 2n$. La conclusion suit et $|t_1| - |t_2| \leq 2(\mu(t_1) - \mu(t_2))$. □

Proposition 4.1 *Étant donné un problème de filtrage* $t \stackrel{?}{=} u$ *dont les inconnues sont* $\{\mathbf{X}_1; \ldots; \mathbf{X}_n\}$ *et qui admet une solution* σ *avec* $\sigma(\mathbf{X}_i) = t_i$ *et* t_i *sous forme normale alors :*

$$\sum_{i=1}^{n} |t_i| \leq |u| + \mu(t) + \sum_{i=1}^{n} |t|_{\mathbf{X}_i} \rho(\mathbf{X_i})$$

Démonstration. Comme on peut, d'après le lemme 4.2, normaliser tout terme en temps polynômial, sans perte de généralité, on peut supposer que u est un terme sous forme normale. Si σ est solution du problème de filtrage $t \stackrel{?}{=} u$ alors u est la forme normale de $t.\sigma$. D'après le lemme précédent, $|t.\sigma| - |u| \leq 2(\mu(t.\sigma) - \mu(u))$, mais comme u est sous forme normale, ce terme ne contient pas de redex et donc $\mu(u) = 0$ et $|t.\sigma| - |u| \leq \mu(t.\sigma)$. D'après le lemme 4.1, $\mu(t.\sigma) = \mu(t) + \sum_{i=1}^{n} |t|_{\mathbf{X}_i}(\mu(t_i) + \rho(t_i))$. Cependant, sous l'hypothèse que σ est une substitution en forme normale, $\mu(t_i) = 0$, de plus $\rho(t_i) = \rho(\mathbf{X}_i)$ et donc $\mu(t.\sigma) = \mu(t) + \sum_{i=1}^{n} |t|_{\mathbf{X}_i} \rho(\mathbf{X}_i)$ et :

$$\sum_{i=1}^{n} |t_i| \leq |t.\sigma| \leq |u| + \mu(t.\sigma) \leq |u| + \mu(t) + \sum_{i=1}^{n} |t|_{\mathbf{X}_i} \rho(\mathbf{X}_i)$$

□

La proposition précédente démontre que si il existe une solution, elle a une taille polynômiale dans les données du problème. De plus, nous avons vu que vérifier qu'une substitution est solution d'une équation de filtrage est polynômial. De ces deux constatations vient que le problème du filtrage dans le λ-calcul linéaire est un problème de NP.

Pour voir que le filtrage dans le λ-calcul linéaire simplement typé est NP-difficile, il suffit d'utiliser une réduction immédiate dans ce problème du problème de filtrage dans les chaînes de caractères dont Dana Angluin [Ang80] a démontré la NP-complétude. Mais nous allons utiliser une technique différente pour démontrer que le filtrage est NP-difficile. Cette technique nous permettra montrer que même dans des cas très simples, le filtrage est NP-complet.

4.2 Analyse détaillée de la complexité du filtrage

Dans le cadre des grammaires catégorielles abstraites, nous avons à résoudre des équations de filtrage bien particulières. Dans ces équations, les inconnues n'ont que des occurrences linéaires, *i.e.* si t est le membre gauche de l'une de ces équations alors $|t|_{\mathbf{X}} \leq 1$ pour toute inconnue $\mathbf{X}$. Nous avons montré dans

[PdG03] que la résolution des équations de filtrage dont les inconnues ont des occurrences linéaires et sont au plus du second ordre est NP-difficile.

La proximité entre ce problème de filtrage très restreint et le filtrage linéaire dans les contextes d'arbre laissait plutôt présager que la résolution de ces équations serait polynômiale. En effet les équations de filtrage linéaires de contextes d'arbre sont connues pour être solubles en temps polynômial [SSS01]. Les équations de filtrage dans le λ-calcul linéaire ont un espace de solution plus important que les équations de filtrage de contexte. En effet, considérée comme une équation de filtrage de contexte, l'équation :

$$\mathbf{X}\, a\, b = f\, b\, a$$

ne possède pas de solution, alors que considérée comme une équation de filtrage dans le λ-calcul linéaire, elle a pour solution la substitution $[\mathbf{X} := \lambda xy.f\, y\, x]$. La souplesse supplémentaire offerte par le λ-calcul linéaire rend le problème de filtrage NP-complet.

Afin de montrer ce résultat, nous utilisons un problème particulier introduit par Kilpeläinen [Kil92]. Il s'agit d'une version contrainte du problème de satisfaisabilité.

Définition 4.4 *Étant donné un ensemble fini de variables booléennes $\mathcal{A} = \{a_1; \ldots; a_n\}$, l'ensemble des* littéraux *construits sur $\mathcal{A}$ est :*

$$\mathcal{A} \cup \{\neg a_1; \ldots; \neg a_n\}$$

un littéral de la forme a est dit positif alors qu'un littéral de la forme $\neg a$ est dit négatif.

Définition 4.5 *Une clause sur un ensemble fini de variables booléennes $\mathcal{A}$ est un ensemble de littéraux construits sur $\mathcal{A}$.*

Définition 4.6 *Une valuation η est une fonction de $\mathcal{A}$, ensemble de variables booléennes, dans $\{0, 1\}$. Une valuation permet de satisfaire des littéraux, des clauses, ou des ensembles de clauses :*

1. *un littéral positif a est satisfait par η si et seulement si $\eta(a) = 1$*
2. *un littéral négatif $\neg a$ est satisfait par η si et seulement si $\eta(a) = 0$.*
3. *une clause C est satisfaite par η si et seulement si il existe un littéral l de C qui soit satisfait par η*
4. *un ensemble de clauses $\mathcal{C}$ est satisfait par η si toute clause de $\mathcal{C}$ est satisfaite par η*

Définition 4.7 *Un problème de statisfaisabilité sur un ensemble de clauses $\mathcal{C}$ consiste à déterminer si il existe une valuation η qui satisfasse $\mathcal{C}$.*

Théorème 4.1 *Le problème de satisfaisabilité est NP-complet [Coo71].*

On définit le problème 1-neg-sat comme il suit :

Définition 4.8 *Étant donné un ensemble de variables booléennes $\mathcal{A}$, et un ensemble de clauses $\mathcal{C}$ construites sur $\mathcal{A}$, un problème 1-neg-sat sur $\mathcal{C}$ est un problème de satisfaisabilité sur $\mathcal{C}$ tel que pour toute variable a de $\mathcal{A}$, il existe une et une seule clause C de $\mathcal{C}$ telle que $\neg a \in C$.*

La restriction apportée au problème général de satisfaisabilité consiste à n'autoriser qu'une et une seule occurrence d'un littéral négatif par variable booléenne intervenant dans le problème de satisfaisabilité considéré. La linéarité des occurrences des littéraux négatifs nous permet de faire le lien avec la linéarité du λ-calcul linéaire. Cette restriction ne rend cependant pas les problèmes 1-neg-sat plus simples à résoudre que les problèmes de satisfaisabilité :

Lemme 4.7 *Le problème 1-neg-sat est NP-complet.*

Démonstration. Comme 1-neg-sat est une restriction du problème de satisfaisabilité, il nous faut simplement montrer que ce problème est NP-difficile. Dans ce but nous allons réduire le problème de satisfaisabilité à 1-neg-sat.

Étant donné $\mathcal{C}$ un problème de satisfaisabilité défini à partir des variables booléennes $\mathcal{A} = \{a_1, \ldots, a_n\}$, nous définissons un ensemble $\mathcal{B} = \{b_1, \ldots, b_n\}$ où les b_i sont des variables booléennes distinctes de celles

de $\mathcal{A}$. On définit $\mathcal{D}$ l'ensemble de clauses $\bigcup_{i=1}^{n}\{\{a_i, b_i\}, \{\neg a_i, \neg b_i\}\}$. Pour tout $i \in [1, n]$, $\mathcal{D}$ définit un *ou exclusif* entre la variable a_i et la variable b_i, et ainsi, il est évident que si une valuation η satisfait $\mathcal{D}$ alors $\eta(a_i) = 1$ si et seulement si $\eta(b_i) = 0$.

Ainsi, pour toute valuation qui satisfait $\mathcal{D}$, b_i se comporte comme la négation de a_i. Il ne nous reste donc plus qu'à remplacer toutes les occurrences de $\neg a_i$ par b_i dans $\mathcal{C}$ pour obtenir $\mathcal{C}^*$. Le problème $\mathcal{C}^* \cup \mathcal{D}$ est, par construction, un problème de 1-neg-sat. Si une valuation η satisfait $\mathcal{C}^* \cup \mathcal{D}$ alors elle satisfait aussi $\mathcal{C}$. Et si il existe une valuation η qui satisfait $\mathcal{C}$, alors la valuation η', définie par $\eta'(a_i) = \eta(a_i)$ et $\eta'(b_i) = 1$ si et seulement si $\eta(a_i) = 0$, satisfait $\mathcal{C}^* \cup \mathcal{D}$.

Résoudre un problème de 1-neg-sat est NP-difficile et donc ce problème est NP-complet. □

Nous allons maintenant faire correspondre à tout problème de 1-neg-sat une équation de filtrage dans le λ-calcul linéaire dont les inconnues sont du second ordre et dans laquelle elles ont de plus des occurrences linéaires.

Étant donné un problème de 1-neg-sat un ensemble de clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ défini à partir d'un ensemble de variables booléennes $\mathcal{A} = \{a_1, \ldots, a_n\}$, on définit une fonction neg qui associe à chaque variable de a_i la clause de $\mathcal{C}$ qui contient $\neg a_i$:

$$\text{neg}(a_i) = C_j \text{ si et seulement si } \neg a_i \in C_j$$

De la même façon, on définit la fonction pos qui associe à chaque variable a_i l'ensemble des clauses de $\mathcal{C}$ qui contiennent le littéral a_i :

$$\text{pos}(a_i) = \{C_j \in \mathcal{C} | a_i \in C_j\}$$

Pour tout $i \in \{1, \ldots, n\}$, soit m_i le cardinal de $\text{pos}(a_i)$, et soit ψ_i une fonction telle que :

$$\text{pos}(a_i) = \{C_{\psi_i(1)}, \ldots, C_{\psi_i(m_i)}\}$$

Dans le cas où m_i est nul on convient de définir ψ_i par la fonction vide. Les fonctions ψ_i servent à désigner sans ambiguïté un élément de $\text{pos}(a_i)$.

Afin de définir l'équation de filtrage associée à $\mathcal{C}$, nous allons utiliser la signature $\Sigma = (C, \{\alpha\}, \tau)$ où C et τ sont donnés par :

1. $C = \{a; f; \overline{C_1}; \ldots; \overline{C_m}\}$
2. $\tau(a) = \alpha$
3. $\tau(f) = \alpha^n \multimap \alpha$
4. pour tout $i \in [1, m]$, $\tau(\overline{C_i}) = \alpha^m \multimap \alpha$

Pour construire l'équation de filtrage, nous allons également utiliser m^2 inconnues $\mathbf{X}_{11}, \ldots, \mathbf{X}_{1m}, \ldots, \mathbf{X}_{m1}, \ldots, \mathbf{X}_{mm}$ de type α et une inconnue $\mathbf{X}$ de type $\alpha^m \multimap \alpha$.

Pour $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$, nous définissons les termes suivants :

$$R_{ij} = \begin{cases} \overline{C_{\psi_i(j)}} a \ldots a \text{ si } j \leq m_i \\ a \text{ sinon} \end{cases}$$

Alors, pour tout $i \in \{1, \ldots, n\}$, nous définissons :

$$R_i = \overline{\text{neg}(a_i)} R_{i1} \ldots R_{im}$$

Finalement, l'équation de filtrage que nous considérons est $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ avec :

$$L_{\mathcal{C}} = \mathbf{X}(\overline{C_1}\mathbf{X}_{11} \ldots \mathbf{X}_{1m}) \ldots (\overline{C_m}\mathbf{X}_{m1} \ldots \mathbf{X}_{mm})$$

et

$$R_{\mathcal{C}} = f R_1 \ldots R_n$$

Par cette réduction, on cherche à faire correspondre à chacun des R_i la variable booléenne a_i. Si $a_i = 0$, alors la clause $\text{neg}(a_i)$ est satisfaite, sinon, c'est l'ensemble des clauses de $\text{pos}(a_i)$ qui est satisfait. Or

l'équation de filtrage consiste à aller chercher parmi les sous-termes qui composent $R_{\mathcal{C}}$, m sous-termes dont les têtes sont les constantes $\overline{C_i}$, mais avec la contrainte que ces sous-termes sont deux à deux indépendants. Pour trouver ces sous-termes il faut choisir parmi les R_i et les R_{ij}. Opter pour l'un des R_i empêche ensuite de choisir R_{ij} puisque R_{ij} est un sous-terme de R_i et le faire reviendrait à violer la contrainte imposée par l'équation ; réciproquement, si on choisit l'un des R_{ij} il devient pour les mêmes raisons impossible de choisir R_i. On retrouve là la dichotomie induite par les valeurs des variables booléennes : si on choisit un terme R_i on choisit un terme qui commence par la constante $\overline{\text{neg}(a_i)}$ (*i.e.* on affecte implicitement 0 à la variable a_i) et si on n'effectue pas ce choix, on peut choisir n'importe lequel de R_{ij} dont la tête est l'une des constantes associées aux clauses de $\text{pos}(a_i)$ (*i.e.* on affecte 1 à la variable a_i).

C'est sur cette base que nous allons démontrer que l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ possède une solution si et seulement si le problème 1-neg-sat sur l'ensemble de clauses $\mathcal{C}$ est satisfaisable. Puisque l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ a une taille polynômiale par rapport à celle de $\mathcal{C}$, nous pourrons conclure que résoudre des équations filtrage dont les inconnues sont du second ordre et ont des occurrences linéaires dans l'équation est NP-difficile.

Correction de la réduction

Nous allons tout d'abord démontrer que si $\mathcal{C}$ est satisfaisable alors l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ admet une solution. La satisfaisabilité de $\mathcal{C}$ donne l'existence d'une valuation η telle que pour tout $i \in [1, m]$, il existe j tel que ou bien $a_j \in \mathcal{C}_i$ et $\eta(a_j) = 1$, ou bien $\neg a_j \in \mathcal{C}_i$ et $\eta(a_j) = 0$. De cela on obtient l'existence d'une fonction ϕ qui à chaque clause de $\mathcal{C}$ associe un littéral satisfait par η. C'est une fonction de $[1, m]$ dans $[1, n]$ telle que :

$$\text{ou bien } \eta(a_{\phi(i)}) = 1 \text{ et } a_{\phi(i)} \in \mathcal{C}_i, \text{ ou bien } \eta(a_{\phi(i)}) = 0 \text{ et } \neg a_{\phi(i)} \in \mathcal{C}_i$$

Du fait que chaque littéral négatif apparaît dans une et une seule clause, si jamais deux clauses $\mathcal{C}_i$ et $\mathcal{C}_j$ sont satisfaites parce qu'elles contiennent toutes deux le littéral $\neg a_k$, et que $\eta(a_k) = 0$ alors $i = j$. Transposé dans nos notations cette propriété s'écrit :

$$\phi(i) = \phi(j) \text{ et } \eta(a_{\phi(i)}) = 0 \text{ implique } i = j$$

La fonction ϕ va nous servir de guide pour construire une solution à l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$. Suivant l'intuition que nous avons donnée, nous allons sélectionner le sous-terme R_j si $\phi(i) = j$ et $\eta(a_j) = 0$, et le sous-terme R_{jk} si $\phi(i) = j$, $\eta(a_j) = 1$ et $\psi_j(k) = i$.

Commençons par construire le terme qui sera substitué à $\mathbf{X}$; pour cela, on se dote d'un ensemble de variables de type α $\{x_1; \ldots; x_m\}$ et on définit la famille de termes $(S_i)_{i \in [1,n]}$:

$$S_i = \begin{cases} x_j \text{ si } \eta(a_i) = 0 \text{ et si il existe } j \text{ tel que } \phi(j) = i \\ \overline{\text{neg}(a_i)} S_{i1} \ldots S_{im} \text{ sinon} \end{cases}$$

avec, dans le deuxième cas, la famille $(S_{ij})_{i \in [1,n]\, j \in [1,m]}$ définie par :

$$S_{ij} = \begin{cases} x_j \text{ si } \eta(a_i) = 1, \text{ il existe } k \text{ tel que } \phi(k) = i \text{ et } \psi_i(j) = k \\ R_{ij} \text{ sinon} \end{cases}$$

finalement, le terme que l'on associera à $\mathbf{X}$ sera :

$$S = \lambda x_1 \ldots x_m . f S_1 \ldots S_n$$

Une première chose à vérifier pour que S puisse effectivement être employé pour construire la solution d'une équation de filtrage dans le λ-calcul linéaire, est que S est bel et bien un λ-terme linéaire. Les quelques lemmes qui vont suivre vont dans ce sens :

Lemme 4.8 *Pour tout $j \in [1, m]$, $x_j \in FV(S_{\phi(j)})$.*

Démonstration. On distingue deux cas en fonction de la valeur de $\eta(a_{\phi(j)})$:

Cas 1: $\eta(a_{\phi(j)}) = 0$

alors par définition, $S_{\phi(j)} = x_j$ et donc $x_j \in S_{\phi(j)}$

Cas 2: $\eta(a_{\phi(j)}) = 1$

alors, par définition de $\phi(j)$, $\mathcal{C}_j \in \text{pos}(a_{\phi(j)})$. Il existe donc k tel $\psi_{\phi(j)}(k) = j$ et il s'ensuit que, par définition de S_{ij}, $S_{\phi(j)k} = x_j$, et donc $x_j \in FV(S_{\phi(j)})$. Dans tous les cas on a $x_j \in FV(S_{\phi(j)})$. □

Lemme 4.9 *Pour tout $i \in [1,n]$ et tout $j \in [1,m]$, si $x_j \in FV(S_i)$ alors $\phi(j) = i$.*

Démonstration. À chaque fois que S_i ou S_{ij} est égal à x_k, on a comme condition que $\phi(k) = i$, la conclusion est donc immédiate. □

Lemme 4.10 *Pour tout $i \in [1,n]$, et pour tout $C_j \in pos(a_i)$, il existe un unique k tel que $\psi_i(k) = j$.*

Démonstration. Ce lemme est une conséquence immédiate de la définition de ψ_i. □

Lemme 4.11 *$S = \lambda x_1 \dots x_m.fS_1 \dots S_n$ est un λ-terme linéaire.*

Démonstration. Démontrer que S est un λ-terme linéaire revient à montrer que pour tout $j \in [1,m]$ il existe une unique occurrence de x_j dans $fS_1 \dots S_n$. Le lemme 4.8 permet de savoir que pour tout $j \in [1,m]$, x_j possède au moins une occurrence dans $fS_1 \dots S_n$. Le lemme 4.9 permet de dire que toutes les occurrences de x_j dans $fS_1 \dots S_n$ se trouvent dans $S_{\phi(j)}$. Il y a alors deux possibilités :

- ou bien $S_i = x_j$ et alors x_j a bel et bien une unique occurrence dans $fS_1 \dots S_n$
- ou bien $S_i = \overline{\text{neg}(a_i)}S_{i1} \dots S_{im}$ et il existe k tel que $S_{ik} = x_j$ et $\psi_i(k) = j$. Or d'après le lemme 4.10 ce k est unique et donc x_j a une unique occurrence dans S_i et donc dans $fS_1 \dots S_n$.

Dans tous les cas x_j a une unique occurrence dans $fS_1 \dots S_n$ et on peut conclure que $\lambda x_1 \dots x_m.fS_1 \dots S_n$ est bien un λ-terme linéaire. □

Maintenant que nous savons que S peut participer à la solution de $L_\mathcal{C} \stackrel{?}{=} R_\mathcal{C}$, nous allons compléter la solution en fournissant les termes que nous allons substituer aux inconnues $\mathbf{X}_{ij}$. Pour cela, on remarque que pour tout $i \in [1,m]$, ou bien il existe k tel que $S_k = x_i$, ou bien il existe k et l tels que $S_{kl} = x_i$. On définit donc la famille $(T_{ij})_{i\in[1,m]\, j\in[1,m]}$ de la manière suivante :

$$T_{ij} = \begin{cases} R_{kj} \text{ si il existe } k \text{ tel que } S_k = x_i \\ a \text{ si il existe } k \text{ et } l \text{ tels que } S_{kl} = x_i \end{cases}$$

Nous allons maintenant montrer que nous avons bien construit une solution pour $L_\mathcal{C} \stackrel{?}{=} R_\mathcal{C}$.

Proposition 4.2 *Soit $\mathcal{C}$ un problème 1-neg-sat et $L_\mathcal{C} \stackrel{?}{=} R_\mathcal{C}$ l'équation de filtrage qui lui est associé. Si $\mathcal{C}$ est satisfaisable, alors $L_\mathcal{C} \stackrel{?}{=} R_\mathcal{C}$ admet une solution.*

Démonstration. Comme $\mathcal{C}$ est satisfaisable, on peut construire les λ-termes linéaire S, T_{ij} comme nous l'avons décrit précédemment. Nous allons prouver que :

$$L_\mathcal{C}[\mathbf{X} := S][\mathbf{X}_{ij} := T_{ij}]_{i=1}^m{}_{j=1}^m \stackrel{*}{\to}_\beta R_\mathcal{C}$$

En effet, on a :

$$\begin{aligned} L_\mathcal{C}[\mathbf{X} := S][\mathbf{X}_{ij} := T_{ij}]_{i=1}^m{}_{j=1}^m &= S(\overline{C_1}T_{11} \dots T_{1m}) \dots (\overline{C_m}T_{m1} \dots T_{mm}) \\ &\stackrel{*}{\to}_\beta (fS_1 \dots S_n)[x_j := \overline{C_j}T_{j1} \dots T_{jm}]_{j=1}^m \end{aligned}$$

Il nous reste à montrer que pour tout $i \in [1,n]$ on a :

$$S_i[x_j := \overline{C_j}T_{j1} \dots T_{jm}]_{j=1}^m = R_i$$

Il y a deux possibilités à explorer :

1. $S_i = x_k$
   dans ce cas, $\eta(a_i) = 0$, $\phi(k) = i$ et donc $\text{neg}(a_i) = C_k$; de plus $T_{kl} = R_{il}$ ce qui nous conduit à :
$$\begin{aligned} S_i[x_j := \overline{C_j} T_{j1} \dots T_{jm}]_{j=1}^m &= x_k[x_j := \overline{C_j} T_{j1} \dots T_{jm}]_{j=1}^m \\ &= x_k[x_k := \overline{C_k} R_{i1} \dots R_{im}] \\ &= \overline{C_k} R_{i1} \dots R_{im} \\ &= \overline{\text{neg}(a_i)} R_{i1} \dots R_{im} \\ &= R_i \end{aligned}$$
2. $S_i = \overline{\text{neg}(a_i)} S_{i1} \dots S_{im}$
   pour démontrer que $S_i[x_j := \overline{C_j} T_{j1} \dots T_{jm}]_{j=1}^m = R_i$, il nous suffit de montrer que pour tout $j \in [1, m]$:
$$S_{ij}[x_j := \overline{C_j} T_{j1} \dots T_{jm}]_{j=1}^m = R_{ij}$$
   il y a alors deux possibilités :
   (a) ou bien $S_{ik} = x_l$ et par définition $l = \psi_i(k)$, $R_{ik} = \overline{C_{\psi_i(k)}} a \dots a$ et $T_{lj} = a$ pour tout $j \in [1, m]$. Par conséquent :
$$\begin{aligned} x_l[x_j := \overline{C_j} T_{j1} \dots T_{jm}]_{i=1}^m &= x_l[x_l := \overline{C_{\psi_i(k)}} a \dots a] \\ &= \overline{C_{\psi_i(k)}} a \dots a \\ &= R_{ik} \end{aligned}$$
   (b) ou bien $S_{ik} = R_{ik}$ et donc on a bien le résultat voulu.

□

Complétude de la réduction

Nous allons maintenant démontrer la propriété réciproque, à savoir que si il existe une solution à l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ alors $\mathcal{C}$ est satisfaisable. Il s'agit dans un premier temps de décrire la forme générale des solutions de $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ pour pouvoir ensuite extrapoler une valuation qui puisse satisfaire $\mathcal{C}$.

Lemme 4.12 *Si l'équation $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ admet une solution σ alors σ substitue à $\mathbf{X}$ un terme de la forme :*
$$\lambda x_1 \dots x_m . f U_1 \dots U_n$$
où les U_i vérifient :
1. *ou $U_i = x_k$ avec $k \in [1, m]$ et dans ce cas, $neg(a_i) = \mathcal{C}_k$,*
2. *ou alors $U_i = \overline{neg(a_i)} U_{i1} \dots U_{im}$ et les U_{ij} vérifient :*
   (a) ou bien $U_{ij} = x_k$ avec $k \in [1, m]$ auquel cas $\mathcal{C}_k \in pos(a_i)$,
   (b) ou alors $U_{ij} = R_{ij}$.

Démonstration. Supposons que σ soit définie par :
$$\begin{cases} \sigma(\mathbf{X}) &= U \\ \sigma(\mathbf{X}_{ij}) &= V_{ij} \end{cases}$$
Nous avons alors que :
$$U(\overline{C_1} V_{11} \dots V_{1m}) \dots (\overline{C_m} V_{m1} \dots V_{mm}) \stackrel{*}{\rightarrow}_\beta f R_1 \dots R_n$$
et donc U est forcément de la forme :
$$\lambda x_1 \dots x_m . f U_1 \dots U_n$$
Il s'ensuit que pour tout $i \in [1, n]$:
$$U_i[x_j := \overline{C_j} V_{j1} \dots V_{jm}]_{j=1}^m \stackrel{*}{\rightarrow}_\beta R_i$$
De là découlent deux cas pour chacun des U_i :

1. $U_i = x_k$
   dans ce cas, on doit avoir :
$$\begin{aligned} x_k[x_j := \overline{C_j} V_{j1} \dots V_{jm}]_{i=1}^{m} &= R_i \\ \overline{C_k} V_{k1} \dots V_{km} &= R_i \end{aligned}$$
   et donc $\text{neg}(a_i) = C_k$.
2. $U_i = \overline{C_k} U_{i1} \dots U_{im}$
   dans ce cas, $\overline{C_k}$ doit être égal à $\text{neg}(a_i)$ et de plus pour tout $k \in [1, m]$:
$$U_{ik}[x_j := \overline{C_j} V_{j1} \dots V_{jm}] \xrightarrow{*}_{\beta} R_{ik}$$
   et si $U_{ik} = x_l$ alors on a :
$$\overline{C_l} V_{l1} \dots V_{lm} = R_{ik}$$
   ce qui implique que $\overline{C_l} \in \text{pos}(a_i)$. Sinon, on doit avoir :
$$U_{ik} = R_{ik}$$

□

La forme d'une solution de $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$, nous permet de repérer quels sous-termes ont été sélectionnés. Leur position nous indique une valuation qui satisfait $\mathcal{C}$ suivant le principe que nous avons vu plus haut.

Proposition 4.3 *Étant donné un problème 1-neg-sat $\mathcal{C}$ et l'équation de filtrage qui lui est associée $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$, si $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ possède une solution, alors $\mathcal{C}$ est satisfaisable.*

Démonstration. D'après le lemme 4.12 si $L_{\mathcal{C}} \stackrel{?}{=} R_{\mathcal{C}}$ admet une solution σ, alors le terme U que σ substitue à $\mathbf{X}$ est de la forme :
$$\lambda x_1 \dots x_m . f U_1 \dots U_n$$
avec :

1. ou bien $U_i = x_k$ pour $k \in [1, m]$
2. ou alors $U_i = \overline{\text{neg}(a_i)} U_{i1} \dots U_{im}$

On définit alors une valuation η de la manière suivante :
$$\eta(a_i) = \begin{cases} 0 \text{ si } U_i = x_j \text{ pour } j \in [1, m] \\ 1 \text{ sinon} \end{cases}$$

Étant donné une clause C_j :

1. ou bien il existe $i \in [1, n]$ tel que $U_i = x_j$, alors d'après le lemme 4.12, $\text{neg}(a_i) = C_j$ *i.e.* $a_i \in C_j$, et donc η vérifie C_j
2. ou alors, puisque U est un λ-terme linéaire, il doit exister $k \in [1, n]$ et $l \in [1, m]$ tels que $U_{kl} = x_j$. Selon le lemme 4.12 $C_j \in \text{pos}(a_k)$ et donc $a_k \in C_j$. Il vient que C_j est satisfaite par η.

La valuation η satisfait donc toutes les clauses de $\mathcal{C}$. □

De là provient le fait que le filtrage dans le λ-calcul linéaire au deuxième ordre et avec occurrence linéaire des inconnues est NP-complet. Ce résultat nous permet de dessiner avec précision les cas pour lesquels le problème de filtrage est polynômial et ceux pour lesquels il est NP-complet. En particulier, ce résultat de complexité tient à la condition qu'il y ait des inconnues dans les arguments des inconnues du second ordre. Si jamais ces arguments étaient purs, alors on serait confronté à une équation de la forme $\mathbf{X} t_1 \dots t_n \stackrel{?}{=} t$ et la résoudre revient à vérifier que l'union des multi-ensembles des sous-termes des t_i est

incluse dans le multi-ensemble des sous-termes de t. Aussi résoudre de telles équations (dites équations d'interpolation du second ordre) est-il polynômial.

Cependant, le fait que l'inconnue soit du second ordre est nécessaire pour assurer cette polynômialité. En effet, à partir du troisième ordre, la résolution des équations d'interpolation devient NP-complète [Yos05]. De plus l'article [Yos05] donne des compléments sur la complexité du filtrage avec d'autres restrictions.

Comme les équations de filtrage dans le λ-calcul linéaire simplement typé sont décidables, nous savons que l'algorithme général d'analyse peut être rendu effectif. Mais pour cela, il nous faut avoir des algorithmes de résolution des équations de filtrage dans le λ-calcul linéaire simplement typé.

Chapitre 5

Algorithmes de filtrage dans le λ-calcul linéaire

Maintenant que nous avons vu que le problème de filtrage dans le λ-calcul linéaire est décidable et NP-complet, il nous reste à construire des algorithmes pour résoudre ces équations. Ces algorithmes viendront compléter l'algorithme général d'analyse, ce qui donnera un algorithme effectif pour analyser les grammaires catégorielles abstraites.

Dans ce chapitre, nous proposons deux algorithmes de résolution d'équation de filtrage. Le premier est celui présenté dans l'article [dG00a], il dérive de l'algorithme de résolution d'équation d'unification d'ordre de Gérard Huet [Hue76]. Le second algorithme est plus spécifique ; il résout seulement les équations linéaires de filtrage, c'est-à-dire les équations dans lesquelles les inconnues n'ont que des occurrences linéaires. Il se base sur un système formel, les descriptions syntaxiques, que nous avons introduit pour permettre de résoudre efficacement des équations de filtrage en utilisant la technique de tabulation.

5.1 Algorithmes de résolution de problèmes de filtrage

Nous allons présenter ici un algorithme de résolution d'équations de filtrage dans le λ-calcul linéaire simplement typé. Cet algorithme est une adaptation de l'algorithme de résolution d'équations d'unification dans le λ-calcul de Huet [Hue76]. Il se présente comme un système de réécriture sur les paires constituées d'un ensemble d'équations et d'une substitution. Nous allons présenter cet algorithme de façon similaire à la façon dont nous avons présenté l'algorithme général d'analyse. Nous commençons par construire un système qui permet de transformer une équation de filtrage en une paire formée d'un ensemble d'équations et d'une substitution. Ce système nous permet ensuite de définir le système général pour les paires constituées d'un ensemble d'équations et d'une substitution.

Dans cette partie, nous gardons les conventions que nous avons utilisées au chapitre précédent, c'est-à-dire que l'on considère que les variables et les inconnues sont implicitement typées et que pour chaque type, on possède un nombre infini et dénombrable de variables et d'inconnues.

Définition 5.1 *Étant donnée une équation de filtrage e, on dit que e se réduit à la paire (S, σ), ce que l'on note $e \rightarrow_{eq} (S, \sigma)$, où S est un ensemble d'équations de filtrage et σ est une substitution, si on peut utiliser une des règles de la figure 5.1 pour transformer e.*

Définition 5.2 *Étant donnés un ensemble d'équations S et une substitution σ, on dit que (S, σ) se réduit à (S', σ') si il existe $e \in S$ telle que $e \rightarrow_{EQ} (S_e, \sigma_e)$ et*

1. $\sigma' = \sigma_e \circ \sigma$
2. $S' = \{f | \text{il existe } f' \in (S \backslash \{e\}) \cup S_e \text{ telle que } f \text{ est la forme normale de } f'.\sigma_e\}$

Cet algorithme permet de résoudre aussi bien les équations de filtrage modulo $=_\beta$ ou modulo $=_{\beta\eta}$. Pour résoudre les équations modulo $=_{\beta\eta}$, il suffit de mettre les termes sous forme η-longue.

Règles de Simplification :

$at_1 \dots t_n \stackrel{?}{=} au_1 \dots u_n \rightarrow_{eq} (\{t_1 \stackrel{?}{=} u_1; \dots; t_n \stackrel{?}{=} u_n\}, Id)$

$\lambda x.t \stackrel{?}{=} \lambda x.u \rightarrow_{eq} (\{t = u\}, Id)$

Règles d'imitation :

$\mathbf{X}t_1 \dots t_n = au_1 \dots u_p \rightarrow_{eq} (\{\mathbf{X}_1\overline{t_{Q_1}} = u_1; \dots; \mathbf{X}_p\overline{t_{Q_p}} = u_p\}, [\mathbf{X} := \lambda x_1 \dots x_n.a(\mathbf{X}_1\overline{x_{Q_1}}) \dots (\mathbf{X}_p\overline{x_{Q_p}})])$
avec $(Q_i)_{i\in[1,p]}$ partition de $[1,n]$ et pour tout $i \in [1,p]$, $\mathbf{X}_i$ est une nouvelle inconnue.

$\mathbf{X}t_1 \dots t_n = \lambda x.u \rightarrow_{eq} (\{\mathbf{Y}xt_1 \dots t_n = u\}, [\mathbf{X} := \lambda x x_1 \dots x_n.\mathbf{Y}xx_1 \dots x_n])$
et $\mathbf{Y}$ est une nouvelle inconnue.

Règles de projection :

$\mathbf{X}t_1 \dots t_n = u \rightarrow_{eq} (\{t_k(\mathbf{Y}_1\overline{t_{Q_1}}) \dots (\mathbf{Y}_p\overline{t_{Q_p}}) = u\}, [\mathbf{X} := \lambda x_1 \dots x_n.x_k(\mathbf{Y}_1\overline{x_{Q_1}}) \dots (\mathbf{Y}_p\overline{x_{Q_p}})])$
où $(Q_i)_{i\in[1,p]}$ est une partition de $[1,n]\backslash\{k\}$ et pour tout $i \in [1,p]$, $\mathbf{Y}_i$ est une nouvelle inconnue.

FIG. 5.1 – Définition de la relation $\rightarrow_{eq}$

5.1.1 Correction

Lemme 5.1 *Si $e \rightarrow_{eq} (S_e, \sigma_e)$ et si σ est solution de chacune des équations de S_e alors $\sigma \circ \sigma_e$ est solution de e.*

Démonstration. Il suffit de démontrer cette propriété pour chacune des règles qui définissent la relation $\rightarrow_{eq}$. Il s'agit simplement d'effectuer des β-réductions pour obtenir à chaque fois le résultat. □

Lemme 5.2 *Si $(S, \sigma) \rightarrow_{EQ} (S', \sigma' \circ \sigma)$ et si τ est solution de l'ensemble des équations de S', alors, $\tau \circ \sigma'$ est solution des équations de S.*

Démonstration. Si $(S, \sigma) \rightarrow_{EQ} (S', \sigma' \circ \sigma)$ c'est qu'il existe $e \in S$ tel que $e \rightarrow_{eq} (S_e, \sigma')$. Il vient ensuite que S' est l'ensemble des équations sous forme normale de $\sigma'(S\backslash\{e\}) \cup S_e$. Soit e' une équation de S, ou bien $e' = e$, et dans ce cas, comme τ est solution des équations contenues dans S_e, le lemme précédent nous permet de conclure que $\tau \circ \sigma'$ est solution de e ; ou bien $e' \neq e$, alors $\sigma_e(e') \in \sigma'(S\backslash\{e\})$, τ est donc solution de $\sigma'(e')$ et $\tau \circ \sigma'$ est solution de e'. □

Lemme 5.3 *Si $(\{e\}, Id) \stackrel{*}{\rightarrow}_{EQ} (\emptyset, \sigma)$ alors σ est solution de l'équation e.*

Démonstration. Il s'agit simplement d'itérer le lemme précédent. □

5.1.2 Terminaison

Nous allons démontrer que l'algorithme défini par la relation $\rightarrow_{EQ}$ termine. Adjointe à la propriété de correction puis à celle de complétude que nous allons bientôt montrer, cette propriété nous donne une nouvelle démonstration de la décidabilité du filtrage dans le λ-calcul linéaire simplement typé.

Pour cela, nous avons besoin de définir quelques normes sur les termes, les équations de filtrage et les ensembles d'équations de filtrage.

Définition 5.3 *On note $\rho_{INC}(t)$ la somme cumulée des complexités des types des inconnues qui ont des occurrences dans t. Cette fonction est définie inductivement comme suit :*

1. *$\rho_{INC}(h) = 0$ si h est un terme atomique et n'est pas une inconnue.*

2. $\rho_{INC}(\mathbf{X}) = \rho(\mathbf{X})$
3. $\rho_{INC}(\lambda x.t) = \rho_{INC}(t)$
4. $\rho_{INC}(t_1 t_2) = \rho_{INC}(t_1) + \rho_{INC}(t_2)$

Définition 5.4 *On note* $||t \stackrel{?}{=} u||$ *la paire* $(|u|, \rho_{INC}(t))$. *Étant donné* S *un ensemble d'équations, on note* $||S||$ *la paire :*

$$(\sum_{t \stackrel{?}{=} u \in S} |u|, \sum_{t \stackrel{?}{=} u \in S} \rho_{INC}(t))$$

Définition 5.5 *On note* $(n_1, n_2) < (m_1, m_2)$ *la propriété suivante :*

$$n_1 < m_1 \text{ ou, } n_1 = m_1 \text{ et } n_2 < m_2$$

Théorème 5.1 *La relation* $<$ *sur les paires d'entiers est un ordre bien fondé.*

Lemme 5.4 *Si* $e \rightarrow_{eq} (S_e, \sigma_e)$ *alors* $||S_e|| < ||e||$ *et pour tout ensemble d'équations* S, $||\sigma_e(S \backslash \{e\})|| < ||S \backslash \{e\}||$.

Démonstration. Pour établir cette propriété, il suffit de la vérifier pour chacune des règles qui définissent la relation $\rightarrow_e q$. □

Ce lemme montre que si $S \rightarrow_{EQ} S'$ alors $||S|| < ||S'||$, il s'ensuit que l'algorithme défini par cette relation termine.

5.1.3 Complétude

Lemme 5.5 *Étant donnés deux termes sous forme normale* t *et* u, *et une substitution* σ, *si* $t.\sigma =_\beta u$ *alors il existe* S, σ' *et* σ'' *tels que :*

1. $t \stackrel{?}{=} u \rightarrow_{eq} (S, \sigma')$
2. $\sigma' \circ \sigma'' = \sigma$
3. σ'' *est solution de* S

Démonstration.

Nous allons distinguer plusieurs cas suivant les structures syntaxiques de t :

Cas 1: $t = a t_1 \ldots t_n$

dans ce cas, on a $t.\sigma = a(t_1.\sigma) \ldots (t_n.\sigma) =_\beta u$ et donc u est de la forme $a u_1 \ldots u_n$. Ainsi $t = u \rightarrow_{eq} (\{t_1 = u_1; \ldots; t_n = u_n\}, Id)$ et il suffit donc de choisir $\sigma' = \sigma$.

Cas 2: $t = \lambda x.t'$

on procède de la même manière que dans le cas précédent.

Cas 3: $t = \mathbf{X} t_1 \ldots t_n$

alors il nous faut distinguer un ensemble de cas selon la forme syntaxique de $\sigma(\mathbf{X})$. Si par exemple $\sigma(\mathbf{X}) = \lambda x_1 \ldots x_n x.w$, alors c'est que $u = \lambda x.u'$, il est clair qu'en utilisant une imitation, on parvient à conclure. Il en va de même pour les cas où $\sigma(\mathbf{X} = \lambda x_1 \ldots x_n.a v_1 \ldots v_p$. Si $\sigma(\mathbf{X}) = \lambda x_1 \ldots x_n.x_k v_1 \ldots v_p$, il faut utiliser la règle d'imitation pour conclure. □

Lemme 5.6 *Étant donnés un ensemble d'équations* S *et deux substitutions* τ *et* σ, *si* σ *est solution de* S *alors il existe* S', σ' *et* σ'' *tels que :*

1. $(S, \tau) \rightarrow_{EQ} (S', \sigma')$
2. $\sigma' \circ \sigma'' = \sigma$
3. σ'' *est solution de* S'

Démonstration. Il s'agit d'une simple application du lemme précédent. □

Proposition 5.1 *Si σ est solution de l'équation de filtrage $t \stackrel{?}{=} u$ alors il existe σ' et σ'' tels que :*

1. $(\{t \stackrel{?}{=} u\}, Id) \stackrel{*}{\rightarrow}_{EQ} (\emptyset, \sigma'')$
2. $\sigma = \sigma' \circ \sigma''$

Démonstration. Il s'agit d'appliquer itérativement le lemme précédent tant que l'ensemble des équations n'est pas vide. Comme le système de réécriture $\rightarrow_{EQ}$ termine, il est nécessaire que l'ensemble des équations devienne vide à un moment. □

Le système de résolution d'équations de filtrage que nous proposons est très indéterministe, certains petits ajustements permettent de réduire cet indéterminisme. L'article [dG00a] propose certaines heuristiques. Il propose notamment d'utiliser préférentiellement les règles d'imitation et il définit un critère à l'aide des variables libres pour déterminer en partie la façon dont les partitions doivent être effectuées dans ce cas.

Nous allons compléter ces heuristiques pour le cas des équations de filtrage dans lesquelles les inconnues ont des occurrences linéaires. Pour cela nous donnons une condition que ces équations de filtrage doivent vérifier pour avoir une solution. Cette condition peut se vérifier en temps polynômial et permet ainsi de réduire l'espace de recherche.

5.1.4 Condition nécessaire à l'existence de solution pour les équations de filtrage linéaires dans le λ-calcul simplement typé

Dans le cadre des grammaires catégorielles abstraites, nous nous intéressons plus particulièrement aux équations dans le λ-calcul linéaire simplement typé et dont les inconnues ont des occurrences linéaires. Par ailleurs, nous cherchons à résoudre ces équations modulo β et η, ce qui nous conduit à n'utiliser que des termes sous forme η-longue. Nous définissons une relation $\succcurlyeq$ entre les termes t et u ; pourvu que t soit un terme dans lequel les inconnues ont des occurrences linéaires, la condition que t soit en relation avec u est nécessaire au fait que l'équation $t \stackrel{?}{=} u$ possède une solution. Si jamais t et u n'étaient pas mis en relation par $\succcurlyeq$ alors l'équation $t \stackrel{?}{=} u$ ne posséderait pas de solution.

Tout d'abord, il est assez évident, compte tenu de la linéarité du calcul, que le multi-ensemble des constantes ou des variables contenues dans le membre de gauche d'une équation de filtrage doit être contenu dans celui du membre droit. Il reste que cela est un critère relativement pauvre pour déterminer si une équation n'a pas de solution. On voudrait par exemple pouvoir montrer simplement que l'équation $\mathbf{X}(\lambda x.b(a(x))) \stackrel{?}{=} \lambda x.a(b(x))$ ne possède pas de solution. Et le seul critère fondé sur les occurrences de constantes n'est pas suffisant pour montrer que cette équation n'a pas de solution. La relation $\succcurlyeq$ a pour but d'enrichir ce critère pour tenir compte d'informations structurelles sur les termes. Cette relation est définie inductivement de la manière suivante :

1. $at_1 \dots t_n \succcurlyeq au_1 \dots u_n$ si pour tout $i \in [1,n]$, $t_i \succcurlyeq u_i$
2. $\lambda x.t \succcurlyeq \lambda x.u$ si $t \succcurlyeq u$
3. $\mathbf{X}t_1 \dots t_n \succcurlyeq u$ si $\bigcup_{i=1}^{n} At(t_i) \subseteq At(u)$ et si pour tout $i \in [1,n]$ si t_i est de type $\overline{\gamma^i_{[1,p_i]}} \multimap \gamma^i$ il existe $C_i[]$ et u_i tels que :
   (a) $u = C_i[u_i]$ et u_i de type γ^i
   (b) $t'_i \succcurlyeq u_i$ où t'_i est la forme normale de $t_i\mathbf{X}^i_1 \dots \mathbf{X}^i_{p_i}$ et où pour tout $j \in [1,p_i]$ $\mathbf{X}^i_j$ est une nouvelle inconnue.

Lemme 5.7 *Si l'équation de filtrage $t \stackrel{?}{=} u$ admet une solution alors $t \succcurlyeq u$.*

Démonstration. On procède par induction sur la structure syntaxique de t :

Cas 1: $t = at_1 \dots t_n$, ou $t = \lambda x.t'$

dans ce cas la conclusion provient directement de l'hypothèse d'induction.

Cas 2: $t = \mathbf{X}t_1 \dots t_n$

comme l'équation possède une solution, il existe une substitution σ telle que $(\mathbf{X}t_1 \dots t_n).\sigma =_{\beta\eta} u$. Si on utilise une nouvelle inconnue $\mathbf{Y}_k$ qui est hors du domaine de σ, alors le terme

$$(\mathbf{X}t_1 \dots t_{k-1}\mathbf{Y}_k t_{k+1} \dots t_n).\sigma$$

a pour forme normale le terme v_k et $v_k = C_k[\mathbf{Y}_k w_1^k \dots w_{p_k}^k]$. Si w_k est la forme normale de $(t_k.\sigma)w_1^k \dots w_{p_k}^k$, alors la forme normale de $v_k[\mathbf{Y}_k := t_k.\sigma]$ est $C_k[w_k]$ et on a donc $C_k[w_k] = u$. Si on note t'_k la forme normale de $t_k\mathbf{X}_1^k \dots \mathbf{X}_{p_k}^k$, il existe une solution à l'équation $t'_k \stackrel{?}{=} w_k$, cette solution est la substitution $\sigma \circ ([\mathbf{X}_i^k := w_i^k]_{i\in[1,p_k]})$. Ainsi par hypothèse d'induction on a $t'_k \succcurlyeq w_k$. Or cela est vrai pour tout $k \in [1, n]$ et donc on a bien que $t \succcurlyeq u$. □

Afin de se rendre compte que l'on peut vérifier en temps polynômial que les termes t et u d'une équation de filtrage linéaire dans le λ-calcul simplement typé vérifient cette relation, il suffit de programmer cet algorithme en utilisant un tableau dont les colonnes sont indexées par les sous-termes de u de type atomique et dont les lignes sont indexées par les sous-termes de t. On note dans chaque case quels sont les sous-termes de t, appliqués à de nouvelles inconnues dans les cas où cela est nécessaire, et de u qui sont mis en relation par $\succcurlyeq$. Pour remplir ce tableau on commence par les plus petits sous-termes de t et de u et on remonte. On peut ainsi vérifier que l'on peut voir si $t \succcurlyeq u$ en $\mathcal{O}(|t||u|^2)$.

Bien entendu, il est assez simple de trouver des cas pour lesquels $t \succcurlyeq u$ et l'équation $t \stackrel{?}{=} u$ ne possède pas de solution. Ainsi l'équation

$$\mathbf{X}(\lambda x.a(b(x)))(\lambda x.a(b(x))) \stackrel{?}{=} c((\lambda x.a(b(x)))(\lambda x.b(a(x))))$$

ne possède pas de solution alors que l'on peut vérifier que ces deux membres sont en relation par $\succcurlyeq$.

Ce critère nous sera utile pour réduire également l'indéterminisme de l'algorithme d'analyse incrémental que nous aborderons plus tard.

5.2 Calcul des descriptions syntaxiques

5.2.1 Présentation

Dans cette partie, nous allons introduire un système formel, le calcul des descriptions syntaxiques, qui nous permettra de définir un algorithme de résolution d'équations de filtrage efficace basé sur la technique de tabulation. En même temps, cet algorithme nous donnera une vision plus précise de la complexité du filtrage dans le λ-calcul linéaire. Il nous permettra de caractériser des ensembles de problèmes dont la résolution s'effectue en temps polynômial. Pour cela nous définissons une nouvelle notion, les descriptions syntaxiques.

Dans ce qui suit, nous travaillerons dans une signature $\Sigma = (\mathcal{C}, \mathcal{A}, \tau)$.

Définition 5.6 *L'ensemble des descriptions syntaxiques sur la signature Σ est donné par la grammaire suivante :*

$$\mathcal{D} ::= \{\mathcal{T}_\Sigma : \mathcal{I}_\mathcal{A}\} \mid \mathcal{D} \multimap \mathcal{D} \mid \forall \mathcal{V} : \mathcal{I}_\mathcal{A}.\mathcal{D}$$

Définition 5.7 *On appelle type sous-jacent d'une description d le type $\overline{d}$ défini inductivement par :*

1. $\overline{\{t : \alpha\}} = \alpha$
2. $\overline{d_1 \multimap d_2} = \overline{d_1} \multimap \overline{d_2}$
3. $\overline{\forall x : \alpha.d} = \overline{d}$

Définition 5.8 *Un contexte de descriptions est un multi-ensemble de descriptions. Étant donné un multi-ensemble d'indices S, nous noterons $\overline{d_S}$ le contexte tel que pour tout k, $\overline{d_S}(d_k) = S(k)$.*

Définition 5.9 *L'ensemble des variables libres, $FV(d)$, d'une description d est définit inductivement par :*

1. $FV(\{t : \alpha\}) = FV(t)$
2. $FV(d_1 \multimap d_2) = FV(d_1) \cup FV(d_2)$
3. $FV(\forall x : \alpha.d) = FV(d) \backslash \{x\}$

Notation 5.1 *Étant donné un multi-ensemble de descriptions $\Delta = d_1, \ldots, d_n$, nous noterons $FV(\Delta)$ l'ensemble de variables $\bigcup_{i=1}^{n} FV(d_i)$.*

Définition 5.10 *Un contexte d'assignation de descriptions à des variables est un ensemble de paires $x : d$ où x est une variable et d est une description. Étant donné un ensemble d'indices $S = i_1, \ldots, i_n$, nous noterons $\overline{x_S : d_S}$ le contexte $x_{i_1} : d_{i_1}, \ldots, x_{i_n} : d_{i_n}$.*

Définition 5.11 *Étant donné un contexte d'assignation de descriptions à des variables $\Gamma = \overline{x_{[1,n]} : d_{[1,n]}}$, on note $\overline{\Gamma}$ l'ensemble de variables $\{x_1; \ldots; x_n\}$ et $\widetilde{\Gamma}$ le contexte $d_1, \ldots, d_n$.*

Définition 5.12 *Étant donné un contexte $\Gamma = \overline{x_L : d_L}$ et un type α, nous noterons $\forall x : \alpha.\Gamma$ le contexte $\overline{x_L : d'_L}$ avec :*

$$d'_k = \begin{cases} \forall x : \alpha.d_k \text{ si } x \in FV(d_k) \\ d_k \text{ sinon} \end{cases}$$

Définition 5.13 *Étant donnés deux contextes d'assignation de descriptions à des variables $\Gamma_1 = x_1 : d_1, \ldots, x_n : d_n$ et $\Gamma_2 = y_1 : e_1, \ldots, y_p : e_p$, si $\overline{\Gamma_1} \cap \overline{\Gamma_2} = \emptyset$ alors on note Γ_1, Γ_2 le contexte $x_1 : d_1, \ldots, x_n : d_n, y_1 : e_1, \ldots, y_p : e_p$. À chaque fois que nous écrirons Γ_1, Γ_2, nous supposerons que $\overline{\Gamma_1} \cap \overline{\Gamma_2} = \emptyset$.*

Définition 5.14 *Nous noterons $\Gamma; \Gamma'; \Delta \vdash_D t : d$ les séquents du calcul des descriptions syntaxiques, où :*

1. *Γ et Γ' sont des contextes d'assignation de types à des variables*
2. *Δ est un contexte d'assignation de description à des variables.*
3. *$\overline{\Gamma} = FV(d) \backslash FV(t)$*
4. *$\overline{\Gamma'} = FV(d) \cap FV(t)$*
5. *$\overline{\Delta} = FV(t) \backslash FV(d)$*

On dit qu'un séquent $\Gamma; \Gamma'; \Delta \vdash_D t : d$ est dérivable si il vérifie ces propriétés et si il peut être dérivé avec les règles de la figure 5.2.

Notation 5.2 *Étant donné un contexte d'assignation de types (resp. descriptions) à des variables $\Gamma = x_1 : d_1, \ldots, x_n : d_n$ et un ensemble de variables V inclus dans $\{x_1, \ldots, x_n\}$, on note $\Gamma_{|V}$ le contexte :*

$$\{x : d | x : d \in \Gamma \text{ et } x \in V\}$$

Définition 5.15 *On associe à une description syntaxique d dans un contexte d'assignation de types à des variables Γ tel que $FV(d) = \overline{\Gamma}$ un ensemble de λ-termes $[\![d]\!]_\Gamma$ défini inductivement par :*

1. $[\![\{t : \alpha\}]\!]_\Gamma = \{v | \Gamma \vdash_D v : \alpha \text{ est dérivable et } v =_\beta t\}$
2. $[\![d_1 \multimap d_2]\!]_\Gamma = \{v | \forall w \in [\![d_1]\!]_{\Gamma_{|FV(d_1)}}. vw \in [\![d_2]\!]_{\Gamma_{|FV(d_2)}}\}$
3. $[\![\forall x : \alpha.d]\!]_\Gamma = \{v | \forall \Delta. \forall w. \overline{\Delta} \cap \overline{\Gamma} = \emptyset, \Delta \vdash_D w \text{ est dérivable et } v \in [\![d[x := w]]\!]_{\Delta,\Gamma}\}$

L'ensemble $[\![d]\!]$ est appelé sémantique de d.

Définition 5.16 *Étant donné un séquent $\Gamma; \Gamma'; \overline{x_{[1,n]} : d_{[1,n]}} \vdash_D t : d$, on dit que ce séquent est valide si pour toute famille de termes $(t_k)_{k \in [1,n]}$ telle pour tout $k \in [1, n]$, $t_k \in [\![d_k]\!]_{\Gamma_{|FV(d_k)}, \Gamma'_{|FV(d_k)}}$ alors $t[x_k := t_k]_{k=1}^n \in [\![d]\!]_{\Gamma,\Gamma'}$.*

Lemme 5.8 ***Correction*** *Si le séquent $\Gamma; \Gamma'; \Delta \vdash_D t : d$ est dérivable alors il est valide.*

Règles de construction des descriptions

$$\frac{\Gamma \vdash_D t : \alpha}{\Gamma \vdash_D \{t : \alpha\} : \text{Type}} \qquad \frac{\Gamma_1 \vdash_D d_1 : \text{Type} \quad \Gamma_1, \Gamma_2 \vdash_D d_2 : \text{Type}}{\Gamma_1, \Gamma_2 \vdash_D d_1 \multimap d_2 : \text{Type}}$$

$$\frac{\Gamma, x : \alpha \vdash_D d : \text{Type}}{\Gamma \vdash_D \forall x : \alpha . d : \text{Type}}$$

Règles de construction des termes

Règles d'égalité

$$\frac{\Gamma \vdash_\Sigma t : \alpha}{\cdot; \Gamma; \cdot \vdash_D t : \{t : \alpha\}} \text{ Ref} \qquad \frac{\Gamma_1; \Gamma_2, x : \alpha; \Delta \vdash_D t : \{u : \beta\}}{\Gamma_1; \Gamma_2; \forall x : \alpha . \Delta \vdash_D \lambda x.t : \{\lambda x.u : \alpha \multimap \beta\}} \lambda\text{-eq}$$

$$\frac{\Gamma_1; \Gamma_2; \Delta_1 \vdash_D t_1 : \{v_1 : \alpha \multimap \beta\} \quad \Gamma_3; \Gamma_4; \Delta_2 \vdash_D t_2 : \{v_2 : \alpha\}}{\Gamma_1, \Gamma_3; \Gamma_2, \Gamma_4; \Delta_1, \Delta_2 \vdash_D t_1 t_2 \{v_1 v_2 : \beta\}} \text{ App-eq}$$

Règles logiques

$$\frac{\Gamma \vdash_D d : \text{Type}}{\Gamma; \cdot; x : d \vdash_D x : d} \text{ Ax-desc}$$

$$\frac{\Gamma_1, x : \alpha; \Gamma_2; \Delta \vdash_D t : d}{\Gamma_1; \Gamma_2; \forall x : \alpha . \Delta \vdash_D t : \forall x : \alpha . d} \forall\text{-desc} \qquad \frac{\Gamma_1; \Gamma_2; \Delta, x : d_1 \vdash_D t : d_2}{\Gamma_1; \Gamma_2; \Delta \vdash_D \lambda x.t : d_1 \multimap d_2} \lambda\text{-desc}$$

$$\frac{\Gamma_1; \Gamma_2; \Delta_1 \vdash_D t_1 : d_1 \multimap d_2 \quad \Gamma_3; \Gamma_4; \Delta_2 \vdash_D t_2 : d_1}{\Gamma_1 \backslash \Gamma_4; \Gamma_2, \Gamma_4; \Delta_1, \Delta_2 \vdash_D t_1 t_2 : d_2} \text{ App-desc}$$

FIG. 5.2 – Calcul des descriptions syntaxiques

Démonstration. Ce lemme se démontre par une induction directe sur la dérivation de $\Gamma;\Gamma';\Delta \vdash_D t : d$. □

Ce lemme de correction donne une sémantique aux descriptions. À travers elle, les règles qui définissent le typage de λ-termes linéaires se comprennent mieux. Le système de typage des descriptions est un moyen abstrait d'exprimer des égalités modulo β-conversion entre des λ-termes du même type, en supposant l'existence de termes ayant certaines descriptions. Les termes dont l'existence est supposée sont les variables auxquelles sont assignées des descriptions par le contexte. Dériver le séquent $\Gamma;\Gamma';\Delta \vdash_D t : \{u : \alpha\}$ revient à montrer que si il existe des termes correspondant aux descriptions associées aux variables de Δ, alors en les substituant dans t, on obtient un terme β-équivalent à u. On peut dire que t et u sont égaux dans le contexte $\Gamma;\Gamma';\Delta$. Les règles d'égalité s'expliquent directement à travers cette métaphore. Si t_1 est égal à u_1 dans un certain contexte et que t_2 est égal à u_2 dans un autre contexte alors $t_1 t_2$ est égal à $u_1 u_2$ dans la réunion de ces contextes. Il en va de même pour l'abstraction et la règle de réflexivité.

Les règles logiques du calcul des descriptions correspondent à celles du λ-calcul linéaire simplement typé et sont sans réelles surprises. Le système présenté ici est une version simplifiée d'un système plus général. En particulier, nous avons supprimé la règle de quantification universelle à gauche et l'avons incorporée directement dans les règles ∀-desc et λ-eq. La raison pour laquelle nous avons procédé ainsi est que le système de la figure 5.2 est très proche des algorithmes pour lesquels nous avons défini les descriptions. À notre sens, il aurait été inutile d'entrer dans les détails du système plus général d'autant plus qu'il est assez facile de s'en faire une idée précise à partir de la figure 5.2.

Cependant, pour ne considérer que des égalités qui contiennent le plus d'information sur la façon dont un terme se réduit à un autre, nous introduisons la notion de description complète.

Définition 5.17 *On dit qu'une description d est sous forme β-normale η-longue si :*

1. $d = \{t : \alpha\}$ *et t est sous forme β-normale η-longue.*
2. $d = d_1 \multimap d_2$ *et d_1 et d_2 sont des descriptions sous forme β-normale η-longue.*
3. $d = \forall x : \alpha.e$ *et e est une description sous forme β-normale η-longue.*

Définition 5.18 *Étant donnée une description d, $\{u : \alpha\}$ est appelée la conclusion de d si :*

1. $d = \{u : \alpha\}$
2. $d = d_1 \multimap d_2$ *et $\{u : \alpha\}$ est la conclusion de d_2*
3. $d = \forall x : \alpha.e$ *et $\{u : \alpha\}$ est la conclusion de e.*

Définition 5.19 *On dit qu'une description d est compatible avec $\{t : \beta\}$ si :*

1. *si $d = \overline{d_P} \multimap \{u : \alpha\}$ et si $t = C[u]$.*
2. *si $d = \forall x : \gamma.e$ et si il existe une variable y telle que $e[x := y]$ est compatible avec $\{t : \beta\}$.*

Définition 5.20 *On dit qu'une description d est complète si elle est sous forme β-normale η-longue, si $d = \forall \overline{x_{[1,p]} : \alpha_{[1,p]}}.\overline{e_{[1,n]}} \multimap \{t : \alpha\}$, si pour tout $i \in [1,n]$ e_i est compatible avec $\{t : \alpha\}$, si α est un type atomique et si e_i est complète.*

Pour se servir des descriptions syntaxiques, il faut qu'elles puissent modéliser les égalités que l'on veut. Comme en général nous cherchons à résoudre les équations modulo $\beta\eta$-conversion, nous allons voir que les séquents dérivables qui utilisent des descriptions complètes et qui typent des termes sous forme η-longue sont stables par β-expansion. Cela est suffisant car nous avons vu que deux termes étaient égaux modulo $\beta\eta$ si et seulement si leurs formes η-longues étaient égales modulo β.

Lemme 5.9 *Étant donnés une liste d'indices S, un terme v sous forme β-normale et η-longue, une famille de variables $(x_S)_{k\in S}$, une famille de types $(\alpha_k)_{k\in S}$ et un type atomique β, si le séquent $\overline{x_S : \alpha_S} \vdash_D v : \beta$ est dérivable, alors il existe une famille de descriptions complètes $(d_j)_{j\in S}$ telles que pour tout $j \in S$, il existe une liste $C_j \subseteq S\backslash\{j\}$ telle que le séquent :*

$$\overline{x_{C_j} : \alpha_{C_j}}; x_j : \alpha_j; \cdot \vdash_D x_j \uparrow^{\eta} : d_j$$

est dérivable et que pour toute partition $\mathcal{P} = \{S_1; S_2; S_3\}$ de S, si on pose :

1. $\sigma_{\mathcal{P}} = [x_k := x'_k]_{k \in S_1}$
2. $t_{\mathcal{P}} = \lambda \overline{x_{S_3}}.v$
3. $d_{j,\mathcal{P}} = \forall \overline{x_{S_3 \cap C_j} : \alpha_{S_3 \cap C_j}}.d_j$
4. $d_{\mathcal{P}} = \forall \overline{x_{S_3} : \alpha_{S_3}}.\overline{d_{S_3}} \multimap \{v : \beta\}$
5. $\alpha_{\mathcal{P}} = \overline{\alpha_{S_3}} \multimap \beta$

les séquents suivants sont dérivables :

$$\mathcal{S}_1 = \overline{x_{S_1 \cup S_2} : \alpha_{S_1 \cup S_2}}; \cdot; x : d_{\mathcal{P}} \vdash_D x \uparrow^{\eta} : \{t_{\mathcal{P}} : \alpha_{\mathcal{P}}\}$$

$$\mathcal{S}_2 = \overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : d_{\mathcal{P}}.\sigma_{\mathcal{P}}$$

$$\mathcal{S}_3 = \overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : \{t_{\mathcal{P}}.\sigma_{\mathcal{P}} : \alpha_{\mathcal{P}}\}$$

Démonstration. Nous procédons par induction sur la structure de v ; comme v est de type atomique nous avons que $v = x_j v_1 \dots v_n$. Soit $\mathcal{P} = \{S_1; S_2; S_3\}$ une partition de S, alors pour tout $i \in [1, n]$ et $j \in [1, 3]$, on définit :

$$S^i_j = \{k | k \in S_j \wedge x_k \in FV(v_i)\}$$

et

$$S^i = \bigcup_{j=1}^{3} S^i_j$$

On remarque que $\{S^1; \dots; S^n\}$ est une partition de $S \backslash \{j\}$ et que pour $k \in [1, 3]$, $\{S^1_k; \dots; S^n_k\}$ est une partition de $S_k \backslash \{j\}$.

De plus x_j est de type α_j et v est sous forme η-longue on a forcément $\alpha_j = \overline{\gamma_{[1,n]}} \multimap \beta$. Enfin, pour les mêmes raisons, v_i est de type γ_i, $v_i = \lambda \overline{x_{S'_i}}.w_i$ et $\gamma_i = \overline{\alpha_{S'_i}} \multimap \beta_i$.

Nous avons ainsi que $\overline{x_{S^i \cup S'_i} : \alpha_{S^i \cup S'_i}} \vdash_D w_i : \beta_i$ et donc par hypothèse d'induction il existe une famille $(e_l)_{l \in S^i \cup S'_i}$ telle que pour tout $l \in S^i \cup S'_i$ il existe une liste $B_l \subseteq (S^i \cup S'_i) \backslash \{l\}$ telle que le séquent :

$$\overline{x_{B_l} : \alpha_{B_l}}; x_l : \alpha_l; \cdot \vdash_D x_l \uparrow^{\eta} : e_l$$

est dérivable et pour toute partition $\mathcal{P}_i = \{P^i_1; P^i_2; P^i_3\}$, si on pose :

1. $\sigma_{\mathcal{P}_i} = [x_k := x'_k]_{k \in P^i_1}$
2. $d_{l,\mathcal{P}_i} = \forall \overline{x_{P^i_3} : \alpha_{P^i_3}}.d_l$
3. $e_{\mathcal{P}_i} = \forall \overline{x_{P^i_3 \cap B_l} : \alpha_{P^i_3 \cap B_l}}.\overline{d_{P^i_3}} \multimap \{v.\sigma_{\mathcal{P}_i} : \beta_i\}$
4. $t_{\mathcal{P}_i} = \lambda \overline{x_{P^i_3}}.w_i$
5. $\gamma_{\mathcal{P}_i} = \overline{\alpha_{P^i_3}} \multimap \beta_i$

les séquents

$$\overline{x'_{P^i_1} : \alpha_{P^i_1}}; \overline{x_{P^i_2} : \alpha_{P^i_2}}; \overline{x_{P^i_1} : e_{P^i_1,\mathcal{P}_i}.\sigma_{\mathcal{P}_i}} \vdash_D t_{\mathcal{P}_i} : d_{\mathcal{P}_i}.\sigma_{\mathcal{P}_i}$$

et

$$\overline{x'_{P^i_1} : \alpha_{P^i_1}}; \overline{x_{P^i_2} : \alpha_{P^i_2}}; \overline{x_{P^i_1} : e_{P^i_1,\mathcal{P}_i}.\sigma_{\mathcal{P}_i}} \vdash_D t_{\mathcal{P}_i} : \{t_{\mathcal{P}_i}.\sigma_{\mathcal{P}_i} : \gamma_{\mathcal{P}_i}\}$$

et

$$\overline{x_{P^i_1 \cup P^i_2} : \alpha_{P^i_1 \cup P^i_2}}; \cdot; x_i : d_{\mathcal{P}_i} \vdash_D x_i \uparrow^{\eta} : \{t_{\mathcal{P}_i} : \gamma_{\mathcal{P}_i}\}$$

sont dérivables.

Si on pose

1. $\mathcal{P}^i_1 = \{S^i_4; S^i_2; S'_i\}$ (avec $S^i_4 = S^i_1 \cup S^i_3$)
2. $\mathcal{P}^i_2 = \{S^i_1; S^i_5; S'_i\}$ (avec $S^i_5 = S^i_2 \cup S^i_3$)

on obtient que :

1. $t_{\mathcal{P}_1^i} = \lambda\overline{x_{S_i'}}.w_i = v_i = t_{\mathcal{P}_2^i}$
2. $\gamma_{\mathcal{P}_1^i} = \gamma_i = \gamma_{\mathcal{P}_2^i}$
3. pour tout $l \in S^i \cup S_i'$ $e_{l,\mathcal{P}_1^i} = e_{l,\mathcal{P}_2^i}$, nous noterons d_l cette description.
4. $d_{\mathcal{P}_1^i} = d_{\mathcal{P}_2^i}$, nous noterons f_i cette description et f_i^1 et f_i^2 les descriptions $f_i.\sigma_{\mathcal{P}_2^i}$ et $f_i.\sigma_{\mathcal{P}_2^i}$

Aussi, d'après ce qui précède, en instanciant la partition $\mathcal{P}_i$ par $\mathcal{P}_1^i$ et $\mathcal{P}_2^i$, on obtient pour tout $l \in S^i \cup S_i'$ la dérivabilité de :

$$\overline{x'_{S_4^i} : \alpha_{S_4^i}}; \overline{x_{S_2^i} : \alpha_{S_2^i}}; \overline{x_{S_4^i} : d_{S_4^i}.\sigma_{\mathcal{P}_1^i}} \vdash_D v_i : f_i^1 \tag{5.1}$$

$$\overline{x'_{S_1^i} : \alpha_{S_1^i}}; \overline{x_{S_5^i} : \alpha_{S_5^i}}; \overline{x_{S_1^i} : d_{S_1^i}.\sigma_{\mathcal{P}_2^i}} \vdash_D v_i : f_i^2 \tag{5.2}$$

$$\overline{x'_{S_4^i} : \alpha_{S_4^i}}; \overline{x_{S_2^i} : \alpha_{S_2^i}}; \overline{x_{S_4^i} : d_{S_4^i}.\sigma_{\mathcal{P}_1^i}} \vdash_D v_i : \{v_i.\sigma_{\mathcal{P}_1^i} : \gamma_i\} \tag{5.3}$$

$$\overline{x'_{S_1^i} : \alpha_{S_1^i}}; \overline{x_{S_5^i} : \alpha_{S_5^i}}; \overline{x_{S_1^i} : d_{S_1^i}.\sigma_{\mathcal{P}_2^i}} \vdash_D v_i : \{v_i.\sigma_{\mathcal{P}_2^i} : \gamma_i\} \tag{5.4}$$

$$\overline{x_{S^i} : \alpha_{S^i}}; \cdot; z_i : f_i \vdash_D z_i \uparrow^\eta : \{v_i : \gamma_i\} \tag{5.5}$$

On remarque tout d'abord que, comme la famille $(S^k)_{k\in[1,n]}$ est une partition de $S\backslash\{j\}$, pour tout $l \in S\backslash\{j\}$, le séquent :

$$\overline{x_{B_l} : \alpha_{B_l}}; x_l : \alpha_l; \cdot \vdash_D x_l \uparrow^\eta : e_l$$

est dérivable. La dérivabilité de ce séquent implique celle du séquent :

$$\overline{x_{B_l \cap S_i'} : \alpha_{B_l \cap S_i'}}, \overline{x_{S_i'} : \alpha_{S_i'}}; x_l : \alpha_l; \cdot \vdash_D x_l \uparrow^\eta : e_l$$

à partir de laquelle, en utilisant la règle d'abstraction universelle, on obtient la dérivabilité du séquent :

$$\overline{x_{B_l \cap S_i'} : \alpha_{B_l \cap S_i'}}; x_l : \alpha_l; \cdot \vdash_D x_l \uparrow^\eta : d_l$$

Il nous suffit donc de poser $C_l = B_l \cap S_i'$ pour vérifier la première conclusion pour tout $l \in S\backslash\{j\}$.

Il nous reste à trouver d_j et C_j puis à vérifier les autres conclusions.

On pose $d_j = \overline{f_{[1,n]}} \multimap \{v : \beta\}$ et $C_j = S\backslash\{j\}$.

Dérivabilité du séquent $\overline{x_{C_j} : \alpha_{C_j}}; x_j : \alpha_j; \cdot \vdash_D x_j \uparrow^\eta : d_j$:

Le séquent $\cdot; x_j : \alpha_j; \cdot \vdash_D x_j : \{x_j : \alpha_j\}$ est dérivable, or $\alpha_j = \overline{\gamma_{[1,n]}} \multimap \beta$ et donc :

$$\cdot; x_j : \alpha_j; \cdot \vdash_D x_j : \{x_j : \overline{\gamma_{[1,n]}} \multimap \beta\}$$

est dérivable. Par ailleurs, nous avons vu (séquent (5.5)) que pour tout $i \in [1, n]$, le séquent :

$$\overline{x_{S^i} : \alpha_{S^i}}; \cdot; z_i : f_i \vdash_D z_i \uparrow^\eta : \{v_i : \gamma_i\}$$

était dérivable ; par conséquent, en utilisant n fois la règle d'application, on obtient que :

$$\overline{x_{S\backslash\{j\}} : \alpha_{S\backslash\{j\}}}; x_j : \alpha_j; \overline{z_{[1,n]} : f_{[1,n]}} \vdash_D x_j z_1 \uparrow^\eta \ldots z_n \uparrow^\eta : \{x_j v_1 \ldots v_n : \beta\}$$

est dérivable.

Finalement, en utilisant n fois la règle d'abstraction on a la dérivabilité de :

$$\overline{x_{S\backslash\{j\}} : \alpha_{S\backslash\{j\}}}; x_j : \alpha_j; \cdot \vdash_D \lambda\overline{z_{[1,n]}}.x_j \overline{z_{[1,n]} \uparrow^\eta} : \overline{f_{[1,n]}} \multimap \{v : \beta\}$$

Dérivabilité de $\mathcal{S}_1$:

On a bien évidemment la dérivabilité du séquent :

$$\overline{x_S : \alpha_S}; \cdot; x : \overline{d_{S_3}} \multimap \{v : \beta\} \vdash_D x : \overline{d_{S_3}} \multimap \{v : \beta\}$$

comme par ailleurs, pour tout $k \in S_3$, nous avons la dérivabilité du séquent :

$$\cdot; x_k : \alpha_k; \cdot \vdash_D x_k \uparrow^\eta : d_k$$

en utilisant autant de fois que nécessaire la règle d'application, on obtient la dérivabilité de :

$$\overline{x_{S_1 \cup S_2} : \alpha_{S_1 \cup S_2}}; \overline{x_{S_3} : \alpha_{S_3}}; x : \overline{d_{S_3}} \multimap \{v : \beta\} \vdash_D x \overline{x_{S_3} \uparrow^\eta} : \{v : \beta\}$$

et finalement l'utilisation de la règle d'abstraction sur les variables de S_3 nous donne le résultat :

$$\overline{x_{S_1 \cup S_2} : \alpha_{S_1 \cup S_2}}; \cdot; x : \forall \overline{x_{S_3} : \alpha_{S_3}}.\overline{d_{S_3}} \multimap \{v : \beta\} \vdash_D x \uparrow^\eta : \{\lambda \overline{x_{S_3}}.v : \alpha_\mathcal{P}\}$$

puisque $d_\mathcal{P} = \forall \overline{x_{S_3} : \alpha_{S_3}}.\overline{d_{S_3}} \multimap \{v : \beta\}$.

Dérivabilité des séquents $\mathcal{S}_2$ et $\mathcal{S}_3$:

Dans ce qui suit, nous noterons σ' la substitution $[x_k := x'_k]_{k \in S_1 \cup S_3}$.
Pour obtenir ces résultats, il nous faut distinguer trois cas suivant que x_j est dans S_1, S_2 ou S_3 :

x_j est dans S_1

Dérivabilité de $\mathcal{S}_2$:
Nous avons la dérivabilité du séquent :

$$\overline{x'_{S_1 \cup S_2 \cup S_3} : \alpha_{S_1 \cup S_2 \cup S_3}}; \cdot; x_j : \overline{f_{[1,n]}} \multimap \{v : \beta\}.\sigma' \vdash_D x_j : \overline{f_{[1,n]}} \multimap \{v : \beta\}.\sigma'$$

cela revient à avoir la dérivabilité du séquent :

$$\overline{x'_{S_1 \cup S_2 \cup S_3} : \alpha_{S_1 \cup S_2 \cup S_3}}; \cdot; x_j : d_j.\sigma' \vdash_D x_j : \overline{f^1_{[1,n]}} \multimap \{v.\sigma' : \beta\}$$

Par ailleurs, nous savons que pour tout $i \in [1, n]$, le séquent :

$$\overline{x'_{S_4^i} : \alpha_{S_4^i}}; \overline{x_{S_2^i} : \alpha_{S_2^i}}; \overline{x_{S_4^i} : d_{S_4^i}.\sigma_1^i} \vdash_D v_i : f_i^1$$

est dérivable.
De la sorte, en utilisant n fois la règle d'application, on obtient que :

$$\overline{x'_{S_1 \cup S_3} : \alpha_{S_1 \cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1 \setminus \{j\} \cup S_3} : d_{S_1 \setminus \{j\} \cup S_3}.\sigma'}, x_j : d_j \vdash_D x_j \overline{v_{[1,n]}} : \{v.\sigma' : \beta\}$$

est dérivable. Or ce séquent n'est autre que le séquent,

$$\overline{x'_{S_1 \cup S_3} : \alpha_{S_1 \cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1 \cup S_3} : d_{S_1 \cup S_3}.\sigma'} \vdash_D v : \{v.\sigma' : \beta\}$$

En appliquant la règle de λ-abstraction à la famille de variables $(x_k)_{k \in S_3}$ on a que :

$$\overline{x'_{S_1 \cup S_3} : \alpha_{S_1 \cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1}.\sigma'} \vdash_D t_\mathcal{P} : (\overline{d_{S_3}} \multimap \{v : \beta\}).\sigma'$$

est dérivable.
Finalement, en utilisant la règle de quantification universelle sur la famille de variables $(x'_k)_{k \in S_3}$ on conclut que :

$$\overline{x'_k : \alpha_k}^{k, S_1}; \overline{x_k : \alpha_k}^{k, S_2}; \overline{x_k : d_{k,\mathcal{P}}.\sigma_\mathcal{P}}^{k, S_1} \vdash_D t_\mathcal{P} : d_\mathcal{P}.\sigma_\mathcal{P}$$

est dérivable.
Dérivabilité de $\mathcal{S}_3$:

Nous avons la dérivabilité du séquent :

$$\overline{x'_{S_1} : \alpha_{S_1}}, \overline{x_{S_3} : \alpha_{S_3}}; \cdot; x_j : d_j.\sigma_{\mathcal{P}} \vdash_D x_j : d_j.\sigma_{\mathcal{P}}$$

ce qui revient à la dérivabilité du séquent :

$$\overline{x'_{S_1} : \alpha_{S_1}}, \overline{x_{S_3} : \alpha_{S_3}}; \cdot; x_j : d_j.\sigma_{\mathcal{P}} \vdash_D x_j : \overline{f^2_{[1,n]}} \multimap \{v.\sigma_{\mathcal{P}} : \beta\}$$

Or nous avons vu (séquent (5.2)) que pour tout $i \in [1, n]$ le séquent :

$$\overline{x'_{S_1^i} : \alpha_{S_1^i}}; \overline{x_{S_5^i} : \alpha_{S_5^i}}; \overline{x_{S_1^i} : d_{S_1^i}.\sigma_2^i} \vdash_D v_i : f_i^2$$

est dérivable.

On peut appliquer n fois la règle d'application et on obtient la dérivabilité de :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2 \cup S_3} : \alpha_{S_2 \cup S_3}}; \overline{x_{S_1 \setminus \{j\}} : d_{S_1 \setminus \{j\}}.\sigma_{\mathcal{P}}}, x_j : d_j.\sigma_{\mathcal{P}} \vdash_D x_j \overline{v_{[1,n]}} : \{v.\sigma_P : \beta\}$$

ce qui revient à avoir la dérivabilité de :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2 \cup S_3} : \alpha_{S_2 \cup S_3}}; \overline{x_{S_1} : d_{S_1}.\sigma_{\mathcal{P}}} \vdash_D v : \{v.\sigma_P : \beta\}$$

Finalement on conclut en effectuant les abstractions des variables de S_3 :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : \{t_{\mathcal{P}}.\sigma_P : \alpha_{\mathcal{P}}\}$$

x_j est dans S_2

Dérivabilité de $\mathcal{S}_2$:

Nous avons que $\cdot; x_j : \alpha_j; \cdot \vdash_D x_j : \{x_j : \overline{\gamma_{[1,n]}}^{\circ} \beta\}$ est dérivable.

De plus nous avons vu que pour tout $i \in [1, n]$, le séquent :

$$\overline{x'_{S_4^i} : \alpha_{S_4^i}}; \overline{x_{S_2^i} : \alpha_{S_2^i}}; \overline{x_{S_4^i} : d_{S_4^i}.\sigma_1^i} \vdash_D v_i : \{v_i.\sigma_1^i : \gamma_i\}$$

est dérivable.

En utilisant n fois la règle d'application, on obtient donc que :

$$\overline{x'_{S_1 \cup S_3} : \alpha_{S_1 \cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1 \cup S_3} : d_{S_1 \cup S_3}.\sigma'} \vdash_D v : \{v.\sigma' : \beta\}$$

En utilisant la règle d'abstraction sur la famille de variables $(x_k)_{k \in S_3}$ on obtient la dérivabilité de :

$$\overline{x'_{S_1 \cup S_3} : \alpha_{S_1 \cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1}.\sigma'} \vdash_D t_{\mathcal{P}} : (\overline{d_{S_3}} \multimap \{v : \beta\}).\sigma'$$

Finalement, on conclut en utilisant la règle de quantification universelle sur la famille de variables $(x'_k)_{k \in S_3}$:

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : d_{\mathcal{P}}.\sigma_{\mathcal{P}}$$

Dérivabilité de $\mathcal{S}_3$:

Nous avons que $\cdot; x_j : \alpha_j; \cdot \vdash_D x_j : \{x_j : \overline{\gamma_{[1,n]}} \multimap \beta\}$ est dérivable.

De plus nous avons vu que pour tout $i \in [1, n]$, le séquent :

$$\overline{x'_{S_1^i} : \alpha_{S_1^i}}; \overline{x_{S_5^i} : \alpha_{S_5^i}}; \overline{x_{S_1^i} : d_{S_1^i}.\sigma_2^i} \vdash_D v_i : \{v_i.\sigma_2^i : \gamma_i\}$$

est dérivable.

En utilisant n fois la règle d'application, on obtient donc que :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2 \cup S_3} : \alpha_{S_2 \cup S_3}}; \overline{x_{S_1} : d_{S_1}.\sigma_{\mathcal{P}}} \vdash_D v : \{v.\sigma_{\mathcal{P}} : \beta\}$$

En utilisant la règle d'abstraction sur la famille de variables $(x_k)_{k\in S_3}$ on obtient le résultat :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : \{t_{\mathcal{P}} : \alpha_{\mathcal{P}}\}$$

x_j est dans S_3

Dérivabilité de $\mathcal{S}_2$:
Nous avons la dérivabilité du séquent :

$$\overline{x'_{S_1\cup S_3} : \alpha_{S_1\cup S_3}}; \cdot; x_j : d_j.\sigma' \vdash_D x_j : \overline{f_{[1,n]}} \multimap \{v : \beta\}.\sigma'$$

cela revient à avoir la dérivabilité du séquent :

$$\overline{x'_{S_1\cup S_3} : \alpha_{S_1\cup S_3}}; \cdot; x_j : d_j \vdash_D x_j : \overline{f^1_{[1,n]}} \multimap \{v.\sigma' : \beta\}$$

Par ailleurs, nous savons que pour tout $i \in [1, n]$, le séquent :

$$\overline{x'_{S_4^i} : \alpha_{S_4^i}}; \overline{x_{S_2^i} : \alpha_{S_2^i}}; \overline{x_{S_4^i} : d_{S_4^i}.\sigma_1^i} \vdash_D v_i : f_i^1$$

est dérivable.
De la sorte, en utilisant n fois la règle d'application, on obtient que :

$$\overline{x'_{S_1\cup S_3} : \alpha_{S_1\cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1\cup S_3\setminus\{j\}} : d_{S_1\cup S_3\setminus\{j\}}.\sigma'}, x_j : d_j \vdash_D x_j \overline{v_{[1,n]}} : \{v.\sigma' : \beta\}$$

est dérivable. Ce qui revient à avoir la dérivabilité du séquent :

$$\overline{x'_{S_1\cup S_3} : \alpha_{S_1\cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1\cup S_3} : d_{S_1\cup S_3}.\sigma'} \vdash_D v : \{v.\sigma' : \beta\}$$

En appliquant la règle d'abstraction à la liste de variables $(x_k)_{k\in S_3}$ on a que :

$$\overline{x'_{S_1\cup S_3} : \alpha_{S_1\cup S_3}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1}.\sigma'} \vdash_D t_{\mathcal{P}} : (\overline{d_{S_3}} \multimap \{v : \beta\}).\sigma'$$

est dérivable.
Finalement, en utilisant la règle de quantification universelle sur la famille de variables $(x'_k)_{k\in S_3}$ on conclut que :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D t_{\mathcal{P}} : d_{\mathcal{P}}.\sigma_{\mathcal{P}}$$

est dérivable.
Dérivabilité de $\mathcal{S}_3$:
Nous avons que $\cdot; x_j : \alpha_j; \cdot \vdash_D x_j : \{x_j : \overline{\gamma_{[1,n]}} \multimap \beta\}$ est dérivable.
De plus nous avons vu que pour tout $i \in [1, n]$, le séquent :

$$\overline{x'_{S_1^i} : \alpha_{S_1^i}}; \overline{x_{S_5^i} : \alpha_{S_5^i}}; \overline{x_{S_1^i} : d_{S_1^i}.\sigma_2^i} \vdash_D v_i : \{v_i.\sigma_2^i : \gamma_i\}$$

est dérivable.
En utilisant n fois la règle d'application, on obtient donc que :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2\cup S_3} : \alpha_{S_2\cup S_3}}; \overline{x_{S_1} : d_{S_1}.\sigma_{\mathcal{P}}} \vdash_D v : \{v.\sigma_{\mathcal{P}} : \beta\}$$

En utilisant la règle d'abstraction sur la famille de variables $(x_k)_{k\in S_3}$ on obtient le résultat :

$$\overline{x'_{S_1} : \alpha_{S_1}}; \overline{x_{S_2} : \alpha_{S_2}}; \overline{x_{S_1} : d_{S_1,\mathcal{P}}.\sigma_{\mathcal{P}}} \vdash_D \lambda t_{\mathcal{P}} : \{t_{\mathcal{P}} : \alpha_{\mathcal{P}}\}$$

□

Corollaire 5.1 *Étant donné un terme v sous forme β-normale η-longue tel que $\Gamma \vdash_D v : \alpha$ est dérivable. Il existe une description complète d telle que les séquents suivants sont dérivables :*

$$\cdot; \Gamma; \cdot \vdash_D v : d$$

$$\Gamma; \cdot; x : d \vdash_D x \uparrow^\eta : \{v : \alpha\}$$

Démonstration. Si $\Gamma = \overline{x_{S_1} : \alpha_{S_1}}$, et si $v = \lambda \overline{x_{S_2}}.w$ et $\alpha = \overline{\alpha_{S_2}} \multimap \beta$ avec β atomique. Il s'ensuit que le séquent :

$$\overline{x_{S_1 \cup S_2} : \alpha_{S_1 \cup S_2}} \vdash w : \beta$$

est dérivable.

D'après le lemme précédent il existe une famille de descriptions complètes $(d_k)_{k \in S_1 \cup S_2}$ telle que pour tout $i \in S_1 \cup S_2$, il existe $C_i \subseteq (S_1 \cup S_2) \backslash \{i\}$ tel que :

$$\overline{x_{C_i} : \alpha_{C_i}}; x_i : \alpha_i; \cdot \vdash_D x_i \uparrow^\eta : d_i$$

est dérivable ; et que pour toute partition $\mathcal{P} = \{P_1, P_2, P_3\}$ de $S_1 \cup S_2$, si on pose :

1. $\sigma_\mathcal{P} = [x_k := x'_k]_{k \in P_1}$
2. $d_{j,\mathcal{P}} = \forall \overline{x_{P_3 \cap C_j} : \alpha_{P_3 \cap C_j}}.d_j$
3. $d_\mathcal{P} = \forall \overline{x_{P_3} : \alpha_{P_3}}.\overline{d_{P_3}}.\{w : \beta\}$
4. $t_\mathcal{P} = \lambda \overline{x_{P_3}}.w$
5. $\alpha_\mathcal{P} = \overline{\alpha_{P_3}} \multimap \beta$

alors les séquents suivants sont dérivables :

$$\overline{x'_{P_1} : \alpha_{P_1}}; \overline{x_{P_2} : \alpha_{P_2}}; \overline{x_{P_1} : d_{P_1,\mathcal{P}}.\sigma_\mathcal{P}} \vdash_D t_\mathcal{P} : d_\mathcal{P}.\sigma_\mathcal{P},$$

$$\overline{x_{P_1 \cup P_2} : \alpha_{P_1 \cup P_2}}; \cdot; x : d_\mathcal{P} \vdash_D x \uparrow^\eta : \{t_\mathcal{P} : \beta\}$$

En particulier si on pose $\mathcal{P} = \{\cdot; S_1; S_2\}$, on a :

1. $\sigma_\mathcal{P} = Id$,
2. $t_\mathcal{P} = \lambda \overline{x_{S_2} : \alpha_{S_2}}.w = v$,
3. $d_\mathcal{P} = \forall \overline{x_{S_2} : \alpha_{S_2}}.\overline{d_{S_2}} \multimap \{w : \beta\}$
4. $\alpha_\mathcal{P} = \alpha$.

Il s'ensuit donc que les séquents :

$$\cdot; \overline{x_{S_1} : \alpha_{S_1}}; \cdot \vdash_D v : d_\mathcal{P},$$

et

$$\overline{x_{S_1} : \alpha_{S_1}}; \cdot; x : d_\mathcal{P} \vdash_D x \uparrow^\eta : \{v : \alpha\}$$

sont dérivables.

Comme $\Gamma = \overline{x_{S_1} : \alpha_{S_1}}$, on peut conclure que les séquents :

$$\cdot; \Gamma; \cdot \vdash_D v : d_\mathcal{P},$$

et

$$\Gamma; \cdot; x : d_\mathcal{P} \vdash_D x \uparrow^\eta : \{v : \alpha\}$$

sont dérivables. Il suffit donc de prendre $d = d_\mathcal{P}$. □

Lemme 5.10 *Soient t_1 et t_2 deux termes tels que $x \in FV(t_1)$ et $t_1[x := t_2]$ est sous forme β-normale et η-longue ; si $\Gamma \vdash t_1[x := t_2] : \beta$ est dérivable, alors il existe une description complète d et trois contextes Γ_1, Γ_2 et Γ_3 tels que :*

1. $\Gamma = \Gamma_1, \Gamma_2, \Gamma_3$
2. $\overline{\Gamma_1} = FV(t_2)$

3. $\Gamma_3;\Gamma_1;\cdot \vdash_D t_2 \uparrow^{\eta} : d$ *est dérivable*
4. $\Gamma_1;\Gamma_2,\Gamma_3;x : d \vdash_D t_1\{t_1[x := t_2] : \beta\}$ *est dérivable*

Démonstration. On procède par induction sur la structure syntaxique de t_1 :

Cas 1 : $t_1 = \lambda y.t'_1$

dans ce cas $\beta = \beta_1 \multimap \beta_2$ et $\Gamma, y : \beta_1 \vdash t'_1 : \beta_2$, on conclut par hypothèse d'induction qu'il existe e, Γ_1, Γ_2 et Γ_3 tels que :

1. $\Gamma, y : \beta_1 = \Gamma_1, \Gamma_2, \Gamma_3$
2. $\overline{\Gamma_1} = FV(t_2)$
3. $\Gamma_1;\Gamma_2,\Gamma_3;x : e \vdash_D t'_1 : \{t'_1[x := t_2] : \beta_2\}$ est dérivable
4. $\Gamma_3;\Gamma_1;\cdot \vdash_D \overline{t_2} : e$ est dérivable

Il y a ensuite deux possibilités, ou bien, $y \in FV(e)$, ou bien $y \notin FV(e)$. Dans le premier cas $\Gamma_3 = \Gamma'_3, y : \beta_1$ et alors de la dérivabilité de

$$\Gamma_3;\Gamma_1;\cdot \vdash_D \overline{t_2} : e$$

on tire celle de :

$$\Gamma'_3;\Gamma_1;\cdot \vdash_D t_2 : \forall y : \beta_1.e$$

puis comme

$$\Gamma_1;\Gamma_2,\Gamma_3;x : e \vdash_D t'_1 : \{t'_1[x := t_2] : \beta_2\}$$

est dérivable,

$$\Gamma_1;\Gamma_2,\Gamma'_3;x : \forall y : \beta_1.e \vdash_D \lambda y.t'_1 : \{\lambda y.t'_1 : \beta_1 \multimap \beta_2\}$$

est dérivable. Pour conclure, il suffit de poser $d = \forall y : \beta_1.e$.

Dans le deuxième cas, si $x \notin FV(e)$ alors $\Gamma_2 = \Gamma'_2, y : \beta_1$ et comme

$$\Gamma_1;\Gamma_2,\Gamma_3;x : e \vdash_D t'_1 : \{t'_1[x := t_2] : \beta_2\}$$

est dérivable,

$$\Gamma_1;\Gamma'_2,\Gamma_3;x : e \vdash_D \lambda y.t'_1 : \{\lambda y.t'_1 : \beta_1 \multimap \beta_2\}$$

est dérivable. Pour conclure, il suffit de poser $d = e$.

Cas 2 : $t_1 = yv_1 \dots v_n$ et $y \neq x$

Comme v est sous forme η-longue β est atomique. De plus, comme $x \neq y$ il existe un unique k de $[1, n]$ tel que $x \in FV(v_k)$. Alors $\Gamma = \Delta_1, \dots \Delta_n, y : \overline{\beta_k}^{k,[1,n]} \multimap \beta$ et pour tout $i \in [1, n]\backslash\{k\}$, $\Delta_i \vdash_D v_i$ est dérivable et $\Delta_k \vdash_D v_k[x := t_2]$ est dérivable. Par hypothèse d'induction, il existe e, Δ^k_1, Δ^k_2 et Δ^k_3 tels que :

1. $\Delta_k = \Delta^k_1, \Delta^k_2, \Delta^k_3$
2. $\overline{\Delta^k_1} = FV(t_2)$
3. $\Delta^k_1;\Delta^k_2,\Delta^k_3;x : d \vdash_D v_k : \{v_k[x := t_2] : \beta_k\}$ est dérivable.
4. $\Delta^k_3;\Delta^k_1;\cdot \vdash_D \overline{t_2} : d$ est dérivable.

Comme pour tout $i \in [1, n]\backslash\{k\}$ $\Delta_i \vdash v_i : \beta_i$ est dérivable, il s'ensuit que

$$\cdot;\Delta_i;\cdot \vdash_D v_i : \{v_i : \beta_i\}$$

est dérivable. De même, le séquent

$$\cdot; y : \overline{\beta_{[1,n]}} \multimap \beta;\cdot \vdash_D y : \{y : \overline{\beta_{[1,n]}} \multimap \beta\}$$

est dérivable. Par application successive, on obtient que

$$\Delta_1^k; \overline{\Delta_{[1,n]\setminus\{k\}}}, \Delta_2^k, \Delta_3^k; x : d \vdash_D y\overline{v_{[1,n]}} : \{y\overline{v_{[1,k]}}[x := t_2] \dots v_n : \beta\}$$

est dérivable. Pour conclure, il suffit de poser $d = e$, $\Gamma_1 = \Delta_1^k$, $\Gamma_2 = \overline{\Delta_{[1,n]\setminus\{k\}}}, \Delta_2^k$ et $\Gamma_3 = \Delta_3^k$.

Cas 3 : $t_1 = xv_1 \dots v_n$

Dans ce cas $\Gamma = \Gamma_1, \Delta_1, \dots, \Delta_n$ et $\overline{\Gamma_1} = FV(t_2)$ et pour tout i, $\Delta_i \vdash_D v_i : \beta_i$ est dérivable.

D'après le corollaire précédent, il existe d_1, ..., d_n tels que :

$$\cdot; \Delta_i; \cdot \vdash_D v_i : d_i$$

et

$$\Delta_i; \cdot; x_i : d_i \vdash_D x_i \uparrow^\eta : \{v_i : \beta_i\}$$

sont dérivables.

On pose $d = \overline{d_{[1,n]}} \multimap \{t_2\overline{v_{[1,n]}} : \beta\}$, il s'ensuit que :

$$\Gamma; \cdot; x : d \vdash_D x : \overline{d_{[1,n]}} \multimap \{t_2\overline{v_{[1,n]}} : \beta\}$$

et par application successive avec les séquents $\cdot; \Delta_i; \cdot \vdash_D v_i : d_i$, on obtient que

$$\Gamma_1; \overline{\Delta_{[1,n]}}; x : d \vdash_D v : \{t_2\overline{v_{[1,n]}} : \beta\}$$

est dérivable. Puis, comme $\cdot; \Gamma_1; \cdot \vdash_D t_2 : \{t_2 : \overline{\beta_{[1,n]}} \multimap \beta\}$ est dérivable, il vient que :

$$\Gamma_3; \Gamma_1; \overline{x_{[1,n]} : d_{[1,n]}} \vdash_D t_2\overline{x_{[1,n]}1} \uparrow^\eta : \{t_2\overline{v_{[1,n]}} : \beta\}$$

est dérivable. Et finalement nous pouvons conclure que :

$$\Gamma_3; \Gamma_1; \cdot \vdash_D \lambda\overline{x_{[1,n]}}.t_2\overline{x_{[1,n]}} \uparrow^\eta : d$$

est dérivable. □

Lemme 5.11 *Soient t_1 et t_2 deux λ-termes sous forme η-longue, d une description et x une variable tels que :*

1. $x \in FV(t_1)$,
2. $\Gamma; \Gamma'; \Delta \vdash_D t_1[x := t_2] : d$ *est dérivable*

alors il existe une description complète e, Γ_1, Γ_2, Γ_3, Γ_4, Δ_1 et Δ_2 tels que :

1. $\Gamma = \Gamma_1 \setminus \Gamma_4$
2. $\Gamma' = \Gamma_2, \Gamma_4$
3. $\Delta = \Delta_1, \Delta_2$
4. $\Gamma_1; \Gamma_2; \Delta_1, x : e \vdash_D t_1 : d$ *est dérivable*
5. $\Gamma_3; \Gamma_4; \Delta_2 \vdash_D t_2 : e$ *est dérivable.*

Démonstration. Nous distinguons deux cas suivant que $t_1 = x$ ou non. Si $t_1 = x$ alors $\Gamma; \Gamma'; \Delta \vdash_D t_2 : d$ est dérivable. On pose $e = d$ et on obtient que $\Gamma, \Gamma'; \cdot; x : e \vdash_D x : d$ est dérivable. Pour conclure, il nous suffit donc de poser $\Gamma_1 = \Gamma, \Gamma'$, $\Gamma_2 = \cdot$, $\Gamma_3 = \Gamma$, $\Gamma_4 = \Gamma'$, $\Delta_1 = \cdot$ et $\Delta_2 = \Delta$.

Dans le cas où $t_1 \neq x$, on démontre cette propriété par induction sur la dérivation de $\Gamma; \Gamma'; \Delta \vdash_D t_1[x := t_2] : d$. Tous les cas se déduisent immédiatement de l'hypothèse d'induction, à l'exception d'un cas qui requiert l'utilisation du lemme précédent. Ce cas est celui où $\Gamma; \Gamma'; \Delta \vdash_D t_1[x := t_2] : d$ se dérive de la façon suivante :

$$\frac{\Gamma' \vdash_D t_1[x := t_2] : \beta}{\cdot;\Gamma';\cdot \vdash_D t_1[x := t_2] : \{t_1[x := t_2] : \beta\}}$$

Comme $\Gamma' \vdash_D t_1[x := t_2]$ est dérivable, d'après le lemme précédent $\Gamma' = \Gamma'_1, \Gamma'_2, \Gamma'_3$ et il existe une description complète d telle que :

$$\Gamma'_1; \Gamma'_2, \Gamma'_3; x : d \vdash_D t_1 : \{t_1[x := t_2] : \beta\}$$

et

$$\Gamma'_2; \Gamma'_1; \cdot \vdash_D t_2 \uparrow^\eta : d$$

mais comme t_2 est sous forme η-longue, $t_2 = t_2 \uparrow^\eta$ et il vient donc que :

$$\Gamma'_2; \Gamma'_1; \cdot \vdash_D t_2 : d$$

est dérivable. Pour conclure, il nous suffit de poser $\Gamma_1 = \Gamma'_1$, $\Gamma_2 = \Gamma'_2, \Gamma'_3$, $\Gamma_3 = \Gamma'_2$, $\Gamma_4 = \Gamma'_1$, $\Delta_1 = \cdot$ et $\Delta_2 = \cdot$. □

Lemme 5.12 *Soient t_1 et t_2 deux λ-termes linéaires sous forme η-longue et soit d une description complète, si $\Gamma; \Gamma'; \Delta \vdash_D t_1 : d$ est dérivable et si $t_2 \to_\beta t_1$ alors $\Gamma; \Gamma'; \Delta \vdash_D t_2 : d$ est dérivable.*

Démonstration. Pour démontrer ce lemme, il faut remarquer que si $t_2 \to_\beta t_1$ alors $t_2 = C[(\lambda x.v_1)v_2]$ et que $t_1 = C[v_1[x := v_2]]$. Ensuite, on distingue deux cas : celui où $C[] = []$ et on peut conclure à l'aide du lemme précédent. En effet, si $C[] = []$ alors $t_2 = (\lambda x.v_1)v_2$ et $t_1 = v_1[x := v_2]$, il s'ensuit que :

$$\Gamma; \Gamma'; \Delta \vdash_D v_1[x := v_2] : d$$

de plus comme d est une description complète et que v_1 est sous forme η-longue (puisque $t_2 = (\lambda x.v_1)v_2$ est sous forme η-longue), on peut utiliser le lemme précédent. On a donc l'existence d'une description complète e et de contextes Γ_1, Γ_2, Γ_3, Γ_4, Δ_1 et Δ_2 tels que :

1. $\Gamma = \Gamma_1 \backslash \Gamma_4$
2. $\Gamma' = \Gamma_2, \Gamma_4$
3. $\Delta = \Delta_1, \Delta_2$
4. $\Gamma_1; \Gamma_2; \Delta_1, x : e \vdash_D v_1 : d$ est dérivable
5. $\Gamma_3; \Gamma_4; \Delta_2 \vdash_D v_2 : e$ est dérivable

En utilisant la règle d'application comme suit, on obtient :

$$\frac{\dfrac{\Gamma_1; \Gamma_2; \Delta_1, x : e \vdash_D v_1 : d}{\Gamma_1; \Gamma_2; \Delta_1 \vdash_D \lambda x.v_1 : e \multimap d} \quad \Gamma_3; \Gamma_4; \Delta_2 \vdash_D v_2 : e}{\Gamma_1 \backslash \Gamma_4; \Gamma_2, \gamma_4; \Delta_1, \Delta_2 \vdash_D (\lambda x.v_1)v_2 : d}$$

Ce qui démontre que $\Gamma; \Gamma'; \Delta \vdash_D t_2 : d$ est dérivable.

Dans le cas où $C[] \neq []$, on utilise une induction sur la dérivation de $\Gamma; \Gamma'; \Delta \vdash_D t_1 : d$. Cette induction est simple et nous l'omettons. □

Lemme 5.13 ***(Substitution)*** *Si $\Gamma_1; \Gamma'_1; \Delta_1, x : e \vdash_D t : d$ et $\Gamma_2; \Gamma'_2; \Delta_2 \vdash_D u : e$ sont dérivables alors $\Gamma_1 \backslash \Gamma'_2; \Gamma'_1, \Gamma'_2; \Delta_1, \Delta_2 \vdash_D t[x := u] : d$ est dérivable.*

Démonstration. On procède par induction sur la dérivation de $\Gamma_1; \Gamma'_1; \Delta_1, x : e \vdash_D t : d$. □

Lemme 5.14 *Soit $(t_k)_{k\in[0,n]}$ une famille de termes sous forme β-normale η-longue, soit $(\Gamma'_k)_{k\in[0,n]}$ une famille de contextes, soit $(\alpha_k)_{k\in[1,n]}$ une famille de types et soit u un terme sous forme β-normale η-longue. Ces familles vérifient les propriétés suivantes :*

1. $\{x_1; \ldots; x_n\} \subseteq FV(t_0)$

2. *pour tout* $k, k' \in [1,n]$, $\overline{\Gamma'_k} \cap \overline{\Gamma'_{k'}} \neq \emptyset$ *implique que* $k = k'$
3. *pour tout* $k \in [1,n]$, $\Gamma'_k \vdash_D t_k : \alpha_k$ *est dérivable.*
4. $\Gamma_0, \overline{x_{[1,n]} : \alpha_{[1,n]}} \vdash_D t_0 : \alpha_0$ *est dérivable.*
5. u *est la forme* β*-normale de* $t_0[x_k := t_k]_{k=1}^n$

si et seulement si il existe une famille de descriptions complètes, compatibles avec $u : \alpha_0$ $(d_k)_{k\in[1,n]}$ *et une famille de contexte* $(\Gamma_k)_{k\in[1,n]}$ *telles que :*

1. $\overline{\Gamma_{[1,n]}}; \Gamma_0; \overline{x_{[1,n]} : d_{[1,n]}} \vdash_D t_0 : \{u : \alpha_0\}$ *est dérivable.*
2. *pour tout* $k \in [1,n]$, $\Gamma_k; \Gamma'_k; \cdot \vdash_D t_k : d_k$ *est dérivable.*

Démonstration. Le second sens de l'implication se démontre en itérant le lemme précédent et en utilisant le lemme de correction. Il nous reste à démontrer le premier sens.

Tout d'abord, comme le λ-calcul linéaire simplement typé vérifie la réduction du sujet, il vient que $\Gamma_0, \overline{\Gamma'_{[1,n]}} \vdash_D u : \alpha_0$ est dérivable et donc le séquent :

$$\cdot; \Gamma_0, \overline{\Gamma'_{[1,n]}}; \cdot \vdash_D u : \{u : \alpha_0\}$$

est dérivable.

Comme $t_0[x_k := t_k]_{k=1}^n \xrightarrow{*}_\beta u$, en itérant le lemme 5.12, on obtient que :

$$\cdot; \Gamma_0, \overline{\Gamma'_{[1,n]}}; \cdot \vdash_D t_0[x_k := t_k]_{k=1}^n : \{u : \alpha_0\}$$

est dérivable.

Ensuite, en itérant le lemme 5.11, on obtient l'existence d'une famille de descriptions complètes $(d_k)_{k\in[1,n]}$ et une famille de contextes $(\Gamma'_k)_{k\in[1,n]}$ telles que le séquent

$$\overline{\Gamma'_{[1,n]}}; \Gamma_0; \overline{x_{[1,n]} : d_{[1,n]}} \vdash_D t_0 : \{u : \alpha_0\}$$

est dérivable et que pour tout $k \in [1,n]$, le séquent :

$$\Gamma_k; \Gamma'_k; \cdot \vdash_D t_k : d_k$$

est dérivable. □

Ce dernier lemme et le lemme de correction nous donne les outils nécessaires à la construction d'un algorithme de résolution d'équations de filtrage. En effet, si on cherche à résoudre une équation $t \stackrel{?}{=} u$, on cherche à trouver des termes $t_1, \ldots, t_n$ tels que $t[\mathbf{X}_1 := t_1; \ldots, \mathbf{X}_n := t_n] =_\beta u$ si u, t et les t_i sont sous forme η-longue. D'après le lemme précédent, cela revient à chercher des descriptions complètes et compatibles avec u $d_1, \ldots, d_n$ dont la sémantique n'est pas vide et qui permettent de dériver que $\lambda\overline{x_{[1,n]}}.t[\mathbf{X}_k := x_k]_{k\in[1,n]} : \overline{d_{[1,n]}} \multimap \{u : \alpha\}$.

5.2.2 Un algorithme de résolution d'équations linéaires de filtrage

L'algorithme que nous nous proposons d'étudier est en fait un algorithme de recherche de dérivation dans le calcul des descriptions syntaxiques. Nous stockons dans un tableau les séquents que l'on peut dériver dans le calcul des descriptions syntaxiques. Afin de passer de ce système de dérivation à un vrai système de résolution d'équation, il nous faut affiner encore notre approche. Il nous faut en particulier prendre en compte l'α-conversion. Comme nous voulons prendre en compte les phénomènes d'α-conversion, nous n'allons plus utiliser l'α-conversion pour identifier des termes ou des descriptions. Nous allons considérer également que toutes les variables liées d'un terme sont deux à deux distinctes. De plus si $d = \forall x : \alpha.d'$ et $e = \forall x : \alpha.e'$ alors la variable x utilisée pour construire d' et e' est la même.

Comme nous aurons à établir qu'il existe un terme qui peut être typé à l'aide d'une certaine description mais que ce terme nous est indifférent, nous allons étudier le système logique associé au calcul des descriptions par la correspondance de Curry-Howard. Nous ne donnons pas les démonstrations des quelques lemmes qui suivent car ils s'établissent avec les méthodes habituellement utilisées pour d'autres calculs.

$$\frac{\Gamma \vdash_D t : \alpha}{\Gamma; \{t : \alpha\} \vdash_{ld} \{t : \alpha\}} \text{ld-sym} \qquad \frac{\Gamma \vdash_D t : \alpha}{\Gamma; \cdot \vdash_{ld} \{t : \alpha\}} \text{ld-ax}$$

$$\frac{\Gamma; \Delta, d_1 \vdash_{ld} d_2}{\Gamma; \Delta \vdash_{ld} d_1 \multimap d_2} \text{ld-}\multimap_D \qquad \frac{\Gamma_1; \Delta_1 \vdash_{ld} d_1 \quad \Gamma_1, \Gamma_2; \Delta_2, d_2 \vdash_{ld} d \quad \mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2, d_1 \multimap d_2 \vdash_{ld} d} \text{ld-}\multimap_G$$

$$\frac{\Gamma, x : \alpha; \Delta \vdash_{ld} \{u : \beta\}}{\Gamma; \forall x : \alpha. \Delta \vdash_{ld} \{\lambda x.u : \alpha \multimap \beta\}} \text{ld-}\lambda \qquad \frac{\Gamma_1; \Delta_1 \vdash_{ld} \{u_1 : \alpha \multimap \beta\} \quad \Gamma_2; \Delta_2 \vdash_{ld} \{u_2 : \alpha\}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash_{ld} \{u_1 u_2 : \beta\}} \text{ld-app}$$

$$\frac{\Gamma, x : \alpha; \Delta \vdash_D d \quad x \in DV(d, \Delta)}{\Gamma; \forall x : \alpha \Delta \vdash_D \forall x : \alpha. d} \text{ld-}\forall$$

FIG. 5.3 – Logique des descriptions syntaxiques

Définition 5.21 *On note $\mathcal{S}_t$ le multi-ensemble des sous-termes de t. Le multi-ensemble des sous-termes utilisés par une description, $\mathcal{S}_d$, est défini inductivement par :*

1. *si $d = \{t : \alpha\}$, $\mathcal{S}_d = \mathcal{S}_t$*
2. *si $d = d_1 \multimap d_2$, $\mathcal{S}_d = \begin{cases} \mathcal{S}_{d_2} \backslash \mathcal{S}_{d_1} \text{ si } \mathcal{S}_{d_1} \neq \bot, \text{ si } \mathcal{S}_{d_2} \neq \bot \text{ et si } \mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2} \\ \bot \text{ sinon} \end{cases}$*
3. *si $d = \forall x : \alpha.e$ alors $\mathcal{S}_d = \mathcal{S}_e$*

Définition 5.22 *Soit d une description, d est bien définie si $\mathcal{S}_d \neq \bot$.*

Notation 5.3 *Si $\Delta = \overline{d_{[1,n]}}$ est un contexte de descriptions alors on note $\mathcal{S}_\Delta$ le multi-ensemble de termes $\bigcup_{i=1}^{n} \mathcal{S}_{d_i}$.*

Définition 5.23 *Étant donné un contexte de description Δ et une description d tels que $\mathcal{S}_\Delta \subseteq \mathcal{S}_d$, on définit deux ensembles de variables :*

1. *l'ensemble des variables utilisées $UV(d, \Delta) = \{x | x \in \mathcal{S}_d \backslash \mathcal{S}_\Delta\}$*
2. *l'ensemble des variables disponibles $DV(d, \Delta) = \{x | x \in FV(d) \cup FV(\Delta) \wedge x \notin \mathcal{S}_d \backslash \mathcal{S}_\Delta\}$*

Définition 5.24 *Un séquent de la logique des descriptions est de la forme $\Gamma; \Delta \vdash_{ld} d$, où Γ est un contexte d'assignation de types à des variables et Δ est un multi-ensemble de descriptions. On dit qu'un tel séquent est prouvable si il peut être dérivé à partir des règles données par la figure 5.3.*

Lemme 5.15 *Si $\Gamma; \Delta \vdash_{ld} d$ est dérivable alors pour tout $e \in \Delta$, $\mathcal{S}_e \neq \bot$, $\mathcal{S}_d \neq \bot$ et $\mathcal{S}_\Delta \subseteq \mathcal{S}_d$.*

Démonstration. On procède par induction sur la dérivation de $S = \Gamma; \Delta \vdash_D d$:

Cas 1: S dérive de ld-sym

dans ce cas, $\Delta = \{t : \alpha\}$ et $d = \{t : \alpha\}$ et $\mathcal{S}_\Delta = \mathcal{S}_d$, ce qui nous permet de conclure.

Cas 2: S dérive de ld-ax

dans ce cas, $\Delta = \cdot$ et $\cdot = \emptyset$; la conclusion est évidente.

Cas 3: S dérive de ld-$\multimap_D$

dans ce cas, $S = \Gamma; \Delta \vdash_{ld} d_1 \multimap d_2$ et S dérive de $\Gamma; \Delta, d_1 \vdash_{ld} d_2$. Or par hypothèse d'induction, pour tout $e \in \Delta$, $\mathcal{S}_e \neq \bot$, $\mathcal{S}_{d_1} \neq \bot$ et $\mathcal{S}_{d_2} \neq \bot$ et $\mathcal{S}_\Delta, \mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$. Il s'ensuit que $\mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$ et donc que $\mathcal{S}_d = \mathcal{S}_{d_1 \multimap d_2} \neq \bot$. De plus, $\mathcal{S}_\Delta \subseteq \mathcal{S}_{d_2} \backslash \mathcal{S}_{d_1} = \mathcal{S}_{d_1 \multimap d_2}$.

Cas 4: S dérive de ld-$\multimap_G$

dans ce cas, $S = \Gamma_1, \Gamma_2; \Delta_1, \Delta_2, d_1 \multimap d_2 \vdash_{ld} d$, $\mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$ et S dérive des séquents $\Gamma_1; \Delta_1 \vdash_D d_1$ et $\Gamma_1, \Gamma_2; \Delta_2, d_2 \vdash_{ld} d$. Par hypothèse d'induction, on obtient que pour tout $e \in \Delta_1, \Delta_2$ $\mathcal{S}_e \neq \bot$, $\mathcal{S}_{d_1} \neq \bot$, $\mathcal{S}_{d_2} \neq \bot$ $\mathcal{S}_d \neq \bot$, $\mathcal{S}_{\Delta_1} \subseteq \mathcal{S}_{d_1}$ et $\mathcal{S}_{\Delta_2}, \mathcal{S}_{d_2} \subseteq \mathcal{S}_d$. Comme $\mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$, $\mathcal{S}_{d_1 \multimap d_2} \neq \bot$. De plus $\mathcal{S}_{\Delta_1}, \mathcal{S}_{\Delta_2}, \mathcal{S}_{d_1 \multimap d_2} = \mathcal{S}_{\Delta_1}, \mathcal{S}_{\Delta_2}, \mathcal{S}_{d_2} \backslash \mathcal{S}_{d_1}$, or comme $\mathcal{S}_{\Delta_1} \subseteq \mathcal{S}_{d_1}$ et $\mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$, $\mathcal{S}_{\Delta_1}, \mathcal{S}_{d_2} \backslash \mathcal{S}_{d_1} \subseteq \mathcal{S}_{d_2}$. De là vient que $\mathcal{S}_{\Delta_1}, \mathcal{S}_{\Delta_2}, \mathcal{S}_{d_1 \multimap d_2} \subseteq \mathcal{S}_{\Delta_2}, \mathcal{S}_{d_2} \subseteq \mathcal{S}_d$.

Cas 5: S dérive de ld-λ

dans ce cas, $S = \Gamma; \forall x : \alpha . \Delta \vdash_{ld} \{\lambda x.u : \alpha \multimap \beta\}$ et S dérive du séquent $\Gamma, x : \alpha; \Delta \vdash_{ld} \{u : \beta\}$. Par hypothèse d'induction, on a que pour tout $e \in \Delta$ $\mathcal{S}_e \neq \bot$ et $\mathcal{S}_\Delta \subseteq \mathcal{S}_u$. De plus $\mathcal{S}_{\forall x:\alpha.\Delta} = \mathcal{S}_\Delta$ et $\mathcal{S}_{\lambda x.u} = \mathcal{S}_u, \lambda x.u$ et on peut conclure.

Cas 6: S dérive de ld-app

dans ce cas, $S = \Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash_{ld} \{u_1 u_2 : \alpha \multimap \beta\}$ et S dérive des séquents $\Gamma_1; \Delta_1 \vdash_{ld} \{u_1 : \alpha \multimap \beta\}$ et $\Gamma_2; \Delta_2 \vdash_{ld} \{u_2 : \alpha\}$. Ici, on conclut directement grâce à l'hypothèse d'induction.

Cas 7: S dérive de ld-$\forall$

dans ce cas, la conclusion vient immédiatement de l'hypothèse d'induction.

□

Définition 5.25 *Un séquent $\Gamma; \Gamma'; \Delta \vdash_D t : d$ est dit complet si d est une description complète et si pour tout $x : e \in \Delta$, e est une description complète.*

Lemme 5.16 *Si les séquents complets $\Gamma_1; \Gamma_1'; \Delta_1, x : e_2 \vdash_D t : d$ et $\Gamma_2; \Gamma_2'; \Delta_2 \vdash_D u : e_1$ sont dérivables et si $FV(e_1) \subseteq FV(e_2)$ alors $\Gamma_1 \backslash \Gamma_2'; \Gamma_1', \Gamma_2'; \Delta_1, \Delta_2, y : e_1 \multimap e_2 \vdash_D t[x := yu] : d$ est dérivable.*

Démonstration. Comme $FV(e_1) \subseteq FV(e_2)$, il existe Γ_2'' tel que $\Gamma_2''; \cdot; y : e_1 \multimap e_2 \vdash_D y : e_1 \multimap e_2$ est dérivable. Ensuite on obtient que $\Gamma_2'' \backslash \Gamma_2'; \Gamma_2'; \Delta_2, y : e_1 \multimap e_2 \vdash_D yu : e_2$ est dérivable. Puis par l'application du lemme de substitution on obtient que $\Gamma_1 \backslash \Gamma_2'; \Gamma_1', \Gamma_2'; \Delta_1, \Delta_2 \vdash_D t[x := yu] : e_2$ est dérivable. □

Lemme 5.17 ***(Élimination des coupures)*** *Si $\Gamma_1; \Delta_1 \vdash_D e$ et $\Gamma_1, \Gamma_2; \Delta_2, e \vdash_{ld} d$ sont dérivables alors $\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash_{ld} d$ est dérivable.*

Démonstration. Ce lemme se démontre de la même façon que pour la logique linéaire, par induction sur la dérivation de $\Gamma_1, \Gamma_2; \Delta_2, e \vdash_{ld} d$. □

Lemme 5.18 ***(Correspondance de Curry-Howard)*** *Le séquent $\Gamma; \Delta \vdash_{ld} d$ est dérivable si et seulement si il existe un terme t tel que $\Gamma_{DV(d,\Delta)}; \Gamma_{UV(d,\Delta)}; \Delta' \vdash_D t : d$ est dérivable, avec $\tilde{\Delta'} = \Delta$.*

Démonstration. Dans un sens comme dans l'autre cette équivalence se démontre par induction sur le séquent dont on sait qu'il est dérivable et en utilisant les lemmes 5.17 et 5.16. □

Nous n'avons pas encore étudié dans le détail la complexité de la logique des descriptions. Mais il semble qu'établir la dérivabilité d'un séquent soit NP-complet. Cependant il y a quelques heuristiques qui permettent de simplifier cette recherche de démonstration. Par exemple, certaines règles sont réversibles, comme ld- $\multimap_D$ et ld-$\forall$. De plus, si on doit démontrer un séquent de la forme $\Gamma; \Delta \vdash_D \{u : \alpha\}$, ou

$$\frac{\Gamma;\cdot\vdash_{ld} d}{\cdot;\mathbf{X}:\overline{d};\Gamma;\cdot;\cdot\vdash_{eq}\mathbf{X}:d}\ \text{Inc}_{eq} \qquad \frac{\Gamma;\cdot\vdash_{ld} d}{\cdot;\cdot;\Gamma;\cdot;x:d\vdash_{eq} x:d}\ \text{Var}_{eq} \qquad \frac{\cdot\vdash_{D} t:\alpha}{\cdot;\cdot;\cdot;\cdot;\cdot\vdash_{eq} t:\{t:\alpha\}}\ \text{Ref}_{eq}$$

$$\frac{}{x:=y;\cdot;\cdot;y:\alpha;\cdot\vdash_{eq} x:\{y:\alpha\}}\ \text{Id-var}_{eq} \qquad \frac{\sigma,x:=y;\Pi;\Gamma;\Gamma',y:\alpha;\Delta\vdash_{eq} t:\{u:\beta\}}{\sigma;\Pi;\Gamma;\Gamma';\forall y:\alpha.\Delta\vdash_{eq}\lambda x.t:\{\lambda y.u:\alpha\multimap\beta\}}\ \lambda\text{-eq}_{eq}$$

$$\frac{\sigma_1;\Pi_1;\Gamma_1;\Gamma'_1;\Delta_1\vdash_{eq} t_1:\{u_1:\alpha\multimap\beta\}\quad \sigma_2;\Pi_2;\Gamma_2;\Gamma'_2;\Delta_2\vdash_{eq} t_2:\{u_2:\alpha\}}{\sigma_1,\sigma_2;\Pi_1,\Pi_2;\Gamma_1,\Gamma_2;\Gamma'_1,\Gamma'_2;\vdash_{eq} t_1t_2:\{u_1u_2:\beta\}}\ \text{App-eq}_{eq}$$

$$\frac{\sigma_1;\Pi_1;\Gamma_1;\Gamma'_1;\Delta_1\vdash_{eq} t_1:d\multimap e\quad \sigma_2;\Pi_2;\Gamma_2;\Gamma'_2;\Delta_2\vdash_{eq} t_2:d}{\sigma_1,\sigma_2;\Pi_1,\Pi_2;\Gamma_1\backslash\Gamma'_2;\Gamma'_1,\Gamma'_2;\vdash_{eq} t_1t_2:e}\ \text{App-desc}_{eq}$$

$$\frac{\sigma;\Pi;\Gamma;\Gamma';\Delta,x:e\vdash_{eq} t:d}{\sigma;\Pi;\Gamma;\Gamma';\Delta\vdash_{eq}\lambda x.t:e\multimap d}\ \lambda\text{-desc}_{eq}$$

FIG. 5.4 – Système formel de résolution d'équations

bien il y a une description de Δ dont la conclusion est $\{u:\alpha\}$ et dans ce cas, on applique la règle ld- $\multimap_G$ sur cette description ou bien l'utilisation des règles ld-λ et ld-app est réversible. Par ailleurs il faut maintenir la propriété énoncée par le lemme 5.15. Tous ces résultats permettent, dans une certaine mesure, de déterminiser la recherche de démonstration dans la logique des descriptions. Dans le pire des cas, la recherche de démonstration d'un séquent $\Gamma;\cdot\vdash_{ld} d$ s'effectue en $\mathcal{O}(2^{\rho(d)})$ (où $\rho(d)$ est le nombre de symbole $\multimap$ dans d).

Avant de décrire précisément l'algorithme de résolution des équations de filtrage, nous allons spécialiser le calcul des descriptions syntaxiques afin qu'il prenne en compte toutes les difficultés techniques qu'un algorithme peut rencontrer, soit essentiellement les problèmes d'α-conversion.

Définition 5.26 *Étant donnés deux ensembles disjoints de variables $\overline{x_{[1,n]}}$ et $\overline{y_{[1,n]}}$, un contexte de renommage sur ces ensembles est un ensemble $\sigma = x_1 := y_1,\ldots,x_n := y_n$, de plus on note $\tilde{\sigma}$ la substitution $[x_1 := y_1;\ldots;x_n := y_n]$.*

Définition 5.27 *Un séquent de résolution d'équation de filtrage est un séquent de la forme :*

$$\sigma;\Pi;\Gamma;\Gamma';\Delta\vdash_{eq} t:d$$

où σ est un contexte de renommage, Π est un contexte d'assignation de types à des inconnues, Γ et Γ' sont des contextes d'assignation de types à des variables et Δ est un contexte d'assignation de descriptions à des variables. On dit d'un tel séquent qu'il est démontrable si on peut le dériver à partir des règles de la figure 5.4

Lemme 5.19 *Le séquent $\sigma;\overline{\mathbf{X}_{[1,n]}:\alpha_{[1,n]}};\Gamma;\Gamma';\overline{y_{[1,p]}:e_{[1,p]}}\vdash_{eq} t:d$ est dérivable si et seulement si il existe une famille de descriptions $(d_k)_{k\in[1,n]}$, deux familles de termes $(v_k)_{k\in[1,n]}$ et $(w_k)_{k\in[1,p]}$, et quatre familles de contextes $(\Gamma_k)_{k\in[1,n]}$, $(\Gamma'_k)_{k\in[1,n]}$, $(\Delta_k)_{k\in[1,n]}$, $(\Delta'_k)_{k\in[1,n]}$ telles que si $\sigma' = [\mathbf{X}_k := x_k]_{k=1}^n$ alors :*

1. *$\Gamma;\Gamma';\overline{y_{[1,p]}:e_{[1,p]}},\overline{x_{[1,n]}:d_{[1,n]}}\vdash_D (t.\sigma).\sigma':d$ est dérivable.*
2. *pour tout $k\in[1,n]$, $\Gamma_k;\Gamma'_k;\cdot\vdash_D t_k:d_k$ est dérivable.*
3. *pour tout $k\in[1,n]$, $\Delta_k;\Delta'_k;\cdot\vdash_D t_k:d_k$ est dérivable.*

Démonstration. Ce lemme se montre par simple induction sur les dérivations en utilisant le lemme 5.18. □

Lemme 5.20 *Étant donnée une équation de filtrage $t \stackrel{?}{=} u$ telle que, t et u sont sous forme β-normale η-longue, leurs variables liées sont deux à deux distinctes et $\overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}; \Delta_1 \vdash t : \alpha$ et $\Delta_1, \Delta_2 \vdash u : \alpha$ sont dérivables, alors $\cdot; \overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}; \Delta_2; \Delta_1; \cdot \vdash_{eq} t : \{u : \alpha\}$ est dérivable si et seulement si il existe une substitution σ telle que $t.\sigma =_\beta u$ (modulo α-conversion).*

Démonstration. Ce lemme est la combinaison des résultats énoncés par le lemme précédent et le lemme 5.14. □

Lors d'une dérivation dans le système formel de résolution d'équations, les substitutions sont entièrement déterminées par les autres contextes.

Lemme 5.21 *Si $\sigma; \Pi; \Gamma; \Gamma'; \Delta \vdash_{eq} t : d$ et $\sigma'; \Pi; \Gamma; \Gamma'; \Delta \vdash_{eq} t : d$ sont dérivables alors $\sigma = \sigma'$.*

Démonstration. Ce lemme se démontre par induction sur les dérivations. □

Les figures 5.6 et 5.7 montrent une dérivation dans le système formel de résolution d'équations qui démontre qu'il existe une solution à l'équation de filtrage $\mathbf{X}(\lambda z_1.b(\lambda z_2.z_1(\lambda x_1 x_2.\mathbf{Z}x_1x_2)z_2) \stackrel{?}{=} b(\lambda x.cxa)$ où a est une constante de type α, b est une constante de type $(\alpha \multimap \alpha) \multimap \alpha$ et c une constant de type $\alpha \multimap \alpha \multimap \alpha$ et les inconnues $\mathbf{X}$ et $\mathbf{Z}$ ont respectivement pour type $(((\alpha \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha) \multimap \alpha) \multimap \alpha$ et $\alpha \multimap \alpha \multimap \alpha$. Dans cet exemple, afin de simplifier les notations, nous avons enlevé les types des descriptions atomiques et avons noté $\{t\}$ la description $\{t : \alpha\}$. Par ailleurs, dans les dérivations des figures 5.6 et 5.7 nous n'avons pas démontré le séquent $x : \alpha; \cdot \vdash_{ld} e$, sa démonstration est donnée par la figure 5.8.

Avant de décrire formellement cet algorithme, nous remarquons qu'après avoir effectué une substitution, les λ-abstractions qui donnent naissance à des β-redex sont déterminées. En fait elles ne dépendent pas de la substitution. Aussi, étant donné un terme t qui contient des inconnues, si $t = C[v]$ alors on distingue en deux catégories les variables libres de v : il y a les variables actives qui seront substituées par un terme après qu'on a appliqué une substitution à t et il y a les variables inactives, les autres. Dans le système formel de résolution d'équations, les variables actives seront systématiquement déclarées dans le contexte de description alors que les variables inactives seront systématiquement déclarées dans le contexte de renommage. Formellement, cette distinction est faite par la définition suivante :

Définition 5.28 *Étant donné un terme t, on définit à l'aide du système de déduction suivant l'ensemble des variables actives et inactives d'un sous-terme de t :*

$$\frac{}{(\emptyset; FV(t); t)_t^-} \qquad \frac{(A; I, x; x\overline{v_{[1,n]}})_t^\epsilon \quad i \in [1,n]}{(A \cap FV(v_i); I \cap FV(v_i); v_i)_t^-}$$

$$\frac{(A; I; \lambda x.v)_t^- \quad i \in [1,n]}{(A; I, x; v)^-} \qquad \frac{(A; I; \lambda x.v)_t^+ \quad i \in [1,n]}{(A, x; I; v)^+}$$

$$\frac{(A; I; \mathbf{X}\overline{v_{[1,n]}})_t^\epsilon \quad i \in [1,n]}{(A \cap FV(v_i); I \cap FV(v_i); v_i)^+} \qquad \frac{(A, x; I; x\overline{v_{[1,n]}})_t^\epsilon \quad i \in [1,n]}{(A \cap FV(v_i); I \cap FV(v_i); v_i)_t^+}$$

Dans le cas où les variables utilisées pour construire t sont deux à deux distinctes, pour tout sous-terme v de t, il existe une unique partition A_v, I_v de $FV(v)$ tel que $(A_v; I_v; v)_t^\epsilon$ soit dérivable. Ainsi dans ce cas, nous noterons A_v par A_v^t et I_v par I_v^t.

L'algorithme que nous proposons mémorise les séquents dérivables dans le système formel de résolution d'équations $\sigma; \Pi; \Gamma; \Gamma'; \Delta \vdash_{eq} t : d$ où t est sous forme η-longue.

Définition 5.29 *On appelle état de résolution d'une équation de filtrage $t \stackrel{?}{=} u$ un ensemble de séquents $\sigma; \Pi; \Gamma; \Gamma'; \Delta \vdash_{eq} v : d$ tels que les descriptions que Δ associe à des variables sont des descriptions complètes et compatibles avec u, tels que d est une description complète et compatible avec u et tels que v est sous forme η-longue.*

Définition 5.30 *Si H est un état de résolution d'une équation de filtrage $t \stackrel{?}{=} u$, alors $H \rightarrow_{EQ} H \cup S$ si $H \rightarrow_{eq} S$ où $\rightarrow_{eq}$ est la relation décrite par la figure 5.5.*

On peut facilement démontrer que si v est un sous-terme de t tel que $S = \sigma; \Pi; \Gamma; \Gamma'; \Delta \vdash_{eq} v : d$ est dérivable alors $\emptyset \stackrel{*}{\rightarrow}_{EQ} H$ et $S \in H$ et réciproquement. Ainsi, d'après le lemme 5.20, l'algorithme que nous venons de définir est correct et complet. Pour ce qui est maintenant de sa complexité, on peut voir que cet algorithme est en $\mathcal{O}(|t||u|^{c+d+1})$ avec

$$c = \max_{v \in \mathcal{S}_t}(\sum_{x \in A_v^t} \rho(x))$$

$$d = \max_{i \in [1,n]}(\rho(\mathbf{X}_i))$$

si $\overline{\mathbf{X}_{[1,n]}}$ est exactement l'ensemble des inconnues qui interviennent dans t. En effet, le nombre de fois que l'on peut appliquer des règles de l'algorithme dépend seulement du nombre de façon dont on peut instancier les descriptions associées aux variables actives du sous-terme de t que type ce séquent (le contexte de substitution étant unique pour une assignation de description donnée), or le nombre des instanciations possibles de ces descriptions est $\mathcal{O}(|u|^c)$. Il faut également tenir compte du fait qu'il est parfois nécessaire d'effectuer une recherche de démonstration dans la logique des descriptions, ce qui s'effectue en un temps au plus de $\mathcal{O}(|u|^d)$. Concernant ces recherches de démonstration, elles peuvent être accélérées en remarquant qu'à chaque fois que le séquent $S = \sigma; \Gamma; \Gamma'; \Delta \vdash_{eq} v : d$ est mémorisé, c'est que $\Gamma, \Gamma'; \cdot \vdash_{ld} d$ est dérivable. Ainsi, pour $c + d$ borné on obtient une classe d'équations de filtrage que l'on peut résoudre en temps polynômial. En particulier, si on utilise seulement des inconnues dont l'arité est bornée par un entier k, comme il n'y a pas de variables actives dans le membre gauche de l'équation, on obtient que l'équation de filtrage peut se résoudre en $\mathcal{O}(|t||u|^k)$. De plus si on déroule l'algorithme dans ce cas, on obtient une approche similaire à l'algorithme proposé par [SSS01] pour résoudre les équations linéaires de filtrage de contextes.

La démarche que nous avons adoptée avec les descriptions syntaxiques n'est sans doute pas particulière au λ-calcul linéaire et semble pouvoir s'étendre au filtrage dans d'autres calculs. On pourrait par exemple l'utiliser pour concevoir des algorithmes pour les problèmes de filtrage avec des contextes d'arbres d'ordre supérieur. Pour essayer de résoudre avec une technique similaire des équations de filtrage non linéaires, nous avons envisagé la possibilité d'utiliser un type intersection, mais cela reste à explorer. La théorie des descriptions pourrait également être étendue au λ-calcul simplement typé et aux équations d'unification. Pour cela, il suffit de briser la distinction entre membre gauche et membre droit induite par le filtrage et autoriser que les descriptions dépendent les unes des autres ainsi qu'une règle de symétrie au niveau des égalités. Ce champ d'investigation permettrait éventuellement de caractériser de nouvelles classes d'équations de filtrage ou d'unification que l'on pourrait résoudre efficacement.

Si pour tout $i \in [1,n]$, $\sigma_i;\Pi_i;\Gamma_i;\Gamma'_i;\Delta_i \vdash_D t_i : \{u_i : \alpha_i\} \in H$, si $t = C[a\overline{t_{[1,n]}}]$ si $u = C'[a\overline{u_{[1,n]}}]$ et si a a pour type $\overline{\alpha_{[1,n]}} \multimap \alpha$ avec α atomique alors $H \rightarrow_{eq} S$ avec :

$$S = \overline{\sigma_{[1,n]}};\overline{\Pi_{[1,n]}};\overline{\Gamma_{[1,n]}};\overline{\Gamma'_{[1,n]}};\overline{\Delta_{[1,n]}} \vdash_D a\overline{t_{[1,n]}} : \{a\overline{u_{[1,n]}} : \alpha\}$$

Si pour tout $i \in [1,n]$, $\sigma_i;\Pi_i;\Gamma_i;\Gamma'_i;\Delta_i \vdash_D t_i : \{u_i : \alpha_i\} \in H$, si $t = C[x\overline{t_{[1,n]}}]$ si $u = C'[y\overline{u_{[1,n]}}]$, si x et y ont pour type $\overline{\alpha_{[1,n]}} \multimap \alpha$ avec α atomique et si $I^t_x = \{x\}$ alors $H \rightarrow_{eq} S$ avec :

$$S = \overline{\sigma_{[1,n]}}, x := y;\overline{\Pi_{[1,n]}};\overline{\Gamma_{[1,n]}};\overline{\Gamma'_{[1,n]}}, y : \overline{\alpha_{[1,n]}} \multimap \alpha;\overline{\Delta_{[1,n]}} \vdash_D x\overline{t_{[1,n]}} : \{y\overline{u_{[1,n]}} : \alpha\}$$

Si $\sigma, y := x;\Pi;\Gamma;\Gamma', y : \alpha;\Delta \vdash_D v : \{w : \beta\} \in H$, si $t = C[\lambda x.v]$, si $u = C'[\lambda y.w]$ et si $\mathscr{D}(\sigma) = \overline{\Gamma'}$ alors $H \rightarrow_{eq} S$ avec :

$$S = \sigma;\Pi;\Gamma;\Gamma';\forall y : \alpha.\Delta \vdash_D \lambda x.v : \{\lambda y.w : \alpha \multimap \beta\} \in H$$

Si $\sigma;\Pi;\Gamma;\Gamma', y : \alpha;\Delta, x : d \vdash_D v : e \in H$, si $t = C[\lambda x.v]$ alors $H \rightarrow_{eq} S$ avec :

$$S = \sigma;\Pi;\Gamma;\Gamma';\Delta \vdash_D \lambda x.v : d \multimap e$$

Si pour tout $i \in [1,n]$, $\sigma_i;\Pi_i;\Gamma_i;\Gamma'_i;\Delta_i \vdash_D t_i : d_i \in H$, si $t = C[\mathbf{X}\overline{t_{[1,n]}}]$ si $u = C'[w]$, si $\mathbf{X}$ a pour type $\overline{\alpha_{[1,n]}} \multimap \alpha$ avec α atomique, si w est un terme de type α et si $\overline{d_{[1,n]}} \vdash_{Ld} \{w : \alpha\}$ est dérivable alors $H \rightarrow_{eq} S$ avec :

$$S = \overline{\sigma_{[1,n]}};\overline{\Pi_{[1,n]}};\overline{\Gamma_{[1,n]}};\overline{\Gamma'_{[1,n]}};\overline{\Delta_{[1,n]}} \vdash_D \mathbf{X}\overline{t_{[1,n]}} : \{w : \alpha\}$$

Si pour tout $i \in [1,n]$, $\sigma_i;\Pi_i;\Gamma_i;\Gamma'_i;\Delta_i \vdash_D t_i : d_i \in H$, si $t = C[x\overline{t_{[1,n]}}]$ si $u = C'[w]$, si $I^t_x = \{x\}$, si x a pour type $\overline{\alpha_{[1,n]}} \multimap \alpha$ avec α atomique, si w est un terme de type α et si $\overline{d_{[1,n]}} \vdash_{Ld} \{w : \alpha\}$ est dérivable alors $H \rightarrow_{eq} S$ avec :

$$S = \overline{\sigma_{[1,n]}};\overline{\Pi_{[1,n]}};\overline{\Gamma_{[1,n]}};\overline{\Gamma'_{[1,n]}};\overline{\Delta_{[1,n]}}, x : \overline{d_{[1,n]}} \multimap \{w : \alpha\} \vdash_D x\overline{t_{[1,n]}} : \{w : \alpha\}$$

FIG. 5.5 – Algorithme de résolution d'équations de filtrage

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x:\alpha;\cdot\vdash_D e \quad x\notin\mathcal{S}_e}{\cdot;\cdot\vdash_{ld}\forall x.e \qquad \cdot;\{b(\lambda x.cxa)\})\vdash_{ld}\{b(\lambda x.cxa)\}}}{\cdot;(\forall x.e\multimap\{b(\lambda x.cxa)\})\vdash_{ld}\{b(\lambda x.cxa)\}}}{\cdot;\cdot\vdash_{ld}(\forall x.e\multimap\{b(\lambda x.cxa)\})\multimap\{b(\lambda x.cxa)\}}}{\cdot;\mathbf{X}:\beta;\cdot;\cdot;\cdot\vdash_{eq}\mathbf{X}:(\forall x.e\multimap\{b(\lambda x.cxa)\})\multimap\{b(\lambda x.cxa)\}} \quad \cdot;\mathbf{Z}:\alpha^2\multimap\alpha;\cdot;\cdot;\cdot\vdash_{eq}\lambda z_1.b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2):(\forall x.e)\multimap\{b(\lambda x.cxa)\}}{\cdot;\mathbf{X}:\beta,\mathbf{Z}:\alpha^2\multimap\alpha;\cdot;\cdot;\cdot\vdash_{eq}\mathbf{X}(\lambda z_1.b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2)):\{b(\lambda x.cxa)\}}$$

$$e=(\{a\}\multimap\{x\}\multimap\{cxa\})\multimap\{x\}\multimap\{cxa\}$$

$$\beta=(((\alpha^2\multimap\alpha)\multimap\alpha\multimap\alpha)\multimap\alpha)\multimap\alpha$$

FIG. 5.6 – Démonstration du séquent $\cdot;\mathbf{X}:\beta,\mathbf{Z}:\alpha^2\multimap\alpha;\cdot;\cdot;\cdot\vdash_{eq}\mathbf{X}(\lambda z_1.b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2)):\{b(\lambda x.cxa)\}$

$$\dfrac{\cdot;\cdot \vdash_{ld} c \qquad x:\alpha;\{x\} \vdash_{ld} \{x\}}{x:\alpha;\{x\} \vdash_{ld} \{cx\}}$$

$$\dfrac{x:\alpha;\{x\} \vdash_{ld} \{cx\} \qquad \cdot;\{a\} \vdash_{ld} \{a\}}{x:\alpha;\{a\},\{x\} \vdash_{ld} \{cxa\}}$$

$$\dfrac{x:\alpha;\{a\},\{x\} \vdash_{ld} \{cxa\}}{x:\alpha;\{a\} \vdash_{ld} \{x\} \multimap \{cxa\}}$$

$$\dfrac{x:\alpha;\{a\} \vdash_{ld} \{x\} \multimap \{cxa\}}{x:\alpha;\cdot \vdash_{ld} \{a\} \multimap \{x\} \multimap \{cxa\}}$$

$$\dfrac{x:\alpha;\cdot \vdash_{ld} \{a\} \multimap \{x\} \multimap \{cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;\cdot \vdash_{eq} \mathbf{Z}:\{a\} \multimap \{x\} \multimap \{cxa\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;\cdot \vdash_{eq} \mathbf{Z}:\{a\} \multimap \{x\} \multimap \{cxa\} \qquad \cdot;\cdot;\cdot;\cdot;x_1:\{a\} \vdash_{eq} x_1:\{a\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\} \vdash_{eq} \mathbf{Z}x_1:\{x\} \multimap \{cxa\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\} \vdash_{eq} \mathbf{Z}x_1:\{x\} \multimap \{cxa\} \qquad \cdot;\cdot;x:\alpha;\cdot;\cdot \vdash_{eq} x_2:\{x\} \vdash_{eq} x_2:\{x\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\},x_2:\{x\} \vdash_{eq} \mathbf{Z}x_1x_2:\{cxa\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\},x_2:\{x\} \vdash_{eq} \mathbf{Z}x_1x_2:\{cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\} \vdash_{eq} \lambda x_2.\mathbf{Z}x_1x_2:\{x\} \multimap \{cxa\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;x_1:\{a\} \vdash_{eq} \lambda x_2.\mathbf{Z}x_1x_2:\{x\} \multimap \{cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;\cdot \vdash_{eq} \lambda x_1x_2.\mathbf{Z}x_1x_2:\{a\} \multimap \{x\} \multimap \{cxa\}}$$

$$\dfrac{x:\alpha;\cdot \vdash_{ld} e}{\cdot;\cdot;x:\alpha;\cdot;z_1:e \vdash_{eq} z_1:e}$$

$$\dfrac{\cdot;\cdot;x:\alpha;\cdot;z_1:e \vdash_{eq} z_1:e \qquad \cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;\cdot \vdash_{eq} \lambda x_1x_2.\mathbf{Z}x_1x_2:\{a\} \multimap \{x\} \multimap \{cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;z_1:e \vdash_{eq} z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2):\{x\} \multimap \{cxa\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;x:\alpha;\cdot;z_1:e \vdash_{eq} z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2):\{x\} \multimap \{cxa\} \qquad z_2 := x;\cdot;\cdot;x:\alpha;\cdot;\cdot \vdash_{eq} z_2:\{x\}}{z_2 := x;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;x:\alpha;z_1:e \vdash_{eq} z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2:\{cxa\}}$$

$$\dfrac{z_2 := x;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;x:\alpha;z_1:e \vdash_{eq} z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2:\{cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;z_1:\forall x.e \vdash_{eq} \lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2:\{\lambda x.cxa\}}$$

$$\dfrac{\cdot;\cdot;\cdot;\cdot;\cdot \vdash_{eq} b:\{b:\alpha \multimap \alpha\} \qquad \cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;z_1:\forall x.e \vdash_{eq} \lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2:\{\lambda x.cxa\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;z_1:\forall x.e \vdash_{eq} b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2):\{b(\lambda x.cxa)\}}$$

$$\dfrac{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;z_1:\forall x.e \vdash_{eq} b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2):\{b(\lambda x.cxa)\}}{\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;\cdot \vdash_{eq} \lambda z_1.b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2):(\forall x.e) \multimap \{b(\lambda x.cxa)\}}$$

$$e = (\{a\} \multimap \{x\} \multimap \{cxa\}) \multimap \{x\} \multimap \{cxa\}$$

FIG. 5.7 – Démonstration du séquent $\cdot;\mathbf{Z}:\alpha^2 \multimap \alpha;\cdot;\cdot;\cdot \vdash_{eq} \lambda z_1.b(\lambda z_2.z_1(\lambda x_1x_2.\mathbf{Z}x_1x_2)z_2):(\forall x.e) \multimap \{b(\lambda x.cxa)\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdot;\cdot \vdash_{ld} \{a\} \qquad \dfrac{x:\alpha;\{x\} \vdash_{ld} \{x\} \quad x:\alpha;\{cxa\} \vdash_{ld} \{cxa\}}{x:\alpha;\{x\} \multimap \{cxa\},\{x\} \vdash_{ld} \{cxa\}}}{x:\alpha;\{a\} \multimap \{x\} \multimap \{cxa\},\{x\} \vdash_{ld} \{cxa\}}}{x:\alpha;\{a\} \multimap \{x\} \multimap \{cxa\} \vdash_{ld} \{x\} \multimap \{cxa\}}}{x:\alpha;\cdot \vdash_{ld} e}$$

$$e = (\{a\} \multimap \{x\} \multimap \{cxa\}) \multimap \{x\} \multimap \{cxa\}$$

FIG. 5.8 – Démonstration du séquent $x:\alpha;\cdot \vdash_{ld} (\{a\} \multimap \{x\} \multimap \{cxa\}) \multimap \{x\} \multimap \{cxa\}$

Chapitre 6

Décidabilité et complexité de l'analyse

La décidabilité de l'analyse des grammaires catégorielles abstraites est un problème qui résiste encore. Dans ce chapitre, nous mettons en rapport ce problème avec celui de la décidabilité de **MELL**, problème encore ouvert. Nous montrons ensuite que si on se limite à des grammaires catégorielles abstraites lexicalisées ou encore aux grammaires semi-lexicalisées le problème de l'analyse est décidable. Nous étudions également le problème de la vacuité du langage d'une grammaire catégorielle abstraite. Finalement, le reste du chapitre est consacré à l'étude de la complexité de l'analyse des grammaires catégorielles abstraites lexicalisées et semi-lexicalisées. Cette étude fixe les caractéristiques de complexité de l'analyse en s'appuyant sur la hiérarchie $\mathcal{L}(n, m)$.

6.1 Analyse des gammaires catégorielles abstraites et décidabilité

6.1.1 Analyse dans le cas général

Le fait que les constantes abstraites, briques de base pour la construction d'analyses, peuvent avoir un type abstrait d'ordre supérieur et sont transformées par le lexique en λ-termes linéaires dont le type peut avoir un ordre plus élevé encore permet d'avoir un contrôle très précis sur la structure des termes du langage décrit. L'influence de l'ordre de ces types sur la capacité des grammaires catégorielles abstraites à structurer les langages est illustrée par la façon dont on parvient à coder les langages réguliers, les langages hors contexte et les grammaires d'arbres adjoints. Dans le cas des langages réguliers, il suffit que les types atomiques abstraits se réalisent en des types objet atomiques, afin de passer aux grammaires hors contexte il faut que les types atomiques abstraits se réalisent en des types objet du deuxième ordre, finalement transformer les types atomiques abstraits en type objet du troisième ordre permet d'obtenir les grammaires d'arbres adjoints. On peut même dédier certaines constantes abstraites du langage au contrôle de la structure des termes du langage en faisant en sorte qu'ils ne contiennent pas de constantes objet. De plus, l'utilisation de l'ordre supérieur au niveau abstrait permet également de fortement structurer des termes d'analyse et donc le langage objet. Cette capacité de structuration a un coût ; en particulier l'analyse, si elle est décidable, a une complexité démesurée. Nous allons montrer que si l'analyse des grammaires catégorielles abstraites est décidable, alors **MELL** est décidable.

Lemme 6.1 *Soit $\mathcal{A}$ un ensemble de types atomiques, et φ une fonction de $\mathcal{I}_\mathcal{A}$ dans $\mathcal{I}_{\{\iota\}}$ définie par induction sur la structure des types de $\mathcal{I}_\mathcal{A}$ comme il suit :*

$$\varphi(\alpha) = \iota \multimap \iota \ \forall \alpha \in \mathcal{A}$$
$$\varphi(\alpha_1 \multimap \alpha_2) = \varphi(\alpha_1) \multimap \varphi(\alpha_2) \ \forall \alpha_1, \alpha_2 \in \mathcal{I}_\mathcal{A}$$

*alors quel que soit $\alpha \in \mathcal{I}_\mathcal{A}$, $\varphi(\alpha)$ est une tautologie de **IILL**.*

Démonstration. Pour démontrer ce lemme, nous allons démontrer une propriété plus forte. Nous allons montrer qu'étant donné un séquent $\mathcal{S}$ on a :

1. si $\mathcal{S} = \varphi(\alpha_1), \ldots, \varphi(\alpha_n), \iota \vdash \iota$ alors S est dérivable

2. si $\mathcal{S} = \cdot \vdash \varphi(\alpha)$ alors S est dérivable.

On démontre cette propriété par induction sur $\rho(\mathcal{S})$.

Cas 1: $\rho(\mathcal{S}) = 0$

Dans ce cas, on a forcément $S = \iota \vdash \iota$ qui est un axiome.

Cas 2: $\rho(\mathcal{S}) > 0$ et $\mathcal{S} = \cdot \vdash \varphi(\alpha)$

Dans ce cas, si α est atomique alors $\varphi(\alpha) = \iota \multimap \iota$ est donc $\mathcal{S}$ est dérivable. Si $\alpha = \alpha_1 \multimap \dots \alpha_n \multimap \beta$ avec β atomique alors comme par hypothèse d'induction, $\varphi(\alpha_1), \dots, \varphi(\alpha_n), \iota \vdash \iota$ est dérivable, on obtient la dérivabilité de $\mathcal{S}$.

Cas 3: $\rho(\mathcal{S}) > 0$ et $\mathcal{S} = \varphi(\alpha_1), \dots, \varphi(\alpha_n), \iota \vdash \iota$

Dans ce cas, si α_n est une formule atomique alors $\varphi(\alpha_n) = \iota \multimap \iota$ et comme par hypothèse d'induction $\varphi(\alpha_1), \dots, \varphi(\alpha_{n-1}), \iota \vdash \iota$ est dérivable, on obtient une dérivation de $\mathcal{S}$ de la façon suivante :

$$\frac{\varphi(\alpha_1),\ \dots,\ \varphi(\alpha_{n-1}),\ \iota \vdash \iota \quad \iota \vdash \iota}{\varphi(\alpha_1),\ \dots,\ \varphi(\alpha_{n-1}),\ \iota \multimap \iota,\ \iota \vdash \iota}$$

Si en revanche $\alpha_n = \beta_1 \multimap \beta_2$, par hypothèse d'induction, nous avons la dérivabilité de

$$\varphi(\alpha_1), \dots, \varphi(\alpha_{n-1}), \varphi(\beta_2), \iota \vdash \iota$$

et de $\cdot \vdash \varphi(\beta_1)$. On peut donc conclure en utilisant la dérivation suivante :

$$\frac{\varphi(\alpha_1), \dots, \varphi(\alpha_{n-1}), \varphi(\beta_2), \iota \vdash \iota \quad \cdot \vdash \varphi(\beta_1)}{\varphi(\alpha_1), \dots, \varphi(\alpha_{n-1}), \varphi(\beta_1) \multimap \varphi(\beta_2), \iota \vdash \iota}$$

□

Nous allons maintenant montrer que si l'analyse des grammaires catégorielles abstraites est décidable alors **MELL** est décidable.

Proposition 6.1 *$\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ étant une grammaire catégorielle abstraite, la décidabilité de l'appartenance à $\mathcal{L}(\mathcal{G})$ d'un terme u implique la décidabilité de **MELL**.*

Démonstration. Nous avons vu que la décidabilité de **MELL** était équivalente à celle de la dérivabilité dans **sIIELL** des séquents de la forme $!\Gamma \vdash \beta$ où β est atomique.

Étant donné un ensemble de formules atomiques $\mathcal{A}$ et un séquent :

$$\mathcal{S} = !\alpha_1,\ \dots,\ !\alpha_n \vdash \gamma$$

tel que pour tout $i \in [1, n]$, $\alpha_i \in \mathcal{I}_\mathcal{A}$ et $\gamma \in \mathcal{A}$, nous allons construire une grammaire catégorielle abstraite $\mathcal{G}_\mathcal{S}$ et un terme objet de cette grammaire $u_\mathcal{G}$ tels que $u_\mathcal{G}$ appartient au langage de $\mathcal{G}_\mathcal{S}$ si et seulement si $\mathcal{S}$ est démontrable.

On pose $\Sigma_{\mathcal{S},1} = (C_1, \mathcal{A}, \tau_1)$ avec :
- $C_1 = \{g_1, \dots, g_n\}$
- τ_1 est tel que $\tau_1(g_i) = \alpha_i$ pour $i \in [1, n]$

On pose $\Sigma_{\mathcal{S},2} = (C_2, A_2, \tau_2)$ avec :
- $A_2 = \{\iota\}$
- $C_2 = \emptyset$

$\mathcal{L}$ est défini par :
- pour tout $\alpha \in \mathcal{A}$, $\mathcal{L}_\mathcal{S}(\alpha) = \iota \multimap \iota$
- comme, d'après le lemme 6.1, $\mathcal{L}_\mathcal{S}(\alpha_i)$ est une tautologie quel que soit $i \in [1, n]$, on peut donc construire un λ-terme non-lexicalisé t_i de type $\mathcal{L}(\alpha_i)$ dans $\Sigma_{\mathcal{S},2}$. On pose $\mathcal{L}_\mathcal{S}(g_i) = t_i$ pour l'un de ces t_i.

Finalement, nous posons $\mathcal{G}_{\mathcal{S}} = (\Sigma_{\mathcal{S},1}, \Sigma_{\mathcal{S},2}, \mathcal{L}_{\mathcal{S}}, \gamma)$ et $u_{\mathcal{S}} = \lambda x.x$. Si il existe un terme abstrait t de type γ tel que $\mathcal{L}_{\mathcal{S}}(t) =_{\beta\eta} u_{\mathcal{S}}$, alors d'après la correspondance de Curry-Howard, c'est que le séquent $!\alpha_1, \ldots, !\alpha_n \vdash \gamma$ est dérivable. Réciproquement, si le séquent $!\alpha_1, \ldots, !\alpha_n \vdash \gamma$ alors il existe un terme t de type γ construit sur $\Sigma_{\mathcal{S},1}$, le terme objet $\mathcal{L}_{\mathcal{S}}(t)$ est un terme de type $\iota \multimap \iota$ qui ne contient pas de constante on a donc $\mathcal{L}_{\mathcal{S}}(t) =_{\beta\eta} \lambda x.x = u_{\mathcal{S}}$. □

La recherche de démonstration dans **MELL** permet simplement de coder le problème de l'accessibilité dans les réseaux de Petri. Ce problème est décidable, mais sa complexité est telle que l'on peut plutôt le considérer comme *non-indécidable*. Il serait utopique d'espérer pouvoir le résoudre en un temps raisonnable. Aussi si **MELL** s'avérait décidable, la complexité de la recherche de démonstration serait trop importante pour qu'elle puisse être effective. On peut considérer que l'analyse des grammaires catégorielles abstraites est au mieux *non-indécidable* au même titre que l'accessibilité dans les réseaux de Pétri.

Nous avions conjecturé que la réciproque (c'est-à-dire, le fait que la décidabilité de **MELL** implique celle de l'analyse de grammaires catégorielles abstraites) était vraie, mais nous n'étions pas parvenus à le démontrer. Récemment, en collaboration avec Matoko Kanazawa, Philippe de Groote et Ryo Yoshinaka, ce résultat a pu être démontré.

6.1.2 Analyse dans le cas lexicalisé

Il reste cependant que le but premier des grammaires catégorielles abstraites est de modéliser les langues naturelles. Or la bonne construction des phrases d'une langue tient essentiellement aux mots eux-mêmes et aux interraction qu'ils entretiennent les uns avec les autres. Ainsi pour modéliser les langues naturelles, n'y a-t-il besoin que de "grammaires lexicalisées". Ces grammaires sont celles dont toutes les constantes abstraites sont lexicalisées. De manière assez évidente, l'analyse de ce type de grammaires catégorielles abstraites est décidable.

Proposition 6.2 *L'analyse des grammaires catégorielles abstraites lexicalisées est décidable.*

Démonstration. Soient $\Sigma_1 = (C_1, A_1, \tau_1)$ et $\Sigma_2 = (C_2, A_2, \tau_2)$, deux signatures et $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$ une grammaire catégorielle abstraite lexicalisée, et u un terme de Σ_2 de type $\mathcal{L}(S)$, pour montrer la décidabilité de l'analyse dans $\mathcal{G}$, il nous faut remarquer que si il existe un terme t de type S tel que $\mathcal{L}(t) =_{\beta\eta} u$ alors, t contient moins de constantes que u. En bornant le nombre de constantes que peut contenir un terme abstrait qui serait une analyse de u, on borne la taille d'un tel terme de sorte que l'espace de recherche pour l'analyse de u est fini. Il s'ensuit qu'une exploration systématique de cet espace permet de vérifier si u a une analyse ou non. L'analyse des grammaires lexicalisées est par conséquent décidable. □

6.1.3 Analyse dans le cas semi-lexicalisé

Il n'en demeure pas moins vrai que les entrées non-lexicalisées peuvent jouer un rôle intéressant du point de vue linguistique. Elles permettent de décrire des schémas généraux et les structures récurrentes d'une langue. On voudrait, par exemple, pouvoir définir en français une phrase simple comme la juxtaposition d'un groupe nominal sujet, d'un groupe verbal et d'un groupe nominal complément sans s'attarder sur les détails de la nature de ces constituants et sans que cela rende indécidable ou insurmontable l'analyse. On voudrait pouvoir utiliser des définitions simples comme celle qui suit :

$$P_{simp} := \lambda svc.s + v + c : NP \multimap V \multimap NP \multimap S$$

Autoriser ce genre de définitions risque de nous faire retomber dans le cadre des grammaires catégorielles abstraites générales dont la décidabilité reste indéterminée, mais dont la complexité, même si il s'avérait que l'analyse soit décidable, serait rédhibitoire. Comme nous l'avons vu, le problème induit par les entrées non-lexicalisées est qu'elles rendent l'analyse au moins aussi difficile que la dérivabilité dans **MELL**. Un fragment décidable de **MELL** est celui constitué des séquents de la forme $!\alpha_1, \ldots, !\alpha_n, \beta_1, \ldots, \beta_m \vdash \gamma$ où les α_i sont des formules implicatives au plus d'ordre 2, où les β_i sont des

formules implicatives et où γ est atomique. Pour propager cette restriction au niveau des grammaires catégorielles abstraites, nous allons utiliser des grammaires *semi-lexicalisées*, c'est-à-dire dont les constantes abstraites non-lexicalisées sont au plus d'ordre 2. Comme l'analyse des grammaires catégorielles abstraites est équivalente à **MELL**, il semble raisonnable de penser que la décidabilité de ce fragment de **MELL** entraîne celle de l'analyse des grammaires semi-lexicalisées.

Pour analyser un terme d'une grammaire semi-lexicalisée, nous procédons en deux temps :

1. tout d'abord, on sélectionne des entrées lexicalisées $a_1 : \alpha_1, \ldots, a_n : \alpha_n$ tel qu'il existe un terme non-lexicalisé s tel que $s\mathcal{L}(a_1)\ldots\mathcal{L}(a_n) =_{\beta\eta} u$;
2. on vérifie qu'il existe un terme t de type $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap S$ construit sur la signature abstraite tel que $\mathcal{L}(t) =_{\beta\eta} s$.

La première étape, le choix des constantes lexicalisées, donne lieu à un nombre fini de possibilités. En effet, le nombre de choix pour une sélection de constantes lexicalisées $a_1 : \alpha_1, \ldots, a_n : \alpha_n$ telle qu'il existe un terme non-lexicalisé s tel que $s\mathcal{L}(a_1)\ldots\mathcal{L}(a_n) =_{\beta\eta} u$ est fini. De plus, pour chacune de ces sélections, il n'existe qu'un nombre fini de λ-termes s. Si il existe une méthode qui permette systématiquement de vérifier, étant donné un terme non-lexicalisé s et un type α, si il existe un terme abstrait t de type α tel que $\mathcal{L}(t) =_{\beta\eta} s$, alors il nous est possible de l'appliquer à chacun des s possibles afin de vérifier si l'un d'entre eux n'est pas l'image par le lexique d'un terme abstrait ayant un type approprié. Si cela est le cas, alors on peut en conclure que u est un terme du langage ; dans le cas contraire, on peut bien évidemment en conclure que u n'est pas un terme du langage.

Nous allons maintenant exhiber un algorithme qui, étant donnée une grammaire semi-lexicalisée $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$, un terme non-lexicalisé de sa signature objet s et un type abstrait α permet de vérifier si il existe un terme abstrait t de type α tel que $\mathcal{L}(t) =_{\beta\eta} s$. On peut immédiatement remarquer que les constantes de la signature abstraite qui interviendraient dans un tel λ-terme t sont forcément non-lexicalisées.

On définit alors deux ensembles T et $\mathcal{S}_\beta$, le premier représente les types que l'on peut associer à un terme abstrait qui serait sous-terme d'un terme t tel que $\mathcal{L}(t) =_{\beta\eta} s$ et le second $\mathcal{S}_\beta$ est l'ensemble des termes objets qui pourraient être l'image par le lexique d'un sous-terme de t de type β.

$$T = \{\beta \mid \beta \text{ est une sous-formule de } \alpha \text{ ou un type atomique de } A_1\}$$

$$\mathcal{S}_\beta = \{v \in \mathcal{T}_{\Sigma_2} | \exists w.\exists C[].C[wv] =_{\beta\eta} s \text{ et } v : \mathcal{L}(\beta)\}$$

Pour tout β, $\mathcal{S}_\beta$ est un ensemble fini. En effet, pour chaque sous-terme s' de s, il existe un ensemble fini de solutions à l'équation $\mathbf{XY} =_{\beta\eta} s'$. Comme les termes abstraits auxquels nous nous intéressons sont des sous-termes éventuels d'une analyse de s et qu'ils ont, comme nous l'avons remarqué, un type contenu dans T, nous allons seulement considérer les ensembles $\mathcal{S}_\beta$ tels que $\beta \in T$. On pose donc :

$$\mathcal{S} = \bigcup_{\beta \in T} \mathcal{S}_\beta$$

en tant que réunion finie d'ensembles finis, $\mathcal{S}$ est un ensemble fini.

Parmi les termes qui appartiennent à $\mathcal{S}$, nous allons chercher à repérer ceux qui sont l'image par le lexique d'un terme abstrait pour essayer de manière itérative de construire une analyse de s. Pour cela on procède par saturation d'un ensemble de termes $\mathcal{H}$. Dans ce qui suit, C_1' représente l'ensemble des constantes abstraites non-lexicalisées de Σ_1 ; d'après les hypothèses, elles sont au plus d'ordre 2 :

1. si il existe $\alpha \in T$ tel que la λ-variable $x \in \mathcal{S}_\alpha$ alors $\mathcal{H} := \mathcal{H} \cup \{x, \alpha\}$
2. si il existe $a \in C_1'$ tel que $a : \overline{\alpha_{[1,n]}} \multimap \beta$ et si $\{(s_1, \alpha_1); \ldots; (s_n, \alpha_n)\} \subseteq \mathcal{H}$ et si $\mathcal{L}(a)\overline{s_{[1,n]}} \in \mathcal{S}$ alors $\mathcal{H} := \mathcal{H} \cup \{(\mathcal{L}(a)\overline{s_{[1,n]}}, \beta)\}$
3. si $(x, \overline{\alpha_{[1,n]}} \multimap \beta) \in \mathcal{H}$, si $\{(v_1, \alpha_1); \ldots; (v_n, \alpha_n)\} \in \mathcal{H}$ et si $x\overline{v_{[1,n]}} \in \mathcal{S}$ alors $\mathcal{H} := \mathcal{H} \cup \{x\overline{v_{[1,n]}}, \beta\}$
4. si $(t : \alpha_2) \in \mathcal{H}$, si $x : \alpha_1 \in \mathcal{H}$ si $x \in FV(t)$ et si $\lambda x.t \in \mathcal{S}$ alors $\mathcal{H} := \mathcal{H} \cup \{(\lambda x.t, \alpha_1 \multimap \alpha_2)\}$

Le fait que $\mathcal{S}$ soit fini nous permet de mener la saturation de $\mathcal{H}$ jusqu'à son terme. On appelle $\mathcal{H}_{fin}$ le sous-ensemble de $\mathcal{S}$ obtenu par cette procédure. La propriété essentielle de $\mathcal{H}_{fin}$ est la suivante :

Proposition 6.3 *Étant donnés $v \in \mathcal{S}$ et $\alpha \in T$, $(v, \alpha) \in \mathcal{H}_{fin}$ si et seulement si les conditions suivantes sont remplies :*

1. *il existe un terme abstrait t_v de type α tel que $\mathcal{L}(t_v) =_{\beta\eta} v$.*
2. *les variables libres contenues dans t_v ont un type qui appartient à T.*

Démonstration. Pour montrer la première implication, on procède par induction sur la saturation de $\mathcal{H}$. Si $(v, \alpha) \in \mathcal{H}_{fin}$ alors lors de la saturation de $\mathcal{H}_{fin}$ l'une des règles a permis de l'introduire dans $\mathcal{H}$:

1. si v est une λ-variable x ayant pour type $\mathcal{L}(\alpha)$ alors le terme abstrait $x : \alpha$ convient.
2. si il existe $a \in C_1'$ et $v_1, \ldots, v_n$ tels que $a : \overline{\alpha_{[1,n]}} \multimap \beta$, $\{(v_1, \alpha_1); \ldots; (v_n, \alpha_n)\} \in \mathcal{H}$ et $\mathcal{L}(a)\overline{v_{[1,n]}} =_{\beta\eta} v$ alors par hypothèse d'induction, il existe $t_{v_i} : \alpha_i$ tels que $\mathcal{L}(t_{v_i}) =_{\beta\eta} t_i$. Ainsi, en prenant $t_v = at_{v_1} \ldots t_{v_n}$ on a bien un terme abstrait ayant les propriétés requises.
3. si $\{(x, \overline{\alpha_{[1,n]}} \multimap \beta); (v_1, \alpha_1); \ldots; (v_n : \alpha_n)\} \subseteq \mathcal{H}$ et si $v = x\overline{v_{[1,n]}}$ alors par hypothèse il existe t_{v_1}, $\ldots$, t_{v_n} tels que $\mathcal{L}(t_{v_i}) =_{\beta\eta} v_i$. Poser $t_v = xt_{v_1} \ldots t_{v_n}$ permet d'obtenir un terme abstrait de type β et dont l'image par $\mathcal{L}$ est v.
4. si $\{(v_1, \alpha_2); (x, \alpha_1)\} \subseteq \mathcal{H}$, $\alpha = \alpha_1 \multimap \alpha_2$ et $v = \lambda x.v_1$ alors par hypothèse d'induction il existe t_{v_1} tel que $\mathcal{L}(t_{v_1}) =_{\beta\eta} v_1$. Comme x est libre dans v_1, x est libre dans t_{v_1} et donc $t_v = \lambda x.t_{v_1}$ est un λ-terme linéaire de la signature abstraite, ayant pour type α et tel que $\mathcal{L}(t_v) =_{\beta\eta} v$

Si en revanche il existe un terme $v \in \mathcal{S}$ tel qu'il existe un terme abstrait t_v de type α tel que $\mathcal{L}(t_v) =_{\beta\eta} v$. Nous allons montrer par induction sur la structure de t_v que $(v, \alpha) \in \mathcal{H}_{fin}$:

1. si $t_v = x$ alors comme le type de x est $\mathcal{L}(\alpha)$ $(x, \alpha) \in \mathcal{H}_{fin}$
2. si $t_v = a\overline{t_{[1,n]}}$ alors, comme $v \in \mathcal{S}$, il existe un terme w et un sous-terme s' de s tels que $wv =_{\beta\eta} s'$. Posons $v_i =_{\beta\eta} \mathcal{L}(t_i)$ et α_i le type de t_i (il faut remarquer que comme a est une constante abstraite non-lexicalisée, par hypothèse, α_i est un type atomique), pour pouvoir utiliser l'hypothèse d'induction, il nous faut montrer que $v_i \in \mathcal{S}$. Or si $u_i = \mathcal{L}(at_1 \ldots t_{i-1}xt_{i+1} \ldots t_n)$, il nous suffit de poser $w_i = \lambda x.(wu_i)$ pour avoir $w_i(\mathcal{L}(t_i)) =_{\beta\eta} s'$. De plus comme t_i est de type α_i, et que $\alpha_i \in T$, $v_i \in \mathcal{S}_{\alpha_i}$ et v_i est finalement un élément de $\mathcal{S}$. Par hypothèse d'induction, il vient que $(v_i, \alpha_i) \in \mathcal{H}_{fin}$ en conséquence de quoi on obtient que $(v, \alpha) \in \mathcal{H}_{fin}$.
3. si $t_v = x\overline{t_{[1,n]}}$ alors par hypothèse, x est une variable qui a un type dans T, et comme T est clos par sous-formule, $t_1, \ldots, t_n$ sont des termes qui ont pour types respectifs $\alpha_1, \ldots, \alpha_n$ et ces types sont dans T. On a donc $(x, \overline{\alpha_{[1,n]}} \multimap \beta) \in \mathcal{H}_{fin}$.Comme précédemment on montre que $t_i \in \mathcal{S}_{\alpha_i}$ et l'hypothèse d'induction implique que $(\mathcal{L}(t_i), \alpha_i)$ est dans $\mathcal{H}_{fin}$. Il s'ensuit qu'avec la règle 4 que $(v, \alpha) \in \mathcal{H}_{fin}$.
4. si $t_v = \lambda x.t'$, alors $\alpha = \alpha_1 \multimap \alpha_2$. Comme $\alpha \in T$ et comme T est clos par sous-formule, α_1 et α_2 sont dans T. De plus toutes les variables libres dans t ont un type appartenant à T et puisque $FV(t') = FV(t) \cup \{x\}$, les variables libres dans t' ont un type appartenant à T. Il s'ensuit, par hypothèse d'induction que $(\mathcal{L}(t'), \alpha_2)$ est dans $\mathcal{H}_{fin}$ et donc que $(\mathcal{L}(t), \alpha)$ y est également.

□

Ainsi vérifier l'existence d'un terme abstrait t de type α tel que $\mathcal{L}(t) =_{\beta\eta} s$ revient à vérifier que $(s, \alpha) \in \mathcal{H}_{fin}$. De là dérive la décidabilité de l'analyse des grammaires catégorielles abstraites semi-lexicalisées.

6.1.4 Vacuité du langage d'une grammaire catégorielle abstraite

Lorsque l'on définit une grammaire complexe se pose rapidement la question de la cohérence du travail que l'on a effectué. En particulier, de petites erreurs dans les définitions peuvent conduire à la définition d'un langage vide. Pour aider à la conception de grammaires à large couverture, il peut être intéressant de pouvoir automatiquement vérifier la vacuité du langage d'une grammaire catégorielle abstraite. Dans cette partie nous allons montrer que dans le cas général la décidabilité de cette vérification est équivalente à celle de **MELL**. Nous verrons également que déterminer la vacuité d'une grammaire de $\mathcal{L}(2, n)$ est décidable et même polynômial.

Théorème 6.1 *Soit $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ une grammaire catégorielle abstraite, on peut décider si il existe un λ-terme de Σ_2 qui appartient au langage de $\mathcal{G}$ si et seulement si **MELL** est décidable.*

Démonstration. Nous avons vu que la décidabilité de **MELL** était équivalente à celle des séquents de la forme $!\Gamma \vdash \gamma$ où Γ est formé seulement de formules implicatives et où γ est atomique. Vérifier si le langage d'une grammaire catégorielle abstraite $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ est vide revient à vérifier que l'on peut construire un terme de type S sur Σ_1 ce qui est équivalent à prouver un séquent de la forme $!\alpha_1, \ldots, !\alpha_n \vdash S$ où les α_i sont les types associés aux constantes de C_{Σ_1} par τ_{Σ_1}. □

Une fois de plus, nous nous trouvons face au problème si ce n'est de l'indécidabilité au moins d'une difficulté rédhibitoire. Cependant, dans le cas restreint où des grammaires catégorielles abstraites de $\mathcal{L}(2, n)$, nous pouvons décider en temps polynômial si leur langage est vide ou non.

Proposition 6.4 *La vacuité d'une ACG de $\mathcal{L}(2, n)$ est décidable en temps polynômial.*

Démonstration. Nous avons déjà vu que les termes clos ayant un type atomique et qui ne contiennent que des constantes du second ordre sont les éléments d'un langage régulier d'arbres. Or, la vacuité des langages réguliers d'arbres est décidable en $\mathcal{O}(n^3)$ [CDG+99]. □

6.2 Complexité de l'analyse

Dans cette partie nous allons traiter de la complexité de l'analyse des grammaires catégorielles abstraites lexicalisées. La complexité d'un problème dépend avant tout de ce que l'on considère comme une donnée du problème et de ce que l'on considère comme inhérent au problème. Pour les analyses grammaticales, cela donne lieu à deux types de problème d'analyse ; le problème de l'analyse de n'importe quelle grammaire et celui de l'analyse d'une grammaire particulière. De la sorte nous allons montrer que l'analyse des grammaires lexicalisée de $\mathcal{L}(2, 2)$ est NP-complète, puis que l'analyse d'une grammaire particulière de $\mathcal{L}(2, n)$ est polynômiale (et cela même si cette grammaire n'est pas lexicalisée), et finalement nous construirons une grammaire de $\mathcal{L}(3, 1)$ pour laquelle l'analyse est NP-complète. Afin de faire la part entre le problème de l'analyse grammaticale et celui de la sélection des items lexicaux nous étudierons le problème de l'existence de l'analyse d'une phrase pour le cas où les choix lexicaux ont déjà été effectués. Paradoxalement, nous verrons que dans ce cas, on obtient la NP-complétude de ce problème à partir des grammaires de $\mathcal{L}(2, 1)$.

La liste des problèmes d'analyse dont nous allons étudier la complexité sont les suivants :

1. **Analyse générale** : étant donnés une grammaire catégorielle abstraite $\mathcal{G}$ et un terme objet u, u est-il un élément du langage de $\mathcal{G}$?
2. **Analyse générale avec présélection lexicale** : étant donnés une grammaire catégorielle abstraite $\mathcal{G}$, un terme u et un ensemble de constantes abstraites $a_1, \ldots, a_n$, existe-t-il un terme abstrait contenant exclusivement les constantes $a_1, \ldots, a_n$ et dont l'image par le lexique serait u ?
3. **Analyse particulière** : pour une grammaire $\mathcal{G}$ fixée, étant donné un terme objet u, u est-il un élément du langage de $\mathcal{G}$?
4. **Analyse particulière avec présélection lexicale** : pour une grammaire $\mathcal{G}$ fixée, étant donnés un terme objet u et un ensemble de constantes abstraites $a_1, \ldots, a_n$, existe-t-il un terme abstrait contenant exclusivement les constantes $a_1, \ldots, a_n$ et dont l'image par le lexique serait u ?

On peut aisément se convaincre que tous ces problèmes sont dans NP si la grammaire est lexicalisée. Si un terme u qui contient n constantes objet est l'image par le lexique d'un terme t, comme toutes les grammaires considérées sont lexicalisées, le nombre de constantes contenues dans t est inférieur à n et donc t a une taille polynômiale dans la taille de u et des types des constantes qu'il contient. Par ailleurs, la vérification du fait qu'un terme t est une analyse d'un terme u revient à normaliser l'image de t par le lexique, ce qui est linéaire.

Ainsi, déterminer la complexité des problèmes que nous avons énoncés revient à déterminer si ils sont polynômiaux ou NP-complets. La distinction entre les problèmes que nous avons qualifiés de *généraux*

et les problèmes que nous avons qualifiés de *particuliers* est que dans les premiers, les grammaires font partie des données alors que dans les seconds elles n'en font pas partie. Ainsi dans le second cas, si on exhibe des algorithmes qui permettent d'analyser des termes objet et dont la complexité dépend exponentiellement de paramètres de la grammaire mais non du terme objet, alors, nous considérerons ses algorithmes comme étant polynômiaux tandis que dans le premier cas nous les aurions considérés exponentiels. Cette distinction provient du fait que lorsque l'on écrit la grammaire d'une certaine langue, nous nous intéressons à la complexité de l'analyse de cette langue en particulier, mais non de celle du formalisme en général.

Ainsi, bien que la dérivabilité d'un séquent dans le calcul de Lambek soit NP-complète, du fait que toute gammaire catégorielle est faiblement équivalente à une grammaire hors contexte, le problème de l'analyse d'une grammaire catégorielle particulière est polynômiale. Il suffit pour cela de construire la grammaire hors contexte qui lui est équivalente (la taille de cette grammaire peut bien entendu être exponentielle dans la taille de la grammaire de départ) et on obtient ainsi un moyen de vérifier en temps polynômial si une phrase admet une analyse ou non. Bien qu'il n'existe pas d'algorithmes qui analysent les grammaires catégorielles en temps polynômial, chaque grammaire possède un algorithme qui vérifie en temps polynômial (dans la taille de la phrase à analyser et exponentiel dans la taille de la grammaire) si une phrase donnée possède une analyse.

6.2.1 Analyse générale

Nous avons déjà vu que les grammaires de $\mathcal{L}(2,1)$ correspondaient exactement aux langages réguliers d'arbres. Comme il existe des algorithmes polynômiaux qui permettent de vérifier si un arbre appartient à un langage régulier, l'analyse des grammaires de $\mathcal{L}(2,1)$ est polynômiale. Dans cette partie nous allons montrer que pour les grammaires de $\mathcal{L}(2,2)$ le problème de **l'analyse générale** est NP-complet.

On prouve ici que l'analyse d'une grammaire catégorielle abstraite $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$ est NP-complète si les constantes de Σ_1 sont au plus d'ordre 2 et que l'image d'un type atomique de Σ_1 par $\mathcal{L}$ est un type d'ordre 2 de Σ_2.

Nous allons réduire le problème de l'analyse dans $\mathcal{L}(2,2)$ au problème X3C.

Définition 6.1 *Un problème X3C est une paire (X, B) telle que :*

1. *$X = \{a_1, \ldots, a_{3n}\}$ est un ensemble fini composé de $3n$ éléments*
2. *$B = \{B_1, \ldots, B_m\}$ est un ensemble de m sous-ensembles de trois éléments de X*
3. *$B_i = \{a_{i1}, a_{i2}, a_{i3}\}$ et on suppose que $i1 < i2 < i3$.*

Définition 6.2 *On dit qu'un problème X3C (X, B) admet une solution si et seulement si il existe $C \subseteq B$ et C est une partition de X.*

Théorème 6.2 *Savoir si un problème X3C admet une solution est NP-complet. [GJ79]*

Étant donnés $X = \{a_1, \ldots, a_{3n}\}$ et $B = \{B_1, \ldots, B_m\}$, on associe au problème X3C (X, B) la grammaire catégorielle abstraite $\mathcal{G}_{X,B} = (\Sigma_1, \Sigma_2, \mathcal{L}, D_0)$ définie de la manière suivante :

Définition de $\Sigma_1 = (C_1, A_1, \tau_1)$:

1. $A_1 = \{D_0, \ldots, D_n\}$;
2. E est une constante de type D_n ;
3. Pour $0 \leq k < n$, B_i et $k_1 < k_2 < k_3$ tels que $k_i \in \{1, \ldots, k\}$, on définit $E_{B_i,k_1,k_2,k_3,k}$ comme étant une constante de C_1 ayant pour type $D_{k+1} \multimap D_k$.

Définition de $\Sigma_2 = (C_2, A_2, \tau_2)$:

1. $A_2 = \{\iota\}$;
2. e est une consante de type $\iota^{3n} \multimap \iota$ de C_2 ;
3. a_i, avec $i \in \{1, \ldots, 3n\}$, est un constante de type ι de C_2.

Définition de $\mathcal{L}$:

1. $\mathcal{L}(D_k) = \iota^{3k} \multimap \iota$;
2. $\mathcal{L}(E) = \lambda x_1 \dots x_{3n}.e x_1 \dots x_{3n}$;
3. $\mathcal{L}(E_{B_i,k_1,k_2,k_3,k}) = \lambda g x_1 \dots x_{3k}.g \quad x_1 \dots x_{k_1-1} a_{i1}$
   $x_{k_1+1} \dots x_{k_2-1} a_{i2}$
   $x_{k_2+1} \dots x_{k_3-1} a_{i3}$
   $x_{k_3+1} \dots x_{3k}$

Nous allons montrer que $u = e a_1 \dots a_{3n}$ appartient au langage de $\mathcal{G}_{X,B}$ si et seulement si le problème X3C constitué de X et de B admet une solution.

Lemme 6.2 *Tout terme clos de Σ_1 de type D_k est de la forme $E_{n-k}(\dots(E_1 E)\dots)$ où E_i est une des constantes $E_{j_i,k_1^i,k_2^i,k_3^i,(n-i)}$.*

Démonstration. On procède par induction sur $n-k$. On a toujours $k \leq n$, ce qui permet de faire une induction effective sur $n-k$.

Dans le cas où $n-k=0$, $k=n$. Or la seule constante dont le type a pour conclusion D_n est E. Le seul terme clos de Σ_1 de type D_n est donc E.

Dans le cas où $n-k = n-k'+1 = n-(k'-1)$, on cherche à construire un terme de type $D_{k'-1}$. Les seules constantes de Σ_1 dont le type a pour conclusion $D_{k'-1}$ sont les $E_{B_i,k_1,k_2,k_3,(k'-1)}$. Les termes clos de Σ_1 ayant pour type $D_{k'-1}$ sont donc de la forme $E_{B_i,k_1,k_2,k_3,(k'-1)}(t)$ où t est un terme de type $D_{k'}$. Or $n-k' < n-k'+1$ donc $n-k' < n-k$, l'hypothèse d'induction peut être appliquée et nous donne le fait que tout terme clos de Σ_1 de type $D_{k'}$ est de la forme $E_{n-k'}(\dots(E_1 E)\dots)$ où pour tout $i \in [1, n-k']$, la constante E_i est l'une des constantes $E_{j_i,k_1^i,k_2^i,k_3^i,(n-i)}$. Il s'ensuit donc que tous les termes clos de Σ_1 de type $D_{k'-1}$ est de la forme $E_{n-k'+1}(E_{n-k}(\dots(E_1 E)\dots))$ où $E_{n-k'+1}$ est l'une des constantes $E_{B_i,k_1,k_2,k_3,(k'-1)}$; cela a pour conséquence que tout terme de Σ_1 de type $D_{k'-1}$ a la forme $E_{n-k'+1}(\dots(E_1 E)\dots)$ et que pour tout $i \in [1, n-k'+1]$ la constante E_i est l'une des constantes $E_{j_i,k_1^i,k_2^i,k_3^i,(n-i)}$. □

Le lemme précédent permet directement de conclure que tout terme de Σ_1 de type D_0 est de la forme $E_n(E_{n-1}\dots(E_1 E)\dots)$ où les E_i sont l'une des constantes $E_{B_{j_i},k_1^i,k_2^i,k_3^i,(n-i)}$. Aussi, si il existe un terme $t = E_n(\dots(E_1 E)\dots)$ de Σ_1 tel que $\mathcal{L}(t) =_{\beta\eta} u$ alors, $C = \{B_{j_1}, \dots, B_{j_n}\}$ est solution du problème X3C défini par X et B puisque si $B_{j_i} = \{a_{j_i 1}; a_{j_i 1}; a_{j_i 3}\}$ alors $\mathcal{L}(E_{B_{j_i},k_1^i,k_2^i,k_3^i,(n-i)})$ contient exactement une occurrence des constantes objet $a_{j_i 1}$, $a_{j_i 2}$ et $a_{j_i 3}$.

Nous allons maintenant démontrer la réciproque à savoir que si il existe une solution au problème X3C défini par X et B alors il existe un terme t de Σ_1 tel que $\mathcal{L}(t) =_{\beta\eta} u$.

Lemme 6.3 *Étant donné $l \in [1,n]$ et $\{B_{i_1}; \dots; B_{i_l}\}$ un sous-ensemble de B tel que :*

$$\forall j,k \in [1,l].B_{i_j} \cap B_{i_l} = \emptyset \text{ si et seulement si } j \neq k$$

il existe alors E_1, ..., E_k des constantes de Σ_1 et f une bijection d'un sous-ensemble de $[1,3n]$ dans $[1, n-3l]$ ayant les propriétés suivantes :

1. *$E_l(\dots(E_1 E)\dots)$ est un terme abstrait de type D_{n-l}.*
2. *$\mathcal{L}(E_l(\dots(E_1 E)\dots)) =_{\beta\eta} \lambda x_1 \dots x_{3n-3l}.e t_1 \dots t_{3n}$*
3. *si $a_j \in \bigcup_{k=1}^{l} B_{i_k}$ alors $t_j = a_j$, sinon f est définie en j et $t_j = x_{f(j)}$.*

Démonstration. On procède par induction sur l.

Si $l=0$ alors E est bien de type D_n et $\mathcal{L}(E) = \lambda x_1 \dots x_{3n}.e x_1 \dots x_{3n}$ et il suffit de prendre $f(j) = j$ pour tout $j \in [1,n]$.

Si $l = l'+1$ alors par hypothèse d'induction, il existe des constantes $E_1, \dots, E_l$ de Σ_1 et une bijection f tels que :

1. $E_{l'}(\dots(E_1 E)\dots)$ est un terme abstrait de type $D_{n-l'}$.
2. $\mathcal{L}(E_{l'}(\dots(E_1 E)\dots)) =_{\beta\eta} \lambda x_1 \dots x_{3n-3l'}.e t_1 \dots t_{3n}$

3. si $a_j \in \bigcup_{k=1}^{l'} B_{i_k}$ alors $t_j = a_j$, sinon f est définie en j et $t_j = x_{f(j)}$.

Mais comme, par hypothèse, pour tout $j \in [1, l']$, $B_{i_l} \cap B_{i_j} = \emptyset$, il s'ensuit que $B_{i_l} \cap \bigcup_{k=1}^{l'} B_{i_k} = \emptyset$. Donc si $B_{i_l} = \{a_{i_l 1}; a_{i_l 2}; a_{i_l 3}\}$ il s'ensuit que f est définie en $i_l 1$, $i_l 2$ et $i_l 3$. On prend

$$E_l = E_{B_{i_l}, f(i_l 1), f(i_l 2), f(i_l 3), (n-l)}$$

et on vérifie les trois propriétés :

1. E_l est une constante de type $D_{n-l+1} \multimap D_{n-l}$ or comme $n - l + 1 = n - (l-1) = n - l'$, E_l est une constante de type $D_{n-l'} \multimap D_{n-l}$. Il vient donc que le terme $E_l(E_{l'}(\ldots(E_1 E)\ldots))$ est un terme de type D_{n-l}.
2. Comme $\mathcal{L}(E_l) = \lambda g x_1 \ldots x_{3n-3l}.g \quad \begin{matrix} x_1 \ldots x_{f(i_l 1)-1} a_{i_l 1} \\ x_{f((i_l 1)+1} \ldots x_{f(i_l 2)-1} a_{i_l 2} \\ x_{f(i_l 2)+1} \ldots x_{f(i_l 3-1} a_{i_l 3} \\ x_{f(i_l 3)+1} \ldots x_{3n-3l} \end{matrix}$, $\mathcal{L}(E_l(E_{l'}(\ldots(E_1 E)\ldots)))$ se réduit à $\lambda x_1 \ldots x_{3n-3l} e t'_1 \ldots t'_n$.
3. de plus $t'_j = a_j$ si $a_j \in \bigcup_{k=1}^{l'} B_{i_k}$ et comme on avait $t_{i_l 1} = x_{f(i_l 1)}$, $t_{i_l 2} = x_{f(i_l 2)}$ et $t_{i_l 3} = x_{f(i_l 3)}$ on a $t'_{i_l 1} = a_{i_l 1}$, $t'_{i_l 2} = a_{i_l 2}$ et $t'_{i_l 3} = a_{i_l 3}$. Donc si $a_j \in \bigcup_{k=1}^{l} B_{i_k}$, alors $t'_j = a_j$. Quant à l'existence d'une bijection f' d'un sous-ensemble de $[1, n]$ dans $[1, 3n - 3l]$ telle que si $t'_j \neq a_j$ alors $t'_j = x_{f'(j)}$, elle est induite par le fait que $\lambda x_1 \ldots x_{3n-3l} e t'_1 \ldots t'_n$ est un λ-terme linéaire.

□

La réduction que nous venons d'effectuer repose essentiellement sur la difficulté de la sélection lexicale. En effet, tout réside dans le fait de choisir les constantes $E_{B_{j_i}, k^i_1, k^i_2, k^i_3, i}$. Il reste donc à savoir si la difficulté de l'analyse repose exclusivement sur des choix lexicaux ou si l'agencement des constantes abstraites est également une source de difficultés.

6.2.2 Analyse générale avec présélection lexicale

La réduction précédente repose essentiellement sur la difficulté d'effectuer les bons choix lexicaux ; nous allons montrer que la construction d'une analyse à partir d'une sélection lexicale est également difficile et même NP-complète pour des grammaires catégorielles abstraites de $\mathcal{L}(2, 1)$. Nous nous proposons dans ce qui suit d'étudier la difficulté de l'analyse dans le cas où le choix des items lexicaux a déjà été effectué. Le problème que nous nous posons est le suivant :
étant données les constantes abstraites a_1, ..., a_n et un terme objet u, existe-t-il un terme abstrait non-lexicalisé t tel que $t a_1 \ldots a_n$ est de type S et $\mathcal{L}(t a_1 \ldots a_n) =_{\beta\eta} u$?

Nous allons montrer qu'en fait ce problème est également NP-complet. Ce résultat vient compléter et mettre en relief le précédent. Il montre que la difficulté de l'analyse des grammaires catégorielles abstraites ne réside pas simplement dans le choix des entrées lexicales, mais également dans la difficulté d'agencer entre elles ces entrées pour construire un terme objet donné.

Pour cela nous allons réduire le problème de 3-PARTITION à ce problème d'analyse.

Définition 6.3 *Un problème de 3-PARTITION est une paire $(\{s_1; \ldots; s_{3m}\}, n)$ constituée d'un ensemble d'entiers et d'un entier tels que pour tout $j \in [1, 3m]$, $\frac{n}{4} < s_j < \frac{n}{2}$.*

Définition 6.4 *On dit qu'un problème de 3-PARTITION $\mathcal{P} = (\{s_1; \ldots; s_{3m}\}, n)$ possède une solution si et seulement si il existe une partition $(S_i)_{i \in [1,m]}$ de $\{s_1; \ldots; s_{3m}\}$ telle que pour tout $i \in [1, m]$:*

$$\sum_{s_j \in S_i} s_j = n$$

La partition $(S_i)_{i \in [1,m]}$ est alors appelée solution de $\mathcal{P}$.

Théorème 6.3 *Savoir si un problème de 3-PARTITION possède une solution est NP-complet.[GJ79]*

Lemme 6.4 *Étant donné un problème de 3-PARTITION $\mathcal{P} = (\{s_1; \ldots; s_{3m}\}, n)$, si $(S_i)_{i\in[1,m]}$ est solution de $\mathcal{P}$, alors pour tout $i \in [1,m]$, $|S_i| = 3$.*

Démonstration. Cette propriété vient directement de l'inégalité $\frac{n}{4} < s_j < \frac{n}{2}$. □

Étant donné un problème de 3-PARTITION $\mathcal{P} = (\{s_1; \ldots; s_{3m}\}, n)$, nous allons construire une grammaire catégorielle abstraite $\mathcal{G}_\mathcal{P} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ à partir de laquelle nous fabriquerons un problème d'analyse avec présélection lexicale qui n'aura de solution que si et seulement si $\mathcal{P}$ en a une.

Définition de $\Sigma_1 = (C_1, A_1, \tau_1)$:

1. $A_1 = \{S\}$
2. $C_1 = \{A; E; S_1; \ldots; S_{3m}\}$
3. $\tau_1(A) = S^m \multimap S$, $\tau_1(E) = S$ et pour tout $j \in [1, 3m]$, $\tau_1(S_j) = S \multimap S$.

Définition de $\Sigma_2 = (C_2, A_2, \tau_2)$:

1. $A_2 = \{*\}$
2. $C_2 = \{a, c, e\}$
3. $\tau_2(a) = *^m \multimap *$, $\tau_2(c) = * \multimap *$ et $\tau_2(e) = *$

Définition de $\mathcal{L}$:

1. $\mathcal{L}(S) = *$
2. $\mathcal{L}(E) = e$
3. pour tout $j \in [1; m]$, $\mathcal{L}(S_j) = \lambda x.c^{s_j}(x)$
4. $\mathcal{L}(A) = \lambda x_1 \ldots x_m.ax_1 \ldots x_m$

Nous allons chercher à analyser le terme $ac^n(e)\ldots c^n(e)$ en utilisant comme sélection lexicale les constantes A, E (m fois) et $S_1, \ldots S_{3m}$.

Lemme 6.5 *Étant donnés $j_1, \ldots, j_p$ des entiers de $[1, 3m]$, $\mathcal{L}(S_{j_p}(\ldots(S_{j_1}E)\ldots)) =_{\beta\eta} c^{\sum_{i=1}^{p} s_{j_i}}(e)$.*

Démonstration. Simple induction sur p. □

Lemme 6.6 *Il existe une solution $P_1, \ldots, P_m$ au problème de 3-PARTITION $\mathcal{P} = (\{s_1; \ldots; s_{3m}\}, n)$ si et seulement si il existe un terme abstrait de $\mathcal{G}_\mathcal{P}$, t, qui contient exactement, une occurrence de A, m occurrences de E, une occurrence de chaque constante $S_1, \ldots, S_{3m}$ et tel que $\mathcal{L}(t) =_{\beta\eta} ac^n(e)\ldots c^n(e)$.*

Démonstration. Si $P = \{i_1; \ldots; i_q\}$ est un sous-ensemble de $[1, k]$, on note t_P le terme abstrait

$$S_{i_1}(\ldots(S_{i_q}E)\ldots).$$

On remarque que t_P contient exactement une occurrence de E, de $S_{i_1}, \ldots$ et de S_{i_q}. Puisque $\{P_1; \ldots; P_m\}$ constitue une partition de l'ensemble $[1, 3m]$, il s'ensuit que le terme abstrait $At_{P_1}\ldots t_{P_m}$ contient exactement une occurrence de A, m occurrences de E, une occurrence de $S_1, \ldots$ et une occurrence de S_{3m}. Or, d'après le lemme précédent, pour tout $i \in [1, m]$ $\mathcal{L}(t_{P_i}) =_{\beta\eta} c^{\sum_{j\in P_i} s_j}(e)$. Mais, par hypothèse, $\sum_{j\in P_i} s_j = n$ et donc $\mathcal{L}(t_{P_i}) =_{\beta\eta} c^n(e)$. Il vient donc que $\mathcal{L}(t) =_{\beta\eta} ac^n(e)\ldots c^n(e)$.

Pour démontrer la réciproque, on part de la constatation que t est forcément de la forme $At_{P_1}\ldots t_{P_m}$. Il s'ensuit, comme pour tout $i \in [1, m]$ $\mathcal{L}(t_{P_i}) =_{\beta\eta} c^n(e)$, que pour tout $i \in [1, m]$, $\sum_{j\in P_i} s_j = n$. Or le fait que pour tout $j \in [1, 3m]$ il n'y ait qu'une et une seule occurrence de chaque S_j a pour conséquence que $\{P_1; \ldots; P_m\}$ est une partition de $[1, 3m]$. On peut donc conclure que $\{P_1; \ldots; P_m\}$ est solution du problème de 3-PARTITION défini par $(\{s_1; \ldots; s_{3m}\}, n)$. □

Comme la grammaire catégorielle abstraite que nous avons employée et comme le terme que l'on cherche à analyser ont tous deux une taille polynômiale par rapport à celle du problème de 3-PARTITION considéré, le lemme précédent démontre la NP-complétude de l'analyse des grammaires catégorielles abstraites de $\mathcal{L}(2, 1)$ avec présélection lexicale.

6.2.3 Analyse particulière

Analyse particulière des grammaires de $\mathcal{L}(2,n)$

Afin de montrer que l'analyse particulière des grammaires de $\mathcal{L}(2,n)$ est polynômiale, nous allons présenter un algorithme qui effectue l'analyse des grammaires de $\mathcal{L}(2,n)$ en temps polynômial. Il s'agit d'un algorithme d'analyse ascendante dans le style de celui développé par Cocke, Kasami et Younger (algorithme de type CKY) pour les grammaires hors contexte. Comme il s'agit d'une *analyse particulière*, certains paramètres de la grammaire dans laquelle on analyse peuvent être des exposants de la fonction de complexité. Nous nous contentons ici d'une présentation succincte de cet algorithme afin de déterminer ses propriétés de correction, de complétude et de complexité. Cet algorithme ne permet pas d'analyser les grammaires hors contexte ou les grammaires d'arbres adjoints respectivement en $\mathcal{O}(n^3)$ et en $\mathcal{O}(n^6)$. Quelques arrangements techniques permettraient d'affiner son comportement pour qu'il en soit ainsi. Nous ne les présenterons cependant pas, étant donné que nous développons plus loin un algorithme d'analyse descendante à la *Earley* dédié aux grammaires de $\mathcal{L}(2,n)$ et qui lui possède cette propriété. Nous avons privilégié ce dernier algorithme parce que, contrairement aux algorithmes de type CKY dont la complexité dans les meilleurs cas est la même que celle dans les pires, sa complexité dans les meilleurs cas peut être linéaire tout comme les algorithmes de type Earley dont il est inspiré.

Nous allons présenter cet algorithme comme un système de réécriture sur des ensembles de paires (d,α) où d est une description complète et α un type abstrait.

Afin de simplifier la présentation de cet algorithme, nous allons nous placer dans le cas de grammaires restreintes.

Définition 6.5 *On dit qu'une grammaire catégorielle abstraite $\mathcal{G}=(\Sigma_1,\Sigma_2,\mathcal{L},S)$ est de conclusion atomique si $\mathcal{L}(S)$ est un type objet atomique.*

On peut toujours ramener l'analyse d'une grammaire catégorielle abstraite à l'analyse d'une grammaire catégorielle abstraite de conclusion atomique.

Théorème 6.4 *Si $\mathcal{G}$ est une grammaire catégorielle abstraite alors il existe une grammaire catégorielle abstraite $\mathcal{G}'$ de conclusion atomique telle que pour tout terme u du langage objet de $\mathcal{G}$, il existe un terme u' du langage objet de $\mathcal{G}'$ tel que u appartient au langage de $\mathcal{G}$ si et seulement si u' appartient au langage de $\mathcal{G}'$.*

Démonstration. Si $\mathcal{G}=(\Sigma_1,\Sigma_2,\mathcal{L},S)$ et si $\mathcal{L}(S)=\overline{\beta_{[1,n]}}\multimap\gamma$ avec γ un type atomique. On pose $\mathcal{G}'=(\Sigma_1',\Sigma_2',\mathcal{L},S')$ avec :

1. $\mathcal{C}_{\Sigma_1'}=\mathcal{C}_{\Sigma_1}\cup\{E\}$ où E est une nouvelle constante.
2. $\mathcal{A}_{\Sigma_2'}=\mathcal{A}_{\Sigma_1}\cup\{S'\}$ où S' est un nouveau type atomique.
3. $\tau_{\Sigma_1'}(c)=\begin{cases} S\multimap S' \text{ si } c=E\\ \tau_{\Sigma_1}(c) \text{ sinon}\end{cases}$
4. $\mathcal{C}_{\Sigma_2'}=\mathcal{C}_{\Sigma_2}\cup\{a_1;\ldots;a_n\}$ où les a_i sont des nouvelles constantes.
5. $\mathcal{A}_{\Sigma_2'}=\mathcal{A}_{\Sigma_2}$
6. $\tau_{\Sigma_2'}(c)=\begin{cases}\beta_i \text{ si } c=a_i\\ \tau_{\Sigma_2}(c) \text{ sinon}\end{cases}$
7. $\mathcal{L}'(c)=\begin{cases}\lambda x.x\overline{a_{[1,n]}} \text{ si } c=E\\ \mathcal{L}(c) \text{ sinon}\end{cases}$

Soit u un terme du langage objet de $\mathcal{G}$, alors $u'=u\overline{a_{[1,n]}}$. Si u appartient au langage de $\mathcal{G}$ soit t un terme abstrait tel que $\mathcal{L}(t)=_{\beta\eta}u$; Et est un terme abstrait de $\mathcal{G}'$ de type S' et $\mathcal{L}(Et)=_{\beta\eta}u'$.

Si, réciproquement, u' appartient au langage de $\mathcal{G}'$ alors forcément tout terme t' tel que $\mathcal{L}'(t')=_{\beta\eta}u'$ est de la forme Et, de plus t est de type S et $\mathcal{L}(t)=_{\beta\eta}u$. □

Définition 6.6 *L'ensemble de descriptions D_u^α est un ensemble de descriptions de support α construites à partir de u. On le définit inductivement sur la structure de α par :*

1. $D_u^\alpha = \{\{v : \alpha\} | u = C[v] \wedge v : \alpha\}$, *si* α *est atomique.*
2. $D_u^{\alpha \multimap \beta} = \{d_1 \multimap d_2 | d_1 \in D_u^\alpha \wedge d_2 \in D_u^\beta\} \cup \{\forall x : \beta.d | D_u^\alpha \wedge x \in FV(d)\}$

On note DC_u^α *l'ensemble des descriptions de* D_u^α *qui sont complètes.*

Il faut remarquer que l'ensemble de D_u^α est bien défini. En effet, les seules variables qui peuvent être libres dans $d \in D_u^\alpha$ sont des variables de u et donc on ne peut quantifier indéfiniment une description de D_u^α.

Définition 6.7 *Étant donnée une grammaire* $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$*, on définit l'ensemble de description* $D_\mathcal{G}$ *par :*

$$D_u^\mathcal{G} = \bigcup_{\alpha \in \mathcal{A}_{\Sigma_1}} DC_u^{\mathcal{L}(\alpha)}$$

Définition 6.8 *Étant donnés une grammaire catégorielle abstraite* $\mathcal{G}$ *et un terme objet* u*, on appelle élément d'analyse une paire* (d, α) *telle que* $d \in D_u^\mathcal{G}$ *et* α *est un type abstrait atomique.*

Définition 6.9 *Soient* $\mathcal{G}$ *une grammaire catégorielle abstraite et* u *un terme objet, on définit la relation* $\rightarrow_{CKY}^{\mathcal{G},u}$ *sur les ensembles d'éléments d'analyse. On dit qu'un tel ensemble* H *se réduit en* H' *si :*

1. *pour tout* $i \in [1, n]$, $(d_i, \alpha_i) \in H$
2. *il existe dans* $\mathcal{G}$ *une constante abstraite* a *de type* $\overline{\alpha_{[1,n]}} \multimap \alpha$ *(*α *étant atomique) et* $d \in D_u^\mathcal{G}$ *tels que* $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(a) : \overline{d_{[1,n]}} \multimap d$ *est dérivable*
3. $H' = H \cup \{(d, \alpha)\}$

Lemme 6.7 *Si* $\emptyset \overset{n}{\rightarrow}_{CKY}^{\mathcal{G},u} \mathcal{H}$ *alors pour tout* $(d, \alpha) \in \mathcal{H}$ *il existe un terme abstrait* t *de type* α *tel que* $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(t) : d$ *est dérivable.*

Démonstration. Ce lemme s'établit simplement par induction sur n. □

Ce lemme nous donne la correction de l'algorithme. En effet, si $\mathcal{G}$ est une grammaire de conclusion atomique alors le fait que $\emptyset \overset{n}{\rightarrow}_{CKY}^{\mathcal{G},u} H$ et que $(\{u : \mathcal{L}(S)\}, S) \in H$ nous assure de l'existence d'un terme abstrait t de type S tel que $\cdot;\cdot;\cdot;\vdash_D \mathcal{L}(t) : \{u : \mathcal{L}(S)\}$ c'est-à-dire, d'après le lemme 5.8, tel que $\mathcal{L}(t) =_\beta u$.

Lemme 6.8 *Étant donnés une grammaire catégorielle abstraite* $\mathcal{G}$ *et un terme* u *sous forme* β*-normale* η*-longue, si* t *est un terme abstrait de type atomique* α *tel qu'il existe* $d \in D_u^\mathcal{G}$ *avec* $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(t) : d$ *est dérivable alors* $\emptyset \overset{*}{\rightarrow}_{CKY}^{\mathcal{G},u} H$ *et* $(d, \alpha) \in H$.

Démonstration. On procède par induction sur la structure de t. On peut considérer sans perte de généralité que t est sous forme β-normale et comme t est de type atomique, $t = a\overline{t_{[1,n]}}$ et pour tout $i \in [1, n]$, t_i est un terme de type atomique α_i. Comme $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(a)\overline{\mathcal{L}(t_{[1,n]})} : d$ est dérivable, d'après le lemme 5.14 il existe $d_1, \ldots, d_n$ dans $D_u^\mathcal{G}$ tels que pour tout $i \in [1, n]$ $\cdot;\cdot;\cdot \vdash_D t_i : d_i$ et que $\cdot;\cdot;\overline{x_{[1,n]} : d_{[1,n]}} \vdash_D \mathcal{L}(a)\overline{x_{[1,n]}} : d$ ce qui revient à avoir que $\cdot;\cdot;\cdot \vdash_D \lambda\overline{x_{[1,n]}}.\mathcal{L}(a)\overline{x_{[1,n]}} : \overline{d_{[1,n]}} \multimap d$ est dérivable, et si $\mathcal{L}(a)$ est sous forme η-longue, cela nous donne que $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(a) : \overline{d_{[1,n]}} \multimap d$ est dérivable. Or par hypothèse d'induction, $\emptyset \overset{*}{\rightarrow}_{CKY}^{\mathcal{G},u} H'$ et pour tout $i \in [1, n]$, $(d_i, \alpha_i) \in H'$. Il s'ensuit que $H' \rightarrow_{CKY}^{\mathcal{G},u} H$ et $(d, u) \in H$. □

Ainsi étant donnée une grammaire catégorielle abstraite de conclusion atomique $\mathcal{G}$, si il existe un terme abstrait de type S tel que $\mathcal{L}(t) =_{\beta\eta} u$ alors $\cdot;\cdot;\cdot \vdash_D \mathcal{L}(t) : \{u : \mathcal{L}(S)\}$ est dérivable pourvu que t et u soient sous forme η-longue. Il s'ensuit que $\emptyset \overset{*}{\rightarrow}_{CKY}^{\mathcal{G},u} H$ et que $(\{u : \mathcal{L}(S)\}, S) \in H$.

Cet algorithme est donc correct et complet pour analyser les grammaires catégorielles abstraites de $\mathcal{L}(2, n)$. Il est de plus polynômial ; étant donnés une grammaire catégorielle abstraite de $\mathcal{L}(2, n)$ $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, \mathrm{S})$ et un terme u, si on pose :

$$p = \max_{\alpha \in \mathcal{A}_{\Sigma_1}} (\rho(\mathcal{L}(\alpha)))$$

et

$$q = \max_{c \in \mathcal{C}_{\Sigma_1}} (\rho(\tau_{\Sigma_1}(c)))$$

alors cet algorithme permet de vérifier si u appartient au langage de $\mathcal{G}$ en $\mathcal{O}(|u|^{p+q+1})$ étapes. En effet, le cardinal de $D_u^{\mathcal{G}}$ est en $\mathcal{O}(|u|^p)$ si bien que l'on a $\mathcal{O}(|u|^{p+q+1})$ façons d'appliquer la règle de réécriture sur les ensembles d'éléments d'analyse.

Analyse particulière des grammaires de $\mathcal{L}(3,1)$

Nous allons exhiber dans cette partie une grammaire catégorielle abstraite lexicalisée de $\mathcal{L}(3,1)$ dont le langage est NP-complet. Cela montre que l'analyse particulière d'une grammaire catégorielle abstraite lexicalisée de $\mathcal{L}(n,p)$ est en général NP-complète si $n \geq 3$. Le principe que nous allons mettre en œuvre consiste à associer un λ-terme linéaire à tout problème de 3-PARTITION (*c.f.* définition 6.3) et ce λ-terme appartiendra au langage de la grammaire que nous allons définir si et seulement si ce problème de 3-PARTITION possède une solution.

Commençons par définir cette grammaire par $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ où les signatures d'ordre supérieur Σ_1 et Σ_2 sont définies respectivement de la manière suivante :

1. $A_{\Sigma_1} = \{B_1, B_2, B_3, C, D, E, S\}$
2. $C_{\Sigma_1} = \{e, e', f_1, f_2, f_3, g, h\}$
3. la fonction τ_{Σ_1} est définie par :
   (a) $\tau_{\Sigma_1}(e) = (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D) \multimap S$
   (b) $\tau_{\Sigma_1}(e') = E \multimap S$
   (c) $\tau_{\Sigma_1}(f_1) = \tau_{\Sigma_1}(f_2) = \tau_{\Sigma_1}(f_3) = (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D) \multimap (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D)$
   (d) $\tau_{\Sigma_1}(g) = E \multimap E \multimap E$
   (e) $\tau_{\Sigma_1}(h) = (E \multimap E \multimap E \multimap E \multimap S) \multimap (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D)$

1. $A_{\Sigma_2} = \{*\}$
2. $C_{\Sigma_2} = \{a, b, c, d, o\}$
3. la fonction τ_{Σ_2} est définie par :
   (a) $\tau_{\Sigma_2}(a) = * \multimap * \multimap *$
   (b) $\tau_{\Sigma_2}(b) = \tau_{\Sigma_2}(c) = \tau_{\Sigma_2}(d) = * \multimap *$
   (c) $\tau_{\Sigma_2}(o) = *$

Il nous reste maintenant à définir le lexique :

1. $\mathcal{L}(e) = \lambda f.f\, o\, o\, o\, o$
2. $\mathcal{L}(e') = \lambda x.d\, x.$
3. $\mathcal{L}(f_1) = \lambda f\, x_1\, x_2\, x_3\, y.f\, (b\, x_1)\, x_2\, x_3\, (c\, y)$
4. $\mathcal{L}(f_2) = \lambda f\, x_1\, x_2\, x_3\, y.f\, x_1\, (b\, x_2)\, x_3\, (c\, y)$
5. $\mathcal{L}(f_3) = \lambda f\, x_1\, x_2\, x_3\, y.f\, x_1\, x_2\, (b\, x_3)\, (c\, y)$
6. $\mathcal{L}(g) = \lambda x\, y.a\, x\, y$
7. $\mathcal{L}(h) = \lambda f\, x_1\, x_2\, x_3\, y.f\, (d\, x_1)\, (d\, x_2)\, (d\, x_3)\, (d\, y)$

Étant donné un problème de 3-PARTITION $\mathcal{P}$, on construit un terme $u_{\mathcal{P}}$ tel que $u_{\mathcal{P}}$ appartient au langage de $\mathcal{G}$ si et seulement si $\mathcal{P}$ possède une solution. Avant de donner la définition de $u_{\mathcal{P}}$, nous allons fixer quelques notations :

Notation 6.1

1. *Nous noterons $t_1 \star t_2$ le terme objet $a\, t_1\, t_2$, nous considérerons de plus que l'opérateur $\star$ est associatif à gauche. Nous noterons par conséquent $t_1 \star t_2 \star \ldots \star t_n$ le terme $(\ldots(t_1 \star t_2) \star \ldots \star t_n)$.*

2. *Nous noterons t^p le terme défini comme il suit :*

$$\begin{cases} t^1 &= t \\ t^{n+1} &= t \star t^n \end{cases}$$

3. *Nous noterons $t_1 \bar{\star} t_2$ le terme abstrait $g\, t_1\, t_2$ et nous considérerons cet opérateur comme associatif à gauche.*
4. *Nous noterons $(v_n^{\mathbf{k}})_{n\in\mathbb{N}}$, où $\mathbf{k}$ représente l'une des constantes objet b ou c, la suite de termes définie inductivement par :*

$$\begin{cases} v_0^{\mathbf{k}} &= o \\ v_{n+1}^{\mathbf{k}} &= \mathbf{k}\, v_n^{\mathbf{k}} \end{cases}$$

finalement, la suite de termes $(u_n^{\mathbf{k}})_{n\in\mathbb{N}}$ est définie par :

$$u_n^{\mathbf{k}} = d\, v_n^{\mathbf{k}}$$

Définition 6.10 *Étant donné un problème de 3-PARTITION $\mathcal{P} = (\{s_1, \ldots, s_{3m}\}, n)$, le terme $u_{\mathcal{P}}$ est défini par :*

$$u_{\mathcal{P}} = u_{s_1}^b \star \ldots \star u_{s_{3m}}^b \star (u_n^c)^m$$

Nous allons dans un premier temps montrer que si $\mathcal{P}$ possède un solution, alors il existe un terme abstrait $t_{\mathcal{P}}$ de type S tel que $\mathcal{L}(t_{\mathcal{P}}) =_\beta t_{\mathcal{P}}$.

Pour cela, nous définissons une suite de famille de termes abstraits $(K_{i,j,k})_{i,j,k\in\mathbb{N}}$ de type $(B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D) \multimap (B_1 \multimap B_2 \multimap B_3 \multimap C \multimap D)$ de la manière suivante :

1. $K_{0,0,0} = \{\lambda f\, x_1\, x_2\, x_3\, y.f\, x_1\, x_2\, x_3\, y\}$
2. $K_{i+1,j,k} = \{t \mid \exists v \in K_{i,j,k}.\ t =_{\beta\eta} \lambda f.v\,(f_1\, f)\}$
3. $K_{i,j+1,k} = \{t \mid \exists v \in K_{i,j,k}.\ t =_{\beta\eta} \lambda f.v\,(f_2\, f)\}$
4. $K_{i,j,k+1} = \{t \mid \exists v \in K_{i,j,k}.\ t =_{\beta\eta} \lambda f.v\,(f_3\, f)\}$

Finalement, on définit l'ensemble de termes $G_{i,j,k}$ de type $(E \multimap E \multimap E \multimap E \multimap S) \multimap S$ par :

$$G_{i,j,k} = \{t \mid \exists v \in K_{i,j,k}.\ t =_{\beta\eta} \lambda f.e(v\,(h\, f))\}$$

Lemme 6.9 *Pour tout $i, j, k \in \mathbb{N}$ et pour tout $t \in G_{i,j,k}$ on a :*

$$\mathcal{L}(t) =_{\beta\eta} \lambda f.f\, u_i^b\, u_j^b\, u_k^b\, u_{i+j+k}^c$$

Démonstration. Ce lemme se montre par simple induction sur $i + j + k$. □

Nous allons maintenant construire le terme $t_{\mathcal{P}}$ tel que $\mathcal{L}(t_{\mathcal{P}}) =_{\beta\eta} u_{\mathcal{P}}$. Constatons tout d'abord que si $\mathcal{P}$ possède une solution $(S_k)_{k\in[1,m]}$ alors il existe une permutation de $[1, m]$, τ, telle que :

$$S_k = \{s_{\tau(3k-3))}, s_{\tau(3k-2))}, s_{\tau(3k-1))}\}$$

Après avoir posé :

$$t_0 = e'(x_1 \bar{\star} \ldots \bar{\star} x_{3m} \bar{\star} y_1 \bar{\star} \ldots \bar{\star} y_m)$$

on choisit pour tout $k \in [1, m]$ un terme abstrait v_k qui est élément de

$$G_{s_{\tau(3k-2))}, s_{\tau(3k-1))}, s_{\tau(3k)}}$$

et finalement on pose

$$\begin{cases} t_1 &= v_1(\lambda x_{\tau(1)}\, x_{\tau(2)}\, x_{\tau(3)}\, y_1.t_0) \\ t_{k+1} &= v_{k+1}(\lambda x_{\tau(3k+1)}\, x_{\tau(3k+2)}\, x_{\tau(3k+3)}\, y_k.t_k) \end{cases}$$

et pour tout $k \in [1, m]$, t_k est de type S.

Si maintenant on note :

$$w_k = w_{1,k} \star \ldots \star w_{3m,k} \star (u_m^c)^k \star y_{k+1} \ldots \star y_m$$

où :

$$w_{i,k} = \begin{cases} u_i^b \text{ si il existe } j \leq 3k \text{ tel que } \tau(j) = i \\ x_i \text{ sinon} \end{cases}$$

on a alors le lemme suivant :

Lemme 6.10 *Pour tout entier k compris entre 1 et n, on a :*

$$\mathcal{L}(t_k) =_\beta w_k$$

Démonstration. Ce lemme se prouve par simple induction sur k. □

De ce lemme, on déduit que $\mathcal{L}(t_n) =_{\beta\eta} u_\mathcal{P}$ et on a donc la propriété suivante :

Proposition 6.5 *Étant donné un problème de 3-PARTITION $\mathcal{P}$, si $\mathcal{P}$ possède une solution alors il existe un terme abstrait $t_\mathcal{P}$ tel que $\mathcal{L}(t_\mathcal{P}) =_{\beta\eta} u_\mathcal{P}$.*

Afin d'obtenir la propriété réciproque, nous allons étudier trois ensembles de termes T_E, T_S et T_D définis par :

1. T_E est l'ensemble des termes abstraits de type E et dont les variables libres sont toutes de type E.
2. T_S est l'ensemble des termes abstraits de type S et dont les variables libres sont toutes de type E.
3. T_D est l'ensemble des termes abstraits de type $B_1 \multimap B_2 \multimap B_3 \multimap D$ et dont les variables libres sont toutes de type E.

Ces ensembles ont les propriétés suivantes :

1. si $t \in T_E$, ou bien $t = x$ avec x une variable de type E, ou bien $t = g\, t_1\, t_2$ avec t_1, $t_2 \in T_E$.
2. si $t \in T_S$, ou bien $t = e\, t'$ avec $t' \in T_D$ ou bien $t = e'\, t'$ avec $t' \in T_E$.
3. si $t \in T_D$, ou bien $t = f_i\, t'$ avec $t' \in T_D$ ou bien $t = h\,(\lambda x_1\, x_2\, x_3\, x_4.t')$ avec $t' \in T_S$ et $\{x_1; x_2; x_3; x_4\} \subseteq FV(t)$.

À partir de cette caractérisation, on peut conclure que tout terme clos de type S est de la forme :

$$t_1(\lambda x_1^1\, x_2^1\, x_3^1\, x_4^1.(\ldots t_k(\lambda x_1^k\, x_2^k\, x_3^k\, x_4^k.e'(v))\ldots))$$

où v est un terme de T_E dont l'ensemble des variables libres est :

$$\{x_1^1; x_2^1; x_3^1; x_4^1; \ldots; x_1^k; x_2^k; x_3^k; x_4^k\}$$

et où pour tout $l \in [1,k]$, il existe $i_l, j_l, k_l \in \mathbb{N}$ tels que $t_l \in G_{i_l,j_l,k_l}$. Il s'ensuit que

$$\mathcal{L}(t_l) = \lambda f.f\, u_{i_l}^b\, u_{j_l}^b\, u_{k_l}^b\, u_{i_l+j_l+k_l}^c.$$

Si t est un terme abstrait tel que $\mathcal{L}(t) = u_\mathcal{P}$, alors :

$$t = t_1(\lambda x_{i_{1,1}}\, x_{i_{1,2}}\, x_{i_{1,3}}\, y_{i_1}.t_2(\ldots(t_m.\lambda x_{i_{m,1}}\, x_{i_{m,2}}\, x_{i_{m,3}}\, y_{i_m}.e'(x_1 \bar{\star} \ldots \bar{\star} x_{3m} \bar{\star} y_1 \bar{\star} \ldots \bar{\star} y_m))\ldots))$$

Avec ce qui précède, on a que $i_l = s_{l,1}$, $j_l = s_{l,2}$, $j_l = s_{l,3}$ et que $i_l + j_l + k_l = n = s_{l,1} + s_{l,2} + s_{l,3}$. Il s'ensuit que la partition $(S_l)_{l\in[1,n]}$ où $S_l = \{s_{l,1}; s_{l,2}; s_{l,3}\}$ est solution de $\mathcal{P}$. On a ainsi la propriété :

Proposition 6.6 *Étant donné un problème de 3-PARTITION $\mathcal{P}$, si il existe un terme t tel que $\mathcal{L}(t) =_{\beta\eta} u_\mathcal{P}$ alors $\mathcal{P}$ possède une solution.*

Il s'ensuit que vérifier qu'un terme appartient au langage de $\mathcal{G}$ est NP-difficile.

Proposition 6.7 *Si $\mathcal{G}$ est une grammaire catégorielle lexicalisée de $\mathcal{L}(n,p)$, avec $n \geq 3$, alors déterminer si v est un terme du langage de $\mathcal{G}$ est NP-difficile.*

6.2.4 Analyse particulière avec présélection lexicale

Les deux solutions que nous avons employées pour préciser la complexité des problèmes de l'analyse particulière ont des implications directes pour ce qui est de l'analyse particulière avec présélection lexicale.

Dans le cas des grammaires de $\mathcal{L}(2,n)$, étant donnés un ensemble de sélections lexicales $a_1, \ldots, a_n$, et un terme u à analyser, il suffit d'énumérer l'ensemble des solutions de l'équation $\mathbf{X}\mathcal{L}(a_1)\ldots\mathcal{L}(a_n) \stackrel{?}{=} u$ qui est polynômial dans la taille de la grammaire catégorielle abstraite et de vérifier si l'une d'entre elle peut être l'image d'un terme abstrait par le lexique.

Pour ce qui est des grammaires de $\mathcal{L}(3,1)$, la réduction que nous avons employée donne lieu a nombre polynômial de sélections lexicales possible. En effet, si étant donné un problème de 3-PARTITION $\mathcal{P} = (\{s_1,\ldots,s_{3m}\},n)$, si un terme abstrait t est tel que $\mathcal{L}(t) = u_{\mathcal{P}}$, alors t contient $4m-1$ fois la constante g, une fois la constante e', m fois la constantes e, m fois la constante h, de $\frac{mn}{4}$ à $\frac{mn}{2}$ fois chacune des constantes f_1, f_2, f_3. Ainsi, effectuer une analyse dans cette grammaire en ayant à l'avance choisi les entrées lexicales laisse le problème NP-complet.

6.2.5 Complexité dans le cas semi-lexicalisé

Pour ce qui est de l'analyse particulière de grammaire semi-lexicalisée, on peut montrer que ce problème est NP-complet. Cela provient du fait que l'analyse particulière de $\mathcal{L}(2,n)$ est polynômiale, cela montre que si il existe une analyse d'un terme u, alors il y en a une qui ne nécessite qu'un nombre polynômial d'entrées non-lexicalisées. Il n'y a pas de difficulté à étendre se résultat pour l'analyse particulière non-lexicalisée et donc montrer que ce problème est dans NP. Pour ce qui est du cas général, la question est un peu plus délicate ; il est possible dans le cas semi-lexicalisé que certaines analyses aient une taille exponentielle par rapport au terme à analyser, cependant, sur les exemples que nous avons étudiés, on pouvait toujours trouver une représentation polynômiale de ce terme. Nous n'avons pas été plus avant sur la question de la complexité de l'analyse dans le cas semi-lexicalisé.

6.3 Récapitulatif

Dans ce chapitre, nous avons exploré la difficulté de l'analyse des grammaires catégorielles abstraites. Nous avons montré que, dans le cas général, l'analyse était au moins aussi difficile que la recherche de démonstration dans **MELL**. Nous montrons également qu'établir si une grammaire catégorielle abstraite a un langage vide est équivalent au problème de recherche de preuve dans **MELL**. Nous établissons enfin la décidabilité de l'analyse pour les cas lexicalisés et semi-lexicalisés. Les tableaux suivants récapitulent les résultats de complexité concernant l'analyse que nous avons obtenus au cours de ce chapitre :

	Analyse générale	
	sans présélection lexicale	avec présélection lexicale
$\mathcal{L}(2,1)$ semi-lexicalisé	polynômial	NP-complet
$\mathcal{L}(2,2)$ lexicalisé	NP-complet	NP-complet
$\mathcal{L}(2,2)$ semi-lexicalisé	décidable	NP-complet
$\mathcal{L}(n,m)$ $n \geq 2$ et $m \geq 2$ lexicalisé	NP-complet	NP-complet
$\mathcal{L}(n,m)$ $n \geq 2$ et $m \geq 2$ semi-lexicalisé	décidable	NP-complet

	Analyse particulière	
	sans présélection lexicale	avec présélection lexicale
$\mathcal{L}(2,n)$	polynômial	polynômial
$\mathcal{L}(3,1)$	NP-complet	NP-complet
$\mathcal{L}(n,m)$ $n \geq 3$ et $m \geq 1$	NP-complet	NP-complet

ce tableau est valable que les grammaires soient lexicalisées ou qu'elles soient semi-lexicalisées.

Chapitre 7

Algorithmes pratiques d'analyse

Ce chapitre a pour objectif de présenter les conditions dans lesquelles les grammaires catégorielles abstraites peuvent être utilisées en pratique. Dans un premier temps, nous allons étudier le comportement de l'algorithme général d'analyse et en particulier ses propriétés de terminaison et de complexité pour des grammaires lexicalisées ou semi-lexicalisées. Ensuite, nous allons voir un *algorithme d'analyse incrémental* des grammaires catégorielles abstraites. Cet algorithme a pour vocation d'analyser des grammaires dédiées à la langue naturelle et utilise des heuristiques spécifiques. Finalement, nous utiliserons le calcul des descriptions syntaxiques pour construire un algorithme de type Earley pour les grammaires de $\mathcal{L}(2, n)$.

7.1 Algorithme général

Dans un premier temps, nous allons étudier les propriétés combinatoires de l'algorithme général descendant afin de voir dans quelle mesure il peut constituer un algorithme pratique d'analyse pour les grammaires catégorielles abstraites.

7.1.1 Propriétés de terminaison dans le cas lexicalisé

Notation 7.1

1. *Étant donné un problème d'analyse* $e = \langle \Gamma; t : \alpha \rangle$*, on note* $At(e)$ *le multi-ensemble* $At(t)$*.*
2. *Étant donné un ensemble de problèmes d'analyse* S*, on note* $At(S)$ *le multi-ensemble* $\bigcup_{e \in S} At(e)$*.*
3. *Étant donné un problème d'analyse* $e = \langle \Gamma; t : \alpha \rangle$*, on note* T_e *la quantité* $|t|$*.*
4. *Étant donné un ensemble de problèmes d'analyse, on note* T_S *la quantité* $\sum_{e \in S} T_e$*.*

Pour le cas lexicalisé, l'algorithme général termine. En effet, il est aisé de constater que si $S \rightarrow_{\mathcal{A}} S'$ alors $(|At(S')|, T_{S'})$ est strictement inférieur à $(|At(S)|, T_S)$ pour l'ordre lexicographique. De plus, dans ce cas, il est non-déterministe et polynômial, ce qui correspond à la complexité de l'analyse des grammaires lexicalisées.

7.1.2 Le cas semi-lexicalisé

Dans le cas semi-lexicalisé l'algorithme d'analyse ne termine pas forcément. Prenons par exemple la grammaire catégorielle abstraite suivante :

1. $A := \lambda fgx.f(g(x)) : S \multimap S \multimap S$
2. $S := * \multimap *$

On remarque tout d'abord que $(\lambda fgx.f(g(x)))(\lambda x.x)(\lambda x.x) = \lambda x.x$ si bien que si l'on utilise systématiquement la règle *utilisation d'une constante* on se retrouve toujours à devoir analyser le même ensemble de problèmes d'analyse :

$$\{\langle\cdot;\lambda x.x:S\rangle\} \rightarrow_{\mathcal{A}} \{\langle\cdot;\lambda x.x:S\rangle\} \rightarrow_{\mathcal{A}} \{\langle\cdot;\lambda x.x:S\rangle\}\ldots$$

L'utilisation des ensembles comme structure de données nous prémunit d'un accroissement sans fin du nombre des problèmes d'analyse qui interviennent au cours d'une analyse. En effet, dans l'exemple qui nous occupe, on obtient à chaque pas l'ensemble $\{\langle\cdot,\lambda x.x,S\rangle\}$ comme la réunion de cet ensemble avec lui-même. Il reste cependant que rien ne permet dans cet algorithme de détecter les boucles. Ici la boucle est évidente à repérer, la question est de savoir si on peut toutes les détecter ou non.

La réponse est affirmative. Pour pouvoir obtenir un algorithme qui termine dans le cas semi-lexicalisé, il faut légèrement modifier l'algorithme précédent. Il s'agit d'effectuer la tabulation des résultats, c'est-à-dire garder en mémoire l'historique de toutes les transformations que l'on a effectuées, pour pouvoir le cas échéant détecter les boucles. Désormais, la structure de données que nous allons manipuler sera une paire (G,S) où G est un graphe dans lequel on mémorise les dérivations que l'on a effectuées et S est l'ensemble des problèmes d'analyse que l'on cherche à analyser.

Définition 7.1 *Un graphe de problèmes d'analyse est une paire (S,R) où S est un ensemble fini d'items et R est un sous-ensemble de $S\times S$. Les éléments de S sont appelés sommets du graphe et ceux de R sont ses arêtes. On dit qu'un problème d'analyse e domine un problème d'analyse e' si il existe une famille de problèmes d'analyse $e_{k\in[1,n]}$ telle que pour tout $i\in[1,n-1]$, $(e_i,e_{i+1})\in R$ et $e_1=e$ et $e_n=e'$.*

Définition 7.2 *On dit qu'un graphe de problèmes d'analyse (S,R) est acyclique si il n'existe pas de famille de problèmes d'analyse $(e_k)_{k\in[1,n]}$ telle que pour tout $i\in[1,n-1]$, $(e_i,e_{i+1})\in R$ et $(e_n,e_1)\in R$.*

Notation 7.2

1. *Si $G_1=(S_1,R_1)$ et $G_2=(S_2,R_2)$ sont deux graphes de problèmes d'analyse, alors on note $G_1\cup G_2$ le graphe de problèmes d'analyse $(S_1\cup S_2,R_1\cup R_2)$.*
2. *Étant donné un graphe de problèmes d'analyse $G=(S,R)$, si $e\in S$, on note $D_{e,G}$ l'ensemble des problèmes d'analyse qui domine e dans G.*
3. *Étant donné un problème d'analyse e et un ensemble de problèmes d'analyse S, on note $e\times S$ l'ensemble des paires (e,e') où $e'\in S$.*

Définition 7.3 *Si $e\in S$ et $e\rightarrow_a S_e$ alors :*

$$(G,S)\rightarrow_{\mathcal{AT}}(G\cup(S_e\cup\{e\},e\times S_e),S\backslash\{e\}\cup S_e)$$

Lemme 7.1 *Si $((S,\emptyset),S)\stackrel{*}{\rightarrow}_{\mathcal{AT}}(G,\emptyset)$ alors il existe un graphe d'items acyclique G'' tel que*

$$((S,\emptyset),S)\stackrel{*}{\rightarrow}_{\mathcal{AT}}(G'',\emptyset).$$

Démonstration. En posant $G_0=(S,\emptyset)$, $S_0=S$ et $G_n=G$, on suppose que $(G_0,S_0)\rightarrow_{\mathcal{AT}}(G_1,S_1)\rightarrow_{\mathcal{AT}}\ldots\rightarrow_{\mathcal{AT}}(G_n,\emptyset)$. On démontre ce lemme par induction sur n.

Si G_n est acyclique, alors la conclusion est évidente. Si G_n possède un cycle alors il existe k tel que G_k est acyclique et G_{k+1} est cyclique. Si $(G_k,S_k)\rightarrow_{\mathcal{AT}}(G_{k+1},S_{k+1})$, c'est que $e\in S_k$ et $e\rightarrow_a S_e$, le fait que G_{k+1} soit acyclique vient de ce que $D_{e,G_k}\cap S_e\neq\emptyset$. Soit $e'\in D_{e,G_k}\cap S_e$, soit j, le plus petit entier tel que $e'\in S_j$, on pose $S'_j=S_j\backslash\{e'\}$ et pour $i\in[j+1,k]$, si

$$(G_i,S_i)\rightarrow_{\mathcal{AT}}(G_i\cup(S_{e_i}\cup\{e_i\},e_i\times S_{e_i}),S_i\backslash\{e_i\}\cup S_{e_i})$$

et $e_i\in S'_i$ alors on pose :

$$(G'_{i+1},S'_{i+1})=(G'_i\cup(S_{e_i}\cup\{e_i\},e_i\times S_{e_i}),S'_i\backslash\{e_i\}\cup S_{e_i})$$

sinon on pose :

$$(G'_{i+1},S'_{i+1})=(G'_i,S'_i)$$

Finalement, on pose $(G''_i, S''_i) = (G'_i, S'_i \cup \{e'\})$, on a $(G_0, S_0) \xrightarrow{p}_{\mathcal{AT}} (G''_k, S''_k)$, $p < k$, $S''_k \subseteq S_k$ et G''_k est acyclique. Or en procédant de la même façon que précédemment, on peut montrer que $(G''_k, S''_k) \xrightarrow{m}_{\mathcal{AT}} (G''_n, \emptyset)$ et $m \leq n - k$. Il nous suffit donc de conclure avec l'hypothèse d'induction. □

Avec le lemme précédent, il est évident que $S \xrightarrow{*}_{\mathcal{A}} \emptyset$ si et seulement si $((S, \emptyset), S) \xrightarrow{*}_{\mathcal{AT}} (G, \emptyset)$ et G acyclique. Ainsi, on peut n'appliquer la relation $\rightarrow_{\mathcal{AT}}$ qu'entre des paires (G, S) où G est acyclique et cet algorithme est correct et complet. Nous allons voir qu'en plus il termine pour les grammaires semi-lexicalisées.

Pour cela, nous allons montrer que l'ensemble des problèmes d'analyse qui peuvent être présents dans un graphe au cours d'une analyse est fini.

Définition 7.4 *Étant donné un problème d'analyse $e = \langle \Gamma; t : \alpha \rangle$, la complexité de e, est définie par $\rho(e) = \rho(\Gamma) + \rho(\alpha)$.*

Définition 7.5 *Étant donnés une grammaire catégorielle abstraite $\mathcal{G}$ et un multi-ensemble de constantes objet $\mathscr{C}$, on définit l'ensemble des problèmes d'analyse $\mathcal{I}_{\mathscr{C},n}$ par :*

$$\mathcal{I}_{\mathscr{C},n} = \{e | At(e) \subseteq \mathscr{C} \wedge \rho(e) \leq n\}$$

Lemme 7.2 *L'ensemble des problèmes d'analyse $\mathcal{I}_{\mathscr{C},n}$ défini sur une grammaire catégorielle abstraite $\mathcal{G}$ est fini.*

Démonstration. Cela provient du fait qu'il existe un nombre fini de démonstrations sans coupure d'un séquent dans **IILL**. □

Définition 7.6 *Étant donné un ensemble de problèmes d'analyse S défini sur une grammaire catégorielle abstraite $\mathcal{G}$, on note $\mathcal{I}_S$ l'ensemble de problèmes d'analyse $\mathcal{I}_{At(S),n}$ où $n = \max_{e \in S}(\rho(e))$.*

Notation 7.3 *Étant donné un graphe $G = (S, R)$, et un ensemble de problèmes d'analyse S' tel que $S' \subseteq S$, on note $\mathcal{I}_{S'} \backslash G$ l'ensemble $\mathcal{I}_{S'} \backslash S$.*

Lemme 7.3 *Si $e \rightarrow_a S_e$ et que la règle utilisée est soit la règle d'élimination des abstractions, soit la règle d'élimination d'une variable soit l'utilisation d'une constante non-lexicalisée du second ordre alors $At(S_e) = At(e)$ et pour tout $e' \in S_e$, $\rho(e') \leq \rho(e)$.*

Démonstration. Ce lemme est évident pour les cas d'élimination d'abstractions ou d'élimination d'une variable.

Si la règle utilisée est l'utilisation d'une constante non-lexicalisée du second ordre a, alors c'est que $e = \langle \Gamma; t : \alpha \rangle$ avec α atomique, et $S_e = \{e_1; \ldots; e_n\}$ avec $e_i = \langle \Gamma_i; t_i : \alpha_i \rangle$, or α_i est atomique, $\Gamma = \Gamma_1, \ldots, \Gamma_n$ et donc $\sum_{i=1}^{n} \rho(e_i) = \rho(e)$. De plus il est évident que $\bigcup_{i=1}^{n} At(t_i) = At(t)$. □

Définition 7.7 *Étant donné un graphe de problèmes d'analyse acyclique $G = (S, R)$ et un problème d'analyse $e \in S$, on appelle hauteur de e dans G, ce que l'on note $h_{e,G}$, le nombre d'items tels e' tel que $e \in D_{e',G}$.*

Étant donné un sous-ensemble S' de S on note $h_{S,G}$ le multi-ensemble d'entiers tel que $h_{S,G}(n) = p$ si et seulement si il y a p éléments de S' de hauteur n dans G. On ordonne les multi-ensembles $h_{S,G}$ suivant l'ordre multi-ensemble.

Définition 7.8 *Étant donné un graphe de problèmes d'analyse $G = (S, R)$, on note A_G le cardinal du plus grand ensemble d'arêtes R' tel que $R' \cap R = \emptyset$ et $(S, R \cup R')$ est acyclique.*

Définition 7.9 *Étant donnés un graphe de problèmes d'analyse G et un ensemble de problèmes d'analyse S, on définit la norme $|(G, S)|$ par :*

$$|(G, S)| = (|At(S)|, |\mathcal{I}_S \backslash G|, A_G, h_{S,G})$$

Nous allons voir que la norme $|(G,S)|$ diminue pour l'ordre lexicographique sous l'action de la relation $\rightarrow_{\mathcal{AT}}$.

Lemme 7.4 *Soient (G,S) et (G',S') des couples de graphe de problèmes d'analyse et d'ensemble de problèmes d'analyse tels que G et G' sont acycliques et $(G,S) \rightarrow_{\mathcal{AT}} (G',S')$ (en utilisant une grammaire semi-lexicalisée) alors $|(G',S')|$ est inférieur à $|(G,S)|$ pour l'ordre lexicographique.*

Démonstration. On suppose que e est un problème d'analyse de S, que $e \rightarrow_a S_e$, que $G' = G \cup (S_e, e \times S_e)$ et que $S' = (S \backslash \{e\} \cup S_e)$.

Si $e \rightarrow_a S_e$ avec la règle utilisation d'une constante et que la constante employée est lexicalisée, alors $|(G',S')|$ est bien inférieur à $|(G,S)|$ puisque $|At(S')| < |At(S)|$.

Si il s'agit d'une autre règle, nous avons vu avec le lemme 7.3 que dans ce cas, $\max_{e' \in S_e}(\rho(e')) \leq \rho(e)$ et que $At(S_e) = At(e)$. Ainsi, $\mathcal{I}_{S_e} \subseteq \mathcal{I}_e$ et donc $\mathcal{I}_{S'} \subseteq \mathcal{I}_S$. Si S_e n'est pas inclus dans les sommets de G alors $|\mathcal{I}_S \backslash G'| < |\mathcal{I}_S \backslash G|$ et comme $\mathcal{I}_{S'}$ est inclus dans $\mathcal{I}_S$, il s'ensuit que $|\mathcal{I}_{S'} \backslash G'| < |\mathcal{I}_S \backslash G|$. Si par contre, S_e est inclus dans l'ensemble des sommets de G, alors on a seulement que $|\mathcal{I}_{S'} \backslash G'| \leq |\mathcal{I}_S \backslash G|$, mais si $e \times S_e$ n'est pas inclus dans la relation de G, alors, $A_{G'} < A_G$, car en ajoutant des arêtes à G, on diminue le nombre maximum d'arêtes que l'on peut lui ajouter tout en le laissant acyclique. Si finalement $e \times S_e$ est inclus dans l'ensemble des arêtes de G, alors $G' = G$ et les items de S_e sont moins hauts dans le graphe que e, il s'ensuit donc que pour l'ordre multi-ensemble, $h_{S',G'}$ est strictement inférieur à $h_{S,G}$.

Ainsi,dans tous les cas, $|(G',S')|$ est strictement inférieur à $|(G,S)|$ pour l'ordre lexicographique. □

Nous avons ainsi démontré que la relation $\rightarrow_{\mathcal{AT}}$ permettait de définir un algorithme d'analyse qui termine dans le cas semi-lexicalisé. On pourrait démontrer que pour l'analyse particulière des grammaires de $\mathcal{L}(2,n)$, cet algorithme est polynômial. Cependant, il n'est pas très efficace. Pour s'en convaincre, il suffit d'étudier son comportement pour l'analyse des grammaires hors contexte. Si on suppose que le terme que l'on cherche à analyser est u alors le nombre de problèmes d'analyse utilisés par l'algorithme est en $\mathcal{O}(|u|^2)$, ce qui a pour conséquence que le graphe aura une taille en $\mathcal{O}(|u|^4)$. La complexité en espace de cet algorithme est donc moins bonne que celle des algorithmes usuels pour les grammaires hors contexte. De plus, si la grammaire hors contexte que l'on code possède une règle de la forme $\alpha \rightarrow \alpha_1 \ldots \alpha_n$, où pour tout $i \in [1,n]$, α_i est un non-terminal alors pour rechercher une analyse, on essaiera tous les découpages en n portions de la chaîne à analyser ce qui aura pour conséquence que l'analyse pourra prendre un temps en $\mathcal{O}(|u|^n)$. Aussi en pratique cet algorithme ne pourra pas être vraiment efficace.

7.2 Algorithme d'analyse incrémentale

L'algorithme que nous venons de voir doit pour fonctionner résoudre des équations de filtrage relativement difficiles. Afin de traiter au plus près les applications linguistiques pour lesquelles les grammaires catégorielles abstraites ont été définies, nous proposons un algorithme incrémental d'analyse dédié aux grammaires catégorielles abstraites lexicalisées et dont la signature objet est une signature de chaînes. Cette approche fait l'économie des algorithmes de filtrage et est inspirée d'un algorithme proposé par Glynn Morrill [Mor00] pour le calcul de Lambek. Elle semble bien adaptée à l'analyse de grammaires qui modélisent la langue naturelle car elle permet de modéliser formellement une notion d'acceptabilité. Un algorithme similaire a été développé pour les grammaires d'interaction et a été implémenté dans l'analyseur syntaxique LEOPAR [GB03]. Les résultats obtenus par LEOPAR dans l'analyse de phrases relativement complexes et ambigües sont encourageants.

L'algorithme d'analyse incrémentale repose sur l'algorithme de recherche de démonstration dans **IILL** que nous avons vu au premier chapitre. Il cherche également à analyser les phrases de manière incrémentale, c'est-à-dire en progressant de la gauche vers la droite. Afin de prendre en compte l'ordre des mots, nous allons utiliser une liste de constantes objet pour pouvoir faire des sélections lexicales au fur et à mesure de la lecture de la phrase.

Définition 7.10 *Étant donné un terme $u = a_1 + \ldots + a_n$, on notera L_u la liste $a_1, \ldots, a_n$.*

Définition 7.11 *Un problème d'analyse sur une grammaire catégorielle abstraite $\mathcal{G}$ est un quadruplet $(\Gamma; \Delta; S; L; u)$ où Γ est un contexte d'assignation de types abstraits à des inconnues, Δ est un contexte*

Lexique
Si $At(\mathcal{L}(a)) \sqsubseteq L$, alors

$$(\Gamma; \Delta; I; L; u) \rightarrow_{incr} (\Gamma; \Delta; I, \mathcal{L}(a) : \tau_{\Sigma_1}(a); L \backslash \{At(\mathcal{L}(a))\}; u)$$

Activation
Si $x \notin \overline{\Delta}$ alors

$$(\Gamma; \Delta; I, t : \alpha \multimap \beta^+; L; u) \rightarrow_{incr} (\Gamma; \Delta, x : \alpha; I, x : \alpha^-, tx : \beta^+; L; u)$$

Si $\mathbf{X} \notin \overline{\Gamma}$ alors

$$(\Gamma; \Delta; I, t : \alpha \multimap \beta^-; L; u) \rightarrow_{incr} (\Gamma, \mathbf{X} : \alpha; \Delta; I, \mathbf{X} : \alpha^+, t\mathbf{X} : \beta^-; L; u)$$

Neutralisation simple
Si $u = \mathbf{X}\overline{x_P}$, $\overline{x_P} \subseteq FV(t)$, $|t|_{\mathbf{X}} = 0$ et $\rho(I) + \rho(\alpha) = 0$ alors :

$$(\Gamma, \mathbf{X} : \overline{\alpha_P} \multimap \alpha; \Delta, \overline{x_P : \alpha_P}; I, u : \alpha^+, t : \alpha^-; L; u) \rightarrow_{incr} (\Gamma; \Delta; I[\mathbf{X} := \lambda\overline{x_P}.t]; L; u)$$

Neutralisation de phrase

$$(\Gamma, \mathbf{X}_s : S; \Delta; I, t : S^-, \mathbf{X}_s : S^+; L; u) \rightarrow_{incr} (\Gamma; \Delta; I, t : S^\circ; L; u)$$

FIG. 7.1 – Règles de réduction de l'algorithme incrémental

d'assignation de types abstraits à des variables, S est un ensemble de couple $t : \alpha^\epsilon$ où t est un terme objet et α^ϵ est un type abstrait polarisé, L est une liste de constantes objets et u est le terme objet que l'on cherche à analyser.

Définition 7.12 *Étant donné un ensemble I de couples de la forme $t : \alpha^\epsilon$, la complexité de I est définie par :*

$$\rho(I) = \sum_{t:\alpha^\epsilon \in I} \rho(\alpha)$$

Définition 7.13 *On dit qu'un multi-ensemble de constantes $\mathscr{C}$ est en accord avec une liste de constante $L = e_1, \ldots, e_n$, ce que l'on note $\mathscr{C} \sqsubseteq L$, si $e_1 \in \mathscr{C}$ et si $\mathscr{C} \subseteq L$. Si $\mathscr{C} \sqsubseteq L$, on note $L \backslash \mathscr{C}$ la liste obtenue à partir de L en enlevant les occurrences des éléments de $\mathscr{C}$ d'indice le plus faible dans L.*

Définition 7.14 *On appelle inconnue de départ une inconnue distinguée que nous noterons $\mathbf{X}_s$.*

Définition 7.15 *On dit qu'un problème d'analyse P se réduit à un problème P', ce que l'on note $P \rightarrow_{incr} P'$ si on peut utiliser une des règles de la figure 7.1.*

Lemme 7.5 *Si $(\mathbf{X}_s : S; \cdot; \mathbf{X}_s : S^+; L_u; u) \stackrel{*}{\rightarrow}_{incr} (\Gamma; \Delta; I; L; u)$ alors on a les propriétés suivantes :*

1. *si $x : \alpha \in \Delta$ alors il existe exactement deux termes v et w tels que $v : \beta^{\epsilon_1}$ et $w : \gamma^{\epsilon_2}$ sont dans I, $x \in FV(v)$ et $x \in FV(w)$, de plus $\epsilon_1 \neq \epsilon_2$.*
2. *si $\mathbf{X} : \alpha \in \Gamma$ et $\mathbf{X} \neq \mathbf{X}_s$ alors il existe exactement deux termes v et w tels que $v : \beta^{\epsilon_1}$ et $w : \gamma^{\epsilon_2}$ sont dans I, $|v|_{\mathbf{X}} = 1$ et $|w|_{\mathbf{X}} = 1$.*
3. *pour tout $v : \alpha^\epsilon \in I$, il existe un terme abstrait de type α tel que $\mathcal{L}(t) =_{\beta\eta} v$.*

Démonstration. Ce lemme se démontre par simple induction sur la dérivation. □

Lemme 7.6 *Si $(\mathbf{X}_s : S; \cdot; \mathbf{X}_s : S^+; L_u; u) \stackrel{*}{\rightarrow}_{incr} (\cdot; \cdot; u : S^\circ; \cdot; u)$ alors il existe un terme abstrait t de type S tel que $\mathcal{L}(t) =_{\beta\eta} u$.*

Démonstration. Ce résultat est la conséquence directe du théorème de complétude 1.12 de l'algorithme de recherche de démonstration que nous avons vu au premier chapitre. □

L'algorithme défini par la relation $\rightarrow_{incr}$ est correct et complet, mais jamais il n'utilise la structure syntaxique de u, en fait il se contente d'énumérer les termes abstraits de type S dont l'image par le lexique contient les mêmes constantes objet que u. Il fait cela jusqu'à trouver un terme dont l'image par le lexique soit égale à u. Afin de le rendre un peu plus efficace, on utilise le résultat suivant :

Lemme 7.7 *Si* $(\mathbf{X}_s : S; \cdot; \mathbf{X}_s : S^+; L_u; u) \overset{*}{\rightarrow}_{incr} (\Gamma; \Delta; I; L; u) \overset{*}{\rightarrow}_{incr} (\cdot; \cdot; u : S^\circ; \cdot; u)$ *alors pour tout* $v : \alpha^\epsilon \in I$ *si* $FV(v) = \overline{x_P}$ *alors il existe une solution à l'équation* $\mathbf{X}\lambda\overline{x_P}.v \overset{?}{=} u$.

Démonstration. On procède par induction sur la réduction. □

Ainsi, afin de tenir compte de la structure de u faut-il essayer de maintenir la propriété nécessaire exhibée par le lemme précédent. Cependant, cette propriété est relativement difficile à établir, il faudrait à chaque transformation résoudre une équation de filtrage, or un point fort de cet algorithme est que justement il évite de résoudre des équations de filtrage. On peut alors se contenter d'utiliser la condition nécessaire que nous avons présenté au cinquième chapitre c'est-à-dire assurer à chaque étape que pour tout élément $v : \alpha^\epsilon$ de I, si $\overline{x_P}$ est l'ensemble des variables libres dans v alors $\mathbf{X}\lambda\overline{x_P}.v \succcurlyeq u$.

Une propriété remarquable de cet algorithme est qu'il permet de modéliser la complexité syntaxique des phrases. Il donne ainsi un moyen formel de comparer les complexités relatives de deux analyses. Des phrases difficiles à comprendre auront une complexité élevée. Des phrases ambigües verront certaines analyses (et le sens qui leur ait associé) privilégiées par rapport aux autres du fait de leur complexité plus faible. L'ensemble des paires polarisées qui restent à un instant donné lors de l'exécution de l'algorithme correspond à l'ensemble des dépendances qui ne sont pas encore satisfaites par l'analyse en cours. Plus ce nombre est important, plus l'analyse est complexe.

Définition 7.16 *Étant donné un ensemble I de couples $t : \alpha^\epsilon$, le nombre de dépendances positives (resp. négatives) non satisfaites, noté $|I|_+$ (resp. $|I|_-$), est le nombre de couples $t : \alpha^+$ (resp. $t : \alpha^-$) contenus dans I. Le nombre de dépendances non satisfaites de I est défini par $|I|_{+-} = |I|_+ + |I|_-$.*

Définition 7.17 *Étant donnée une réduction :*

$$(\mathbf{X}_s : S; \cdot; \mathbf{X}_s : S^+; L_u; u) \rightarrow_{incr} (\Gamma_1; \Delta_1; I_1; L_1; u) \rightarrow_{incr} \ldots \rightarrow_{incr} (\Gamma_n; \Delta_n; I_n; L_n; u) \rightarrow_{incr} (\cdot; \cdot; u : S^\circ; \cdot; u)$$

on appelle complexité de la réduction la quantité

$$\max_{i \in [1,n]} (|I_i|_{+-})$$

Nous allons voir sur quelques exemples que cette notion de complexité permet de mettre en évidence que certains sens d'une phrase peuvent être préférés à d'autres ou encore que certaines phrases bien que grammaticales ne sont pas acceptables car trop complexes. Ces exemples sont des adaptations aux grammaires catégorielles abstraites de ceux proposés par Morrill pour le calcul de Lambek [Mor00]. La phrase en anglais *everyone loves someone* peut avoir pour sens le fait que pour toute personne il existe une autre personne que la première aime. Mais cette phrase peut avoir un sens différent, moins immédiat, à savoir qu'il existe une personne que tout le monde aime. Nous allons voir que l'analyse qui correspond au second sens est plus complexe. Pour cela on définit les grammaires catégorielles abstraites suivantes :

1. $E := \lambda f.f(everyone) : (np \multimap S) \multimap S$
   $:= \lambda P.\forall y.P(y)$
2. $Q := \lambda f.f(someone) : (np \multimap S) \multimap S$
   $:= \lambda P.\exists y.P(y)$
3. $L := \lambda xy.x + loves + y : np \multimap np \multimap S$
   $:= \lambda xy.Love\, x\, y$

Les termes abstraits $t_1 = E(\lambda x.Q(\lambda y.L\,x\,y))$ et $t_2 = Q(\lambda y.P(\lambda x.L\,x\,y))$ se réalisent tous deux par le premier lexique en $everyone + loves + someone$, mais avec le second, t_1 se réalise en $\forall x.\exists y.Love\,x\,y$ alors que t_2 se réalise en $\exists y.\forall x.Love\,x\,y$. Ainsi ces deux analyses correspondent aux deux lectures possibles de la phrase $everyone + loves + someone$. Les exécutions de l'algorithme défini par la relation $\rightarrow_{incr}$ qui conduisent aux analyses t_1 et t_2 sont respectivement présentées par les figure 7.2 et 7.3. Afin de rendre plus compacte la représentation et plus lisible l'exécution, nous n'avons pas représenté les contextes, nous n'avons pas non plus fait apparaître les étapes d'activation et nous avons également laissé la trace du terme abstrait que l'analyse était en train de construire. Nous avons encadré les éléments qui se neutralisent au cours d'une étape de réduction. On constate que la réduction qui conduit à analyser la phrase *everyone loves someone* avec le terme t_1 a une complexité qui s'élève à 6 alors que la phrase qui conduit à l'analyser avec t_2 a une complexité qui s'élève à 7.

Une autre source de complexité dans les phrases est l'imbrication de propositions subordonnées, on peut ainsi composer une phrase du genre *l'élève dont le devoir que j'ai lu hier soir était mauvais est votre fils* qui n'est pas compréhensible au premier abord mais qui est pourtant grammaticale. Pour se convaincre que la complexité de cette phrase provient de l'enchâssement des propositions relatives, il suffit d'enlever la relative la plus profonde : *l'élève dont le devoir [...] était mauvais est votre fils.* Or si on effectue l'analyse d'une telle phrase avec l'algorithme définit par la relation $\rightarrow_{incr}$, à chaque fois que l'on entre dans l'analyse d'une relative, on analyse le groupe nominal sujet, mais celui-ci attend d'être neutralisé contre un verbe ce qui est retardé par l'analyse de la relative enchâssée suivante. Ainsi, à chaque enchâssement de relative on est conduit à avoir un groupe nominal supplémentaire qui doit être neutralisé. Si bien que la complexité d'analyse de ce type de phrases est directement lié à la profondeur des enchâssements qu'elles contiennent.

En pratique, on peut borner la complexité des phrases que l'on cherche à analyser. Ainsi, on borne le nombre de couples polarisés que l'on peut avoir. Cela revient à modéliser plutôt qu'une notion étroite de grammaticalité une notion plus large d'acceptabilité, c'est-à-dire que l'algorithme n'acceptera plus les phrases qui sont valides seulement parce qu'elles obéissent aux règles édictées par la grammaire, mais les phrases qui en plus d'obéir à la grammaire sont suffisamment simples pour être comprises. De plus, la définition de cette borne est suffisamment simple pour que l'on puisse la moduler suivant les besoins de l'application. Par ailleurs, en bornant le nombre de couples polarisés qu'une analyse peut manipuler, on borne la complexité de types utilisés dans le contexte si bien que l'ensemble des couples polarisés qui peuvent intervenir devient fini et même borné par un polynôme dont le degré est la somme des complexités maximales que peuvent avoir ces types.

Finalement, borner la complexité des phrases que l'on cherche à analyser permet également de développer des heuristiques particulières. Par exemple, cela permet d'orienter la sélection lexicale. Celle-ci doit permettre de neutraliser suffisamment de couples polarisés pour pouvoir poursuivre l'analyse sinon, on ne pourra pas poursuivre l'analyse sans dépasser la borne. L'utilisation de probabilités pour faire les meilleurs choix d'appariements ou les meilleurs choix lexicaux en premier serait également une voie à explorer pour améliorer les performances de cet algorithme. Une telle approche serait d'ailleurs relativement innovante car les approches probabilistes pour le traitement des langues naturelles sont basées sur des modèles qui ne font dépendre la probabilité de ce qui va suivre que des quelques mots qui précèdent. Ici, ces données seront également enrichies par la connaissance de dépendances à satisfaire, ce qui fait dépendre la suite de l'analyse de ce que l'on attend pour satisfaire ce qui précède.

7.3 Algorithme d'analyse à la Earley

L'algorithme d'analyse général utilise deux types d'opérations pour mener à bien une analyse. Ces deux types d'opérations pourraient échanger des informations pertinentes et le fait qu'elles sont utilisées indépendamment peut provoquer une perte d'efficacité. D'un côté, il utilise un ensemble d'opérations qui effectue des démonstrations dans **IILL**. Pour ces opérations, la construction d'un terme d'analyse n'est autre que la construction d'un terme abstrait possédant les bonnes propriétés de typage, ce qui techniquement revient à la construction d'une démonstration en logique linéaire. D'un autre côté, il utilise une opération qui permet de trouver les solutions d'une équation de filtrage linéaire. L'emploi de la règle *utilisation d'une constante* fait forcément appel à un algorithme de filtrage puisqu'il faut trouver les

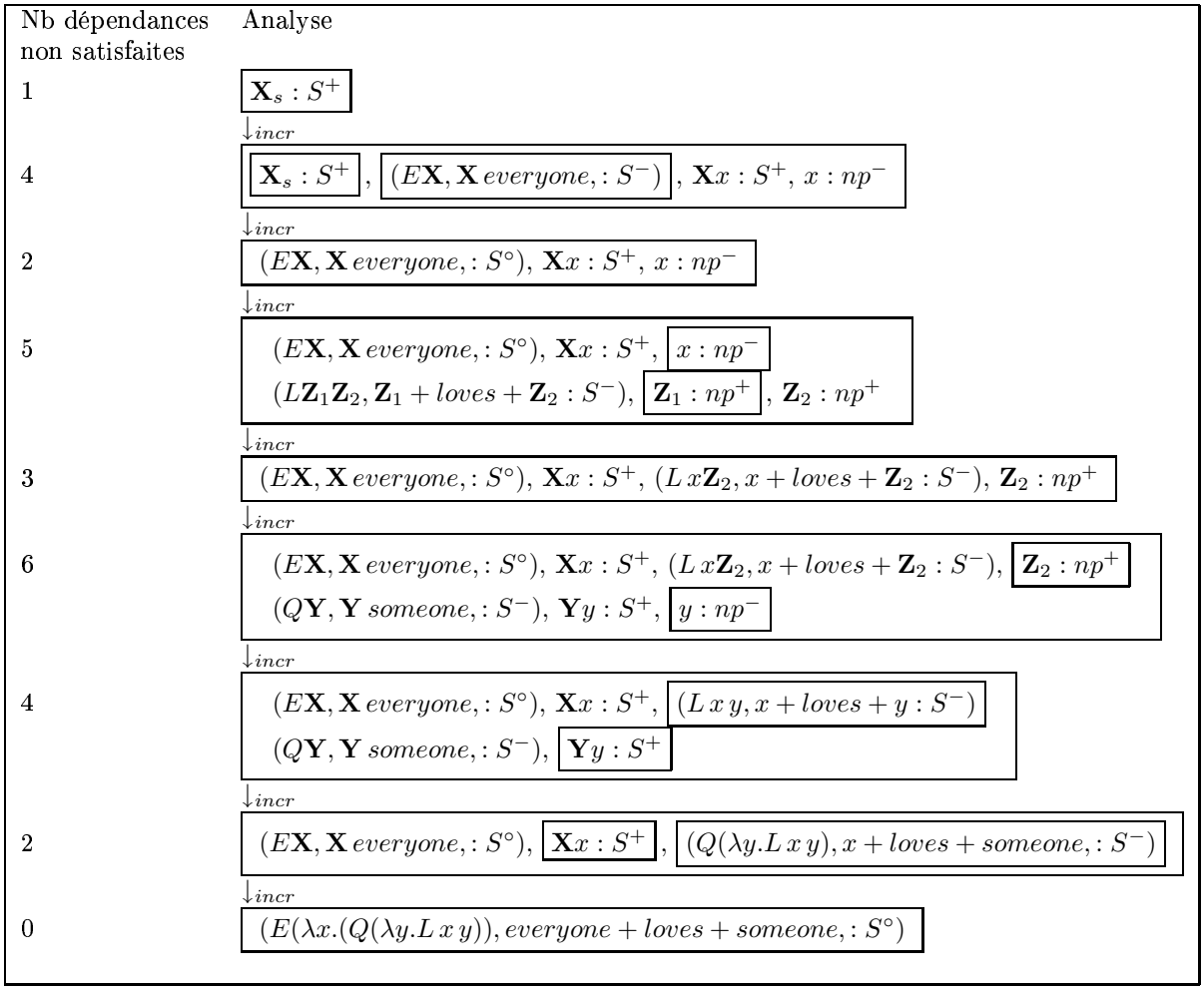


FIG. 7.2 – Analyse de *everyone loves someone* avec la sémantique $\forall x.\exists y.Love\, x\, y$

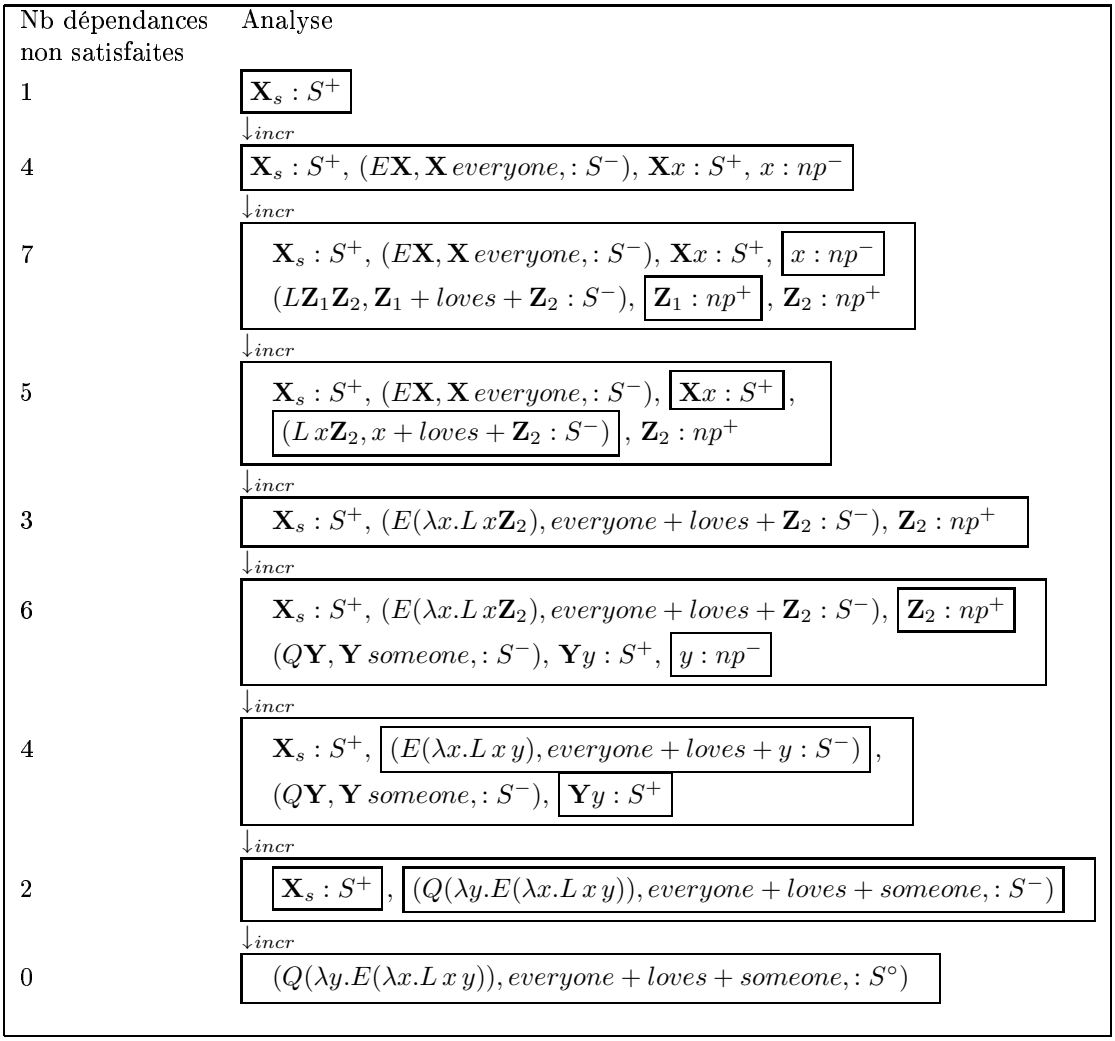


FIG. 7.3 – Analyse de *everyone loves someone* avec la sémantique $\exists y.\forall x.Love\, x\, y$

termes $t_1, \ldots, t_n$ tels que $\mathcal{L}(a)t_1 \ldots t_n =_{\beta\eta} u$. Cette opposition entre ces deux types d'opérations conduit à traiter séparément les informations concernant le langage abstrait (construction d'un terme d'analyse) de celles concernant le langage objet (résolution des équations de filtrage). Or ces deux types d'information sont complémentaires. Étant donnée a une constante abstraite de type $\alpha_1 \multimap \ldots \multimap \alpha_n \multimap \alpha$, dans le cadre d'une analyse, chercher à résoudre l'équation $\mathcal{L}(a)\mathbf{X}_1 \ldots \mathbf{X}_n \stackrel{?}{=} u$ ne consiste pas seulement à trouver des termes v_i tels que $\mathcal{L}(a)v_1 \ldots v_n =_{\beta\eta} u$. On cherche plus précisément des termes v_1, ..., v_n qui sont respectivement l'image par le lexique de termes t_1, ..., t_n ayant pour types abstraits α_1, ..., α_n. L'algorithme général d'analyse descendante parcourt *a priori* toutes les solutions de l'équation $\mathcal{L}(a)\mathbf{X}_1 \ldots \mathbf{X}_n \stackrel{?}{=} u$ en vérifiant ensuite pour chacune d'entre elles si les termes qui la composent sont l'image par le lexique de termes abstraits ayant les types appropriés. Résoudre l'équation $\mathcal{L}(a)\mathbf{X}_1 \ldots \mathbf{X}_n \stackrel{?}{=} u$ en tenant compte du fait que les solutions doivent être l'image par le lexique de termes abstraits de type α_1, ..., α_n semble être une optimisation intéressante qui pourrait permettre d'éviter d'avoir à analyser bon nombre de solutions de cette équation.

Cette section est une première tentative pour rassembler ces opérations de *démonstration* et de *filtrage* pour améliorer les performances de l'algorithme général. Nous nous basons sur la notion de description syntaxique pour créer un système déductif dont l'objectif est d'unifier les deux opérations nécessaires à l'algorithme général. Pour l'instant ce système déductif s'intéresse seulement à l'analyse des grammaires de $\mathcal{L}(2, n)$. Nous espérons que nous pourrons le généraliser pour les autres grammaires. Une conséquence de cette démarche est que ce système déductif permet, pour une grammaire donnée, la construction d'un algorithme d'analyse polynômial utilisant la tabulation. Cet algorithme est une extension de l'algorithme à la Earley pour les grammaires hors contexte et permet d'analyser les grammaires d'arbres adjoints en $\mathcal{O}(n^6)$. Cet algorithme se comporte de la même manière que l'algorithme de Earley dédié aux grammaires hors contexte dans le cas où la grammaire catégorielle abstraite employée correspond à l'encodage d'une grammaire hors contexte. En revanche, il est quelque peu différent de l'algorithme proposé par Joshi et Shabes [AKJ92], pour les grammaires d'arbres adjoints.

Le système déductif que nous allons définir se fonde sur deux formalismes : tout d'abord sur le système des descriptions syntaxiques que nous avons vu en section 5.2 et ensuite sur le système des décompositions lexicales que nous allons détailler dans la prochaine sous-section.

7.3.1 Les décompositions lexicales

Un des écueils que doit éviter un système d'analyse lexicale pour être efficace est de générer explicitement l'ensemble des analyses. Les techniques d'analyse qui se basent sur la tabulation cherchent à représenter des propriétés sur des morceaux de phrases. L'algorithme de Cocke, Kasami et Younger pour les grammaires hors contexte utilise comme propriété que le non-terminal α se récrit avec la grammaire en telle portion de la phrase à analyser. Mais jamais les façons dont ce terminal se récrit en ce bout de phrase ne sont stockées explicitement. Pour ce qui est des grammaires catégorielles abstraites, les morceaux de phrases seront représentés par des descriptions syntaxiques. Ainsi, nous aurons des informations du genre "tel bout de terme correspond à telle description syntaxique". Mais toujours nous éviterons de rendre explicites les raisons pour lesquelles cette correspondance est effective. En particulier, nous ne stockerons pas de terme abstrait de type α dont la transformation par le lexique explique pourquoi tel bout de terme correspond à telle description syntaxique.

Définition 7.18 *Étant données une grammaire catégorielle abstraite $\mathcal{G}$ appartenant à $\mathcal{L}(2, n)$ et une constante abstraite a de $\mathcal{G}$ ayant pour type $\overline{\alpha_{[1,n]}} \multimap \alpha$, si $\Gamma = \overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}$ et t_a est la forme β-normale et η-longue de $\mathcal{L}(a)\overline{\mathbf{X}_{[1,n]}}$ alors le couple (Γ, t_a) est une entrée lexicale de type α. En particulier, on dit que (Γ, t_a) est l'entrée lexicale associée à la constante abstraite a. Si (Γ, t) est une entrée lexicale de type α de la grammaire $\mathcal{G}$, alors on notera $(\Gamma, t) \in \mathcal{G} : \alpha$.*

Le fait d'utiliser des inconnues à la place des λ-abstractions nous permettra de travailler modulo une substitution, ainsi au lieu de stocker une propriété sur la variable de type α pour expliquer comment l'analyse procède, on se contentera de prétendre, sans plus de précision, l'existence d'une substitution justifiant l'analyse. Cela nous permettra d'évacuer des informations inutiles ; informations du même ordre

pour l'analyse de grammaires d'arbres adjoints que celles des adjonctions qui ont été nécessaires pour que tel arbre puisse analyser tel fragment de chaîne.

Tout comme nous l'avons fait pour l'algorithme de résolution d'équation de filtrage au cinquième chapitre, nous allons répartir dans deux ensembles les variables liées dans les entrées lexicales, les variables actives et les variables inactives. Cependant en plus de cette discrimination, nous allons ajouter des informations de typage abstrait aux variables actives.

En effet, les types des variables actives sont reliés à des types abstraits. Prenons le terme $t = \lambda f.f(\lambda xy.axy)$, si f a pour type $(* \multimap * \multimap *) \multimap *$, on peut facilement déduire que x et y ont forcément le type $*$. Supposons maintenant que t soit la réalisation par le lexique d'une constante abstraite dont le type abstrait est $\alpha_1 \multimap \alpha_2$. On ne peut pas donner un type abstrait aux variables x et y, puisque α_1 est un type atomique. Cependant, on sait que quelque soit le terme abstrait u de type α_1, si on applique t à $\mathcal{L}(u)$, alors $\lambda xy.axy$ passera en argument de $\mathcal{L}(u)$. C'est comme si le terme $\lambda xy.axy$ avait pour type abstrait d'être le *premier argument d'un terme de type abstrait* α_1. Ensuite, au cours de la β-réduction, x et y vont être instanciés. Si on considère que $\lambda xy.axy$ a pour type abstrait d'être *le premier argument d'un terme de type abstrait* α_1, on peut considérer que x a pour type abstrait d'être *le premier argument d'un terme qui est le premier argument d'un terme de type abstrait* α_1 et que y est *le second argument d'un terme qui est le premier argument d'un terme de type abstrait* α_1. Il apparaît à travers cet exemple que l'on peut faire descendre au niveau objet une partie de l'information du typage abstrait. Les décompositions lexicales font apparaître cette information et pour cela utilisent la notion de *type abstrait pointé.*

Les seules entrées lexicales pour lesquelles la notion de variable active peut varier sont les entrées lexicales ayant le type de départ de la grammaire dans laquelle on effectue l'analyse. Si une de ces entrées lexicales est à la tête d'un terme d'analyse alors la λ-abstraction avec lesquelles elle débute éventuellement ne donneront lieu à aucun redex. Si par contre une telle entrée lexicale est utilisée au cœur d'une analyse, alors ces abstractions formeront des β-redex au cours de la réduction. Ce phénomène est pris en compte par le système qui permet de caractériser les variables actives.

Définition 7.19 *Étant donné un type α et une liste d'entiers L, si $\alpha = \overline{\gamma_{[1,n]}} \multimap \gamma$ (avec γ atomique) alors $\alpha.L$ désigne une sous-formule de α de la manière suivante :*

$$\alpha.L = \begin{cases} \alpha \text{ si } L = \cdot \\ \gamma_i.L' \text{ si } L = i, L' \text{ et } i \in [1,n] \\ \gamma \text{ si } L = 0 \\ \bot \text{ sinon} \end{cases}$$

On dit que α et L sont compatibles si $\alpha.L \neq \bot$.

Définition 7.20 *Dans une grammaire catégorielle abstrait $\mathcal{G}$, un type abstrait pointé est un couple (α, L) où α est un type abstrait atomique et L est une liste d'entiers et tel que la réalisation par le lexique de α et L sont compatibles.*

En général, nous noterons le couple (α, L) par $\alpha.L$.

Définition 7.21 *Un contexte d'assignation de type abstrait pointé à des variables est un ensemble de paires de la forme $x : \alpha.L$ où x est une variable et $\alpha.L$ est un type abstrait pointé.*

Notation 7.4

1. *Si Γ est un contexte $\overline{x_P : \theta_P}$ où les θ_i sont des types ou des types abstraits pointés et si t est un terme, alors on note $\Gamma_{|t}$ le contexte $\overline{x_Q : \theta_Q}$ où $Q = \{i \in P | x_i \in FV(t)\}$*
2. *Si Γ est un contexte $\overline{\mathbf{X}_P : \alpha_P}$ et si t est un terme alors on note $\Gamma_{|t}$ le contexte $\overline{\mathbf{X}_Q : \theta_Q}$ où $Q = \{i \in P \mid |t|_{\mathbf{X}_i} > 0\}$*

Définition 7.22 *Étant donnés une grammaire catégorielle abstraite $\mathcal{G}$, une décomposition syntaxique et un séquent de la forme $\Gamma; \Gamma'; \Delta \vdash_{\mathcal{G}} t : \theta$ où :*

1. *Γ est un contexte d'assignation de types abstraits à des inconnues*

$$\frac{(\Gamma, t) \in \mathcal{G} : \alpha}{\Gamma; \cdot; \cdot \vdash_{\mathcal{G}} t : \alpha.\cdot} \qquad \frac{(\Gamma, \lambda\overline{x_{[1,n]}}.t) \in \mathcal{G} : S \quad \mathcal{L}(S) = \overline{\gamma_{[1,n]}} \multimap \gamma}{\Gamma; \overline{x_{[1,n]} : \gamma_{[1,n]}}; \cdot \vdash_{\mathcal{G}} t : S.0}$$

$$\frac{\Gamma, \mathbf{X} : \alpha; \Gamma'; \Delta \vdash_{\mathcal{G}} \mathbf{X}\overline{t_{[1,n]}} : \beta}{\Gamma, \mathbf{X} : \alpha; \Gamma'; \Delta \vdash_{\mathcal{G}} \mathbf{X}\overline{t_{[1,n]}} : \alpha.0} \qquad \frac{\Gamma, \mathbf{X} : \alpha; \Gamma'; \Delta \vdash_{\mathcal{G}} \mathbf{X}\overline{t_{[1,n]}} : \beta}{\Gamma_{|t_i}; \Gamma'_{|t_i}; \Delta_{|t_i} \vdash_{\mathcal{G}} t_i : \alpha.i}$$

$$\frac{\Gamma; \Gamma'; \Delta, x : \alpha.L \vdash_{\mathcal{G}} x\overline{t_{[1,n]}} : \beta}{\Gamma; \Gamma'; \Delta, x : \alpha.L \vdash_{\mathcal{G}} x\overline{t_{[1,n]}} : \alpha.(L, 0)} \qquad \frac{\Gamma; \Gamma'; \Delta, x : \alpha.L \vdash_{\mathcal{G}} x\overline{t_{[1,n]}} : \beta}{\Gamma_{|t_i}; \Gamma'_{|t_i}; \Delta_{|t_i} \vdash_{\mathcal{G}} t_i : \alpha.(L, i)}$$

$$\frac{\Gamma; \Gamma'; \Delta \vdash_{\mathcal{G}} \lambda\overline{x_{[1,n]}}.h\overline{t_{[1,p]}} : \overline{\beta_{[1,n]}} \multimap \beta}{\Gamma; \Gamma', \overline{x_{[1,n]} : \beta_{[1,n]}}; \Delta \vdash_{\mathcal{G}} h\overline{t_{[1,p]}} : \beta} \qquad \frac{\Gamma; \Gamma'; \Delta \vdash_{\mathcal{G}} \lambda\overline{x_{[1,n]}}.h\overline{t_{[1,p]}} : \alpha.L}{\Gamma; \Gamma'; \Delta, x_1 : \alpha.(L, 1), \ldots, x_n : \alpha.(L, n) \vdash_{\mathcal{G}} h\overline{t_{[1,p]}} : \alpha.(L, 0)}$$

$$\frac{\Gamma; \Gamma', x : \overline{\beta_{[1,n]}} \multimap \beta; \Delta \vdash_{\mathcal{G}} x\overline{t_{[1,n]}} : \beta}{\Gamma_{|t_i}; \Gamma'_{|t_i)}; \Delta_{|t_i} \vdash_{\mathcal{G}} t_i : \beta_i} \qquad \frac{\Gamma; \Gamma'; \Delta \vdash_{\mathcal{G}} a\overline{t_{[1,n]}} : \theta \quad \tau_{\Sigma_2}(a) = \overline{\beta_{[1,n]}} \multimap \beta}{\Gamma_{|t_i}; \Gamma'_{|t_i}; \Delta_{|t_i} \vdash_{\mathcal{G}} t_i : \beta_i}$$

On suppose que $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$, h désigne un terme atomique, α désigne un type abstrait atomique, β désigne un type objet et θ désigne soit un type objet soit un type abstrait pointé.

FIG. 7.4 – Les décompositions lexicales

2. Γ' *est un contexte d'assignation de types objet à des variables*
3. Δ *est un contexte d'assignation de types abstraits pointés à des variables*
4. t *est un terme objet*
5. θ *est soit un type abstrait pointé soit un type objet.*

et $\Gamma; \Gamma'; \Delta \vdash_{\mathcal{G}} t : \theta$ *est dérivable à partir des règles de la figure 7.4.*

7.3.2 Le système déductif

Le système que nous allons utiliser (*c.f.* figure 7.5)pour formaliser l'analyse des grammaires catégorielles abstraites s'apparente fortement au système $\vdash_{eq}$ (*c.f.* section 5.2.2) pour résoudre les équations de filtrage linéaire du λ-calcul linéaire simplement typé. On peut en fait le voir comme le même système pour lequel on imposerait que le membre gauche de l'équation soit une décomposition lexicale. Ainsi, au lieu de chercher un terme quelconque satisfaisant une description d (ce que l'on faisait avec le système $\vdash_{eq}$) on essaie de le construire à l'aide des entrées lexicales. On peut donc voir ce système déductif comme une adaptation contrainte du système $\vdash_{eq}$.

Définition 7.23 *Un séquent d'analyse dans une grammaire catégorielle abstraite* $\mathcal{G}$ *est un séquent de la forme* $\sigma; \Pi; \Gamma; \Gamma'; \Delta \vdash^{\mathcal{G}}_{\mathcal{A}} t : d$ *où :*

1. σ *est un contexte de renommage de variables*
2. Π *est un contexte d'assignation de types abstraits à des inconnues*
3. Γ *est un contexte d'assignation de types objet à des variables*
4. Γ' *est un contexte d'assignation de types objet à des variables*
5. Δ *est un contexte d'assignation de description à des variables*

On dit qu'un tel séquent est dérivable si on peut le démontrer à l'aide des règles de la figure 7.5

Définition 7.24 *Un terme* t *est appelé décomposition lexicale saturée de type* $\alpha.L$ *si il existe une décomposition lexicale* $\Pi; \Gamma; \Delta \vdash_{\mathcal{G}} v : \alpha.L$ *et une substitution* σ *telle que :*

1. $v.\sigma \equiv t$ *(égalité syntaxique)*

2. *pour tout* $\mathbf{X} : \alpha \in \Pi$, *il existe un terme abstrait* w *de type* α *tel que* $\sigma(\mathbf{X}) \equiv \mathcal{L}(w)$
3. *pour tout* $x : \gamma.L' \in \Delta$ *il existe une décomposition lexicale saturée de type* $\gamma.L'$ w *telle que* $\sigma(x) \equiv w$.
4. *pour tout* $x : \beta \in \Gamma$, $\sigma(x) = x$.

Lemme 7.8 *Le séquent* $\sigma; \overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}; \Gamma; \Gamma'; \overline{x_{[1,p]} : d_{[1,p]}} \vdash^{\mathcal{G}}_{\mathcal{A}} t : d$ *est dérivable si et seulement si et il existe une famille de termes abstraits* $(t_k)_{k\in[1,n]}$, *une famille de décompositions lexicales saturées* $(v_k)_{k\in[1,n]}$, *un renommage* σ' *et deux contextes* Γ_1 *et* Γ_2 *tels que :*

1. $\Gamma = \Gamma_1, \Gamma_2$
2. $\tau = [\mathbf{X}_k := \mathcal{L}(t_k)]_{k=1}^n \circ [x_k := v_k.\sigma']_{k=1}^p$
3. $\Gamma_1; \Gamma_2, \Gamma'; \cdot \vdash_D (t.\tilde{\sigma}).\tau : d$ *est dérivable.*

Démonstration. Ce lemme se démontre simplement par induction sur la dérivation de

$$\sigma; \overline{\mathbf{X}_{[1,n]} : \alpha_{[1,n]}}; \Gamma; \Gamma'; \overline{x_{[1,p]} : d_{[1,p]}} \vdash^{\mathcal{G}}_{\mathcal{A}} t : d$$

. □

Proposition 7.1 *Étant donnée une grammaire catégorielle abstraite* $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ *telle que* $\mathcal{L}(S) = \overline{\beta[1,n]} \multimap \beta$ *avec* β *atomique, si* $\lambda\overline{y_{[1,n]}}.u$ *est un terme objet sous forme* β*-normale* η*-longue de type* $\mathcal{L}(S)$ *alors il existe un terme abstrait qui analyse* u *si et seulement si il existe une entrée lexicale de type* S $(\Pi, \lambda\overline{x_{[1,n]}}.t)$ *telle que* $\overline{x_{[1,n]} := y_{[1,n]}}; \Pi; \cdot; \overline{y_{[1,n]} : \alpha_{[1,n]}}; \cdot \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{u : \beta\}$ *est dérivable.*

Démonstration. Si il existe une entrée lexicale de type S $(\Pi, \lambda\overline{x_{[1,n]}}.t)$ telle que

$$\overline{x_{[1,n]} := y_{[1,n]}}; \Pi; \cdot; \overline{y_{[1,n]} : \alpha_{[1,n]}}; \cdot \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{u : \beta\}$$

est dérivable, alors, d'après le lemme précédent, en supposant que $\Pi = \overline{\mathbf{X}_{[1,p]} : \alpha_{[1,p]}}$, il existe une famille de termes abstraits $(t_k)_{[1,p]}$ telle que t_k est de type α_k et $\cdot; \overline{y_{[1,n]} : \alpha_{[1,n]}}; \cdot \vdash_D t.\sigma.[\mathbf{X}_k := \mathcal{L}(t_k)]_{k=1}^p : \{u : \beta\}$ est dérivable. Or d'après le lemme de correction du calcul des descriptions syntaxiques (lemme 5.8) cela signifie que $t.\sigma.[\mathbf{X}_k := \mathcal{L}(t_k)]_{k=1}^p =_\beta u$. Si bien que si $(\Pi, \lambda\overline{x_{[1,n]}}.t)$ est l'entrée lexicale associée à la constante abstraite c alors $c\overline{t_{[1,p]}}$ constitue une analyse de $\lambda\overline{y_{[1,n]}}.u$.

Si réciproquement, v est une analyse de $\lambda\overline{y_{[1,n]}}.u$ alors $v = c\overline{t_{[1,p]}}$. Et si $(\overline{\mathbf{X}_{[1,p]} : \alpha_{[1,p]}}, \lambda\overline{x_{[1,n]}}.t)$ est l'entrée lexicale associée à c, alors $t[\mathbf{X}_k := \mathcal{L}(t_k)]_{k=1}^p =_\beta u$ et d'après le lemme précédent, cela implique que $\overline{x_{[1,n]} := y_{[1,n]}}; \Gamma; \cdot; \overline{y_{[1,n]} : \alpha_{[1,n]}}; \cdot \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{u : \beta\}$ est dérivable. □

7.3.3 Vers une implémentation

Tout comme pour la conception de l'algorithme de résolution d'équations de filtrage, nous mémorisons les séquents dérivables dans le système $\vdash^{\mathcal{G}}_{\mathcal{A}}$ qui ont un type atomique et qui sont sous forme η-longue. Nous allons présenter cet algorithme de façon informelle, sans donner les démonstrations de sa correction ou de sa complétude. Ces propriétés sont directement induites par le système formel que nous avons vu précédemment. La solution que nous présentons ici n'est certainement pas la seule possible. Nous nous sommes basés sur les idées des algorithmes de Earley pour les grammaires hors contexte et les grammaires d'arbres adjoints. D'autres algorithmes pour ces formalismes ont des propriétés de complexité équivalentes et sont mieux adaptés à certaines situations. Nous espérons que le cadre formel apporté par les descriptions offre suffisamment de souplesse pour permettre de concevoir des algorithmes d'analyse pour les grammaires de $\mathcal{L}(2, n)$ qui se basent sur d'autres idées que celles que nous avons retenues.

L'algorithme que nous allons voir est optimisé ; il fait des distinctions techniques relativement fines qui lui permettent d'analyser les grammaires hors contexte de façon tout à fait identique à l'algorithme proposé par Earley [Ear70]. Il permet également d'analyser les grammaires d'arbres adjoints en $\mathcal{O}(n^6)$, mais son exécution sur les grammaires catégorielles abstraites qui codent des grammaires d'arbres adjoints diffère de l'algorithme proposé par Joshi et Shabes [AKJ92]. Cependant les similitudes que l'on

$$\frac{\cdot;\Pi;\cdot;\cdot;\cdot \vdash^{\mathcal{G}}_{\mathcal{A}} t : d \quad t \in \mathcal{G} : \alpha}{\cdot;\mathbf{X} : \alpha;\Gamma;\cdot;\cdot \vdash^{\mathcal{G}}_{\mathcal{A}} \mathbf{X} : d}\ \text{Inc}_{\mathcal{A}} \qquad \frac{\sigma;\Pi;\Gamma;\Gamma';\Delta \vdash^{\mathcal{G}}_{\mathcal{A}} t : d \quad \Pi;\Gamma';\widetilde{\Delta} \vdash_{\mathcal{G}} t : \alpha.L}{\cdot;\cdot;\Gamma,\Gamma';\cdot;x : d \vdash^{\mathcal{G}}_{\mathcal{A}} x : d}\ \text{Var}_{\mathcal{A}}$$

$$\frac{\cdot \vdash_{\Sigma_2} t : \alpha}{\cdot;\cdot;\cdot;\cdot;\cdot \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{t : \alpha\}}\ \text{Ref}_{\mathcal{A}}$$

$$\frac{}{x := y;\cdot;\cdot;y : \alpha;\cdot \vdash^{\mathcal{G}}_{\mathcal{A}} x : \{y : \alpha\}}\ \text{Id-var}_{\mathcal{A}} \qquad \frac{\sigma, x := y;\Pi;\Gamma;\Gamma',y : \alpha;\Delta \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{u : \beta\}}{\sigma;\Pi;\Gamma;\Gamma';\forall y : \alpha.\Delta \vdash^{\mathcal{G}}_{\mathcal{A}} \lambda x.t : \{\lambda y.u : \alpha \multimap \beta\}}\ \lambda\text{-eq}_{\mathcal{A}}$$

$$\frac{\sigma_1;\Pi_1;\Gamma_1;\Gamma'_1;\Delta_1 \vdash^{\mathcal{G}}_{\mathcal{A}} t_1 : \{u_1 : \alpha \multimap \beta\} \quad \sigma_2;\Pi_2;\Gamma_2;\Gamma'_2;\Delta_2 \vdash^{\mathcal{G}}_{\mathcal{A}} t_2 : \{u_2 : \alpha\}}{\sigma_1,\sigma_2;\Pi_1,\Pi_2;\Gamma_1,\Gamma_2;\Gamma'_1,\Gamma'_2;\vdash^{\mathcal{G}}_{\mathcal{A}} t_1 t_2 : \{u_1 u_2 : \beta\}}\ \text{App-eq}_{\mathcal{A}}$$

$$\frac{\sigma_1;\Pi_1;\Gamma_1;\Gamma'_1;\Delta_1 \vdash^{\mathcal{G}}_{\mathcal{A}} t_1 : d \multimap e \quad \sigma_2;\Pi_2;\Gamma_2;\Gamma'_2;\Delta_2 \vdash^{\mathcal{G}}_{\mathcal{A}} t_2 : d}{\sigma_1,\sigma_2;\Pi_1,\Pi_2;\Gamma_1 \backslash \Gamma'_2;\Gamma'_1,\Gamma'_2;\vdash^{\mathcal{G}}_{\mathcal{A}} t_1 t_2 : e}\ \text{App-desc}_{\mathcal{A}}$$

$$\frac{\sigma;\Pi;\Gamma;\Gamma';\Delta,x : e \vdash^{\mathcal{G}}_{\mathcal{A}} t : d}{\sigma;\Pi;\Gamma;\Gamma';\Delta \vdash^{\mathcal{G}}_{\mathcal{A}} \lambda x.t : e \multimap d}\ \lambda\text{-desc}_{\mathcal{A}}$$

On suppose que $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$.

FIG. 7.5 – Système formel d'analyse dans $\mathcal{L}(2, n)$

constate avec ces deux algorithmes laissent entrevoir la possibilité de construire une machine abstraite qui produirait les langages de λ-termes des grammaires de $\mathcal{L}(2, n)$.

Afin de mieux comprendre les intuitions qui sous-tendent cet algorithme, il faut voir les descriptions syntaxiques comme une généralisation des notions d'indices utilisées dans les algorithmes à la Earley. Pour représenter la sous-chaîne $a_i \dots a_j$ de la chaîne $a_1 \dots a_n$, ces algorithmes utilisent la paire d'indices (i, j). Informellement, avec les descriptions syntaxiques, cette sous-chaîne se représente par :

$$a_{j+1} \dots a_n \multimap a_i \dots a_n$$

c'est-à-dire la chaîne qui si on lui concatène à droite (ce qui est une application dans le λ-calcul linéaire) la chaîne $a_{j+1} \dots a_n$ donne pour résultat la chaîne $a_i \dots a_n$. Cette représentation peut sembler complexe, mais si on implémentait un système basé sur les descriptions, on utiliserait des pointeurs de chaînes.

Pour représenter le contexte de chaîne suivant,

$$a_1 \dots \underline{a_i \dots a_k} \dots \underline{a_l \dots a_j} \dots a_n$$

les analyseurs de grammaires d'arbres adjoints utilisent le quadruplet (i, k, l, j), nous utiliserons la description

$$(a_{l+1} \dots a_n \multimap a_k \dots a_n) \multimap (a_{j+1} \dots a_n \multimap a_i \dots a_n)$$

c'est à dire le contexte qui si on y insère la sous-chaîne $a_k \dots a_l$ produit la sous-chaîne $a_i \dots a_j$.

L'algorithme que nous avons défini gère les problèmes d'α-conversion et pour qu'il fonctionne, nous partons de l'hypothèse que les variables utilisées pour construire les entrées lexicales sont deux à deux distinctes et ces variables ne changent pas lors de la construction des décompositions lexicales. La correction de l'algorithme repose également sur l'hypothèse que les variables utilisées pour construire le terme à analyser sont deux à deux distinctes.

Avant de présenter l'algorithme nous allons voir quelques définitions techniques, notamment le moyen de définir des contextes comportant des valeurs indéfinies, d'extraire et de compléter de tels contextes.

Définition 7.25 *On appelle contexte de renommage mixte un contexte qui contient des paires de la forme $x := y$ ou de la forme $x : \beta$, les parties gauches des paires sont deux à deux disjointes.*

Définition 7.26 *Étant donné un contexte de renommage $\sigma = \overline{x_P := y_P}$ et deux termes v et w on dit que v et w sont compatibles avec σ si $FV(w) \subseteq \overline{y_P}$ et pour tout $i \in P$ $y_i \in FV(w)$ si et seulement si $x_i \in FV(v)$. Si v et w sont compatibles avec σ, on note $\sigma_{|v,w}$ le contexte de renommage $\overline{x_Q := y_Q}$ où $Q = \{i \in P | y_i \in FV(w)\}$.*

Définition 7.27 *Une description généralisée est soit une description, soit $\perp$. Le symbole $\perp$ est la description indéfinie.*

Définition 7.28 *On appelle contexte de désignation un ensemble de paires $x : (d, \alpha.L)$ où x est une variable, d est une description généralisée et $\alpha.L$ est un type abstrait pointé.*

Un contexte de désignation Δest dit complètement instancié si pour tout $x : (d, \alpha.L) \in \Delta$, $d \neq \perp$.

Définition 7.29 *On appelle contexte d'abstraction une liste dont le multi-ensemble sous-jacent est un contexte de désignation, ce contexte de désignation est dit sous-jacent au contexte d'abstraction.*

On dit qu'un contexte d'abstraction est complètement instancié si le contexte de désignation sous-jacent est complètement instancié.

Notation 7.5

1. *Étant donné un ensemble d'indices $P = \{i_1; \ldots; i_n\}$, on note $\overline{x_P : (d_P, \alpha_P.L_P)}$ le contexte de désignation $x_{i_1} : (d_{i_1}, \alpha_{i_1}.L_{i_1}), \ldots, x_{i_n} : (d_{i_n}, \alpha_{i_n}.L_{i_n})$.*
2. *Étant donné une liste d'indices $P = \{i_1; \ldots; i_n\}$, on note $\overline{x_P : (d_P, \alpha_P.L_P)}$ le contexte d'abstraction $x_{i_1} : (d_{i_1}, \alpha_{i_1}.L_{i_1}), \ldots, x_{i_n} : (d_{i_n}, \alpha_{i_n}.L_{i_n})$.*
3. *Étant donné un contexte d'abstraction complètement instancié $\Lambda = \overline{x_P : (d_P, \alpha_P.L_P)}$, et une description d, on note $\Lambda \multimap d$ la description $\overline{d_P} \multimap d$.*
4. *Étant donnés un contexte de désignation (resp. d'abstraction) $\Delta = \overline{x_P : (d_P, \alpha_P.L_P)}$ et un contexte de désignation totalement instancié $\Delta' = \overline{x_Q : (e_Q, \alpha_Q.L_Q)}$, si on pose*

$$\Delta \leftarrow \Delta' = \overline{x_P : (d'_P, \alpha_P.L_P)} \text{ où } d'_i = \begin{cases} e_i \text{ si } i \in Q \\ d_i \text{ sinon} \end{cases}$$

5. *Si $\Delta = \overline{x_P : (d_P, \alpha_P.L_P)}$ est un contexte de désignation(resp. d'abstraction), on note $\overleftarrow{\Delta}$ le contexte $\overline{x_P : d_P}$ et $\overrightarrow{\Delta}$ le contexte $\overline{x_P : \alpha_P.L_P}$*
6. *Étant donné un contexte d'assignation de type abstrait pointé $\Delta = \overline{x_P : \alpha_P.L_P}$, on note $\Delta^\perp$ le contexte de désignation $\overline{x_P : (\perp, \alpha_P.L_P)}$*
7. *Étant donnée une liste de variable $\overline{x_{[1,n]}}$ et un type abstrait pointé $\alpha.L$, on note $\Lambda^\perp_{x_{[1,n]},\alpha.L}$ le contexte d'abstraction $x_1 : (\perp, \alpha.(L, 1)), \ldots, x_n : (\perp, \alpha.(L, n))$*
8. *Étant donné un contexte d'abstraction $\Lambda = \overline{x_P : (d_P, \alpha_P.L_P)}$ et un terme t, on note $\lambda\Lambda.t$ le terme $\lambda\overline{x_P}.t$.*
9. *Étant donné un contexte de renommage mixte $\sigma = \overline{x_P := y_P}, \overline{x_Q : \beta_Q}$ et un contexte de renommage $\tau = \overline{x_R := y_R}$ tel que $R \subseteq Q$, on note $\sigma \leftarrow \tau$ le contexte de renommage mixte*

$$\overline{x_{P,R} := y_{P,R}}, \overline{x_{Q \setminus R} : \beta_{Q \setminus R}}.$$

10. *Étant donné un contexte d'assignation de type objet à des variables $\Gamma = \overline{y_P : \beta_P}$ et une description d, nous notons $\forall\Gamma.d$ la description $\forall\overline{y_P : \beta_P}.d$ (l'ordre des abstractions est arbitraire et n'est pas important).*

Définition 7.30 *Un élément d'analyse intermédiaire est un quintuplet $(\tau; \Gamma; v; w; \beta)$ où :*

1. *τ est un contexte de renommage*
2. *Γ est un contexte d'assignation de types objet à des variables*
3. *v et w sont des termes objets*
4. *β est un type objet*

Les éléments d'analyse intermédiaires représentent des démonstrations à effectuer dans le système formel d'analyse. Soit $(\tau; \Gamma; v; w; \beta)$ un élément d'analyse intermédiaire, il représente le fait que l'on cherche à substituer aux variables et aux inconnues de v des décompositions complètement instanciées pour le rendre égal à w. Le contexte de renommage σ et le contexte Γ sont utilisés pour tenir compte de l'α-conversion et donner des informations de typage sur les variables libres de ces termes. Les éléments d'analyse intermédiaires ne sont pas autonomes et l'information concernant les décompositions lexicales avec lesquelles on cherche à instancier v est contenue dans les éléments d'analyse principaux ou secondaires.

Définition 7.31 *On appelle élément d'analyse principal dans une grammaire $\mathcal{G}$ un 10-uplet*

$$(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)$$

où :

1. $\alpha.L$ *est un type abstrait pointé*
2. σ *est un contexte de renommage mixte*
3. Π *est un contexte d'assignation de types abstraits à des inconnues*
4. Δ *est un contexte de désignation*
5. Λ *est un contexte d'abstraction*
6. t *est un terme objet*
7. I *est un ensemble d'analyses intermédiaires*
8. Γ *est un contexte d'assignation de types objet à des variables*
9. u *est un terme objet*
10. β *est un type objet*

Les éléments d'analyse principaux représentent un état courant d'analyse. Ils sont le pendant dans notre algorithme des règles pointées de l'algorithme de Earley pour les grammaires hors contexte. Si $(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; L; \Gamma; u; \beta)$ est un élément d'analyse principal, alors nous aurons systématiquement que, dans la grammaire $\mathcal{G}$ dans laquelle se déroule l'analyse en cours, $\Pi; \Gamma; \overleftarrow{\Delta} \vdash_{\mathcal{G}} \lambda\Lambda.t.\sigma : \alpha.L$ est une décomposition syntaxique valide. Un tel élément d'analyse représente la tentative en cours de démonstration d'un séquent de la forme $\sigma; \Pi; \Gamma_1; \Gamma_2; \Delta' \vdash^{\mathcal{G}}_{\mathcal{A}} t : \{u : \beta\}$ (avec $\Gamma = \Gamma_1, \Gamma_2$) et on stocke les valeurs des descriptions que nous avons déjà trouvées dans Δ et Λ. Si on distingue Δ et Λ c'est que Λ est la liste des variables qui vont donner lieu à des β-redex au niveau de t et que pour pouvoir en tenir compte on ne peut les traiter simplement comme des variables libres dans t. La liste d'éléments d'analyse intermédiaires I représente ce qu'il reste encore à faire pour pouvoir achever la construction de la démonstration.

Définition 7.32 *On appelle élément d'analyse secondaire un 9-uplet* $[\alpha.L; \sigma\Pi; \Delta; t; I; \Gamma; u; \beta]$ *où :*

1. $\alpha.L$ *est un type abstrait pointé*
2. σ *est un contexte de renommage mixte*
3. Π *est un contexte d'assignation de types abstraits à des inconnues*
4. Δ *est un contexte de désignation*
5. t *est un terme objet de la forme* $\mathbf{X}\overline{v_P}$
6. I *est un ensemble d'analyses intermédiaires*
7. Γ *est un contexte d'assignation de types objet à des variables*
8. u *est un terme objet*
9. β *est un type objet*

Les éléments d'analyse secondaires sont de même nature que les éléments d'analyse principaux, leur présence se justifie essentiellement en vue d'une optimisation des performances de l'algorithme.

Nous allons décrire l'algorithme comme un système de réécriture sur des ensembles d'éléments d'analyse principaux et secondaires.

Définition 7.33 *Étant donné un ensemble $\mathcal{H}$ d'éléments d'analyse principaux et secondaires, on dit que $\mathcal{H}$ produit l'ensemble des éléments d'analyse $\mathcal{H}'$ en analysant u dans $\mathcal{G}$, ce que l'on note $\mathcal{H} \rightarrow_{ea}^{u,\mathcal{G}} \mathcal{H}'$ si on peut appliquer l'une des règles des figures 7.6, 7.7, 7.8 et 7.10.*

Définition 7.34 *Étant donné ensemble d'éléments d'analyse principaux et secondaires $\mathcal{H}$, on dit que $\mathcal{H}$ se réduit à $\mathcal{H}'$ si il existe un ensemble d'éléments d'analyse $\mathcal{H}_a \subseteq \mathcal{H}$ tel que $\mathcal{H}_a \rightarrow_{ea}^{u,\mathcal{G}} \mathcal{H}'_a$ et que $\mathcal{H}' = \mathcal{H} \cup \mathcal{H}'_a$.*

Proposition 7.2 *Il existe un terme abstrait t de type S tel que $\mathcal{L}(t) =_{\beta\eta} u$ dans $\mathcal{G}$ si et seulement si $\emptyset \rightarrow_{\mathcal{EA}}^{*u,\mathcal{G}} \mathcal{H}$ et $\mathcal{H}$ contient un élément d'analyse de la forme :*

$$(S.0; \overline{x_P := y_P}; \Pi; \cdot; \cdot; t'; \cdot; \overline{y_P : \gamma_P}; u'; \gamma)$$

si $\mathcal{L}(S) = \overline{\gamma_P} \multimap \gamma$ et $u = \lambda\overline{x_P}.u'$.

Nous allons maintenant étudier une à une les grandes familles de règles qui définissent l'algorithme et tenter de les expliquer informellement.

La règle d'initiation

Cette règle est similaire à la règle Init de l'algorithme de Earley. Elle crée les premiers éléments d'analyse principaux nécessaires pour commencer une analyse. Comme l'objectif de l'algorithme est d'établir qu'un séquent de la forme

$$\overline{x_P := y_P}; \Pi; \cdot; \overline{y_P : \alpha_P}; \cdot \vdash_{\mathcal{A}}^{\mathcal{G}} t : \{u : \beta\}$$

est dérivable, avec $(\Pi, \lambda\overline{x_P}.t) \in \mathcal{G} : S$ ($\mathcal{L}(S)$ étant égal à $\overline{\beta_P} \multimap \beta$). On crée donc les entrées lexicales $\lambda\overline{x_P}.t \in \mathcal{G} : S$ un élément principal d'analyse qui correspond à cette recherche à savoir :

$$(S.0; \overline{x_P := y_P}; \Pi; \cdot; \cdot; t; (\cdot; \overline{y_P : \beta_P}; t; u; \beta); \overline{y_P : \beta_P}; v; \beta)$$

Initiation

Pour cette règle, on suppose que :

1. $\mathcal{L}(S) = \overline{\beta_P} \multimap \beta$ et β est un type atomique
2. $(\Pi, \lambda\overline{x_P}.t) \in \mathcal{G} : S$
3. $u = \lambda\overline{y_P}.v$

$\mathcal{H} = \emptyset$ et $\mathcal{H}' = \{(S.0; \overline{x_P := y_P}; \Pi; \cdot; \cdot; t; (\cdot; \overline{y_P : \beta_P}; t; u; \beta); \overline{y_P : \beta_P}; v; \beta)\}$

FIG. 7.6 – Règle d'initiation

Les règles de prédiction

Ces règles correspondent aux règles du même nom dans les autres algorithmes de type Earley. Lorsqu'un élément d'analyse intermédiaire a pour tête une inconnue ou une variable active, il nous faut créer un nouvel élément d'analyse qui puisse le cas échéant se substituer à cette inconnue ou à cette variable active et permettre de poursuivre l'analyse.

La règle $\mathbf{Prédiction_{Var}}$ effectue ce travail dans le cas où cette tête est une variable active. Le fait que cette variable soit bien une variable active est vérifié par la troisième hypothèse de cette règle. La cinquième hypothèse propose de choisir une décomposition lexicale ayant le même type que la variable considérée. À partir de cette décomposition lexicale, on produit un élément d'analyse principal dont l'objectif est d'unifier la décomposition lexicale choisie avec le terme objet de l'élément d'analyse intermédiaire considéré.

Cependant, la règle $\mathbf{Prédiction_{Var}}$ requiert, avec la seconde hypothèse, que le type objet que le lexique associe à cette variable ne soit pas atomique. En effet, si son type est atomique, on sait comment instancier cette variable pour qu'elle s'unifie avec le terme objet considéré. Aussi la règle $\mathbf{Prédiction_{Var}}\,at$ procède à cette inférence et met à jour les contextes de décomposition et d'abstraction en accord avec cette inférence. En définissant Δ' dans l'hypothèse 5 de cette règle, on lie dans la description avec un quantificateur universel les variables qui sont libres au niveau de l'élément d'analyse intermédiaire mais liées au niveau de l'élément d'analyse principal. Il faudra cependant *a posteriori* vérifier qu'il existe bien une décomposition lexicale qui puisse s'unifier avec ce terme. Cela se fera au niveau des règles d'application.

Pour les inconnues, la prédiction fonctionne sur le même principe que pour les variables. Nous faisons cependant une nouvelle distinction entre les inconnues suivant que leur ordre est strictement supérieur à 2 ou non. Il s'agit là d'une optimisation qui permet de gagner un degré d'exposant dans le polynôme qui borne la complexité d'exécution de l'algorithme. Lorsque l'ordre de l'inconnue est strictement plus grand que 2, on produit deux éléments d'analyse au lieu d'un seul. Comme on ne stocke à aucun moment d'informations relatives aux descriptions des termes qui doivent remplacer les inconnues, le fait de créer un élément d'analyse secondaire puis de le transformer avec une règle d'application permet de ne pas faire dépendre cette application du terme objet objectif de l'élément d'analyse principal. Ensuite l'utilisation de la règle de complétion a pour effet que les détails de cette opération d'application ne sont pas visibles depuis l'élément d'analyse principal. C'est cela qui fait gagner un indice d'exposant dans la complexité de l'algorithme pour les grammaires dont les types abstraits se linéarisent en des types d'ordre strictement supérieur à 2. Cette distinction n'a pas d'intérêt pour les variables parce que celles-ci rapporteraient tous les détails de l'opération d'application lors de l'utilisation d'une règle de complétion et parce que la complexité de leur type est toujours inférieur à celle des inconnues.

Prédiction

Prédiction$_{\mathbf{Var}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; x\overline{v_P}; w; \gamma), I'$
3. $x : (\bot, \alpha'.L') \in \Delta \cup \Lambda$
4. $\rho(\mathcal{L}(\alpha').L') > 0$
5. $\Pi''; \Gamma''; \Delta'' \vdash_{\mathcal{G}} \lambda\overline{x_{[1,n]}}.t' : \alpha'.L'$

$\mathcal{H}' = \{(\alpha'.L'; \Gamma''; \Pi''; \Delta''^{\bot}; \Lambda^{\bot}_{x_{[1,n]}, \alpha'.L'}; t'; (\cdot; \Gamma'; t'; w; \gamma); \Gamma'; w; \gamma)\}$

Prédiction$_{\mathbf{Var}\,at}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\cdot; \Gamma'; x; w; \gamma), I'$
3. $x : (\bot, \alpha'.L') \in \Delta \cup \Lambda$
4. $\rho(\mathcal{L}(\alpha').L') = 0$
5. $\Delta' = x : (\forall(\Gamma' \backslash \Gamma).\{w : \gamma\}, \alpha'.L')$

$\mathcal{H}' = \{(\alpha.L; \sigma; \Pi; \Delta \leftarrow \Delta'; \Lambda \leftarrow \Delta'; t; I'; \Gamma; u; \beta)\}$

Prédiction$^{>2}_{\mathbf{Inc}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; v; w; \gamma), I'$ avec $v = \mathbf{X}\overline{v_{[1,n]}}$
3. $\mathbf{X} : \alpha' \in \Pi$
4. $ord(\mathcal{L}(\alpha')) > 2$
5. $(\Pi'', \lambda\overline{x_{[1,n]}}.t') \in \mathcal{G} : \alpha'$

$$\mathcal{H}' = \left\{ \begin{array}{l} (\alpha'.\cdot; \cdot; \Pi''; \cdot; \Lambda^{\bot}_{x_{[1,n]}, \alpha'.\cdot}; t'; (\cdot; \Gamma'; t'; w; \gamma); \Gamma'; w; \gamma) \\ [\alpha'.0; \sigma_{|v}; \Pi_{|v}; \Gamma_{|v}, \Lambda_{|v}; v; (\tau; \Gamma'; v; w; \gamma); \Gamma'; w; \gamma] \end{array} \right\}$$

Prédiction$^{\leq 2}_{\mathbf{Inc}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; v; w; \gamma), I'$ avec $v = \mathbf{X}\overline{v_{[1,n]}}$
3. $\mathbf{X} : \alpha' \in \Pi$
4. $ord(\mathcal{L}(\alpha')) \leq 2$
5. $(\Pi'', \lambda\overline{x_{[1,n]}}.t') \in \mathcal{G} : \alpha'$

$\mathcal{H}' = \{(\alpha'.\cdot; \cdot; \Pi''; \cdot; \Lambda^{\bot}_{x_{[1,n]}, \alpha'.\cdot}; t'; (\cdot; \Gamma'; t'; w; \gamma); \Gamma'; w; \gamma)\}$

FIG. 7.7 – Règles de prédiction

Les règles d'application

Ces règles recouvrent une partie des règles de complétion utilisées par les algorithmes à la Earley. Elles permettent d'utiliser les éléments d'analyse principaux dont on a achevé l'analyse, *i.e.* dont l'ensemble des éléments d'analyse intermédiaires est vide, pour pouvoir poursuivre l'analyse des autres éléments d'analyse. Ces règles correspondent à l'utilisation successive de plusieurs règles d'application dans le système formel $\vdash_{\mathcal{A}}^{\mathcal{G}}$.

Le cas des variables actives est le cas le plus général. Le traitement des inconnues se fait de façon similaire, mais dans le cas où leur ordre est strictement plus grand que 2 la règle d'application doit s'effectuer au niveau d'un élément d'analyse secondaire, et dans le cas contraire, elle doit s'effectuer au niveau d'un élément d'analyse principal. Dans ce dernier cas, la règle est simplifiée du fait que tous les arguments de l'inconnue sont du premier ordre. Nous ne présenterons donc que la règle $\mathbf{Application_{Var}}$ car les autres règles ne sont que l'adaptation aux inconnues de cette règle.

La règle $\mathbf{Application_{Var}}$ nécessite l'ensemble des éléments d'analyse $\{e; e_0; e_{i_1}; \ldots; e_{i_p}\}$ pour être utilisée. L'élément e est celui sur lequel on effectue l'application, il contient un élément d'analyse intermédiaire dont la tête est une variable active x. L'élément d'analyse e_0 est l'élément avec lequel on veut instancier x. Les éléments d'analyse $e_{i_1}, \ldots, e_{i_p}$ correspondent à des analyses menées avec les arguments de x qui ne sont pas de type atomique. Tout cela est établi par les hypothèses 1, 2, 3 et 4 pour justifier l'utilisation de la règle. L'hypothèse 5 impose que l'élément d'analyse e_0 permet d'analyser le terme avec lequel on cherche à unifier $x\overline{v_{[1,n]}}$. Les hypothèses 6 et 7 établissent que les descriptions des éléments e_{i_1}, $\ldots$, e_{i_p} sont cohérentes avec la description de e_0, cela permet de faire respecter la discipline de typage imposée par les descriptions. L'hypothèse 8 permet de vérifier que le renommage des variables des e_{i_1}, $\ldots$, e_{i_p} est cohérent par rapport au renommage τ de l'élément d'analyse intermédiaire. Enfin, on crée un nouvel élément d'analyse en mettant à jour, la description de x et les descriptions associées aux variables actives contenues dans les v_{i_k} et dans lequel on remplace l'élément d'analyse intermédiaire qui contenait $x\overline{v_{[1,n]}}$ par l'ensemble des éléments d'analyse intermédiaires qui permettent de vérifier si les arguments de type atomique de x sont en cohérence avec la description de e_0. C'est à ce niveau que l'on vérifie la pertinence des inférences que l'on fait avec la règle $\mathbf{Prédiction_{Var}}_{at}$.

Application

Application$_{\mathbf{Var}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{e; e_0; e_{i_1}; \ldots; e_{i_p}\}$
2. $e = (\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)$
3. $I = (\tau; \Gamma'; x\overline{v_{[1,n]}}; w; \gamma), I'$ et $x : (\bot, \alpha'.L') \in \Delta \cup \Lambda$ et $\mathcal{L}(\alpha').L = \overline{\gamma_{[1,n]}} \multimap \gamma$.
4. pour tout $k \in [1,p]$ on a $i_k \in [1,n]$ et pour tout $j \in [1,n]$, il existe $k \in [1,p]$ tel que $i_k = j$ si et seulement si $\rho(\gamma_j) > 0$. On note $Q = [1,n]\backslash\overline{i_{[1,p]}}$ (on a $j \in Q$ si et seulement si $\rho(\gamma_j) = 0$).
5. $e_0 = (\alpha'.L'; \sigma''; \Pi''; \Delta''; \Lambda''; t''; \cdot; \Gamma'; w; \gamma)$
6. pour $k \in [1,p]$, $e_{i_k} = (\alpha'.(L', i_k); \sigma_{i_k}; \Pi_{i_k}; \Delta_{i_k}; \Lambda_{i_k}; v'_{i_k}; \cdot; \Gamma_{i_k}; w_{i_k}; \gamma_{i_k})$ et $\lambda\Lambda_{i_k}.v'_{i_k} = v_{i_k}$
7. $\Lambda'' \multimap \{w : \gamma\} = \overline{d_{[1,n]}} \multimap \{w : \gamma\}$ et pour $k \in [1,p]$ $d_{i_k} = \forall(\Gamma_{i_k}\backslash\Gamma').\Lambda_{i_k} \multimap \{w_{i_k} : \gamma_{i_k}\}$ et pour tout $j \in Q$, $d_j = \forall\Gamma_j.\{w_j : \gamma_j\}$
8. si $x := y \in \sigma_{i_k}$ (pour $k \in [1,p]$) et $x := z \in \tau$ alors $y = z$.
9. on note $\Delta' = \forall(\Gamma_{i_1}\backslash\Gamma).\Delta_{i_1}, \ldots, \forall(\Gamma_{i_p}\backslash\Gamma).\Delta_{i_p}, x : (\forall(\Gamma'\backslash\Gamma).\Lambda'' \multimap \{w : \gamma\}, \alpha'.L')$; $\sigma' = \sigma_{i_1}, \ldots, \sigma_{i_p}$
10. on note $I'' = \overline{(\tau_{|v_Q, w_Q}; \Gamma_Q, \Gamma_{|w_Q}; v_Q; w_Q; \gamma_Q)}$

$\mathcal{H}' = \{(\alpha.L; \sigma \leftarrow \sigma'; \Pi; \Delta \leftarrow \Delta'; \Lambda \leftarrow \Delta'; t; I', I''; \Gamma; u; \beta)\}$

Application$^{\geq 2}_{\mathbf{Inc}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{e; e_0; e_{i_1}; \ldots; e_{i_p}\}$
2. $e = [\alpha.0; \sigma; \Pi; \Delta; t; (\cdot; \Gamma; t; u; \beta); \Gamma; u; \beta]$
3. $t = \mathbf{X}\overline{v_{[1,n]}}$ et $\mathbf{X} : \alpha \in \Pi$ et $\mathcal{L}(\alpha) = \overline{\gamma_{[1,n]}} \multimap \gamma$.
4. pour tout $k \in [1,p]$ on a $i_k \in [1,n]$ et pour tout $j \in [1,n]$, il existe $k \in [1,p]$ tel que $i_k = j$ si et seulement si $\rho(\gamma_j) > 0$. On note $Q = [1,n]\backslash\overline{i_{[1,p]}}$ (on a $j \in Q$ si et seulement si $\rho(\gamma_j) = 0$).
5. $e_0 = (\alpha.\cdot; \cdot; \Pi''; \cdot; \Lambda''; t''; \cdot; \Gamma'; u; \beta)$
6. pour $k \in [1,p]$, $e_{i_k} = (\alpha'.i_k; \sigma_{i_k}; \Pi_{i_k}; \Delta_{i_k}; \Lambda_{i_k}; v'_{i_k}; \cdot; \Gamma_{i_k}; w_{i_k}; \gamma_{i_k})$ et $\lambda\Lambda_{i_k}.v'_{i_k} = v_{i_k}$
7. $\Lambda'' \multimap \{w : \gamma\} = \overline{d_{[1,n]}} \multimap \{w : \gamma\}$ et pour $k \in [1,p]$, $d_{i_k} = \forall(\Gamma_{i_k}\backslash\Gamma').\Lambda_{i_k} \multimap \{w_{i_k} : \gamma_{i_k}\}$ et pour tout $j \in Q$, $d_j = \forall\Gamma_j.\{w_j : \gamma_j\}$
8. si $x := y \in \sigma_{i_k}$ et $x := z \in \sigma$ alors $y = z$.
9. on note $\Delta' = \forall(\Gamma_{i_1}\backslash\Gamma).\Delta_{i_1}, \ldots, \forall(\Gamma_{i_p}\backslash\Gamma).\Delta_{i_p}$ et $\sigma' = \sigma_{i_1}, \ldots, \sigma_{i_p}$
10. on note $I'' = \overline{(\tau_{|v_Q, w_Q}; \Gamma_Q, \Gamma_{|w_Q}; v_Q; w_Q; \gamma_Q)}$

$\mathcal{H}' = \{[\alpha.0; \sigma \leftarrow \sigma'; \Pi; \Delta \leftarrow \Delta'; t; I''; \Gamma; u; \beta]\}$

Application$^{\leq 2}_{\mathbf{Inc}}$
Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{e; e_0\}$
2. $e = (\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)$
3. $I = (\tau; \Gamma'; \mathbf{X}\overline{v_{[1,n]}}; w; \gamma), I'$ et $\mathbf{X} : \alpha' \in \Pi$ et $\mathcal{L}(\alpha') = \overline{\gamma_{[1,n]}} \multimap \gamma$.
4. $ord(\mathcal{L}(\alpha')) \leq 2$
5. $e_0 = (\alpha'.\cdot; \cdot; \Pi''; \cdot; \Lambda''; t''; \cdot; \Gamma'; w; \gamma)$
6. $\Lambda \multimap \{w : \gamma\} = \overline{d_{[1,n]}} \multimap \{w : \gamma\}$, et pour tout $j \in [1,n]$, $d_j = \forall\Gamma_j.\{w_j : \gamma_j\}$
7. on note $I'' = \overline{(\tau_{|v_{[1,n]}, w_{[1,n]}}; \Gamma_{[1,n]}, \Gamma_{|w_{[1,n]}}; v_{[1,n]}; w_{[1,n]}; \gamma_{[1,n]})}$

$\mathcal{H}' = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I', I''; \Gamma; u; \beta)\}$

FIG. 7.8 – Règles d'application

Les règles de simplification

Les règles de simplification sont une généralisation de la règle Scan des algorithmes de Earley. Lorsqu'un élément d'analyse intermédiaire présente deux termes qui ont des têtes de même nature, pour vérifier finalement qu'ils peuvent devenir égaux en cas de substitution, il suffit de vérifier qu'il existe une substitution qui unifie les sous-termes immédiats. C'est ce que font les règles $\mathbf{Simplification}_{\mathbf{const}}$, $\mathbf{Simplification}_{\lambda-\mathbf{var}}$ et $\mathbf{Simplification}_{FV\mathbf{var}}$. La première traite le cas où les deux termes considérés ont pour tête une même constante. Les deux suivantes traitent le cas où les deux termes commencent par une variable. Ici, nous utilisons deux règles parce que ces variables peuvent avoir des origines différentes. Il se peut que ces deux variables aient été introduites par l'élimination d'une abstraction à quelque étape antérieure, il faut alors vérifier dans le contexte de renommage qu'elles peuvent être égalisées. Mais il se peut également que ces deux variables soient des variables libres de l'élément d'analyse principal. Il faut alors noter que l'analyse les fait se correspondre en mettant à jour le contexte de renommage mixte de l'élément d'analyse principal.

Si par contre les deux termes sont des abstractions, pour tenir compte de l'α-conversion, on note dans le contexte de renommage local à l'élément d'analyse intermédiaire que les deux variables considérées doivent être égalisées pour la suite de l'analyse. Pour cela, on utilise la règle $\mathbf{Simplification}_{\lambda-\mathbf{abst}}$.

Pour améliorer l'efficacité de l'algorithme, on aurait pu utiliser les règles de simplification également sur les éléments d'analyse secondaire, mais nous avons jugé que cela surchargerait un nombre de règles déjà suffisamment important.

Simplification

$\mathbf{Simplification}_{\mathbf{const}}$
Hypothèses d'utilisation de la règle :
1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; a\overline{v_P}; a\overline{w_P}; \gamma), I'$
3. $\tau_{\Sigma_1}(a) = \overline{\gamma_P} \multimap \gamma$
4. $I'' = \overline{(\tau_{|v_P,w_P}; \Gamma_{|w_P}; v_P; w_P; \gamma_P)}$

$\mathcal{H}' = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I', I''; \Gamma; u; \beta)\}$

$\mathbf{Simplification}_{\lambda-\mathbf{var}}$
Hypothèses d'utilisation de la règle :
1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; x\overline{v_P}; y\overline{w_P}; \gamma), I'$
3. $y : \overline{\gamma_P} \multimap \gamma \in \Gamma'$
4. $x := y \in \tau$
5. $I'' = \overline{(\tau_{|v_P,w_P}; \Gamma_{|w_P}; v_P; w_P; \gamma_P)}$

$\mathcal{H}' = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I', I''; \Gamma; u; \beta)\}$

$\mathbf{Simplification}_{FV\mathbf{var}}$
Hypothèses d'utilisation de la règle :
1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; x\overline{v_P}; y\overline{w_P}; \gamma), I'$
3. $y : \overline{\gamma_P} \multimap \gamma \in \Gamma$
4. $x : \overline{\gamma_P} \multimap \gamma \in \sigma$
5. $I'' = \overline{(\tau_{|v_P,w_P}; \Gamma_{|w_P}; v_P; w_P; \gamma_P)}$
6. $\sigma' = x := y$

$\mathcal{H}' = \{(\alpha.L; \sigma \leftarrow \sigma'; \Pi; \Delta; \Lambda; t; I', I''; \Gamma; u; \beta)\}$

$\mathbf{Simplification}_{\lambda-\mathbf{abst}}$
Hypothèses d'utilisation de la règle :
1. $\mathcal{H} = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)\}$
2. $I = (\tau; \Gamma'; \lambda x.v; \lambda y.w; \gamma_1 \multimap \gamma_2), I'$
3. $I'' = (\tau, x := y; \Gamma', y : \gamma_1; v; w; \gamma_2), I'$

$\mathcal{H}' = \{(\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I''; \Gamma; u; \beta)\}$

FIG. 7.9 – Règles de simplification

Les règles de complétion

La règle de complétion sert à reporter au niveau des éléments d'analyse principaux dont ils sont issus les résultats des applications qui ont été faites sur une inconnue d'ordre plus grand que 2. Les mises à jour effectuées par cette règle sont du même ordre que celles effectuées par les règles d'application.

Complétion

Hypothèses d'utilisation de la règle :

1. $\mathcal{H} = \{e; e'\}$
2. $e = (\alpha.L; \sigma; \Pi; \Delta; \Lambda; t; I; \Gamma; u; \beta)$
3. $I = (\tau; \Gamma'; \mathbf{X}\overline{v_{[1,n]}}; w; \gamma), I'$ et $\mathbf{X} : \alpha' \in \Pi$
4. $e' = [\alpha'.0; \sigma'; \Pi'; \Delta'; \mathbf{X}\overline{v_{[1,n]}}; I''; \Gamma'; w; \gamma]$
5. $\Delta'' = \forall(\Gamma' \backslash \Gamma).\Delta'$

$\mathcal{H}' = \{(\alpha.L; \sigma \leftarrow \sigma'; \Pi; \Delta \leftarrow \Delta''; \Lambda \leftarrow \Delta''; t; I', I''; \Gamma; u; \beta)\}$

FIG. 7.10 – Règle de complétion

L'algorithme que nous venons de voir permet d'analyser les grammaires catégorielles abstraites qui codent des grammaires hors contexte de la même manière que l'algorithme défini par Earley [Ear70]. Nous allons voir cela sur un exemple. Nous allons procéder à une analyse dans le langage algébrique $a^n b^n$. La grammaire catégorielle abstraite $\mathcal{G}$ qui code pour ce langage est :

$$\begin{array}{lcl} A & := & \lambda xz.a(x(b(z))) : S \multimap S \\ E & := & \lambda y.y : S \end{array}$$

Les décompositions lexicales utilisées pour les analyses de cette grammaire sont :

1. $\mathbf{X} : S; z : *; \cdot \vdash_{\mathcal{G}} a(\mathbf{X}(b(z))) : S.0$
2. $\mathbf{X} : S; \cdot; \cdot \vdash_{\mathcal{G}} \lambda z.a(\mathbf{X}(b(z))) : S.\cdot$
3. $\cdot; y : *; \cdot \vdash_{\mathcal{G}} y : S.0$
4. $\cdot; \cdot; \cdot \vdash_{\mathcal{G}} \lambda y.y : S.\cdot$

Afin de simplifier la lisibilité de l'exécution de l'algorithme, on supprime l'information de typage des éléments d'analyse, ainsi que des descriptions et comme les décompositions lexicales ne contiennent pas de variables actives libres, nous enlevons également le contexte de désignation. Les éléments d'analyse qui seront manipulés seront de la forme

$$(\alpha.L; \sigma; \Lambda; t; (v, w); u)$$

où $\alpha.L$ est un type abstrait pointé, σ est un contexte de renommage, Λ un contexte d'abstraction, t est la décomposition lexicale utilisée,(v, w) est l'ensemble des éléments d'analyse intermédiaires qui ici est toujours réduit à une paire et ne nécessite jamais de contexte de renommage et u est le terme que l'on est en train d'analyser.

L'analyse qui suit est celle du terme $u = \lambda x.a(a(b(b(x))))$. Toujours pour des raisons de lisibilité, nous avons enlevé les parenthèses des termes. Nous avons noté à chaque étape la famille de règle utilisée ainsi que les éléments d'analyse utilisés pour instancier cette règle :

1. **Init.** $(S.0; y := x; \cdot; y; (y, a\,a\,b\,b\,x); a\,a\,b\,b\,x)$
2. **Init.** $(S.0; z := x; \cdot; a\,\mathbf{X}\,b\,z; (a\,\mathbf{X}\,b\,z, a\,a\,b\,b\,x); a\,a\,b\,b\,x)$
3. $\mathbf{Simp_{const}}$**-(2)** $(S.0; z := x; \cdot; a\,\mathbf{X}\,b\,z; (\mathbf{X}\,b\,z, a\,b\,b\,x); a\,a\,b\,b\,x)$
4. $\mathbf{Pred^{\leq 2}_{Inc}}$**-(3)** $(S.\cdot; \cdot; y : \bot; y; (y, a\,b\,b\,x); a\,b\,b\,x)$
5. $\mathbf{Pred_{Var\ at}}$**-(4)** $(S.\cdot; \cdot; y : \{a\,b\,b\,x\}; y; \cdot; a\,b\,b\,x)$

6. **Pred$^{\leq 2}_{\mathbf{Inc}}$-(3)** $(S.\cdot;\cdot;z:\bot;a\,\mathbf{X}\,b\,z;(a\,\mathbf{X}\,b\,z,a\,b\,b\,x);a\,b\,b\,x)$
7. **App$^{\leq 2}_{\mathbf{Inc}}$-(3-5)** $(S.0;z:=x;a\,\mathbf{X}\,b\,z;(b\,z,a\,b\,b\,x);a\,a\,b\,b\,x)$
8. **Simp$_{\mathbf{const}}$-(6)** $(S.\cdot;\cdot;z:\bot;a\,\mathbf{X}\,b\,z;(\mathbf{X}\,b\,z,b\,b\,x);a\,b\,b\,x)$
9. **Pred$^{\leq 2}_{\mathbf{Inc}}$-(8)** $(S.\cdot;\cdot;y:\bot;y;(y,b\,b\,x);b\,b\,x)$
10. **Pred$^{\leq 2}_{\mathbf{Inc}}$-(8)** $(S.\cdot;\cdot;z:\bot;a\,\mathbf{X}\,b\,z;(a\,\mathbf{X}\,b\,z,b\,b\,x);b\,b\,x)$
11. **Pred$_{\mathbf{Var}\ at}$-(9)** $(S.\cdot;\cdot;y:\{b\,b\,x\};y;\cdot;b\,b\,x)$
12. **App-(8-11)** $(S.\cdot;\cdot;z:\bot;a\,\mathbf{X}\,b\,z;(b\,z,b\,b\,x);a\,b\,b\,x)$
13. **Simp$_{\mathbf{const}}$-(12)** $(S.\cdot;\cdot;z:\bot;a\,\mathbf{X}\,b\,z;(z,b\,x);a\,b\,b\,x)$
14. **Pred$_{\mathbf{Var}\ at}$-(13)** $(S.\cdot;\cdot;z:\{b\,x\};a\,\mathbf{X}\,b\,z;\cdot;a\,b\,b\,x)$
15. **App$^{\leq 2}_{\mathbf{Inc}}$-(3-14)** $(S.0;z:=x;\cdot;a\,\mathbf{X}\,b\,z;(b\,z,b\,x);a\,a\,b\,b\,x)$
16. **Simp$_{\mathbf{const}}$-(15)** $(S.0;z:=x;\cdot;a\,\mathbf{X}\,b\,z;(z,x);a\,a\,b\,b\,x)$
17. **Simp$_{FV\ \mathbf{var}}$-(16)** $(S.0;z:=x;\cdot;a\,\mathbf{X}\,b\,z;\cdot;a\,a\,b\,b\,x)$

D'une façon plus générale, dans l'algorithme de Earley, les items sont de la forme $(\alpha, w_1 \bullet w_2, i, j)$ si $\alpha \rightarrow w_1 w_2$ est une règle. En utilisant la traduction des grammaires hors contexte que nous avons vues au chapitre 2, une telle règle serait représentée par l'entrée lexicale de type $\alpha\ \tilde{w_1} + \tilde{w_2}$ où $\tilde{w_i}$ est le terme qui représente la chaîne w_i et où les non-terminaux ont été remplacés par des inconnues convenablement typées. Dans la notation que nous avons adoptée, un tel item serait représenté par l'élément d'analyse $(\alpha.\cdot;\cdot;z:\bot;\tilde{w_1}+\tilde{w_2};(\tilde{w_2}z,u_j);u_i)$ si u_k est le suffixe qui commence à la $k^{\text{ème}}$ lettre de la chaîne que l'on analyse. En poussant la comparaison, on constate que la règle **Initiation** fonctionne comme la règle Init de Earley, la règle **Simplification$_{\mathbf{const}}$** comme la règle Scan de Earley, la règle **Prédiction$^{\leq 2}_{\mathbf{Inc}}$** comme la règle Predict et la règle **Application$^{\leq 2}_{\mathbf{Inc}}$** comme la règle Complete. Les règles **Prédiction$_{\mathbf{Var}\,at}$** et **Simplification$_{FV\mathbf{var}}$** interviennent seulement pour achever le parcours des termes qui codent les règles. Cette comparaison montre que sur les grammaires catégorielles abstraites qui codent des grammaires hors contexte, cet algorithme a une complexité en $\mathcal{O}(n^3)$.

Informellement, pour constater que cet algorithme permet d'analyser avec une complexité de $\mathcal{O}(n^6)$ les grammaires catégorielles abstraites représentant des grammaires d'arbres adjoints, il nous suffit de voir le coût de l'opération d'adjonction. Dans le codage de grammaires d'arbres adjoints que nous avons vu au second chapitre, les nœuds d'ajonction étaient codés à l'aide de variables d'ordre 3. Si bien que l'opération d'adjonction s'effectue sur des éléments d'analyse secondaire. Au niveau d'un nœud d'adjonction, une entrée lexicale codant pour un arbre est de la forme $\mathbf{X}(\lambda x.w)v$ où w représente l'arbre dominé par le nœud d'adjonction et v représente la partie de l'arbre située à droite de ce nœud d'adjonction, et contient donc une variable libre y qui ferme la chaîne. La réalisation du type de $\mathbf{X}$ par le lexique est $(* \multimap *) \multimap (* \multimap *)$, si de plus $\mathbf{X}$ est le nœud d'adjonction d'un arbre d'adjonction, w peut contenir une variable libre de type $* \multimap *$. Les éléments d'analyse qui vont intervenir dans cette opération seront donc de la forme :

$$[\alpha.0;\cdot;\Pi,\mathbf{X}:\alpha;s:(\bot,\beta.1),y:(\bot,\gamma.1);\mathbf{X}(\lambda x.w)v;(\cdot;x:*;\mathbf{X}(\lambda x.w)v;u_1;*);x:*;u_1;*]$$

$$(\alpha.\cdot;\cdot;\Pi';\cdot;s':(\{u_2:*\}\multimap\{u_3:*\},\alpha.1),z:(\{u_4:*\},\gamma.1);t;\cdot;x:*;u_1;*)$$

$$(\alpha.1;\cdot;\Pi_{|w};s:(\{u_5:*\}\multimap\{u_6:*\},\beta.1);x:(\{u_2:*\},\alpha.1.1);w;\cdot;x:*;u_3;*)$$

Et ainsi, l'opération d'adjonction dépend seulement de 6 sous-termes du terme que l'on analyse. On obtient ensuite l'élément d'analyse secondaire

$$[\alpha.0;\cdot;\Pi,\mathbf{X}:\alpha;s:(\{u_5:*\}\multimap\{u_6:*\},\beta.1),y:(\bot,\gamma.1);\mathbf{X}(\lambda x.w)v;(\cdot;x:*;v;u_4;*);x:*;u_1;*]$$

Dans ce nouvel élément d'analyse il n'y a pas trace du fait que $\lambda x.w$ avait pour description $\{u_2:*\}\multimap\{u_3:*\}$ pour que l'adjonction fonctionne. La règle de **Complétion** dépendra seulement de 5 sous-termes

au cours de l'analyse d'une telle grammaire. Si on avait effectué l'adjonction directement au niveau d'un élément d'analyse principal, on aurait eu une complexité en $\mathcal{O}(n^7)$.

Plus généralement, pour évaluer la complexité de l'algorithme d'analyse, il faut remarquer que les contextes de renommage sont entièrement déterminés par les descriptions associées aux variables actives. Aussi, lorsque l'on évalue pour une grammaire donnée, il faut seulement prendre en compte la complexité des types objet associés à ces variables et la façon dont elles se répartissent dans les termes. Celles qui sont contenues dans des termes du premier ordre ne seront pas instanciées au moment d'une application. D'une façon plus grossière, on peut évaluer la complexité de l'algorithme sur une grammaire quelconque comme étant en $\mathcal{O}(|u|^n)$ où u est le terme à analyser et n est la somme des complexités des types des variables actives contenues dans un terme et à laquelle on ajoute la plus grande complexité des types associés aux inconnues.

Nous allons maintenant effectuer une analyse qui utilise de nombreuses possibilités offertes par cet algorithme, en particulier celle de gérer la liaison des variables. Soit $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ la grammaire définie par :

1. $\mathcal{C}_{\Sigma_1} = \{L; T; Q\}$
2. $\mathcal{A}_{\Sigma_1} = \{np; S\}$
3. $\tau_{\Sigma_1}(L) = np \multimap np \multimap S$ et $\tau_{\Sigma_1}(T) = \tau_{\Sigma_1}(Q) = np$
4. $\mathcal{C}_{\Sigma_2} = \{\forall; \exists; love\}$
5. $\mathcal{A}_{\Sigma_2} = \{e; t\}$
6. $\tau_{\Sigma_2}(love) = e \multimap e \multimap t$ et $\tau_{\Sigma_2}(\forall) = \tau_{\Sigma_2}(\exists) = e \multimap t$
7. $\mathcal{L}(S) = t$, $\mathcal{L}(np) = (e \multimap t) \multimap t$
8. $\mathcal{L}(L) = \lambda so.s(\lambda x.(o(\lambda y.love\, x\, y)))$, $\mathcal{L}(T) = \lambda p.\forall(\lambda x.px)$ et $\mathcal{L}(Q) = \lambda p.\exists(\lambda x.px)$

Les décompositions lexicales utilisées au cours de l'analyse sont :

1. $\mathbf{X} : np, \mathbf{Y} : np; \cdot; \cdot \vdash_{\mathcal{G}} \mathbf{X}(\lambda x.(\mathbf{Y}(\lambda y.love\, x\, y))) : S.\cdot$, nous noterons $\mathcal{L}_L$ le terme $\mathbf{X}(\lambda x.(\mathbf{Y}(\lambda y.love\, x\, y)))$
2. $\mathbf{Y} : np; \cdot; \cdot \vdash_{\mathcal{G}} \lambda x.\mathbf{Y}(\lambda y.love\, x\, y) : np.1$ nous noterons $\mathcal{L}_{L,1}$ le terme $\mathbf{Y}(\lambda y.love\, x\, y)$
3. $\cdot; x : np.1.1; \cdot \vdash_{\mathcal{G}} \lambda y.love\, x\, y : np.1$ nous noterons $\mathcal{L}_{L,2}$ le terme $love\, x\, y$
4. $\cdot; \cdot; \cdot \vdash_{\mathcal{G}} \lambda p.\forall(\lambda x.px) : np.\cdot$ nous noterons $\mathcal{L}_T$ le terme $\forall(\lambda x.px)$
5. $\cdot; \cdot; \cdot \vdash_{\mathcal{G}} \lambda p.\exists(\lambda x.px) : np.\cdot$ nous noterons $\mathcal{L}_Q$ le terme $\exists(\lambda x.px)$

La figure 7.11 présente l'analyse du terme $\forall(\lambda x.(\exists(\lambda y.love\, x\, y)))$. Afin de ne pas surcharger les éléments d'analyse, ceux-ci ne comportent pas l'information de typage objet, il en est de même pour les descriptions syntaxiques. Les éléments d'analyse principaux sont de la forme :

$$(\alpha.L; \sigma; \Delta; \Lambda; t; I; u)$$

où $\alpha.L$ est un type abstrait pointé, σ un contexte de renommage, Δ un contexte de désignation, Λ un contexte d'abstraction, t le terme de la décomposition lexicale utilisée, I un ensemble d'éléments d'analyse intermédiaires et u le terme à analyser.

Les éléments d'analyse seccondaire seront de la forme :

$$[\alpha.0; \sigma; \Delta; t; I; u]$$

et les divers éléments qui les composent ont la même définition que pour les éléments d'analyse principaux.

Les éléments d'analyse intermédiaires ne comportent également pas d'information de typage, ils seront des triplets $(\sigma; t; u)$ σ étant un contexte de renommage, t un sous-terme de décomposition lexicale et u le terme à analyser. La règle $\mathbf{Prédiction}^{\geq 2}_{\mathbf{Inc}}$ sera utilisée et plusieurs éléments d'analyse pourront être créés en une étape. Aussi les éléments d'analyse seront numérotés sur la droite et les numéros de gauche correspondront aux étapes de l'algorithme.

1	**Init**	$(S.0; \cdot; \cdot; \cdot; \mathcal{L}_L; (\cdot; \mathbf{X}(\lambda x.(\mathbf{Y}(\lambda y.love\, x\, y))), \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(1)
2.	**Pred$^{\geq 2}_{\mathbf{Inc}}$-(1)**	$[S.0; \cdot; \cdot; \mathcal{L}_L; (\cdot; \mathbf{X}(\lambda x.(\mathbf{Y}(\lambda y.love\, x\, y))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))]$	(2)
		$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_T; (\cdot; \forall(\lambda x.px); \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(3)
3.	**Pred$^{\geq 2}_{\mathbf{Inc}}$-(1)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_Q; (\cdot; \exists(\lambda x.px); \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(4)
4.	**Simp$_{\mathbf{const}}$-(3)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_T; (\cdot; \lambda x.px, \lambda x.(\exists(\lambda y.love\, x\, y))); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(5)
5.	**Simp$_{\lambda-\mathbf{abst}}$-(5)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_T; (x := x; px, \exists(\lambda y.love\, x\, y)); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(6)
6.	**Pred$_{\mathbf{Var}}$-(6)**	$(np.1; \cdot; \cdot; x : (\bot, np.(1,1)); \mathcal{L}_{L1}; (\cdot; \mathbf{Y}(\lambda y.love\, x\, y); \exists(\lambda y.love\, x\, y)); \exists(\lambda y.love\, x\, y))$	(7)
7.	**Pred$_{\mathbf{Var}}$-(6)**	$(np.1; \cdot; x : (\bot, np.(1,1)); y : (\bot, np.(1,1)); \mathcal{L}_{L2}; (\cdot; love\, x\, y; \exists(\lambda y.love\, x\, y)); \exists(\lambda y.love\, x\, y))$	(8)
8.	**Pred$^{\geq 2}_{\mathbf{Inc}}$-(7)**	$[S.0; \cdot; x : (\bot, np.(1,1)); \mathcal{L}_{L1}; (\cdot; \mathbf{Y}(\lambda y.love\, x\, y); \exists(\lambda y.love\, x\, y)); \exists(\lambda y.love\, x\, y)))]$	(9)
		$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_T; (\cdot; \forall(\lambda x.px); \exists(\lambda y.love\, x\, y)); \exists(\lambda y.love\, x\, y))$	(10)
9.	**Pred$^{\geq 2}_{\mathbf{Inc}}$-(7)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_Q; (\cdot; \exists(\lambda x.px); \exists(\lambda y.love\, x\, y)); \exists(\lambda y.love\, x\, y))$	(11)
10.	**Simp$_{\mathbf{const}}$-(11)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_Q(\cdot; \lambda x.px; \lambda y.love\, x\, y); \exists(\lambda y.love\, x\, y))$	(12)
11.	**Simp$_{\lambda-\mathbf{abst}}$-(12)**	$(np.\cdot; \cdot; \cdot; p : (\bot, np.1); \mathcal{L}_Q; (x := y; px; love\, x\, y); \exists(\lambda y.love\, x\, y))$	(13)
12.	**Pred$_{\mathbf{Var}}$-(13)**	$(np.1; \cdot; \cdot; x : (\bot, np.(1,1)); \mathcal{L}_{L1}; (\cdot; \mathbf{Y}(\lambda y.love\, x\, y); love\, x\, y); love\, x\, y)$	(14)
13.	**Pred$_{\mathbf{Var}}$-(13)**	$(np.1; \cdot; x : (\bot, np.(1,1)); y : (\bot, np.(1,1)); \mathcal{L}_{L2}; (\cdot; love\, x\, y; love\, x\, y); love\, x\, y)$	(15)
14.	**Simp$_{\mathbf{const}}$-(15)**	$(np.1; \cdot; x : (\bot, np.(1,1)); y : (\bot, np.(1,1)); \mathcal{L}_{L2}; (\cdot; x; x), (\cdot; y; y); love\, x\, y)$	(16)
15.	**Pred$_{\mathbf{Var}\ at}$-(16)**	$(np.1; \cdot; x : (\{x\}, np.(1,1)); y : (\bot, np.(1,1)); \mathcal{L}_{L2}; (\cdot; y; y); love\, x\, y)$	(17)
16.	**Pred$_{\mathbf{Var}\ at}$-(17)**	$(np.1; \cdot; x : (\{x\}, np.(1,1)); y : (\{y\}, np.(1,1)); \mathcal{L}_{L2}; \cdot; love\, x\, y)$	(18)
17.	**App$_{\mathbf{Var}}$-(13-18)**	$(np.\cdot; \cdot; \cdot; p : (\forall y.\{y\} \multimap \{love\, x\, y\}, np.1); \mathcal{L}_Q; (x := y; x; y); \exists(\lambda y.love\, x\, y))$	(19)
18.	**Simp$_{\lambda-\mathbf{var}}$-(19)**	$(np.\cdot; \cdot; \cdot; p : (\forall y.\{y\} \multimap \{love\, x\, y\}, np.1); \mathcal{L}_Q; \cdot; \exists(\lambda y.love\, x\, y))$	(20)
19.	**App$^{\geq 2}_{\mathbf{Inc}}$-(9-20-18)**	$[S.0; \cdot; x : (\{x\}, np.(1,1)); \mathcal{L}_{L1}; \cdot; \exists(\lambda y.love\, x\, y)]$	(21)
20.	**Compl-(7-21)**	$(np.1; \cdot; \cdot; x : (\{x\}, np.(1,1)); \mathcal{L}_{L1}; \cdot; \exists(\lambda y.love\, x\, y))$	(22)
21.	**App$_{\mathbf{Var}}$-(6-22)**	$(np.\cdot; \cdot; p : (\forall x.\{x\} \multimap \{\exists(\lambda y.love\, x\, y)\}, np.1); \mathcal{L}_T; (x := x; x, x); \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(23)
22.	**Simp$_{\lambda-\mathbf{var}}$-(23)**	$(np.\cdot; \cdot; p : (\forall x.\{x\} \multimap \{\exists(\lambda y.love\, x\, y)\}, np.1); \mathcal{L}_T; \cdot; \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(24)
23.	**App$^{\geq 2}_{\mathbf{Inc}}$-(2-22-24)**	$[S.0; \cdot; \cdot; \mathcal{L}_L; \cdot; \forall(\lambda x.(\exists(\lambda y.love\, x\, y)))]$	(25)
24.	**Compl-(1-25)**	$(S.0; \cdot; \cdot; \cdot; \mathcal{L}_L; \cdot; \forall(\lambda x.(\exists(\lambda y.love\, x\, y))))$	(26)

FIG. 7.11 – Analyse du terme $\forall(\lambda x.(\exists(\lambda y.love\, x\, y)))$

7.3.4 Perspectives offertes par les descriptions en analyse

La technique mise en œuvre par les descriptions permet d'effectuer des analyses efficaces sur les grammaires de $\mathcal{L}(2,n)$. À terme il serait intéressant de la généraliser à l'ensemble des grammaires catégorielles abstraites. La difficulté ici réside dans le fait que les algorithmes que l'on peut concevoir avec les descriptions reposent sur la tabulation. Une bonne façon de contourner le problème est de ne mémoriser que les éléments d'analyse issus des entrées lexicales et garder un espace de travail pour les termes construits au cours de l'algorithme et qui correspondent à des recherches de démonstration dans **sIIELL**. Il reste cependant à définir la façon dont les inférences logiques faites par l'algorithme peuvent se combiner avec les résultats mémorisés. En effet, dans l'approche que nous avons proposée, l'efficacité réside dans le fait que les descriptions des termes qui remplacent des inconnues ne sont pas stockées au niveau des inconnues. Or la combinaison avec des termes d'ordre supérieur nécessiterait de stocker ces descriptions au niveau des inconnues. Il serait dommage que la généralisation de l'algorithme lui fasse perdre son efficacité sur les grammaires de $\mathcal{L}(2,m)$. Nous n'avons pas encore regardé de suffisamment près tous les problèmes supplémentaires posés par des langages abstraits d'ordre supérieurs pour pouvoir nous prononcer sur la possibilité de conserver de bonnes propriétés de complexité sur les grammaires de $\mathcal{L}(2,m)$ en généralisant cet algorithme.

Un autre domaine sur lequel les descriptions pourraient se révéler un outil précieux est celui de l'expressivité des grammaires catégorielles abstraites. En dehors d'exemples de langages ou de familles de langages qui peuvent être codés dans les grammaires catégorielles abstraites, les langages que peuvent générer les grammaires catégorielles abstraites ne sont pas caractérisés. Or les descriptions syntaxiques permettent de modéliser la combinatoire des λ-termes linéaires et pourraient donc permettre de caractériser formellement les opérations que ceux-ci peuvent effectuer.

Enfin, comme nous l'avons déjà remarqué concernant le filtrage, les descriptions syntaxiques reposent sur des idées qui dépassent le cadre restreint du λ-calcul linéaire. Une source d'inefficacité de l'algorithme que nous avons proposé est le fait que les arguments d'un terme ne sont pas ordonnés de la même façon

que les endroits où ils peuvent être substitués dans le terme. En passant à une logique plus faible avec des flèches orientées, on peut réutiliser ces idées pour construire des algorithmes d'analyse pour des grammaires d'arbres d'ordre supérieur. Il est également possible de porter ces idées au niveau d'un sous-ensemble du λ-calcul simplement typé, pour résoudre des problèmes de filtrage en cas de non linéarité. La restriction imposée aux termes serait que les arguments d'une même variable possèdent au cours d'une analyse la même description. Cette contrainte est une version généralisée de la notion de stratification proposée par [SSS01] pour résoudre en temps polynômial des équations de filtrage de contexte d'arbres. Nous conjecturons qu'une telle démarche permettrait d'analyser bon nombre des termes de sémantique en temps polynômial pour les grammaires de $\mathcal{L}(2, m)$ où le langage objet n'est pas linéaire. Un tel outil serait précieux pour effectuer de la génération. Nous ne prétendons pas que le problème de la génération est polynômial, car nous ne cherchons qu'à analyser des termes sémantiques et ne faisons en somme que de la syntaxe. Tout le problème de la génération consiste alors, partant d'une représentation sémantique, de produire une formule logique représentant cette sémantique et qui puisse être analysée. Une dernière généralisation possible de description serait de les étendre avec les autres connecteurs de la logique linéaire afin d'effectuer des analyses dans des grammaires offrant des structures de contrôle plus fines.

Conclusions et perspectives

L'étude détaillée du problème d'analyse dans les grammaires catégorielles abstraites montre que ce formalisme peut être utilisé comme base pour le traitement automatique des langues. Même si dans le cas général l'analyse est aussi difficile que le problème de recherche de démonstration dans **MELL**, dans des cas particuliers suffisamment riches, une analyse efficace est possible. De plus les algorithmes d'analyse proposés dans cette thèse laissent entrevoir de nombreuses possibilités de développements ultérieurs.

Le premier d'entre eux consiste à trouver une machine abstraite qui permette de caractériser les langages générés par les grammaires catégorielles abstraites. Une telle machine permettrait d'exprimer formellement diverses stratégies d'analyse et de concevoir des algorithmes dédiés à certaines situations. Les descriptions syntaxiques, parce qu'elles permettent des analyses efficaces pour les grammaires de $\mathcal{L}(2,n)$, semblent un point de départ intéressant. De plus, récemment, Makoto Kanazawa a démontré que les grammaires catégorielles abstraites sur les chaînes constituaient une famille complètement abstraite de langages. Ce résultat a pour conséquence qu'il existe une machine abstraite pour ces grammaires. La tâche de la définir n'est cependant pas triviale et il resterait à la généraliser pour une grammaire quelconque.

Par ailleurs, l'algorithme de type Earley que nous proposons pour les grammaires de $\mathcal{L}(2,n)$ est un algorithme tabulaire. La plupart des algorithme de ce genre peuvent se modéliser par l'exécution d'une machine abstraite. Ainsi, la plupart des algorithmes d'analyse des grammaires hors contexte peuvent être interprétés comme l'exécution d'un automate à pile [NS04], ce qui permet de partager pour les stratégies d'analyse une structure commune qui en autorise une implémentation rapide. Trouver une machine abstraite rendant compte des diverses stratégies d'analyse des grammaires catégorielles abstraites ouvrirait la possibilité de transférer aux grammaires catégorielles abstraites l'ensemble des techniques de tabulation déjà développées pour les autres formalismes grammaticaux. En particulier, il nous faudrait comparer les résultats est les performances obtenus par cette machine abstraite avec les *thread automata* proposés par de la Clergerie [de 02].

Concerant cet algorithme que nous proposons, il reste également à voir si on peut le transformer de sorte qu'il possède la propriété des validité des préfixes sans que cela le rende moins efficace. Pour le moment, il nous semble que les idées développées par Nederhof [Ned99] pour l'analyse des grammaires d'arbres adjoints pourraient s'adapter sans trop de difficulté aux grammaires catégorielles abstraites.

Il reste également à caractériser quelles sortes de langages les grammaires catégorielles abstraites peuvent modéliser. En particulier, il n'y a pas de résultat exhibant un langage qui ne puisse être généré par une grammaire catégorielle abstraite. Là encore, les descriptions syntaxiques peuvent être d'un grand secours. Une première approche pour résoudre ce problème consisterait à le résoudre pour les grammaires de $\mathcal{L}(2,n)$ afin de caractériser les opérations que peuvent effectuer les termes linéaires. Ensuite, il faudrait prendre en compte l'apport à l'expressivité des langages abstraits d'ordre supérieur.

Ces résultats quant à l'expressivité ne sont pas anodins et peuvent avoir des retombées concernant les algorithmes d'analyse, mais aussi sur la façon dont on peut concevoir des grammaires catégorielles abstraites à large couverture. En particulier, l'utilisation de l'ordre supérieur rend élégante la formalisation de certains phénomènes linguistiques. Parmi ces phénomènes, il y a la construction des propositions relatives. Un pronom relatif serait modélisé par un type abstrait de la forme $np \multimap (np \multimap S) \multimap np$ et prendrait en premier argument son antécédent et en second argument la proposition relative. Mieux connaître le fonctionnement des grammaires catégorielles abstraites permettrait de savoir dans quelle mesure une telle modélisation permet une analyse efficace. L'algorithme incrémental est déjà une première solution, mais on peut souhaiter d'avoir un algorithme complet pour effectuer une telle tâche.

Un travail qui permettrait de résoudre ce problème consisterait à affiner les résultats obtenus sur la complexité de l'analyse des grammaires catégorielles abstraites. Nous avons établi des critères précis pour connaître la complexité *a priori* des grammaires catégorielles abstraites. Cependant, la démarche adoptée consistait à suivre la hiérarchie des langages établie par $\mathcal{L}(n, m)$. Une étude de la complexité du problème de l'analyse qui soit transversale à cette classification reste à faire. Il serait en particulier intéressant de trouver une classe de grammaires catégorielles abstraites dont le langage abstrait contient des constantes d'ordre supérieur à 3 et dont l'analyse est polynômiale. Une raison qui nous fait penser qu'une telle classe existe est que l'un des premiers encodages des grammaires d'arbres adjoints dans les grammaires catégorielles abstraites utilisait une grammaire de $\mathcal{L}(3, 2)$.

Une partie du travail que nous avons mené mais qui ne nous semblait pas suffisamment mûre pour figurer dans ce mémoire porte sur les extensions des grammaires catégorielles abstraites. Étendre le formalisme des grammaires catégorielles abstraites peut se faire suivant plusieurs directions. Tout d'abord on peut étendre le système de typage en utilisant les autres connecteurs de la logique linéaire. Nous avons étudié dans quelle mesure cela était possible en montrant que le filtrage restait décidable si on ajoute le connecteur & [PdG04a]. Il demeure même NP-complet dans le cas où le membre de droite de l'équation est sous forme normale. Nous avons pu également étendre ce résultat à l'ensemble des connecteurs de la logique linéaire ($\multimap$, $\otimes$, $\mathbf{1}$, & et $\oplus$). Ces résultats sont intéressants, mais nous n'avons pas encore de techniques efficaces d'analyse. Une autre direction consiste à passer au premier ordre. Cela peut se faire de deux façons, ou bien on peut ajouter des quantificateurs du premier ordre, ce qui ajoute au grammaires catégorielles abstraites une notion rudimentaire de structures de traits. Les quantificateurs du premier ordre donnent également un contrôle plus précis de la structure des termes d'analyse. Il devient en effet possible de coder les grammaires de Lambek, alors qu'aucun codage n'est connu pour les grammaires catégorielles abstraites. La seconde façon de passer à l'ordre supérieur est d'utiliser les types dépendants. Cette direction semble prometteuse car on peut coder simplement les grammaires de concaténation d'intervalles (RCG) [Bou00] dans les grammaires ainsi obtenues. Or les RCG permettent de modéliser tout langage analysable en temps polynômial. De plus l'algorithme d'analyse incrémental peut facilement être étendu à ce type de grammaire. Systématiser la démarche que nous avons déjà utilisée pour étudier les grammaires catégorielles abstraites étendues devrait donner rapidement des résultats pour ces diverses extensions.

Finalement, pour compléter un système de modélisation du langage il faut lui ajouter des capacités d'apprentissage. Cette nécessité se fait fortement ressentir pour créer des grammaires à large couverture. Or Makoto Kanazawa propose une technique d'apprentissage à partir de la sémantique se basant sur des calculs d'interpolants [Kan03], [Kan05]. Cette technique nécessite encore du travail et il reste à voir si elle peut se transposer au niveau de la syntaxe.

Bibliographie

[Ajd36] Kazimierz Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica I*, pages 1–27, 1936.

[AKJ75] Masako Takahashi Aravind K. Joshi, Leon S. Levy. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1) :136–163, 1975.

[AKJ92] Yves Shabes Aravind K. Joshi. Tree-adjoining grammars and lexicalized grammars. *Tree Automata and LGS*, 1992.

[Ang80] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer ans System Sciences*, 21 :46–62, 1980.

[BG01] Guillaume Bonfante and Bruno Guillaume. Vector addition word automata. 2001.

[BH50] Yehoshua Bar-Hillel. On syntactical categories. *The Journal of Symbolic Logic*, 15 :1–16, 1950.

[Bou00] Pierre Boullier. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February 2000.

[CDG+99] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. http ://www.grappa.univ-lille3.fr/tata/, 1999.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Transaction on Information Theory*, 2(3) :113–124, september 1956.

[Chu41] Alonso Church. *The calculi of Lambda Conversion*. Princeton University Press, 1941.

[Coo71] Stephen A. Cook. The complexity of theorem proving procedures. *Proceedings of the 3rd annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[dB72] de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Math.*, 34 :381–392, 1972.

[de 02] Eric de la Clergerie. Parsing MCS languages with thread automata. In *TAG+6, Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 193–200, Venice, Italy, May 2002.

[dG00a] Philippe de Groote. Higher-order linear matching is np-complete. In *proc of the 11th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, pages 127–140. RTA, 2000.

[dG00b] Philippe de Groote. Proof-search in implicative linear logic as a matching problem. In *LPAR*, pages 257–274, 2000.

[dG01] Philippe de Groote. Towards abstract categorial grammars. In Association for Computational Linguistic, editor, *Proceedings 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155. Morgan Kaufmann Publishers, 2001.

[dG02] Philippe de Groote. Tree-adjoining grammars as abstract categorial grammars. *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150, 2002.

[dGP03] Philippe de Groote and 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, USA*, pages 71–80, Jun 2003.

[Dow94] Gilles Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 69 :135–155, 1994.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2) :94–102, February 1970.

[GB03] G. Perrier G. Bonfante, B. Guillaume. Analyse syntaxique électostatique. *Traitement Automatique des Langues*, 44 :3 Évolutions en analyse syntaxique, 2003.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13 :225–230, 1981.

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω*. Thèse de doctorat es sciences mathématiques, Université Paris VII, 1976.

[Kan92] Max I. Kanovitch. Horn programming in linear logic is np-complete. *7-th IEEE symposium on Logic in Computer Science*, pages 200–210, 1992.

[Kan03] Makoto Kanazawa. Computing word meanings by interpolation. In *Proceedings of the Fourteenth Amsterdam Colloquium*, pages 157–162, ILLC/Department of Philosophy, University of Amsterdam, 2003.

[Kan05] Makoto Kanazawa. Computing interpolants in implicational logics. NII Technical Report NII-2005-003E, National Institute of Informatics, Tokyo, 2005.

[Kas65] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical Report Science Report AFCRL-65-758, Air Force Cambridge Research Laboratory, 1965.

[Kil92] Pekka Kilpeläinen. *Tree Matching Problems with Aplications to Structured Text Databases*. Ph.D. thesis, Departement of Computer Science, University of Helsinki, Finland, 1992.

[Kip91] J.R. Kipps. *GLR parsing in time $\mathcal{O}(n^3)$*, chapter 4, pages 43–59. Kluwer Academic Publishers, 1991.

[KNR03] J. Khegai, B. Nordström, and A. Ranta. Multilingual syntax editing in gf. *Lecture Notes in Computer Science*, 2588 :453–464, 2003.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65 :154–170, 1958.

[Loa03] Ralph Loader. Higher order β matching is undecidable. *Logic Journal of the IGPL*, 11(1) :51–68, 2003.

[Mar97] Diego Marconi. *La philosophie du langage au vingtième siècle*. éditions de l'éclat, 1997. traduit de l'italien par Michel Valensi.

[Mon74] Richard Montague. *Formal Philosophy : Selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.

[Mor00] Glyn Morrill. Incremental processing and acceptability. *Comput. Linguist.*, 26(3) :319–338, 2000.

[Ned93] Mark-Jan Nederhof. Generalized left-corner parsing. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 305–314, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[Ned99] Mark-Jan Nederhof. The computational complexity of the correct-prefix property for tags. *Comput. Linguist.*, 25(3) :345–360, 1999.

[NS96] Mark-Jan Nederhof and Giorgio Satta. Efficient tabular lr parsing. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 239–246, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[NS04] Mark-Jan Nederhof and Giorgio Satta. *Formal Languages and Applications, Studies in Fuzziness and Soft Computing*, volume 148, chapter Tabular Parsing, pages 529–549. Springer, 2004.

[Pad00] Vincent Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3) :361–372, 2000.

[PdG03] Sylvain Salvati Philippe de Groote. On the compexity of higher order matching in the linear λ-calculus. *Lecture Notes in Computer Science*, 2706 :234–245, 2003.

[PdG04a] Sylvain Salvati Philippe de Groote. Higher-order matching in the linear λ-calculus with pairing. In *Proc. of the 13th Conference, CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 220–234, 2004.

[PdG04b] Sylvain Salvati Philippe de Groote, Bruno Guillaume. Vector addition tree automata. *19-th IEEE symposium on Logic in Computer Science*, pages 63–74, 2004.

[Pog04] 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.

[Ran04] Aarne Ranta. Grammatical framework : A type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2) :145–189, 2004.

[Shi85] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8 :333–343, 1985. Reprinted in Walter J. Savitch, Emmon Bach, William Marsh, and Gila Safran-Navah, eds., *The Formal Complexity of Natural Language*, pages 320–334, Dordrecht, Holland : D. Reidel Publishing Company, 1987. Reprinted in Jack Kulas, James H. Fetzer, and Terry L. Rankin, eds., *Philosophy, Language, and Artificial Intelligence*, pages 79–92, Dordrecht, Holland : Kluwer Academic Publishers, 1988.

[SJ88] Yves Schabes and Aravind K. Joshi. An earley-type parsing algorithm for tree adjoining grammars. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pages 258–269, Morristown, NJ, USA, 1988. Association for Computational Linguistics.

[SSS01] Manfred Schmidt-Schauß and Jürgen Stuber. On the complexity of linear and stratified context matching problems. Rapport de recherche A01-R-411, LORIA, December 2001.

[Tes59] Lucien Tesnière. *Élément de syntaxe structurale.* librairie C. Klinksieck, Paris, 1959.

[Tom87] Masaru Tomita. An efficient augmented-context-free parsing algorithm. *Comput. Linguist.*, 13(1-2) :31–46, 1987.

[Tro92] Anne S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.

[VSJ85] K. Vijay-Shankar and Aravind K. Joshi. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd annual meeting on Association for Computational Linguistics*, pages 82–93, Morristown, NJ, USA, 1985. Association for Computational Linguistics.

[Yos05] Ryo Yoshinaka. Higher-order matching in the linear lambda calculus in the absence of constants is np-complete. In *proc of the 16th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science. RTA, 2005.

[You67] D.H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10 :572–597, 1967.