Jeux de typage et analyse de lambda-grammaires non-contextuelles

Pierre Bourreau

N° d'ordre : 4538

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Pierre BOURREAU**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Jeux de Typage et Analyse de λ-Grammaires Non-Contextuelles

Soutenue le : 29 Juin

Après avis des rapporteurs :

Mme Isabelle TELLIER	Professeure, Université Paris 3
M. Philippe DE GROOTE	Professeur, INRIA

Devant la commission d'examen composée de :

M. Sylvain SALVATI	Chargé de Recherche, INRIA	Directeur de thèse
M. Christian RETORÉ	Professeur, Université Bordeaux 1	Directeur de thèse
Mme Isabelle TELLIER	Professeure, Université Paris 3	Rapporteuse
M Philippe DE GROOTE	Directeur de Recherche, INRIA	Rapporteur
Mme Claire GARDENT	Directrice de Recherche, CNRS	Examinatrice
M Christophe FOUQUERÉ	Professeur, Université Paris 13	Examinateur
M. Géraud SÉNIZERGUES	Professeur, Université Bordeaux 1	Président du Jury

– 2012 –

Je tiens tout d'abord à remercier Sylvain, qui m'a soutenu tout au long de ce travail et sans qui ce manuscrit n'aurait jamais pu être. J'ai appris énormément à ton contact pendant ces années, je suis arrivé à Bordeaux en tant qu'étudiant, et je pense avoir pu aiguiser mon esprit scientifique grâce à toi. Je te remercie donc de m'avoir transmis ta passion et ton enthousiasme. Je remercie également Christian, qui m'a été d'une grande aide dans l'aboutissement de ce travail et qui a su me faire part de ses remarques toujours constructives. Une des personnes sans qui je ne serai jamais arrivé à Bordeaux, et sans qui je ne me serai jamais intéressé à la linguistique informatique est Glyn Morrill. Si je ne l'avais pas rencontré lors de mon passage à Barcelone, je ne me serais sans doute pas embarqué dans l'aventure du monde de la recherche. Je remercie bien entendu Philippe de Groote et Isabelle Tellier d'avoir accepté de rapporter mon mémoire de doctorat, Christophe Fouqueré et Claire Gardent d'avoir accepté de participer à mon jury, et Géraud Sénizergues qui a présidé ce jury.

Les années passées au LaBRI m'auront permis de rencontrer de nombreuses personnes avec lesquelles j'ai passé de très bons moments. Anaïs, je te souhaite d'arriver au bout de ton travail et de garder la même énergie que celle qui t'a caractérisée depuis ton arrivée à Bordeaux ; Natalia, j'espère que la Belgique te permettra de reprendre des forces et de construire l'avenir que tu mérites ; Julien, merci pour les blagues perpétuelles et ta bonne humeur qui résiste à toute épreuve (on doit encore réparer mon ampli, n'oublie pas...) ; Adrien, sans qui je ne connaîtrais pas l'existence de ratpoison (Nay est vraiment une planète à part) ; Aurélie, plein de bonheur avec tes petits ; Hugo, tu auras été mon parrain lors de ces premières années de thèse... à quand le prochain mascaret à 6h du matin ? Simon et Michael, pour vos chamailleries de geeks qui venaient pimenter nos repas de midi. Valentin, Julien, Florent, Bruno, Willy et tous les doctorants que j'oublie et avec qui j'ai pu partager un peu de temps.

Ces années à Bordeaux m'ont également permis de rencontrer beaucoup de personnes qui m'ont apporter énormément d'équilibre. Nico, ces talents de conteur, sa nonchalance, et son amour des 4L ; Benoît et sa patience, Romain, Julie, Henri, Cindy, Patrick, Thierno, et toutes les personnes qui font vivre Survie Gironde. Il y a bien sûr Juliette et Steven, qui ont toujours été présents et qui ont une énergie et une soif de vie tellement communicatives ; Flore et son interprétation des proverbes français ; Hernan : j'espère que tu finiras ta thèse bientôt, et que je pourrais en savoir plus sur le passage des Romains en Tunisie. J'oublie certainement de nombreuses personnes, mais je remercie tous ces gens qui font que j'ai pu trouver des racines à Saint-Michel, à Bordeaux.

Parmi ces personnes, il y a bien sûr Sian, qui a dû me supporter cette dernière année. Je te remercie mille fois de ton soutien, de ta patience et de me rendre heureux tous les jours. J'espère te le rendre encore très longtemps.

Je pense à ma famille et à une personne très particulière pour moi : mon oncle. Je sais que tu seras aussi heureux que moi le 29 Juin prochain. Je tiens à dédier ce travail à ma soeur et à mon père, pour les longues et difficiles semaines que nous avons passées cet hiver, à partager nos angoisses, nos espoirs ; ces semaines qui marquent une vie, et qui nous ont permis de mieux nous comprendre. Bien entendu, je dédie également ce travail à ma mère, qui nous fait le bonheur d'être encore présente aujourd'hui.

Enfin, je dédie ce travail à Moussa, Baba, Mohamed et tous les amis maliens. On est ensemble.

Jeux de Typage et Analyse de λ-Grammaires Non-Contextuelles

Résumé : Les grammaires catégorielles abstraites (ou λ-grammaires) sont un formalisme basé sur le λ-calcul simplement typé. Elles peuvent être vues comme des grammaires générant de tels termes, et ont été introduites afin de modéliser l'interface entre la syntaxe et la sémantique du langage naturel, réunissant deux idées fondamentales : la distinction entre tectogrammaire (*c.a.d.* structure profonde d'un énoncé) et phénogrammaire (*c.a.d* représentation de la surface d'un énoncé) de la langue, exprimé par Curry ; et une modélisation algébrique du principe de compositionnalité afin de rendre compte de la sémantique des phrases, due à Montague. Un des avantages principaux de ce formalisme est que l'analyse d'une grammaires catégorielle abstraite permet de résoudre aussi bien le problème de l'analyse de texte, que celui de la génération de texte. Des algorithmes d'analyse efficaces ont été découverts pour les grammaires catégorielles abstraites de termes linéaires et quasi-linéaires, alors que le problème de l'analyse est non-élémentaire dans sa forme la plus générale. Nous proposons d'étudier des classes de termes pour lesquels l'analyse grammaticale reste solvable en temps polynomial. Ces résultats s'appuient principalement sur deux théorèmes de typage : le théorème de cohérence, spécifiant qu'un λ-terme donné est l'unique habitant d'un certain typage ; et le théorème d'expansion du sujet, spécifiant que deux termes β-équivalents habitent les même typages. Afin de mener cette étude à bien, nous utiliserons une représentation abstraite des notions de λ-termes et de typages, sous forme de jeux. En particulier, nous nous appuierons grandement sur cette notion afin de démontrer le théorème de cohérence pour de nouvelles familles de λ-termes et de typages. Grâce à ces résultats, nous montrerons qu'il est possible de construire de manière directe, un reconnaisseur dans le langage Datalog, pour des grammaires catégorielles abstraites de λ-termes quasi-affines.

Mots clés : grammaires catégorielles abstraites, effacement, λ-calcul simplement typé, génération textuelle, génération sémantique, Datalog, cohérence, jeux de typage.

Discipline : Informatique

LaBRI (UMR CNRS 5800)
Université Bordeaux 1
351, cours de la Libération
33405 Talence Cedex (FRANCE)

Table des matières

Table des figures

Chapitre 1

Introduction

L'informatique, d'après l'Académie française se définit comme "la science du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances humaines et des communications dans les domaines technique, économique et social". D'après cette définition, les outils informatiques développées le sont à deux fins : traiter certains raisonnements de manière automatique et créer des outils de communication efficaces. Une forme naturelle à l'Homme d'exprimer et de véhiculer l'information est le langage. Il est donc légitime de se poser la question du traitement par les ordinateurs d'informations véhiculées par le langage. Cette question fut donc soulevée dès l'avènement de l'informatique, comme en atteste, par exemple, le célèbre test de Turing [Turing, 1950]. Bien que largement discutable, ce test met en exergue différents défis de l'intelligence artificielle, en particulier, la possibilité pour une machine de raisonner, et de communiquer, tel un être humain. Le test se réalise de la manière suivante : une personne A communique, par le langage humain, avec deux interlocuteurs I1 et I2, dont elle est physiquement isolée. Un de ces interlocuteurs est une personne, l'autre étant une machine. Si la personne A ne peut différencier l'interlocuteur humain, de l'interlocuteur artificiel, alors le test est passé avec succès, ce qui est censé conclure à la possibilité pour une machine d'être dôtée de facultés de communication et de raisonnement similaires à celles de l'Homme. Dans cette thèse, nous nous intéressons uniquement à la faculté d'analyse et de génération de langues humaines par une machine ; en particulier, nous regarderons comment une machine peut analyser ou générer des phrases. Toute composante cognitive du langage sera donc exclue de notre étude.

En abordant cette tâche, nous pouvons constater, tout d'abord, la complexité même de cet outil qu'est la langue. Les énoncés sont effectivement des chaînes très structurées, et d'une complexité souvent élevée, comme chacun peut s'en rendre compte durant l'apprentissage d'une langue nouvelle ; par ailleurs, la complexité d'une langue se retrouve tout simplement dans l'ambiguïté de certains énoncés. Qui plus est, les mécanismes de construction de ces énoncés sont, en apparence, très différents d'une langue à une autre. Le travail des linguistes consiste, entre autre, à répertorier et à classifier les langues, d'une part, et d'autre part, à étudier les différents aspects de ces dernières. Nous pouvons ainsi citer la phonétique, qui est l'étude des sons d'une langue ; la prosodie, qui est l'étude des variations mélodiques, temporelles, ... des sons dans un énoncé ; la morphologie qui est l'étude de la construction des mots et leur catégorisation ; la syntaxe, qui est l'étude de la correction grammaticale d'une phrase ; la sémantique, qui est l'étude du sens d'un énoncé ; la pragmatique, enfin, qui est l'étude de la relation entre un énoncé et un utilisateur de celui-ci (locuteur, auditeur, lecteur, ...). La question

du traitement par les ordinateurs de l'information transmise par la langue peut donc également être divisée en plusieurs tâches dépendantes, consistant chacune à traiter une des facettes de la langue. Les travaux que nous présentons par la suite, se focalisent sur l'étude de l'interface entre syntaxe et sémantique de phrases. Plus exactement, nous adopterons des modèles de représentation de ces deux aspects d'une phrase, et étudierons des méthodes automatiques permettant de passer de l'une à l'autre de ces représentations.

D'un point de vue technique, une des premières questions est de connaître la puissance calculatoire nécessaire afin de traiter la langue, ou du moins, les problèmes liés à l'interface entre syntaxe et sémantique. Une des théories qui a été créé afin de décrire la notion de puissance de calcul a été exprimée dans [Church, 1936] sous une forme algébrique. Les éléments de cette algèbre, appelés des λ-termes, peuvent être réécrits par itérations d'opérations représentant des étapes de calcul. Ces termes peuvent être vus comme des fonctions, la composition de fonction correspondant à un calcul. Le λ-calcul a par ailleurs été à l'origine de l'avènement d'un style de programmation particulier, la programmation fonctionnelle. Différents fragments du λ-calcul peuvent être étudiés ; nous nous intéresserons ici au λ-calcul simplement typé. De plus, cet outil sera utilisé non seulement afin de parler de puissance calculatoire mais également et sutout comme modèles de représentation pour des structures telles que les arbres et des formules de la logique du premier-ordre, ces dernières étant communément utilisées en syntaxe et sémantique formelle.

Ainsi, les travaux que nous présenterons s'intéressent à la calculabilité des sens à partir des phrases (ce que nous appellerons l'analyse grammaticale, ou génération sémantique) et inversement (ce que nous appellerons la génération de textes). Qui plus est, l'informatique met à notre disposition de nombreux outils qui traitent l'information ; en particulier, des outils d'analyse et de génération de langages différents du langage humain (langages de programmations, codes binaires, ...). Ainsi, le langage humain est-il si distinct des langages utilisés en informatique ? Nous est-il possible de réutiliser des techniques informatiques existantes et, en particulier, celles utilisées dans le traitement des langages informatiques ?

Richard Montague, philosophe et mathématicien américain proposa une solution partielle à cette question dans les années 1970. Selon lui, le langage humain peut être traité comme un langage informatique. Il proposa de modéliser syntaxe et sémantique sous une forme algébrique, et leur interface par des morphismes. En particulier, il introduisit le λ-calcul simplement typé dans la sémantique formelle des langues, en l'utilisant afin de représenter le sens des entrées lexicales composant une phrase. Le calcul des sens d'une phrase se réduit alors à un problème calculatoire sur le λ-calcul. Cette proposition offrit une perspective nouvelle à des travaux issus de la logique, sur l'analyse grammaticale des phrases. Ainsi, différents systèmes logiques ont été proposés par [Adjukiewicz, 1935, Bar-Hillel, 1950] puis par [Lambek, 1958], sous le nom de *grammaires catégorielles*, où l'analyse grammaticale des phrases se réduit à un problème de déduction logique. Or, le λ-calcul simplement typé étant en lien étroit avec la logique (à travers un morphisme, à l'image de ceux décrits par Montague), l'analyse grammaticale des phrases permet donc, de vérifier la correction grammaticale de celles-ci, d'une part, et de calculer leurs représentations sémantiques, d'autre part.

Les grammaires catégorielles abstraites (ou λ-grammaires), formalisme créé indépendamment par [de Groote, 2001] et [Muskens, 2001], sont issues de ces idées. De fait, il est possible de décrire ces grammaires par la théorie de Montague : les algèbres représentant les formes syntaxiques ou les formes sémantiques des phrases sont des algèbres de construction de λ-termes simplement typés, et l'interface entre la syntaxe et la sémantique est réalisée par des morphismes de termes. Ainsi, le λ-calcul y est utilisé afin de représenter les sens d'une phrase (*c.a.d.* sa représentation sémantique), ses dérivations, mais aussi la structure des chaînes de caractères les composant (*c.a.d.* sa représentation de surface). Le formalisme des grammaires catégorielles abstraites se présente donc comme

un formalisme uniforme, puisque les termes du λ-calcul simplement typé sont utilisés afin de décrire chaque forme de représentation d'une phrase. De plus, la distinction entre les dérivations d'une phrase (ses structures) et sa réalisation de surface (les chaînes de caractères la composant) y est exprimée de manière explicite. En effet, les constructions de surface sont également obtenues par image homomorphique des termes des dérivations syntaxiques. Ainsi, à l'image des idées de Montague, mais aussi de celles exprimées par [Curry, 1961], le sens et la réalisation de surface sont toutes les deux obtenues comme images homomorphiques des dérivations syntaxiques. Ce point est essentiel afin de comprendre la symétrie entre les tâches de génération de textes et d'analyse grammaticale. Dans le cadre des grammaires catégorielles abstraites, une solution au problème de l'analyse de ces grammaires offre une solution à ces deux tâches.

De plus, ces grammaires s'inscrivent dans une longue tradition de l'informatique fondamentale, puisque de nombreux travaux ont été menés sur les grammaires de chaînes, sur les grammaires d'arbres, ou encore sur les grammaires de graphes. La structure des données considérée dans le cas des grammaires catégorielles abstraites est le λ-terme. De plus, si les termes représentant les dérivations de ces grammaires ont une structure arborescente, nous parlerons de λ-grammaires non-contextuelles (ou de grammaires catégorielles abstraites non-contextuelles), faisant ainsi référence au fait que les grammaires non-contextuelles de chaînes ou d'arbres construisent des dérivations qui peuvent etre représentées sous forme d'arbres. Dans le cadre des présents travaux, nous ne nous intéresserons qu'aux grammaires catégorielles abstraites non-contextuelles, entre autres raisons du fait que les dérivations syntaxiques des phrases ont généralement une forme arborescentes pour les linguistes. De plus, l'utilisation des grammaires catégorielles abstraites non-contextuelles suffit à modéliser nombre de phénomènes syntaxiques ou sémantiques.

Par ailleurs, ces grammaires ont également l'avantage de permettre de représenter certaines classes de langages connues. Ainsi, il est possible de représenter les grammaires de chaînes ou les grammaires d'arbres comme des grammaires catégorielles abstraites. Il est également possible de représenter certains formalismes utilisés dans le domaine de la recherche en linguistique informatique (tels que les grammaires d'arbres adjoints ou les grammaires non-contextuelles multiples de chaînes) comme des λ-grammaires non-contextuelles. Les grammaires catégorielles abstraites peuvent donc être vues comme une généralisation de grammaires connues. Par conséquent, elles nous donnent un cadre nouveau afin de résoudre des problèmes rencontrés dans d'autres formalismes, et nous amènent, de plus, à définir des méthodes nouvelles de traitement du langage applicables à ces formalismes.

Il en est ainsi pour les méthodes d'analyse et de génération de textes. En effet, les premières méthodes furent étudiées dans [Salvati, 2005], qui montre que le problème de l'analyse dans ces grammaires se réduit à un problème de typage de λ-termes. Le slogan "parsing as deduction" (*analyse comme déduction logique*) que l'on peut appliquer aux grammaires catégorielles ou aux analyseurs logiques, peut donc être raffiné en "parsing as type checking" (*analyse comme vérification de typage*) dans le cas des grammaires catégorielles abstraites. Le principe de cette méthode d'analyse est effectivement d'identifier des termes équivalents grâce à leurs propriétés de typage. Ces idées donnèrent lieu à deux résultats importants : tout d'abord, le problème de l'analyse dans les grammaires catégorielles abstraites non-contextuelles est décidable [Salvati, 2007]. Ce problème montre donc que la puissance calculatoire décrite par Turing est suffisante afin de résoudre les problèmes d'analyse et de génération de textes dans le cadre de la sémantique de Montague. Néanmoins, les algorithmes ainsi mis en lumière requièrent une puissance calculatoire particulièrement élevée, leur classe de complexité étant non-élémentaire, et ne sont donc pas des algorithmes efficaces. Par ailleurs, [Kanazawa, 2007] montre que, sous certaines contraintes de linéarité sur les λ-termes, ces algorithmes s'exécutent en temps polynomial. La question à laquelle nous nous intéressons dans ces travaux est de savoir jusqu'à quel point il nous est possible de relâcher les contraintes de linéarité tout en gardant des algorithmes

d'analyse de λ-grammaires s'exécutant en temps polynomial.

Cette thèse se situe donc à l'intersection entre la linguistique informatique, la théorie des langages formels, et le λ-calcul simplement typé. Les méthodes d'analyse que nous étudions se basant sur des propriétés de typage de λ-termes, nous ferons appel à une représentation particulière de cette notion de typage nous permettant d'avoir un accès direct aux notions essentielles à l'étude des propriétés de typage que nous cherchons. Ces propriétés sont principalement la recherche de termes qui peuvent être uniquement identifiés par certains de leurs typages. De telles propriétés reposent sur la notion d'interaction entre types atomiques composant un typage. L'introduction de modèles du λ-calcul (et de la logique) sous forme de jeux s'intéresse exactement à cette interaction. Une telle notion de jeux date des années 1950, et a pour vocation initiale l'étude des dérivations dans les systèmes déductifs logiques sous forme d'interaction entre joueurs. L'utilisation des jeux que nous proposons ici n'a pas pour but d'étudier des modèles du λ-calcul simplement typé, mais plutôt d'utiliser cette notion d'interaction afin de prouver des propriétés d'unicité entre typages et λ-termes. En particulier, certaines notions telles que la polarité des types ou encore la notion d'occurrences de types atomiques seront plus facilement accessibles grâce à la notion de jeux.

Les travaux menés durant cette thèse seront présentés selon l'ordre suivant ; dans un premier temps, nous introduirons les notions relatives au λ-calcul qui nous seront utiles au cours de cette recherche. Ainsi, le Chapitre 2 sera consacré à une présentation succincte et formelle du λ-calcul et de la notion de typage sur celui-ci. Nous énoncerons les propriétés calculatoires fondamentales de ce formalisme, la relation entre λ-calcul simplement typé et logique. Enfin, nos définirons les familles de termes linéaires et affines, ces dernières étant connues pour vérifier les propriétés de typage auxquelles nous nous intéressons. Au cours du Chapitre 3, nous présenterons la notion de jeux relative au λ-calcul simplement typé. Nous y détaillerons les analogies entre les coups d'un jeu, les types atomiques d'un typage, et les variables d'un terme ; entre les notions de joueurs et de polarités de types ; et enfin, entre stratégies, dérivations logiques et λ-termes. Enfin, nous présenterons un premier résultat concernant la caractérisation de typages principaux comme stratégies recouvrantes.

La seconde partie de cette thèse sera dédiée l'introduction des grammaires catégorielles abstraites dans le cadre de la linguistique informatique. Ainsi, après une brève introduction à la notion de grammaire formelle, le Chapitre 4 présentera deux formalismes utilisés dans l'étude du langage humain : les grammaires d'arbres adjoints, et les grammaires non-contextuelles multiples. La classe de langages à laquelle appartiennent les langages générés par ces formalismes sera comparée à d'autres classes de langages connues ; enfin, nous présenterons les grammaires catégorielles abstraites et mettrons en avant le lien entre ces grammaires et les formalismes grammaticaux précédemment introduits. Dans le Chapitre 5, nous présenterons quelques notions fondamentales de la programmation logique et montrerons l'utilité de ce style de programmation dans la conception d'analyseurs grammaticaux, en particulier pour les formalismes grammaticaux décrits au Chapitre précédent. Nous conclurons par la présentation d'un reconnaisseur de grammaires catégorielles abstraites non-contextuelles de λ-termes linéaires et quasi-linéaires dans le langage Datalog [Kanazawa, 2007], donnant lieu à une construction directe et élégante de reconnaiseurs à partir d'une grammaire donnée.

Finalement, nous conclurons ce document par une étude approfondie des théorèmes nécessaires à la construction de reconnaiseur de λ-grammaires non-contextuelles de termes, au-delà des termes quasi-linéaires, en réutilisant les idées de [Kanazawa, 2007]. Dans un premier temps, nous nous attacherons à l'étude de la propriété d'unicité entre typages et λ-termes. Nous aborderons ainsi la complexité de la forme la plus générale de ce problème, puis nous caractériserons certaines familles de termes respectant cette propriété avec leurs typages principaux, et un particulier, la famille des termes quasi-affines. Le Chapitre 7 s'intéressera, enfin, au problème de l'analyse pour des λ-grammaires non-contextuelles de termes quasi-affines. Au cours de cette dernière partie, la notion de jeu sera largement

utilisée afin de résoudre les problèmes techniques auxquels nous nous confrontons. Nous détaillerons les problèmes théoriques relatifs à l'étude de l'analyse dans ces grammaires, puis la construction des programmes Datalog permettant de résoudre ce problème.

Première partie

λ-calcul, typage et jeux

Chapitre 2

λ-calcul et typage

Le λ-calcul est né des travaux d'Alonzo Church au milieu du vingtième siècle [Church, 1936, Church, 1941]. Ce calcul, qui se caractérise par sa simplicité syntaxique et son pouvoir expressif, est un outil fondamental de l'informatique théorique ; en effet, il permet, au même titre que les machines de Turing, de représenter sous forme mathématique un processus de calcul informatique, c'est-à-dire l'exécution d'un programme informatique. C'est ainsi que les langages de programmation dits *fonctionnels* tels que LISP, Scheme ou plus récemment CaML, sont basés sur le λ-calcul. Comme nous le verrons dans le Chapitre 4, il représente également un intérêt certain du point de vue de la sémantique formelle des langues naturelles. En effet, il est possible de représenter des formules logiques (que l'on peut considérer comme une représentation sémantique d'un énoncé) comme des λ-termes. De manière plus générale, nous utiliserons le λ-calcul afin de représenter différentes structures de données, tels les arbres ou les chaînes.

2.1 Syntaxe et opérations du λ-calcul non-typé

Dans cette section, nous définissons la syntaxe générale du λ-calcul, ainsi que les règles de réécriture usuelles.

2.1.1 Syntaxe du λ-calcul non-typé

Définition 2.1.1.1. *Soient $\mathcal{V}$ un ensemble dénombrable de variables et C un ensemble dénombrable de constantes. L'ensemble Λ des λ-*termes *est défini inductivement comme suit :*

1. *si $x \in \mathcal{V}$, alors $x \in \Lambda$*
2. *si $\mathbf{c} \in C$, alors $\mathbf{c} \in \Lambda$*
3. *si $x \in \mathcal{V}$ et $M \in \Lambda$, alors $(\lambda x.M) \in \Lambda$*
4. *si $M, N \in \Lambda$, alors $(MN) \in \Lambda$*

Nous adoptons les conventions usuelles de parenthésage pour les λ-termes ; par exemple, le terme $\lambda x(\lambda y.M)$ s'écrira simplement $\lambda x\lambda y.M$; un terme $((M_1 M_2)M_3)$ s'écrira $(M_1 M_2 M_3)$.

Notation 2.1.1. Par défaut, nous noterons les variables par des lettres minuscules $x, y, z, f, g, \ldots$, les constantes par des lettres minuscules grasses $\mathbf{c}, \mathbf{d}, \mathbf{e}, \ldots$ et les λ-termes par des lettres majuscules

$M, N, O, P, \ldots$. Une séquence de variables $x_1 x_2 \ldots x_n$ pourra être notée $\overline{x}$. Enfin, nous utiliserons la notation usuelle permettant d'omettre la répétition de λ : par exemple, un terme $\lambda_1 \lambda x_2 \lambda x_3 . M$ sera simplement écrit $\lambda x_1 x_2 x_3 . M$.

Définition 2.1.1.2. *Soit Λ, l'ensemble des λ-termes. L'ensemble des* contextes $\Lambda_{[]}$ *est défini inductivement de la manière suivante :*

1. $[] \in \Lambda_{[]}$
2. *Si* $C \in \Lambda_{[]}$ *alors* $\lambda x.C \in \Lambda_{[]}$
3. *Si* $M \in \Lambda$ *et* $C \in \Lambda_{[]}$, *alors* $MC \in \Lambda_{[]}$ *et* $CM \in \Lambda_{[]}$

Nous noterons les contextes de termes par $C[], C_1[], C_2[] \ldots$ Un terme N peut être inséré dans un contexte $C[]$, formant un nouveau terme noté $C[N]$ selon la définition suivante :

1. si $C[] = []$ alors $C[N] = N$.
2. si $C[] = \lambda x.C'[]$ alors $C[N] = \lambda x.C'[N]$.
3. si $C[] = MC'[]$ alors $C[N] = MC'[N]$.
4. si $C[] = C'[]M$ alors $C[N] = C'[N]M$

Définition 2.1.1.3. *Étant donnés deux termes M et N, si il existe un contexte $C[]$ tel que $M = C[N]$, nous dirons alors que N* est un sous-terme de M.

Il est possible qu'un sous-terme N ait plusieurs occurrences dans un terme M. Tel est le cas si $M = C_1[N] = C_2[N]$ et $C_1[] \neq C_2[]$. Nous désignerons donc par la paire $(C[], N)$, l'*occurrence* de N dans le terme M telle que $M = C[N]$.

Définition 2.1.1.4. *Soit un λ-terme M. L'ensemble $FV(M)$ des* variables libres *de M est défini inductivement par :*

1. *Si* $M = \mathbf{c} \in \mathcal{C}$, *alors* $FV(M) = \emptyset$
2. *Si* $M = x \in \mathcal{V}$, *alors* $FV(M) = \{x\}$
3. *Si* $M = \lambda x.N$, *alors* $FV(M) = FV(N) - \{x\}$
4. *Si* $M = M_1 M_2$, *alors* $FV(M) = FV(M_1) \cup FV(M_2)$

Définition 2.1.1.5. *Un terme ne contenant pas de variables libres est appelé un* terme clos *; un terme ne contenant pas de constantes est appelé un* terme pur.

Remarque 2.1.1.1. L'utilisation que nous ferons des constantes dans ce document s'apparente à celui de variables libres d'un terme. Par conséquent, nous étudierons régulièrement les propriétés de termes purs, et réutiliserons ensuite ces propriétés sur des termes contenant des constantes.

La syntaxe même d'un λ-terme n'impose pas de contraintes sur le nommage des variables. Afin de travailler sur les variables d'un terme M, nous parlerons donc d'*occurrences de variables* $(C[], x)$ dans M.

Définition 2.1.1.6. *Étant donnés un terme M et une occurrence de variable $(C[], x)$ dans M, cette occurrence est dite* liée dans *M si il existe $C'[], \lambda x.C''[]$ tels que $C[] = C'[\lambda x.C''[]]$. Autrement, $(C[], x)$ est dite libre dans M.*

Dans la section suivante, nous verrons que certaines règles de réécriture sur les λ-termes permettent de renommer les occurrences liées des variables d'un terme. En particulier, il est possible d'assigner un nom distinct à chaque variable d'un terme. Si le terme M vérifie cette condition, nous désignerons par $V(M)$ l'ensemble des variables de M.

2.1.2 Règles de réécriture : α-conversion, β-réduction et η-conversion

Dans cette section, nous introduisons les opérations pouvant être réalisées dans le λ-calcul. Ces opérations sont présentées sous forme de règles de réécriture. Afin de les définir, nous introduisons la notion de *substitution de variables* :

Définition 2.1.2.1. *Une substitution de variables est une fonction de $\mathcal{V}$ dans Λ.*

Notation 2.1.2. Une substitution de variables sera notée $\sigma = [x_1 := N_1, \ldots, x_m := N_m]$ lorsque $\sigma(x_i) = N_i$ pour tout $i \in \{1, \ldots, n\}$ et $\sigma(x) = x$ pour tout $x \in \mathcal{V} - \{x_1, \ldots, x_n\}$. De plus, étant données une variable x et une telle substitution de variables, nous noterons par $\sigma - \{x\}$ la substitution de variables telle que $\sigma - \{x\}(y) = \sigma(y)$ pour tout $y \in \mathcal{V} - \{x\}$ et $\sigma - \{x\}(x) = x$.

L'application d'une substitution de variables $\sigma = [x_1 := N_1, \ldots, x_m := N_m]$ sur M (notée $M \cdot \sigma$) remplace les occurrences libres de $x_1, \ldots, x_m$ dans M par $N_1, \ldots, N_m$ respectivement ; de manière formelle :

- $y \cdot \sigma = \sigma(y)$ pour toute variable $y \in \mathcal{V}$
- $\mathbf{c} \cdot \sigma = \mathbf{c}$ pour toute constante $\mathbf{c} \in C$
- $(\lambda x.M) \cdot \sigma = \begin{cases} (\lambda x.M \cdot \sigma - \{x\}) & \text{si } x \notin FV(N_1) \cup \ldots \cup FV(N_m) \\ (\lambda y.M \cdot [x := y] \cdot \sigma) & \text{où } y \notin FV(M) \cup FV(N_1) \cup \ldots \cup FV(N_m), \text{ sinon} \end{cases}$
- $(M_1 M_2) \cdot \sigma = (M_1 \cdot \sigma M_2 \cdot \sigma)$.

Dans un premier temps, nous nous intéressons à une restriction des substitutions de variables de telle sorte que le codomaine de σ soit l'ensemble des variables $\mathcal{V}$.

Définition 2.1.2.2. ***(α-conversion)*** *Soit un terme $M = C[\lambda x.N]$ et une variable $y \notin FV(N)$. Nous dirons que M se α-*contracte *en $M' = C[\lambda y.(N \cdot [x := y])]$ (noté $M \to_\alpha M'$), et nous appellerons α-*contraction *le fait de remplacer $(C[], \lambda x.N)$ par $\lambda y.(N \cdot [x := y])$ dans M.*

*Si un terme M est α-contractable en un terme M' en $n \in \mathbb{N}$ étapes d'α-contractions, nous dirons que M et M' sont α-*convertibles*.*

On peut vérifier que l'α-contraction est une relation symétrique ; l'α-conversion étant la clôture réflexive et transitive de l'α-contraction, il s'agit d'une relation d'équivalence sur les termes du λ-calcul. Par la suite, nous dirons que pour un terme M α-convertible en un terme M', les termes M et M' sont *α-équivalents*.

Grâce à l'α-conversion, il est donc possible de renommer les occurrences liées de variables dans un terme, ce qui permet, entre autre, de donner un nom distinct à chaque variable d'un terme. Ainsi, le terme $x(\lambda x.x)$ écrit précédemment est *α-équivalent* au terme $x(\lambda x'.x')$. Il faut cependant remarquer que la définition impose d'éviter certains effets dits de *capture de variables*. Ainsi, le terme $M' = x(\lambda x'.x)$ n'est pas α-équivalent au terme $M = x(\lambda x.x)$; en effet, l'occurrence de variable $(x(\lambda x.[]), x)$ est une occurrence liée dans M, ce qui n'est pas le cas de $([](\lambda x.x), x)$; ces deux occurrences désignent donc deux variables différentes dans M. Dans M', les occurrences de variable $([](\lambda x'.x), x)$ et $(x(\lambda x'.[]), x)$ sont, par contre, deux occurrences libres de la même variable x.

Par la suite, nous adopterons régulièrement la convention de nommage de Barendregt, où chaque variable d'un terme porte un nom distinct, afin de simplifier la lecture des termes.

Définition 2.1.2.3. ***(β-contraction)*** *Nous appelons un terme de la forme $(\lambda x.M_1)M_2$ un β-*redex*.*

*Un terme $M = C[(\lambda x.M_1)M_2]$ se β-contracte en $M' = C[M_1[x := M_2]]$. Une telle étape de réécriture (notée $M \to_\beta M'$) est appelée β-*contraction*.*

Si un terme M se β-contracte en un terme M' en $n \in \mathbb{N}$ étapes de β-contraction, nous écrirons $M \to_\beta^n M'$, et dirons que M se *β-réduit* en M'. La β-réduction se définit comme la clôture transitive et

réflexive de la β-contraction et nous notons par $M \to_\beta^* M'$ cette relation entre M et M'. Par ailleurs, la relation inverse est appelée β-expansion : un terme M se β-réduit en un terme M', ssi M' se β-étend en M.

Étant donné un terme M, nous appellerons *séquence de β-contractions* toute séquence $M \to_\beta M_1 \ldots \to_\beta M_n \to_\beta \ldots$. Une séquence de β-contractions $M \to_\beta M_1 \to_\beta \ldots \to_\beta M_n$ est dite *normalisante* si M_n ne contient pas de β-redex. Le terme M_n est alors dit *sous forme normale*.

Remarque 2.1.2.1. Un λ-terme M est sous forme normale si il existe $p \in \mathbb{N}$ tels que $M = \lambda\overline{x}.NM_1 \ldots M_p$ où $N \in \mathcal{V} \cup C$ et si pour tout $i \in \{1, \ldots, p\}$, les termes M_i sont sous forme normale. En effet, un tel terme ne peut contenir d'occurrence de β-redex.

La notion de β-réduction permet de modéliser l'exécution de programmes. En effet, si nous considérons le terme $\lambda x.M_1$ comme un programme ayant un paramètre d'entrée x, et un terme M_2 comme étant un autre programme, le terme $(\lambda x.M_1)M_2$ simule l'appel du programme $\lambda x.M_1$ au programme M_2 ; la β-contraction de ce β-redex en $M_1[x := M_2]$ représente le remplacement du paramètre x dans le programme M_1 par le programme M_2.

Le théorème suivant (également appelé propriété du losange) montre la *confluence* de cette règle de réécriture :

Théorème 2.1.2.1. ***(Propriété de Church-Rosser)*** *Soient M, M_1, M_2 des λ-termes tels que $M \to_\beta^* M_1$ et $M \to_\beta^* M_2$; alors il existe un terme N vérifiant $M_1 \to_\beta^* N$ et $M_2 \to_\beta^* N$.*

Nous remarquerons que si pour un terme M, il existe une séquence terminante de β-contractions de M, alors, d'après la propriété de Church-Rosser, celle-ci se finit en un unique terme N. Dans ce cas, N est appelé *la forme (β-)normale de M* et sera noté $|M|_\beta$. Remarquons qu'il est néanmoins possible que le processus de β-réduction ne se termine jamais, ce qui est par exemple le cas du terme $M = (\lambda x.xx)(\lambda x.xx)$ qui ne peut se β-réduire qu'en lui-même.

Définition 2.1.2.4. ***(η-contraction)*** *Un terme $\lambda x.Mx$ où $x \notin FV(M)$ est appelé un η-*redex.

*Soient $M = C[\lambda x.Mx]$ tel que $x \notin FV(M)$. Nous dirons que M se η-*contracte *en $M' = C[M]$. Nous appelons η-*contraction *une telle étape de réécriture (noté $M \to_\eta M'$).*

Les notations $\to_\eta^n$ et $\to_\eta^*$ sont utilisées avec la même signification que pour la β-contraction. Nous parlerons également d'*η-réduction* afin de désigner la relation $\to_\eta^*$, et d'η-expansion pour la relation inverse ; de même, nous parlerons de séquences d'η-contractions, de séquences d'η-contractions normalisantes et de termes η-normaux.

Nous verrons dans la section suivante que la notion d'η-expansion offre de nombreux avantages dans le cas des λ-termes simplement typés.

Théorème 2.1.2.2. ***(Propriété de Church-Rosser)*** *Soient M, M_1, M_2 des λ-termes tels que $M \to_\eta^* M_1$ et $M \to_\eta^* M_2$; alors il existe un terme N vérifiant $M_1 \to_\eta^* N$ et $M_2 \to_\eta^* N$.*

De plus, pour un terme fini M, toute séquence d'η-contractions de M est terminante. Nous noterons donc par $|M|_\eta$ l'unique terme η-normal tel que $M \to_\eta^* |M|_\eta$; ce terme sera appelé *la forme η-normale de M*.

Notation 2.1.3. Étant donnés deux λ-termes M_1et M_2, nous écrivons :

- $M_1 = M_2$ lorsque M_1 et M_2 sont α-équivalents ;
- $M_1 =_\beta M_2$ si il existe un λ-terme M tel que $M_1 \to_\beta^* M$ et $M_2 \to_\beta^* M$ (M_1 et M_2 seront dits *β-équivalents*) ;

- $M_1 =_\eta M_2$ si il existe un λ-terme M tel que $M_1 \to_\eta^* M$ et $M_2 \to_\eta^* M$ (auquel cas M_1 et M_2 sont dits η-équivalents).

Remarque 2.1.2.2. On remarquera que les relations $=_\beta$ et $=_\eta$ sont effectivement des relations d'équivalence, ce qui se démontre en utilisant la propriété de Church-Rosser dans les deux cas.

2.2 Le λ-calcul simplement typé

Nous venons d'introduire la syntaxe générale du λ-calcul. Nous nous intéressons à présent à un sous-ensemble de l'ensemble des λ-termes : l'ensemble des λ-termes simplement typés. Les éléments de ce sous-ensemble se caractérisent par le fait de pouvoir être associés à un ou plusieurs *typages* (selon le système de typage considéré). Nous présentons ce sous-ensemble sous ses deux formalisations les plus courantes.

2.2.1 Typage à la Curry

La premier système de typage de λ-termes que nous présentons considère des λ-termes, a priori non typés, et permet de leur associer des typages si ces derniers existent. Dans ce système, la syntaxe des termes et leurs typages sont mises en relation à travers la notion d'*arbres de typages*. Qui plus est, ce système laisse clairement apparaître une correspondance avec un formalisme logique (la logique minimale), correspondance que nous expliquerons à la fin de ce chapitre.

Définition 2.2.1.1. *Étant donné un ensemble énumérable $\mathcal{A}$ d'éléments appelés* types atomiques*, l'ensemble $\mathcal{T}(\mathcal{A})$ des* types simples sur $\mathcal{A}$ *est défini par induction :*

$$\mathcal{T}(\mathcal{A}) ::= \mathcal{A} \mid (\mathcal{T}(\mathcal{A}) \to \mathcal{T}(\mathcal{A}))$$

Nous adopterons la convention usuelle d'associativité à droite pour l'opérateur $\to$: un type $(\alpha_1 \to (\alpha_2 \to \alpha_3))$ sera donc noté $\alpha_1 \to \alpha_2 \to \alpha_3$.

Notation 2.2.1. Nous noterons les types atomiques (*c.a.d* ceux qui appartiennent à $\mathcal{A}$) par les lettres minuscules de l'alphabet latin, éventuellement indexées par des entiers : $a, b, c, a_1, \ldots$ Les éléments de $\mathcal{T}(\mathcal{A})$ seront quant à eux notés par les lettres minuscules de l'alphabet grec (excepté λ et σ), éventuellement indexées par des entiers : $\alpha, \beta, \alpha_1, \ldots$

Définition 2.2.1.2. *Étant donné un type α appartenant à $\mathcal{T}(\mathcal{A})$,* l'ordre *de α est inductivement défini par :*

- $\mathtt{ord}(\alpha) = 1$ *si* $\alpha \in \mathcal{A}$
- *si* $\alpha = \alpha_1 \to \alpha_2$ *alors* $\mathtt{ord}(\alpha) = max(1 + \mathtt{ord}(\alpha_1), \mathtt{ord}(\alpha_2))$.

Pour un type α tel que $\mathtt{ord}(\alpha) > 1$, nous dirons que α est un *type d'ordre supérieur*.

Définition 2.2.1.3. *Soit $\mathcal{A}$ un ensemble de types atomiques. Une fonction partielle $\Gamma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{A})$ (resp. $\Delta : C \mapsto \mathcal{T}(A)$) sera appelée une fonction d'assignation de types sur les variables (resp. sur les constantes).*

Un environnement de typage *$\langle \Gamma; \Delta \rangle$ est un couple de fonctions d'assignation de types sur les variables, Γ, et sur les constantes, Δ.*

Notation 2.2.2. Nous noterons par $\Gamma, \Gamma', \Gamma_1, \ldots$ les fonctions d'assignation de types sur les variables et $\Delta, \Delta', \Delta_1, \ldots$ les fonctions d'assignation de types sur les constantes.

Notation 2.2.3. Étant données Γ, Γ' des fonctions d'assignation de types sur les variables :

1. pour $\mathrm{Dom}(\Gamma) = \{x_1, \ldots, x_n\}$ le domaine de Γ, nous noterons Γ par $x_1 : \Gamma(x_1), \ldots, x_n : \Gamma(x_n)$ afin de désigner le domaine de Γ et le type associé à chacun des éléments de ce domaine.
2. si $\mathrm{Dom}(\Gamma) = \emptyset$, nous noterons Γ par _.
3. si $\mathrm{Dom}(\Gamma) \cap \mathrm{Dom}(\Gamma') = \emptyset$, nous noterons par Γ, Γ', la fonction d'assignation de types définie par $\mathrm{Dom}(\Gamma, \Gamma') = \mathrm{Dom}(\Gamma) \cup \mathrm{Dom}(\Gamma')$, et telle que pour tout $x \in \mathrm{Dom}(\Gamma, \Gamma')$, $(\Gamma, \Gamma')(x) = \Gamma(x)$ si $x \in \mathrm{Dom}(\Gamma)$ et $(\Gamma, \Gamma')(x) = \Gamma'(x)$ si $x \in \mathrm{Dom}(\Gamma')$.
4. nous écrirons $\Gamma \subseteq \Gamma'$ et $\Gamma = \Gamma'$ pour les relations usuelles d'inclusion et d'égalité entre fonctions.

Ces mêmes notations seront également utilisées pour les fonctions d'assignation de types sur les constantes. Par ailleurs, nous abrégerons la notion d'un séquent de typage de la forme $\langle \_ ; \Gamma \rangle \vdash \alpha$, en $\Gamma \vdash \alpha$.

Définition 2.2.1.4. *Étant donnés $\langle \mathbf{\Delta}; \Gamma \rangle$ un environnement de typage, un λ-terme M, et $\alpha \in \mathcal{T}(\mathcal{A})$, nous appellerons :*

- typage*, une structure de la forme $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$.*
- séquent de typage *une structure $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$.*

Axiome

$$\frac{\Gamma(x) = \alpha}{\langle \mathbf{\Delta}; \Gamma \rangle \vdash x : \alpha}\ \text{(Identité - Var)} \qquad \frac{\mathbf{\Delta}(\mathbf{c}) = \alpha}{\langle \mathbf{\Delta}; \Gamma \rangle \vdash \mathbf{c} : \alpha}\ \text{(Identité - Cons)}$$

Application et Abstraction

$$\frac{\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha \to \beta \quad \langle \mathbf{\Delta}; \Gamma \rangle \vdash N : \alpha}{\langle \mathbf{\Delta}; \Gamma \rangle \vdash MN : \beta}\ \text{(Application)}$$

$$\frac{\langle \mathbf{\Delta}; \Gamma, x : \alpha \rangle \vdash M : \beta}{\langle \mathbf{\Delta}; \Gamma \rangle \vdash \lambda x.M : \alpha \to \beta}\ \text{(Abstraction)}$$

FIG. 2.1 – Système de typage à la Curry

Définition 2.2.1.5. *Étant donné un λ-terme M, si il existe un séquent de typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$ dérivable dans le système de la Figure 2.1, alors nous dirons que $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ est un* typage de M *(alternativement, que M est un* habitant *de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$).*

De plus, M sera alors dit simplement typable.

Une dérivation d'un séquent de typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$ sera appelée un *arbre de typage*.

Remarque 2.2.1.1. Étant donnés un terme M et un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M, nous pouvons remarquer que, à toute occurrence d'une variable (*resp.* constante) dans M est assigné le même type dans l'arbre de typage de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$.

Exemple 2.2.1.1. L'arbre de typage de

$$\Gamma \vdash \lambda f.f(\lambda x.gx) : ((a \to b) \to c) \to c$$

où $\Gamma = g : a \to b, y : d$, est représenté en Figure 2.2 :

$$\dfrac{\dfrac{\dfrac{}{\Gamma, f : (a \to b) \to c \vdash f : (a \to b) \to c} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, f : (a \to b) \to c \vdash g : a \to b} \qquad \dfrac{}{\Gamma, x : a, f : (a \to b) \to c \vdash x : a}}{\Gamma, x : a, f : (a \to b) \to c \to b \vdash gx : b}}{\Gamma, f : (a \to b) \to c \vdash \lambda x.gx : a \to b}}{\Gamma, f : (a \to b) \to c \vdash f(\lambda x.gx) : c}}{\Gamma \vdash \lambda f.f(\lambda x.gx) : ((a \to b) \to c) \to c}$$

FIG. 2.2 – Exemple d'arbre de typage

Comme nous pouvons le remarquer, à un arbre de typage correspond un terme simplement typable. Ainsi, dans le système de typage à la Curry, l'information de typage n'est pas intégrée dans la syntaxe même du terme, mais est une information qui se déduit de la syntaxe du terme. En particulier, un terme peut être typé par plusieurs typages. Par exemple, le terme $M = \lambda x.x$ peut être typé par tous les typages de la forme $\alpha \to \alpha$ où α appartient à $\mathcal{T}(\mathcal{A})$.

Nous définissons, à présent, les opérations permettant de générer, à partir d'un typage d'un terme M, d'autres typages de M.

Définition 2.2.1.6. *Étant donné un environnement de typage $\langle \mathbf{\Delta}; \Gamma \rangle$, nous appelons $\langle \mathbf{\Delta}'; \Gamma' \rangle$ un* affaiblissement de l'environnement $\langle \mathbf{\Delta}; \Gamma \rangle$ *si et seulement si* $\mathrm{Dom}(\Gamma) \subseteq \mathrm{Dom}(\Gamma')$ *et* $\mathrm{Dom}(\mathbf{\Delta}) \subseteq \mathrm{Dom}(\mathbf{\Delta}')$.

Lemme 2.2.1.1. *Soient un terme M et un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M, alors :*

- $FV(M) \subseteq \mathrm{Dom}(\Gamma)$ *et* $Cst(M) \subseteq \mathrm{Dom}(\mathbf{\Delta})$.
- *Pour tout affaiblissement $\langle \mathbf{\Delta}'; \Gamma' \rangle$ de $\langle \mathbf{\Delta}; \Gamma \rangle$, le typage $\langle \mathbf{\Delta}'; \Gamma' \rangle \vdash \alpha$ est habité par M.*

D'après ce lemme, la notion de typage d'un terme est donc stable par l'opération d'affaiblissement de l'environnement.

Définition 2.2.1.7. *Soit $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ un typage pour un terme M. Nous dirons que $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ est sous* forme canonique *pour M si et seulement si* $\mathrm{Dom}(\Gamma) = FV(M)$ *et* $\mathrm{Dom}(\mathbf{\Delta}) = Cst(M)$.

Exemple 2.2.1.2. Ainsi, pour le terme $M = \lambda f.f(\lambda x.gx)$, le typage $g : a \to b \vdash ((a \to b) \to c) \to c$ est un typage canonique de M, ce qui n'est pas le cas pour le typage $g : a \to b, y : d \vdash ((a \to b) \to c) \to c$.

D'après le Lemme 2.2.1.1, pour tout typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ d'un terme M, il existe un typage canonique de M générant $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ par affaiblissement de l'environnement.

Définition 2.2.1.8. *(Substitution, ré-étiquetage et renommage de types)*

1. *Une* substitution *de types σ est une fonction de $\mathcal{A}$ dans $\mathcal{T}(\mathcal{A})$.*
2. *Un* ré-étiquetage *est une substitution de types tel que pour $E = \mathrm{Dom}(\sigma)$, $\mathrm{Im}(\sigma) \subseteq E$.*
3. *Un* renommage *est un ré-étiquetage bijectif.*

Une substitution de types σ induit un endomorphisme, noté également σ, de $\mathcal{T}(\mathcal{A})$ dans $\mathcal{T}(\mathcal{A})$: ainsi, pour un type $\alpha \in \mathcal{T}(\mathcal{A})$, l'application de l'endomorphisme σ sur α (notée $\alpha \cdot \sigma$) est définie inductivement sur α : si $\alpha = a \in \mathcal{A}$, alors $\alpha \cdot \sigma = \sigma(a)$; si $\alpha = \alpha_1 \to \alpha_2$, alors $\alpha \cdot \sigma = (\alpha_1 \cdot \sigma) \to (\alpha_2 \cdot \sigma)$. Par la suite, nous confondrons une substitution de types avec l'endomorphisme induit.

L'application d'une substitution de types σ sur une fonction de typage Γ sera notée $(\Gamma \cdot \sigma)$ de telle sorte que $\mathrm{Dom}(\Gamma \cdot \sigma) = \mathrm{Dom}(\Gamma)$ et pour tout $x \in \mathrm{Dom}(\Gamma \cdot \sigma)$, $(\Gamma \cdot \sigma)(x) = \Gamma(x) \cdot \sigma$; de même, pour un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, nous écrirons $\langle \mathbf{\Delta}; \Gamma \rangle \cdot \sigma \vdash \alpha \cdot \sigma = \langle \mathbf{\Delta} \cdot \sigma; \Gamma \cdot \sigma \rangle \vdash \alpha \cdot \alpha$ l'application de σ à ce typage.

Lemme 2.2.1.2. *Étant donnés un terme simplement typable M, un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de ce terme, et une substitution de types σ, alors $\langle \mathbf{\Delta}; \Gamma \rangle \cdot \sigma \vdash \alpha \cdot \sigma$ est un typage pour M.*

Nous avons donc deux opérations transformant un typage d'un terme en un nouveau typage de ce terme : l'affaiblissement d'environnements et la substitution de types. Grâce à ces deux opérations, nous définissons des typages particuliers associés à un terme M :

Définition 2.2.1.9. *Soit un terme M. Un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M est appelé* typage principal *(ou* typage le plus général*) de M si tout autre typage $\langle \mathbf{\Delta}'; \Gamma' \rangle \vdash \alpha'$ de M est généré par affaiblissements d'environnement et substitutions sur $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$.*

(ou bien, de manière équivalente, il existe une substitution de types σ telle que $\Gamma \cdot \sigma \subseteq \Gamma'$, $\mathbf{\Delta} \cdot \sigma \subseteq \mathbf{\Delta}'$ et $\alpha \cdot \sigma = \alpha'$)

Remarquons que, étant donnés un terme M et un typage principal de ce terme $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, alors pour tout renommage σ, $\langle \mathbf{\Delta}; \Gamma \rangle \cdot \sigma \vdash \alpha \cdot \sigma$ est un typage principal de M. Il est, plus exactement, possible de quotienter l'ensemble des typages d'un terme par renommage de manière à obtenir un ensemble bien ordonné de classe d'équivalences de typages pour chaque terme M (l'ordre bien fondé étant induit par les opérations de substitutions et d'affaiblissements) ; nous nous permettrons donc de parler du typage principal de M, uniquement défini à renommage près.

Propriété 2.2.1.1. *Pour tout λ-terme simplement typable M, il existe un typage principal $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, unique à renommage près.*

Exemple 2.2.1.3. Soit le terme $(\lambda xy.f(\mathbf{c}(fx)))(g\mathbf{e})$. Son typage principal est $\langle \mathbf{c} : b \to a, \mathbf{e} : c; f : a \to b, g : c \to a \rangle \vdash a \to d \to a$.

Plus de détails sur le notion de typages principaux d'un terme sont donnés dans [Wells, 2002].

2.2.2 Typage à la Church

Une système alternatif du λ-calcul simplement typé est régulièrement rencontré. Dans ce système, que l'on doit à Alonzo Church [Church, 1941], les variables et constantes sont typées *a priori*.

Notation 2.2.4. À une variable x (*resp.* une constante $\mathbf{c}$), nous associons un type α. Cette relation est notée x^α (*resp.* $\mathbf{c}^\alpha$).

Définition 2.2.2.1. *Étant donné un type α de $\mathcal{T}(\mathcal{A})$, nous définissons l'ensemble Λ_α des* termes simplement typés par le type α *de manière inductive sur les termes :*

1. *x^α et $\mathbf{c}^\alpha$ appartiennent à Λ_α.*
2. *si $\alpha = \alpha_1 \to \alpha_2$ et que $M \in \Lambda_{\alpha_2}$, alors le terme $\lambda x^{\alpha_1}.M$ appartient à Λ_α*
3. *si $M_1 \in \Lambda_{\beta \to \alpha}$ et $M_2 \in \Lambda_\beta$, alors le terme $M_1 M_2$ appartient à Λ_α.*

Étant donné un type $\alpha \in \mathcal{T}(\mathcal{A})$, tout terme appartenant à Λ_α sera dit *simplement typé*. De plus, nous appellerons α *le type de M*.

L'ensemble des termes simplement typés est donc défini par $\Lambda_{st} = \{\Lambda_\alpha \mid \alpha \in \mathcal{T}(\mathcal{A})\}$.

Remarque 2.2.2.1. Pour tout terme simplement typé, il existe un unique type associé, la notion de typage ne sera pas utilisée dans le système de types à la Church. Par exemple, pour un type α donné, le terme $\lambda x^\alpha.x^\alpha$ est typé par le seul type $\alpha \to \alpha$.

$x^\alpha \in \Lambda_\alpha$	$\dfrac{}{\langle \mathbf{\Delta}; \Gamma_1, x:\alpha, \Gamma_2\rangle \vdash x:\alpha}$ (Identité - Var)
$\mathbf{c}^\alpha \in \Lambda_\alpha$	$\dfrac{}{\langle \mathbf{\Delta}_1, \mathbf{c}:\alpha, \mathbf{\Delta}_2; \Gamma\rangle \vdash \mathbf{c}:\alpha}$ (Identité - Con)
$\lambda x^\alpha.M \in \Lambda_{\alpha\to\beta}$ if $M \in \Lambda_\beta$	$\dfrac{\langle \mathbf{\Delta}; \Gamma_1, x:\alpha, \Gamma_2\rangle \vdash M:\beta}{\langle \mathbf{\Delta}; \Gamma_1, \Gamma_2\rangle \vdash \lambda x.M : \alpha\to\beta}$ (Abstraction)
$M_1M_2 \in \Lambda_\beta$ if $M_1 \in \Lambda_{\alpha\to\beta}$ or $M_2 \in \Lambda_\alpha$	$\dfrac{\langle \mathbf{\Delta}; \Gamma\rangle \vdash M_1 : \alpha\to\beta \quad \langle \mathbf{\Delta}; \Gamma\rangle \vdash M_2 : \alpha}{\langle \mathbf{\Delta}; \Gamma\rangle \vdash M_1M_2 : \beta}$ (Application)

FIG. 2.3 – Correspondance termes à la Church, dérivation de typages à la Curry

Nous parlerons du typage à la Church d'un terme $M \in \Lambda_\alpha$ comme de la connaissance du type α et des relations variables-types $x^\gamma \in \Lambda_\gamma$ et constantes-types $\mathbf{c}^\gamma \in \Lambda_\gamma$ pour chaque variable et chaque constante de M.

Remarque 2.2.2.2. Un terme appartenant à Λ_{st} sera dit simplement typé, contrastant ainsi avec les termes simplement typables que nous avons définis dans le système de typage à la Curry. La différence résidant dans le fait que nous associons un unique *type* à un terme simplement typé, celui donné par la syntaxe du terme, alors que dans le cas d'un terme simplement typable, nous ne connaissons que l'existence d'un *typage* pour ce terme.

Bien que certaines différences existent entre les deux systèmes de typage que nous avons présentés, ils permettent de typer exactement les mêmes termes. Nous donnons ici une preuve informelle du fait que l'ensemble des termes simplement typables est exactement l'ensemble des termes simplement typés.

Pour ce faire, considérons la table de la Figure 2.3. À une règle de construction d'un terme M typé à la Church correspond donc une règle de dérivation d'un typage de M dans le système de typage à la Curry. Ainsi, en considérant un terme M typé à la Church, nous pouvons construire un ensemble de dérivations de typages de M, et inversement, à une dérivation d'un typage de M, nous pouvons associer son typage dans le système de typage à la Church.

Définition 2.2.2.2. *Étant donné un terme $M \in \Lambda_\alpha$ typé à la Church, nous parlerons du typage $\langle \Gamma; \mathbf{\Delta}\rangle \vdash \alpha$ comme du* typage canonique associé *M dans Λ_α, si ce typage est déterminé par la correspondance définie en Figure 2.3 et que* $\mathrm{Dom}(\Gamma) = FV(M)$ *et* $\mathrm{Dom}(\mathbf{\Delta}) = Cst(M)$.

Nous nous en référons à [Barendregt, 1993, Hindley, 1997, Sørensen and Urzyczyn, 2006] pour plus de détails.

Remarque 2.2.2.3. Afin de voir plus clairement la différence entre ces deux systèmes de typage, voyons les λ-termes comme des programmes ; dans la système de typage à la Church, le programmeur associe des types aux paramètres et aux résultats des fonctions. A contrario, dans le système de typage à la Curry, le programme n'est pas typé, et l'on a besoin d'un algorithme d'inférence de types afin de connaître les types possibles des paramètres et résultats.

Nous avons montré que les deux systèmes permettent de typer exactement les mêmes termes. Nous nous permettrons donc d'utiliser l'un ou l'autre de ces systèmes par la suite. De plus, les propriétés que nous décrivons dans la suite de ce chapitre s'appliqueront à ces deux systèmes.

Un des avantages du système de typage à la Church sur le système de typage à la Curry est le fait de pouvoir directement parler du type associé à un sous-terme d'un terme M. En effet, un sous-terme est lui-même un terme typé à la Church, et il lui est donc associé un unique type. Dans le système de typage à la Curry, nous avons vu que plusieurs typages sont associés à un même λ-terme. Il existe donc potentiellement plusieurs typages associés à un sous-terme d'un terme. Néanmoins, nous avons vu qu'à une dérivation d'un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M dans le système de typage à la Curry, nous pouvons associer un typage de M dans le système de typage à la Church ; à un sous-terme N de M est alors assigné un unique typage à la Church. En particulier, il existe un type β tel que $N \in \Lambda_\beta$. Nous dirons alors que *β est le type de N dans la dérivation de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$.*

2.3 Propriétés du λ-calcul simplement typé

Après avoir introduit le λ-calcul simplement typé, nous examinons certaines propriétés importantes de celui-ci.

2.3.1 Termes η-longs pour un typage

Nous définissons, à présent, la notion de termes simplement typés η-longs. Cette notion est également applicable aux termes simplement typables ; nous nous limiterons à rappeler l'analogie entre les deux systèmes de typage en guise d'explications.

Définition 2.3.1.1. *Soit un terme $M \in \Lambda_\alpha$, pour $\alpha \in \mathcal{T}(\mathcal{A})$. Nous dirons que M est* η-long *si, pour toute occurrence $(C[], N)$ de sous-terme de M, telle que $N \in \Lambda_{\alpha_1 \to \alpha_2}$, alors, soit $C[] = C'[[]N']$, soit $N = \lambda x^{\alpha_1}.N'$.*

De plus, étant donnés deux termes $M, M' \in \Lambda_\alpha$ tels que $M =_\eta M'$ et M' est η-long, nous dirons que M' est une forme η-longue de M.

Remarque 2.3.1.1. D'après l'analogie entre typage à la Church et typages à la Curry, cette notion s'applique également à un typage à la Curry d'un terme M. Plus précisément, pour $M \in \Lambda_\alpha$ η-long, nous dirons que *M est η-long relativement à un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$*, si le typage à la Church de M associé à la dérivation de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$ à affaiblissement près.

De plus, étant donné un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M, nous dirons que le terme M' est une *forme η-longue de M pour $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$* si et seulement si $M =_\eta M'$ et M' est η-long pour ce typage.

Exemple 2.3.1.1. Le terme $M = f^{((o\to o)\to o)\to o\to o} g^{(o\to o)\to o}$ n'est pas η-long. D'après la définition, il est nécessaire de faire apparaître les arguments de f et de g, tout en conservant le typage induit par la syntaxe, ce qui est réalisé en abstrayant les variables ainsi introduites, comme dans le terme $\lambda x_1^o.f^{((o\to o)\to o)\to o\to o}(\lambda x_2^{o\to o}.g^{(o\to o)\to o}(\lambda x_3^o.x_2^{o\to o} x_3^o))x_1^o$, qui est η-équivalent à M.

Remarque 2.3.1.2. Les termes η-longs relativement à un certain typage présentent donc l'avantage d'avoir une syntaxe fortement contrainte par ce dernier. Ceci permet de faire apparaître dans la syntaxe du terme, des informations sur ce typage. Ces informations ne sont pas nécessaires lorsque l'on adopte les typages à la Church, puisque déjà incluses dans la syntaxe des termes, mais ne sont pas apparentes dans le cas des termes typés à la Curry.

Les termes η-longs présentent, de plus, un intérêt particulier, partiellement exprimé à travers la propriété suivante :

Propriété 2.3.1.1. *Soit un terme $M \in \Lambda_\alpha$ η-long ; alors tout terme $M' \in \Lambda_\alpha$ tel que $M' \to_\eta^* M$ vérifie $M' \to_\beta^* M$ (c.a.d. un terme simplement typé à une unique forme normale η-longue) [Huet, 1976].*

Exemple 2.3.1.2. Par exemple, pour le terme $M = \lambda f^{o\to o} y^o . f^{o\to o} y^o$ qui est η-long, le terme $M' = \lambda f^{o\to o} x^o . (\lambda y^o . f^{o\to o} y^o) x^o$ se η-réduit en M. Par ailleurs, M' se β-réduit également en :

$$\lambda f^{o\to o} x^o . (f^{o\to o} y^o)[y^o := x^o] = \lambda f^{o\to o} x^o . f^{o\to o} x^o = M$$

Qui plus est, deux termes M_1 et M_2 vérifient $M_1 =_{\beta\eta} M_2$ si $|M|_1 =_\eta |M|_2$, où $|M|$ est la forme β-normale de M, lorsque celle-ci existe. Ainsi, nous travaillerons, par la suite, régulièrement sur la forme normale η-longue d'un terme pour cette raison.

2.3.2 Préservation de typage et β-réduction

Le lien entre typages et λ-termes nous permet d'obtenir certaines propriétés importantes sur les termes simplement typés. Cependant, nous avons vu qu'un même terme peut-être réécrit sous plusieurs formes, à travers les règles d'α-conversion, de β-réduction et d'η-conversion. Il est donc nécessaire de connaître l'effet de ces réécritures sur les typages (ou le type) d'un terme. La préservation de typage par α-équivalence étant claire, nous nous intéressons à la β-réduction et à l'η-réduction.

Propriété 2.3.2.1. *Soient un terme M simplement typé, et $\langle \mathbf{\Delta}; \Gamma\rangle \vdash \alpha$ un typage de M ; alors $\langle \mathbf{\Delta}; \Gamma\rangle \vdash \alpha$ est un typage de tout terme M' tel que $M \to_\eta^* M'$.*

Par contre, un typage n'est pas forcément préservé par η-expansion ; ainsi, étant donnés deux termes M et M' typables à la Curry tels que $M \to_\eta M'$, il se peut qu'un typage de M' ne type pas M. Par exemple, $\langle \mathbf{c} : a; \_\rangle \vdash a$ (où a est un type atomique) est un typage de $\mathbf{c}$ mais pas de $\lambda x.\mathbf{c}x$.

Remarque 2.3.2.1. Remarquons cependant que, dans le cas où les termes sont typés à la Church, tout typage de M' est un typage de M (sur le contre-exemple précédent, pour $M' = \mathbf{c}^a$, le terme $M = \lambda x^o . \mathbf{c}^a x^o$ est η-équivalent à M mais n'est pas un terme simplement typé).

La β-réduction a un comportement similaire à l'η-réduction pour ce qui est de la préservation des propriétés de typage :

Propriété 2.3.2.2. ***(Réduction du sujet)*** *Soient deux termes M et M' tels que $M \to_\beta^* M'$; alors tout typage de M est aussi un typage de M'. [Seldin, 1968]*

À nouveau, la β-expansion ne préserve pas forcément la typabilité d'un terme : si nous considérons deux termes M et M' tels que $M \to_\beta^* M'$, et un typage $\langle \mathbf{\Delta}; \Gamma\rangle \vdash \alpha$ de M', il est possible que ce typage ne soit pas un typage de M. En effet, si nous prenons $M = \lambda x.((\lambda y.\mathbf{c})(\lambda z.xz))$ et $M' = \lambda x.\mathbf{c}$, alors $\langle \mathbf{c} : a; \_\rangle \vdash b \to a$ est un typage de M', mais n'est pas un typage de M. Nous montrons, dans la section 2.3.4, que la préservation de typage pour certaines β-expansions est cependant vérifiée.

2.3.3 Normalisation

Une des propriétés fondamentales du λ-calcul simplement typé concerne la normalisation forte de la β-réduction. En effet, cette propriété permet de s'assurer de la terminaison des règles de β-réduction. Nous avons vu que cette propriété n'est pas vérifiée dans le cas le plus général à travers le contre-exemple $(\lambda x.xx)(\lambda x.xx)$, puisqu'il faudrait pouvoir assigner un type de la forme $\alpha \to \beta$ à l'occurrence $(\lambda x.[]x, x)$ de x et le type α à l'occurrence $(\lambda x.x[], x)$, ce qui est impossible puisque, dans le système de typage simple, un même type est assigné à toutes les occurrences d'une même variable dans un terme.

Théorème 2.3.3.1. ***(Normalisation)*** *Pour tout terme simplement typé M, il existe une séquence* normalisante *de β-contractions c.a.d. une séquence de la forme $M \to_\beta M_1 \ldots \to_\beta M_n$ telle que M_n est sous forme normale.*

Les propriétés de la β-réduction dans le cas des termes simplement typés sont en fait plus fortes, comme indiqué dans le théorème suivant :

Théorème 2.3.3.2. ***(Normalisation forte)*** *Étant donné un terme M simplement typé, il n'existe pas de séquence non-normalisante de β-contractions de M.*

Remarque 2.3.3.1. Comme nous l'avons vu à travers le terme $(\lambda x.xx)(\lambda x.xx)$, cette propriété n'est pas assurée dans le cas général du λ-calcul non-typé.

Grâce à la propriété de Church-Rosser, nous pouvons assurer que, étant donné un terme simplement typé M, il existe un unique terme M' tel que pour toute β-réduction $M \to_\beta^* M''$, $M'' \to_\beta^* M'$, *i.e.* il existe une unique forme β-normale de M.

Nous finissons cette partie sur les propriétés de β-réduction de termes simplement typés, en introduisant la notion d'arbres de typage normaux.

Définition 2.3.3.1. *Nous appelons* détour *toute succession de règles de la forme suivante :*

$$\dfrac{\dfrac{\langle \mathbf{\Delta}; \Gamma_1, x : \alpha, \Gamma_2 \rangle \vdash M : \beta}{\langle \mathbf{\Delta}; \Gamma_1, \Gamma_2 \rangle \vdash \lambda x.M : \alpha \to \beta} \qquad \dfrac{}{\langle \mathbf{\Delta}; \Gamma_1, \Gamma_2 \rangle \vdash N : \alpha}}{\langle \mathbf{\Delta}; \Gamma_1, \Gamma_2 \rangle \vdash (\lambda x.M)N : \beta}$$

Un arbre de typage sans détours est dit normal.

Il est important de remarquer que les détours dans une arbre de typage correspondent à la création de β-redex dans le terme associé. En règle générale, nous nous intéresserons aux arbres de dérivation et de typage normaux, et travaillerons donc à introduction de détours près.

Lemme 2.3.3.1. *Soient un terme M sous forme β-normale et un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de M. Il existe un unique arbre de typage de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$*

Remarquons que cette propriété n'est cependant pas vraie dans le cas le plus général où le terme M n'est pas sous forme normale. En effet, il suffit de considérer le terme $M = (\lambda x.\mathbf{c})(\lambda x.x)$ et le typage $\langle \mathbf{c} : a; \_ \rangle \vdash a$. Il existe alors autant de dérivations de $\langle \mathbf{c} : a; \_ \rangle \vdash M : a$ que de typages pour $\lambda x.x$.

2.3.4 Structures de termes et β-expansion

Nous nous intéressons à présent à la préservation des typages par β-expansion, c'est-à-dire aux conditions sous lesquelles tout typage d'un terme M' tel que $M \to_\beta^* M'$ est également un typage de M. Bien que cette propriété ne soit pas vraie en générale, l'ajout de certaines contraintes sur les β-réductions permet de définir des familles de termes simplement typés vérifiant la préservation de typages par β-expansion.

Définition 2.3.4.1. *Une β-contraction $C[(\lambda x.N)P] \to_\beta C[N[x := P]]$ est dite :*

- effaçante *si $x \notin FV(N)$; dans un tel cas, $C[N[x := P]] = C[N]$.*
- dupliquante *si x a plus d'une occurrence libre dans N.*

Nous dirons qu'une β-contraction est linéaire *si elle n'est ni effaçante, ni dupliquante ; elle sera dite* affine *si elle n'est pas dupliquante.*

Enfin, une β-réduction sera dite effaçante (resp. dupliquante, linéaire, affine) si elle forme une séquence de β-contractions effaçantes (resp. dupliquantes, linéaires, affines).

Propriété 2.3.4.1. ***(Expansion du sujet)*** *Soit un terme M tel que $M \to_\beta^* M'$ par une β-réduction linéaire. Alors tout typage de M' est un typage de M.*

Afin d'illustrer ce dernier résultat, considérons le terme $(\lambda f.g(\lambda x.fx)(\lambda y.fy))(\lambda x.x)$ dont le typage principale est $\langle \_; g : (b \to b) \to (b \to b) \to a \rangle \vdash a$. Ce terme se β-réduit en $g(\lambda x.x)(\lambda y.y)$ dont le typage principale est $\langle \_; g : (b \to b) \to (c \to c) \to a \rangle \vdash a$; de plus, cette β-réduction n'est pas linéaire, et le typage principale de M' ne peut être un typage de M. D'un autre côté, le terme $N = (\lambda x.fx)(\mathbf{c})$ se β-réduit par une β-réduction linéaire, en $N' = f\mathbf{c}$, et N et N' partagent le même typage principale $\langle \mathbf{c} : a; f : a \to b \rangle \vdash b$.

Définition 2.3.4.2. *Un terme M est dit* linéaire *si :*

1. $M = x \in \mathcal{V}$ *ou* $M = \mathbf{c} \in C$
2. $M = \lambda x.M'$*, il existe une et une seule occurrence libre de x dans M' et M' est linéaire.*
3. $M = M_1 M_2$*, où $FV(M_1) \cap FV(M_2) = \emptyset$ et M_1, M_2 sont linéaires.*

Un terme M est dit affine *si les conditions 2 et 3 sont remplacées par :*

2' . $M = \lambda x.M'$*, il existe au plus une occurrence libre de x dans M' et M' est affine.*

3' $M = M_1 M_2$*, où $FV(M_1) \cap FV(M_2) = \emptyset$ et M_1, M_2 sont affines.*

Exemple 2.3.4.1. Le terme $(\lambda x.\mathbf{c}(\mathbf{c}(xz)))(\lambda y.y)$ est un terme linéaire (et donc affine) ; par contre, le terme $(\lambda x.\mathbf{c}(\mathbf{c}z))(\lambda y.y)$ n'est pas linéaire, mais est affine. Le terme $(\lambda x.y(y(xz)))(\lambda y.y)$ n'est ni linéaire, ni affine.

Le lien entre termes linéaires (*resp.* affines) et β-réductions linéaires (*resp.* affines) est direct : étant donné un terme linéaire (*resp.* affine) M, toute β-réduction de M est linéaire (*resp.* affine). De la Propriété 2.3.4.1, nous pouvons déduire le théorème suivant :

Théorème 2.3.4.1. *Étant donnés deux termes simplement typables et linéaires M et M' tels que $M \to_\beta^* M'$, tout typage de M' est un typage de M.*

Nous finissons par souligner une dernière propriété intéressante des termes linéaires et affines :

Propriété 2.3.4.2. ***(Stabilité par β-réduction)*** *Soit M un terme simplement typé.*

- *si M est linéaire, tout terme M' tel que $M \to_\beta^* M'$ est un terme linéaire.*
- *si M est affine, tout terme M' tel que $M \to_\beta^* M'$ est un terme affine.*

2.4 Logique intuitionniste et λ-calcul simplement typé

Comme nous l'avons brièvement vu plus haut, le λ-calcul est amplement utilisé en informatique comme un modèle de calculabilité, et permet de représenter des programmes et leurs exécutions. Par ailleurs, nous avons mentionné le fait que le typage d'un λ-terme peut également être vu comme le typage d'un programme. Nous montrons à présent qu'un système de typage peut également être vu comme un système logique, permettant de prouver la validité de certaines formules.

2.4.1 Logique minimale et correspondance de Curry-Howard

Étant donnée un ensemble énumérable de formules atomiques, nous définissons l'ensemble des formules d'une logique implicative, comme la clôture de cet ensemble par l'opérateur $\rightarrow$.

Nous allons à présent définir les règles de la *logique intuitionniste implicative* (ou *logique minimale*) ; ces règles peuvent être représentées sous différentes formes, principalement le *calcul des séquents* ou la *déduction naturelle*, formes qui sont toutes les deux décrites dans [Girard, 1989]. Nous choisirons ici la deuxième représentation, tout en dérivant des séquents.

Définition 2.4.1.1. *Soit un ensemble E d'éléments. Un* multi-ensemble *F de E est une fonction totale de E dans $\mathbb{N}$.*

Les éléments d'un multi-ensemble $F : E \mapsto \mathbb{N}$ sont les éléments $e \in E$ tels que $F(e) \neq 0$.

Exemple 2.4.1.1. Soit $E = \{a, b, c\}$. Nous pouvons construire un multi-ensemble F sur E défini par $F(a) = 3, F(b) = 2, F(c) = 0$, et que nous noterons comme une séquence non-ordonnée : a, b, a, a, b.

Notation 2.4.1. Par la suite, nous prendrons E comme un ensemble fini et adopterons les notations suivantes :

1. $|\Gamma| = n$ désignera la taille du multi-ensemble fini $\Gamma : E \mapsto \mathbb{N}$ et sera défini comme $|\Gamma| = \Sigma_{e \in E} \Gamma(e)$.
2. L'union (disjointe) de deux multi-ensembles de formules $\Gamma_1 : E \mapsto \mathbb{N}$ et $\Gamma_2 : E \mapsto \mathbb{N}$ forme un nouveau multi-ensemble $\Gamma : E \mapsto \mathbb{N}$ tel que, pour tout élément e de E, $\Gamma(e) = \Gamma_1(e) + \Gamma_2(e)$; nous noterons donc cette union comme la concaténation des deux séquences associées : Γ_1, Γ_2.

Un *séquent de la logique minimale* est une expression de la forme $\Gamma \vdash \alpha$, où $\Gamma = \alpha_1, \ldots, \alpha_n$ est un multi-ensemble fini sur l'ensemble des formules de la logique minimale, appelé *l'antécédent* et α est une formule de la même logique appelée le *conséquent*. Un tel séquent s'interprète de la manière suivante :

sous les hypothèses $\alpha_1, \alpha_2 \ldots$ et α_n, nous pouvons déduire α.

Les règles de la logique minimale sont données sous la forme générale $\dfrac{S_1 \ldots S_n}{S}$ où $S_1, \ldots, S_n$ sont appelés les *séquents hypothèses*, et S le *séquent conclusion*. Dans le cas de la règle d'Identité, $n = 0$, et il n'y a pas de séquent hypothèse.

Axiome

$$\frac{}{\Gamma, \alpha \vdash \alpha}\ \text{(Identité)}$$

Implication

$$\frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta}\ \text{(Élimination} \rightarrow\text{)} \qquad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta}\ \text{(Introduction} \rightarrow\text{)}$$

FIG. 2.4 – Système de dérivation pour la logique minimale

Nous appellerons *arbre de dérivation* tout arbre construit en utilisant les règles du système de la Figure 2.4. La racine $\Gamma \vdash \alpha$ d'un tel arbre sera appelée un *séquent valide* de la logique minimale. Nous dirons alors que cet arbre de dérivation est une *démonstration* de $\Gamma \vdash \alpha$ dans la logique minimale.

Exemple 2.4.1.2. Une démonstration du séquent $a, a \rightarrow b, b \rightarrow c \vdash c$ est la suivante :

$$\dfrac{\dfrac{}{a, a \to b, b \to c \vdash a} \qquad \dfrac{}{a, a \to b, b \to c \vdash a \to b}}{\dfrac{a, a \to b, b \to c \vdash b}{a, a \to b, b \to c \vdash c}} \qquad \dfrac{}{a, a \to b, b \to c \vdash b \to c}$$

Remarque 2.4.1.1. La règle d'élimination de $\to$ est plus connue sous le nom de *modus ponens*.

Nous venons de définir les arbres de dérivation dans la logique minimale. Il apparaît clairement que les règles servant à la construction de tels arbres sont similaires à celles de la Figure 2.1, pour les arbres de typage. Comme nous l'avons vu précédemment, un arbre de typage correspond exactement à un λ-terme simplement typé. Cette correspondance peut être décrite selon le schéma suivant (voire également [Girard, 1989]) :

logique minimale	λ-calcul simplement typé
formule	type
démonstration	λ-terme simplement typé

Cette propriété est en fait plus connue sous le nom de correspondance de Curry-Howard (voire [Sørensen and Urzyczyn, 2006] pour plus de détails) :

Théorème 2.4.1.1. ***(Correspondance de Curry-Howard)***
Le séquent $\alpha_1, \ldots, \alpha_n \vdash \alpha$ est valide dans la logique minimale ssi il existe un terme pur simplement typé M tel que $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash \alpha$ soit un typage de M.

Finalement, la notion de démonstration normale peut également être définie pour des démonstrations de la logique minimale. Celles-ci correspondent effectivement aux arbres de typages normaux dans le cadre du typage de λ-termes à la Curry.

2.4.2 Unicité de démonstrations en logique minimale

Nous abordons ici un résultat qui est au centre de l'étude que nous mènerons par la suite sur les grammaires de λ-termes simplement typés : la caractérisation de formules pour lesquelles il existe une unique démonstration (à élimination des détours près) dans la logique minimale. La caractérisation de ces formules est basée sur des critères syntaxiques introduits par la notion de polarité qui suit :

Définition 2.4.2.1. *Soit un séquent $\Gamma \vdash \alpha$ de la logique minimale. La* polarité *d'une formule est définie par induction comme :*

1. *α a une occurrence positive dans $\Gamma \vdash \alpha$*
2. *toute formule β de Γ a une occurrence négative dans $\Gamma \vdash \alpha$*
3. *pour une formule positive $\beta \to \gamma$ dans $\Gamma \vdash \alpha$, β a une occurrence négative et γ une occurrence positive.*
4. *pour une formule négative $\beta \to \gamma$ dans $\Gamma \vdash \alpha$, β a une occurrence positive et γ une occurrence négative.*

Nous noterons parfois p la polarité d'une formule et $\overline{p}$ la polarité inverse de p.

Définition 2.4.2.2. *Un séquent $\Gamma \vdash \alpha$ est dit* balancé *si :*

1. *toute formule atomique de $\Gamma \vdash \alpha$ a au plus une occurrence positive dans $\Gamma \vdash \alpha$.*
2. *toute formule atomique de $\Gamma \vdash \alpha$ a au plus une occurrence négative dans $\Gamma \vdash \alpha$.*

Le séquent sera dit négativement non-dupliquant *si et seulement si le critère 2 est vérifié.*

Le problème de l'unicité des démonstrations dans la logique minimale a d'abord été formulé dans [Mints, 1979].

Théorème 2.4.2.1. ***(Unicité)***

- *[Babaev and Soloviev, 1982] Il existe une unique démonstration normale de tout séquent balancé valide dans la logique minimale.*
- *[Aoto, 1999] Il existe une unique démonstration normale de tout séquent négativement non-dupliquant valide dans la logique minimale*

D'après la correspondance de Curry-Howard, nous savons qu'une démonstration correspond à un λ-terme. Par ailleurs, une démonstration sans détours correspond à un λ-terme sous forme β-normale. Ces théorèmes permettent donc d'identifier un λ-terme sous forme β-normale de manière unique avec certains de ces typages, si ces derniers respectent les contraintes syntaxiques exprimées dans les Théorèmes 2.4.2.1. Ainsi, si un terme M habite un typage balancé ou négativement non-dupliquant $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, tout terme habitant $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ a la même forme β-normale que M. L'étude de familles de termes uniquement identifiés par un certain typage sera au centre de notre étude.

Remarquons que les termes linéaires et affines que nous avons introduits en Définition 2.3.4.2 sont en fait des termes pour lesquels cette propriété est vérifiée :

Propriété 2.4.2.1. ***(Unicité terme-typage)***

1. *[Belnap, 1976] le typage principal $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ d'un terme linéaire est balancé.*
2. *[Hirokawa, 1991] tout habitant d'un typage balancé est une terme affine.*

Corollaire 2.4.2.1. *Un terme linéaire (resp. affine) est le seul habitant de son typage principal.*

2.5 Conclusion

Au cours de ce chapitre, nous avons introduit le λ-calcul simplement typé, qui sera l'objet principal de l'étude qui suit. Nous avons présenté ce formalisme sous ses deux formes les plus connues : le λ-calcul simplement typé à la Curry, et le λ-calcul simplement typé à la Church. Nous avons montré que ces deux formalismes sont équivalents, dans le sens où ils permettent de typer exactement les mêmes termes. Par la suite, nous utiliserons donc indistinctement l'un ou l'autre de ces systèmes. Nous avons également donné certaines notions de typages importantes, telles que la notion de typage associé à un sous-terme d'un terme relativement à un typage de ce dernier, ou encore la notion de typage principal. Enfin, nous avons défini certaines familles de termes ayant des propriétés de typage que nous étudierons plus précisément dans la suite de ces travaux : les termes linéaires et les termes affines. Ces termes ont, en particulier, la propriété d'être les uniques habitants de leur typage principal. Nous montrons par la suite que cette propriété est au centre de l'analyse des grammaires catégorielles abstraites, et qu'il est possible d'étendre ce résultat à de nouvelles familles de termes plus grande que les famille de termes linéaires et quasi-linéaires.

Chapitre 3

Jeux et λ-calcul simplement typé

Au cours du chapitre précédent, nous avons vu le lien entre logique et λ-calcul. De même, nous avons brièvement évoqué le fait que le λ-calcul soit un modèle de la calculabilité. Une approche différente de la logique est proposée par les sémanticiens, qui étudient les modèles mathématiques dans lesquels les systèmes logiques s'interprètent.

Un des modèles développés en sémantique de la logique est la sémantique des jeux ; des modèles des preuves y sont donnés sous forme de jeux à deux joueurs ce qui permet de mettre en valeur le rôle et l'interaction entre les différentes formules atomiques dans la construction de preuves. La discipline est née des travaux de Lorenz [Lorenz, 1968] et Lorenzen [Lorenzen, 1959] qui définirent des jeux de dialogue pour la logique intuitionniste. Plus récemment et avec l'avènement de la logique linéaire [Girard, 1987, Girard, 1995], un intérêt grandissant s'est développé pour la sémantique des jeux à partir des travaux de Blass [Blass, 1992], lesquels furent ensuite repris par Abramsky [Abramsky and McCusker, 1999], Ong et Hyland [Hyland and Ong, 2000] afin de résoudre le problème d'abstraction complète pour le langage PCF grâce à une notion de jeu.

Nous introduisons ici des jeux que nous utiliserons par la suite afin de caractériser, d'une part, des familles de typages ayant au plus un habitant, et d'autre part, les termes normaux ayant pour typage principal des typages ayant la propriété précédente. Au cours du Chapitre précédent, nous avons évoqué les typages balancés et négativement non-dupliquants. Ces derniers ont au plus un habitant, et sont caractérisés par des propriétés sur les polarités des types atomiques les composant. Comme nous le verrons, la sémantique des jeux s'intéresse exactement à l'étude de dérivations d'arbres de typage à travers l'interaction des types atomiques polarisés d'un typage et offre donc un modèle où la caractérisation de typages ayant au plus un habitant est aisé. De plus, elle donne une représentation abstraite aussi bien de la notion de typage que de celle de λ-termes ; une fois certaines familles de typage caractérisées, il est donc possible de déterminer les termes qui, par exemple, ont pour typage principal un typage de la famille précédemment évoquée. À l'image des réseaux de preuves pour la logique linéaire multiplicative [Danos and Regnier, 1989] ou pour la logique linéaire intuitionniste [Lamarche, 1994], nous nous appuierons sur une représentation graphique de la notion de jeux. L'utilisation que nous ferons ici de ces jeux n'est donc pas destinée à donner une sémantique du λ-calcul simplement typé, mais plutôt à obtenir une représentation alternative des démonstrations dans la logique minimale, tout en ayant un accès plus précis à l'interaction des occurrences de formules atomiques des séquents ainsi dérivés.

3.1 Typage et interaction

Dans cette première partie, nous donnons quelques intuitions sur l'apport de la sémantique des jeux à l'étude de propriétés de typage. Comme nous venons de l'énoncer, la sémantique des jeux permet d'observer plus précisément le rôle des formules atomiques polarisées dans la structure des démonstrations, et plus exactement l'interaction entre ces formules atomiques. La mise en exergue de cette interaction se retrouve par exemple dans la géométrie de l'interaction [Girard, 1994] ou la ludique [Girard, 2001].

Afin d'illustrer notre propos, prenons le typage $f : a \to b, x : a \vdash b$ du λ-terme fx et intéressons-nous à l'arbre de typage normal de $f : a \to b, x : a \vdash fx : b$. La construction de cet arbre peut être vue comme un jeu à deux joueurs : *P (le proposant)* et *O (l'opposant)*. Nous associons les types atomiques de polarité positive aux coups joués par le joueur O et les types atomiques de polarité négative aux coups joués par le joueur P. Pour plus de clarté, nous laissons apparaître ces polarités dans le typage : $f : a^+ \to b^-, x : a^- \vdash b^+$. Sur cet exemple, le but du jeu est donc de construire un terme de type b avec les types donnés $a^+ \to b^-$ et a^-. Le joueur O demande à chaque tour au joueur P de construire un terme d'un certain type. Ainsi, le joueur O commence la partie, et demande au joueur P si il est capable de construire un terme de type b sur notre exemple. Ce dernier n'a qu'une seule possibilité : jouer le coup correspondant à b^-. Le jeu se poursuit, et afin de pouvoir valider le coup précédent de P, le joueur O demande à présent à P de créer un terme de type a. Le joueur P peut alors jouer le dernier coup correspondant à $x : a^-$ et le jeu se termine sur une victoire de P, puisque O ne peut plus jouer.

0	$f : a^+$	$\to$	b^-	$x : a^-$	$\vdash$	b^+
1						O
2			P			O
3	O		P			O
4	O		P	P		O

Sur cet exemple très simple, nous pouvons voir que la séquence de coups joués alternativement par les joueurs P et O correspond à la dérivation du terme fx ; en effet, le joueur O demande au joueur P de créer un terme de type b, ce dernier répond en proposant d'utiliser f ; le joueur O demande alors de créer un nouveau terme de type a en argument de f, ce à quoi P répond en utilisant x.

L'interprétation est analogue sur l'exemple suivant, où f est typé par $a_1 \to a_2 \to b$. Ainsi, le joueur O a deux possibilités de jeu après l'étape 2, correspondant chacune à un des arguments de f. Il existe donc deux séquences de coups possibles, P l'emportant pour chacune d'elles :

0	$f : a_1^+$	$\to a_2^+$	$\to b^-$	$x_1 : a_1^-$	$x_2 : a_2^-$	$\vdash$	b^+
1							O
2			P				O
3.1	O		P				O
4.1	O		P	P			O
3.2		O	P				O
4.2		O	P		P		O

Les exemples précédents cachent cependant une des contraintes sur les coups joués par les joueurs O et P, exprimée par la *relation d'accessibilité* ; cette relation spécifie que les coups que O ou P peuvent jouer dépendent des coups précédemment joués. Typiquement, sur un type $a \to b$, le coup associé à l'occurrence du type atomique a n'est jouable que si le coup associé à l'occurrence du type atomique b a été joué au préalable. De la même manière, sur le typage $f : o^+ \to o^-, x : o^- \vdash o^+$,

les coups correspondants aux occurrences de o^- ne sont jouables que si O a joué le premier coup correspondant à l'occurrence de o^+ en conclusion de ce typage.

3.2 Arènes et typages

3.2.1 Arènes

Nous formalisons à présent les intuitions données sur les exemples qui précèdent.

Définition 3.2.1.1. *Nous noterons $\mathbb{N}^* = \langle \mathbb{N}, \cdot, \epsilon \rangle$ le monoïde libre sur $\mathbb{N}$, où :*

1. *$\mathbb{N}$ est l'ensemble des entiers naturels (l'alphabet).*
2. *$\cdot$ est la loi de concaténation associative.*
3. *ϵ est l'élément neutre pour $\cdot$.*

Autrement dit, $\mathbb{N}^*$ représente l'ensemble des mots construits sur l'alphabet des entiers naturels. Pour une partie $\mathcal{P}$ de $\mathbb{N}^*$, la loi de composition est étendue de sorte que $\mathcal{P} \cdot i$ (pour $i \in \mathbb{N}$) soit égale à l'ensemble $\{s \cdot i \mid s \in \mathcal{P}\}$; de même $i \cdot \mathcal{P} = \{i \cdot s \mid s \in \mathcal{P}\}$. Nous définissons également la longueur d'un élément de $\mathbb{N}^*$ par induction sur cet élément : $|\epsilon| = 0$, $|i| = 1$, pour $i \in \mathbb{N}$ et $|s_1 \cdot s_2| = |s_1| + |s_2|$ sinon. Enfin, pour un mot $s = s_1 \cdot s_2$, nous dirons que s_1 est un préfixe de s.

Notation 3.2.1. Nous noterons les éléments de $\mathbb{N}$ par $i, j, k, \ldots$ éventuellement indexés ; les éléments de $\mathbb{N}^*$ seront notés $s, s_1, s_2, \ldots$ Nous adopterons la notation usuelle permettant d'omettre l'opération de concaténation par défaut (par exemple, 21 sera l'élément $2 \cdot 1$; $10 \cdot 2$ ne pourra pas s'écrire 102).

Nous introduisons à présent la notion d'*arène de type*. De manière analogue à [Nickau, 1994, Ker et al., 2002, Laurent, 2004], nous présentons l'arène associée à un type comme un arbre étiqueté enraciné :

Définition 3.2.1.2. *Soit un type $\alpha = \alpha_1 \to \ldots \to \alpha_n \to a \in \mathcal{T}(\mathcal{A})$, où $n \geq 0$ est un entier et a un type atomique. L'*arène de type *$A_\alpha = (M_\alpha, \tau_\alpha)$ associée à α est définie inductivement par :*

1. *$M_\alpha \subseteq \mathbb{N}^*$ est un ensemble fini de* coups *tel que*

$$M_\alpha = \{\epsilon\} \cup \bigcup_{i \in \{1,\ldots,n\}} i \cdot M_{\alpha_i}$$

2. *$\tau_\alpha : M_\alpha \mapsto \mathcal{A}$, appelée* fonction de typage *de A_α, est définie par :*
   - *$\tau_\alpha(\epsilon) = a$*
   - *$\tau_\alpha(i \cdot s) = \tau_{\alpha_i}(s)$, pour $i \in \mathbb{N}$ et $s \in \mathbb{N}^*$.*

Pour un type α donné, l'arène associée est donc représentée par un ensemble fini de coups, clos par préfixes, où chaque coup correspond à une occurrence de type atomique dans α (via la fonction de typage τ_α).

Définition 3.2.1.3. *Étant donnée une arène (M_α, τ_α), nous définissons :*

1. *la* relation d'accessibilité *entre coups par : $s_1, s_2 \in M_\alpha$ vérifient $s_1 \vdash s_2$ si et seulement si il existe $i \in \mathbb{N}$ tel que $s_2 = s_1 \cdot i$.*
2. *la relation de* précédence *$\prec$ entre coups : $s_1 \prec s_2$ est vérifiée ssi il existe $i, j \in \mathbb{N}$ et $s \in M_\alpha$ tels que $s_1 = s \cdot i$, $s_2 = s \cdot j$ et $i < j$.*

3. *une fonction totale* $pl : M_\alpha \mapsto \{O, P\}$ *telle que* $pl(s)$ *est défini par induction sur* s *:*
   - $pl(\epsilon) = O$
   - *si* $pl(s_1) = P$ *(resp.*$pl(s_1) = O$*) et que* $s_1 \vdash s_2$ *alors* $pl(s_2) = O$ *(resp.* $pl(s_2) = P$*).*

Remarque 3.2.1.1. Les relations d'accessibilité et de précédence impliquent un ordre total sur (M_α, τ_α) tel que s_1 est plus petit que s_2 si il existe un préfixe s_1' de s_1 et un préfixe s_2' de s_2 tels que $s_1' \prec s_2'$ ou bien $s_1 \vdash s_2'$.

La fonction pl associe un joueur à chaque coup de l'arène. L'idée est donc de déterminer les coups pouvant être joués par chacun des deux joueurs O (l'opposant) et P (le proposant). Cette fonction induit donc une partition des coups de A_α en deux parties : M_α^P l'ensemble des coups assignés au joueur P, et M_α^O ceux assignés au joueur O. De plus, nous définissons une fonction d'inversion inv : $\{O, P\} \mapsto \{O, P\}$ par $inv(P) = O$ et $inv(O) = P$. Nous adopterons la notation $\overline{pl}$ pour $inv \circ pl$.

Remarque 3.2.1.2.
1. Pour toute fonction d'assignation de joueur pl, nous obtenons $\overline{\overline{pl}} = pl$ (autrement dit, $inv \circ inv$ est l'identité).
2. Le joueur assigné à un coup correspond en fait à la polarité des occurrences de types atomiques dans un type α. Ainsi, considérant que le type α est de polarité positive par défaut, les coups de O correspondent aux occurrences positives de types atomiques dans α, et les coups de P aux occurrences négatives de ces derniers.
3. Les coups des arènes définies dans [Hyland and Ong, 2000, Abramsky et al., 2000] possèdent une étiquette supplémentaire permettant de séparer les coups qui représentent des questions (**Q**) de ceux qui correspondent à des réponses (**A**). Ces étiquettes ne sont pas nécessaires dans les arènes que nous utiliserons par la suite.

Exemple 3.2.1.1. Au type $a \to (b \to c) \to d$, nous associons l'arène (M, τ), telle que :

1. $M = \{\epsilon, 1, 2, 2 \cdot 1\}$
2. $M^O = \{\epsilon, 2 \cdot 1\}$ et $M^P = \{1, 2\}$
3. $\tau(\epsilon) = d$, $\tau(1) = a$, $\tau(2) = c$, $\tau(2 \cdot 1) = b$

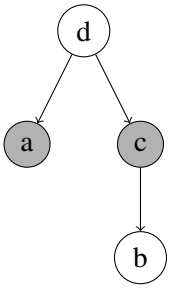


FIG. 3.1 – Exemple d'arène pour $a \to (b \to c) \to d$

L'arène associée à ce type peut être représentée sous la forme de l'arbre étiqueté de la Figure 3.1. Les coups du joueur P y sont représentés en gris, et les coups du joueur O en blanc. Une flèche dirigée d'un nœud s_1 vers un nœud s_2 représente la relation d'accessibilité $s_1 \vdash s_2$.

La définition d'arène associée à un type est étendue, de manière à associer une arène à une assignation de type $x : \alpha$. Les coups de l'arène associée seront alors des éléments de $\mathbb{N}^* \times \mathcal{V}$, notés s_x. Ainsi, l'arène $A_{x:\alpha} = (M_{x:\alpha}, \tau_{x:\alpha})$ associée à l'assignation $x : \alpha$ est définie comme :

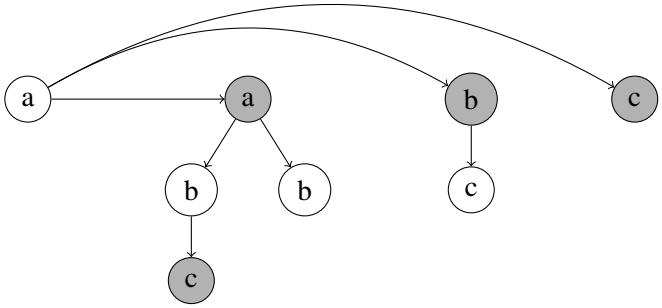


FIG. 3.2 – Exemple d'arène pour $x : (c \to b) \to b \to a, y : c \to b, z : c \vdash a$

1. s_x est un coup de $M_{x:\alpha}$ si et seulement si $s \in M_\alpha$
2. Pour tout $s \in \mathbb{N}$, $\tau_{x:\alpha}(s_x) = \tau_\alpha(s)$.

De plus, les relations d'accessibilité et de précédence sont préservées (*i.e.* $s_x \vdash s'_x$ ssi $s \vdash s'$, et $s_x \prec s'_x$ ssi $s \prec s'$), et il en est de même pour la fonction d'assignation d'un joueur à un coup : $pl(s) = pl(s_x)$. Par ailleurs, tout préfixe d'un coup s_x est de la forme s'_x de telle sorte que s' est un préfixe de s.

Notation 3.2.2. De manière générale, nous noterons les coups d'une arène par $m, m_1, n, n_i, \ldots$. Pour un coup $s_x \in \mathbb{N} \times \mathcal{V}$ et $i \in \mathbb{N}$, nous noterons par $s_x \cdot i$ le coup $(s \cdot i)_x$. Ainsi, nous étendons l'opération de concaténation aux coups d'une arène.

Nous définissons à présent les arènes associées aux typages :

Définition 3.2.1.4. *Soit un typage $\Gamma \vdash \alpha$, où $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$. L'*arène de typage $A = (M, \tau)$ *associée est définie comme :*

1. $M = M_\alpha \cup \bigcup_{i \in \{1,\ldots,n\}} M_{x_i:\alpha_i}$
2. $\tau(m) = \begin{cases} \tau_\alpha(m) & \text{pour } m \in M_\alpha \\ \tau_{x_i:\alpha_i}(m) & \text{pour } m \in M_{x_i:\alpha_i} \end{cases}$

De plus, la relation d'accessibilité est étendue de sorte que $m_1 \vdash m_2$ soit vérifiée ssi
- $m_2 = m_1 \cdot j$ pour un certain $j \in \mathbb{N}$ ou
- $m_1 = \epsilon \in M_\alpha$ and $m_2 = \epsilon_{x_i}$ pour un certain $i \in \{1, \ldots, n\}$.

De même, nous définissons la fonction $pl : M \mapsto \{O, P\}$ associée à une arène de typage de manière identique à celle associée à une arène de type. Nous remarquerons que cette fonction correspond bien à la définition des polarités pour les types atomiques du typage.

Par ailleurs, les relations de précédence (*resp.* de suffixes) ne sont pas modifiées : $m_1 \prec m_2$ (*resp.* m_1 est un suffixe de m_2) dans M si il existe $M' \in \{M_{x_i:\alpha_i} \mid i \in \{1, \ldots, n\}\} \cup \{M_\alpha\}$ tel que $m_1, m_2 \in M'$ et $m_1 \prec m_2$ (*resp.* m_1 est un suffixe de m_2) dans M'.

Une arène de typage peut à nouveau être représentée comme un arbre étiquetée. L'ordre induit par cette structure arborescente n'est pas totalement ordonnée, du fait que la relation de précédence ne soit pas totale, rendant ainsi compte du fait que les environnements de typage dans le λ-calcul simplement typé sont des multi-ensembles non ordonnés.

Étant donnée une arène de typage $A = (M, \tau)$, le coup $m = \epsilon$ est le seul coup tel qu'il n'existe pas de coup $m' \in M$ vérifiant $m' \vdash m$. Ce coup sera donc appelé coup *initial*. Nous donnons un exemple

d'arène de typages en Figure 3.2, en étendant la représentation graphique donnée en Figure 3.1 pour les arènes de types.

Nous pouvons également remarquer que la notation adoptée pour les coups associés à une assignation de types permet de différencier syntaxiquement les arènes $\Gamma, x : \beta \vdash \alpha$ et $\Gamma \vdash \beta \to \alpha$. En effet, nous désirons utiliser les jeux de typages afin d'y représenter des typages et des λ-termes habitant ces derniers. Il nous faut donc des arènes différentes pour représenter les typages d'un terme M (habitant du premier typage), et ceux du terme $\lambda x.M$ (habitant du second).

3.2.2 Occurrences de types et sous-arènes

Nous introduisons, à présent, la notion de sous-arènes d'une arène de typage. L'objectif de cette définition est de donner une représentation des occurrences de types, dans le cadre de la sémantique des jeux.

Définition 3.2.2.1. *Soit une arène de typage $A = (M, \tau)$ et un ensemble fini de coups $M' \subseteq M$. Nous appellerons (M', τ) une* sous-arène de A *si :*

1. *il existe $m \in M'$ (noté $\widehat{M'}$) préfixe commun à tout mouvement $m' \in M'$.*
2. *pour $m_1, m_2 \in M'$, si $\widehat{M'} \vdash m_1$ et $\widehat{M'} \vdash m_2$, alors pour tout $m_3 \in M$ tel que $m_1 \prec m_3 \prec m_2$, $m_3 \in M'$.*
3. *M' est clos par suffixes.*

La notion de sous-arène nous sera particulièrement utile pour désigner des occurrences de types dans un typage. En effet, il est possible d'associer un type à une sous-arène d'une arène de typage, par le simple fait qu'une sous-arène d'une arène est toujours en bijection avec l'arène associée à un certain type.

Exemple 3.2.2.1. Sur l'exemple de la Figure 3.2, le sous-ensemble de coups $M' = \{1_x, 11_x\}$ permet de construire une sous-arène de l'arène représentée. Cette sous-arène est associée à une occurrence du type $c \to b$ dans le typage $x : (c \to b) \to b \to a, y : c \to b, z : c \vdash a$

Notation 3.2.3. Pour une sous-arène (M', τ) d'une arène $A = (M, \tau)$, nous noterons par $\sigma_A(M') = \gamma$ le type de $\Gamma \vdash \alpha$ ainsi associé à (M', τ). Enfin, nous noterons par $\mathcal{F}(A)$ l'ensemble des sous-arènes d'une arène A.

3.2.3 Curryfication, substitutions de types et morphismes d'arènes

Nous venons d'introduire la notion d'arène de typage. Nous présentons à présent des opérations de transformation d'arènes, pour répondre aux deux objectifs suivants : obtenir une notion de (dé)curryfication (*i.e.* passer d'une arène associée à un typage $\Gamma, x : \alpha \vdash \beta$ à l'arène associée à $\Gamma \vdash \alpha \to \beta$ et vice-versa) et une notion de substitution de type sur les arènes.

Étant données deux arènes $A_1 = (M_1, \tau_1)$ et $A_2 = (M_2, \tau_2)$, nous définissons une *substitution de coups* comme une fonction bijective associant à chaque coup de M_1 une sous-arène de A_2 ; de plus, cette fonction préserve certaines conditions sur la fonction de typage et la relation d'accessibilité.

Définition 3.2.3.1. *Soient deux arènes de typage $A_1 = (M_1, \tau_1)$ et $A_2 = (M_2, \tau_2)$. Une* substitution de coups $F : A_1 \mapsto \mathcal{F}(A_2)$ *vérifie :*

1. $\widehat{F(\epsilon)} = \epsilon$.
2. *soient m, n des coups de M_1 ; si $m \vdash n$ alors $\widehat{F(m)} \vdash \widehat{F(n)}$.*
3. *soient m_1, m_2 des coups de M_1 ; si $\tau_1(m_1) = \tau_2(m_2)$ alors $\sigma_{A_2}(F(m_1)) = \sigma_{A_2}(F(m_2))$*

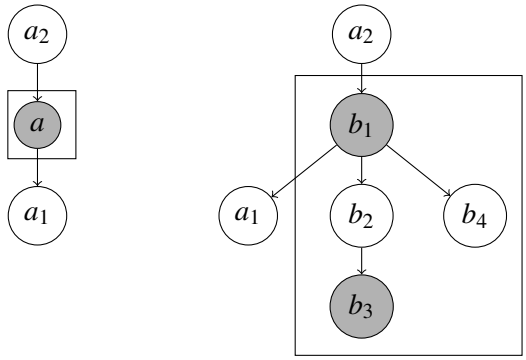


FIG. 3.3 – Substitution de types $[a \mapsto (b_3 \to b_2) \to b_4 \to b_1]$ sur le type $(a_1 \to a) \to a_2$

Remarquons que les points 1. et 2. impliquent que la fonction pl est préservée par application de substitution de coups de la manière suivante : pour tout coup m de M_1, $pl(m) = pl(\widehat{F(m)})$.

Nous dirons qu'une substitution d'arène $F : (M_1, \tau_1) \mapsto \mathcal{F}((M_2, \tau_2))$ est un *ré-étiquetage de coups* si pour tout coup m de M_1, $F(m)$ est un singleton. De plus, si $\tau_2(\widehat{F(m_1)}) = \tau_2(\widehat{F(m_2)})$ ssi $\tau_1(m_1) = \tau_2(m_2)$, alors F sera appelé un *renommage de coups*. Dans ce cas, il existe un renommage de coups noté F^{-1} de (M_2, τ_2) vers $\mathcal{F}((M_1, \tau_1))$.

Notation 3.2.4. Si F est un ré-étiquetage de coups, alors $F(m)$ est le singleton $\{\widehat{F(m)}\}$; dans ce cas, nous confondrons les notations $F(m)$ et $\widehat{F(m)}$.

Notation 3.2.5. Étant données deux arènes A_1 et A_2, nous noterons par $A_1 \cong A_2$ l'existence d'un ré-étiquetage de coups $F : A_1 \mapsto \mathcal{F}(A_2)$.

Nous décrivons à présent deux substitutions de coups qui correspondent aux substitutions de types et à la (dé-)curryfication, et que nous utiliserons au cours de nos démonstrations :

1. étant données une arène $A = (M, \tau)$ associée à un typage $\Gamma \vdash \alpha$, une substitution de types $\sigma = [a \mapsto \gamma]$ et une arène $A' = (M', \tau')$ associée à $\Gamma \cdot \sigma \vdash \alpha \cdot \sigma$, il existe une substitution de coups $F_\sigma : A \mapsto \mathcal{F}(A')$ telle que $\widehat{F(m)} = m$ et :
   - si $\tau(m) \neq a$, $F(m) = \{m\}$ et $\tau(m) = \tau'(\widehat{F(m)})$
   - sinon, $\sigma_{A'}(F(m)) = \gamma$

   Un exemple graphique est donné en Figure 3.3.
2. pour une arène $A_1 = (M_1, \tau_1)$ associée à un typage $\Gamma, x : \gamma \vdash \alpha$ et $A_2 = (M_2, \tau_2)$ associée à $\Gamma \vdash \gamma \to \alpha$, il existe un renommage de coups $\mathtt{abs}_x : A_1 \mapsto \mathcal{F}(A_2)$ vérifiant :
   - $\mathtt{abs}_x(s_x) = \{1 \cdot s\}$ pour tout coup $s_x \in M_1$
   - $\mathtt{abs}_x(i \cdot s) = \{(i+1) \cdot s\}$ pour tout coup $i \cdot s$ de M_1, où $i \in \mathbb{N}$ et $s \in \mathbb{N}^*$
   - $\mathtt{abs}_x(m) = \{m\}$ pour tout autre coup m de M_1
   - pour tout coup m de M_1, $\tau_2(\mathtt{abs}_x(m)) = \tau_1(m)$.

3.3 Stratégies et λ-termes

Les arènes sont donc des objets associés à un typage, où les coups sont des objets associés aux types atomiques polarisés de ce dernier. Il nous reste à définir les conditions de jeu sur une arène, et les stratégies de jeu.

3.3.1 Justification, innocence et jeux

Une partie entre les joueurs P et O est représentée par une séquence de coups joués par ces joueurs. Nous définissons à présent des séquences particulières qui correspondent aux parties possibles sur une arène. Les définitions des séquences que nous introduisons seront affinées, jusqu'à définir les séquences innocentes, qui seront celles étudiées pour les arènes de typage.

Notation 3.3.1. Étant donnés une arène de typage $A = (M, \tau)$ et des coups $m_1, m_2 \ldots \in M$, une séquence de coups sur A sera notée $m_1.m_2 \ldots$; nous utiliserons les symboles $S, S_1, \ldots$ pour dénoter de telles séquences. Toutes les séquences de coups que nous étudierons seront des séquences finies.

Notation 3.3.2. Étant données deux séquences de coups $S_1 = m_1 \ldots m_i$ et $S_2 = m_{i+1} \ldots m_n$, nous écrirons la séquence $m_1 \ldots m_i.m_{i+1} \ldots m_n$ alternativement comme $S_1.S_2$. La relation de préfixe entre séquences de coups sur une arène de typage sera notée $\sqsubseteq$: $S_1 \sqsubseteq S$ ssi il existe une séquence de coups S_2 telle que $S = S_1.S_2$.

Soit S une séquence de coups sur une arène de typage $A = (M, \tau)$. Dans un premier temps, nous noterons les *occurrences de coups* dans S comme des paires $(m, n) \in M \times \mathbb{N}$ afin d'indiquer que le coup m apparaît en n-ième position en partant de la gauche sur S. Ainsi, une séquence finie S sera notée sous la forme : $(m_1, 1).(m_2, 2). \ldots (m_n, n)$; de même, nous noterons $(m, i) \in S$ si S est de la forme $S_1.(m, i).S_2$.

Définition 3.3.1.1. *Soit une arène $A = (M, \tau)$, et une séquence finie de coups $S = m_1 \ldots m_n$ sur cette arène. Nous introduisons une relation binaire $\vdash_S$, appelée* justification d'occurrences de coups dans S, *telle que $(m_i, i) \vdash_S (m_j, j)$ implique que :*

1. $(m_i, i), (m_j, j) \in S$, *et* $i < j$
2. $m_i \vdash m_j$
3. *si il existe (m_k, k) tel que $(m_k, k) \vdash_S (m_j, j)$ alors $m_k = m_i$ et $k = i$.*

Nous dirons que l'occurrence (m_i, i) justifie l'occurrence (m_j, j) *dans* S.
La séquence S est dite justifiée *si pour tout $i \in \{2, \ldots, n\}$, il existe $j \in \{1, \ldots, i-1\}$ tel que $(m_j, j) \vdash_S (m_i, i) \in S$.*

Cette relation est donc une extension de la relation d'accessibilité aux occurrences de coups dans une séquence donnée. Il est important de remarquer que, dans une séquence justifiée, toute occurrence de coups (hormis l'occurrence initiale) est justifiée par une (et une seule) occurrence de coup apparaissant précédemment dans la séquence.

Notation 3.3.3. Une occurrence de coups dans une séquence justifiée S sera notée $(m, i, j) \in M \times \mathbb{N} \times \mathbb{N}$ (où $j < i$). Étant donnée une telle séquence S et $(m, i, j) \in S$ où $i > 1$, il existe $k \in \mathbb{N}$ et $n \in M$ tels que (n, j, k) justifie (m, i, j) dans S. Cette relation sera à nouveau notée $(n, j, k) \vdash_S (m, i, j)$. Par ailleurs, pour le coup initial ϵ, nous noterons $(\epsilon, 1, 0)$ sa seule occurrence dans $S = (\epsilon, 1, 0).S'$.

Définition 3.3.1.2. *Soit une séquence justifiée non-vide de coups S sur une arène de typage A ; nous dirons que S est* bien formée *si :*

- ***(Initialisation)*** *$S = (m, 1, 0).S'$ implique que $m = \epsilon$.*
- ***(Alternance)*** *pour $S = S_1.(m_1, i_1, j_1).(m_2, i_2, j_2).S_2$, alors $pl(m_1) = \overline{pl}(m_2)$.*

Soit S une séquence justifiée de coups sur une arène de typage A. Nous définissons à présent les *vues* des joueurs P et O sur cette séquence comme des fonctions d'une séquence de coups sur A vers une autre séquence de coups sur A : ainsi, la P-vue de S (notée $\ulcorner S \urcorner$) est définie par :

$$\begin{array}{ll}
\ulcorner (m,1,0) \urcorner = (m,1,0) & \\
\ulcorner S.(m,i,j) \urcorner = \ulcorner S \urcorner.(m,i,j) & \text{si } m \text{ est un } P\text{-coup} \\
\ulcorner S_1.(m_1,j,k).S_2.(m_2,i,j) \urcorner = \ulcorner S_1 \urcorner.(m_1,j,k).(m_2,i,j) & \text{si } m_2 \text{ est un } O\text{-coup.}
\end{array}$$

De la même manière, la O-vue de S, notée $\llcorner S \lrcorner$ est définie inductivement sur la longueur de S comme :

$$\begin{array}{ll}
\llcorner (m,1,0) \lrcorner = (m,1,0) & \\
\llcorner S.(m,i,j) \lrcorner = \llcorner S \lrcorner.(m,i,j) & \text{si } m \text{ est un } O\text{-coup} \\
\llcorner S_1.(m_1,j,k).S_2.(m_2,i,j) \lrcorner = \llcorner S_1 \lrcorner.(m_1,j,k).(m_2,i,j) & \text{si } m_2 \text{ est un } P\text{-coup.}
\end{array}$$

Remarque 3.3.1.1. Étant donnée une séquence justifiée $S.(m,i,j)$, où m est un P-coup (*resp.* O-coup), sa P-vue (*resp.* sa O-vue) est notée $\ulcorner S \urcorner.(m,i,j)$ (*resp.* $\llcorner S \lrcorner.(m,i,j)$). Néanmoins, cette notation n'est pas cohérente puisque, d'une part, $\ulcorner S \urcorner.(m,i,j)$ (*resp.* $\llcorner S \lrcorner.(m,i,j)$) n'est pas forcément une séquence justifiée, et d'autre part, l'occurrence (m,i,j) n'apparaît pas forcément en position i dans $\ulcorner S \urcorner.(m,i,j)$ (*resp.* $\llcorner S \lrcorner.(m,i,j)$). Nous verrons que dans le cas spécifique des stratégies justifiées que nous utiliserons, cette notation reste cohérente.

Étant donnée une séquence bien formée $S.(m,i,j)$ où m est un P-coup , l'occurrence qui justifie (m,i,j) n'est pas forcément présente dans la P-vue de $S.(m,i,j)$. Dans le cadre des jeux de typage, nous introduisons des séquences bien formées particulières respectant cette condition :

Définition 3.3.1.3. *Soit une arène $A = (M,\tau)$. Une séquence bien formée S de coups sur A sera dite* innocente *si $\ulcorner S \urcorner = S$.*

Remarque 3.3.1.2. Cette condition implique que pour $S.(m,i,j_1).(n,i+1,j_2)$ où n est un O-coup, $j_2 = i$.

Notation 3.3.4. Nous simplifions la notation des occurrences de coups dans une séquence innocente S en (m,i), de telle sorte que :

- si $m \in M^O$, m est le i-ème O-coup joué dans la séquence S
- si $m \in M^P$, alors (m,i) est justifiée par l'occurrence (n,i) dans S, où $n \in M^O$.

Cette notation ne permet pas de distinguer différentes occurrences d'un P-coup justifiées par une même occurrence de O-coup. Nous verrons, au moment d'interpréter des stratégies comme des λ-termes, que ceci ne pose cependant pas de problèmes.

Étant donnée une arène de typage A, nous noterons l'ensemble de ses séquences innocentes par L_A. Pour une arène de typage $A = (M,\tau)$, le *jeu de typage* sur A est habituellement défini par $G = (A, L_A)$.

Un jeu est donc une arène sur laquelle nous rajoutons un certain nombre de séquences qui définissent les positions possibles dans l'arène. Ainsi, étant donnée une séquence de coups $S.(m,i)$ de longueur paire, le joueur O (le joueur suivant) doit jouer un coup n où $m \vdash n$. À l'inverse, pour $S.(n,j)$ de longueur impaire, le joueur P doit jouer un coup m justifié dans $S.(n,j)$; de plus, nous imposons $\tau(m) = \tau(n)$ dans le cas des séquences innocentes qui seront utilisées pour définir les jeux de typage.

Remarque 3.3.1.3. Contrairement aux notions de la littérature, la notion de jeu que nous donnons n'est pas usuelle, puisque nous contraignons les séquences à respecter des contraintes plus fortes que celles données habituellement. Ainsi, la condition d'innocence est traditionnellement imposée aux stratégies des joueurs, et non aux séquences du jeu. Néanmoins, en nous plaçant dans un cadre restreint d'utilisation des jeux, cette définition de séquences innocentes nous permettra de confondre jeux et arènes et ainsi de simplifier nos démonstrations. La notion de jeu se confondant avec celle d'arène dans notre cas, elle ne sera pas utilisée dans la suite de ce document.

3.3.2 Stratégies de typage

L'ensemble des séquences innocentes d'un jeu permet de représenter les parties possibles entre les joueurs O et P. Nous définissons à présent la notion de stratégie pour ces joueurs, en particulier pour le joueur P :

Définition 3.3.2.1. *Un ensemble fini Σ de séquences innocentes sur une arène de typage A est appelée une* P-stratégie *(ou plus simplement stratégie) sur A si :*

- *toute séquence $S \in \Sigma$ est de longueur paire et non-nulle.*
- *si $S.(m,i).(n,j)$ appartient à Σ, alors S appartient à Σ*
- *si $S.(m,i).(n_1,j_1)$ et $S.(m,i).(n_2,j_2)$ appartiennent à Σ, alors $n_1 = n_2$ et $j_1 = j_2$.*

Puisque une séquence $S = (m,1).S'$ bien formée vérifie la condition d'initialisation (*i.e.* $m = \epsilon$ est un O-coup) et d'après la première condition sur la définition d'une P-stratégie Σ, les séquences de Σ sont de la forme $S.S''.(m,i)$ où m est un P-coup. D'après la deuxième condition, une P-stratégie vérifie une propriété de clôture par préfixes de longueurs paires. Enfin, la troisième condition nous indique que, pour une séquence S dans une stratégie donnée Σ et une séquence $S.(n,i)$, un seul coup parmi l'ensemble des coups que P peut jouer est considéré dans la stratégie.

Nous exprimons à présent les conditions de victoire du joueur P par le fait que pour tout choix de O, la stratégie adoptée par P l'amène dans une séquence gagnante (*i.e.* où O ne peut plus jouer).

Définition 3.3.2.2. *Soit une arène de typage $A = (M,\tau)$. Une stratégie Σ sur A est dite* gagnante pour P *si :*

- ***(Séquence gagnante)** pour toute séquence $S \in \Sigma$, il existe une séquence $S'.(m,i)$ telle que $S.S'.(m,i) \in \Sigma$ et il n'existe pas de coup $n \in M^O$ tel que $m \vdash n$.*
- ***(Complétion des O-coups)** pour tout $S.(m,i) \in \Sigma$ et tout $n \in M^O$ tel que $m \vdash n$, il existe $m' \in M^P$ et $j,k \in \mathbb{N}$ tels que $S.(m,i).(n,j).(m',k) \in \Sigma$.*

D'après cette définition, il est possible d'identifier une stratégie gagnante par les séquences maximales (selon l'ordre $\sqsubseteq$) qu'elle contient. Ainsi, étant donnée une stratégie gagnante Σ, nous noterons par $max(\Sigma) \subseteq \Sigma$ l'ensemble de ces séquences.

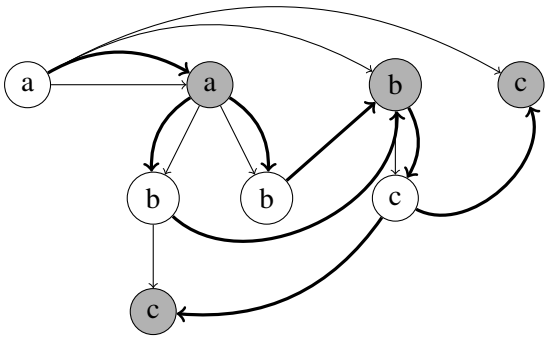


FIG. 3.4 – Exemple de stratégie de typage

En guise d'exemple, la Figure 3.4 représente une arène pour le typage $f : (c \to b) \to b \to a, g : c \to b, x : c \vdash a$, où les chemins dirigés faits de flèches épaisses, commençant par la racine de l'arène, et pour lesquels la justification est vérifiée, sont des séquences innocentes sur l'arène.

Exemple 3.3.2.1. Pour l'arène présentée en Figure 3.4, il existe une stratégie Σ gagnante pour le joueur P, définie par $max(\Sigma) = \{S_1, S_2\}$, où :

- $S_1 = (\epsilon,1).(\epsilon_f,1).(1_f,2).(\epsilon_g,1).(1_g,3).(11_f,2)$
- $S_2 = (\epsilon,1).(\epsilon_f,1).(2_f,2).(\epsilon_g,1).(1_g,3).(\epsilon_x,1)$

Nous remarquerons qu'une autre stratégie gagnante pour P est définie par $max(\Sigma') = \{S'_1, S_2\}$ où $S'_1 = (\epsilon,1).(\epsilon_f,1).(1_f,2).(\epsilon_g,1).(1_g,3).(\epsilon_x,1)$.

3.3.3 Préservation de stratégies gagnantes par substitutions de coups

Nous avons vu en Section 3.2.3 que nous pouvons transformer des arènes de typage grâce à des substitutions de coups. Il est donc naturel de se demander quel effet a l'application de ces fonctions sur les séquences et les stratégies de l'arène initiale.

Notation 3.3.5. Étant donnée un ré-étiquetage de coups F entre deux arènes A et B, et une séquence innocente $S = (m_1,0)\dots(m_n,i)$ de coups sur A, nous écrirons $F(S) = (F(m_1),0)\dots(F(m_n),i)$. Nous montrons à présent que cette notation se justifie puisque $F(S)$ est une séquence innocente.

Lemme 3.3.3.1. *Soient deux arènes $A_1 = (M_1,\tau_1)$ et $A_2 = (M_2,\tau_2)$ telles qu'il existe un ré-étiquetage de coups $F : A_1 \mapsto \mathcal{F}(A_2)$. Pour toute séquence innocente S sur A_1, $F(S)$ est une séquence innocente sur A_2.*

Démonstration. Soit une séquence innocente S sur A_1. Montrons dans un premier temps, que $F(S)$ est une séquence justifiée de coups de A_2. Puisque S est une séquence innocente sur A_1, nous savons que tout préfixe $S' = S_1.(n,j).S_2.(m,i)$ de S de longueur supérieure ou égale à 2 vérifie la condition de justification ; supposons ainsi que $(n,j) \vdash_S (m,i)$. La séquence $F(S') = F(S_1).(F(n),i).F(S_2).(F(m),i)$ reste justifiée dans A_2 puisque $F(n) \vdash F(m)$. De plus, la séquence $F(S)$ est bien formée puisqu'elle vérifie la condition d'initialisation (car $F(\epsilon) = \{\epsilon\}$) et la condition d'alternance (car $pl(F(m)) = pl(m)$).

Finalement, nous vérifions que pour S innocente, $F(S)$ est innocente, *c.a.d.* $\ulcorner F(S) \urcorner = F(S)$, par induction sur $|S|$. En effet

- si $S = (\epsilon,1)$, alors $\ulcorner F(S) \urcorner = \ulcorner (\epsilon,1) \urcorner = (\epsilon,1)$.
- si $S = S'.(m,i)$ où m est un coup appartenant à M_1^P, alors $\ulcorner S'.(m,i) \urcorner = \ulcorner S' \urcorner.(m,i)$ et, puisque $F(m) \in M_2^P$, $\ulcorner F(S'.(m,i)) \urcorner = \ulcorner F(S').(F(m,i)) \urcorner = \ulcorner F(S') \urcorner.(F(m),i)$. Grâce à l'hypothèse d'induction $\ulcorner F(S') \urcorner = F(S')$, donc $\ulcorner F(S) \urcorner = F(S)$
- si $S = S'.(m,i).(n,j)$ et $n \in M_1^O$, alors $F(S) = F(S').(F(m),i).(F(n),j)$ et, par préservation de la fonction pl et de la relation d'accessibilité par application du ré-étiquetage de coups, nous obtenons $(F(m),i) \vdash_{F(S)} (F(n),j)$, d'où $\ulcorner F(S) \urcorner = \ulcorner F(S') \urcorner.(F(m),i).(F(n),j)$; à nouveau, l'hypothèse d'induction nous donne $\ulcorner F(S) \urcorner = F(S)$.

□

Lemme 3.3.3.2. *Soient deux arènes A_1 et A_2 telles que $A_1 \cong A_2$. Il existe une stratégie gagnante sur A_1 si et seulement si il existe une stratégie gagnante sur A_2.*

Démonstration. Prenons $A_1 = (M_1,\tau_1)$ et $A_2 = (M_2,\tau_2)$ et le renommage de coups F de A_1 sur $\mathcal{F}(A_2)$. Supposons qu'il existe une stratégie gagnante Σ sur A_1 ; d'après le Lemme 3.3.3.1, pour toute séquence $S \in \Sigma$, $F(S)$ est une séquence innocente sur (M_2,τ_2). De plus, par définition nous obtenons que $m \vdash n$ ssi $F(m) \vdash F(n)$, ce qui implique que la stratégie $F(\Sigma) = \{F(S) \mid S \in \Sigma\}$ est gagnante sur A_2. L'inverse se prouve de la même manière, en considérant F^{-1}. □

3.3.4 Interprétation de stratégies comme λ-termes

Nous avons vu que les arènes que nous utilisons représentent des typages du λ-calcul simplement typés. À présent, nous montrons qu'une stratégie gagnante pour le joueur P correspond à un λ-terme sous forme normale, typable par le typage associée à l'arène. Cette correspondance bien connue est présentée de manière informelle dans [Felscher, 1986, Hughes, 2000, Hyland and Ong, 2000]. Nous donnons ici un algorithme permettant d'extraire un λ-terme sous forme β-réduite d'une stratégie, d'une manière similaire à, par exemple, la procédure réalisée sur les réseaux de preuve du calcul de Lambek [Moot, 1996, de Groote and Retoré, 1996].

Pour ce faire, nous exhibons la structure arborescente d'une stratégie : en effet, d'après la définition d'une P-stratégie, la condition de clôture par préfixes de longueur paire de Σ (point 2 de la définition 3.3.2.1), il est possible de représenter Σ comme un arbre noté $\mathbb{T}_\Sigma$.

Définition 3.3.4.1. *Étant donnés une arène $A = (M, \tau)$ et un coup $m \in M$, nous définissons, $\vdash_m = \{n \in M \mid m \vdash n\}$. On remarquera que pour $m \neq \epsilon$, $\vdash_m$ est ordonné par $\prec$.*

Définition 3.3.4.2. *Soit une arène de typage $A = (M, \tau)$ et une stratégie Σ sur A ;* l'arborescence de Σ, *notée $\mathbb{T}_\Sigma$ est définie inductivement sur un ensemble clos par préfixes de longueur paire de séquences alternées, comme $\mathbb{T}_\Sigma = (n,i).(m,j)[\mathbb{T}_{\Sigma_1} \dots \mathbb{T}_{\Sigma_p}]$ où :*

1. *toute séquence $S \in max(\Sigma)$ vérifie $(n,i).(m,j) \sqsubseteq S$ et*
2. *étant donné l'ensemble fini de coups $\vdash_m = \{n_1, \dots, n_p\}$, tel que $n_i \prec n_j$ ssi $i < j$ (où $i, j \in \{1, \dots, p\}$), pour tout $k \in \{1, \dots, p\}$, $\Sigma_k = \{(n_k, i+1).S \mid (n,i).(m,j).(n_k, i+1).S \in \Sigma\}$.*

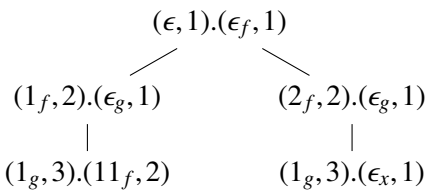


FIG. 3.5 – Exemple d'arborescence de stratégie

Ainsi, la stratégie donnée dans l'Exemple 3.3.2.1 a l'arborescence donnée en Figure 3.5 et nous la notons sous la forme :

$$\mathbb{T}_\Sigma = (\epsilon,1).(\epsilon_f,1)[(1_f,2).(\epsilon_g,1)[(1_g,3).(11_f,2)], (2_f,2).(\epsilon_g,1)[(1_g,3).(\epsilon_x,1)]]$$

Cette structure arborescente est en fait proche des arbres de Böhm (une notation alternative de λ-termes) pour des termes sous forme β-réduites, les entiers introduits dans la notation d'occurrence de coups indiquant les liens de référence vers les variables liées. Cette correspondance est plus précisément établie à travers l'algorithme d'interprétation suivant :

Notation 3.3.6. Pour une arène de typage donnée $A = (M, \tau)$, nous définissons l'ensemble associé $FV_A = \{((\epsilon_x, 1), x)\}$

Algorithme 3.3.4.1. *Soit une arène de typage A et une stratégie Σ sur A, gagnante pour P.* L'interprétation de Σ *est le λ-terme $[\![\Sigma]\!]_A = [\![\mathbb{T}_\Sigma, FV_A]\!]_A$ inductivement défini sur $\mathbb{T}_\Sigma$ par :*

$$[\![(n,i).(m,j)[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_q}], V]\!]_A = \lambda x_1 \dots x_p . x [\![\mathbb{T}_{\Sigma_1}, W]\!]_A \dots [\![\mathbb{T}_{\Sigma_q}, W]\!]_A$$

tel que, pour $\vdash_n = \{m_1, \dots, m_p\}$ (où $m_{k_1} \prec m_{k_2}$ ssi $k_1 < k_2$, pour $k_1, k_2 \in \{1, \dots, p\}$)

- *l'ensemble W est égal à $V \cup \{((m_k, i), x_k) \mid k \in \{1, \dots, p\}, x_k$ variable fraîche dans $W\}$,*
- *$((m, j), x)$ appartient à W.*

Dans cet algorithme, nous remarquons que les occurrences de variables du terme $[\![\Sigma]\!]_A$ sont générées par les occurrences de P-coups dans Σ. Nous dirons donc que l'occurrence $(C'[], x)$ de x dans $[\![\Sigma]\!]_A$ est une *réalisation d'une occurrence* (m, j) (ou d'un coup m) dans Σ quand $C'[] = C[\lambda x_1 \dots x_p.[]N_1 \dots N_p]$ et

$$[\![\Sigma]\!]_A = C[\lambda x_1 \dots x_p.xN_1 \dots N_p] = C[[\![(n, i).(m, j)[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_p}], V]\!]_A]$$

pour des ensembles V et W tels que définis précédemment, et où $((m, i), x)$ appartient à W.

Remarquons également que l'on impose aux variables introduites dans l'ensemble V d'être des variables fraîches pour cet ensemble afin d'éviter la capture de variable dans le terme $[\![\Sigma]\!]_A$. Il est également aisé de voir qu'il existe une correspondance unique entre une variable du terme $[\![\Sigma]\!]_A$ et une occurrence de P-coup dans la séquence Σ. Afin de pouvoir mettre cette relation en évidence, nous utiliserons régulièrement la convention de nommage de Barendregt, permettant d'attribuer un nom distinct à chaque variable d'un terme. De cette manière, nous pourrons parler directement de *la* variable x associée à une occurrence de P-coup donnée dans une certaine stratégie gagnante. Nous dirons alors que *x est la réalisation de l'occurrence (m, i) dans S*.

Nous avons vu au cours de la section précédente que, étant donné un typage, nous pouvons lui associer une arène de typage unique. Nous pouvons à présent établir le même type de correspondance entre stratégies et λ-termes.

Théorème 3.3.4.1. *Soient un typage $\Gamma \vdash \alpha$ et son arène de typage A. Il existe un λ-terme N tel que $\Gamma \vdash \alpha$ est un typage de N si et seulement il existe une stratégie gagnante Σ sur A vérifiant $[\![\Sigma]\!]_A =_{\beta\eta} N$.*

Démonstration. À partir d'un terme N sous forme normale, nous montrons l'existence d'une stratégie par induction sur N. Inversement, par induction sur l'arborescence d'une stratégie, nous montrons l'existence d'un terme N habitant le typage représenté par l'arène.

Voire section 3.5.1 pour une démonstration complète. □

3.3.5 Stratégies recouvrantes et principalité

Dans les sections qui précèdent, nous avons vu que les arènes correspondent aux typages, et que les stratégies sur ces arènes correspondent aux formes β-normales et η-longues (pour ces typages) des termes habitant ces typages. Or, parmi les typages d'un λ-terme, nous avons évoqué son typage principal, qui est un typage canonique de ce terme. Il existe essentiellement deux définitions du typage principal d'un terme N :

- Une définition extensionnelle, telle que nous l'avons donnée en Définition 2.2.1.9, et qui caractérise le typage principal comme étant celui à partir duquel tout typage de N peut être généré par application de substitutions de types et d'affaiblissements de l'environnement.
- Une définition intentionnelle donnée par l'algorithme d'inférence de type de Damas-Hindley-Milner [Damas and Milner, 1982] que nous ne développerons pas ici.

Nous donnons à présent une définition du typage principal d'un terme à travers les jeux que nous venons d'introduire. Cette nouvelle définition a pour principal avantage le fait d'être intentionnelle tout en permettant d'exhiber les contraintes structurelles nécessaires et suffisantes entre un terme et son typage principal. Néanmoins, et contrairement aux deux définitions citées ci-dessus, nous ne nous intéressons ici qu'au typage principal de λ-termes simplement typés sous forme β-réduite, et η-longue pour ce typage principal.

Afin de donner quelques intuitions sur les définitions à venir, nous pouvons voir l'assignation de types atomiques (ou l'étiquetage des coups d'une arène) comme des contraintes qui guide la construction d'une stratégie. Certaines de ces contraintes ne sont pas nécessaires puisque n'ayant aucune influence sur la façon dont la stratégie est construite ; dans ce cas, il est possible de reétiqueter l'arène, et ainsi d'obtenir une arène d'un typage moins général que celui de l'arène précédente. À partir de cette notion d'assignation de types comme contraintes, nous donnons les définitions suivantes :

Définition 3.3.5.1. *Soient une arène de typage $A = (M, \tau)$ et une stratégie de typage Σ sur A ; nous définissons les relations binaires d'*antécédence potentielle $\lhd$ *et de* Σ-antécédence $\lhd_\Sigma$ *sur $M^O \times M^P$ de la manière suivante :*

1. $n \lhd m$ *ssi* $\tau(m) = \tau(n)$
2. $n \lhd_\Sigma m$ *ssi il existe une séquence de coups $S \in \Sigma$, telle que $S = S'.(n,i).(m,j)$, où i et j sont des entiers positifs.*

Ainsi, étant donné un P-coup m, nous notons par $\lhd^m$ l'ensemble $\{n \in M^O \mid n \lhd m\}$ des *antécédents potentiels* de m ; de même, pour une stratégie Σ donnée, $\lhd_\Sigma^m$ désignera l'ensemble $\{n \in M^O \mid n \lhd_\Sigma m\}$ des Σ-*antécédents* de m. Nous remarquons que, par définition d'une stratégie de typage, l'ensemble $\lhd_\Sigma^m$ est un sous-ensemble de $\lhd^m$ pour tout P-coup m. Enfin, pour tous P-coups m_1 et m_2, $\tau(m_1) = \tau(m_2)$ est vérifié si et seulement si $\lhd^{m_1} = \lhd^{m_2}$.

Nous définissons à présent une relation de partage de Σ-antécédents entre P-coups.

Définition 3.3.5.2. *Soit une arène de typage $A = (M, \tau)$ et deux P-coups $m_1, m_2 \in M$. Nous définissons la relation $R_\Sigma \subseteq M^P \times M^P$ par $m_1 R_\Sigma m_2$ si et seulement si $\lhd_\Sigma^{m_1} \cap \lhd_\Sigma^{m_2} \neq \emptyset$*

Ainsi, tous P-coups m_1, m_2 tels que $m_1 R_\Sigma m_2$, ont au moins un Σ-antécédent commun $n \in M^O$. Par définition d'une stratégie de typage, ceci implique $\tau(m_1) = \tau(m_2) = \tau(n)$.

Nous noterons la clôture transitive de cette relation par R_Σ^* ; il s'agit d'une relation d'équivalence. Il en découle, que pour tout élément $m_1, m_2 \in M^P$ tels que $m_1 R_\Sigma^* m_2$, l'égalité $\tau(m_1) = \tau(m_2)$ est vraie.

Définition 3.3.5.3. *Étant donnée une arène de typage $A = (M, \tau)$, une stratégie gagnante Σ sur A est dite* recouvrante *si*

1. *Pour tout P-coup $\epsilon_x \in M^P$, où x est une variable, il existe une séquence $S.(\epsilon_x, 1)$ dans Σ* ***(ϵ-complétude)***
2. *Pour tout O-coup n de M, il existe une séquence $S.(n,i).(m,j)$ dans Σ telle que, soit $S \in \Sigma$, soit S est la séquence de longueur nulle* ***(complétude des O-coups)***
3. *Pour tout P-coup m_1, m_2 de M^P, $\tau(m_1) = \tau(m_2)$ ssi $m_1 R_\Sigma^* m_2$* ***(plus faible contrainte de typage)***

Nous pouvons remarquer la règle de plus faible contrainte de typage peut s'interpréter comme un problème de coloration ; en effet, considérons une stratégie Σ comme un ensemble de chemins sur les noeuds d'un arbres (qui est l'arène A) ; typer ce terme revient à colorer les noeuds du graphe ainsi défini, de sorte que pour tout sous-chemin nm de Σ tel que $n \in M^O$ et $m \in M^P$, n et m ont la même couleur. Le typage principal correspond au nombre maximal de couleurs pouvant être utilisées sur une arène vérifiant les propriétés 1 et 2.

Théorème 3.3.5.1. *([Bourreau and Salvati, 2011b])Étant donnés un terme N sous forme β-réduite, un typage $\Gamma \vdash \alpha$ et l'arène associée $A = (M, \tau)$. Les deux propriétés suivantes sont équivalentes*

1. *il existe une stratégie recouvrante Σ sur A telle que $N = [\![\Sigma]\!]_A$*
2. *$\Gamma \vdash \alpha$ est le typage principal de $N = [\![\Sigma]\!]_A$ et N est η-long pour $\Gamma \vdash \alpha$*

Démonstration. Nous démontrons qu'il n'existe pas de stratégie recouvrante Σ sur A vérifiant $N = [\![\Sigma]\!]_A$ ssi $\Gamma \vdash \alpha$ n'est pas le typage principal de N.

Si la stratégie Σ telle que $N = [\![\Sigma]\!]_A$ n'est pas recouvrante, nous montrons que la non-validité de chaque point de la définition d'une telle stratégie recouvrante contredit la définition d'un typage principal d'un terme.

Si $\Gamma \vdash \alpha$ n'est pas le typage principal de N, alors nous montrons que Σ ne peut être recouvrante en considérant un typage $\Gamma' \vdash \alpha'$ comme le typage principal de N, et la substitution de coups F_σ telle que $\alpha = \alpha' \cdot \sigma$ et $\Gamma \subseteq \Gamma' \cdot \sigma$.

Voire section 3.5.2 pour une démonstration complète. □

3.4 Conclusion

Au cours de ce chapitre, nous avons introduit les arènes de typage, comme une formalisation alternative de la notion de typage ; sur ces arènes, des stratégies sont définies, et certaines stratégies vérifiant des conditions de victoires pour le joueur P sont interprétables comme des λ-termes sous forme normale habitant le typage associé à l'arène. Nous pouvons donc compléter l'analogie entre logique minimale et λ-calcul simplement typé de la manière suivante :

Logique minimale	λ-calcul simplement typé	Sémantique des jeux
Séquent	Typage	Arène
Dérivation sans détour	λ-terme β-normal	Stratégie gagnante
Occurrence de formule atomique	Occurrence de type atomique	Coup
Polarité	Polarité	Fonction pl

Cette notion de jeu sera utilisée afin d'étudier les typages ayant une unique dérivation dans la logique minimale (de manière équivalente, les termes qui sont les uniques habitants sous forme normale de certains typages, et en particulier, de leurs typages principaux, pour lesquels nous avons défini la notion de stratégie recouvrante). En effet, de tels résultats sont liés à la notion de polarité, notion aisément manipulable dans la sémantique des jeux.

3.5 Annexes

3.5.1 Démonstration du Théorème 3.3.4.1

Notation 3.5.1. Étant données deux arènes $A_1 = (M_1, \tau_1)$ et $A_2 = (M_2, \tau_2)$ et une fonction $f = M_1 \mapsto M_2$ préservant l'accessibilité et la fonction d'assignation de joueurs :

- pour $S = (m_1, 1).\ldots.(m_n, k)$ une séquence de coups sur A_1, nous noterons par $f(S)$ la séquence $(f(m_1), 1)\ldots(f(m_n), k)$.
- pour une stratégie Σ sur A_1, nous noterons par $f(\Sigma)$ l'ensemble $\{f(S) \mid S \in \Sigma\}$.

Lemme 3.5.1.1. *Soit l'arène $A = (M, \tau)$ associée à un typage $\Gamma, x : \alpha_1 \to \ldots \to \alpha_n \to a \vdash a$. Pour tout $i \in \{1, \ldots, n\}$, nous considérons l'arène $A_i = (M_i, \tau_i)$ associée à $\Gamma, x : \alpha_1 \to \ldots \to \alpha_n \to a \vdash \alpha_i$.*

Si il existe une stratégie gagnante Σ sur A, telle que $\mathbb{T}_\Sigma = (\epsilon, 1).(m, 1).\mathbb{T}_{\Sigma_1} \ldots \mathbb{T}_{\Sigma_n}$ et vérifiant $[\![\Sigma]\!]_A = xN_1 \ldots N_n$, alors il existe une stratégie gagnante Σ'_i sur A_i et une fonction $f : M_i \mapsto M$ telle que $[\![\Sigma'_i]\!]_{A_i} = N_i$ et $\Sigma'_i = f(\Sigma_i)$

Démonstration. Nous construisons la fonction $f : M_i \mapsto M$ de sorte que $f(s) = (i \cdot s)_x$, pour tout $s \in M_i \cap \mathbb{N}^*$ et $f(m) = m$ sinon. On remarquera que $pl(f(m)) = pl(m)$ et que $m \vdash n \Rightarrow f(m) \vdash f(n)$ pour tout $m, n \in M_i$.

On suppose l'existence d'une stratégie gagnante Σ sur A, telle que $\mathbb{T}_\Sigma = (\epsilon, 1).(m, 1).[\mathbb{T}_{\Sigma_1} \dots \mathbb{T}_{\Sigma_n}]$, vérifiant $[\![\Sigma]\!]_A = xN_1 \dots N_n$. Alors

$$[\![\mathbb{T}_\Sigma, FV_A]\!]_A = x[\![\mathbb{T}_{\Sigma_1}, V']\!]_A \dots [\![\mathbb{T}_{\Sigma_n}, V']\!]_A$$

et $V' = FV_A$, car sinon, $[\![\Sigma]\!]_A$ serait de la forme $\lambda x_1 \dots x_p.xN_1 \dots N_n$, où $p > 0$. Donc, $[\![\mathbb{T}_{\Sigma_i}, FV_A]\!]_A = N_i$. La propriété se montre par induction sur $\mathbb{T}_{\Sigma_i} = (n_i, 2).(m_i, k)[\mathbb{T}_{\Sigma_{i1}} \dots \mathbb{T}_{\Sigma_{ip}}]$. Tout d'abord, $n_i = i_x$ par construction. Nous construisons donc $\Sigma'_i = (\epsilon, 0).(m_i, 0)[\mathbb{T}_{\Sigma'_{i1}} \dots \mathbb{T}_{\Sigma'_{ip}}]$, où $f(\epsilon) = i_x$ et $f(m_i) = s_x$ si $m_i = (i \cdot s)_x$ et $f(m_i) = m_i$ sinon. Alors, nous pouvons construire un ensemble V' tel que

$$[\![\Sigma'_i]\!]_{A_i} = \lambda \overline{x}.x'[\![\mathbb{T}_{\Sigma'_{i1}, V'}]\!]_{A_i} \dots [\![\mathbb{T}_{\Sigma'_{ip}, V'}]\!]_{A_i}$$

et $N_i = \lambda \overline{x}.x'N'_1 \dots N'_p$. D'après l'hypothèse d'induction sur les arbres $\mathbb{T}_{\Sigma_{i1}}, \dots, \mathbb{T}_{\Sigma_{ip}}$, nous obtenons que $[\![\Sigma'_i]\!]_{A_i} = [\![\mathbb{T}_{\Sigma_i}, FV_A]\!]_A$ et que $f(\Sigma'_i) = \Sigma$. □

Lemme 3.5.1.2. *Étant données l'arène de typage A associée à un typage $\Gamma \vdash \alpha$, et une stratégie gagnante Σ sur A, $\Gamma \vdash \alpha$ est un typage de $[\![\Sigma]\!]_A$.*

Démonstration. Soit $A = (M, \tau)$. La preuve se fait par induction sur l'arbre $\mathbb{T}_\Sigma$:

- si $\mathbb{T}_\Sigma = (\epsilon, 1)(m, 1)$, alors $[\![\Sigma]\!]_A = \lambda x_1, \dots x_n.x$. Posons $\tau(m) = a \in \mathcal{A}$. Tout d'abord, si il existe une variable x pour laquelle $m = \epsilon_x$, alors par construction de $[\![\Sigma]\!]_A$, l'arène A est associée à un typage de la forme $\Gamma, x : a \vdash \alpha_1 \to \dots \to \alpha_n \to a$ qui est un typage correcte pour $[\![\Sigma]\!]_A$. Par ailleurs, si il existe $i \in \mathbb{N}$ pour lequel $m = i$, alors $i \in \{1, \dots, n\}$, $x = x_i$ et le typage associée à A est de la forme $\Gamma \vdash \alpha_1 \to \dots \to \alpha_{i-1} \to a \to \dots \to \alpha_n \to a$, qui est alors un typage correcte pour $[\![\Sigma]\!]_A$.
- si $\mathbb{T}_\Sigma = (\epsilon, 1)(m, 1)[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_p}]$, alors

  $$[\![\mathbb{T}_\Sigma, FV_A]\!]_A = \lambda x_1 \dots x_n.x[\![\mathbb{T}_{\Sigma_1}, FV_A \cup V]\!]_A \dots [\![\mathbb{T}_{\Sigma_p}, FV_A \cup V]\!]_A$$

  où $V = \{((i, 1), x_i) \mid i \in \mathbb{N} \text{ et } m = i \in M\}$. Étant donné $\Gamma \vdash \alpha_1 \to \dots \to \alpha_n \to a$ le typage associé à l'arène A, $FV_A \cup V$ correspond donc aux variables présentes dans l'environnement de typage $\Gamma, x_1 : \alpha_1, \dots, x_n : \alpha_n$; de plus, cet environnement de typage doit contenir une assignation de type $x : \gamma_1 \to \dots \to \gamma_p \to a$, tel que pour tout $j \in \{1, \dots, p\}$, γ_j est le type associé à la sous-arène de A faite de l'ensemble de coups qui ont $m \cdot j$ comme préfixe commun. Alors, pour tout $j \in \{1, \dots, p\}$, considérons l'arène $A_j = (M_j, \tau_j)$ associée à $\Gamma, x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash \gamma_j$. D'après le lemme 3.5.1.1, il existe une stratégie Σ'_j gagnante sur A_j telle que $[\![\Sigma'_j]\!]_{A_j} = [\![\mathbb{T}_{\Sigma_j}, FV_A \cup V]\!]_A$ et une fonction $f : M_j \mapsto M$ telle que $f(\Sigma'_j) = \Sigma_j$; d'après l'hypothèse d'induction, $\Gamma, x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash \gamma_j$ est un typage de $[\![\mathbb{T}_{\Sigma_j}, FV_A \cup V]\!]_A$. Nous obtenons finalement que $\Gamma \vdash \alpha_1 \to \dots \to \alpha_n \to a$ est un typage de $[\![\Sigma]\!]_A$, en appliquant les règles d'application et d'abstraction.

  □

Lemme 3.5.1.3. *Soient un typage $\Gamma \vdash \alpha$, son arène de typage A et un λ-terme N tel que $\Gamma \vdash \alpha$ soit un typage de N ; il existe alors une stratégie gagnante Σ sur A vérifiant $[\![\Sigma]\!]_A =_{\beta\eta} N$.*

Démonstration. Sans perte de généralité, nous supposons que N est sous forme β-réduite et η-longue par rapport à $\Gamma \vdash \alpha$. Nous construisons Σ par induction sur N

- si $N = x$: tout typage pour lequel N est sous forme η-longue est donc de la forme $\Gamma, x : a \vdash a$, où $a \in \mathcal{A}$. L'arène associée $A = (M, \tau)$ vérifie donc l'existence de deux coups $m_1 = \epsilon$ et $m_2 = \epsilon_x$ tels que $\tau(m_1) = \tau(m_2)$. De plus, pour tout $s \in \mathbb{N}^* - \{\epsilon\}$, il n'existe pas de coups s ou s_x dans M. Nous obtenons donc une stratégie $\Sigma = \{(\epsilon, 1).(\epsilon_x, 1)\}$ gagnante sur A et qui vérifie $[\![\Sigma]\!]_A = x$.
- si $N = \lambda x.N'$: considérons l'arène $A = (M, \tau)$ associée à un typage $\Gamma \vdash \alpha \to \beta$ pour lequel N est η-long. Soit $\mathtt{abs}_x$ le renommage de coups défini en section 3.2.3 ; alors $\mathtt{abs}_x^{-1}(A) = A'$ est l'arène associée à $\Gamma, x : \alpha \vdash \beta$, qui est un typage pour N'. D'après l'hypothèse d'induction, il existe une stratégie Σ' gagnante sur A et qui vérifie $[\![\Sigma']\!]_A = N'$. D'après le Lemme 3.3.3.2, $\Sigma = \mathtt{abs}_x(\Sigma')$ est une stratégie gagnante sur A. Finalement, étant donné $\mathbb{T}_{\Sigma'}$ sous sa forme générale $(\epsilon, 1).(m, 1)[\mathbb{T}_{\Sigma'_1}, \dots, \mathbb{T}_{\Sigma'_n}]$, nous obtenons

  $$[\![\mathbb{T}_{\Sigma'}, FV_{A'}]\!]_{A'} = \lambda x_1 \dots \lambda x_n.y[\![\mathbb{T}_{\Sigma'_1}, FV_{A'} \cup V']\!]_{A'} \dots [\![\mathbb{T}_{\Sigma'_n}, FV_{A'} \cup V']\!]_{A'} = N'$$

  où $V' = \{((i, 1), i) \mid i \in \mathbb{N} \cap M'\}$, et l'interprétation suivante de Σ sur A :

  $$[\![\mathbb{T}_{\Sigma}, FV_A]\!]_A = \lambda x \lambda x_1 \dots \lambda x_n.y[\![\mathbb{T}_{\mathtt{abs}_x(\Sigma'_1)}, FV_A \cup V]\!]_A \dots [\![\mathbb{T}_{\mathtt{abs}_x(\Sigma'_n)}, FV_A \cup V]\!]_A$$

  où V est analogue à V' dans A. Alors $[\![\Sigma]\!]_A$ est égal à $\lambda x.N'$ pour $FV_A \cup V = FV_{A'} \cup V'$.
- si $N = xN_1 \dots N_n$: soit $\Gamma, x : \alpha_1 \to \dots \to \alpha_n \to a \vdash a$ un typage pour lequel N est η-long. Alors, pour tout $i \in \{1, \dots, n\}$, $\Gamma \vdash \alpha_i$ est un typage du terme N_i ; de plus N_i est sous forme η-longue pour ce typage. Posons pour chaque $i \in \{1, \dots, n\}$, A_i l'arène de typage associée à $\Gamma, x : \alpha_1 \to \dots \to \alpha_n \to a \vdash \alpha_i$; l'hypothèse d'induction donne l'existence d'une stratégie Σ_i sur A_i, telle que $N_i = [\![\Sigma_i]\!]_{A_i}$. Nous construisons à présent un renommage des coups $\mathtt{app} : M_1 + \dots + M_n \to M$ (où $M_1 + \dots + M_n$ désigne l'union disjointe de $M_1, \dots, M_n$) défini de la manière suivante :
  – si $m \in M_i$ pour un certain $i \in \{1, \dots, n\}$ et si il existe $s \in \mathbb{N}^*$ tel que $m = s$, alors $\mathtt{app}(m) = (i \cdot s)_x$
  – sinon $\mathtt{app}(m) = m$.
  Nous définissons à présent une nouvelle arène de typage $A = (M, \tau)$ telle que $M = \mathtt{app}(M_1 + \dots + M_n) \cup \{\epsilon\}$ et où la fonction de typage τ vérifie $\tau(\epsilon) = \tau(\epsilon_x)$ et, pour tout $i \in \{1, \dots, n\}$, $\tau(\mathtt{app}(m)) = \tau_i(m)$, où $m \in M_i$. Alors A est l'arène associée au typage $\Gamma, x : \alpha_1 \to \dots \to \alpha_n \to a \vdash a$ puisque chaque coup de la forme $s_x \in M_i$ est préservé dans M (Γ n'est pas modifié) et $\tau(\epsilon) = \tau(\epsilon_x)$. Nous prenons à présent la stratégie Σ sur A, telle que

  $$\mathbb{T}_\Sigma = (\epsilon, 1).(\epsilon_x, 1)[\mathtt{app}([\![\Sigma_1]\!]_A), \dots, \mathtt{app}([\![\Sigma_n]\!]_A)]$$

  Alors,

  $$[\![\mathbb{T}_\Sigma, FV_A]\!]_A = x[\![(\mathtt{app}(\mathbb{T}_{\Sigma_1}), FV_A)]\!]_A \dots [\![\mathtt{app}(\mathbb{T}_{\Sigma_n}), FV_A]\!]_A$$

  Puisque $\mathtt{app}$ ne renomme pas les coups de la forme ϵ_y, pour tout $i \in \{1, \dots, n\}$, le terme N_i vérifie $N_i = [\![(\mathtt{app}(\mathbb{T}_{\Sigma_i}), FV_{A_i}]\!]_A$. Ainsi $[\![\Sigma]\!]_A = xN_1 \dots N_n$.

□

Corollaire 3.5.1.1. *Soient un typage $\Gamma \vdash \alpha$, son arène de typage A. Il existe un λ-terme N tel que $\Gamma \vdash \alpha$ est un typage de N si et seulement il existe une stratégie gagnante Σ sur A vérifiant $[\![\Sigma]\!]_A =_{\beta\eta} N$.*

3.5.2 Démonstration du Théorème 3.3.5.1

Étant données deux stratégies Σ et Σ', nous dirons que ces stratégies sont égales (noté $\Sigma = \Sigma'$) si toute séquence de Σ est une séquence de Σ', et inversement.

Lemme 3.5.2.1. *Soit un λ-terme N sous forme η-longue pour les typages $\Gamma \vdash \alpha$ et $\Gamma' \vdash \alpha$, A et A' les arènes respectivement associées à ces typages, et Σ et Σ' les stratégies sur A et A' respectivement, telles que $[\![\Sigma]\!]_A =_\beta [\![\Sigma']\!]_{A'} =_\beta N$. Alors $\Sigma = \Sigma'$.*

Démonstration. La preuve se fait par induction sur l'arborescence de Σ. Celle-ci est de la forme générale : $(\epsilon, 1).(m, 1).[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_{n_1}}]$. De même, nous notons l'arborescence de Σ' par sa forme générale $(\epsilon, 1).(m', 1).[\mathbb{T}_{\Sigma'_1}, \dots, \mathbb{T}_{\Sigma'_{n_2}}]$. Supposons que N est sous forme β-normale, sans perte de généralité ; alors

$$N = \begin{cases} [\![\Sigma]\!]_A = \lambda x_1 \dots x_{p_1}.x[\![\mathbb{T}_{\Sigma_1}, FV_A \cup V]\!]_A \dots [\![\mathbb{T}_{\Sigma_{n_1}}, FV_A \cup V]\!]_A \\ [\![\Sigma']\!]_{A'} = \lambda x_1 \dots x_{p_2}.x'[\![\mathbb{T}_{\Sigma'_1}, FV_{A'} \cup V']\!]_{A'} \dots [\![\mathbb{T}_{\Sigma'_{n_2}}, FV_{A'} \cup V']\!]_{A'} \end{cases}$$

La terme N étant sous forme β-normale, il est de la forme $\lambda x_1 \dots x_p.yN_1 \dots N_n$. Nous obtenons donc $p_1 = p_2 = p$, ce qui implique que nous pouvons également construire V et V' de telle sorte que $V = V'$. Ainsi, nous obtenons que x et x' sont syntaxiquement égaux, et donc $m = m'$. De plus, $n_1 = n_2 = n$ et, par hypothèse d'induction, $\Sigma_i = \Sigma'_i$ pour tout $i \in \{1, \dots, n\}$ ce qui implique finalement que $\Sigma = \Sigma'$. □

Lemme 3.5.2.2. *Soient un typage $\Gamma \vdash \alpha$, son arène associée $A = (M, \tau)$ et une stratégie gagnante Σ sur A telle que $[\![\Sigma]\!]_A = N$. Si $\Gamma \vdash \alpha$ n'est pas le typage principal de N, alors Σ n'est pas une stratégie recouvrante sur A.*

Démonstration. D'après le Théorème 2.2.1.1 il existe un typage principal $\Gamma' \vdash \alpha'$ pour le terme N. On considère l'arène $A' = (M', \tau')$ associée à ce typage, et Σ' la stratégie de typage sur A' telle que $[\![\Sigma']\!]_{A'} = N$. D'après le Lemme 3.5.2.1,

$$\Sigma = \Sigma' \tag{3.1}$$

Par hypothèse, $\Gamma \vdash \alpha$ n'est pas le typage principal de N ; une des trois conditions suivantes est donc vérifiée sur $\Gamma \vdash \alpha$.

1. Il existe une assignation de type $x : \gamma$ dans Γ telle que x n'est pas une variable libre de N. Il existe donc un coup ϵ_x dans M, qui n'apparaît sur aucune séquence de Σ (autrement, x serait une variable libre de N). Ainsi, Σ n'est pas une stratégie recouvrante car la condition de ϵ-complétude n'est pas respectée.
2. Supposons à présent l'existence d'une substitution de types σ, telle que $\Gamma \vdash \alpha = \Gamma' \cdot \sigma \vdash \alpha \cdot \sigma$. Nous supposons de plus que σ contient une substitution de la forme $[a \mapsto \alpha]$, où α est un type non-atomique. Il existe donc une substitution de coups correspondant à σ et que nous noterons $F_\sigma : A' \mapsto \mathcal{F}(A)$. Nous supposons alors l'existence d'un coup $m \in M'$ tel que $\tau(m) = a$; puisque α n'est pas atomique, il existe au moins un coup $n \in F_\sigma(m)$ tel que $\widehat{F_\sigma(m)} \vdash n$:
   - si $\widehat{F_\sigma(m)}$ est un O-coup. Dans un premier temps, si il n'existe pas de séquence S dans Σ qui contienne une occurrence de m, la condition de complétude pour les O-coups n'est pas vérifiée et Σ ne peut donc être une stratégie recouvrante sur A. Nous supposons donc que :

$$\begin{aligned} [\![\Sigma]\!]_A &= C[[\![(\widehat{F_\sigma(m)}, j).(m', k)[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_n}], V]\!]_A] \\ &= C[\lambda x_1 \dots x_p.y[\![\mathbb{T}_{\Sigma 1}, V']\!]_A \dots [\![\mathbb{T}_{\Sigma n}, V']\!]_A] \end{aligned}$$

     pour j et k des entiers positifs. Puisque il existe un P-coup n tel que $\widehat{F_\sigma(m)} \vdash n$, par définition de l'interprétation de Σ, nous obtenons $p \geq 1$. De plus, il existe $i \in \{1, \dots, p\}$ tel que $((n, j), x_i)$ appartient à V'. Mais n n'étant pas un coup de l'arène A' l'interprétation de Σ dans A' vérifie :

$$\begin{aligned} [\![\Sigma]\!]_{A'} &= C[[\![(\widehat{F_\sigma(m)}, j).(m', k).[\mathbb{T}_{\Sigma_1}, \dots, \mathbb{T}_{\Sigma_n}], V]\!]_{A'}] \\ &= C[y[\![\mathbb{T}_{\Sigma 1}, V']\!]_{A'} \dots [\![\mathbb{T}_{\Sigma n}, V']\!]_{A'}] \end{aligned}$$

     ce qui implique $[\![\Sigma]\!]_{A'} \neq N$ et nous obtenons une contradiction vis-à-vis de 3.1.

- si $\widehat{F_\sigma(m)}$ est un P-coup. Tout d'abord, supposons qu'il existe une séquence de la forme $S.(n,j).(m,i)$ dans Σ, ce qui implique que

$$\begin{aligned}[\![\Sigma]\!]_A &= C[[\![(n,j).(\widehat{F_\sigma(m)},j)[\mathbb{T}_{\Sigma_1}\dots\mathbb{T}_{\Sigma_p}],V]\!]_A] \\ &= C[\lambda x_1\dots x_n.xN_1\dots N_p]\end{aligned}$$

où $p>0$ et $n\geq 0$. Mais $[\![(n,j).(\widehat{F_\sigma(m)},j)\mathbb{T}_{\Sigma_1}\dots\mathbb{T}_{\Sigma_p},V]\!]_{A'}$ est égal à $\lambda x_1\dots x_n.xN'_1\dots N'_q$ tel que $q<p$ et nous obtenons à nouveau une contradiction avec 3.1. Si par ailleurs, il n'existe d'occurrences de m dans aucune séquence de Σ, alors il n'y a pas de séquence de la forme $S.(m,i).(n,j).S'$ dans Σ sur A et donc n est un O-coup pour lequel il n'existe aucune occurrence dans les séquences de Σ ; Σ n'est donc pas une stratégie recouvrante car elle ne vérifie pas la condition de complétude d'O-coups.

3. nous supposons enfin que toute substitution dans σ est de la forme $[a\mapsto b]$, où a et b sont des types atomiques. La substitution de coups F_σ associé à σ est donc un ré-étiquetage de coups (en effet, $F_\sigma(m)$ est restreint à un singleton pour tout coup m de M'). Supposons que σ ne soit pas un renommage : il existe donc deux types atomiques a_1 et a_2, distincts l'un de l'autre, tels que $\sigma(a_1)=\sigma(a_2)=b$. Nous considérons deux coups $m_1,m_2\in M'$ tels que $\tau'(m_1)=a_1$ et $\tau'(m_2)=a_2$
   - si m_1,m_2 sont des O-coups de l'arène A'. Nous pouvons supposer que les trois premières conditions de la définition 3.3.5.3 sont vérifiées pour la stratégie Σ sur A. Ceci assure l'existence de séquences $S_1.(m_1,i_1).(n_1,j_1)$ et $S_2.(m_2,i_2).(n_2,j_2)$ dans Σ. Par définition d'une stratégie de typage, tout P-coup n tel que $F_\sigma(m_i)$ appartient à $\lhd_\Sigma^{F_\sigma(n)}$ (pour $i=\{1,2\}$) doit vérifier $\tau(F_\sigma(n))=\tau(F_\sigma(m_i))=b$. Si la condition de plus faible contrainte de typage est vérifiée, alors la relation $F_\sigma(n_1)R_\Sigma^*F_\sigma(n_2)$ est vraie. Supposons simplement que $F_\sigma(n_1)R_\Sigma F_\sigma(n_2)$ (si ce n'est pas le cas, la démonstration reste vraie par transitivité) ; alors il existe un coup m tel que $F_\sigma(m)\in\lhd_\Sigma^{n_1}\cap\lhd_\Sigma^{n_2}$ et tel que n_1 et n_2 vérifient $\tau'(n_1)=\tau'(n_2)$ sur A', puisque $[\![\Sigma]\!]_{A'}=N$. Ce résultat contredit l'hypothèse $a_1\neq a_2$.
   - si m_1 est un P-coup de l'arène A'. Alors, si $F_\sigma(m_1)$ et $F_\sigma(m_2)$ ont des occurrences dans certaines séquences de Σ, le problème se réduit au cas précédent. Sinon, nous supposons à nouveau que les trois premières conditions de la Définition 3.3.5.3 sont vérifiées pour Σ sur A, et nous rajoutons l'hypothèse qu'il n'existe pas de séquence de la forme $F_\sigma(S_1).F_\sigma(m_1)$ dans Σ. Si la condition de plus faible contrainte de typage est vérifiée, alors nous obtenons $\lhd_\Sigma^{F_\sigma(m_1)}=\emptyset$, ce qui implique $\tau(F_\sigma(m_1))\neq\tau(F_\sigma(m))$, pour tout coup $m\neq m_1$ de M, et en particulier pour m_2, si m_2 est un P-coup ; si m_2 est un O-coup, il existe un P-coup m vérifiant $m_2\in\lhd_\Sigma^m$ et sur lequel le même raisonnement s'applique.

□

Lemme 3.5.2.3. *Soient un typage $\Gamma\vdash\alpha$, l'arène associée A et une stratégie gagnante Σ sur A telle que $[\![\Sigma]\!]_A=N$. Si Σ n'est pas recouvrante, alors $\Gamma\vdash\alpha$ n'est pas le typage principal de N.*

Démonstration. Supposons que Σ ne soit pas une stratégie recouvrante sur A :

- si Σ n'est pas une stratégie gagnante, $\Gamma\vdash\alpha$ ne peut être un typage de N d'après le Corollaire 3.3.4.1.
- si la condition de ϵ-complétude n'est pas vérifiée, le domaine de Γ est plus grand que $FV(N)$.
- si la condition de complétude pour les O-coups n'est pas vérifiée, il existe un O-coup n pour lequel il n'existe aucune occurrence dans les séquences de Σ. Ce O-coup ne peut être ϵ. De plus, étant donné le P-coup m tel que $m\vdash n$, le coup m ne peut pas avoir d'occurrences dans les séquences de Σ ; autrement, n et tout O-coup n' tel que $m\vdash n'$ aurait au moins une occurrence

dans les séquences de Σ, par la définition d'une stratégie gagnante. Soit l'ensemble de coups, clos par préfixes, suivant : $E = \{m \cdot s \in M \mid s \in \mathbb{N}^*\}$. Alors (E, τ) est une sous-arène de A, et, étant donné $\sigma_A((E,\tau)) = \alpha$ le type associé à cette sous-arène, nous construisons un nouveau typage $\Delta \vdash \beta$ pour lequel l'arène associée est obtenue en remplaçant E par le coup m, étiqueté par un type atomique frais a. Alors Σ reste une stratégie sur cette nouvelle arène A', et vérifie $[\![\Sigma]\!]_{A'} = N$ (Lemme 3.5.2.1). Le typage $\Delta \vdash \beta$ ainsi obtenu vérifie $\Delta \cdot [a \mapsto \alpha] \vdash \beta \cdot [a \mapsto \alpha] = \Gamma \vdash \alpha$, et $\Gamma \vdash \alpha$ ne peut donc être le typage principal de N.

- si la plus faible contrainte de typage n'est pas vérifiée, il existe alors deux P-coups m_1 et m_2 dans M^P tels que $\tau(m_1) = \tau(m_2) = a$, mais pour lesquels $m_1 R_\Sigma^* m_2$ n'est pas vérifié. Pour $i \in \{1,2\}$, nous considérons les ensembles

$$E_i = \{m \in M^P \mid m_i R_\Sigma^* m\} \cup \{n \in \lhd_\Sigma^m \mid m_i R_\Sigma^* m\}$$

Alors $E_1 \cap E_2 = \emptyset$ est vrai et nous construisons une nouvelle arène de typage (M, τ') qui diffère de A en étiquetant les coups qui appartiennent à E_1 par un type atomique frais a_1, et les coups de A_2 par un type atomique frais a_2. Le typage $\Delta \vdash \beta$ de N associé à l'arène ainsi construite vérifie $\Gamma \vdash \alpha = \Delta \cdot [a_1 \mapsto a, a_2 \mapsto a] \vdash \beta \cdot [a_1 \mapsto a, a_2 \mapsto a]$, et $\Gamma \vdash \alpha$ ne peut donc être le typage principal de N.

□

Théorème 3.5.2.1. *Étant donnés un terme N sous forme β-réduite, un typage $\Gamma \vdash \alpha$ et l'arène associée $A = (M, \tau)$. Les deux propriétés suivantes sont équivalentes*

1. *il existe une stratégie recouvrante Σ sur A telle que $N = [\![\Sigma]\!]_A$*
2. *$\Gamma \vdash \alpha$ est le typage principal de $N = [\![\Sigma]\!]_A$ et N est η-long pour $\Gamma \vdash \alpha$*

Démonstration. L'équivalence entre les deux affirmations suivantes :

1'. Σ n'est pas une stratégie recouvrante sur A

2'. $\Gamma \vdash \alpha$ n'est pas le typage principal de N

est donnée par les Lemmes 3.5.2.2 et 3.5.2.3. □

Deuxième partie

Linguistique formelle, λ-calcul et reconnaissance

Chapitre 4

Linguistique formelle et grammaires de λ-termes

Comme nous l'avons commenté en préambule à ce document, la question du traitement du langage naturel par un ordinateur et une question qui est apparue rapidement après l'avènement de l'informatique. Les machines sont capables de traiter l'information sous une certaine forme, et sont en particulier capable de traiter certains langages, tels que les langages de programmation. Par la suite, nous appellerons le langage humain, *langage naturel* afin de le dissocier de tout système de communication (*langage*), et en particulier des *langages informatiques* ou langages de programmation. Nous allons à présent montrer que de nombreux modèles introduits en informatique théorique permettent d'étudier et de modéliser le langage naturel, afin de traiter ce dernier de manière automatique ou algorithmique. En particulier, la théorie des langages formels, utilisée dans des domaines aussi vastes que la compilation ou la vérification de programmes, peut se révéler être utile à la description de phénomènes syntaxiques des langues naturels.

Au cours de ce chapitre, nous nous intéresserons à deux aspects du langage naturel : la syntaxe et la sémantique, ainsi qu'à l'interface entre ces deux aspects. Nous considérerons les phrases comme les unités les plus larges, et nous ne proposerons donc pas d'analyses pour les structures discursives, plus larges.

Afin d'introduire le formalisme des grammaires catégorielles abstraites, nous ferons quelques rappels sur le syntaxe et la sémantique formelles, afin de montrer comme les ACGs se situent dans le panorama des formalismes existants. Dans un premier temps, nous nous intéresserons à la syntaxe du langage naturel. Nous introduirons ainsi les notions de grammaires génératives de la théorie des langages formels, faisant apparaître la hiérarchie de Chomsky. À partir de cette hiérarchie, nous nous intéresserons à la famille des langages faiblement sensibles au contexte ; les langages de cette famille sont en effet considérés comme les plus adéquats pour modéliser les phénomènes syntaxiques du langage naturel. Nous définirons certains formalismes grammaticaux largement utilisés dans la modélisation de la syntaxe du langage naturel. Dans un second temps, nous nous intresserons à la sémantique formelle des langages naturel, à travers la théorie de la compositionnalité qui peut-être modélisée en utilisant le λ-calcul simplement typé, à partir des idées de Richard Montague. Grâce à ces idées sur la modélisation de la syntaxe et de la sémantique, nous introduirons enfin le formalisme sur lequel nous nous appuierons pour mener notre étude à bien : les grammaires catégorielles abstraites. Ces grammaires, qui peuvent être vues comme des grammaires de λ-termes simplement typés,

permettent l'étude de l'interface entre syntaxe et sémantique du langage naturel sous un modèle qui se caractérise par son uniformité (chaque aspect d'une phrase est représenté par un λ-terme) et par la symétrie entre réalisations de surface et réalisations sémantiques d'une phrase. Dans une dernière partie, nous verrons d'ailleurs que des structures syntaxiques très riches et complexes peuvent y être décrites, tout en préservant une transformation de ces structures vers des descriptions sémantiques sous forme de λ-termes.

4.1 Syntaxe : les langages faiblement sensibles au contexte

4.1.1 La hiérarchie de Chomsky

La langage naturel est un système de communication parmi tant d'autres, un type particulier de langage. Cette notion de langage peut être définie sous sa forme *générativiste* : il existe des règles (appelées règles grammaticales) permettant de combiner des symboles (ou mots pour le langage), tout en respectant une certaine structure. Les mots générés par ces règles forment un langage.

La hiérarchie de Chomsky [Chomsky, 1956] permet de séparer des langages selon la forme des règles grammaticales les générant. Il nous faut donc, tout d'abord, introduire la notion de grammaire formelle. Pour ce faire, nous introduisons la notion de *monoïde* par $A^* = (A, \epsilon, .)$ où A est un ensemble (communément appelé *alphabet*) d'éléments (que nous appellerons *symboles*), . est une opération binaire associative de concaténation de symboles, et ϵ est le neutre pour ., appelé *mot vide*. Un élément w appartient à A^* ssi w appartient à A ou si $w = w_1.w_2$ et w_1 et w_2 appartiennent à A^*. Ces éléments seront appelés des *mots* sur l'alphabet A. Toute partie L de A^* sera appelée un *langage*.

Étant donné un mot $w \in A^*$, la longueur de w est définie par induction de la manière suivante : $|\epsilon| = 0$, $|a| = 1$ où $a \in A$ et $|w_1.w_2| = |w_1| + |w_2|$. De même, pour un mot $w \in A^*$ et une lettre $a \in A$, $|w|_a$ est le nombre d'occurrences du symbole a dans le mot w construit sur l'alphabet A, et est défini inductivement par $|\epsilon|_a = 0$, $|w|_a = 0$ si $w \in A - \{a\}$, $|a|_a = 1$ et enfin $|w_1.w_2|_a = |w_1|_a + |w_2|_a$.

Enfin, étant donnés deux langages L_1 et L_2, nous notons par :

- $L_1 + L_2$ l'union de L_1 et L_2.
- $L_1.L_2$ l'ensemble $\{w_1.w_2 \mid w_1 \in L_1 \text{ et } w_2 \in L_2\}$.
- $L_1^0 = \{\epsilon\}$, et $L_1^{n+1} = L_1.L_1^n$, pour $n \geq 0$.
- $L_1^* = \bigcup_{n \geq 0} L_1^n$.

Définition 4.1.1.1. *Une* grammaire *(formelle) est un tuple $G = (T, N, S, P)$ où :*

- *T est un ensemble fini de symboles appelés* terminaux.
- *N est un ensemble fini de symboles appelés* non-terminaux.
- *$S \in N$ est un symbole non-terminal particulier appelé* symbole initial.
- *P est un ensemble fini de* règles de production *de la forme $\alpha \Rightarrow \beta$, où $\alpha \in (N+T)^* N (N+T)^*$ et $\beta \in (N+T)^*$.*

Il est important de noter que, pour toute règle de production $\alpha \Rightarrow \beta$, le symbole α contient au moins un symbole non-terminal.

Notation 4.1.1. En règle générale, nous noterons par des lettres latines majuscules les symboles non-terminaux d'une grammaire, par des lettres latines minuscules les symboles terminaux, et par des lettres grecques minuscules (ou $w, w_1, \ldots$) les mots d'un langage.

Définition 4.1.1.2. *Soient une grammaire $G = (T, N, S, P)$, et $\alpha', \beta' \in (N+T)^*$. Nous dirons que α'* se dérive directement *en β', si il existe $\alpha, \beta, \alpha_1, \alpha_2 \in (N+T)^*$ tels que $\alpha' = \alpha_1 \alpha \alpha_2$, $\beta' = \alpha_1 \beta \alpha_2$ et $\alpha \Rightarrow \beta \in P$.*

Nous noterons par $\Rightarrow^*$ la relation de dérivation, qui est la clôture transitive et réflexive de la relation de dérivation directe.

Exemple 4.1.1.1. Nous définissons une grammaire $G = (T, N, S, P)$ telle que

- $T = \{$`Jean, boit, une, bière, blonde`$\}$
- $N = \{NP, S, N, DET, ADJ\}$ et
- P est donné par les règles de réécriture suivantes :

$$\begin{aligned} S &\Rightarrow NP\ V\ NP \\ NP &\Rightarrow \texttt{Jean} \\ NP &\Rightarrow DET\ N \\ V &\Rightarrow \texttt{boit} \\ DET &\Rightarrow \texttt{une} \\ N &\Rightarrow \texttt{bière} \\ N &\Rightarrow N\ ADJ \\ ADJ &\Rightarrow \texttt{blonde} \end{aligned}$$

Sur cet exemple, nous pouvons voir que la relation de dérivation suivante est vérifiée :

$$S \Rightarrow^* \texttt{Jean boit une bière}$$

Définition 4.1.1.3. *Étant donnée une grammaire $G = (T, N, S, P)$, nous notons par l'ensemble $L(G) = \{\alpha \in T^* \mid S \Rightarrow^* \alpha\}$ le* langage généré par *G.*

Le langage généré par la grammaire de l'Exemple 4.1.1.1 est donc

$L(G) = \{$`(Jean + (une bière (blonde)*)) boit (Jean + (une bière (blonde)*))`$\}$

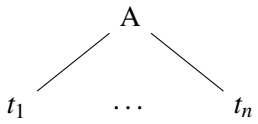
Dans la hiérarchie de Chomsky, les langages ainsi générés sont séparés selon la forme des règles de production des grammaires associées. Quatre classes de grammaires sont ainsi définies :

Définition 4.1.1.4. *Soit une grammaire $G = (N, T, S, P)$, et $\alpha \Rightarrow \beta$; nous dirons que :*

- *G est une grammaire* régulière *si toutes les règles de production de P sont, soit de la forme $A \Rightarrow a$, soit de la forme $A \Rightarrow aB$, où $A, B \in N$ et $a \in T$.*
- *$L(G)$ est une grammaire* non-contextuelle *si toutes les règles de production de P sont de la forme $A \Rightarrow \alpha$ où $A \in N$ et $\alpha \in (N + T)^*$.*
- *$L(G)$ est une grammaire* sensible au contexte *si toutes les règles de production de P sont de la forme $\alpha A \beta \Rightarrow \alpha\gamma\beta$ où $A \in N$ et $\alpha, \beta, \gamma \in (N + T)^*$.*
- *$L(G)$ est une grammaire* générale *si toutes les règles de production de P sont de la forme $\alpha \Rightarrow \beta$ où $\alpha, \beta \in (N + T)^*$.*

Un langage L est dit *régulier* (*i.e. non-contextuel, sensible au contexte, récursivement énumérable*) si il peut être engendré par une grammaire régulière (*i.e. non-contextuelle, sensible au contexte, générale*). D'après cette définition, il apparaît clairement que chacune des classes de langages ainsi définie est strictement incluse dans la suivante.

Remarque 4.1.1.1. Dans le cas d'une grammaire non-contextuelle $G = (N, T, S, P)$, il est usuel de représenter la relation de dérivation sous forme d'arbres, appelés *arbres de dérivation*. Un arbre de dérivation se construit de manière itérée : une règle de production $A \Rightarrow t_1 \dots t_n$ (où pour tout $i \in \{1, \dots, n\}$, $t_i \in \{N, T\}$) est représentée par un arbre dont le non-terminal A en partie gauche de règle est la racine, et les symboles en partie droite de règle, les fils de la racine ; ces fils sont ordonnés selon l'ordre donné par la règle.



Ainsi, pour une dérivation $A \Rightarrow^* w$, il est possible de construire un arbre de dérivation en itérant la réécriture des règles de production utilisées dans cette dérivation. La lecture de gauche à droite de la frontière nous donne le mot dérivé. Des exemples d'arbres de dérivation seront donnés dans les sections qui suivent.

Définition 4.1.1.5. *Étant donnés une grammaire G et un mot α, nous appellerons* problème de reconnaissance *le problème de savoir si α appartient à L(G), en fonction du mot α.*

Le problème de reconnaissance universelle *est identique, mais se résout en fonction de α et de G.*

*Le problème de l'*analyse *de α dans G revient à savoir si $\alpha \in L(G)$ et à retourner les arbres de dérivation associés.*

La classe des langages réguliers et celle des langages non-contextuels sont largement utilisées en informatique pour la définition de langages de programmation. En effet, les langages réguliers ont une importance particulière dans la reconnaissance lexicale, c'est-à-dire la reconnaissance de chaînes de caractères. Par ailleurs, les langages non-contextuels sont eux utilisés pour définir les grammaires des langages de programmation et la structure de ceux-ci. De par la simplicité de leur construction, le problème de la reconnaissance peut y être résolu efficacement, *c.a.d* en temps polynomial.

D'un point de vue linguistique, la question qui se pose est de caractériser une classe de langages qui corresponde aux langages naturels. Ainsi, comme nous allons le voir dans la section suivante, les langages non-contextuels semblent être trop peu expressifs pour rendre compte de certains phénomènes linguistiques complexes. Une classe proposée dans [Joshi, 1985, Weir, 1988] est la classe des langages *faiblement sensibles au contexte* qui est incluse dans la classe des langages sensibles au contexte, tout en contenant la classe des langages non-contextuels.

4.1.2 Les langages faiblement sensibles au contexte

Afin de donner quelques intuitions, nous donnons à présent des exemples significatifs de langages qui permettent de séparer les classes de langages précédemment décrites. Ainsi, le langage $\{a^n b^n\}$ n'est pas un langage régulier mais est un langage non-contextuel. En effet, il peut être généré par une grammaire dont les règles de production sont les suivantes :

$$S \Rightarrow \epsilon$$
$$S \Rightarrow aSb$$

Ainsi, par exemple, le mot a^2b^2 est dérivé de la manière suivante : $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$, que nous pouvons également représenter sous la forme de l'arbre de dérivation de la figure 4.1.

De la même manière, les langages de Dyck sont des langages non-contextuels, mais ne sont pas des langages réguliers. Prenons, par exemple, le langage Dyck(2) (noté $\mathcal{D}_2$), qui peut être vu comme le langage de "parenthésage correct" ; il contient donc des mots sur un alphabet à deux lettres (les parenthèses ouvrantes et fermantes) ; ainsi les mots () ou ((()(()))() appartiennent à $\mathcal{D}_2$, mais pas le mot)(. Ce langage peut être généré par une grammaire dont les règles sont les suivantes :

$$S \Rightarrow \epsilon$$
$$S \Rightarrow (S)$$
$$S \Rightarrow SS$$

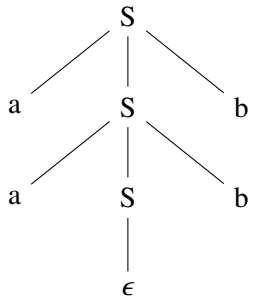


FIG. 4.1 – Arbre de dérivation pour le mot a^2b^2

L'idée principale permettant de faire la distinction entre langages réguliers et non-contextuels (ou sensibles au contexte, ou récursivement énumérables) est que les automates à états finis (qui permettent de reconnaître les langages réguliers) ont une mémoire finie. Ainsi, pour reconnaître le langage $\{a^n b^n \mid n \geq 0\}$, il faut garder en mémoire le nombre n de a qui ont été lus afin de dire si un mot appartient à ce langage, et n est potentiellement infini ; dans le cas du langage $\mathcal{D}_2$, il faut s'assurer que, au moment où un symbole) est lu, il correspond à un symbole (précédent, la distance entre ces deux occurrences étant également potentiellement infinie.

Bien entendu, certains langages ne font pas partie de la classe des langages non-contextuels et en délimitent les frontières. Ainsi, nous pouvons citer le langage de copie $\{ww \mid w \in \{a,b\}^*\}$, les langages où apparaissent des *dépendances croisées* (par exemple : $\{a^n b^m c^n d^m \mid n,m \in \mathbb{N}\}$) ou bien le langage MIX : $\{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$. Or, certains de ces langages sont représentatifs de phénomènes linguistiques particuliers ; par exemple, la construction suivante est correcte en Néerlandais (dans cet exemple, nous indexons les groupes nominaux qui sont en position de sujet pour le verbe dont l'index est identique) :

> *dat ik*$_1$ *Marie*$_2$ *Henk*$_3$ *de nijlpaarden zag*$_1$ *helpen*$_2$ *voeren*$_3$.
> [que je Marie Henk les hippopotames vis aider alimenter].
> que je vis Marie aider Henk à nourrir les hippopotames.

Nous pouvons alors nous rendre compte qu'un tel phénomène ne peut être pris en compte que par une grammaire générant un langage avec dépendances croisées ; en effet, les dépendances entre les syntagmes *ik* et *zag*, d'une part, et *Marie* et *helpen*, d'autre part, se croisent.

Il en est de même pour le Suisse Allemand, d'après l'exemple suivant, dû à [Huybregts, 1984] :

> *wil mer*$_1$ *de maa*$_2$ *em chind*$_3$ *lönd*$_1$ *hälffe*$_2$ *schwüme*$_3$.
> [puisque nous les hommes les enfants laisser aider nager].
> puisque nous laissons les hommes aider les enfants à nager.

Ces deux exemples mettent en valeur des constructions de la forme $NP_1 NP_2 \dots NP_n V_1 V_2 \dots V_n$, où NP_i est lié à VP_i, pour chaque $i \in \{1,\dots,n\}$.

D'autres exemples existent, comme le suivant en Français :

> *Marcel et Jean prendront un demi et un rouge respectivement.*

On peut y noter le phénomène identique de croisement pour les liens entre chacun des groupes nominaux (*Marcel* et *Jean*) et leurs objets respectifs (*un demi* et *un rouge*).

La question que nous pouvons nous poser est donc la définition d'une classe de langages qui corresponde à la structure syntaxique du langage naturel. Ainsi, une première approximation en fut donnée dans [Joshi, 1985] à travers la classe des langages *faiblement sensibles au contexte*.

Définition 4.1.2.1. *La* propriété de croissance constante *est vérifiée par un langage L si il existe une constante $c \in \mathbb{N}$ telle que pour tout mot $w \in L$, il existe un mot $w' \in L$ vérifiant $|w| \leq |w'| \leq |w| + c$.*

Définition 4.1.2.2. *La classe des langages* faiblement sensibles au contexte *est telle que :*

1. *elle comprend les langages non-contextuels.*
2. *le problème de reconnaissance peut être résolu en temps polynomial pour ces langages.*
3. *elle comprend certains langages où apparaissent des dépendances croisées.*
4. *les langages de cette classe vérifient la propriété de croissance constante.*

La classe des langages faiblement sensibles au contexte (que nous noterons MCSL pour *mildly-context sensitive languages*, pour plus de commodités) se situe donc entre la classe de langages non-contextuelles et celle des langages sensibles au contexte. De plus, d'après la deuxième propriété, une contrainte de complexité y est associée, afin d'imposer un traitement efficace du problème de la reconnaissance dans ces langages. D'un point de vue linguistique, une telle contrainte est, bien entendu, tout à fait discutable. Le troisième point est tout autant discutable et informel, puisque les phénomènes de dépendances croisées pouvant être pris en compte n'y sont pas donnés explicitement ; de plus, la notion même de dépendance croisée n'est pas définie.

Remarque 4.1.2.1. Une réponse pouvant expliquer ces lacunes est que la classe des langages faiblement sensibles au contexte fut définie de sorte à inclure les langages générés par un formalisme grammatical que nous décrivons dans la section qui suit : les *grammaires d'arbres adjoints*. Or, les dépendances croisées pouvant être traitées dans ce formalisme ne sont pas entièrement connues (cf. [Salvati, 2011] pour une discussion sur le langage MIX).

Enfin, la dernière propriété exigée sur les MCSGs est une propriété plus faible que la propriété de *semi-linéarité*, largement étudié dans le cadre de la théorie des langages formels. Nous ne développerons pas cette notion dans ce document, mais les lecteurs intéressés peuvent se référer à [Parikh, 1966] pour l'introduction de cette dernière pour les langages non-contextuels, et à [Kanazawa, 2010a] pour une démonstration de la semi-linéarité des langages faiblement sensibles au contexte.

4.1.3 Quelques formalismes MCS

Comme nous venons de le voir, la classe des langages faiblement sensibles au contexte est liée à la définition d'un formalisme particulier : les grammaires d'arbres adjoints. Par la suite, il a été prouvé que de nombreux formalismes grammaticaux génèrent également des langages faiblement sensibles au contexte (bien que les langages qu'ils génèrent ne puissent éventuellement pas être générés par une grammaire d'arbres adjoints). Nous pouvons citer, entre autres les grammaires de tête (HG pour *head grammars* [Pollard, 1984]), ou les grammaires linéaires indexées (LIG pour *linear indexed grammars*).

D'autres formalismes grammaticaux génèrent des langages strictement plus larges que les langages générés par les grammaires d'arbres adjoints ; tel est le cas des transducteurs déterministes de parcours d'arbres (*deterministic tree walking transducers* [Weir, 1992]), des grammaires de remplacement d'hyper-arêtes (*hyperedge replacement grammars* [Engelfriet and Heyker, 1991]), des grammaires minimalistes (*minimalist grammars* [Michaelis, 1998, Michaelis, 2001]) ou des grammaires non-contextuelles multiples (MCFG pour *multiple context-free grammars* [Seki et al., 1991]). Nous nous intéresserons ici aux TAGs et aux MCFGs.

Les grammaires d'arbres adjoints [Joshi et al., 1975, Joshi and Schabes, 1997]

Les grammaire d'arbres adjoints sont des grammaires d'arbres comportant deux opérations : *la substitution* et *l'adjonction*.

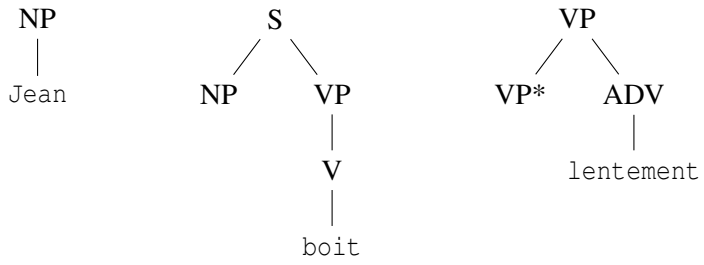


FIG. 4.2 – Exemple de grammaire d'arbres adjoints

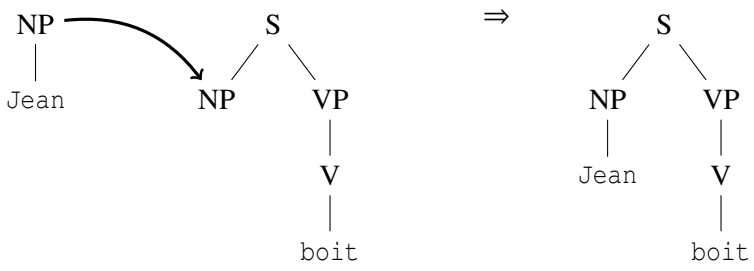
Définition 4.1.3.1. *Une grammaire d'arbres adjoints est un tuple* $G = (N, \Sigma, S, \mathcal{I}, \mathcal{A})$ *où :*

- N *est un ensemble fini de symboles* non-terminaux.
- Σ *est un ensemble fini de symboles appelés* terminaux *(vérifiant* $N \cap \Sigma = \emptyset$*).*
- $S \in N$ *est appelé* le symbole initial.
- $\mathcal{I}$ *est un ensemble fini d'arbres finis et enracinés appelés* initiaux *et qui vérifient les propriétés suivantes :*
  - *chaque feuille est étiquetée, soit par un élément de* Σ *(ou par le mot vide* ϵ*), soit par un élément de* N*.*
  - *chaque nœud interne est étiqueté par un élément de* N*.*
- $\mathcal{A}$ *est un ensemble fini d'arbres finis et enracinés dits* auxiliaires *vérifiant les propriétés suivantes :*
  - *chaque feuille est étiquetée par un élément de* Σ *(ou par le mot vide* ϵ*), ou par un élément de* N *; l'une de ces feuilles (appelé* le pied de l'arbre*) est étiquetée par* A^* *où* A *est l'étiquette de la racine de l'arbre.*
  - *tous les nœuds internes sont étiquetés par des éléments de* N*.*

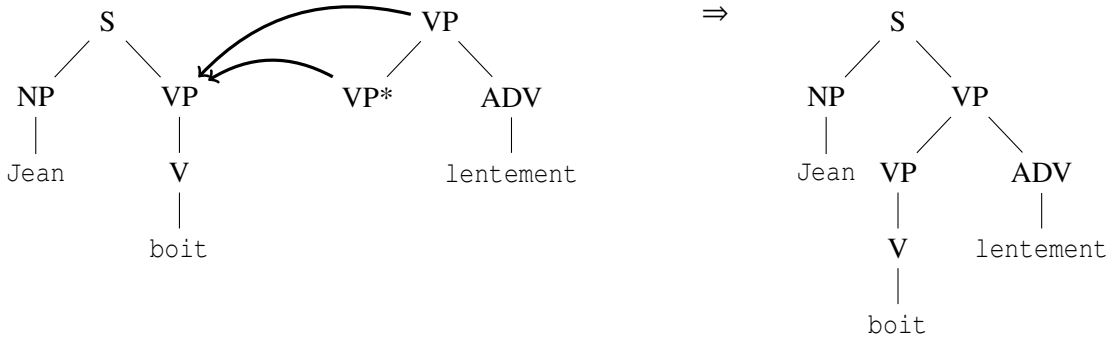
Exemple 4.1.3.1. Nous nous donnons une grammaire simple composée des arbres en Figure 4.2. Cette grammaire est donc composée de deux arbres initiaux, et d'un arbre auxiliaire.

Nous pouvons donner, de manière informelle, les deux opérations qui se réalisent sur ces arbres. Ces deux opérations permettent la création d'un nouvel arbre à partir de deux arbres.

Dans le cas de la première de ces opérations, la substitution, nous considérons deux arbres de la grammaire (*i.e.* appartenant à $\mathcal{I} \cup \mathcal{A}$ ou dérivés dans la grammaire) t_1 et t_2, de telle sorte que $A \in N$ est le symbole en racine de t_2 et est également présent comme feuille de t_1 (cette feuille est différente du pied de t_1 dans le cas où ce dernier est un arbre auxiliaire). L'arbre t résultant est construit par substitution de cette feuille de t_1 par l'arbre t_2. Nous pouvons illustrer cet exemple par l'arbre suivant, dérivé dans la grammaire de l'exemple 4.1.3.1.



La deuxième opération, l'adjonction, permet d'insérer un arbre t_2 à l'intérieur d'un autre arbre t_1. Il faut donc comprendre l'opération de substitution comme une substitution externe (*i.e.* s'opérant sur les feuilles de t_1), et l'adjonction comme une substitution interne (*i.e.* s'opérant sur les nœuds internes de t_1). Par l'adjonction, nous pouvons donc créer un nouvel arbre à partir d'un arbre t_1 et d'un arbre adjoint t_2 (*i.e.* appartenant à $\mathcal{A}$ ou dérivé dans la grammaire tout en respectant la définition d'arbre auxiliaire), en remplaçant un nœud interne étiqueté par $A \in N$ dans t_1 par t_2 à condition que la racine de t_2 soit également étiquetée par le symbole A ; tous les fils du nœud interne à t_1 ainsi remplacé deviennent des fils du pied de l'arbre adjoint t_2 dans l'arbre t nouvellement créé. L'exemple suivant, créé à partir de la grammaire de l'exemple 4.1.3.1, illustre cette opération :



Pour les deux exemples précédents, nous obtenons donc des *arbres dérivés*. Étant donnée une grammaire d'arbres adjoints $G = (N, \Sigma, S, \mathcal{I}, \mathcal{A})$, le langage d'arbres $L_{tree}(G)$ généré par cette grammaire est donné par l'ensemble des arbres dérivés de racine S et ne contenant que des terminaux sur leurs feuilles. Le langage $L_{str}(G)$ est ensuite donné par la frontière (*i.e.* une lecture des feuilles de gauche à droite) des arbres de $L_{tree}(G)$. Sur notre exemple, $L_{tree}(G)$ est constitué d'une infinité d'arbres et $L_{str}(G)$ est égal à $\{\texttt{Jean boit (lentement)}^n \mid n \geq 0\}$.

Les grammaires non-contextuelles multiples [Seki et al., 1991]

À présent, nous introduisons un second formalisme qui peut se présenter comme une extension des grammaires non-contextuelles : alors que les chaînes sont les objets dérivés par des grammaires non-contextuelles, dans le cas des MCFGs, ces objets seront des tuples de chaînes.

Définition 4.1.3.2. *Un alphabet* gradué *est un ensemble fini* $F = \bigcup_{n \in \mathbb{N}} F^n$*, où* $F^n \cap F^m = \emptyset$*, pour tout* $m, n \in \mathbb{N}$ *tels que* $m \neq n$*. Pour tout élément* $f \in F^n$*,* n *est appelé le* rang *ou l'*arité *de* f*.*

Définition 4.1.3.3. *Une grammaire non-contextuelle multiple est un tuple* $G = (N, \Sigma, P, S)$ *où :*

- N *est un alphabet gradué de symboles appelés* non-terminaux.

- Σ *est un alphabet (non-gradué) de symboles appelés* terminaux*, tel que* $\Sigma \cap N = \emptyset$.
- P *est un ensemble fini de règles de réécriture (appelées* règles de production*) de la forme :*

$$A(\alpha_1, \ldots, \alpha_r) \Rightarrow A_1(x_{1,1}, \ldots, x_{1,r_1}), \ldots, A_n(x_{n,1}, \ldots, x_{n,r_n})$$

où :
  - $n \geq 0$
  - $A, A_1, \ldots, A_n$ *sont des symboles non-terminaux de rang* $r, r_1, \ldots, r_n$ *respectivement.*
  - $x_{1,1}, \ldots, x_{1,r_1}, \ldots, x_{n,1}, \ldots, x_{n,r_n}$ *sont des variables distinctes deux-à-deux.*
  - $\alpha_1, \ldots, \alpha_r$ *sont des éléments de* $(\Sigma + \{x_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq r_i\})^*$ *tels que toute variable* $x_{i,j}$ *a exactement une occurrence dans la chaîne* $\alpha_1 \ldots \alpha_r$.
- $S \in N^1$.

Étant donnée une MCFG G, nous dirons qu'il s'agit d'une $m-$MCFG si le rang maximal des non-terminaux de G est égal à m.

Exemple 4.1.3.2. Nous reprenons l'exemple 4.1.3.1, sous forme d'une grammaire non-contextuelle multiple $G = (N, \Sigma, P, S)$ où :

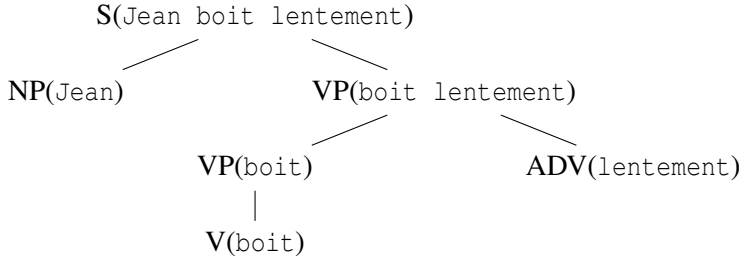
- $N = \{S, NP, VP, V, ADV\}$ et $S, NP, VP, V \in N^1$
- $\Sigma = \{\texttt{Jean}, \texttt{boit}, \texttt{lentement}\}$
- P est fait des règles :

$$\begin{array}{lcl} S(x_1 x_2) & \Rightarrow & NP(x_1), VP(x_2) \\ NP(\texttt{Jean}) & & \\ VP(x) & \Rightarrow & V(x) \\ VP(x_1 x_2) & \Rightarrow & VP(x_1), ADV(x_2) \\ V(\texttt{boit}) & & \\ ADV(\texttt{lentement}) & & \end{array}$$

Étant donnés une MCFG $G = (N, \Sigma, P, S)$ et un non-terminal $A \in N^r$, où $r \geq 0$, et $\alpha_1, \ldots, \alpha_r \in \Sigma^*$, nous définissons la relation de *dérivation* $\Rightarrow^*$ par :

1. $\Rightarrow^* A(\alpha_1, \ldots, \alpha_r)$ si $A(\alpha_1, \ldots, \alpha_r) \in P$, ou bien
2. $\Rightarrow^* A(\alpha_1 \cdot \sigma, \ldots, \alpha_r \cdot \sigma)$ si la règle $A(\alpha_1, \ldots, \alpha_r) \Rightarrow A_1(x_{1,1}, \ldots, x_{1,r_1}), \ldots, A_n(x_{n,1}, \ldots, x_{n,r_n})$ appartient à P, et s'il existe un ensemble $\{\beta_{i,j} \in \Sigma^* \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, r_i\}\}$ de sorte que :
   - la relation $\Rightarrow^* A_i(\beta_{i,1}, \ldots, \beta_{i,r_i})$ est vérifiée pour tout $i \in \{1, \ldots, n\}$
   - σ est la fonction qui substitue $x_{i,j}$ par $\beta_{i,j}$ pour tout $1 \leq i \leq n$ et $1 \leq j \leq r_i$.

Le langage $L(G)$ d'une MCFG $G = (N, \Sigma, P, S)$ est défini par $L(G) = \{w \in \Sigma^* \mid \Rightarrow^* S(w)\}$. De même que pour les grammaires non-contextuelles, nous pouvons représenter les dérivations dans une MCFG par un arbre (l'arbre de dérivation). Par exemple, nous obtenons l'arbre suivant correspondant à la dérivation de la phrase "Jean boit lentement" :



Ainsi, pour la grammaire de l'exemple 4.1.3.2, nous obtenons à nouveau le langage $L(G) = \{\texttt{Jean boit (lentement)}^n \mid n \geq 0\}$. Le langage $L(G)$ généré par une MCFG G est appelé *un langage non-contextuel multiple*.

4.2 Sémantique : la compositionnalité

Après avoir donné quelques pistes sur l'apport réciproque de l'informatique théorique à l'étude de la syntaxe des langues naturelles, nous nous intéressons à présent à l'étude de la sémantique de ces langues, et plus particulièrement à la sémantique associée à une phrase. Pour les formalismes s'adressant à la sémantique d'un discours, les lecteurs intéressés peuvent consulter la théorie de la représentation du discours (DRT) [Kamp, 1981] (cf. [van Eijck, 2005] pour une présentation), la théorie de la représentation du discours segmenté (SDRT) [Lascarides and Asher, 2001], la logique dynamique prédicative [Groenendijk and Stokhof, 1991] ou plus récemment l'utilisation des continuations [de Groote, 2006, de Groote and Lebedeva, 2010].

4.2.1 Compositionnalité, logique et λ-calcul

Une des idées fondamentales de la philosophie du langage est que le langage naturel est utilisé afin de communiquer sur les objets du monde. La sémantique de Montague se place sous cette hypothèse, et décrit les conditions pour lesquelles une phrase est acceptée par un auditeur ou lecteur. Ainsi, pour une phrase telle que "James bond est un espion", cette phrase est acceptée si l'on sait, par exemple, que James Bond est un personnage fictif, et que dans le monde créé par Ian Flemming, Bond est effectivement un espion. Elle est également acceptée (comme vraie) si dans le monde que l'on considère, il existe effectivement un espion du nom de James Bond. Nous parlerons alors de *sémantique véri-conditionnelle*.

Afin d'exprimer ces conditions d'acceptation, nous pouvons utiliser une représentation du sens d'une phrase comme une formule de la *logique du premier ordre* :

Définition 4.2.1.1. *Étant donné $\mathcal{L} = (\mathcal{P}, E, X)$ où $\mathcal{P}$ est un ensemble de* prédicats*, E un ensemble de* constantes *et X un ensemble de variables, le langage du premier ordre défini sur $\mathcal{L}$ contient toutes les formules $\mathcal{F}$ telles que :*

- *$\mathcal{F} = P(a_1, \ldots, a_n)$ où $P \in \mathcal{P}$, n est son arité, et $a_1, \ldots, a_n \in E \cup X$.*
- *$\mathcal{F} = \neg\mathcal{F}'$ et $\mathcal{F}'$ est une formule.*
- *$\mathcal{F}$ est égale à $\mathcal{F}_1 \wedge \mathcal{F}_2$, $\mathcal{F}_1 \vee \mathcal{F}_2$, $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$ ou $\mathcal{F}_1 \Leftrightarrow \mathcal{F}_2$ et $\mathcal{F}_1$ et $\mathcal{F}_2$ sont des formules.*
- *$\mathcal{F}$ est égale à $\forall x.\mathcal{F}'$ ou $\exists x.\mathcal{F}'$ si x est une variable et $\mathcal{F}'$ une formule.*

Prenons pour exemple, la phrase "Jean rentre dans un bar". Nous obtenons la représentation sémantique associée suivante : $\exists x.(bar(x) \wedge rentre\_dans(Jean, x))$. Afin de donner des intuitions sur la suite, nous pouvons regarder l'interprétation d'une telle formule dans le modèle ensembliste de la logique des prédicats.

Définition 4.2.1.2. *Étant donné un langage du premier-ordre défini sur $(\mathcal{P}, E, X)$, un modèle $\mathcal{M}$ de ce langage :*

- *associe à toute constante $c \in E$ un élément a_c d'un ensemble U, appelé l'univers.*
- *associe à tout prédicat n-aire $P \in \mathcal{P}$, une relation $R_P \subseteq U^n$.*

De plus, nous définissons un fonction de variables $f : X \cup E \mapsto U$, vérifiant $f(c) = a_c$ pour toute constante c de E.

Un modèle $\mathcal{M}$ et une fonction d'assignation f valident une formule du premier-ordre $\mathcal{F}$, (noté $\mathcal{M}, f \vDash \mathcal{F}$) ssi :

- $\mathcal{F} = P(x_1, \ldots, x_n)$ et $(f(x_1), \ldots, f(x_n)) \in R_P$.
- $\mathcal{F} = \neg\mathcal{F}$ et $\mathcal{M}, f \vDash \mathcal{F}$ n'est pas valide.
- $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$ (*resp.* $\mathcal{F}_1 \vee \mathcal{F}_2$), $\mathcal{M}, f \vDash \mathcal{F}_1$ et (*resp.* ou) $\mathcal{M}, f \vDash \mathcal{F}_2$.
- $\mathcal{F} = \forall x.\mathcal{F}'$ si pour toute fonction d'assignation d'assignation f' telle que $f'(y) = f(y)$ pour tout $y \in X \cup E - \{x\}$, alors $\mathcal{M}, f' \vDash \mathcal{F}'$.
- $\mathcal{F} = \exists x.\mathcal{F}'$ si il existe une fonction d'assignation d'assignation f' telle que $f'(y) = f(y)$ pour tout $y \in X \cup E - \{x\}$, et qui vérifie $\mathcal{M}, f' \vDash \mathcal{F}'$.

Selon ces définitions, la formule $\exists x.(bar(x) \wedge rentre\_dans(Jean, x))$ est donc vraie, si il existe un élément c de l'univers, tel que c appartient à l'ensemble *bar* sur U et tel que $(Jean, c)$ appartient à l'ensemble *rentre_dans*. Il convient de remarquer que les constantes sont ici interprétées comme des éléments de l'univers.

Étant donnée une expression, le problème qui se pose alors est le calcul de sa représentation sémantique. Pour ce faire, nous nous appuyons sur la théorie de la *compositionnalité*, dont voici un énoncé :

Le sens d'une expression composée est fonction du sens de ses parties et des règles syntaxiques qui les lient.

La notion de compositionnalité, que nous devons originellement à Frege, a été reprise et implémentée de manière algébrique dans les années 1970 par Richard Montague [Montague, 1970b, Montague, 1970a] sous l'énoncé ci-dessus. Dans son inteprétation de cette théorie, Montague utilisa le λ-calcul simplement typé, comme une représentation de formules de la logique du premier-ordre. De cette manière, il nous est possible d'associer un λ-terme à la sémantique d'une partie d'une phrase, puis de composer ces termes entre eux selon certaines règles guidées par la syntaxe, afin d'obtenir le contenu sémantique de la nouvelle expression ainsi analysée. Ce contenu est alors un λ-terme pouvant s'interpréter comme une formule de la logique du premier-ordre.

Dans la théorie montagovienne, il est fait usage de trois types : e pour les entités, t pour les valeurs de vérité, et s pour les mondes possibles. Par souci de simplicité et puisque tel n'est pas le propos de ce travail, nous n'utiliserons pas la notion de mondes possibles pour représenter la sémantique des phrases. Les types associés aux termes de notre représentation sémantique seront donc construits exclusivement sur l'ensemble $\{e, t\}$. Prenons à présent un exemple de phrase : "Jean aime Roméo". Cette phrase comporte trois unités : les noms propres "Jean" et "Roméo", ainsi que le verbe transitif "aime". Nous avons donc trois λ-termes, M_{jean}, M_{romeo} et M_{aime} associés à ces trois unités. Intuitivement, les termes M_{jean} et M_{romeo} seront typés par le type atomique e puisqu'ils désignent chacun un élément de l'univers (une constante logique) ; quant au verbe "aime", il est représenté comme un prédicat logique binaire ; sous forme de λ-terme, il s'agit donc d'une fonction prenant deux entités en arguments, le résultat étant une valeur de vérité. Ainsi, nous obtenons :

- $M_{jean} = \mathbf{Jean} : e$
- $M_{romeo} = \mathbf{Romeo} : e$
- $M_{aime} = \mathbf{aime} : e \to e \to t$

D'après l'énoncé du principe de compositionnalité, il nous serait a priori possible d'associer les termes M_{aime}, M_{romeo} et M_{jean} aux unités lexicales "aime", "Roméo" et "Jean" respectivement. Nous verrons cependant en section 4.2.3 que les termes associés aux unités lexicales sont en fait plus complexes, afin de pouvoir traiter certaines phénomènes sémantiques, et en particulier la portée des quantificateurs.

Dans la théorie de la compositionnalité, la phrase “Jean aime Roméo” aurait donc une représentation sémantique donnée par le λ-terme **aime Jean Romeo** qui est de type t. Ce λ-terme s'interprète, en fait, comme une formule de la logique du premier ordre (plus communément écrite **aime**(**Jean**, **Romeo**), d'après la définition que nous avons donnée d'une formule de la logique du premier ordre), et le type t stipule qu'une telle affirmation, c'est-à-dire le fait que Jean aime Roméo, est soit vraie, soit fausse.

Comme nous venons de le dire, dans le théorie de la compositionalité ainsi présentée, les λ-termes représentant la sémantique d'une expression s'interprètent comme des formules de la logique du premier ordre. Il existe donc des termes correspondant à chaque constante, chaque variable et chaque prédicat que nous introduisons, mais aussi des termes correspondant à chaque connecteur de cette logique. De par l'interprétation informelle que nous avons donnée des termes comme formules logiques, dans le paragraphe précédent, nous obtenons les types suivants pour ces connecteurs :

- $\neg$ inverse une valeur de vérité fournie en paramètre ; il s'agit donc d'un terme de type $t \to t$.
- de même, les connecteurs $\wedge, \vee, \Rightarrow, \Leftrightarrow$ sont des connecteurs binaires qui agissent sur des valeurs de vérité, et sont donc de type $t \to t \to t$.
- $\forall$ (*resp.* $\exists$) est un connecteur du premier ordre : à partir d'un prédicat unaire (donc d'un ensemble), il permet de vérifier que tous les éléments sont (*resp.* qu'un élément est) dans cet ensemble ; les termes correspondants sont donc de type $(e \to t) \to t$.

4.2.2 De la syntaxe à la sémantique

Maintenant que nous avons introduit les outils permettant de construire la sémantique d'une phrase, nous nous intéressons au lien entre syntaxe et sémantique. Le problème revient donc à obtenir une correspondance entre la catégorie syntaxique d'une unité de surface, et un type sémantique. En premier lieu, nous donnons quelques acronymes qui seront régulièrement utilisés pour désigner les catégories syntaxiques les plus récurrentes :

- S pour une phrase (*sentence*).
- NP pour un groupe (ou syntagme) nominal (*nominal phrase*).
- VP pour un groupe verbal (*verbal phrase*).
- PP pour un groupe pronominal (*pronominal phrase*).
- V pour un verbe (on séparera éventuellement les verbes intransitifs IV des verbes transitifs TV) (*resp. verb, intransitive verb* et *transitive verb*).
- Det pour les articles (*determiner*).

On peut voir, sur les exemples en section 4.2.1, que le type e est associé aux noms propres. De manière plus générale, nous pouvons associer le type e à tout groupe nominal. Concernant les verbes intransitifs, nous pouvons les voir comme des prédicats unaires (les sujets de ces verbes correspondant à l'unique argument du prédicat en question), et sont donc de type $e \to t$; les verbes transitifs seront, quant à eux, de type $e \to e \to t$, c'est-à-dire des prédicats binaires sur le sujet et l'objet du verbe. Dans le cas d'un nom, par exemple “table”, le type e ne peut être assigné. En effet, ce type correspondrait à une entité qui est une table, donc, par exemple, ou groupe nominal “la table” ou “une table”. Nous assignons donc le type $e \to t$ à un nom. Ce type nous indique que pour une entité donnée, le prédicat associé à “table” retourne une valeur de vérité indiquant si cette entité est une table ou pas.

Remarque 4.2.2.1. Une telle assignation est réalisée de manière semblable à l'assignation de catégories syntaxiques dans les grammaires catégorielles (grammaires AB [Adjukiewicz, 1935], grammaires de Lambek [Lambek, 1958], grammaires de Lambek multi-modales [Moortgat, 1997], calcul de la discontinuité [Morrill et al., 2007, Morrill et al., 2010]). Ces grammaires étant basées sur la définition d'une logique permettant de rendre compte des opérations syntaxiques (par exemple, à un

verbe intransitif serait associée une catégorie syntaxique $NP \rightarrow S$), l'isomorphisme de Curry-Howard permet d'obtenir les termes sémantiques associés aux unités syntaxiques. En effet, une démonstration logique correspond alors à une dérivation syntaxique, dont l'interprétation sémantique est donnée par le λ-terme associé à la démonstration.

Le cas des grammaires catégorielles est en fait un cas bien particulier d'implémentation de la théorie de Montague. En effet, à travers la formalisation de grammaires universelles [Montague, 1970b], syntaxe et sémantique sont représentées par des algèbres, liées par un morphisme. Néanmoins, la nature de ces algèbres n'est pas donnée. Typiquement, l'algèbre associée à la syntaxe sera une algèbre sortée, représentant les arbres de dérivation syntaxique. Nous introduirons en Section 4.3 les grammaires catégorielles abstraites comme une formalisation de l'interface entre syntaxe et sémantique proposée par Montague.

Enfin, un des aspects essentiels de l'interface syntaxe/sémantique dans la théorie de Montague, est que la sémantique des phrases y est exprimée sous forme de formules de la logique du premier ordre, grâce des λ-termes simplement typés. Or, cette représentation peut être prise comme un langage pivot, et faciliter ainsi la création d'outils de traduction automatique : à partir d'une phrase syn_1 dans une langue L_1, nous obtenons sa représentation sémantique *sem*, (éventuellement ses représentations sémantiques, si la phrase est ambiguë), puis générons une phrase syn_2 sémantiquement équivalente dans une langue L_2.

4.2.3 Quantificateurs et typage

Nous avons donné une interprétation ensembliste des types assignés aux termes sémantiques. Selon cette interprétation, nous associons le type e au terme sémantique d'une phrase nominale (**Jean** sur notre exemple). Il est également possible de définir une entité par l'ensemble des propriétés qu'elle vérifie. Ainsi, le type associé à un groupe nominal serait $(e \rightarrow t) \rightarrow t$, et le terme correspondant deviendrait $\lambda P.P\mathbf{Jean}$. La possibilité de modifier ainsi les types a un objectif linguistique précis : l'ordre des mots dans une phrase ne correspond pas forcément à l'ordre de leurs interprétations respectives dans la forme sémantique de la phrase. Ainsi, certaines unités lexicales ont une portée large d'un point de vue sémantique. Afin d'illustrer ce fait, prenons pour exemple la phrase :

"Jean mange une pizza."

L'interprétation de cette phrase est donc donnée par la formule de logique du premier ordre suivante :

$$\exists x.\mathbf{pizza}(x) \wedge \mathbf{mange}(\mathbf{Jean}, x)$$

c'est-a-dire l'affirmation qu'il existe une pizza que Jean mange.

Il apparaît donc que les quantificateurs ont une certaine portée, dont dépend le sens des phrases. Sur notre exemple, le quantificateur est en fait introduit dans l'interprétation sémantique de l'article *une*, et doit se trouver au niveau le plus extérieur de la formule logique ; en effet, une formule telle que $(\exists x.\mathbf{pizza}(x)) \wedge \mathbf{mange}(\mathbf{Jean}, x)$, n'aurait pas de sens, ou éventuellement celui d'affirmer qu'il existe une pizza et que Jean mange quelque chose.

La solution au problème de portée des quantificateurs proposée dans [Montague, 1973], est d'assigner des types sémantiques qui décrivent les entités non plus de manière directe, mais par l'ensemble des propriétés qui les caractérisent. Ainsi, la représentation sémantique de Jean ne doit plus être un terme **Jean**, mais un terme $\lambda P.P\mathbf{Jean}$. Sans décrire les modèles du λ-calcul et leurs liens avec les modèles de la logique du premier-ordre, il s'agit ici de décrire l'ensemble des propriétés P tels que $R_P\mathbf{Jean}$ est vraie dans le modèle de la logique du premier ordre. Ainsi, d'un terme de type e, nous

passons à un terme de type $(e \to t) \to t$. Les termes sémantiques se trouvent donc affectés, par ce qu'on appelle *la montée de type* (ou *type raising*), qui nous permet d'obtenir, par exemple, les termes sémantiques suivants pour chacune des unités lexicales de notre phrase :

Terme syntaxique	Catégorie	Terme sémantique	Type
Jean	np	$M_{jean} = \lambda P.P\,\mathbf{Jean}$	$(e \to t) \to t$
mange	tv	$M_{manger} = \lambda PQ.P(\lambda x.Q(\lambda y.\mathbf{mange}\ xy))$	$((e \to t) \to t) \to ((e \to t) \to t) \to t$
une	det	$M_{un} = \lambda PQ.\exists(\lambda x.\wedge\ (Px)\ (Qx))$	$(e \to t) \to (e \to t) \to t$
pizza	n	$M_{pizza} = \lambda x.\mathbf{pizza}\ x$	$e \to t$

On remarque que, dans ce cas, le terme associé à la sémantique de cette phrase est le suivant :

$$M_{manger} M_{jean} (M_{une} M_{pizza})$$

qui se réduit dans la forme sémantique désirée :

$$\exists(\lambda x.\wedge(\mathbf{pizza}\ x)(\mathbf{mange}\ \mathbf{Jean}\ x))$$

Le déplacement du quantificateur peut être réalisé par le fait que dans les nouveaux types sémantiques utilisés, certains prédicats peuvent prendre place au-dessus de la constante **mange** dans le terme associé au verbe "mange".

4.3 Les grammaires catégorielles abstraites

Lors de la section précédente, nous avons parlé de grammaires de chaînes, de grammaires de tuples de chaînes, et de grammaires d'arbres adjoints ; tous ces formalismes sont utilisés pour rendre compte de la grammaticalité des langues naturelles, en construisant les arbres de dérivation associés à des structures de surface, qui sont typiquement des phrases ou expressions. Nous avons également évoqué les grammaires catégorielles, dont l'originalité est d'offrir une interface directe entre syntaxe et sémantique par la théorie de la compositionnalité et l'isomorphisme de Curry-Howard.

Les grammaires catégorielles abstraites [de Groote, 2001, Muskens, 2001] ont des liens avec les deux types de formalismes évoqués ci-dessus. En effet, nous verrons que toute TAG ou MCFG peut être vue comme une grammaire de λ-termes. Par ailleurs, nous verrons que l'utilisation exclusive de λ-termes simplement typés permet de définir une interface avec la sémantique de Montague pour ces formalismes, à l'image de ce qui est réalisé dans le cas des grammaires catégorielles. Néanmoins, et à la différence des grammaires catégorielles usuelles, l'interface syntaxe-sémantique ne se fait pas grâce un morphisme "figé" (la correspondance de Curry-Howard), mais par des morphismes intégrés dans le formalisme, et dépendant uniquement de la construction de la grammaire. Par ailleurs, la notion de grammaires de λ-termes étend de manière naturelle les grammaires de chaînes et les grammaires d'arbres.

Dans un premier temps, nous présentons donc ces grammaires de manière formelle, avant d'exhiber certaines de leurs propriétés, en particulier la mise à jour d'une hiérarchie de classes de langages au sein du formalisme. Nous montrons ensuite que des formalismes connus peuvent être encodés comme des grammaires catégorielles abstraites, avant de conclure en montrant comment certains phénomènes linguistiques peuvent être traités dans ce formalisme.

4.3.1 Des grammaires de λ-termes simplement typés

Nous avons vu que les grammaires de la hiérarchie de Chomsky traitent de chaînes de mots, et que la structure de monoïde libre sur un alphabet définit un ensemble de chaînes (ou de mots). En

considérant les grammaires catégorielles abstraites comme des grammaires de termes simplement typés, nous avons donc besoin d'une structure permettant de définir un ensemble de termes, à l'image du monoïde libre sur les mots.

Définition 4.3.1.1. *Une signature de λ-termes est un triplet $\Sigma = (\mathcal{A}, C, type)$ tel que :*
- *$\mathcal{A}$ est un ensemble fini de types atomiques.*
- *C est un ensemble fini de constantes.*
- *$type : C \mapsto \mathcal{T}(\mathcal{A})$ est une fonction d'assignation de types de $\mathcal{T}(\mathcal{A})$*

Pour une signature de λ-termes $\Sigma = (\mathcal{A}, C, type)$, nous définissons $\Lambda(\Sigma) = (\Lambda_\alpha(\Sigma))_{\alpha \in \mathcal{T}(\mathcal{A})}$ comme étant la plus petite famille de λ-termes typés à la Church sur un ensemble de variables $\mathcal{X}$ et sur C vérifiant :

1. $x^\alpha \in \Lambda_\alpha(\Sigma)$, pour toute variable $x \in \mathcal{X}$ et tout type $\alpha \in \mathcal{T}(\mathcal{A})$.
2. $\mathbf{c} \in \Lambda_{type(\mathbf{c})}(\Sigma)$, pour toute constante $\mathbf{c} \in C$
3. si $M \in \Lambda_\beta(\Sigma)$, alors $\lambda x^\alpha.M \in \Lambda_{\alpha \to \beta}(\Sigma)$,
4. si $M_1 \in \Lambda_{\beta \to \alpha}(\Sigma)$, $M_2 \in \Lambda_\beta(\Sigma)$, alors $M = M_1 M_2 \in \Lambda_\alpha(\Sigma)$.

Ainsi, pour une signature de λ-termes $\Sigma = (\mathcal{A}, C, type)$ et un type $\alpha \in \mathcal{T}(\mathcal{A})$, l'ensemble $\Lambda_\alpha(\Sigma)$ est fait de tous les λ-termes (typés à la Church) de type α, dont les constantes appartiennent à C, les types assignés à ces dernières étant donnés par la fonction d'assignation de types *type*.

Définition 4.3.1.2. *Étant donnée une signature de λ-termes $\Sigma = (\mathcal{A}, C, type)$, l'ordre de Σ est défini par* $\mathtt{ord}(\Sigma) = max_{\mathbf{c} \in C} \mathtt{ord}(type(\mathbf{c}))$

Exemple 4.3.1.1. Nous reprenons l'exemple de la section 4.2.3 afin de construire le terme représentant la sémantique de la phrase "Jean mange une pizza". Une signature de λ-termes permettant de générer ce terme en n'utilisant que les types "entités" (*i.e.* e) et "valeurs de vérité" (*i.e.* t) est $\Sigma = (\mathcal{A}, C, type)$ où :
- $\mathcal{A} = \{e, t\}$
- $C = \{\mathbf{Jean}, \mathbf{mange}, \exists, \wedge, \mathbf{pizza}\}$
- la fonction *type* est définie par :
  - $type(\mathbf{Jean}) = e$
  - $type(\mathbf{mange}) = e \to e \to t$
  - $type(\exists) = (e \to t) \to t$
  - $type(\wedge) = t \to t \to t$
  - $type(\mathbf{pizza}) = e \to t$

Il est ainsi aisé de voir que $\exists(\lambda x.\wedge(\mathbf{pizza}x)(\mathbf{mange\ Jean}\ x))$ appartient à $\Lambda_t(\Sigma)$.

Par analogie avec la construction algébrique des grammaires de Montague pour l'interface entre syntaxe et sémantique, une signature d'ordre supérieur peut s'interpréter comme une algèbre de λ-termes. Nous définissons à présent la notion de morphisme de signatures d'ordre supérieur suivant :

Définition 4.3.1.3. *Étant données $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ et $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ deux signatures de λ-termes, un* morphisme de signatures $\mathcal{H} = [\mathcal{F}_1, \mathcal{F}_2]$ *de $\Lambda(\Sigma_1)$ sur $\Lambda(\Sigma_2)$ est un couple de fonctions tel que $\mathcal{F}_1 : C_1 \mapsto \Lambda(\Sigma_2)$ et $\mathcal{F}_2 : \mathcal{A}_1 \mapsto \mathcal{T}(\mathcal{A}_2)$, et que vérifie :*
- *pour tout type $\alpha \in \mathcal{T}(\mathcal{A})$,*
  - *si $\alpha = a \in \mathcal{A}$ alors $\mathcal{H}(a) = \mathcal{F}_2(a)$.*
  - *$\alpha = \alpha_1 \to \alpha_2 \in \mathcal{T}(\mathcal{A})$, $\mathcal{H}(\alpha) = \mathcal{H}(\alpha_1) \to \mathcal{H}(\alpha_2)$.*
- *$\mathcal{H}(M) =$*

– $x^{\mathcal{H}(\alpha)}$ *si* $M = x^{\alpha}$.
– $\mathcal{F}_1(\mathbf{c})$ *si* $M = \mathbf{c} \in C$ *; qui plus est* $\mathcal{H}(M) \in \Lambda_{\alpha}(\Sigma_2)$ *où* $\alpha = \mathcal{H}(type_1(\mathbf{c}))$.
– $\lambda x^{\mathcal{H}(\alpha)}.\mathcal{H}(M')$ *si* $M = \lambda x^{\alpha}.M'$.
– $\mathcal{H}(M_1)\mathcal{H}(M_2)$ *si* $M = M_1 M_2$.

Par induction, nous vérifions aisément que si $M \in \Lambda_{\alpha}(\Sigma_1)$, alors $\mathcal{H}(M) \in \Lambda_{\mathcal{H}(\alpha)}(\Sigma_2)$.

Définition 4.3.1.4. *Une* grammaire catégorielle abstraite *(ou ACG pour Abstract Categorial Grammar) est un tuple* $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ *tel que :*

- $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ *est une signature de termes, appelée* signature abstraite *de G.*
- $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ *est une signature de termes, appelée* signature objet *de G.*
- $\mathcal{L}$ *est un morphisme de signature de* $\Lambda(\Sigma_1)$ *sur* $\Lambda(\Sigma_2)$ *appelé* lexique *de G.*
- s *est un type atomique de* Σ_1 *appelé* type initial *de G.*

Une telle grammaire est dite linéaire *(resp.* affine*) si pour toute constante* $\mathbf{c} \in C_1$*, le terme* $\mathcal{L}(\mathbf{c})$ *est linéaire (resp. affine).*

Remarque 4.3.1.1. Nous nous démarquons ici de la définition originelle de [de Groote, 2001], où les deux signatures, abstraite et objet d'une ACG définissent des termes linéaires. Néanmoins, la contrainte de linéarité apparaîtra de manière naturelle pour les grammaires non-contextuelles.

Définition 4.3.1.5. *Étant donnée une ACG* $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$*, l'ordre de G est défini comme l'ordre de* Σ_1.

Cette notion d'ordre nous permet en particulier de nous intéresser aux ACGs d'ordre 2, que nous appellerons ACGs non-contextuelles (ou λ-grammaires non-contextuelles). Pour ces dernières, comme nous le verrons par la suite, la signature abstraite est équivalente à une signature d'arbres.

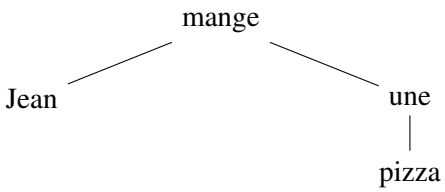
Afin de poursuivre l'analogie avec les grammaires non-contextuelles de chaînes, nous pouvons regarder Σ_1 comme l'alphabet gradué permettant de construire l'ensemble des arbres de dérivation pour cette grammaire, Σ_2 comme le monoïde sur l'ensemble des symboles terminaux, et enfin $\mathcal{L}$ comme la fonction de lecture de la frontière des arbres de dérivation. Cette analogie donne en fait lieu à un encodage des grammaires non-contextuelles [de Groote and Pogodalla, 2004], et à de nombreux autres formalismes dont les dérivations ont une forme arborescente comme des ACGs non-contextuelles (cf. section suivante).

Comme nous l'avons dit au préalable, les idées des grammaires universelles de Montague sont ici strictement implémentées ; en effet, l'interface syntaxe-sémantique est réalisée par le morphisme $\mathcal{L}$, défini lors de la construction de la grammaire. Un autre aspect des ACGs, et qui distingue celles-ci des grammaires catégorielles, est la spécification d'un niveau intermédiaire entre réalisations syntaxique et sémantique des phrases. L'application de cette idée aux langages naturels fut originellement formulée dans [Curry, 1961] (voir également [Muskens, 2006] pour une discussion), où deux niveaux de représentation des expressions d'un langage sont spécifiés : la *tectogrammaire* décrit les mécanismes de dérivation d'une expression, sa structure, typiquement sous forme d'arbres de dérivation ; la *phénogrammaire* décrit la réalisation de cette structure. Cette dernière peut donc décrire la réalisation d'une phrase sous forme de concaténations de chaînes, ce que nous nommons *réalisation de surface*, ou bien sous forme de formules de la logique du premier ordre, sa *réalisation sémantique*.

Exemple 4.3.1.2. En guise d'exemple, nous définissons une grammaire catégorielle abstraite associée à la sémantique de la phrase "Jean mange une pizza". Dans un premier temps, nous donnons une signature de λ-termes rendant compte de la structure de cette phrase : $\Sigma_{tecto} = (\mathcal{A}_{tecto}, C_{tecto}, type_{tecto})$ où :

- $\mathcal{A}_{tecto} = \{np, s, n\}$
- $C_{tecto} = \{jean, mange, une, pizza\}$
- $type_{tecto}$ est telle que :
  - $type_{tecto}(jean) = np$
  - $type_{tecto}(mange) = np \to np \to s$
  - $type_{tecto}(une) = n \to np$
  - $type_{tecto}(pizza) = n$

Nous pouvons voir que cette signature est une signature d'ordre 2. Or, les λ-termes du second-ordre décrivent des structures arborescentes puisque nous ne pouvons pas dériver un terme à partir de termes d'ordre 2 en utilisant la règle d'abstraction ; ainsi, une représentation arborescente de la structure de la phrase "Jean mange une pizza" est la suivante :



Le λ-terme correspondant à cette structure syntaxique est *mange* (*une pizza*) *jean*, qui est correctement typé et appartient à $\Lambda_s(\Sigma_{tecto})$. La signature associée à la réalisation sémantique de cette phrase est donnée dans l'Exemple 4.3.1.1. Nous la noterons par $\Sigma_{sem} = (\mathcal{A}_{sem}, C_{sem}, type_{sem})$. Il nous reste à définir le lexique de cette grammaire catégorielle abstraite $G_{sem} = (\Sigma_{tecto}, \Sigma_{sem}, \mathcal{L}_{sem}, s)$. Concernant, les types, nous obtenons les correspondances suivantes : $\mathcal{L}_{sem}(np) = (e \to t) \to t$, $\mathcal{L}_{sem}(s) = t$, $\mathcal{L}_{sem}(n) = e \to t$.

Les termes sont, quant à eux, transformés de la manière suivante :

- $\mathcal{L}_{sem}(jean) = \lambda P.P$**Jean**
- $\mathcal{L}_{sem}(mange) = \lambda PQ.P(\lambda x.Q(\lambda y.$**mange y x**$))$
- $\mathcal{L}_{sem}(une) = \lambda PQ.\exists(\lambda x.\wedge(Px)(Qx))$
- $\mathcal{L}_{sem}(pizza) = \lambda x.$**pizza** x

Nous définissons à présent le langage reconnu par une grammaire catégorielle abstraite en deux étapes, la première étape rendant compte du calcul des dérivations dans cette grammaire, la deuxième, de l'interprétation de ces dérivations par le morphisme donnée dans la grammaire :

Définition 4.3.1.6. *Étant donnée une grammaire catégorielle abstraite $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$, nous définissons :*

- *le* langage abstrait *de G par $\mathcal{A}(G) = \{M \in \Lambda_s(\Sigma_1) \mid FV(M) = \emptyset\}$*
- *le* langage objet *(ou plus simplement le langage de G) par $\mathcal{O}(G) = \{M \in \Lambda(\Sigma_2) \mid$ il existe $N \in \mathcal{A}(G)$ tel que $M = |\mathcal{L}(N)|_\beta\}$*

Remarque 4.3.1.2. Dans le cas d'une ACG du second-ordre, les termes du langage abstrait sont uniquement composés de constantes (on peut directement prouver que l'abstraction de variables ne peut être utilisée) dont les types ont un ordre au plus 2 ; ces termes peuvent alors s'interpréter comme des arbres.

En poursuivant l'analogie avec les grammaires non-contextuelles de chaînes, le langage abstrait correspond à l'ensemble des arbres de dérivation dont la racine est étiquetée par le symbole S, et le langage objet au langage généré par la grammaire.

Sur notre exemple, nous obtenons donc $\mathcal{A}(G_{sem}) = \{$*mange jean* (*une pizza*),*mange* (*une pizza*) *jean*,*mange jean jean*,*mange* (*une pizza*) (*une pizza*)$\}$.

Le langage de cette grammaire est donc donné par les formes β-réduites des termes images de $\mathcal{A}(\mathcal{G})$ par $\mathcal{L}_{sem}$; en particulier, nous obtenons $\mathcal{L}_{sem}$(*mange* (*une pizza*) *jean*) =

$$
\begin{array}{ll}
 & (\lambda PQ.P(\lambda x.Q(\lambda y.\textbf{mange } y\ x))) ((\lambda PQ.\exists(\lambda x.\wedge(Px)(Qx)))(\lambda x.\textbf{pizza } x)) (\lambda P.P\textbf{Jean}) \\
\rightarrow_\beta^* & (\lambda PQ.P(\lambda x.Q(\lambda y.\textbf{mange } y\ x))) (\lambda Q.\exists(\lambda x.\wedge(\textbf{pizza } x)(Qx))) (\lambda P.P\textbf{Jean}) \\
\rightarrow_\beta & (\lambda Q.(\lambda Q.\exists(\lambda x.\wedge(\textbf{pizza } x)(Qx)))(\lambda x.Q(\lambda y.\textbf{mange } y\ x))) (\lambda P.P\textbf{Jean}) \\
\rightarrow_\beta^* & (\lambda Q.\exists(\lambda x.\wedge(\textbf{pizza } x)(Q(\lambda y.\textbf{mange } y\ x)))) (\lambda P.P\textbf{Jean}) \\
\rightarrow_\beta & \exists(\lambda x.\wedge(\textbf{pizza } x)((\lambda P.P\textbf{Jean})(\lambda y.\textbf{mange } y\ x))) \\
\rightarrow_\beta^* & \exists(\lambda x.\wedge(\textbf{pizza } x)(\textbf{mange Jean } x))
\end{array}
$$

Le langage de cette grammaire est donc composé des termes

- $\exists(\lambda x.\wedge(\textbf{pizza } x)(\textbf{mange Jean } x)))$,
- $\exists(\lambda x.\wedge(\textbf{pizza } x)(\textbf{mange } x\ \textbf{Jean})))\}$,
- **mange Jean Jean** et
- $\exists(\lambda x.\wedge(\textbf{pizza x})(\exists(\lambda y.\wedge(\textbf{pizza } y)(\textbf{mange } x\ y))))$

Afin de montrer l'articulation de l'interface syntaxe-sémantique de manière complète, nous définissons une seconde ACG, $G_{syn} = (\mathcal{A}_{syn}, C_{syn}, type_{syn})$ décrivant la réalisation de surface de cette phrase comme suit :

- $\mathcal{A}_{syn} = \{str\}$.
- $C_{syn} = \{$`Jean, mange, une, pizza`$\}$.
- $type(\mathbf{c}) = str$ pour toute constante $\mathbf{c} \in C_{str}$.

La grammaire catégorielle abstraite produisant la réalisation de surface de cette phrase est finalement définie par $G_{syn} = (\Sigma_{tecto}, \Sigma_{syn}, \mathcal{L}_{syn}, s)$ telle que $\mathcal{L}_{syn}(a) = str$ pour tout $a \in A_{syn}$ et :

- $\mathcal{L}_{syn}(jean) =$ `Jean`
- $\mathcal{L}_{syn}(mange) = \lambda xy.y +$ `mange` $+ x$
- $\mathcal{L}_{syn}(une) = \lambda x.$`une` $+ x$
- $\mathcal{L}_{syn}(pizza) =$ `pizza`

où + est l'opération de concaténation de chaînes [1]. Un des aspects importants que cet exemple met en valeur est que l'ordre des mots dans une phrase est traité au niveau le plus bas possible, c'est-à-dire au niveau de la réalisation de surface de la phrase. Cette propriété différencie également fortement les ACGs des grammaires catégorielles, où les structures de dérivation (*i.e.* des preuves) et les structures de chaînes sont fortement liées, au point que l'ordre des mots se traitent au niveau dérivatif.

4.3.2 ACGs et formalismes grammaticaux

Dans cette section, nous montrons que les grammaires catégorielles abstraites présentent un cadre de travail très général, et en particulier, que de nombreux formalismes linguistiques connus peuvent y être représentés.

[1] cette notation est en fait une abréviation de la modélisation suivante, de chaînes de caractère par des λ-termes simplement typés : le type $o \rightarrow o$ est assigné à toute chaîne de caractères, sauf au mot vide ϵ, qui serait de type o ; une chaîne $a_1 \ldots a_n$ s'écrirait donc $\lambda x.(a_1(\ldots(a_n x)\ldots))$.

Les grammaires d'arbres adjoints

Un encodage des grammaires d'arbres adjoints a été donné dans [de Groote, 2002]. Nous apportons quelques modifications à cette construction : tout d'abord, remarquons que les grammaires d'arbres adjoints que nous avons définies considèrent toute feuille d'un arbre qui n'est pas une feuille d'adjonction, comme une feuille de substitution (dans certaines définitions, les feuilles pouvant être substituées doivent être spécifiées). Ensuite, nous donnons une grammaire catégorielle abstraite dont le langage objet est la frontière des arbres reconnus par la grammaire d'arbres adjoints initiale, contrairement à [de Groote, 2002], où deux étapes sont clairement dissociés : la reconnaissance des arbres dérivés, puis la construction des chaînes formant la frontière de ces derniers. La construction que nous donnons considère en fait un lexique qui est la composition des deux lexiques données dans [de Groote, 2002].

Soit une grammaire d'arbres adjoints $G = (N, \Sigma, S, \mathcal{I}, \mathcal{A})$. Nous construisons une signature de λ-termes $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ telle que, pour chaque non-terminal $B \in N$, il existe deux types atomiques B_I et B_A associés dans $\mathcal{A}_1$ (pour rendre compte, respectivement, de la substitution et de l'adjonction sur B) et, pour tout arbre $t \in \mathcal{I} \cup \mathcal{A}$ fait de m nœuds internes étiquetés par des non-terminaux $L_1, \ldots, L_m$ et n feuilles étiquetées par des non-terminaux $E_1, \ldots, E_n$, nous associons une constante $\mathbf{c}_t$ telle que :

- si t est un arbre initial de racine B, alors

$$type_1(\mathbf{c}_t) = E_{1,I} \to \ldots, E_{n,I} \to L_{1,A} \to \ldots L_{m,A} \to B_I$$

- si t est un arbre auxiliaire de racine B, alors

$$type_1(\mathbf{c}_t) = E_{1,I} \to \ldots, E_{n,I} \to L_{1,A} \to \ldots L_{m,A} \to B_A \to B_A$$

De plus, à tout symbole $B \in N$, nous associons une constante $\mathbf{c}_B$ telle que $type_1(\mathbf{c}_B) = B_A$.

Cette première signature décrit donc les contraintes d'adjonction et de substitution entre les noeuds d'un arbre, à travers la composition de constantes du λ-calcul simplement typé. Nous pouvons remarquer que c'est grâce aux contraintes de typage que l'on limite la composition des termes, et donc que l'on décrit les opérations pouvant être réalisées sur les nœuds des arbres. Enfin, les éléments terminaux ne sont pas pris en compte au niveau des termes de Σ_1, où l'on ne fait que décrire les mécanismes de dérivation.

Nous construisons donc une seconde signature de λ-termes $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ permettant d'obtenir la frontière des arbres reconnus par la grammaire d'arbres adjoints initiale. Les objets manipulés étant des chaînes de caractères, l'ensemble $\mathcal{A}_2$ est réduit à un singleton $\{str\}$. De plus, à chaque terminal $a \in \Sigma$, nous associons une constante $\mathbf{c}_a$ telle que $type_2(\mathbf{c}_a) = str$.

Enfin, nous donnons le lexique $\mathcal{L}$ associé à cette grammaire. Intuitivement, à un arbre initial nous faisons correspondre sa frontière, et donc une chaîne de caractères, alors qu'un arbre auxiliaire est vu comme une fonction qui transforme une chaînes de caractères en une nouvelle chaîne. Ainsi, pour tout non-terminal $B \in N$, nous obtenons $\mathcal{L}(type_1(B_I)) = str$ et $\mathcal{L}(type_1(B_A)) = str \to str$. Enfin, étant donné un arbre $t \in \mathcal{I} \cup \mathcal{A}$, le terme $\mathcal{L}(\mathbf{c}_t)$ décrit la chaîne de caractères associée, par concaténation des chaînes associées aux branches de la racine de l'arbre, de gauche à droite. Ainsi, dans un premier temps, nous construisons le terme $\phi(t)$ associé un certain arbre t par induction sur t :

- si $t = a \in \Sigma$ alors $\phi(t) = \mathbf{c}_a$.
- si $t = B*$ (*i.e.* le pied d'un arbre adjoint) tel que $B \in N$, alors $\phi(t) = Fx$, où F et x sont des variables fraîches.
- si $t = B \in N$ est un nœud de rang n dont les fils sont $t_1, \ldots, t_n$, alors

$$\phi(t) = F(\phi(t_1) + \ldots + \phi(t_n))$$

où F est une variable fraîche.

Le lexique transforme une constante $\mathbf{c}$ de C_1 de la manière suivante :

- si $\mathbf{c}$ est associée à un arbre initial $t \in \mathcal{I}$, alors $type_1(\mathbf{c}_t) = E_{1,I} \to \ldots, E_{n,I} \to L_{1,A} \to \ldots L_{m,A} \to R_I$; nous construisons

$$\mathcal{L}(\mathbf{c}) = \lambda F_1 \ldots F_n G_1 \ldots G_m.\phi(t)$$

  où, pour tout $i \in \{1,\ldots,n\}$, la variable F_i apparaît dans $\phi(t)$ et est associée à la feuille $E_{i,I}$; de même, pour tout $j \in \{1,\ldots,m\}$, G_j a une occurrence libre dans $\phi(t)$ et est associée au nœud interne $L_{j,A}$.
- si $\mathbf{c}$ est associé à un arbre auxiliaire $t \in \mathcal{A}$, alors

$$\mathcal{L}(\mathbf{c}) = \lambda F_1 \ldots F_n G_1 \ldots G_m x.\phi(t)$$

  où les mêmes remarques que précédemment s'appliquent aux variables $F_1,\ldots,F_n$ et $G_1,\ldots,G_m$, et x est la variable associée au pied de t dans $\phi(t)$.

On remarquera que la grammaires catégorielle abstrait ainsi construite, est une ACG du second-ordre linéaire.

Afin d'illustrer cette construction, considérons la grammaire d'arbres adjoints de la Figure 4.2. La grammaire catégorielle abstraite $\mathcal{G}$ associée à cette grammaires est $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S_I)$ où $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ est construite comme suit :

- $\mathcal{A}_1 = \{S_I, S_A, NP_I, NP_A, VP_I, VP_A, ADV_I, ADV_A\}$
- $C_1 = \{\mathbf{c}_{t-jean}, \mathbf{c}_{t-boit}, \mathbf{c}_{t-lentement}, \mathbf{c}_{NP}, \mathbf{c}_S, \mathbf{c}_{VP}, \mathbf{c}_V, \mathbf{c}_{ADV}\}$
- $type_1(\mathbf{c}_{t-jean}) = NP_A \to NP_I$, $type_1(\mathbf{c}_{t-boit}) = NP_I \to S_A \to VP_A \to V_A \to S_I$, $\mathbf{c}_{t-lentement} = VP_A \to ADV_A \to VP_A \to VP_A$
- $type_1(\mathbf{c}_{NP}) = NP_A$, $type_1(\mathbf{c}_S) = S_A$, $type_1(\mathbf{c}_{VP}) = VP_A$, $type_1(\mathbf{c}_V) = V_A$, $type_1(\mathbf{c}_{ADV}) = ADV_A$

La signature $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ est construite de la manière suivante :

- $\mathcal{A}_2 = \{str\}$
- $C_2 = \{\texttt{jean}, \texttt{boit}, \texttt{lentement}\}$
- pour toute constante $\mathbf{c}$ de C_2, $type_2(\mathbf{c}) = str$

Enfin, le lexique projette types et constantes de la manière suivante :

- $\mathcal{L}(S_I) = \mathcal{L}(VP_I) = \mathcal{L}(NP_I) = \mathcal{L}(V_I) = \mathcal{L}(ADV_I) = str$
- $\mathcal{L}(S_A) = \mathcal{L}(VP_A) = \mathcal{L}(NP_A) = \mathcal{L}(V_A) = \mathcal{L}(ADV_A) = str \to str$
- $\mathcal{L}(\mathbf{c}_S) = \mathcal{L}(\mathbf{c}_{NP}) = \mathcal{L}(\mathbf{c}_{VP}) = \mathcal{L}(\mathbf{c}_V) = \mathcal{L}(\mathbf{c}_{ADV}) = \lambda x.x$
- $\mathcal{L}(\mathbf{c}_{t-jean}) = \lambda F_{NP}.F_{NP}\texttt{jean}$
- $\mathcal{L}(\mathbf{c}_{t-boit}) = \lambda x_{NP} F_S F_{VP} F_V.F_S(x_{NP} + F_{VP}(F_V\texttt{boit}))$
- $\mathcal{L}(\mathbf{c}_{t-lentement}) = \lambda F_{VP} F_{ADV} F_{VP*} x_{VP^*}.F_{VP}(F_{VP*} x_{VP^*} + F_{ADV}\texttt{lentement})$

On remarquera l'utilisation particulière des constantes $\mathbf{c}_B$ de C_1 associée à chaque non-terminal de N ; en effet, par application du lexique, ces constantes deviennent le terme identité $\lambda x.x$; ce terme est utilisé afin de n'effectuer aucune opération sur certains nœuds internes (qui sont susceptibles de subir une opération d'adjonction). Ainsi, pour la phrase "Jean boit lentement", nous obtenons le terme $\mathbf{c}_{t-boit}(\mathbf{c}_{t-jean}\mathbf{c}_{NP})\mathbf{c}_S(\mathbf{c}_{t-lentement}\mathbf{c}_{VP}\mathbf{c}_{ADV}\mathbf{c}_{VP})\mathbf{c}_V$ du langage abstrait de cette ACG.

Un des avantages majeurs de l'encodage des grammaires d'arbres adjoints en grammaires catégorielles abstraites est celui de donner aux grammaires d'arbres adjoints, une interface vers la sémantique de Montague. Cet avantage fut exploité dans [Pogodalla, 2004] afin de résoudre certains problèmes connus comme difficiles dans le cadre de la sémantique associée aux TAGs.

Les grammaires non-contextuelles multiples

Nous donnons à présent un encodage des grammaires non-contextuelles multiples de chaînes comme des grammaires catégorielles abstraites du second-ordre. Cette transformation est extraite de [de Groote and Pogodalla, 2004] pour les systèmes de réécriture linéaire non-contextuelle, un formalisme équivalent aux MCFGs. L'idée de ces grammaires est de coder un tuple de chaînes $w_1, \ldots, w_n$ comme un λ-terme linéaire $\lambda P.PM_1 \ldots M_n$, où $M_1, \ldots, M_n$ s'interprète comme $w_1, \ldots, w_n$ respectivement.

Soit une grammaire non-contextuelle multiple $G = (\Sigma, N, P, S)$; nous construisons une ACG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ de telle sorte que le langage abstrait de $\mathcal{G}$ corresponde aux arbres de dérivation des mots de $L(G)$, et les termes de $O(\mathcal{G})$ aux mots reconnus. Ainsi, la signature $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ est telle que :

- pour tout non-terminal $B \in N$, il existe un type B dans $\mathcal{A}_1$.
- à toute règle $A(t_1, \ldots, t_r) \Rightarrow A_1(x_{1,1}, \ldots, x_{1,r_1}), \ldots, A_n(x_{n,1}, \ldots, x_{n,r_n})$ de P, nous associons une constante $\mathbf{c} \in C_1$ telle que :

$$type_1(\mathbf{c}) = A_1 \to \ldots \to A_n \to A$$

La signature $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ est, quant à elle, construite de la manière suivante :

- $\mathcal{A}_2 = \{str\}$, str étant le type associé aux chaînes de caractères.
- pour tout terminal $a \in \Sigma$ dans la MCFG, il existe une constante $\mathbf{c}_a$ qui appartient à $\Lambda_{str}(\Sigma_2)$

Le lexique associé à une MCFG est ensuite défini de la manière suivante :

Sur les types : pour tout non-terminal $B \in N$ de rang r

$$\mathcal{L}(B) = \underbrace{str \to \ldots \to str}_{r+1\text{fois}} \to str$$

Sur les termes étant donnée une règle $p = A(t_1, \ldots, t_r) \Rightarrow A_1(x_{1,1}, \ldots, x_{1,r_1}), \ldots, A_n(x_{n,1}, \ldots, x_{n,r_n})$, et la constante $\mathbf{c}$ associée dans la signature abstraite,

$$\mathcal{L}(\mathbf{c}) = \lambda P_1 \ldots P_n P.P_1(\lambda \overline{x_1}.P_2(\lambda \overline{x_2}. \ldots P_n(\lambda \overline{x_n}.PM_1 \ldots M_r)))$$

où, pour tout $1 \leq i \leq n$, $\overline{x_i} = x_{i,1} \ldots x_{i,r_i}$ et pour tout $1 \leq j \leq r$, M_j s'interprète comme la chaîne t_1.

Afin d'illustrer cette transformation, considérons la MCFG $G = \{\Sigma, N, P, S\}$ dont les règles de dérivation sont les suivantes :

- $S(x_1 c x_2) \Rightarrow A(x_1, x_2)$.
- $A(ax_1 b, dx_2 e) \Rightarrow A(x_1, x_2)$.
- $A(ab, de)$.

Le langage reconnu par cette grammaire est donc $L(G) = a^n b^n c d^n e^n$, où $n > 0$.

D'après la méthode spécifiée ci-haut, nous obtenons une ACG du second ordre non-linéaire $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ telle que $\Sigma_1 = (\mathcal{A}_1, C_1, type_1)$ est définie par :

- $\mathcal{A}_1 = \{S, A\}$
- $C_1 = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$
- $type_1 = \{\mathbf{c_1} \mapsto A \to S, \mathbf{c}_2 \mapsto A \to A, \mathbf{c}_3 : A\}$

Et le langage abstrait de $\mathcal{G}$ est donc $\mathbf{c}_1(\mathbf{c}_2^n \mathbf{c}_3)$, où $n \geq 0$.

La signature objet est $\Sigma_2 = (\mathcal{A}_2, C_2, type_2)$ où :

- $\mathcal{A}_2 = \{str\}$
- $C_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$
- $type_2(\mathbf{c}) = str$ pour toute constante $\mathbf{c} \in C_2$.

Finalement, le lexique est donné par $\mathcal{L}(S) = str \to str$ et $\mathcal{L}(A) = str \to str \to str$; de plus :

- $\mathcal{L}(\mathbf{c}_1) = \lambda QP.Q(\lambda x_1 x_2.P(x_1 + \mathtt{c} + x_2))$
- $\mathcal{L}(\mathbf{c}_2) = \lambda QP.Q(\lambda x_1 x_2.P(\mathtt{a} + x_1 + \mathtt{b})(\mathtt{d} + x_2 + \mathtt{e}))$
- $\mathcal{L}(\mathbf{c}_3) = \lambda P.P(\mathtt{a} + \mathtt{b})(\mathtt{d} + \mathtt{e})$

On peut aisément vérifier que le langage reconnu par cette ACG est l'ensemble des termes de la forme $\lambda P.PM$ tel que M s'interprète comme la chaîne reconnue par la MCFG associée à cette construction.

Enfin, on remarquera à nouveau que cette grammaire est une grammaire non-contextuelle linéaire.

Nous montrons à présent que ces grammaires définissent en réalité une hiérarchie sur les langages de chaînes, et les langages d'arbres.

4.3.3 Une hiérarchie de langages

Les grammaires catégorielles abstraites sont en fait un formalisme très général. En effet, selon l'ordre de la signature abstraite et l'ordre de la signature objet, que nous introduisons à la suite, il est possible de définir une hiérarchie de langages. Dans cette section, les grammaires catégorielles abstraites que nous considérons sont des ACGs linéaires.

Définition 4.3.3.1. *Soit une ACG* $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$*, où* $\Sigma_1 = (A_1, C_1, type_1)$ *; nous appelons* ordre lexical de G *la valeur* $\mathtt{ord}(\mathcal{L}(\Sigma_1)) = max_{a \in A_1} \mathtt{ord}(\mathcal{L}(a))$

Définition 4.3.3.2. *Une grammaire catégorielle abstraite* $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ *appartient à la famille* $\mathcal{G}(n,m)$*, si* $\mathtt{ord}(\Sigma_1) \leq n$ *et* $\mathtt{ord}(\mathcal{L}(\Sigma_1)) \leq m$*.*

En particulier, nous dirons qu'une ACG $G \in \mathcal{G}(n,m)$ est d'ordre n. Tout d'abord, nous pouvons remarquer les grammaires catégorielles abstraites d'ordre 1 correspondent à des langages finis. Peu de choses sont connues sur les grammaires catégorielles dont l'ordre est supérieur à 3 ; nous nous intéresserons par la suite uniquement aux ACGs d'ordre 2. Dans ce cas, comme nous l'avons remarqué dans la section précédente, les termes de la signature abstraite sont des arbres ; les ACGs du second-ordre reconnaissent donc des langages abstraits qui sont les arbres de dérivations du langage objet reconnu.

ACG		Langages de chaînes
$\mathcal{G}(2,1)$	=	Réguliers
$\mathcal{G}(2,2)$	=	Non-contextuels
$\mathcal{G}(2,3)$	=	Non-contextuels multiples avec imbrication
$\bigcup_{m \geq 4} \mathcal{G}(2,m)$	=	Non-contextuels multiples

FIG. 4.3 – Hiérarchie des langages de chaînes reconnus par des ACGs du second-ordre linéaires

Une hiérarchie de classes a été proposée pour la définition initiale des ACGs du second-ordre de [de Groote, 2001], c'est-à-dire, pour ce que nous appelons les ACGs linéaires du second-ordre, en fonction de l'ordre lexical des grammaires. Tout d'abord, nous considérons les langages de chaînes ainsi reconnus, dans le tableau de la Figure 4.3. Il est important de remarquer que la hiérarchie ainsi exhibée met en évidence une nouvelle classe de langages : *les langages non-contextuels multiples bien imbriqués*, que nous noterons par *wn-MCFL* pour *well-nested multiple context-free languages*. Des formalismes tels que les grammaires d'arbres adjoints font en fait partie de cette classe de langages, qui a des propriétés de robustesse intéressantes [Kanazawa, 2009, Kanazawa and Salvati, 2010]. De

fait, la question se pose de savoir si cette classe de langages ne correspond pas mieux à la description de phénomènes syntaxiques des langues naturelles que la classe des langages faiblement sensibles au contexte. En particulier, le langage $\texttt{MIX} = \{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$ [Salvati, 2011], est reconnu par une grammaire non-contextuelle multiple, et il reste à démontrer qu'il n'est pas un langage non-contextuel multiple bien imbriqué. Or, ce langage ne rend, a priori, compte d'aucun phénomène linguistique.

Par ailleurs, il a été montré dans [Salvati, 2006] que tout langage reconnu par une ACG linéaire d'ordre supérieur à 4 est également reconnu par une ACG linéaire d'ordre 4. La hiérarchie de langages ainsi obtenue n'est donc pas infinie.

ACG		Langages d'arbres
$\mathcal{G}(2,1)$	=	Réguliers
$\mathcal{G}(2,2)$	=	Linéaires Non-Contextuels
$\mathcal{G}(2,3)$	=	Multiples et Réguliers
$\bigcup_{m\geq 4}\mathcal{G}(2,m)$	=	Multiples et linéaires non-contextuels

FIG. 4.4 – Hiérarchie des langages d'arbres reconnus par des ACGs du second-ordre linéaires

Enfin, il est possible d'étudier non pas les langages de chaînes mais les langages d'arbres reconnus pas des ACGs linéaires. À nouveau, les grammaires catégorielles abstraites font apparaître une distinction claire entre *langages réguliers d'arbres* (dont la frontière donne un langage non-contextuel de chaînes), *langages linéaires et non-contextuels d'arbres* (dont la frontière donne un langage faiblement sensible au contexte avec imbrication), *les langages multiples et réguliers d'arbres* et enfin *les langages multiples, linéaires et non-contextuels d'arbres* (voire Figure 4.4).

4.3.4 Traitement de certains phénomènes linguistiques

Nous montrons à présent comment certains phénomènes linguistiques peuvent être traités par les grammaires catégorielles abstraites, mettant ainsi en avant certaines propriétés obtenues grâce à la différenciation entre phénogrammaires et tectogrammaires, ainsi que l'utilisation du λ-calcul sur les différentes représentations d'une phrase.

Dans ce qui suit, nous proposons des pistes quant à l'utilisation des grammaires catégorielles abstraites pour traiter certains phénomènes ; une solution complète du traitement de ces derniers demanderait un travail supplémentaire qui n'a, pour l'instant, pas été réalisé.

Discontinuité

Le traitement de phénomènes syntaxiques discontinus a été à la base de la création de formalismes étendant le calcul de Lambek. Ainsi, les grammaires multi-modales [Moortgat, 1996, Moortgat, 1997] ou le calcul de déplacement [Morrill and Solias, 1993, Morrill et al., 2010] permettent de traiter certains cas de discontinuité. Comme nous l'avons déjà exposé, le calcul de Lambek a pour inconvénient de traiter de l'ordre des mots au niveau des dérivations. Les grammaires catégorielles abstraites diffèrent du calcul de Lambek et de ses extensions sur ce point. Nous montrons ici comment traiter deux phénomènes particuliers de discontinuité parmi les exemples donnés dans [Morrill et al., 2010] : les expressions figées discontinues et l'ellipse verbale.

Dans le premier cas, considérons la phrase "John gives Mary the cold shoulder" (*John tourne le dos à Mary*). Nous avons ici une expression figée "to give someone the cold shoulder", pour lequel

nous pouvons donner l'interprétation sémantique suivante : $\lambda PQ.P(\lambda x.Q(\mathbf{g}-\mathbf{cold}-\mathbf{shoulder}\ x\ y))$, et la représentation de surface suivante : $\lambda xy.x + \texttt{give} + y + \texttt{the} + \texttt{cold} + \texttt{shoulder}$. De manière plus générale, l'ACG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$ permettant de réaliser la structure de surface de cette phrase peut être définie par :

1. $\Sigma_1 = \begin{cases} \{NP, S\}, \\ \{\mathbf{c}_{john}, \mathbf{c}_{mary}, \mathbf{c}_{cold-shoulder}\}, \\ \{\mathbf{c}_{john} \mapsto NP, \mathbf{c}_{mary} \mapsto NP, \mathbf{c}_{cold-shoulder} \mapsto NP \to NP \to S\} \end{cases}$
2. $\Sigma_1 = \begin{cases} \{str\}, \\ \{\texttt{john}, \texttt{mary}, \texttt{give}, \texttt{the}, \texttt{cold}, \texttt{shoulder}\}, \\ \mathbf{c} \mapsto str \text{ pour toute constante de la signature.} \end{cases}$
3. $\mathcal{L} = \begin{cases} NP \mapsto str, S \mapsto str, \\ \mathbf{c}_{john} \mapsto \texttt{John}, \mathbf{c}_{mary} \mapsto \texttt{Mary}, \mathbf{c}_{cold-shoulder} \mapsto \lambda xy.x + \texttt{give} + y + \texttt{the} + \texttt{cold} + \texttt{shoulder}\} \end{cases}$

Il est aisé de vérifier que le langage abstrait de cette grammaire contient $\mathbf{c}_{cold-shoulder}\mathbf{c}_{john}\mathbf{c}_{mary}$, dont l'image par $\mathcal{L}$ et par β-réduction correspond à la phrase "John gives Mary the cold shoulder".

Le deuxième phénomène linguistique pour lequel nous proposons une solution via les grammaires catégorielles abstraites est l'ellipse verbale. En guise d'exemple, prenons la phrase "John eats salmon and Mary, beef". La difficulté est alors d'obtenir une représentation sémantique où la constante associée au verbe "to eat" apparaît deux fois, sans que cela soit le cas sur la réalisation de surface. Nous utilisons ici la possibilité de rendre compte de phénomènes de copie à travers le λ-calcul, en construisant l'ACG $\mathcal{G}_{syn} = \{\Sigma, \Sigma_{syn}, \mathcal{L}_{syn}\}$ de la manière suivante :

1. $\Sigma = \begin{cases} \{NP, S\}, \\ \{\mathbf{c}_{john}, \mathbf{c}_{eat}, \mathbf{c}_{mary}, \mathbf{c}_{beef}, \mathbf{c}_{salmon}, \mathbf{c}_{and}\}, \\ \{\mathbf{c}_{john} \mapsto NP, \mathbf{c}_{eat} : NP \to NP \to S, \mathbf{c}_{mary} \mapsto NP, \mathbf{c}_{beef} \mapsto NP, \mathbf{c}_{salmon} \mapsto NP, \\ \mathbf{c}_{and} \mapsto (NP \to NP \to S) \to NP \to NP \to NP \to NP \to S\} \end{cases}$
2. $\Sigma_{syn} = \begin{cases} \{str\}, \\ \{\texttt{John}, \texttt{eat}, \texttt{Mary}, \texttt{beef}, \texttt{salmon}\}, \\ \{\mathbf{c} \mapsto str, \text{ pour toute constante de } \Sigma_{syn}\} \end{cases}$
3. $\mathcal{L}(NP) = \mathcal{L}(S) = str$
4. $\mathcal{L}(\mathbf{c}_{eat}) = \lambda xy.x + \texttt{eat} + y$, $\mathcal{L}(\mathbf{c}_{and}) = \lambda xy_1z_1y_2z_2.x + y_1 + z_1 + \texttt{and} + y_2 +, +z_2)$, $\mathcal{L}(\mathbf{c}_{john}) = \texttt{John}$, $\mathcal{L}(\mathbf{c}_{mary}) = \texttt{Mary}$, $\mathcal{L}(\mathbf{c}_{beef}) = \texttt{beef}$, $\mathcal{L}(\mathbf{c}_{salmon}) = \texttt{salmon}$

Ainsi, le terme $M = \mathbf{c}_{and}\mathbf{c}_{john}\mathbf{c}_{salmon}\mathbf{c}_{mary}\mathbf{c}_{beef}$ appartient au langage abstrait de cette grammaire, et $\mathcal{L}(M)$ correspond à la phrase prise en exemple "John eats salmon and Mary, beef.".

Du côté sémantique, nous construisons l'ACG $\mathcal{G}_{sem} = (\Sigma, \Sigma_{sem}, \mathcal{L}_{sem}, S)$ de telle sorte que :

1. $\Sigma_{sem} = \begin{cases} \{e, t\}, \\ \{\mathbf{j}, \mathbf{e}, \mathbf{m}, \mathbf{b}_{part}, \mathbf{s}_{part}\}, \\ \{\mathbf{j} \mapsto e, \mathbf{m} \mapsto e, \mathbf{e} : e \to (e \to t) \to t, \mathbf{b}_{part} : e, \mathbf{s}_{part} : e\} \end{cases}$
2. $\mathcal{L}(N) = e \to t, \mathcal{L}(NP) = e, \mathcal{L}(S) = t$
3. $\mathcal{L}(\mathbf{c}_{john}) = \lambda P.P\mathbf{j}$, $\mathcal{L}(\mathbf{c}_{mary}) = \lambda P.P\mathbf{m}$, $\mathcal{L}(\mathbf{c}_{beef}) = \lambda P.P\mathbf{b}_{part}$, $\mathcal{L}(\mathbf{c}_{salmon}) = \lambda P.P\mathbf{s}_{part}$, $\mathcal{L}(\mathbf{c}_{eat}) = \lambda PQ.P(\lambda x.\mathbf{e}xQ)$, $\mathcal{L}(\mathbf{c}_{and} = \lambda PQ_1R_1Q_2R_2.\wedge(PQ_1R_1)(PQ_2R_2)$

Ainsi, pour le terme M cité ci-dessus, le terme sémantique $\mathcal{L}_{sem}(M)$ se β-réduit en

$$\wedge(\mathbf{e}\ \mathbf{j}\ \mathbf{s}_{part})(\mathbf{e}\ \mathbf{m}\ \mathbf{b}_{part})$$

Premièrement, remarquons que Σ est une signature du troisième ordre, bien que le terme M qui nous intéresse dans Σ ait une structure arborescente. Par ailleurs, nous avons ici utilisé un mécanisme de copie ; en effet, dans le terme $\mathcal{L}(\mathbf{c}_{and})$, nous avons copié la variable P en deux endroits. La variable P, de plus, est d'ordre 2. Comme nous le verrons au cours du chapitre suivant, des méthodes d'analyse efficaces pour les ACGs ont été données pour des grammaires où il est possible de réaliser de la copie uniquement sur les variables d'ordre 1, dans le langage objet.

Par ailleurs, cette solution est peu élégante et n'est pas extensible : si l'ellipse se réalise sur $n > 2$ syntagmes, nous ne pouvons en rendre compte de manière fine avec une telle modélisation. Une manière de combler ce défaut serait, par exemple, de traiter l'ellipse verbale d'une manière identique à l'anaphore, sous certaines conditions (par exemple, qu'un verbe apparaisse dans la même phrase, dans une position précédente) ; ainsi, lorsqu'une phrase à laquelle manque un verbe est rencontrée, le verbe serait à chercher dans le contexte précédent cette phrase. Un tel traitement peut donc être étudié dans le cadre de la sémantique avec continuations, par exemple.

Notons qu'il serait intéressant d'explorer la traitement dans les ACGs d'autres phénomènes linguistiques tels que les réflexifs, et d'autres problèmes liés à la coordination et aux ellipses. Des phénomènes d'extraction ont également été traités dans [Pogodalla and Pompigne, 2011] en utilisant une extension des grammaires catégorielles abstraites avec types dépendants [de Groote et al., 2007], permettant ainsi un contrôle fin sur les dérivations.

Accords syntaxiques

Nous montrons à présent comment intégrer certains aspects morpho-syntaxiques dans les ACGs, en utilisant l'effacement dans le λ-calcul. Les problèmes d'accords apparaissent naturellement en Français sur les genres et nombres ; par exemple, l'adjectif "grand", se décline de différentes façons selon le nom auquel il est rattaché. Il en est de même pour les verbes, selon le sujet qui leur est associé. Dans le cadre de formalismes linguistiques basés sur un système de typage, les solutions proposées font appel à l'introduction de systèmes de typage plus compliqués que le système de typage simple, et plus exactement les types dépendants. Nous proposons ici de rester dans le cadre du λ-calcul simplement typé, afin de préserver la simplicité du système de typage, et les propriétés d'efficacité de l'analyse, comme nous le verrons par la suite.

Dans ce qui suit, nous noterons par $a^n \to a$ le type $\underbrace{a \to \ldots \to a}_{n \text{ fois}} \to a$. De même, nous noterons par $\pi_{i,j}$, où $0 < i \leq j$ sont des entiers, le terme $\lambda x_1 \ldots x_j.x_i$

Prenons pour exemple, la phrase "Jean et Marie mangent". Nous pouvons ainsi définir la grammaire catégorielle abstraite suivante $\mathcal{G} = (\Sigma, \Sigma_{syn}, \mathcal{L}, S)$, permettant de gérer les accords :

1. $\Sigma = \begin{cases} \{NP, S\}, \\ \{\mathbf{c}_{jean}, \mathbf{c}_{et}, \mathbf{c}_{marie}, \mathbf{c}_{manger}, \}, \\ \{\mathbf{c}_{jean} \mapsto NP, \mathbf{c}_{et} \mapsto NP \to NP \to NP, \mathbf{c}_{marie} : NP, \mathbf{c}_{manger} : NP \to S, \} \end{cases}$
2. $\Sigma_{syn} = \begin{cases} \{str\}, \\ \{\texttt{Jean, Marie, mange, manges, mangeons, mangez, mangent, et}\}, \\ \{\text{pour toute constante } \mathbf{c}, \mathbf{c} \mapsto str\} \} \end{cases}$

3. $\mathcal{L} = \begin{cases} NP \mapsto (((str \to str)^6) \to str) \to str) \to str, S \mapsto str, \\ \mathbf{c}_{jean} \mapsto \lambda P.P\pi_{3,6}\texttt{Jean}, \\ \mathbf{c}_{marie} \mapsto \lambda P.P\pi_{3,6}\texttt{Marie}, \\ \mathbf{c}_{manger} \mapsto \lambda P.P(\lambda x_1 x_2.x_2 + (x_1 M_1 M_2 M_3 M_4 M_5 M_5)), \\ \mathbf{c}_{et} \mapsto \lambda P P_1 P_2.P\pi_{6,6}((P_1(\lambda x_1 x_2.x_2)) + \texttt{et} + (P_2(\lambda x_1 x_2.x_2))) \end{cases}$

où

(a) $M_1 = M_3 = \lambda x.x + \texttt{mange}$

(b) $M_2 = \lambda x.x + \texttt{manges}$

(c) $M_4 = \lambda x.x + \texttt{mangeons}$

(d) $M_5 = \lambda x.x + \texttt{mangez}$

(e) $M_6 = \lambda x.x + \texttt{mangent}$

Ainsi, pour le terme $\mathbf{c}_{manger}\mathbf{c}_{marie}$ du langage abstrait de cette grammaire, nous obtenons la réalisation de surface "Jean+mange". Par contre, pour le terme $\mathbf{c}_{manger}$ (**et** $\mathbf{c}_{marie}$ $\mathbf{c}_{jean}$), nous obtenons la réalisation de surface "Marie+et+Jean+mangent".

Les accords sont ici gérés grâce à l'effacement dans le λ-calcul : en effet, nous pouvons voir que le terme $\mathcal{L}(\mathbf{c}_{manger})$ contient une liste de termes (les termes $M_1, \ldots, M_6$) et la variable x_1 sert de sélecteur sur cette liste, afin de sélectionner le verbe en accord. Le sélecteur est donc fonction du sujet associé au verbe, et pour les termes $\mathcal{L}(\mathbf{c}_{jean})$ et $\mathcal{L}(\mathbf{c}_{marie})$, il nous faut donc sélectionner le verbe sous sa forme à la troisième personne du singulier (grâce à la projection $\pi_{3,6}$) ; au cas où la coordination est utilisée, la projection nécessaire devient $\pi_{6,6}$. Bien entendu, un tel exemple ne permet pas de gérer correctement les accords pour une phrase telle que "Jean et moi mangeons", ce qui nécessiterait un raffinement de la gestion des projections.[2]

Un mécanisme identique peut être mis en place pour la gestion des cas dans les langues flexionnelles telles que l'Allemand ou le Néerlandais, le contrôle étant, cette fois, donné au verbe, qui sélectionnerait la forme morpho-syntaxique correcte sur les noms selon que ces derniers seraient placés en tant que sujet, complément d'objet ou complément indirect dans la phrase.

Remarque 4.3.4.1. La grammaire ainsi définie est une ACG du second-ordre affine.

Facettes sémantiques

L'effacement dans le λ-calcul, comme nous venons de le voir pour la gestion des aspects morpho-syntaxiques, peut être utilisé afin de sélectionner les termes désirés lorsqu'une même unité lexicale prend différentes formes. En sémantique, les différents aspects d'une entité peuvent correspondre aux facettes lexicales de celle-ci. Ainsi, par exemple, "Hamlet" peut désigner un livre sous sa forme d'objet (par exemple dans la phrase "Jean déchire Hamlet"), ou un livre sous sa forme de contenu informatif (par exemple dans "Hamlet se lit facilement") ou encore le personnage Hamlet (comme par exemple dans "Hamlet est le prince du Danemark"). L'étude de ces différentes facettes et de leur accès est une branche de la linguistique appelée la *sémantique lexicale*, décrite largement dans [Pustejovsky, 1991]. Des travaux récents faisant appel au λ-calcul d'ordre supérieur ont également décrit une solution à l'accès à ces facettes tout en préservant une sémantique compositionnelle [Bassac et al., 2010] ; dans

[2] On peut, par exemple, avoir deux types de projections, sélectionnant la personne pour la première, et le genre pour la seconde ; par un encodage des booléens ($\top$ est représenté par $\lambda xy.x$ et $\bot$ par $\lambda xy.y$ par exemple) et des structures conditionnelles ($\lambda Fxy.Fxy$ pour "if F then x else y") dans le λ-calcul, il serait alors possible de sélectionner la forme correcte du verbe.

ces travaux, la sélection de facette et la vérification de la validité de cette sélection se fait au niveau du typage des termes ; la solution que nous proposons ici réalise cette gestion au niveau du terme.

Nous prenons, cette fois-ci, la phrase “John lit Hamlet” comme exemple, et supposons que la tectogrammaire permette d’obtenir l’arbre associé au terme $\mathbf{c}_{lit}\mathbf{c}_{jean}\mathbf{c}_{hamlet}$ dans la signature de termes suivante, faisant office de tectogrammaire pour cette phrase :

$$\Sigma = \begin{cases} \{NP, S\}, \\ \{\mathbf{c}_{jean}, \mathbf{c}_{lit}, \mathbf{c}_{hamlet}\}, \\ \{\mathbf{c}_{jean} \mapsto NP, \mathbf{c}_{lit} \mapsto NP \to NP \to S, \mathbf{c}_{hamlet} \mapsto NP\} \end{cases}$$

Nous proposons alors de décrire les différentes facettes sémantiques des noms propres par une liste de facettes ; ainsi, nous construisons la signature de termes suivante pour la représentation sémantique de notre exemple :

$$\Sigma_{sem} = \begin{cases} \{e, t\}, \\ \{\mathbf{jean}_{pers}, \mathbf{lit}, \mathbf{hamlet}_{pers}, \mathbf{hamlet}_{obj}, \mathbf{hamlet}_{info}, \mathbf{undefined}\}, \\ \{\mathbf{jean}_{pers} \mapsto e, \mathbf{undefined} \mapsto e, \\ \mathbf{lit} \mapsto e \to e \to t, \\ \mathbf{hamlet}_{pers} \mapsto e, \mathbf{hamlet}_{obj} \mapsto e, \mathbf{hamlet}_{info} \mapsto e\} \end{cases}$$

Remarque 4.3.4.2. La constante **undefined** correspond à un cas où une facette sémantique n’est pas définie pour l’entrée lexicale. Dans notre exemple, il n’existe pas de facette d’objet physique ou de contenu informationnel pour l’entrée Jean. Une analyse générant une forme sémantique ou **undefined** apparaît serait donc rejetée.

Enfin, le lexique est finalement défini par :

$$\mathcal{L} = \begin{cases} NP \mapsto (e^3 \to e) \to (e \to t) \to t, S \mapsto t\}, \\ \mathbf{c}_{jean} \mapsto \lambda QP.P(Q\mathbf{jean}_{pers}\ \mathbf{undefined}\ \mathbf{undefined}), \\ \mathbf{c}_{hamlet} \mapsto \lambda QP.P(Q\mathbf{hamlet}_{pers}\ \mathbf{hamlet}_{obj}\mathbf{hamlet}_{info}), \\ \mathbf{lit} \mapsto \lambda PQ.P\pi_{1,3}(\lambda x.Q\pi_{3,3}(\lambda y.\mathbf{lit}\ x\ y)), \end{cases}$$

Nous obtenons alors $\mathcal{L}(\mathbf{c}_{lit}\mathbf{c}_{jean}\mathbf{c}_{hamlet}) \to^*_\beta \mathbf{lit}\ \mathbf{jean}_{pers}\ \mathbf{hamlet}_{info}$.

À nouveau, nous ne donnons ici, pas une solution complète à des phénomènes complexes de la sémantique compositionnelle, tels que la possibilité de rejeter des phrases comme : “Jean mangea un saumon rapide et délicieux”.

Remarque 4.3.4.3. Cette grammaire est une ACG du second-ordre affine.

4.4 Conclusion

Au cours de chapitre, nous avons présenté différentes propositions pour le traitement automatisé de la syntaxe et de la sémantique des langues naturelles. Dans un premier temps, nous avons introduit le concept de grammaire générative, et certains formalismes étendant le concept de grammaires non-contextuelles de chaînes (tels que les grammaires d’arbres adjoints ou les grammaires non-contextuelles multiples), permettant de rendre compte de phénomènes syntaxiques complexes.

Les langages générés par de tels grammaires appartiennent à une classe de langages appelées faiblement sensibles au contexte, se situant entre les langages non-contextuels et les langages sensibles au contexte dans la hiérarchie de Chomsky.

Par ailleurs, nous avons présenté les apports de l'introduction du λ-calcul simplement typé à l'étude de la sémantique des langues, que nous devons à Richard Montague. Dans le théorie de ce dernier, les bases d'une construction algébrique de l'interface entre syntaxe et sémantique ont été posées, définissant ces deux différents niveaux du langage par des algèbres, et l'interface entre ces niveaux étant réalisés par application de morphismes.

Les grammaires catégorielles abstraites se nourrissent de ces idées ; elles peuvent en effet être vues comme une implémentation complète de la théorie montagovienne : les algèbres sont des algèbres de λ-termes simplement typés, et les morphismes, des morphismes sur ces algèbres. Qui plus est, l'interface entre réalisation de surface et réalisation sémantique d'une phrase est enrichie d'une algèbre supplémentaire, représentant les dérivations des phrases. Nous avons pu voir que ce formalisme défini un cadre général dans lequel étudier l'interface syntaxe-sémantique, dans le sens où nombreux formalismes grammaticaux connus peuvent être vus comme des grammaires catégorielles abstraites. De cette manière, les ACGs permettent de définir une hiérarchie de classes de langage qui permet, en particulier, de dissocier clairement les langages générés par des TAGs de ceux générés par des MCFGs.

Finalement, nous avons mis en avant certains apports des ACGs quant à l'étude de phénomènes complexes, tels que certains cas de discontinuité (grâce au fait que l'ordre des mots peut n'être traité qu'au niveau de la réalisation de surface dans les ACGs), les accords syntaxiques, ou encore l'ajout d'information de la sémantique lexicale. Ces apports sont, pour le moment, à relativiser, car l'étude de ces phénomènes méritent d'être approfondie, et certaines solutions que nous proposons sont rudimentaires. Néanmoins, elles permettent de mettre en valeur, par exemple, l'apport de l'effacement dans les ACGs.

Par la suite, nous montrerons que des algorithmes efficaces de reconnaissance d'ACGs affines (*c.a.d.* linéaires avec effacement) ont été mis en valeur au cours de cette thèse. En particulier, nous donnerons une construction de reconnaisseurs dans le langage Datalog, un langage de programmation logique. Un telle construction fait suite à une longue tradition de l'utilisation de ce style de programmation en analyse du langage naturel, comme montré au chapitre suivant.

Chapitre 5

Programmation logique et analyse grammaticale

Nous montrons à présent comment construire des reconnaisseurs pour les grammaires catégorielles abstraites du second-ordre, *i.e.* dont les structures de dérivation sont des arbres. Les reconnaisseurs que nous allons construire font appel à des éléments de la programmation logique, et en particulier au langage Datalog, à la suite des travaux de Kanazawa [Kanazawa, 2007, Kanazawa, 2008]. Ce langage est basé sur des règles d'inférence de la logique du premier ordre, et peut être vu comme une restriction du langage Prolog [Colmerauer and Roussel, 1996, Sterling and Shapiro, 1986]. Comme nous le verrons, ce dernier a déjà fait l'objet d'études dans le cadre de l'analyse de grammaires non-contextuelles de chaînes ; les reconnaisseurs construits par Kanazawa étendent donc des constructions connues pour des formalismes grammaticaux plus faibles. Ces consructions d'outils de reconnaissance grammaticale s'appuient essentiellement sur la correspondance naturelle entre les arbres de dérivation dans une grammaire, et les arbres d'évaluation d'une requête. Par ailleurs, Prolog a été amplement utilisé pour la construction d'analyseurs syntaxiques de langages naturels, via le formalisme des grammaires de clauses définies, par exemple et comme nous l'évoquerons dans ce chapitre. Un autre des avantages offerts par la programmation logique est que la construction d'algorithmes d'analyse connus pour les grammaires non-contextuelles, en particulier les algorithmes CYK et de Earley, correspondent à des paradigmes d'évaluation et de transformation de programmes spécifiques dans Datalog, et nous nous appuierons sur ces paradigmes pour donner des algorithmes de reconnaissance de grammaires catégorielles abstraites du second-ordre.

Au cours de ce chapitre, nous présenterons, dans un premier temps, les algorithmes d'analyse CYK et Earley, pour le cas des grammaires non-contextuelles de chaînes ; ensuite, nous présenterons le langage Datalog de manière succincte, en s'attachant à montrer les similitudes entre des techniques de réécriture de programmes et des techniques de construction d'algorithmes d'analyse grammaticale ; enfin, nous développerons la technique d'analyse pour les ACGs du second-ordre donnée dans [Kanazawa, 2007, Kanazawa, 2008], et s'appuyant sur les paradigmes précédemment introduits.

5.1 Algorithmes de reconnaissance grammaticale

Les algorithmes de reconnaissance grammaticale peuvent être divisés en deux familles : les *algorithmes descendants* et les *algorithmes ascendants*. Considérons une grammaire non-contextuelle de chaînes $G = (N, \Sigma, P, S)$ et un mot à reconnaître $w = a_1 \dots a_n$, où pour tout $i \in \{1, \dots, n\}$, a_i est un symbole terminal appartenant à l'ensemble Σ. Comme nous avons pu l'évoquer au cours du chapitre précédent, la reconnaissance de w consiste à rechercher l'existence d'un arbre de dérivation de G dont la frontière est w, et la racine est S. Dans le cas, d'un algorithme *descendant*, la recherche d'un tel arbre est réalisée en considérant, en premier lieu, le terminal S et en construisant l'arbre désiré par application des règles de production de la grammaire ; ce procédé est itéré de manière descendante sur les noeuds de l'arbre étiqueté par des non-terminaux. Dans le cas d'un algorithme *ascendant*, la reconnaissance est réalisée en considérant, en premier lieu, la frontière w, puis en prédisant les règles de production de la grammaire permettant de construire ce mot, de manière ascendante jusqu'à obtenir le symbole terminal S ; alors qu'un seul arbre de dérivation est construit en effectuant une reconnaissance descendante (pour obtenir l'ensemble des arbres, la requête doit être relancée, soit sur la requête initiale, soit sur une des itérations), la reconnaissance ascendante permet de construire l'ensemble des arbres de dérivation pour le mot w. Si plusieurs arbres de dérivation existent pour un mot w, nous dirons que la grammaire est ambiguë.

Par ailleurs, les algorithmes de reconnaissance descendante souffrent du fait qu'il faille mettre en place une méthode de "backtracking", c'est-à-dire de retour arrière dans le construction de l'arbre de dérivation. En effet, si la méthode utilise une certaine règle à l'étape k, mais que cette règle s'avère erronée à l'étape $k+l$, il nous faut pouvoir reprendre la recherche à l'étape k. Nous nous intéresserons donc à deux algorithmes de reconnaissance ascendante : les algorithmes CYK (en référence à Cocke, Younger et Kasami) [Kasami, 1965, Younger, 1967, Cocke and Schwartz, 1970], et l'algorithme de Earley [Earley, 1970] qui réalise la reconnaissance ascendante guidée par une prédiction descendante.

5.1.1 L'algorithme de Cocke-Younger-Kasami

Cet algorithme est un algorithme ascendant pur, qui nécessite que la grammaire en entrée soit, au préalable, mise sous forme normale de Chomsky :

Définition 5.1.1.1. *Une grammaire non-contextuelle $G = (N, \Sigma, P, S)$ est sous* forme normale de Chomsky *(abrégé en CNF) si les règles de production de P sont de la forme :*

1. $A \Rightarrow BC$, *où* $A, B, C \in N$, *ou*
2. $A \Rightarrow a$ *où* $A \in N$ *et* $a \in \Sigma$.

Remarque 5.1.1.1. Il est connu que pour toute grammaire non-contextuelle G, il existe une grammaire non-contextuelle G' en CNF telle que $L(G) = L(G')$.

L'algorithme CYK se caractérise par la maintenance d'une table T, contenant des éléments de la forme $[i, A, j]$, où $A \in N$ et $0 \leq i < j \leq n$, n étant la taille du mot $w = a_1 \dots a_n$ à reconnaître (nous parlerons donc d'*algorithme tabulaire*). Ainsi, l'ajout d'un élément $[i, A, j]$ à la table T indique que la sous-chaîne $a_{i+1} \dots a_j$ a pu être dérivée à partir du non-terminal A (*i.e.* $A \Rightarrow^* a_{i+1} \dots a_j$). L'algorithme s'arrête lorsque $[0, S, n]$ appartient à la table T (dans ce cas, $w \in L(G)$), ou lorsque plus aucun élément ne peut être ajouté à la table (dans ce cas, $w \notin L(G)$).

```
1 :   function CYK(G, w)
2 :   T := ∅
3 :   for all j ∈ {1,...,n}
4 :     for all A ⇒ a_j ∈ P
5 :       ajouter([j−1, A, j], T)
6 :     for all i ∈ {j−1,...,0}
7 :       for all k ∈ {i+1,...,j−1}
8 :         for all A ⇒ BC ∈ P
9 :           if [i,B,k], [k,C,j] ∈ T
10 :            ajouter([i,A,j], T)
11 :  return [0,S,n] ∈ T
```

FIG. 5.1 – Algorithme CYK

$$\frac{}{[j-1,A,j]}\ (A \to a_j) \qquad \frac{[i,B,k] \qquad [k,C,j]}{[i,A,j]}\ (A \to BC)$$

FIG. 5.2 – Règles de déduction pour l'algorithme CYK

L'algorithme est donné sous forme de pseudo-code en Figure 5.1. Une version alternative de cette algorithme est donnée en Figure 5.2, sous forme de règles déductives (le système déductif ainsi créé est appelé *schéma d'analyse* ou *parsing schemata*)

D'après cette représentation, nous pouvons déterminer les complexités en temps et en espace de l'algorithme CYK. En effet, la complexité en temps est bornée par la règle correspondant à l'analyse des règles de production de la forme $A \to BC$. Cette règle de déduction fait intervenir trois entrées dans la table T, et une règle de production $A \to BC$; la complexité en temps est donc de l'ordre $O(|P|n^3)$. De plus, le nombre d'entrées de type $[i,A,j]$ est borné par $O(|P|n^2)$, qui est donc la complexité en espace de cet algorithme.

Cependant, l'algorithme CYK n'est pas optimal ; en effet, d'une part, nous avons vu qu'il est nécessaire de transformer la grammaire initiale en une grammaire sous forme CNF ; d'autre part, la reconnaissance se fait sans tenir compte des informations pouvant être données par les arbres de dérivation en cours de construction. Il est en effet possible que la reconnaissance se poursuive sur un non-terminal inapproprié, impliquant ainsi que des informations inutiles à la construction des arbres de dérivation finaux soient rajoutées à la table T.

5.1.2 L'algorithme de Earley

Comme nous venons de le voir, une méthode da reconnaissance ascendante pure n'exploite pas les informations données par les arbres de dérivation en cours de construction ; de manière identique, pour une reconnaissance descendante, nous n'utilisons aucune information donnée par les symboles terminaux composant le mot à reconnaître, et il se peut qu'une branche de recherche s'avère erronée, forçant ainsi le "backtracking". La méthode donnée par l'algorithme de Earley permet de combler à ces deux manques, en utilisant certaines informations exploitées par les reconnaisseurs descendants (informations données par les non-terminaux) et par les reconnaisseurs ascendants (données pas les terminaux). De plus, et contrairement à l'algorithme CYK, l'algorithme de Earley ne nécessite pas de transformation préalable de la grammaire en entrée.

```
1 :   function Earley(G, w)
2 :   T := {[0, S ⇒ •γ, 0] | S ⇒ γ ∈ P};
      A := {[0, S ⇒ •γ, 0] | S ⇒ γ ∈ P};    /* INITIALISER */
3 :   for all j ∈ {0, ..., n}
4 :     for all [i, A ⇒ α • aα', j − 1] ∈ T        /* SCAN a */
5 :       if a = a_j
6 :         ajouter([i, A ⇒ αa • α', j], T)
7 :         ajouter([i, A ⇒ αa • α', j], A)
8 :     while A ≠ ∅
9 :       [k, A ⇒ α • α', j] :=retirer_quelconque(A)
10 :      if α' = Bβ
11 :        for all B ⇒ γ ∈ P                    /* PRÉDIRE B ⇒ γ */
12 :          if [j, B ⇒ •γ, j] ∉ T
13 :            ajouter([j, B ⇒ •γ, j], T)
14 :            ajouter([j, B ⇒ •γ, j], A)
15 :        for all [j, B ⇒ γ•, j] ∈ T           /* COMPLÉTER */
16 :          if [k, A ⇒ αB • β, j] ∉ T
17 :            ajouter([k, A ⇒ αB • β, j], T)
18 :            ajouter([k, A ⇒ αB • β, j], A)
19 :      if α' = ϵ
20 :        for all [i, B ⇒ β • Aγ, k] ∈ T
21 :          if [i, B ⇒ βA • γ, j] ∉ T
22 :            ajouter([i, B ⇒ βA • γ, j], T)
23 :            ajouter([i, B ⇒ βA • γ, j], A)
24 :  return [O, S, n] ∈ T
```

FIG. 5.3 – Algorithme de Earley

Cet algorithme est également de type tabulaire. Les éléments de cette table sont cependant différents et de la forme $[i, A \Rightarrow \alpha \bullet \beta, j]$, où

- α,β appartiennent à $(N \cup \Sigma)^*$
- $A \Rightarrow \alpha\beta$ appartient à P
- le symbole $\bullet$ sépare les éléments déjà reconnus (à gauche de $\bullet$), des éléments à reconnaître (à droite).

Un tel élément est ajouté à la table T, si $S \Rightarrow^* a_1 \ldots a_i A\gamma$ est vérifié pour un certain $\gamma \in (N \cup \Sigma)^*$ et si $\alpha \Rightarrow^* a_{i+1} \ldots a_j$ où $i < j$. Une telle condition peut être vue comme une complexification de la condition d'ajout d'éléments à la table T dans l'algorithme CYK afin d'intégrer une condition de *correction par préfixe* : à toute étape de la reconnaissance, nous vérifions que la sous-chaîne jusqu'à présent reconnue est un préfixe du mot w en entrée.

Afin de mettre en place cet algorithme, nous introduisons une deuxième table qui contient l'ensemble des éléments de T devant encore être reconnus. Nous appelons cette table l'*agenda* et la notons A. Deux entrées identiques ne seront jamais ajoutées dans l'agenda afin d'assurer la terminaison de l'algorithme.

L'algorithme est donné en Figure 5.1.2. Nous pouvons y distinguer plusieurs étapes : lors de l'étape d'initialisation, nous incluons les éléments de la forme $S \Rightarrow \bullet\gamma$ (où $S \Rightarrow \gamma$ est une règle de

production de la grammaire) dans la table et l'agenda ; l'intuition est que nous devons pouvoir utiliser une de ces règles afin de construire l'arbre de dérivation ; contrairement à l'algorithme CYK, l'utilisation de cette règle se fait au début de l'algorithme, en prédisant que cette règle devra être utilisée dans la construction de la dérivation. L'étape de lecture (ou SCAN) permet de vérifier que les arbres en cours de construction sont correctes à chaque étape pour le mot w à reconnaître ; en effet, lors de cette étape, nous nous assurons que la sous-chaîne reconnue jusqu'à présent est un préfixe du mot w. Ainsi, nous remarquerons que les étapes de lecture se font de gauche à droite sur le mot w. L'étape de PRÉDICTION permet de construire de manière descendante la branche courante, amenant à la reconnaissance d'une sous-chaîne. La COMPLÉTION permet de valider cette recherche en clôturant une branche.

On remarquera que cet algorithme se caractérise par le fait qu'il réalise une reconnaissance ascendante d'un mot en exploitant les règles de production de la grammaire de manière descendante, puisque pour chaque nouveau symbole analysé (par une opération SCAN), cette analyse intermédiaire est validée en construisant la branche de l'arbre correspondant. Nous dirons également que c'est un *algorithme ascendant avec prédiction descendante*. Une autre manière de voir cet algorithme est de voir que la reconnaissance se fait par le symbole terminal le plus à gauche. Comme pour l'algorithme CYK, nous donnons une définition alternative de cet algorithme sous forme de schéma d'analyse :

$$\frac{}{[0, S \to \bullet\gamma, 0]}\ \text{(Initialiser)} \qquad \frac{[i, A \to \alpha \bullet a\beta, j]}{[i, A \to \alpha a \bullet \beta, j+1]}\ \text{(Scan)}$$

$$\frac{[i, A \to \alpha \bullet B\beta, j]}{[j, B \to \bullet\gamma, j]}\ \text{(Prédire } B \to \gamma\text{)} \qquad \frac{[i, A \to \alpha \bullet B\beta, k] \qquad [k, B \to \gamma\bullet, j]}{[i, A \to \alpha B \bullet \beta, j]}\ \text{(Compléter)}$$

FIG. 5.4 – Règles de déduction pour l'algorithme de Earley

Du point de vue des performances de cet algorithme, la complexité en temps est bornée par la règle de complétion, qui fait appel à trois entrées dans la table et à deux règles de production. Ainsi, cette complexité est $O|(P|^2 n^3)$. La complexité en espace est à nouveau polynomiale et est donnée par $O(|P|n^2)$.

5.2 Reconnaissance grammaticale et déduction logique

5.2.1 Un langage de programmation logique : Datalog

Datalog [Ceri et al., 1989] est un langage de requêtes sur des bases de données relationnelles et qui est basé sur un paradigme logique. En effet, l'accès aux données se fait par l'intermédiaire de règles, qui correspondent à des formules de la logique du premier ordre, et plus particulièrement à des clauses de Horn. Ainsi, lorsqu'une requête est lancée sur un programme Datalog, de nouveaux faits sont construits à partir des faits initialement donnés et de ceux construits précédemment au cours de l'exécution de la requête. Pour une requête donnée, la programme renvoie donc une valeur de vérité (au cas où la requête ne contient pas de variables) correspondant à la validité de cette requête dans le programme (le fait exprimé dans la requête peut-il être dans ce programme ?), ou un ensemble d'assignations de valeurs aux variables de la requête, spécifiant ainsi les valeurs pour lesquelles la requête est valide.

Syntaxe et évaluation

Définition 5.2.1.1. *Une* règle *r d'un programme Datalog est donnée sous forme de clauses de Horn*

$$L_0 \text{ :- } L_1, \ldots, L_n$$

où chaque littéral L_i *est de la forme* $P(t_1, \ldots, t_m)$ *tel que*

- P *est un* prédicat *et*
- *pour tout* $j \in \{1, \ldots, m\}$, t_j *est un* terme *(i.e. une* constante *ou une* variable*).*

L_0 *est la* tête *de la règle ;* $L_1, \ldots, L_n$ *son* corps*, éventuellement vide. Au cas où le corps de la règle est vide, nous dirons que r est un* fait*.*

Un littéral, une règle ou un fait ne contenant pas de variables sera dit *pur*.

Notation 5.2.1. Nous noterons par $\mathcal{P}, \mathcal{P}_1, \mathcal{P}_2, \ldots$ les programmes Datalog, par des lettres latines majuscules les prédicats, par des lettres latines minuscules les constantes et par des lettres latines minuscules grasses les variables d'un tel programme.

Remarque 5.2.1.1. L'ensemble des prédicats d'un programme Datalog définit un alphabet rangé ; ainsi, à l'image des non-terminaux d'une grammaire non-contextuelle multiple, à chaque prédicat d'un programme est associé une arité fixe (étant donné un littéral $P(t_1, \ldots, t_n)$ dans un programme $\mathcal{P}$, le prédicat P peut donc être vu comme une relation n-aire).

On remarquera que les règles d'un programme Datalog se différencient dans leur syntaxe de celles d'un programme Prolog par le fait que :

- les termes d'un prédicat sont, soit des constantes, soit des variables, mais ne peuvent pas être des termes fonctionnels.
- le négation n'est pas utilisée (néanmoins, des extensions de Datalog incluant la négation ont été étudiées).

Définition 5.2.1.2. *Étant donné un programme Datalog* $\mathcal{P}$*, nous dirons que celui-ci vérifie la* condition de sûreté *si pour toute règle r de* $\mathcal{P}$*, toute variable apparaissant dans la tête de r apparaît dans le corps de celle-ci.*

Nous dirons alors que le programme $\mathcal{P}$ *est sûr.*

Cette condition donne lieu à une propriété importante : l'ensemble des faits pouvant être dérivés dans un programme sûr est fini. En effet, grâce à cette condition, toute variable apparaissant en tête d'une règle r a au moins une occurrence dans le corps de r. Cette condition empêche donc les variables de tête de règle de prendre une valeur quelconque lors de l'évaluation d'une requête.

Un programme Datalog $\mathcal{P}$ est usuellement divisé en deux parties :

1. *la base de donnée extensionnelle* (notée $\texttt{ext}(\mathcal{P})$) qui contient des faits purs. Ceux-ci peuvent être stockés dans une base de données relationnelle.
2. *la base de données intentionnelle* (notée $\texttt{int}(\mathcal{P})$) qui est faite de règles (la plupart du temps sans faits). Dans le cas d'un programme sûr, il s'agit donc de règles dont les corps ne sont pas vides.

Enfin, nous donnons la syntaxe suivante pour une requête sur un programme Datalog $\mathcal{P}$:

$$?\text{:- } P(t_1, \ldots, t_n)$$

où P est un prédicat n-aire du programme $\mathcal{P}$ et $t_1, \ldots, t_n$ sont des termes.

Exemple 5.2.1.1. Nous nous proposons de construire une base de données permettant de savoir si deux personnes sont de la même génération ; nous définissons tout d'abord la base de de données extensionnelle permettant de déclarer des personnes et leurs liens de parenté.

$$\begin{array}{ll} PERS(marcel). & PARENT(marcel, baptiste). \\ PERS(nicole). & PARENT(nicole, clotilde). \\ PERS(marc). & PARENT(marc, marcel). \\ PERS(baptiste). & PARENT(marc, nicole). \\ PERS(clotilde). & \end{array}$$

Puis, nous donnons les règles de la base de données intentionnelle qui permettent d'obtenir les liens familiaux entre les individus :

$$\begin{array}{ll} r_1 : & GEN(\mathbf{x}, \mathbf{x}) \text{ :- } PERS(\mathbf{x}). \\ r_2 : & GEN(\mathbf{x}, \mathbf{y}) \text{ :- } PARENT(\mathbf{x}, \mathbf{x_1}), GEN(\mathbf{x_1}, \mathbf{y_1}), PARENT(\mathbf{y}, \mathbf{y_1}). \end{array}$$

Nous nous intéressons, à présent, à de méthodes d'évaluation de requêtes. Dans la première, dite *descendante* (ou *top-down*), la recherche de solutions dans la base de données est réalisée en considérant le prédicat P de la requête, puis en essayant d'appliquer chaque règle de $\texttt{ext}(\mathcal{P}) \cup \texttt{int}(\mathcal{P})$ dont le prédicat de tête est P, et où les variables ont été éventuellement instanciées par des constantes de la requête si celles-ci sont sur la même position en argument du prédicat P ; la méthode est ensuite itérée sur chaque littéral du corps de la règle sélectionnée. À l'image des programmes Prolog pour lesquels cette méthode est utilisée, il nous faut mettre en place une technique permettant de remonter dans l'historique de recherche lorsqu'une branche en cours d'exploration échoue à donner une réponse à la requête (de telles méthodes portent le nom de *backward chaining*). Selon les méthodes, il est possible qu'il faille également éviter des cycles dans l'arbre de recherche (c'est le cas pour les méthodes en profondeur d'abord).

Remarque 5.2.1.2. On notera que, dans Prolog, l'évaluation est réalisée de manière descendante, avec une implémentation en profondeur d'abord avec backward chaining. Ainsi, la terminaison d'une évaluation dépend de l'ordre des règles et de l'ordre des littéraux dans celles-ci. En effet, prenons le programme suivant, équivalent au programme précédent écrit pour Datalog :

$$\begin{array}{ll} r'_1 : & GEN(\mathbf{x}, \mathbf{y}) \text{ :- } GEN(\mathbf{x_1}, \mathbf{y_1}), PARENT(\mathbf{x}, \mathbf{x_1}), PARENT(\mathbf{y}, \mathbf{y_1}). \\ r'_2 : & GEN(\mathbf{x}, \mathbf{x}) \text{ :- } PERS(\mathbf{x}). \end{array}$$

Alors, une évaluation descendante et en profondeur d'abord de la requête $?\text{ :- } GEN(\mathbf{x}, \mathbf{y})$ ne s'arrêterait pas, puisque elle bouclerait sur $GEN(\mathbf{x_1}, \mathbf{y_1})$.

La deuxième famille de méthodes d'évaluation, est dite *ascendante* (ou *bottom-up*). Elle est utilisé dans Datalog et est celle que nous considérons par la suite. Ces méthodes consistent à prendre les faits de $\texttt{ext}(\mathcal{P})$ et à dériver de nouveaux faits en utilisant les règles de $\texttt{int}(\mathcal{P})$; la méthode est itérée sur le nouvel ensemble de faits ainsi obtenu jusqu'à ce que cet ensemble se stabilise (la méthode consiste donc en la recherche d'un point fixe sur l'ensemble des faits). Une fois l'ensemble final obtenu, nous sélectionnons les faits répondant à la requête. Parmi ces méthodes, la version dite "naïve" souffre de deux inconvénients majeurs :

1. il est possible qu'un résultat intermédiaire soit calculé plusieurs fois, alors que les faits obtenus ne modifient plus l'ensemble des faits en cours de construction.
2. la méthode produit l'ensemble des faits pouvant être dérivés dans la base de données ; ceux qui ne sont pas nécessaires ne sont éliminés qu'à la fin de la méthode.

Par la suite, nous verrons que la méthode dite *semi-naïve* permet de résoudre les problèmes exposés en 1. ; des méthodes de transformation de la base de données extensionnelle telles que la *réécriture par ensembles magiques* permettent de résoudre les problèmes décrits en 2.

L'évaluation semi-naïve

Nous avons décrit une version naïve de l'évaluation ascendante. Nous donnons une optimisation de cette méthode à travers l'évaluation dite *semi-naïve* ou de *point-fixe différentiel*. Cette méthode est simple dans sa formulation : plutôt que de calculer un nouvel ensemble de faits à partir de l'ensemble total de faits à l'itération précédente, nous calculons simplement les faits à rajouter. L'idée générale de cette méthode est que, si à l'étape k de calcul de l'ensemble des faits, un nouveau fait a été rajouté, il ne peut provenir que de l'utilisation d'une règle dont un littéral du corps a été instancié par un fait déduit à l'étape $k-1$.

Afin de réaliser cette procédure, l'ensemble des faits est divisé en deux : l'ensemble F des faits présents et l'ensemble différentiel de faits ΔF. Ce dernier contient l'ensemble des faits déduits lors de la dernière étape de calcul. Nous notons par $\mathcal{F}^k$ et $\Delta^k\mathcal{F}$ les ensembles $\mathcal{F}$ et $\Delta\mathcal{F}$ à l'étape de calcul k. Remarquons que $\Delta^k\mathcal{F} = \mathcal{F}^k - \mathcal{F}^{k-1}$.

Un fait $P(a_1,\ldots,a_n)$ appartient à $\Delta^{k+1}\mathcal{F}$ ssi il existe une règle

$$P(\mathbf{x}_1,\ldots,\mathbf{x}_n) \text{ :- } P_1(\mathbf{x}_{11},\ldots,\mathbf{x}_{1n_1}),\ldots,P_m(\mathbf{x}_{m1},\ldots,\mathbf{x}_{mn_m})$$

telle que :

1. $P(a_1,\ldots,a_n) \text{ :- } P_1(a_{11},\ldots,a_{1n_1}),\ldots,P_m(a_{m1},\ldots,a_{mn_m})$ est dérivable
2. pour tout $i \in \{1,\ldots,m\}$, $P_i(a_{i1},\ldots,a_{in_i})$ appartient à $\mathcal{F}^k$.
3. il existe $i \in \{1,\ldots,m\}$ tel que $P_i(a_{i1},\ldots,a_{in_i})$ appartient à $\Delta^k\mathcal{F}$.

La méthode se termine lorsqu'un point fixe est atteint.

Remarque 5.2.1.3. On remarquera une certaine analogie entre les deux ensembles maintenus par cette méthode et les deux tables de l'algorithme de Earley décrit précédemment. Ainsi, la table T contient l'ensemble des éléments dérivées, de même que l'ensemble $\mathcal{F}$ contient l'ensemble des faits dérivés ; l'agenda A contient l'ensemble des éléments dérivés lors de la dernière étape de calcul, ces éléments étant les seuls à être utilisés pour dériver de nouveau éléments ; il en est de même pour $\Delta\mathcal{F}$.

Réécriture par ensembles magiques

Nous donnons à présent une méthode de transformation de programmes Datalog permettant de pallier au fait que les méthodes ascendantes ne rejettent les faits inutiles qu'à la fin de l'évaluation. Grâce à cette méthode, les faits créés sont les mêmes que ceux qui seraient créés par une évaluation descendante. Le programme obtenu par cette méthode, est plus grand que le programme initial, incluant de nouveaux prédicats dans la base de données intentionnelle ; ces prédicats supplémentaires servent en fait de contraintes, forçant les variables à vérifier certaines conditions supplémentaires. Plusieurs méthodes et extensions de la méthode originale ont été décrites, et nous nous intéressons ici uniquement à la méthode de *réécriture par ensembles magiques supplémentaires*, exposée dans [Beeri and Ramakrishnan, 1991]. Cette méthode permet d'inclure un moyen de réaliser de la prédiction descendante pour une telle méthode d'évaluation. Nous pouvons remarquer l'analogie entre une évaluation ascendante d'un tel programme et l'algorithme d'Earley pour l'analyse grammaticale de grammaires non-contextuelles de chaînes. Nous verrons plus loin que cette méthode de

réécriture permet effectivement de construire de tels algorithmes, non seulement pour les grammaires non-contextuelles de chaînes, mais également pour les grammaires catégorielles abstraites du second-ordre.

Définition 5.2.1.3. *Étant donnés un programme Datalog $\mathcal{P}$ et une requête ?:-$P(t_1,\dots,t_n)$ sur ce programme, pour toute règle $P(t_{01},\dots,t_{0n})$:-$L_1,\dots,L_m$ de $\mathcal{P}$ nous dirons que :*

- *pour $i \in \{1,\dots,n\}$, si t_{0i} est une occurrence de variable, elle sera dite liée (resp. libre) si t_i est une constante (resp. une variable).*
- *pour tout $j \in \{1,\dots,m\}$, posons $L_j = P_j(t_{j1},\dots,t_{jn_j})$; étant donné $l \in \{1,\dots,n_j\}$, si t_{jl} est une occurrence de variable, elle sera dite liée si elle apparaît dans un littéral L_i tel que $i < j$ ou dans les variables de $\{t_{01},\dots,t_{0n}\}$; sinon, elle sera dite libre.*
- *la méthode est ensuite itérée sur les requêtes ?:-L'_j formées en remplaçant les variables libres de L_j par des constantes.*

On peut donc facilement voir que déterminer si une variable est libre ou liée revient à déterminer les occurrences de variables non instanciées pour une évaluation descendante, en profondeur d'abord, pour une requête donnée et un programme donné.

La méthode de réécriture de programmes Datalog par ensemble magiques s'appuie sur cette notion d'occurrences libres et liées de variables, afin de tirer profit des avantages d'une évaluation descendante. Nous illustrons et commentons cette méthode grâce au programme fait des règles suivantes :

$$\begin{aligned} r_1 :&\quad ANCETRE(\mathbf{x},\mathbf{y}) \text{:-} PARENT(\mathbf{x},\mathbf{y}).\\ r_2 :&\quad ANCETRE(\mathbf{x},\mathbf{y}) \text{:-} ANCETRE(\mathbf{x},\mathbf{z}), ANCETRE(\mathbf{z},\mathbf{y}).\end{aligned}$$

Nous ajoutons les faits $PARENT(a,c)$ et $PARENT(c,b)$, et la requête ?:-$ANCETRE(a,b)$ sur ce programme.

La réécriture d'un programme se divise en plusieurs étapes :

- Étape 1 : Décorer le programme par des labels b et f sur les prédicats, indiquant respectivement le statut lié ou libre des occurrences de variables en argument. Ainsi, à la requête initiale $A(t_1,\dots,t_n)$, nous associons la décoration $d : \{1,\dots,n\} \mapsto \{b,f\}$, telle que $d(i) = b$ ssi t_i est une constante et $d(i) = f$ ssi t_i est une variable. Étant donnés A et d, pour chaque règle

$$A(s_1,\dots,s_n) \text{:-} A_1(s_{1,1},\dots,s_{1,n_1}),\dots,A_r(s_{1,1},\dots,s_{1,n_r})$$

  nous créons une règle

$$A^d(s_1,\dots,s_n) \text{:-} A_1^{d_1}(s_{1,1},\dots,s_{1,n_1}),\dots,A_r^{d_r}(s_{r,1},\dots,s_{r,n_r})$$

  où, pour tout $i \in \{1,\dots,r\}$, $d_i(j) = b$ ssi $s_{i,j} = \begin{cases} t_k & k \in \{1,\dots,n\} \text{ et } d(k) = b, \text{ ou bien} \\ s_{l,k} & l \in \{1,\dots,i-1\} \end{cases}$

  La méthode est itérée pour les non-terminaux A_i et les requêtes d_i, jusqu'à aboutir à un point fixe.

  Exemple 5.2.1.2. Pour la requête ?:-$ANCETRE(a,b)$, nous obtenons le programme décoré suivant :

$$\begin{aligned} r_1 :&\quad ANCETRE^{bb}(\mathbf{x},\mathbf{y}) \text{:-} PARENT^{bb}(\mathbf{x},\mathbf{y}).\\ r_2 :&\quad ANCETRE^{bb}(\mathbf{x},\mathbf{y}) \text{:-} ANCETRE^{bf}(\mathbf{x},\mathbf{z}), ANCETRE^{bb}(\mathbf{z},\mathbf{y}).\\ r'_1 :&\quad ANCETRE^{bf}(\mathbf{x},\mathbf{y}) \text{:-} PARENT^{bf}(\mathbf{x},\mathbf{y}).\\ r'_2 :&\quad ANCETRE^{bf}(\mathbf{x},\mathbf{y}) \text{:-} ANCETRE^{bf}(\mathbf{x},\mathbf{z}), ANCETRE^{bf}(\mathbf{z},\mathbf{y}).\end{aligned}$$

Remarque 5.2.1.4. Étant donné que la règle récursive r_2 introduit une deuxième décoration pour le prédicat $ANCETRE$, il nous faut ajouter de nouvelles règles au programme [1].

- Étape 2 : À partir du nouveau programme ainsi obtenu, nous transformons chaque règle p, dont la forme générale est $A^d(t_1,\ldots,t_n)$:-$A_1^{d_1}(t_{1,1},\ldots,t_{1,n_1}),\ldots,A_r^{d_r}(t_{r,1},\ldots,t_{r,n_r})$, de la manière suivante :

  Introduction de prédicats magiques et supplémentaires la règle p est binarisée grâce à r nouvelles règles comme suit :

  $$A^d(t_1,\ldots,t_n)\text{:-}\,m\_A\_d(s_1,\ldots,s_l),A_r^{d_r}(t_{r,1},\ldots,t_{r,n_r})$$

  où $\{s_1,\ldots,s_l\}=\{t\in\{t_1,\ldots,t_n\}\mid d(t)=b\}$.
  De plus, si $r>1$, alors, pour tout $k\in\{1,\ldots,r-1\}$, nous ajoutons la règle :

  $$sup_{p,k}(u_1,\ldots,u_m)\text{:-}\,sup_{p,k+1}(s_1,\ldots,s_l),A_k(t_{k,1},\ldots,t_{k,n_k})$$

  de sorte que $sup_{p,r}=m\_A\_d$ et $\{u_1,\ldots,u_m\}=\{s_1,\ldots,s_l\}\cup\{t_{k,1},\ldots,t_{k,n_k}\}$
  Exemple 5.2.1.3. La règle r_2 du programme ci-dessus donne donc lieu à deux nouvelles règles :

  $$ANCETRE\_bb(\mathbf{x},\mathbf{y})\text{:-}\,m\_ANCETRE\_bb(\mathbf{x},\mathbf{y}),PARENT\_bb(\mathbf{x},\mathbf{y}).$$
  $$sup_{2,1}(\mathbf{x},\mathbf{y},\mathbf{z})\text{:-}\,m\_ANCETRE\_bb(\mathbf{x},\mathbf{y}),ANCETRE\_bf(\mathbf{x},\mathbf{z}).$$

  Déclenchement des prédicats magiques Étant donné un prédicat intentionnel (*i.e.* apparaissant en partie gauche d'une règle intentionnelle) B^e pour lequel un prédicat magique m_B_e a été introduit au cours de l'étape précédente, et la règle p précédemment donnée, alors, pour tout $1\leq k\leq r$, tel que $A_k=B$, nous introduisons la règle :

  $$m\_B\_e(s_1,\ldots,s_l)\text{:-}\,sup_{k,p}(u_1,\ldots,u_m)$$

  telle que $sup_{1,p}=m\_A\_d$ et $\{s_1,\ldots,s_l\}=\{t\in\{t_{k,1},\ldots,t_{k,r_k}\}\mid e(t)=b\}$
  Enfin, pour la requête ?:-$A(t_1,\ldots,t_n)$ initialement donnée, nous intialisons le prédicat magique associée :

  $$m\_A\_d(t_{i_1},\ldots,t_{i_k})$$

  où $d(i)=b$ ssi $i\in\{i_1,\ldots,i_k\}$.
  Exemple 5.2.1.4. Sur notre exemple, la règle r_2 donne lieu à la règle :

  $$m\_ANCETRE\_bb(\mathbf{z},\mathbf{y})\text{:-}\,sup_{2,1}(\mathbf{x},\mathbf{y},\mathbf{z}).$$

  De plus, pour la requête ?:-$ANCETRE(a,b)$, il nous faut initialiser le prédicat magique $m\_ANCETRE\_bb$ de la façon suivante :

  $$m\_ANCETRE\_bb(a,b).$$

Sur notre exemple, nous obtenons donc le programme Datalog en Figure 5.5. Les idées principales de cette méthode de réécriture sont donc :

[1] Il est cependant possible d'ordonner les décorations afin de ne garder que les plus faibles, grâce à différentes heuristiques, décrites dans [Ullman, 1988b]

$$ANCETRE\_bb(\mathbf{x},\mathbf{y}) :\text{-} m\_ANCETRE\_bb(\mathbf{x},\mathbf{y}), PARENT\_bb(\mathbf{x},\mathbf{y}).$$

$$ANCETRE\_bb(\mathbf{x},\mathbf{y}) :\text{-} sup_{2,1}(\mathbf{x},\mathbf{y},\mathbf{z}), ANCETRE\_bb(\mathbf{z},\mathbf{y}).$$
$$sup_{2,1}(\mathbf{x},\mathbf{y},\mathbf{z}) :\text{-} m\_ANCETRE\_bb(\mathbf{x},\mathbf{y}), ANCETRE\_bf(\mathbf{x},\mathbf{z}).$$
$$m\_ANCETRE\_bb(\mathbf{z},\mathbf{y}) :\text{-} sup_{2,1}(\mathbf{x},\mathbf{y},\mathbf{z}).$$
$$m\_ANCETRE\_bf(\mathbf{x}) :\text{-} m\_ANCETRE\_bb(\mathbf{x},\mathbf{z}).$$

$$ANCETRE\_bf(\mathbf{x},\mathbf{y}) :\text{-} m\_ANCETRE\_bf(\mathbf{x}), PARENT\_bf(\mathbf{x},\mathbf{y}).$$

$$ANCETRE\_bf(\mathbf{x},\mathbf{y}) :\text{-} sup_{4,1}(\mathbf{x},\mathbf{z}), ANCETRE\_bf(\mathbf{z},\mathbf{y}).$$
$$sup_{4,1}(\mathbf{x},\mathbf{z}) :\text{-} m\_ANCETRE\_bf(\mathbf{x}), ANCETRE\_bf(\mathbf{x},\mathbf{z}).$$
$$m\_ANCETRE\_bf(\mathbf{z}) :\text{-} sup_{4,1}(\mathbf{x},\mathbf{z}).$$

FIG. 5.5 – Programme Datalog réécrit selon la méthode des ensembles magiques supplémentaires

ETAPE	FAIT AJOUTÉ
1	$m\_ANCETRE\_bb(a,b)$
2	$m\_ANCETRE\_bf(a)$
3	$ANCETRE\_bf(a,c)$
4	$sup_{2,1}(a,b,c)$ $sup_{4,1}(a,c)$
5	$m\_ANCETRE\_bb(c,b)$ $m\_ANCETRE\_bf(c)$
6	$ANCETRE\_bb(c,b)$ $m\_ANCETRE\_bf(c)$ **ANCETRE_bb(a, c)** $ANCETRE\_bf(c,b)$

FIG. 5.6 – Exemple d'évaluation semi-naïve

1. d'introduire des prédicats *magiques* afin de simuler une évaluation descendante de la requête.
2. d'introduire des prédicats *supplémentaires* afin d'obtenir une binarisation de chaque règle du programme initial.

On remarquera que les règles du programme initial ont été modifiées afin, d'une part, d'utiliser les règles et prédicats nouvellement rajoutées, et d'autre part, afin d'utiliser les prédicats magiques pour que l'appel à une règle force la prédiction descendante des variables liées.

L'évaluation de le requête $?\text{-}ANCETRE(a,b)$ se fait comme indiquée en Figure 5.6, et le fait $ANCETRE\_bb(a,c)$ est donc vérifié à l'étape 6 de l'analyse. On remarquera que le calcul successif des faits s'apparente à une évaluation descendante du programme décoré, suite à l'étape 1.

Nous finissons cette section par une remarque importante concernant Datalog :

Propriété 5.2.1.1. *Étant donné un programme Datalog $\mathcal{P}$ et $\texttt{ext}(\mathcal{P})$ la base extensionnelle de ce programme. Une requête sur $\mathcal{P}$ s'exécute en temps $O(|\texttt{ext}(\mathcal{P})|^n)$, pour $n \in \mathbb{N}$.*

Cette propriété est particulièrement intéressante pour démontrer l'efficacité des algorithmes que nous construirons.

5.2.2 L'exemple des grammaires de clauses définies

Les méthodes d'évaluation en programmation logique et les méthodes de reconnaissance de grammaires formelles présentent, comme nous venons de le voir, certaines propriétés similaires. Tout d'abord, le problème de reconnaissance d'un mot dans une grammaire revient à découvrir un arbre de dérivation pour ce mot ; de même, l'évaluation d'une requête sur un programme logique revient à construire un arbre d'évaluation de cette requête. De plus, nous avons pu voir que les méthodes d'évaluation en programmation logique peuvent être divisées en deux familles : les évaluations ascendantes, et les évaluations descendantes. De même, les algorithmes de reconnaissance de grammaires non-contextuelles sont répartis entre les algorithmes ascendants et descendants. L'analogie entre ces deux thématiques est en fait essentielle puisque, à sa création, Prolog avait pour objectif l'analyse du langage naturel [Colmerauer, 1978], les méthodes initiales ayant ensuite été étendues à des formalismes plus complexes.

Les grammaires de clauses définies (DCG pour *definite clause grammars*) sont des grammaires écrites sous forme de programmes logiques, et plus particulièrement pour Prolog. Ainsi, les DCGs reposent sur la définition de clauses de Horn que nous avons évoquées au cours de la section précédente. Remarquons cependant que les clauses ici définies étant des clauses de Prolog et non plus de Datalog, pour une clause $L_0 :\!\text{-}\, L_1, \ldots, L_n$, les littéraux contiennent des termes qui sont soit des constantes, soit des variables, soit des *termes fonctionnels* (*c.a.d.* de la forme $f(t_1, \ldots, t_m)$, où $t_1, \ldots, t_m$ sont eux-mêmes des termes, et f est une fonction m-aire). Un fait de Prolog est donc une clause n'ayant pas de corps $P(t_1, \ldots, t_n)$, telle que $t_1, \ldots, t_n$ ne contient pas de variables.

Définition 5.2.2.1. *Une grammaire de clauses définies $G = (N, P, S)$ est donnée par :*

- *un ensemble de prédicats $N = \bigcup_{n\in\mathbb{N}} N^n$ formant un alphabet gradué ; pour un prédicat $R \in N^n$, n est appelé l'arité de R,*
- *un ensemble de clauses (ou règle Prolog) P, les prédicats des littéraux de ces clauses appartenant à N,*
- *$S \in N$ le prédicat initial.*

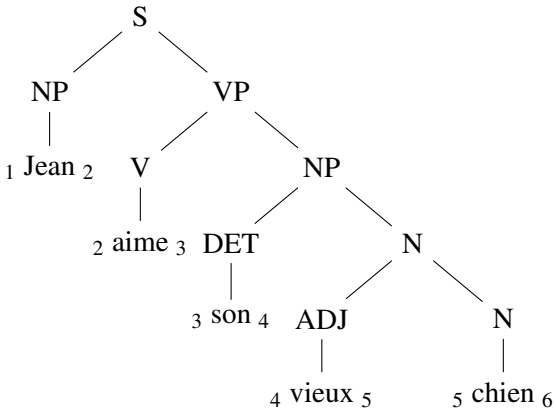
Nous dirons que le terme $t_1 \ldots t_n$ appartient au langage $L(G)$ de G si il existe une réponse positive à la requête $S(t_1, \ldots, t_n)$ sur le programme logique exprimé par G, augmenté d'un ensemble de faits F. Le langage $L(G)$ reconnu par une DCG G est formé de l'ensemble des termes ainsi reconnus.

Un des avantages premiers de ces grammaires et qu'elles permettent d'encoder facilement les grammaires non-contextuelles de chaînes. En effet, une règle de production non-contextuelle $P \Rightarrow \alpha_1\alpha_2 \ldots \alpha_n$ peut être transformée en une clause $P(i_0, i_n) :\!\text{-}\, \alpha_1(i_0, i_1), \alpha_2(i_1, i_2), \ldots, \alpha_n(i_{n_1}, i_n)$, où $i_0, \ldots, i_n$ sont des entiers tels que $i_j < i_{j+1}$ pour tout $j \in \{0, \ldots, n-1\}$. Ces indices permettent d'indiquer l'ordre de concaténation des terminaux et non-terminaux dans la règle de production associée.

Exemple 5.2.2.1. Considérons la grammaire $G = (\Gamma, \Sigma, P, S)$ où $\Gamma = \{S, NP, VP, V, DET, ADJ, N\}$, $\Sigma = \{Jean, aime, son, vieux, chien\}$, et l'ensemble P des règles suivantes :

$$
\begin{array}{lll}
S \Rightarrow NP\ VP & VP \Rightarrow V\ NP & ADJ \Rightarrow vieux \\
NP \Rightarrow DET\ N & N \Rightarrow ADJ\ N & \\
NP \Rightarrow Jean & V \Rightarrow aime & \\
N \Rightarrow chien & DET \Rightarrow son &
\end{array}
$$

Pour $w = Jean\ aime\ son\ vieux\ chien$, nous obtenons l'arbre de dérivation suivant que nous décorons par des indices spécifiant l'ordre des unités lexicales dans w.



Remarque 5.2.2.1. Dans les grammaires de clauses définies, les règles ne codent pas directement les positions des syntagmes dans une phrase par des indices, mais, de manière équivalente, ces syntagmes sont codés comme des listes ordonnées de mots. L'ordre de ces listes est imposé par l'ordre des mots, et est donc répercuté dans les règles grammaticales. La grammaire non-contextuelle précédente peut alors être encodée sous la DCG suivante [2] :

$S(\mathbf{L_1},\mathbf{L_3})$:- $NP(\mathbf{L_1},\mathbf{L_2}), VP(\mathbf{L_2},\mathbf{L_3})$
$NP(\mathbf{L_1},\mathbf{L_3})$:- $DET(\mathbf{L_1},\mathbf{L_2})\ N(\mathbf{L_2},\mathbf{L_3})$
$NP([Jean|\mathbf{L}],\mathbf{L})$
$N([chien|\mathbf{L}],\mathbf{L})$
$VP(\mathbf{L_1},\mathbf{L_3})$:- $V(\mathbf{L_1},\mathbf{L_2})\ NP(\mathbf{L_2},\mathbf{L_3})$
$N(\mathbf{L_1},\mathbf{L_3})$:- $ADJ(\mathbf{L_1},\mathbf{L_2})\ N(\mathbf{L_2},\mathbf{L_3})$
$V([aime|\mathbf{L}],\mathbf{L})$
$DET([son|\mathbf{L}],\mathbf{L})$
$ADJ([vieux|\mathbf{L}],\mathbf{L})$.

Afin d'obtenir la dérivation de la phrase *Jean aime son vieux chien*, nous lançons alors la requête

$$S([Jean, aime, son, vieux, chien], [])$$

Le programme cherche alors à reconstruire la liste $[Jean, aime, son, vieux, chien]$ à partir des règles grammaticales spécifiées.

À travers cet exemple, il apparaît donc clairement qu'il existe alors une correspondance entre les arbres de dérivation d'une grammaire non-contextuelle ainsi codée, et les arbres d'évaluation du programme logique correspondant.

Rajoutons qu'il est possible de s'appuyer sur certains outils de la programmation logique afin de traiter d'autres phénomènes linguistiques tels que les accords syntaxiques, par le biais de l'unification [Pereira and Shieber, 1987]. Ainsi, nous donnons en guise d'exemple, la DCG suivante permettant d'illustrer la gestion des accords syntaxiques :

c_1 : $S(\mathbf{L_1},\mathbf{L_4})$:- $NP(\mathbf{Nb_1},\mathbf{Pers_1},\mathbf{L_1},\mathbf{L_2}), V(\mathbf{Nb_1},\mathbf{Pers_1},\mathbf{L_2},\mathbf{L_3}), NP(\mathbf{Nb_2},\mathbf{Pers_2},\mathbf{L_3},\mathbf{L_4})$.
c_2 : $NP(sing, 3, [Jean|\mathbf{L}],\mathbf{L})$.
c_3 : $NP(sing, 3, [Marie|\mathbf{L}],\mathbf{L})$.
c_4 : $V(sing, 1, [aime|\mathbf{L}],\mathbf{L})$.
c_5 : $V(sing, 2, [aime|\mathbf{L}],\mathbf{L})$.
c_6 : $V(sing, 3, [aime|\mathbf{L}],\mathbf{L})$.
c_7 : $V(pl, 1, [aimons|\mathbf{L}],\mathbf{L})$.
c_8 : $V(pl, 2, [aimez|\mathbf{L}],\mathbf{L})$.
c_9 : $V(pl, 3, [aiment|\mathbf{L}],\mathbf{L})$.

[2] Ce programme fait intervenir les opérations de manipulation de listes ([|]) propres à Prolog et que nous ne détaillerons pas. Notons simplement que $[a|L]$ construit une nouvelle liste à partir d'une liste L, en rajoutant l'élément a en tête de liste

Grâce à l'instanciation des variables $\mathbf{Nb_1}$ et $\mathbf{Pers_1}$ sur la clause c_1, cette grammaire permet de dériver correctement la phrase *Jean aime Marie*, à partir de la requête $S([Jean, aime, Marie], [])$. L'approche de la programmation logique permet donc de voir la reconnaissance grammaticale comme un problème de déduction logique, ce que l'on connaît sous le slogan *parsing as deduction*.

Par ailleurs, nous avons pu voir que nous pouvons séparer les algorithmes d'analyse d'un côté, et les algorithmes d'évaluation en programmation logique de l'autre, en deux familles semblables : les algorithmes ascendants et les algorithmes descendants. De fait, il apparaît naturellement qu'un algorithme d'évaluation ascendante simple sur une grammaire non-contextuelle encodée dans Datalog correspond à l'algorithme CYK. De même, il est possible de simuler l'algorithme de Earley grâce à certaines méthodes [Pereira and Warren, 1983] ; la méthode la plus naturelle d'un point de vue de la programmation logique étant d'exploiter la méthode de réécriture par ensembles magiques que nous avons détaillée précédemment.

5.3 Reconnaissance grammaticale et vérification de typage

La programmation logique exploitant la structure arborescente des dérivations dans les grammaires non-contextuelle de chaînes, il semble naturel de chercher à étendre les méthodes d'analyse de ces grammaires à des grammaires non-contextuelles plus complexes, et en particulier, aux ACGs du second-ordre. Nous montrons que la construction de reconnaisseurs en programmation logique de telles grammaires est possible, en réduisant le problème de la reconnaissance grammaticale non pas à un problème de déduction logique, mais plus précisément à un problème de vérification de typage.

5.3.1 Identifier des λ-termes par des typages

Le problème de reconnaissance pour une grammaire catégorielle abstraite peut être formulé de la manière suivante :

Soient un λ-terme M et une ACG du second-ordre $G = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$.
Existe-t-il $N \in \mathcal{A}(G)$ tel que $\mathcal{L}(N) \to_\beta^* M$?

De manière intuitive, résoudre ce problème revient à résoudre un problème syntaxique sur les λ-termes, où l'on souhaite trouver des β-expansions nous permettant de déterminer le terme N. Un tel problème peut être reformulé comme un problème connu sous le nom de problème de *filtrage dans le λ-calcul simplement typé*. Néanmoins, [de Groote, 2000, Salvati and de Groote, 2003] montrèrent que, dans les cas les plus simples, de tels problèmes sont NP-complets. Les problèmes de filtrage ne donnent donc pas lieu à des algorithmes efficaces pour résoudre le problème de la reconnaissance dans les ACGs du second ordre.

Une solution alternative fut proposée dans [Salvati, 2005], et peut être reformulée de la façon suivante :

Soient un λ-terme M et une ACG du second-ordre $G = (\Sigma_1, \Sigma_2, \mathcal{L}, S)$.
Existe-t-il un système de typage $\mathcal{S}$, un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash_{\mathcal{S}} \alpha$ de M et un terme $N \in \mathcal{A}(G)$ tels que si $\langle \mathbf{\Delta}; \Gamma \rangle \vdash_{\mathcal{S}} \mathcal{L}(N) : \alpha$ alors $\mathcal{L}(N) \to_\beta^* M$?

Dans cette reformulation, on cherche donc à trouver un système de typage nous assurant l'existence d'une stratégie de β-expansion entre M et $\mathcal{L}(N)$. Ce système de typage peut en fait être vu comme un modèle du λ-calcul simplement typé.

Grâce aux propriétés de typage fortes que l'on souhaite associer au système de typage $\mathcal{S}$, nous souhaitons nous épargner la recherche d'une stratégie de β-expansion, afin de pouvoir montrer que le problème de la reconnaissance se résout en temps polynomial dans le cas de certaines ACGs du second-ordre.

Dans ce qui suit, nous introduisons de manière informelle ces propriétés de typage et esquissons les raisons pour lesquelles elles permettent de construire des analyseurs Datalog pour les ACGs du second ordre. Les détails techniques seront développés au cours de la Partie III. Dans un premier temps, nous considérons des termes purs (*i.e.* sans constantes) ; en effet, nous verrons que les occurrences de constantes dans un λ-terme M seront considérées comme des variables libres d'un point de vue du typage associé à M.

Théorème de cohérence

Les problèmes de cohérence furent initialement posés par [Mints, 1979] et reviennent à identifier des séquents de la logique intuitionniste pour lesquels une unique preuve normale existe. Par l'isomorphisme de Curry-Howard, nous pouvons également formuler ce problème par l'identification de typages simples pour lesquels il existe un unique habitant. Nous nous plaçons donc sous l'hypothèse que le système de typage $\mathcal{S}$ recherché est le système de typage simple.

Une première solution aux problèmes de cohérence fut donnée dans [Babaev and Soloviev, 1982], où il fut démontré que les typages balancés vérifient la propriété de cohérence. Plus tard, [Aoto, 1999] démontra que la même propriété était vérifiée par les typages négativement non-dupliquants (nous en donnons une preuve dans le Chapitre suivant). À partir de ce résultat, il est possible de mettre en évidence une certaine famille de λ-termes qui habitent ces typages. Ainsi, en considérant uniquement des termes purs (*i.e.* ne contenant pas de constantes) [Belnap, 1976] montra que les termes affines ont un typage principal balancé, et [Hirokawa, 1991] montra que tout typage balancé est habité par un terme affine. Dans le cas des typages négativement non-dupliquants, [Kanazawa, 2007] montre que les termes dits *quasi-linéaires* ont un typage principal qui est négativement non-dupliquant :

Définition 5.3.1.1. *Un λ-terme M est* quasi-linéaire

- *si $M = x^{\alpha}$.*
- *si $M = \lambda x^{\alpha}.N$ où N est quasi-linéaire, et $x^{\alpha} \in FV(N)$.*
- *si $M = M_1 M_2$, que M_1 et M_2 sont quasi-linéaires, et que pour toute variable $x^{\alpha} \in FV(M_1) \cap FV(M_2)$, $\alpha = a$ est atomique.*

De plus, pour un terme M quasi-linéaire, tout terme N tel que $M \rightarrow_{\beta}^{*} N$ sera également dit quasi-linéaire.

Remarquons que tout terme linéaire est quasi-linéaire.

Exemple 5.3.1.1. Le terme $\lambda x^{a}.f^{a\rightarrow b}x^{a}$ est linéaire (et donc quasi-linéaire). Le terme $\lambda x^{a}.f^{a\rightarrow a\rightarrow b}x^{a}x^{a}$ est un terme quasi-linéaire, mais n'est pas linéaire.

La propriété de cohérence que les termes quasi-linéaires vérifient permet, par ailleurs, de les identifier uniquement grâce à leur typage principal ; en effet, étant donné un terme quasi-linéaire M sous forme normale, tout terme N habitant le typage principal de M vérifie forcément $N \rightarrow_{\beta}^{*} M$.

Cette propriété est particulièrement intéressante dans le cas de la reconnaissance d'un terme M dans une ACG du second ordre $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$. En effet, si le terme M est quasi-linéaire, et si nous imposons à tout terme M' du langage objet de G d'être quasi-linéaire, M appartient à $O(G)$ si il existe un terme $N \in \mathcal{A}(G)$ tel que $\mathcal{L}(N)$ habite le typage principal de M.

Clôture par composition et expansion du sujet

Nous venons de voir que les termes quasi-linéaires vérifient la propriété de cohérence qui offre une solution au problème de l'analyse dans les ACGs du second ordre, sous forme de contraintes de typage. Étant donnée une ACG du second ordre $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$, il nous faut donc un moyen de nous assurer que tout terme du langage objet de G est quasi-linéaire, ou, du moins, vérifie la propriété de cohérence. Or, cette grammaire étant du second ordre, et étant donnée deux constantes, $\mathbf{c_1}$ et $\mathbf{c_2}$, la seule opération permettant de construire des termes du langage objet de G est la composition $\mathcal{L}(\mathbf{c_1})\mathcal{L}(\mathbf{c_2})$, sous les contraintes imposées par le typage de $\mathbf{c_1}$ et $\mathbf{c_2}$. Les termes du langage objet d'une telle grammaire sont donc quasi-linéaires. Il ne reste donc plus qu'à s'assurer que le théorème de cohérence est vérifié pour les termes quasi-linéaires.

Propriété 5.3.1.1. *Le typage principal d'un terme quasi-linéaire est négativement non-dupliquant.*

Considérons une ACG quasi-linéaire $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$: pour toute constante $\mathbf{c}$ de la signature Σ_1, le terme $\mathcal{L}(\mathbf{c})$ est quasi-linéaire. La propriété de clôture est donc que, étant donnés une telle ACG et un terme $N \in \mathcal{A}(G)$, le terme $\mathcal{L}(N)$ a un typage principal négativement non-dupliquant.

Afin de s'assurer de la complétude de cette méthode, nous nous appuyons ensuite sur le théorème d'expansion du sujet qui permet de caractériser deux termes quasi-linéaires M_1 et M_2 par l'ensemble des typages qu'ils habitent :

Propriété 5.3.1.2. *Étant donnés deux termes quasi-linéaires M_1 et M_2 tels que $M_1 \rightarrow_\beta^* M_2$, tout typage (simple) de M_2 est un typage de M_1.*

Ainsi, grâce à la propriété de réduction du sujet dans le λ-calcul simplement typé, les termes M_1 et M_2 partagent exactement les mêmes typages simples. De plus, cette propriété est en particulier vraie pour le typage principal de M_2, qui est alors également le typage principal de M_1. Grâce au théorème de cohérence, nous obtenons donc le corollaire suivant :

Corollaire 5.3.1.1. *Soient un terme quasi-linéaire M_1 sous forme normale et $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ le typage principal de M_1. Un terme quasi-linéaire M_2 habite $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ ssi $M_2 \rightarrow_\beta^* M_1$.*

Grâce à ce corollaire, nous obtenons donc la correction et la complétude de la méthode.

Typages moins généraux préservant les théorèmes de cohérence et d'expansion du sujet

Finalement, un des problèmes rencontrés pour la reconnaissance de termes dans une ACG quasi-linéaire est lié à la gestion de la copie de sous-termes. Tout d'abord, lorsque nous considérons un terme N appartenant au langage abstrait d'une ACG du second-ordre linéaire, et le terme $\mathcal{L}(N)$, alors ce terme habite un typage balancé si nous avons la possibilité d'assigner un type distinct est assigné à chaque occurrence d'une constante dans $\mathcal{L}(N)$. Nous introduirons donc un système de typage, que nous appellerons *système de typage listé*, qui nous permet d'associer un type intersection [Dezani-Ciancaglini et al., 1998] (sous certaines contraintes) à une constante. En utilisant ce système de typage, nous souhaitons distinguer les occurrences de constantes dans un terme M au niveau du typage de M, en ayant la possibilité d'associer un type distinct à chaque occurrence d'une constante dans un terme. Sur ce système de typage, nous héritons de nombreuses propriétés du système de typage simple, et en particulier, de la réduction du sujet, et des théorèmes d'expansion du sujet et de cohérence pour les termes quasi-linéaires ; de plus, à un terme M donné, nous pouvons associer un certain typage (que nous appellerons *typage principal listé de M*), déterminé par le typage principal du terme M' construit en remplaçant chaque occurrence d'une constante dans M par une constante fraîche dans M'.

Exemple 5.3.1.2. Considérons une ACG du second-ordre linéaire $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ et le terme $\mathbf{c_1c_2} \in \mathcal{A}(G)$ tels que $\mathcal{L}(\mathbf{c_1}) = \lambda P.P(\mathbf{ge})$ et $\mathcal{L}(\mathbf{c_2}) = \lambda x.\mathbf{g}x$. Alors, le typage principal de $(\lambda P.P(\mathbf{ge}))(\lambda x.\mathbf{g}x) = \mathcal{L}(\mathbf{c_1c_2})$ est $\langle \mathbf{e} : a, \mathbf{g} : a \to a; \_\rangle \vdash a$ qui n'est pas un typage balancé. Néanmoins, le typage principal listé de ce terme est $\langle \mathbf{e} : a, \mathbf{g} : a \to b \cap b \to c; \_\rangle \vdash c$, qui est un typage balancé.

À présent, considérons deux termes quasi-linéaires M_1 et M_2 tels que $M_1 \to_\beta^* M_2$ et une constante $\mathbf{c} \in Cst(M_2)$, il est aisé de voir que le nombre d'occurrences de $\mathbf{c}$ dans M_2 est supérieur ou égal au nombre d'occurrences de $\mathbf{c}$ dans M_1, puisqu'il est possible de copier des sous-termes par β-réduction. Dans ce cas, il faut pouvoir associer le même type à toutes les occurrences d'une constante $\mathbf{c}$ dans M_2 provenant de la copie d'une même occurrence de $\mathbf{c}$ dans M_1. Nous verrons que résoudre ce problème peut se faire en appliquant des substitutions préservant la propriété de cohérence du typage initial. En effet, grâce à ces substitutions de types et à l'idempotence du connecteur $\cap$, nous réduirons le nombre de types assignés à une constante dans un typage de M_2, simulant ainsi le fait que plusieurs occurrences de cette constante dans M_2 sont identifiées comme provenant d'une même occurrence dans M_1. Nous dirons alors que le typage ainsi obtenu par application successives d'un maximum de substitutions de types est le *typage négativement non-dupliquant le moins général de* M_2.

Remarque 5.3.1.1. Ce mécanisme revient à remplacer, dans un premier temps, chaque occurrence d'une constante dans le terme M_2 par une variable libre, puis, dans un second temps, à unifier autant que possible les variables libres associées à une même constante tout en s'assurant que le typage principal du terme reste négativement non-dupliquant. Cette dernière méthode est celle présentée dans [Kanazawa, 2007].

Exemple 5.3.1.3. Considérons une ACG du second-ordre quasi-affine $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ et le terme $\mathbf{c_1c_2} \in \mathcal{A}(G)$ tels que $\mathcal{L}(\mathbf{c_1}) = \lambda P.P(\mathbf{gee})$ et $\mathcal{L}(\mathbf{c_2}) = \lambda x.\mathbf{g}xx$. Le typage principal listé du terme $M_1 = \mathcal{L}(\mathbf{c_1c_2})$ est :

$$\langle \mathbf{e} : a_1 \cap a_2, \mathbf{g} : a_1 \to a_2 \to b_1 \cap b_1 \to b_1 \to c; \_\rangle \vdash c$$

La forme normale de M_1 est $M_2 = \mathbf{g}(\mathbf{gee})(\mathbf{gee})$, dont le typage principal listé est :

$$\langle \mathbf{e} : a_1 \cap a_2 \cap a_3 \cap a_4, \mathbf{g} : a_1 \to a_2 \to b_1 \cap a_3 \to a_4 \to b_2 \cap b_1 \to b_2 \to c; \_\rangle \vdash c$$

Par la substitution $\sigma = [a_3 \mapsto a_1, a_4 \mapsto a_2, b_2 \mapsto b_1]$, et l'idempotence de $\cap$, ces deux typages sont égaux.

La difficulté de cette méthode revient alors à démontrer qu'il existe un unique typage négativement non-dupliquant le moins général pour M. Il est ensuite facile de montrer que, étant donnée la grammaire catégorielle abstraite $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ en entrée, si il existe un terme $N \in \mathcal{A}(G)$ tel que $\mathcal{L}(N) \to_\beta^* M$, alors $\mathcal{L}(N)$ est également un habitant du typage négativement non-dupliquant le moins général de M, selon le schéma en Figure 5.7 (la flèche en pointillés est obtenue par application de substitutions de types permettant d'obtenir ce typage de M). Ainsi, le typage négativement non-dupliquant le moins général du terme M sera le typage qui nous servira à identifier M avec un terme $\mathcal{L}(N)$ où N appartient à $\mathcal{A}(G)$ et tel que $\mathcal{L}(N) \to_\beta^* M$.

5.3.2 Construction de reconnaisseurs Datalog pour les ACGs du second-ordre

Comme nous l'avons vu, les possibilités offertes par la programmation logique afin de construire des analyseurs de grammaires non-contextuelles de chaînes ont été exploitées depuis les années 1970. En particulier, [Ullman, 1988a] montra que le problème de la reconnaissance dans les grammaires

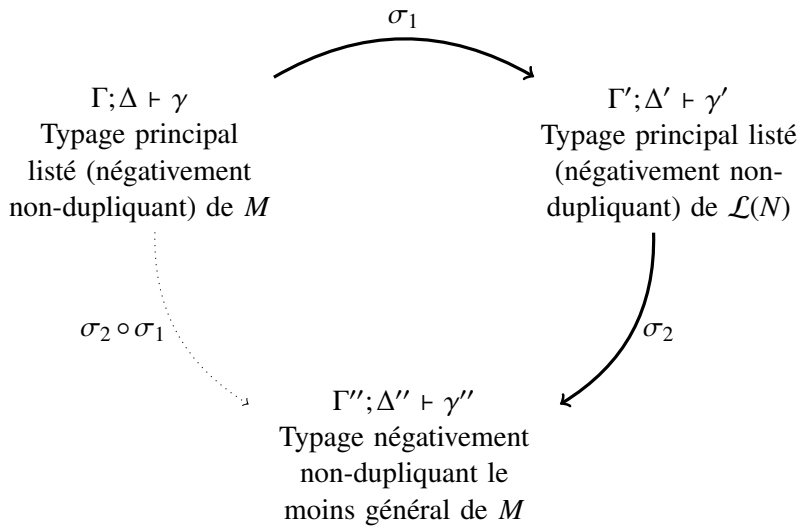


FIG. 5.7 – Relations de typage pour l'analyse dans les ACGs du second ordre

non-contextuelles de chaînes peut être réalisée en temps polynomial dans un programme Datalog, et que ce problème peut donc se réduire à un problème de requêtes dans ce langage de programmation.

Les analyseurs Datalog pour les ACGs du second ordre, présentés dans [Kanazawa, 2007], sont en fait une généralisation de ces analyseurs à des formalismes non-contextuels plus complexes, tels que les grammaires d'arbres adjoints ou les grammaires non-contextuelles multiples. Par ailleurs, nous venons de voir, au cours de la section précédente, que ces méthodes d'analyse reposent sur des propriétés de typage dans le λ-calcul simplement typé. Nous montrons comment ces propriétés s'articulent dans Datalog et donnent lieu à la construction de programmes.

Définition 5.3.2.1. *Étant donné un type simple $\alpha \in \mathcal{T}(\mathcal{A})$, nous définissons la* base $\overrightarrow{\alpha}$ *de α de manière inductive :*

- $\overrightarrow{\alpha} = a$ *si* $\alpha = a$
- $\overrightarrow{\alpha_1 \to \alpha_2} = \overrightarrow{\alpha_1}, \overrightarrow{\alpha_2}$

La base d'un type α est donc une séquence de types atomiques, ordonnée selon l'ordre d'apparition des occurrences de types atomiques dans α de gauche à droite.

Notation 5.3.1. De manière similaire, étant donnée une occurrence d'un prédicat Datalog, nous noterons par $\overrightarrow{x}$ la liste de ses arguments (constituée de variables et constantes Datalog).

Nous donnons les détails de construction de l'analyseur Datalog pour une ACG du second ordre quasi-linéaire $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ et un terme M en entrée.

Construction de la base de données intentionnelle

Le principe est d'associer une règle Datalog à chaque constante $\mathbf{c}$ de $\Lambda(\Sigma_1)$ de la manière suivante :

$$r_{\mathbf{c}} : P(\overrightarrow{\mathbf{x}}) :\!\text{-}\, P_1(\overrightarrow{\mathbf{x_1}}), \ldots, P_m(\overrightarrow{\mathbf{x_m}}), c_1(\overrightarrow{\mathbf{y_{1,1}}}), \ldots, c_1(\overrightarrow{\mathbf{y_{1,p_1}}}) \ldots, c_{f(n)}(\overrightarrow{\mathbf{y_{n,1}}}), \ldots, c_{f(n)}(\overrightarrow{\mathbf{y_{n,p_n}}}).$$

de telle sorte que, étant donné $\langle \mathbf{c_1} : \alpha_{1,1} \cap \ldots \cap \alpha_{1,p_1}, \ldots \mathbf{c_n} : \alpha_{n,1} \cap \ldots \cap \alpha_{n,p_n}; \_ \rangle \vdash \gamma_1 \to \ldots \to \gamma_m \to \gamma$ le typage principal listé de $\mathcal{L}(\mathbf{c})$:

- $\mathbf{c}$ appartient à $\Lambda^{P_1 \to \ldots \to P_m \to P}(\Sigma_1)$.
- les séquences $\overrightarrow{\mathbf{x_1}}, \ldots, \overrightarrow{\mathbf{x_m}}, \overrightarrow{\mathbf{y_{1,1}}}, \ldots, \overrightarrow{\mathbf{y_{n,p_n}}}$ sont uniquement constituées de variables Datalog.
- pour tout $i \in \{1, \ldots, m\}$, $\overrightarrow{\gamma_i} = \overrightarrow{\mathbf{x_i}}$. De plus, $\overrightarrow{\gamma} = \overrightarrow{\mathbf{x}}$
- pour tout $i \in \{1, \ldots, n\}$ et tout $j \in \{1, \ldots, p_n\}$, $\overrightarrow{\alpha_{i,j}} = \overrightarrow{\mathbf{y_{i,j}}}$.

Construction de la base de données extensionnelle

Considérons le typage négativement non-dupliquant le moins général du terme M :

$$\langle \mathbf{e_1} : \delta_{1,1} \cap \ldots \cap \delta_{1,k_1}, \ldots, \mathbf{e_m} : \delta_{m,1} \cap \ldots \delta_{m,k_m}; \_ \rangle \vdash \gamma$$

Alors, pour tout $i \in \{1, \ldots, m\}$ et tout $j \in \{1, \ldots, k_i\}$, nous introduisons le fait $e_i(\overrightarrow{\delta_{i,j}})$. dans la base de données extensionnelle.

Finalement, la requête à exécuter est $?\text{-}S(\overrightarrow{\gamma})$. Si cette requête est évaluée avec la valeur vrai dans le programme ainsi construit, le programme a réussi à construire un terme M' habitant le typage le moins général de M ; ce typage étant uniquement habité, nous obtenons donc que $M =_\beta M'$. De plus, lorsqu'une règle $r_{\mathbf{c}}$ a été instanciée au cours de l'évaluation, alors un typage du terme $\mathcal{L}(\mathbf{c})$ a été construit ; en conséquence le terme M' est de la forme $\mathcal{L}(\mathbf{c_0})M_1 \ldots M_n$, et inductivement, les termes $M_1, \ldots, M_n$ sont de la même forme. Il existe donc un terme $N \in \mathcal{A}(G)$ tel que $M' = \mathcal{L}(N)$.

Finalement, l'évaluation de requête en Datalog étant une évaluation ascendante pure, par analogie avec les algorithmes de reconnaissance de mots dans une grammaire, nous obtenons intuitivement un programme de reconnaissance à la CYK pour les ACGs non-contextuelles, ce qui se démontre aisément.

Exemple

Afin d'illustrer ces résultats, nous donnons un exemple simple, où le typage principal d'un terme quasi-linéaire se confond avec son typage négativement non-dupliquant le moins général. Ainsi, prenons la grammaire $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ où :

- $\Sigma_1 = (\{np, s\}, \{\mathbf{jean}, \mathbf{marie}, \mathbf{aime}\}, \{\mathbf{jean} \mapsto np, \mathbf{marie} \mapsto np, \mathbf{aime} \mapsto np \to np \to s\})$
- $\Sigma_2 = (\{e, t\}, \{jean, marie, aime\}, \{jean \mapsto e, marie \mapsto e, aime \mapsto e \to e \to t\})$
- $\mathcal{L} = \begin{cases} \{np \mapsto (e \to t) \to t, s \mapsto t, \\ \mathbf{jean} \mapsto \lambda P.P\ jean, \mathbf{marie} \mapsto \lambda P.P\ marie, \ \mathbf{aime} \mapsto \lambda PQ.P(\lambda x.Q(\lambda y.aime\ x\ y\} \end{cases}$

et le terme *aime jean marie* en entrée de notre analyseur.

Constante $\mathbf{c}$	Typage principal listé de $\mathcal{L}(\mathbf{c})$
jean	$\langle jean : x_2; \_ \rangle \vdash (x_2 \to x_1) \to x_1$
marie	$\langle marie : x_2; \_ \rangle \vdash (x_2 \to x_1) \to x_1$
aime	$\langle \_; aime : x_2 \to x_3 \to x_4 \rangle \vdash ((x_2 \to x_5) \to x_1) \to ((x_3 \to x_4) \to x_5) \to x_1$

Constante $\mathbf{c}$	Règle Datalog associée
jean	$NP(\mathbf{x_2}, \mathbf{x_1}, \mathbf{x_1})$:- $jean(\mathbf{x_2})$.
marie	$NP(\mathbf{x_2}, \mathbf{x_1}, \mathbf{x_1})$:- $marie(\mathbf{x_2})$.
aime	$S(\mathbf{x_1})$:- $NP(\mathbf{x_2}, \mathbf{x_5}, \mathbf{x_1}), NP(\mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}), aime(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4})$.

FIG. 5.8 – Construction de BD intentionnelle pour un analyseur d'ACG d'ordre 2

Nous obtenons alors un programme Datalog dont la base de données intentionnelle est celle donnée en Figure 5.8.

La base de données extensionnelle est construite sur la base du typage principale $\langle \_; aime : a \to b \to c, jean : a, marie : b \rangle \vdash aime\ jean\ marie : c$. Ainsi, les faits sur cette base sont :

$$jean(a). \quad marie(b). \quad aime(a,b,c).$$

Étape	Typage de $\mathcal{L}(\mathbf{c})$	Instanciation de règles Datalog
1	$\langle \_; aime : a \to b \to c \rangle \vdash \lambda PQ.P(\lambda x.Q(\lambda y.aime\ x\ y)) :$ $((a \to x_5) \to c) \to ((b \to c) \to x_5) \to c$ $\langle \_; jean : a \rangle \vdash \lambda P.P\ jean : (a \to x_1) \to x_1$ $\langle \_; marie : b \rangle \vdash \lambda P.P\ marie : (b \to x_1) \to x_1$	$S(c) :\text{-} NP(a, \mathbf{x_5}, c), NP(b, c, \mathbf{x_5}), aime(a,b,c).$ $NP(a, \mathbf{x_1}, \mathbf{x_1}) :\text{-} jean(a).$ $NP(b, \mathbf{x_1}, \mathbf{x_1}) :\text{-} marie(b).$
2	$\langle \_; jean : a \rangle \vdash \lambda P.P\ jean : (a \to c) \to c$ $\langle \_; marie : b \rangle \vdash \lambda P.P\ marie : (b \to c) \to c$	$NP(a,c,c) :\text{-} jean(a).$ $NP(b,c,c) :\text{-} marie(b).$
3	$\langle \_; aime : a \to b \to c \rangle \vdash \lambda PQ.P(\lambda x.Q(\lambda y.aime\ x\ y)) :$ $((a \to c) \to c) \to ((b \to c) \to c) \to c$	$S(c) :\text{-} NP(a,c,c), NP(b,c,c), aime(a,b,c).$

FIG. 5.9 – Exemple d'exécution d'une requête sur un analyseur Datalog

Finalement la requête Datalog est $?\text{-} S(c)$; l'évaluation de cette requête se déroule ensuite selon le tableau en Figure 5.9. Dans ce tableau, nous montrons comment se construit le dérivation du λ-terme $\mathcal{L}(\mathbf{aime\ jean\ marie})$ pour le typage $\langle \_; aime : a \to b \to c, jean : a, marie : b \rangle \vdash aime\ jean\ marie : c$ représenté par la BD extensionnelle. L'exécution de la requête provoque donc la construction d'un terme $\mathcal{L}(N)$ tel que $N \in \mathcal{A}(G)$ et $\mathcal{L}(N)$ est un habitant du typage donné par la BD extensionnelle. D'après la propriété de cohérence, nous en déduisons que ce terme se β-réduit en M ; ce dernier est donc un terme appartenant au langage objet de G.

Remarque 5.3.2.1. L'instanciation correcte de variables Datalog par des constantes Datalog dans une règle de la base intentionnelle correspond à la construction d'un typage correct pour le terme associé à la règle, ce typage n'étant pas obligatoirement le typage principal de ce dernier.

Nous pouvons remarquer que la propriété de sûreté n'est pas assurée par cette construction. En effet, là règle associée à la constante *jean*, par exemple, ne vérifie pas cette propriété. Par ailleurs, la construction d'un tel programme Datalog assure que son exécution se fera en temps polynomial. Plus précisément, [Kanazawa, 2007] montre que cet algorithme appartient à la classe de complexité **LOGCFL** [3].

5.3.3 Réécriture par ensembles magiques supplémentaires et algorithme de Earley

Nous venons de voir comment construire des analyseurs *à la CYK* pour les grammaires catégorielles abstraites non-contextuelles quasi-linéaires. Par ailleurs, nous avons vu que par une simple réécriture de programmes Datalog, il est possible d'obtenir une évaluation Datalog ascendante avec prédiction descendante. Il est en fait possible d'exploiter la réécriture de programmes par ensembles

[3] **LOGCFL** est la classe des problèmes réductibles en espace logarithmique au problème de la reconnaissance d'une grammaire non-contextuelle (de mots). Cette caractérisation permet donc de relier directement le problème de reconnaissance pour une ACG du second-ordre à celui d'une grammaire non-contextuelle, le premier généralisant le second

magiques supplémentaires afin d'implémenter des analyseurs *à la Earley* pour des grammaires catégorielles abstraites non-contextuelles, comme cela est proposé dans [Kanazawa, 2008].

Considérons le langage faiblement sensible au contexte $\{a^n b^m c^n d^m \mid n,m \in \mathbb{N} - \{0\}\}$, généré par la grammaire catégorielle abstraite $G = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$ où :

- $\Sigma_1 = \begin{cases} \{s, a_1, a_2\} \\ \{\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}, \mathbf{r_4}, \mathbf{r_5}\} \\ \{\mathbf{r_1} \mapsto a_1 \to a_2 \to s, \mathbf{r_2} \mapsto a_1 \to a_1, \mathbf{r_3} \mapsto a_1, \mathbf{r_4} \mapsto a_2 \to a_2, \mathbf{r_5} \mapsto a_2\} \end{cases}$
- $\Sigma_2 = \begin{cases} \{o\} \\ \{a, b, c, d, \epsilon\} \\ \{a \mapsto o \to o, b \mapsto o \to o, c \mapsto o \to o, d \mapsto o \to o, \epsilon \mapsto o\} \end{cases}$
- $\mathcal{L} = \begin{cases} \{s \mapsto o, a_1 \mapsto ((o \to o) \to (o \to o) \to o) \to o, \\ a_2 \mapsto ((o \to o) \to (o \to o) \to o) \to o\} \\ \{\mathbf{r_1} \mapsto \lambda PQ.P(\lambda x_1 x_3.Q(\lambda x_2 x_4.x_1(x_2(x_3(x_4\epsilon))))), \\ \mathbf{r_2} \mapsto \lambda PQ.P(\lambda x_1 x_2.Q(\lambda z.a(x_1 z))(\lambda z.c(x_2 z))), \\ \mathbf{r_3} \mapsto \lambda S.S\,(\lambda z.az)(\lambda z.cz), \\ \mathbf{r_4} \mapsto \lambda PS.P(\lambda x_1 x_2.S\,(\lambda z.b(x_1 z))(\lambda z.d(x_2 z))), \\ \mathbf{r_5} \mapsto \lambda S.S\,(\lambda z.bz)(\lambda z.dz)\} \end{cases}$

Cette grammaire donne lieu à la construction du programme Datalog suivant, que nous décorons afin d'exhiber les variables libres et les variables liées dans le programme, pour la requête $?\text{-}S\,(0)$. Nous remarquerons l'introduction de règles supplémentaires afin de traiter plusieurs décorations du prédicat A_2.

$$
\begin{array}{lll}
r_1 & S^b(x_1) & \text{:-} A_1^{ffffffb}(x_2,x_3,x_4,x_5,x_6,x_1), A_2^{bbfbbbb}(x_5,x_2,x_7,x_4,x_3,x_6), \epsilon^b(x_7). \\
r_2 & A_1^{ffffffb}(x_5,x_2,x_6,x_4,x_7,x_8) & \text{:-} A_1^{fffffb}(x_5,x_1,x_6,x_3,x_7,x_8), a^{bf}(x_1,x_2), c^{bf}(x_3,x_4). \\
r_3 & A_2^{bbfbbb}(x_5,x_2,x_6,x_4,x_7,x_8) & \text{:-} A_2^{bfffbb}(x_5,x_1,x_6,x_3,x_7,x_8), b^{bb}(x_1,x_2), d^{bb}(x_3,x_4). \\
r_4 & A_1^{fffffb}(x_1,x_2,x_3,x_4,x_5,x_5) & \text{:-} a^{ff}(x_1,x_2), c^{ff}(x_3,x_4). \\
r_5 & A_2^{bbfbbb}(x_1,x_2,x_3,x_4,x_5,x_5) & \text{:-} b^{bb}(x_1,x_2), d^{fb}(x_3,x_4). \\
r'_4 & A_2^{bfffbb}(x_5,x_2,x_6,x_4,x_7,x_8) & \text{:-} A_2^{bfffbb}(x_5,x_1,x_6,x_3,x_7,x_8), b^{bb}(x_1,x_2), d^{bf}(x_3,x_4). \\
r'_5 & A_2^{bfffbb}(x_1,x_2,x_3,x_4,x_5,x_5) & \text{:-} b^{bf}(x_1,x_2), d^{ff}(x_3,x_4).
\end{array}
$$

Il ne reste plus qu'à appliquer les règles de transformation selon la méthode des ensembles magiques supplémentaires, afin d'obtenir le programme Datalog de la Figure 5.10.

En guise d'exemple, nous prenons une base extensionnelle où les faits suivants sont fournis : $a(1,0), a(2,1), b(3,2), c(4,3), c(5,4), d(6,5), \epsilon(6)$. Ces faits correspondent donc au λ-terme suivant : $a(a(b(c(c(d\epsilon)))))$, autrement dit, à la chaîne a^2bc^2d en entrée. L'évaluation semi-naïve de la requête $?\text{-}S\,(0)$ est donnée en Figure 5.11.

On obtient donc, par cette méthode, des reconnaisseurs *à la Earley* pour des grammaires ayant des structures de dérivation arborescentes. Un des avantages de cette technique, est que l'algorithme de type Earley est obtenu de manière automatique, en utilisant une méthode de transformation sur le programme initial ; il existe donc une correspondance entre la grammaire, et l'algorithme ascendant avec prédiction descendante ainsi obtenu.

Il faut néanmoins remarquer que la correction par préfixe n'est pas assurée par cette transforma-

m_1	$m\_A_1(x_1)$	$:\text{-}\, m\_S(x_1).$
m_2	$m\_A_2(x_5,x_2,x_4,x_3,x_6)$	$:\text{-}\, sup_{1,1}(x_1,x_2,x_3,x_4,x_5,x_6).$
m_3	$m\_A'_2(x_5,x_7,x_8)$	$:\text{-}\, m\_A_2(x_5,x_2,x_4,x_7,x_8).$
s_{11}	$sup_{1,1}(x_1,x_2,x_3,x_4,x_5,x_6)$	$:\text{-}\, m\_S(x_1), A_1(x_2,x_3,x_4,x_5,x_6,x_1).$
s_{12}	$sup_{1,2}(x_1,x_7)$	$:\text{-}\, sup_{1,1}(x_1,x_2,x_3,x_4,x_5,x_6), A_2(x_5,x_2,x_7,x_4,x_3,x_6).$
s_{21}	$sup_{2,1}(x_8,x_5,x_1,x_6,x_3,x_7)$	$:\text{-}\, m\_A_1(x_8), A_1(x_5,x_1,x_6,x_3,x_7,x_8).$
s_{22}	$sup_{2,2}(x_8,x_5,x_6,x_3,x_7,x_2)$	$:\text{-}\, sup_{2,1}(x_8,x_5,x_1,x_6,x_3,x_7), a(x_1,x_2).$
s_{31}	$sup_{3,1}(x_5,x_2,x_4,x_7,x_8,x_1,x_6,x_3)$	$:\text{-}\, m\_A_2(x_5,x_2,x_4,x_7,x_8), A'_2(x_5,x_1,x_6,x_3,x_7,x_8).$
s_{32}	$sup_{3,2}(x_5,x_2,x_4,x_7,x_8,x_6,x_3)$	$:\text{-}\, sup_{3,1}(x_5,x_2,x_4,x_7,x_8,x_1,x_6,x_3), b(x_1,x_2).$
s_{41}	$sup_{4,1}(x_5,x_1,x_2)$	$:\text{-}\, m\_A_1(x_5), a(x_1,x_2).$
s_{51}	$sup_{5,1}(x_1,x_2,x_4,x_5)$	$:\text{-}\, m\_A_2(x_1,x_2,x_4,x_5,x_5), b(x_1,x_2).$
s_{61}	$sup_{6,1}(x_5,x_7,x_8,x_5,x_1,x_6,x_3)$	$:\text{-}\, m\_A'_2(x_5,x_7,x_8), A'_2(x_5,x_1,x_6,x_3,x_7,x_8).$
s_{62}	$sup_{6,2}(x_5,x_7,x_8,x_6,x_3,x_2)$	$:\text{-}\, sup_{6,1}(x_5,x_7,x_8,x_5,x_1,x_6,x_3), b(x_1,x_2).$
s_{71}	$sup_{7,1}(x_1,x_5,x_2)$	$:\text{-}\, m\_A'_2(x_1,x_5,x_5), b(x_1,x_2).$
r_1	$S(x_1)$	$:\text{-}\, sup_{1,2}(x_1,x_7), \epsilon(x_7).$
r_2	$A_1(x_5,x_2,x_6,x_4,x_7,x_8)$	$:\text{-}\, sup_{2,2}(x_8,x_5,x_6,x_3,x_7,x_2), c(x_3,x_4).$
r_3	$A_2(x_5,x_2,x_6,x_4,x_7,x_8)$	$:\text{-}\, sup_{3,2}(x_5,x_2,x_4,x_7,x_8,x_6,x_3), d(x_3,x_4).$
r_4	$A_1(x_1,x_2,x_3,x_4,x_5,x_5)$	$:\text{-}\, sup_{4,1}(x_5,x_1,x_2), c(x_3,x_4).$
r_5	$A_2(x_1,x_2,x_3,x_4,x_5,x_5)$	$:\text{-}\, sup_{5,1}(x_1,x_2,x_4,x_5), d(x_3,x_4).$
r'_4	$A'_2(x_5,x_2,x_6,x_4,x_7,x_8)$	$:\text{-}\, sup_{6,2}(x_5,x_7,x_8,x_6,x_3,x_2), d(x_3,x_4).$
r'_5	$A'_2(x_1,x_2,x_3,x_4,x_5,x_5)$	$:\text{-}\, sup_{7,1}(x_1,x_5,x_2) d(x_3,x_4).$

FIG. 5.10 – Analyseur *à la Earley* pour le langage $a^n b^m c^n d^m$

ÉTAPE	FAIT	RÈGLE et FAITS
1	$m\_S(0)$	*INIT*
2	$m\_A_1(0)$	$m_1, 1$
3.1	$sup_{4,1}(0,1,0)$	$s_{41}, 2, a(1,0)$
3.2	$sup_{4,1}(0,2,1)$	$s_{41}, 2, a(2,1)$
4.1	$A_1(1,0,5,4,0,0)$	$r_4, 3.1, c(5,4)$
4.2	$A_1(1,0,4,3,0,0)$	$r_4, 3.1, c(4,3)$
4.3	$A_1(2,1,5,4,0,0)$	$r_4, 3.2, c(5,4)$
4.4	$A_1(2,1,4,3,0,0)$	$r_4, 3.2, c(4,3)$
5.1	$sup_{1,1}(0,1,0,5,4,0)$	$s_{11}, 1, 4.1$
5.2	$sup_{1,1}(0,1,0,4,3,0)$	$s_{11}, 1, 4.2$
5.3	$sup_{2,1}(0,1,0,5,4,0)$	$s_{21}, 2, 4.1$
5.4	$sup_{2,1}(0,1,0,4,3,0)$	$s_{21}, 2, 4.2$
5.5	$sup_{1,1}(0,2,1,5,4,0)$	$s_{11}, 1, 4.1$
5.6	$sup_{1,1}(0,2,1,4,3,0)$	$s_{11}, 1, 4.2$
5.7	$sup_{2,1}(0,2,1,5,4,0)$	$s_{21}, 2, 4.1$
5.8	$sup_{2,1}(0,2,1,4,3,0)$	$s_{21}, 2, 4.2$
6.1	$sup_{2,2}(0,2,5,4,0,0)$	$s_{22}, 5.7, a(1,0)$
6.2	$sup_{2,2}(0,2,4,3,0,0)$	$s_{22}, 5.8, a(1,0)$
6.3	$m\_A_2(4,1,5,0,0)$	$m_2, 5.1$
6.4	$m\_A_2(3,1,4,0,0)$	$m_2, 5.2$
6.5	$m\_A_2(4,2,5,1,0)$	$m_2, 5.5$
6.6	$m\_A_2(3,2,4,1,0)$	$m_2, 5.6$
7.1	$A_1(2,0,5,3,0,0)$	$r_2, 6.1, c(4,3)$
7.2	$m\_A'_2(4,0,0)$	$m_3, 6.3$
7.3	$m\_A'_2(3,0,0)$	$m_3, 6.4$
7.4	$m\_A'_2(4,1,0)$	$m_3, 6.5$
7.5	$m\_A'_2(3,1,0)$	$m_3, 6.6$
8.1	$sup_{1,1}(0,2,0,5,3,0)$	$s_{11}, 7.1, 1$
8.2	$sup_{2,1}(0,2,0,5,3,0)$	$s_{21}, 7.1, 2$
8.3	$sup_{7,1}(3,0,2)$	$s_{71}, 7.3, b(3,2)$
9.1	$m\_A_2(3,2,5,0,0)$	$m_2, 8.1$
9.2	$A'_2(3,2,6,5,0,0)$	$r'_5, 8.3, d(6,5)$
10.1	$sup_{5,1}(3,2,5,0)$	$s_{51}, 9.1, b(3,2)$
10.2	$sup_{3,1}(3,2,5,0,0,6,5)$	$s_{31}, 9.1, 9.2$
11	$A_2(3,2,6,5,0,0)$	$r_5, 10.1, d(6,5)$
12	$sup_{1,2}(0,6)$	$s_{12}, 11, 12$
13	$S(0)$	$r_1, 12, \epsilon(6)$

FIG. 5.11 – Évaluation de la requête $?\text{:-}\, S(0)$ sur le programme en Figure 5.10

tion ; ce point est discuté dans [Kanazawa, 2008] où une étape de transformation supplémentaire est mise en place afin d'assurer cette propriété.

Remarquons enfin, que sur l'exemple des grammaires d'arbres adjoints, la complexité en temps de l'algorithme est de l'ordre de n^6, et donne donc des résultats équivalents en complexité aux meilleurs algorithmes connus pour la reconnaissance dans les TAGs [Nederhof, 1999]. Qui plus est, un tel algorithme est obtenu de manière directe à partir de l'algorithme, en utilisant des techniques connues de transformation de programmes. À titre de comparaison, un langage de programmation logique dédié à l'analyse de la langue naturelle a été créé sous le nom de DyALog [de la Clergerie, 2002, de La Clergerie, 2005] ; ce langage permet de créer des analyseurs pour divers formalismes faiblement sensibles au contexte, dont les grammaires d'arbres adjoints, en prenant une grammaire et en construisant l'analyseur tabulaire pendant l'étape de compilation, grâce à l'utilisation d'automates particuliers, tels que les automates de threads [de La Clergerie, 2002]. Cette dernière méthode est donc moins directe que celle proposée par transformation de programmes qui est, de plus, une extension naturelle de techniques existantes pour l'analyse de grammaires non-contextuelles de chaînes.

5.4 Conclusion

Au cours de ce Chapitre, nous avons présenté des algorithmes classiques de reconnaissance de grammaires non-contextuelles que sont les algorithmes CYK et Earley, et des méthodes d'évaluation de requêtes dans des programmes logiques. L'analogie entre les deux permet en fait de construire de manière naturelle des analyseurs grammaticaux dans des langages tels Prolog ou Datalog. Ainsi, à un arbre d'évaluation de requête, correspond un arbre de dérivation grammaticale ; un algorithme descendant peut être simulé par une évaluation de requêtes descendante ; inversement, un algorithme de reconnaissance ascendant, tel l'algorithme CYK, peut être simulé par une évaluation de requêtes ascendante, comme l'évaluation semi-naïve implémentée dans Datalog.

Nous avons décrit de manière informelle les travaux de [Kanazawa, 2007, Kanazawa, 2008], qui étendent cette analogie à des algorithmes de reconnaissance pour les ACGs du second-ordre quasi-linéaires. Cette dernière construction repose sur des propriétés de typage fortes, que sont les propriétés de cohérence et d'expansion du sujet. En effet, suite aux travaux de [Salvati, 2005], Kanazawa montre que le problème de la reconnaissance dans une grammaire catégorielles abstraite du second-ordre peut se reformuler en un problème de typage, instanciant le slogan "parsing as deduction" (reconnaissance comme déduction) en "parsing as type checking" (reconnaissance comme vérification de typage). Cette construction donne lieu à des algorithmes efficaces implémentés dans le langage Datalog ; de plus, grâce à la technique de réécriture de programmes Datalog par la méthode des ensembles magiques supplémentaires, il est possible de transformer un reconnaisseur à la CYK en un reconnaisseur à la Earley.

Au cours de la dernière partie, nous nous attacherons à présenter les travaux qui ont été réalisés afin d'étendre la construction donnée par Kanazawa à des ACGs du second-ordre générant des termes qui ne sont pas quasi-linéaires. En effet, d'après les avantages de cette technique, il est légitime de chercher à l'étendre à des langages de λ-termes plus généraux que les termes quasi-linéaires, sachant que [Salvati, 2010] a démontré que le problème de la reconnaissance pour des ACGs du second-ordre est décidable, mais non-élémentaire. Ce dernier résultat important prouve la décidabilité de la génération de phrases sous l'hypothèse montagovienne mais voue à l'échec l'objectif d'étendre les techniques de Kanazawa à des ACGs du second ordre de termes simplement typés quelconques. Nous restreindrons alors notre étude à des λ-grammaires non-contextuelles générant des termes quasi-affines. Ces grammaires se caractérisent par l'introduction d'opérations d'effacement. La difficulté

principale que nous rencontrerons sera alors de contourner le fait que le théorème d'expansion du sujet n'est pas vérifié pour de tels termes. Néanmoins, nous monterons que l'effacement peut être pris en compte par le reconnaisseur, en lui donnant suffisamment d'informations de manière à ce qu'il puisse déterminer le typage d'occurrences de constantes effacées par β-réduction. Nous détaillerons ainsi, la correction et la complétude de la construction de tels reconnaisseurs.

Troisième partie

Analyse de λ-grammaires non-contextuelles effaçantes

Chapitre 6

Théorèmes de cohérence

Comme nous l'avons vu au cours de la partie précédente, le problème de la reconnaissance dans les ACGs du second-ordre repose en partie sur le théorème dit de cohérence, permettant d'identifier de manière unique un λ-terme avec un de ces typages, dans un système de typage $\mathcal{S}$. Les théorèmes de cohérence de [Babaev and Soloviev, 1982] et [Aoto, 1999] permettent d'identifier des typages uniquement habités (modulo β-expansion) dans le système de typage simple. De plus, ces théorèmes permettent de démontrer la polynomialité de la reconnaissance pour les ACGs du second-ordre de termes linéaires et quasi-linéaires respectivement.

Dans cette partie, nous cherchons, dans un premier temps, à étuider le problème générale de cohérence, à savoir, pour un typage donné, ce typage est-il uniquement habité. En particulier, nous montrerons que ce problème est **PSPACE**-complet. Dans un second-temps, nous étudierons le théorème de cohérence de [Aoto, 1999], et nous caractériserons la famille des termes quasi-affines comme étant la plus grande famille de termes caractérisés par des typages négativement non-dupliquants. Enfin, nous donnerons un nouveau théorème de cohérence pour des typages que nous appellerons *négativement séparants*, et nous comparerons cette famille de typages à d'autres résultats connus sur les théorèmes de cohérence.

6.1 Complexité du problème général

Dans cette partie, nous nous intéressons à l'étude de la complexité du problème général suivant :

UniTerme : "*Étant donné un typage* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \gamma$*, existe-t-il un et un seul habitant de celui-ci dans le système de typage simple ?*"

Grâce à la correspondance de Curry-Howard, un énoncé alternatif du problème **UniTerme** est de considérer le séquent de la logique minimale $\Gamma' \vdash \gamma$ et de chercher à savoir s'il existe une et une seule preuve sous forme normale de ce séquent dans cette logique.

Afin d'étudier la complexité de **UniTerme**, nous introduisons les machines de Turing alternantes, introduites dans [Chandra and Stockmeyer, 1976] :

Définition 6.1.0.1. *Soit un entier* $k \in \mathbb{N}$. *Une* k-machine de Turing alternante *(ATM), est un tuple* $\mathcal{M} = (Q, \Sigma, \delta, q_0, f)$ *où :*

- Q *est un ensemble fini d'éléments appelés* états.
- Σ *est un ensemble fini d'éléments appelés* alphabet de bande.

- $\delta : Q \times \Sigma^k \mapsto \mathcal{P}(Q \times \Sigma^k \times \{\texttt{L}, \texttt{R}\})$ *est la* fonction de transition *(pour un ensemble E, $\mathcal{P}(E)$ désigne l'ensemble des parties de E).*
- $q_0 \in Q$ *est appelé* état initial.
- $f : Q \mapsto \{\wedge, \vee, \texttt{det}, \texttt{accept}, \texttt{reject}\}$ *est la* fonction caractérisante.

Une machine de Turing alternante est en fait une machine de Turing non-déterministe, qui réagit en fonction de l'état dans lequel elle se trouve, et des symboles $a \in \Sigma$ en cours de lecture sur les k-bandes. La fonction de transition δ définit ce comportement en spécifiant le nouvel état, les symboles à écrire sur les bandes, et le mouvement de la tête de lecture sur les bandes (`L` pour gauche et `R` pour droit). La particularité des machines de Turing alternantes par rapport aux machines de Turing réside dans le fait que les états peuvent être de plusieurs types, et en particulier, de type $\wedge$ ou $\vee$. Étant donnés un mot w en entrée et une ATM $\mathcal{M}$, l'exécution de $\mathcal{M}$ sur w se réalise donc de la manière suivante :

- si l'état courant est de type `accept` (*resp.* `reject`), l'exécution est acceptée (*resp.* rejetée).
- si l'état courant est de type $\wedge$ et que toutes les exécutions passant par cet état sont acceptées, alors l'exécution est acceptée ; si une de ces exécutions est rejetée, l'exécution à l'état courant est rejetée.
- si l'état courant est de type $\vee$ et que toutes les exécutions passant par cet état sont rejetées, alors l'exécution est rejetée ; si une de ces exécutions est acceptée, l'exécution à l'état courant est acceptée.

Un état de type `det` est un état à partir duquel il existe une seule exécution possible en une étape d'exécution. Un tel état peut donc être considéré aussi bien de type $\wedge$ que de type $\vee$.

Remarque 6.1.0.1. Si pour tout état q d'une ATM $\mathcal{M}$, $f(q) \neq \wedge$, la machine $\mathcal{M}$ est simplement une machine de Turing non-déterministe.

L'exécution d'une ATM pourra donc se représenter sous une forme d'arbre dont les nœuds sont étiquetés par $\wedge$ ou $\vee$. Une exécution acceptante sera alors représentée par un arbre où tout nœud étiqueté $\vee$ n'aura qu'un fils, *c.a.d.* l'état suivant de l'exécution acceptante, et tout nœud $\wedge$ aura plusieurs fils, assurant l'acceptation de l'exécution.

Dans le cas du problème **UniTerme**, nous utiliserons le résultat suivant :

Propriété 6.1.0.1. *Un problème est résolu en temps polynomial par une ATM si et seulement s'il est dans la classe **PSPACE** des problèmes solubles en espace polynomial.*

Théorème 6.1.0.1. *Le problème **UniTerme** est **PSPACE**-complet.*

Démonstration. Nous montrons d'abord que **UniTerme** est dans **PSPACE**, en construisant une 2-machine de Turing alternante $\mathcal{M}$ résolvant le problème suivant, équivalent à **UniTerme** : "étant donné un séquent de la logique intuitionniste, existe-t-il une unique démonstration de ce séquent, modulo détours ?".

Considérons, en entrée du problème, un séquent intuitionniste $\Gamma \vdash \gamma$ dont le nombre d'occurrences de formules atomiques est n. De plus, nous notons par :

- A l'ensemble des occurrences des formules atomiques ayant une occurrence dans $\Gamma \vdash \gamma$
- pour une formule atomique $a \in A$, H_a est le multi-ensemble, contenant les occurrences de formules dans Γ, dont la conclusion est a. Ainsi, étant donnés un environnement Γ, et une formule α, nous écrirons $\alpha \in H_a(\Gamma)$ ssi α est de la forme $\alpha_1 \to \ldots \to \alpha_n \to a$, et $\Gamma = \alpha_1, \ldots, \alpha, \ldots, \alpha_n$.

La machine utilise deux bandes : sur la première (notée S), nous écrivons le séquent à démontrer. Sur la seconde (notée H), nous enregistrons les formules atomiques qui apparaissent en conclusion du séquent à démontrer lorsque ce dernier est de la forme $\Delta \vdash a$, où a est un type atomique. Ainsi, la

bande H nous permet de garder en mémoire les formules à démontrer afin d'éviter de rentrer dans des boucles d'exécution infinies.

Nous construisons l'ATM $\mathcal{M} = (Q, \Sigma^2, \delta, q^+_\top, f)$ de la manière suivante :

1. $Q = \{q_{accept}, q_{reject}, q^+_\bot \cup q^-_\bot \cup q^+_\top \cup q^-_\top\}$. Intuitivement, nous séparons l'exécution en deux parties exécutées en parallèle ; la première cherche à construire une démonstration du séquent, la seconde cherche à montrer qu'il n'en existe pas d'autres. Cette partition est donnée par l'ensemble $\{q^+_\top \cup q^-_\top\}$ pour la recherche d'une démonstration, et $\{q^+_\bot \cup q^-_\bot\}$ pour la recherche de la non-dérivabilité d'un séquent donné. Par ailleurs, nous utilisons des polarités afin de marquer deux étapes distinctes dans la recherche de démonstration : la sélection d'une formule dans l'antécédent permettant de démontrer la conclusion du séquent (pour un état négatif) et la dérivation d'un séquent (en positif).
   De plus, $f(q^+_\top) = f(q^-_\bot) = \vee$, $f(q^-_\top) = f(q^+_\bot) = \wedge$, $f(q_{accept}) =$ `accept` et $f(q_{reject}) =$ `reject`.
2. l'alphabet Σ est défini comme $A \cup \{\rightarrow, (,), \vdash, \epsilon\}$. Tous ces symboles peuvent être écrits sur S ; seuls les types atomiques de A et ϵ peuvent être écrits sur H.
3. étant donnés un séquent $s = \Delta \vdash \alpha_1 \rightarrow \ldots \rightarrow \alpha_m \rightarrow a$ $(m \geq 0)$ sur S, un ensemble de types atomiques A' sur H et un état $q \in Q$, la fonction de transition $\delta(q, s, A')$ est définie de la manière suivante :
   - si $q = q^+_\top$ (*resp.* $q^+_\bot$) et $m > 0$, nous écrivons $\Delta, \alpha_1, \ldots, \alpha_n \vdash a$ sur S et restons dans l'état q. Si $m = 0$, alors :
     (a) si $a \in A'$ ou $H_a(\Delta) = \emptyset$, l'état suivant est q_{reject} (*resp.* q_{accept}).
     (b) sinon, a est écrit sur H et pour chaque $\gamma \in H_a(\Delta)$, la machine écrit le séquent $\Delta \vdash \gamma$ sur S et passe dans l'état $q^-_\top$ (*resp.* $q^-_\bot$).
   - si $q = q^-_\top$ (*resp.* $q^-_\bot$). Alors, pour $a \in A$ tel que $\alpha \in H_a(\Delta)$, pour tout $\beta \in H_a(\Gamma) - \{\alpha\}$, le séquent $\Delta \vdash \beta$ est écrit sur S et l'état suivant est $q^+_\top$ (*resp.* $q^-_\bot$). De plus,
     (a) si $m = 0$, il existe une exécution se poursuivant dans l'état q_{accept} (*resp.* q_{reject}).
     (b) si $m \geq 0$, il existe m exécutions suivantes telles que pour tout $i \in \{1, \ldots, m\}$ nous écrivons le séquent $\Delta \vdash \alpha_i$ sur S et passons dans l'état $q^+_\top$ (*resp.* $q^+_\bot$).

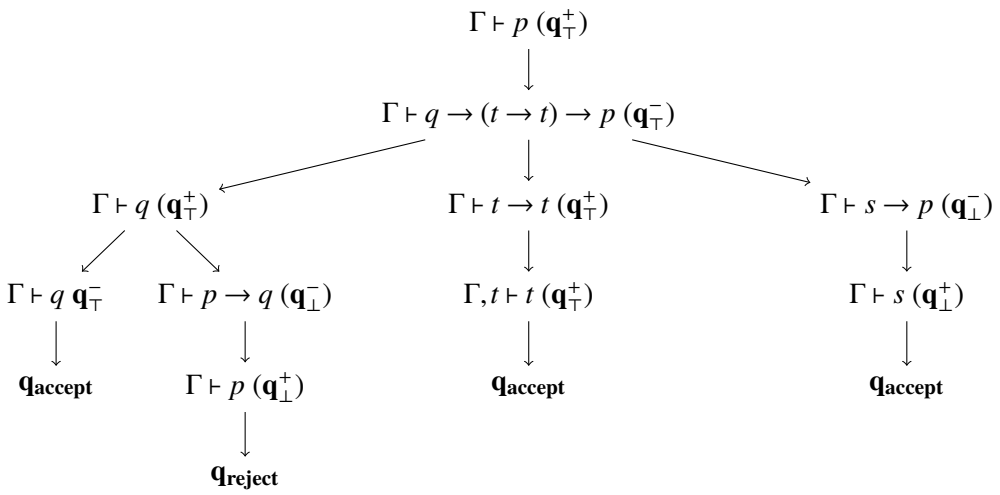


FIG. 6.1 – Exemple d'exécution d'une ATM pour un problème **UniTerme**

Afin de montrer comment un telle machine résout le problème **UniTerme**, considérons le séquent $s \to p, q, p \to q, q \to (t \to t) \to p \vdash p$ (noté $\Gamma \vdash p$) ; nous donnons un arbre d'exécution pour ce séquent en Figure 6.1, où tous les fils d'un nœud sont présentés ssi ce nœud est un nœud $\wedge$. L'exécution, dans ce cas, échoue ; en effet, sur la première et la troisième feuille (en partant de la gauche), la machine trouve une preuve du séquent d'origine. De plus, la quatrième feuille montre que le premier choix de formules dans l'environnement Γ est bien le seul pouvant aboutir à l'unicité de la démonstration. Par contre, la deuxième feuille nous amène à redémontrer un type atomique déjà prouvé, ce qui nous indique qu'il existe une infinité de démonstrations.

La complexité en temps associée à cette machine est polynomiale ; en effet, ce paramètre de complexité correspond à la profondeur de l'arbre d'exécution. Or, étant donné un séquent $\Delta \vdash \alpha$, son traitement est borné par le cas où toutes les formules ont été utilisées pour démontrer la prouvabilité (ou non-prouvabilité d'une formule) ; dans ce cas, il est possible de retomber sur le même état à la suite d'un nombre d'étapes en $O(n)$. La complexité en espace étant linéaire (celle-ci est donnée par le nombre maximal de symboles sur H, qui est linéaire en n, et sur S, qui est également linéaire en n), nous pouvons en conclure que **UniTerme** est dans **PSPACE**.

La complétude se vérifie aisément à partir du résultat de [Statman, 1979] montrant que la prouvabilité dans la logique intuitionniste implicative est un problème **PSPACE**-complet : considérons le séquent $\alpha \to p \vdash p \to p$, tel que la formule atomique p n'apparaît pas dans α. Il existe alors une preuve de ce séquent à partir de la règle axiome produisant le séquent $\alpha \to p, p \vdash p$. Par ailleurs, cette démonstration est unique si et seulement si $\vdash \alpha$ n'est pas prouvable, ce qui est un problème **PSPACE**-complet (car **co-PSPACE=PSPACE**). Nous pouvons en conclure que **UniTerme** est **PSPACE**-complet. □

En considérant uniquement la propriété de cohérence, il n'est donc pas possible d'utiliser une méthode d'analyse d'ACGs du second-ordre, où pour chaque terme du vocabulaire objet, nous serions amenés à vérifier si ce terme est le seul habitant de son typage principal. Une telle méthode serait en effet inefficace (en supposant que $\mathbf{P} \neq \mathbf{PSPACE}$), d'après le résultat ci-dessus. Nous nous attachons donc à caractériser des typages qui sont uniquement habités, puis à caractériser les termes habitant de tels typages, afin de s'assurer qu'une telle propriété est vérifiée pour tout terme du langage objet d'une ACG du second-ordre.

6.2 Termes quasi-affines et typages négativement non-dupliquants

Dans cette partie, nous étudions le théorème de cohérence de [Aoto, 1999], en donnant une preuve alternative de ce théorème. Cette preuve est donnée en utilisant la sémantique des jeux, permettant ainsi de voir l'apport de cette notion à l'étude de ces théorèmes. Qui plus est, à la manière de [Belnap, 1976] et [Hirokawa, 1991] pour les termes affines et les typages balancés, nous mettons en avant la relation entre termes quasi-affines et typages négativement non-dupliquants.

6.2.1 Cohérence et typages négativement non-dupliquants

Dans un premier temps, nous nous intéressons aux typages négativement non-dupliquants caractérisés dans [Aoto, 1999]. Par le biais de la sémantique des jeux (cf. Chapitre 3 pour les notions de jeux et stratégies), il est en effet, évident que de tels typages sont uniquement habités :

Lemme 6.2.1.1. *Soit $\Gamma \vdash \alpha$ un typage négativement non-dupliquant et $A = (M, \tau)$ l'arène associée et Σ une stratégie gagnante sur A. Pour chaque séquence S de Σ, il existe au plus une occurrence de chaque O-coup n dans S.*

Démonstration. Supposons que ce ne soit pas le cas : alors il existe une séquence $S \in \Sigma$ de la forme suivante :

$$S = S_1.(n,i_1).(m_1,j_1).S_2.(n,i_2).(m_2,j_2)$$

où n est un O-coup de M. Or, puisque le typage est négativement non-dupliquant, il existe un unique P-coup $m = m_1 = m_2$ tel que $\tau(m) = \tau(n)$. Étant donné un préfixe de longueur paire S'_2 de la séquence $S_2.(n,i_2).(m,j_2)$, nous montrons par induction sur $|S'_2|$ que $S.S'_2$ appartient à Σ (à indice d'occurrences de coups près) :

- Si $|S'_2| = 0$, le résultat est direct.
- Sinon, nous supposons que le résultat est vrai pour tout préfixe de S_2 de longueur p, et considérons $|S'_2| = p+2 \leq |S_2|$. Alors $S'_2 = S''_2.(n',k).(m',h)$, et par l'hypothèse d'induction, $S.S''_2$ appartient à Σ. Par construction d'une stratégie, il existe une séquence $S.S''_2.(n',k').(m'',h')$ dans Σ et le typage étant négativement non-dupliquant, $m = m''$.

Par conséquent, il existe une séquence $S.S_2.(n,i').(m,k')$ dans Σ ; ce résultat peut être itéré, ce qui implique que Σ est un ensemble infini de séquences, ce qui est impossible par définition d'une stratégie. Par l'absurde, dans tout séquence de Σ, un O-coup a au plus une occurrence. □

Théorème 6.2.1.1 ([Aoto, 1999])**.** *Un typage négativement non-dupliquant est habité par au plus un terme M sous forme β-normale.*

Démonstration. Prenons un typage négativement non-dupliquant $\Gamma \vdash \alpha$ et supposons qu'il soit habité par un λ-terme N sous forme β-normale et η-longue pour ce typage. Soit $A = (M,\tau)$ l'arène de typage associée à $\Gamma \vdash \alpha$, et Σ la stratégie gagnante sur A telle que $[\![\Sigma]\!]_A = N$. Supposons qu'il existe un deuxième habitant N' de $\Gamma \vdash \alpha$, *i.e.* une stratégie gagnante $\Sigma' \neq \Sigma$ sur A. Considérons alors une séquence $S \in \Sigma'$. Par induction sur $|S|$, nous montrons que S appartient à Σ.

- si $|S| = 2$ alors $S = (\epsilon,0).(m,0)$, pour un P-coup m. Comme le typage est négativement non-dupliquant, m est le seul P-coup tel que $\tau(m) = \tau(\epsilon)$. Donc $S \in \Sigma$.
- si $|S| = 2p+2$, alors S est de la forme $S_1.(n,k).(m,i)$. D'après l'hypothèse d'induction, $S_1 \in \Sigma$; de plus, étant donné $S_1 = S'_1.(p,j)$, le P-coup p vérifie $p \vdash n$. La stratégie Σ étant gagnante, elle est complète pour tous les coups o tels que $p \vdash o$, et il existe donc un P-coup m' et $i' \in \mathbb{N}$ tels que $S_1.(n,k).(m',i')$ appartient à Σ. Puisque le typage est négativement non-dupliquant, $m = m'$. De plus, d'après le Lemme 6.2.1.1, le O-coup n' tel que $n' \vdash m$ a une unique occurrence dans $S_1.(n,k)$, d'où $i = i'$.

□

On peut remarquer que l'idée selon laquelle les typages négativement non-dupliquants ont au plus un habitant est simple : lors de la construction d'une stratégie Σ, étant donnés une séquence $S.(m,i)$ dans Σ et un O-coup n tel que $m \vdash n$, il existe au plus un P-coup m' tel que $S.(m,i).(n,j).(m',k)$ soit une séquence innocente, *i.e.* tel que $\tau(n) = \tau(m')$ et m' est accessible.

Afin de pouvoir exploiter ce résultat dans le cadre de l'analyse de grammaires de termes, la deuxième partie de notre recherche consiste à caractériser les λ-termes habitant de tels typages. Pour ce faire, nous adoptons, à nouveau, une approche basée sur la sémantique des jeux.

6.2.2 Termes quasi-affines

Nous cherchons à présent, à caractériser les habitants des typages négativement non-dupliquants. Nous caractérisons ces termes en restant dans le cadre de la sémantique des jeux, par l'intermédiaire des stratégies gagnantes sur les arènes de ces typages. Les termes que nous mettrons ainsi en évidence ont une caractérisation essentiellement basée sur les occurrences de P-coups dans la stratégie qui leur

est associée, et donc, basée sur les variables du terme (nous avons effectivement vu, au cours du Chapitre 3, qu'il existe une analogie entre les occurrences de P-coups dans une stratégie et les variables du terme correspondant à cette stratégie). La principale difficulté dans la définition de cette relation étant liée au nommage des variables et au fait de pouvoir les désigner aisément, nous adopterons la convention de nommage de Barendregt qui permet de distinguer chaque variable d'un terme par son nom.

Définition 6.2.2.1. *Étant donné un terme N sous forme β-normale et η-longue pour un typage $\Gamma \vdash \alpha$, pour tout $i, j \in \mathbb{N}$, nous définissons la relation binaire $\mathbb{I}_{ij}^N$ sur $V(N) \times V(N)$ telle que $x \mathbb{I}_{ij}^N y$ est vraie ssi*

$$N = C[xN_1 \dots N_{i-1}(\lambda x_1 \dots x_{j-1} y x_{j+1} \dots x_n.N)N_{i+1} \dots N_m]$$

Définition 6.2.2.2. *Soit un λ-terme N sous forme normale et η-longue pour un certain typage. En supposant que ce terme vérifie la convention de nommage de Barendregt, la relation $\approx_N$ sur $V(N) \times V(N)$ est définie par $x \approx_N y$ ssi :*

1. *$x = y$.*
2. *ou bien il existe deux variables z_1, z_2 et i, j dans $\mathbb{N}$ tels que $z_1 \mathbb{I}_{ij}^N x$, $z_2 \mathbb{I}_{ij}^N y$, et $z_1 \approx_N z_2$*

Ainsi, deux variables x et y d'un terme N, qui vérifient la relation $\approx_N$ sont récursivement introduites par la même variable.

Exemple 6.2.2.1. Soit le terme $N = \lambda f g.g(f(\lambda x_1 y_1.x_1(\lambda z_1.z_1)))(f(\lambda x_2 y_2.y_2(x_2(\lambda z_2.z_2))))$; les relations $x_1 \approx_N x_2$, $y_1 \approx_N y_2$ et $z_1 \approx_N z_2$ sont vérifiées.

La définition de la relation $\approx_N$ peut être naturellement étendue à des sous-termes du terme N :

Définition 6.2.2.3. *Soient un λ-terme N sous forme normale et η-longue pour un typage $\Gamma \vdash \alpha$, et deux de ses sous-termes N_1 et N_2 ; alors la relation $N_1 \approx_N N_2$ est vérifiée ssi :*

- *$N_1 = x_1$, $N_2 = x_2$ et $x_1 \approx_N x_2$*
- *$N_1 = \lambda x_1.P_1$, $N_2 = \lambda x_2.P_2$, $x_1 \approx_N x_2$ et $P_1 \approx_N P_2$*
- *$N_1 = x_1 P_1 \dots P_n$, $N_2 = x_2 Q_1 \dots Q_n$, $x_1 \approx_N x_2$ et pour tout $i \in \{1, \dots, n\}$, $P_i \approx_N Q_i$.*

Exemple 6.2.2.2. Dans l'exemple 6.2.2.1, la relation $x_1(\lambda z_1.z_1) \approx_N x_2(\lambda z_2.z_2)$ est vérifiée. Par contre, la relation $f(\lambda x_1 y_1.x_1(\lambda z_1.z_1)) \approx_N f(\lambda x_2 y_2.y_2(x_2(\lambda z_2.z_2)))$ est fausse.

Remarquons que, pour un terme $N = C_1[C_2[P_1][P_2]]$ sous forme normale et η-longue pour un typage $\Gamma \vdash \alpha$ et où la relation $P_1 \approx_N P_2$ est vérifiée, et un terme $N' = C_1[(\lambda x.C_2'[x][x])P]$ β-réductible en N, alors N' est également un habitant du typage $\Gamma \vdash \alpha$. Ainsi, sur l'exemple 6.2.2.1, le terme $(\lambda F f g.g(f(\lambda x_1 y_1.F x_1))(f(\lambda x_2 y_2.y_2(F x_2))))(\lambda x.x(\lambda z.z))$ est un terme simplement typable dont le typage principal est celui de N.

Définition 6.2.2.4. *Soit un terme N sous forme normale et η-longue pour son typage principal $\Gamma \vdash \alpha$. Alors, N est dit* quasi-affine *si un même type γ est assigné à deux sous-termes N_1 et N_2 de N dans $\Gamma \vdash \alpha$ ssi $N_1 \approx_N N_2$.*

Exemple 6.2.2.3. Le λ-terme de l'exemple 6.2.2.1 n'est pas quasi-affine puisque $f(\lambda x_1 y_1.x_1(\lambda z_2.z_2))$ et $f(\lambda x_2 y_2.y_2(x_2(\lambda z_2.z_2)))$ ont le même type dans le typage principal de N mais ne vérifient pas la relation $\approx_N$. Par contre, le terme $N = f(h(\lambda x_1 y_1.y_1))(h(\lambda x_2 y_2.y_2))$ est quasi-affine et peut être β-étendu en $(\lambda z.f z z)(h(\lambda x y.y))$. Ce dernier a le même typage principal que N. Remarquons qu'un type atomique est assigné à la variable z dans tout typage pour lequel N est η-long.

Remarque 6.2.2.1. Ces termes sont appelés quasi-affines en référence aux termes quasi-linéaires de [Kanazawa, 2007]. En effet, étant donné un terme N quasi-affine tel que nous l'avons défini et son typage principal $\Gamma \vdash \alpha$, il existe un terme N' tel que $N' \to_\beta^* N$, ayant $\Gamma \vdash \alpha$ pour typage principal et tel que si une variable a plusieurs occurrences dans N, alors il lui est assigné un type atomique dans $\Gamma \vdash \alpha$.

Nous montrons à présent que de la même manière que [Belnap, 1976, Hirokawa, 1991] ont démontré une certaine équivalence entre termes affines et typages balancés, les typages négativement non-dupliquants ne peuvent être habités que par des termes quasi-affines ; de même, ces derniers ont pour typage principal un typage négativement non-dupliquant.

Remarque 6.2.2.2. Pour la définition d'une variable comme réalisation d'une occurrence de P-coups, se référer à l'algorithme 3.3.4.1.

Lemme 6.2.2.1. *Étant donnés une arène de typage $A = (M, \tau)$, une stratégie gagnante Σ sur A et le terme $N = [\![\Sigma]\!]_A$, s'il existe un P-coup $n \in M^P$ tel que les occurrences de variables x_1 et x_2 sont des réalisations de n dans $[\![\Sigma]\!]_A$, alors $x_1 \approx_N x_2$.*

Démonstration. Tout d'abord, pour une séquence justifiée S sur A, nous introduisons la mesure $J(S)$ comme suit :

- $J((\epsilon, 0)) = 0$.
- $J(S_1.(n,i).S_2.(m,i)) = 1 + J(S_1.(n,i))$ si m est un P-coup.
- $J(S.(n,i)) = J(S)$ si n est un O-coup.

Cette fonction retourne donc le nombre d'occurrences de P-coups justifiant, au sens de $\vdash_S^*$ (la clôture transitive et réflexive de $\vdash_S$) la dernière occurrence de P-coup jouée.

À présent, supposons que deux variables x_1 et x_2 sont des réalisations du même P-coup m dans $[\![\Sigma]\!]_A$, c'est-à-dire que :

$$N = \begin{cases} C_1[[\![(n_1,i_1).(m,j_1).[\mathbb{T}_1 \dots \mathbb{T}_n], V_1]\!]_A] = C_1[\lambda\overline{z_1}.x_1[\![\mathbb{T}_1, V_1']\!]_A \dots [\![\mathbb{T}_n, V_1']\!]_A] \\ C_2[[\![(n_2,i_2)(m,j_2).[\mathbb{T}_1' \dots \mathbb{T}_n'], V_2]\!]_A] = C_2[\lambda\overline{z_2}.x_2[\![\mathbb{T}_1', V_2']\!]_A \dots [\![\mathbb{T}_n', V_2']\!]_A] \end{cases}$$

Soient $S_1.(m,j_1)$ et $S_2.(m,j_2)$ les séquences appartenant à Σ finissant sur les occurrences de m dans les équations ci-dessus. Ces séquences sont de longueur paire, puisque se terminant sur une occurrence de P-coup. Nous prouvons $x_1 \approx_N x_2$ par induction sur $p = max(J(S_1.(m,j_1)), J(S_2.(m,j_2)))$:

- Si $p = 1$, alors $S_1 = S_2 = (\epsilon, 0)$, $j_1 = j_2 = 0$ et $x_1 = x_2$ sont des occurrences de la même variable.
- Supposons, à présent, que la propriété est vraie pour tout entier $p > 1$. Puisqu'il existe un unique O-coup n tel que $n \vdash m$, nous obtenons $S_1'.(n,j_1) \sqsubseteq S_1$ et $S_2'.(n,j_2) \sqsubseteq S_2$. Puisque $p > 1$, il existe un P-coup m' tel que $m' \vdash n$, et des entiers positifs k_1 et k_2 tels que $S_1' = S_1''.(m',k_1)$ et $S_2' = S_2''.(m',k_2)$. Par définition, nous obtenons

  $$max(J(S_1''.(m',k_1)), J(S_2''.(m',k_2))) = max(J(S_1.(m,j_1)), J(S_2.(m,j_2))) - 1$$

  Par conséquent, étant données y_1 et y_2 les variables réalisant respectivement (m',k_1) et (m',k_2) dans $[\![\Sigma]\!]_A$, l'hypothèse d'induction nous permet d'obtenir $y_1 \approx_N y_2$. Finalement, pour $r \in \{1,2\}$,

  $$\begin{aligned} N &= C_k'[[\![(n',l_r).(m',k_r)[\mathbb{T}_{k,1} \dots \mathbb{T}_{k,j} \dots \mathbb{T}_{k,q}], W_k]\!]_A] \\ &= C_k'[\lambda\overline{z_k'}.y_k[\![\mathbb{T}_{k,1}, W_k]\!]_A \dots [\![\mathbb{T}_{k,j}, W_k]\!]_A \dots [\![\mathbb{T}_{k,q}, W_k]\!]_A] \\ &= C_k'[\lambda\overline{z_k'}.y_k[\![\mathbb{T}_{k,1}, W_k]\!]_A \dots (\lambda x_{1,k}^j \dots x_{i-1,k}^j x_k x_{i+1,k}^j \dots x_{r,k}^j.N_{k,j}) \dots [\![\mathbb{T}_{k,q}, W_k]\!]_A] \end{aligned}$$

  Par conséquent $y_k \mathtt{I}_{ji}^N x_k$ est vérifié, ce qui permet de déduire $x_1 \approx_N x_2$ d'après le fait que $y_1 \approx_N y_2$ et la définition de $\approx_N$.

□

Lemme 6.2.2.2. *Soient une arène $A = (M, \tau)$, une stratégie Σ sur A et le terme $N = [\![\Sigma]\!]_A$.*
Si deux variables x_1, x_2 de N vérifient $x_1 \approx_N x_2$, alors il existe un P-coup $n \in M^P$ dont x_1 et x_2 sont des réalisations dans $[\![\Sigma]\!]_A$

Démonstration. Considérons deux variables x_1 et x_2 dans le terme N, telles que $x_1 \approx_N x_2$. Nous procédons par induction sur la définition de la relation $\approx_N$:

- si $x_1 = x_2 = x$. D'après la définition de l'interprétation de Σ,

$$N = \begin{cases} C_1[[\![(n_1, j_1).(m_1, i_1).\mathbb{T}_1 \dots \mathbb{T}_n, V_1]\!]_A] = C_1[\lambda\overline{z_1}.x_1 N_1 \dots N_n] \\ \\ C_2[[\![(n_2, j_2).(m_2, i_2).\mathbb{T}_1' \dots \mathbb{T}_n', V_2]\!]_A] = C_2[\lambda\overline{z_2}.x_2 P_1 \dots P_n] \end{cases}$$

  et $m_1 = m_2 = m$ appartient à M^P, et $i_1 = i_2 = i$. De plus, (m, i) est l'unique occurrence de P-coup tel que $((m, i), x)$ appartient à $V_1 \cap V_2$.
- s'il existe deux variables z_1 et z_2 telles que $z_1 \approx_N z_2$ et des entiers positifs i, j tels que $z_1 \mathtt{I}_{ij}^N x_1$ et $z_2 \mathtt{I}_{ij}^N x_2$. D'après l'hypothèse d'induction, il existe un P-coup m dont z_1 et z_2 sont des réalisations dans Σ :

$$N = \begin{cases} C_1[[\![(n_1, j_1).(m, i_1).\mathbb{T}_1 \dots \mathbb{T}_n, V_1]\!]_A] = C_1[\lambda\overline{z_1'}.z_1 N_1 \dots N_n] \\ \\ C_2[[\![(n_2, j_2).(m, i_2).\mathbb{T}_1' \dots \mathbb{T}_n', V_2]\!]_A] = C_2[\lambda\overline{z_1'}.z_2 P_1 \dots P_n] \end{cases}$$

  Puisque les relations $z_1 \mathtt{I}_{ij}^N x_1$ et $z_2 \mathtt{I}_{ij}^N x_2$ sont vérifiées, nous savons que les termes N_i et P_i sont respectivement égaux à $\lambda y_1 \dots y_{j-1} x_1 y_{j+1} \dots y_p.N'$ et $\lambda y_1' \dots y_{j-1}' x_2 y_{j+1}' \dots y_p'.P'$. Alors, par définition d'une interprétation, x_1 et x_2 doivent être des réalisations des coups $(m \cdot i) \cdot j$ dans Σ.

□

Des deux lemmes précédents, nous déduisons l'équivalence suivante :

Lemme 6.2.2.3. *Étant donnés une arène de typage $A = (M, \tau)$, une stratégie Σ sur A et le terme $N = [\![\Sigma]\!]_A$, alors deux variables x_1, x_2 de N vérifient $x_1 \approx_N x_2$ ssi il existe $m \in M^P$ tel que x_1 et x_2 sont des réalisations du P-coup m dans $[\![\Sigma]\!]_A$.*

À présent, nous montrons les résultats énoncés plus haut établissant l'équivalence entre typages négativement non-dupliquants et termes quasi-affines. Indépendamment de ces travaux, le résultat a également été démontré par Kanazawa (la preuve est à paraître dans [Kanazawa, 2011]).

Théorème 6.2.2.1. *Si un typage négativement non-dupliquant $\Gamma \vdash \alpha$ est habité par un terme N, ce dernier est quasi-affine.*

Démonstration. Considérons la stratégie Σ sur l'arène A associée au typage $\Gamma \vdash \alpha$ telle que $[\![\Sigma]\!]_A = N$. D'après le Théorème 6.2.1.1, Σ est l'unique stratégie gagnante sur A. Supposons que $N = C_1[N_1] = C_2[N_2]$, où $N_1 = \lambda x_1 \dots x_q.z_1 P_1 \dots P_{p_1}$ et $N_2 = \lambda y_1 \dots y_q.z_2 Q_1 \dots Q_{p_2}$ sont tous les deux d'un certain type δ dans $\Gamma \vdash \gamma$. Alors, nous pouvons écrire $[\![\Sigma]\!]_A$ sous la forme :

$$[\![\Sigma]\!]_A = \begin{cases} C_1[[\![\mathbb{T}_1, V_1]\!]_A] = C_1[[\![(n_1, i_1).(m_1, j_1).[\mathbb{T}_{1,1}, \dots, \mathbb{T}_{1,p_1}], V_1]\!]_A] = C_1[N_1] \\ C_2[[\![\mathbb{T}_2, V_2]\!]_A] = C_2[[\![(n_2, i_2).(m_2, j_2).[\mathbb{T}_{2,1}, \dots, \mathbb{T}_{2,p_2}], V_2]\!]_A] = C_2[N_2] \end{cases}$$

où n_1, n_2 sont des O-coups, et m_1, m_2 des P-coups. Puisque N_1 et N_2 sont de même type, alors $\tau(m_1) = \tau(m_2)$. Or $\Gamma \vdash \alpha$ est un typage négativement non-dupliquant ce qui implique $m_1 = m_2 = m$. Nous pouvons en déduire l'égalité $p_1 = p_2 = p$. Par induction sur l'interprétation de $\mathbb{T}_{1,i}$ et de $\mathbb{T}_{2,i}$, nous prouvons que $N_1 \approx_N N_2$. □

Théorème 6.2.2.2. *Soit un terme N quasi-affine, sous forme β-normale et η-longue pour son typage principal $\Gamma \vdash \alpha$. Alors ce dernier est un typage négativement non-dupliquant.*

Démonstration. Étant données l'arène de typage $A = (M, \tau)$ associée au typage $\Gamma \vdash \gamma$, et la stratégie gagnante Σ sur A, telle que $N = [\![\Sigma]\!]_A$, nous savons que Σ est une stratégie recouvrante (d'après le théorème 3.3.5.1). Par l'absurde, supposons qu'il existe deux P-coups m_1 et m_2 de A qui vérifient $m_1 \neq m_2$ et $\tau(m_1) = \tau(m_2)$. D'après la définition d'une stratégie recouvrante, nous obtenons $m_1 R^*_\Sigma m_2$, ce qui implique l'existence de deux séquences $S_1.(m_1, j_1)$ et $S_2.(m_2, j_2)$ dans Σ telles que $N = C_i[[\![(n_i, k_i).(m_i, j_i)[\mathbb{T}_{i1}, \ldots, \mathbb{T}_{im}], V_i]\!]_A] = C_i[\lambda \overline{z_i}.N_i]$ pour $i \in \{1, 2\}$. Les sous-termes N_1 et N_2 ayant le même type atomique $\tau(m_2) = \tau(m_1)$, nous obtenons $N_1 = x_1 P_1 \ldots P_n$ et $N_2 = x_2 Q_1 \ldots Q_m$. Or, d'après le lemme 6.2.2.3, $x_1 \approx_N x_2$ n'est pas vérifiée, ce qui implique que $N_1 \approx_N N_2$ est également faux : nous obtenons donc une contradiction avec l'hypothèse que N est quasi-affine. □

6.3 Typages négativement séparants

Dans cette section, nous caractérisons une nouvelle famille de typages pour lesquels il existe un unique habitant : les typages *négativement séparants*.

Nous rappelons que, étant donnée une arène de typage $A = (M, \tau)$, la clôture transitive et réflexive de la relation d'accessibilité sur A est notée $\vdash^*$. Ainsi, étant donnés deux coups m_1 et m_2 de M, nous appellerons m_1 un *accesseur* de m_2 si $m_1 \vdash^* m_2$ est vérifié.

Définition 6.3.0.5. *Soient une arène de typage $A = (M, \tau)$ et deux coups m_1 et m_2 de M. Nous appelons un coup $n \in M$ un* accesseur commun *à m_1 et m_2 si $n \vdash^* m_1$ et $n \vdash m_2^*$.*

De plus, n sera appelé le plus petit accesseur commun *à m_1 et m_2 ssi pour tout coup n' tel que $n \vdash n'$, n' n'est pas un accesseur commun à m_1 et m_2.*

Nous remarquerons que, puisqu'une arène de typage a une structure arborescente, pour toute paire de coups (m_1, m_2) d'une arène, il existe un accesseur commun à ces deux coups.

Définition 6.3.0.6. *Soient un typage $\Gamma \vdash \alpha$ et $A = (M, \tau)$ l'arène associée. Alors, $\Gamma \vdash \alpha$ est dit* négativement séparant *si pour tous les P-coups m_1 et m_2 de M vérifiant $\tau(m_1) = \tau(m_2)$, le plus petit accesseur commun à m_1 et m_2 est un P-coup m différent de m_1 et m_2.*

Exemple 6.3.0.4. Nous représentons les arènes pour les typages $\vdash b^- \to a^- \to (a \to b^-) \to c$ (à gauche) et $(((s \to p^-) \to s^- \to q) \to (p^- \to q) \to r^-) \to (p \to q^-) \to r$ (à droite) en Figure 6.2. Le typage $\vdash b^- \to a^- \to (a \to b^-) \to c$ n'est pas négativement séparant, puisque le plus petit accesseur commun aux P-coups 1 et 3 est ϵ, alors que $\tau(1) = \tau(3) = b$. Par contre, le typage $(((s \to p^-) \to s^- \to q) \to (p^- \to q) \to r^-) \to (p \to q^-) \to r$ est négativement séparant. En effet, les P-coups 112 et 121 ont pour plus petit accesseur commun 1 et sont tous les deux de type p.

Remarque 6.3.0.3. Ce critère peut évidemment s'exprimer plus directement sur le typage et sans utiliser la notion d'arène, en introduisant la notion d'accesseur sur les occurrences de types atomiques d'un typage.

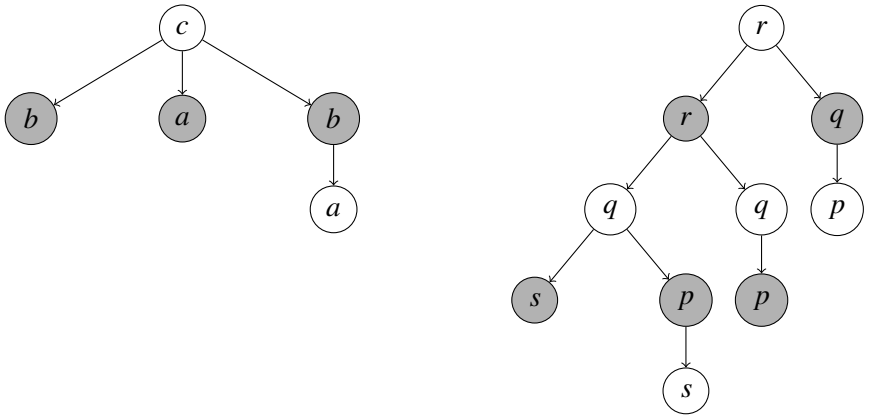


FIG. 6.2 – Typages négativement séparant : exemples

Lemme 6.3.0.4. *Soient $\Gamma \vdash \alpha$ un typage négativement séparant, $A = (M, \tau)$ l'arène associée, et Σ une stratégie gagnante sur A. Alors, pour tout P-coup m de M, et toute séquence S de Σ, il existe au plus une occurrence de m dans S.*

Démonstration. Nous procédons par l'absurde, et supposons qu'il existe une séquence $S \in \Sigma$ contenant deux occurrences d'un même P-coup m. La séquence S est alors de la forme $S_1.(m,i_1).S_2.(m,i_2)$. De plus, nous prenons la contrainte plus forte que m est le premier P-coup ayant plus d'une occurrence dans cette séquence, et que ces deux occurrences sont celles mises en évidence dans la séquence ; ainsi tout P-coup ayant une occurrence dans $S_1.(m,i_1).S_2$ a une unique occurrence dans celle-ci. Soit $S'_2 \sqsubseteq S_2.(m,i_2)$ une séquence de longueur paire. Par induction sur $|S'_2|/2$, nous montrons que la séquence $S_1.(m,i_1).S_2.(m,i_2).S'_2$ apparaît dans Σ, à indice d'occurrences de coups près.

- si $|S'_2| = 0$, le résultat est immédiat.
- supposons que la propriété est vraie pour un entier k tel que $0 \leq 2k < |S_2.(m,i_2)|$, et que $|S'_2| = 2k+2$, où $p \in \mathbb{N}$. Donc, S'_2 est de la forme $S''_2.(n_2,j_1).(m_2,j_2)$. D'après l'hypothèse d'induction, $S_1.(m,i_1).S_2.(m,i_2).S''_2$ appartient à Σ et il existe donc, par construction d'une stratégie gagnante, un P-coup m'_2 et un entier $j'_2 \in \mathbb{N}$ tel que $S_1.(m,i_1).S_2.(m,i_2).S''_2.(n_2,j_1).(m'_2,j'_2)$ appartient à Σ. Nous obtenons donc $\tau(m'_2) = \tau(m_2)$ et le coup m_2 est accessible dans la séquence $S_1.(m,i_1).S_2.(m,i_2).S''_2.(n_2,j_1)$. Supposons que $m_2 \neq m'_2$. Par définition, il existe un P-coup p qui est le plus petit accesseur commun à m_2 et m'_2. De plus, il existe deux O-coups o_2 et o'_2 tels que :
  – $p \vdash o_2$ et $p \vdash o'_2$
  – $o_2 \vdash^* m_2$ et $o'_2 \not\vdash^* m_2$
  – $o'_2 \vdash^* m'_2$ et $o_2 \not\vdash^* m'_2$

  Et o_2 et o'_2 ont chacun au moins une occurrence dans $S_1.(m,i_1).S_2.(m,i_2).S''_2.(n_2,j_1)$. Alors, par construction d'une stratégie, il faut que p ait deux occurrences dans cette dernière séquence. Dans ce cas, soit $p = m$, soit p apparaît dans S''_2. Dans les deux cas, puisque S''_2 est un préfixe de $S_2.(m,i_2)$, ceci implique que $o_2 = o'_2$, et nous obtenons une contradiction. D'où $m_2 = m'_2$ et donc, $S_1.(m,i_1).S_2.(m,i_2).S'_2$ apparaît dans Σ.

Finalement, ce résultat nous amène à l'existence d'une stratégie formée d'une infinité de séquences, ce qui est impossible. □

Nous remarqueront que ce lemme implique également que tout O-coup a au plus une occurrence

dans une séquence S sur une arène associée à un typage négativement séparant.

Théorème 6.3.0.3. *Étant donné un typage négativement séparant $\Gamma \vdash \alpha$, il existe au plus un habitant de ce typage.*

Démonstration. Supposons que ce typage est habité par un terme N : soit l'arène de typage $A = (M, \tau)$ associé à $\Gamma \vdash \alpha$ et la stratégie gagnante Σ associée au terme N (en supposant que ce dernier est sous forme normale longue pour ce typage). Le typage $\Gamma \vdash \alpha$ est uniquement habité ssi Σ est l'unique stratégie gagnante sur A. Supposons que ce ne soit pas le cas : il existe une stratégie $\Sigma' \neq \Sigma$ sur A. Étant donnée $S \in \Sigma'$, nous montrons par induction sur $|S|/2$ que $S \in \Sigma$:

- si $S = (\epsilon, 0).(m, 0)$. Alors m n'a pas de préfixe qui soit un P-coup (hormis m lui-même) ; par définition d'un typage négativement séparant, m est l'unique P-coup vérifiant $\tau(m) = \tau(\epsilon)$, donc $S \in \Sigma$.
- supposons que cette propriété est vraie pour toute séquence $S' \in \Sigma$ de longueur $2p$, pour un certain $p \in \mathbb{N}$. Soit S de longueur $2p + 2$, c'est-à-dire que S est de la forme $S'.(n, i).(m, j)$. D'après l'hypothèse d'induction, S' appartient à Σ. Par construction d'une stratégie, il existe $m' \in M^P$ et $j' \in \mathbb{N}$ tels que $S'.(n, i).(m', j') \in \Sigma$. Or $\tau(m) = \tau(m')$, donc il existe un P-coup p qui est le plus petit accesseur commun à m et m'. Par construction d'une stratégie, si m est différent de m', il existe deux occurrences de p dans $S'.(n, i).(m, j)$, ce qui est impossible d'après le lemme 6.3.0.4.

□

Nous remarquerons que l'idée générale de ces preuves est assez simple : si deux P-coups m_1 et m_2 ont un plus petit accesseur p qui est un P-coup, ces deux P-coups ne peuvent jamais être accessibles dans une séquence justifiée $S.(n, i)$, où n est un O-coup tel que $\tau(n) = \tau(m_1) = \tau(m_2)$, car, une fois une occurrence de p trouvée dans S, un seul des fils (et l'ensemble des sous-fils de ce dernier) est utilisable pour construire le reste de la séquence. Nous obtenons donc ainsi une forme de déterminisme dans la construction de toute séquence d'une arène associée à un typage négativement séparant.

Dans un second temps, il nous faut déterminer les λ-termes habitant ces typages, et vérifier que ces termes ont un typage principal qui soit négativement séparant. Dans ce qui suit, nous donnons une famille de λ-termes approchant ces conditions : les *λ-termes contextuellement déterministes*.

Définition 6.3.0.7. *Étant donné un terme N sous forme β-normale et η-longue pour un certain typage $\Gamma \vdash \alpha$. Le terme N est dit* contextuellement déterministes pour $\Gamma \vdash \alpha$ *si, pour toutes les occurrences de sous-termes $(C_1[], N_1)$ et $(C_2[], N_2)$ telles qu'il existe $\beta \in \mathcal{T}(\mathcal{A})$ vérifiant que $\Gamma_i \vdash \beta$ est le typage de $(C_i[], N_i)$ relatif à $\Gamma \vdash N : \alpha$ (et $i \in \{1, 2\}$), alors :*

$$\textit{Pour tout } \sigma : \mathcal{V} \mapsto \mathcal{V}, \Gamma_2 \vdash N_1 \cdot \sigma : \beta \iff N_1 \cdot \sigma = N_2$$

D'après cette définition, il est aisé de montrer que si un typage $\Gamma \vdash \alpha$ est uniquement habité par un terme N, alors ce terme N est contextuellement déterministe pour ce typage :

Théorème 6.3.0.4. *Soit un typage $\Gamma \vdash \alpha$ uniquement habité par un terme N sous forme β-normale et η-longue pour ce typage. Alors le terme N est contextuellement déterministe.*

Démonstration. Supposons que l'unique habitant N de ce typage n'est pas contextuellement déterministe. Il existe donc deux occurrences de sous-termes $(C_1[], N_1)$ et $(C_2[], N_2)$, un type β et un renommage de variables σ tels que :

- pour $i \in \{1, 2\}$, le typage de $(C_i[], N_i)$ relativement à $\Gamma \vdash N$ est $\Gamma_i \vdash \beta$.

- $\Gamma_2 \vdash N_1 \cdot \sigma : \beta$ et $N_2 \neq N_1 \cdot \sigma$.

Le terme $N' = C_2[N_1 \cdot \sigma]$ est donc un habitant de $\Gamma \vdash \alpha$. De plus, $N' \neq N$, ce qui contredit le fait que $\Gamma \vdash \alpha$ est uniquement habité. □

Ainsi, d'après cette preuve, nous pouvons exprimer le fait qu'un terme N soit contextuellement déterministe pour un typage $\Gamma \vdash \alpha$ par l'impossibilité qu'une occurrence de sous-terme de N soit remplacée par une autre occurrence d'un sous-terme de N tout en préservant le typage. Sur un arbre de dérivation, cette définition revient à exprimer l'impossibilité de remplacer un sous-arbre de dérivation par un autre sous-arbre de dérivation (sur les arènes de typages, par l'impossibilité de compléter une séquence S d'une stratégie gagnante Σ par un suffixe d'une autre séquence $S' \neq S$ de Σ).

Exemple 6.3.0.5. Considérons le terme $N = \lambda FG.F(\lambda Hx.G(Hx))(\lambda y.Gy)$ dont le typage principal est donné par la deuxième arène de la Figure 6.2. Alors ce terme est contextuellement déterministe : en effet, les deux occurrences de sous-termes

$$(\lambda FG.F(\lambda Hx.[])(\lambda y.Gy), G(Hx)) \text{ et } (\lambda FG.F(\lambda Hx.G(Hx))(\lambda y.[]), Gy)$$

ont un typage $\Gamma \vdash q$ relativement à ce typage, et il n'existe pas de substitution de variables σ vérifiant les conditions citées plus-haut ; sur cet exemple, il n'existe pas de substitution de variables permettant de remplacer une de ces occurrences par l'autre.

Bien que les termes contextuellement déterministes apparaissent comme des candidats potentiels à être les termes qui habitent exactement les typages négativement séparants, le contre-exemple suivant montre que tel n'est pas le cas :

Exemple 6.3.0.6. Le terme $N = F(\lambda z.G(Hz))(\lambda y.G(H'y))$ est un terme contextuellement déterministe. Son typage principal est :

$$G : a \to b, H : c \to a, H' : d \to a, F : (c \to b) \to (d \to b) \to d \vdash d$$

Or, ce typage n'est pas négativement séparant ; en effet, les deux occurrences négatives du type atomique a ont comme plus petit accesseur commun l'unique occurrence positive de d.

D'après ces quelques remarques, plusieurs problèmes restent donc ouverts ; en effet, nous n'avons pas pu déterminer quels sont les termes habitant exactement les typages négativement séparants. Bien que ces typages étendent les typages négativement non-dupliquants, il ne prennent pas en compte l'ensemble des termes qui sont les uniques habitants de leurs typages principaux. Il nous faut donc chercher une sous-famille de la famille des termes contextuellement déterministes pour les typages négativement séparants, en rajoutant certaines contraintes sur la forme syntaxique des termes.

Par ailleurs, nous n'avons pas pu donner une caractérisation syntaxique des typages principaux des termes contextuellement déterministes. Les relations d'accessibilité entre occurrences négatives de types atomiques ne semblent pas suffisantes dans ce cas. Le même type de relation entre occurrences positives de types atomiques semblent devoir être étudié.

De plus, afin d'exploiter la famille de termes contextuellement déterministes dans le cadre de l'analyse de grammaires catégorielles abstraites, il reste également à démontrer que, étant donné un ensemble $\mathcal{T}$ de termes contextuellement déterministes, l'ensemble $\mathcal{T}'$ de termes obtenus par clôture par composition de $\mathcal{T}$ vérifie que tout terme $N \in \mathcal{T}'$ est l'unique habitant de son typage principal. Cette propriété nous semble plus difficile à prouver. De plus, il n'est pas certain que les termes ainsi obtenus aient un intérêt dans la conception de reconnaisseurs pour des grammaires de langues naturelles.

Finalement, remarquons qu'une caractérisation complète des termes qui sont les uniques habitants de leur typage principal semble être une extension de la définition des termes contextuellement déterministes, qui n'est pas suffisante, comme le montre l'exemple suivant :

Exemple 6.3.0.7. Considérons le terme $N = F(\lambda H.G_1(F_1(\lambda z.G_2(Hz))))(\lambda F_2 x_2.G_1(F_2(G_2 x_2)))$.

Ce terme est contextuellement déterministe ; néanmoins, il existe un autre terme $N' \neq N$ habitant le typage principal de N : le terme $F(\lambda H.G_1(F_1(\lambda z.G_2(Hz))))(\lambda F_2 x_2.G_1(F_1(\lambda z.G_2 x_2)))$.

Ainsi, la définition d'un terme contextuellement déterministe impose l'impossibilité de remplacer des occurrences de sous-termes par d'autres occurrences de sous-termes afin de construire un nouveau terme N' habitant le même typage principal. Il est possible d'étendre cette idée aux *sous-contextes* d'un terme. Pour ce faire, il convient effectivement d'empêcher que des sous-contextes puissent remplacer d'autres contextes. Ainsi, il est possible que, étant donné un terme N sous forme normale et η-longue pour son typage principal $\Gamma \vdash \alpha$ tel que $N = C_1[C'_1[N_1] = C_2[C'_2[N_2]]$ où :

- les occurrences de sous-termes $(C_1[], C'_1[N_1])$ et $(C_2[], C'_2[N_2])$ ont le même type relativement à $\Gamma \vdash N : \alpha$.
- les occurrences de sous-termes $(C_1[C'_1[]], N_1)$ et $(C_2[C'_2[]], N_2)$ ont le même type relativement à $\Gamma \vdash N : \alpha$.
- N est contextuellement déterministe relativement à $\Gamma \vdash \alpha$,

il existe néanmoins un renommage de variables tel que le terme $N' = C_1[C'_2[N_1] \cdot \sigma]$ habite le typage $\Gamma \vdash \alpha$. Ainsi, il semble que vérifier qu'un terme est l'unique habitant de son typage principal revienne à vérifier l'impossibilité de remplacer des sous-contextes de termes par d'autres sous-contextes ; de tels termes peuvent être nommés *contextuellement déterministes d'ordre supérieur*. Le problème de l'unicité d'habitation pour les typages principaux semble donc être **PSPACE**-complet, c'est-à-dire équivalent en complexité à **UniTerme**.

Par ailleurs, [Broda and Damas, 2005] ont défini de manière algorithmique la famille des typages déterministes. Cette caractérisation revient à montrer que, pour toute séquence $S.(n,i)$, où S est une séquence d'une stratégie Σ sur l'arène de ce typage, il n'existe qu'un P-coup m tel que $\tau(m) = \tau(n)$ et m est un coup jouable. L'algorithme vérifiant qu'un typage est déterministe est un algorithme linéaire en temps pour la taille du typage, et qui revient simplement à montrer que le terme est habité : si, au cours de la dérivation, pour une occurrence positive de type atomique il existe aucune ou plus d'une occurrence négative permettant de poursuivre la démonstration, ce typage n'est pas déterministe.

D'après les résultats prouvés dans cette section, nous nous limiterons, dans le reste de cette étude, à présenter les résultats concernant l'analyse de grammaires catégorielles abstraites du second-ordre quasi-affines.

6.4 Conclusion

Au cours de ce chapitre, nous avons donné quelques résultats relatifs au problème de caractériser les typages uniquement habités (également appelé théorèmes de cohérence). Dans un premier temps, nous nous sommes intéressés à la complexité de la version la plus générale de ce problème, à savoir : "étant donné un typage $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, celui-ci est-il uniquement habité ?" Nous avons montré qu'il s'agit d'un problème **PSPACE**-complet. Ensuite, et grâce à l'utilisation de la sémantique des jeux, nous avons donné une nouvelle preuve du théorème de [Aoto, 1999], et nous avons caractérisé la famille des termes quasi-affines comme étant les termes habitant exactement les typages négativement non-dupliquants. Finalement, nous avons donné des pistes vers d'autres théorèmes de cohérence, en prouvant, en particulier, que les typages négativement séparants sont uniquement habités. Ce théorème donne lieu à la caractérisation syntaxique la plus forte sur les typages vérifiant la théorème de cohérence.

Nous nous appuyons sur les résultats relatifs aux théorèmes de cohérence pour les termes quasi-affines afin d'étudier, dans le Chapitre suivant, les techniques de reconnaissance grammaticale pour

les ACGs du second-ordre de termes quasi-affines.

Chapitre 7

Polynomialité de la reconnaissance pour les ACGs du second-ordre quasi-affines

Dans la partie précédente, nous nous sommes appuyés sur les jeux du Chapitre 3 afin de caractériser la famille de termes qui correspond exactement aux typages négativement non-dupliquants de [Aoto, 1999]. Cette famille est une extension de la famille des termes quasi-linéaires utilisées dans [Kanazawa, 2007], et ses individus vérifient la propriété de cohérence. Il est donc naturel de se poser la question de savoir si des reconnaisseurs d'ACGs du second-ordre quasi-affines peuvent être construits à la manière des reconnaisseurs de [Kanazawa, 2007], leur assurant ainsi une exécution en temps polynomial. Qui plus est, une telle construction nous permet de bénéficier des propriétés de réécriture de programmes Datalog décrites dans le Chapitre 5, et de l'analogie de ces dernières avec les algorithmes de reconnaissance de type CYK et de type Earley.

Remarque 7.0.0.4. Nous appelons variable effaçante dans un terme M, toute variable x telle que $M = C[\lambda x.N]$ et $x \notin FV(N)$.

Afin d'obtenir de tels résultats, nous avons vu que deux propriétés supplémentaires sont nécessaires : l'expansion du sujet et la clôture par composition. Alors que nous verrons que la seconde est vérifiée pour les termes quasi-affines, tel n'est pas le cas de la première, principalement à cause de β-réduction effaçante :

Exemple 7.0.0.8. Pour les termes quasi-affines $M = \lambda x.gy$ et $M' = \lambda x.(\lambda z.gy)(xe)$, il apparaît directement que $M' \rightarrow_\beta^* M$. Par ailleurs, le typage principal de M est $y : a, g : a \rightarrow b \vdash c \rightarrow b$. Or, le typage principal de M' est $y : a, g : a \rightarrow b, e : c \vdash (e \rightarrow c) \rightarrow b$. Tout typage de M' doit donc contenir une assignation de types pour la variable e, ce qui n'est pas le cas pour les typages de M.

Sur cet exemple, nous voyons que M' se β-réduit en M grâce à une β-réduction effaçante. Ceci a pour effet de faire disparaître toute occurrence de la variable e dans M, et une occurrence de la variable x dans ce dernier terme. La suppression d'occurrences de variables (ou de constantes) modifie les contraintes de typage de ces variables (ou constantes). Sur notre exemple, e peut être assigné n'importe quel type dans un typage de M, et il en est de même pour le type de x. Ces deux variables doivent par contre, être assignés des types d'une certaine forme dans un typage de M'.

Nous montrerons qu'il est néanmoins possible de construire des reconnaisseurs Datalog pour les ACGs du second-ordre quasi-affines d'après les idées de Kanazawa, en donnant une version plus faible

du théorème d'expansion du sujet pour ces λ-termes, dans le système de typage listé, informellement introduit au chapitre précédent.

7.1 Système de typage listé

Le *système de typage listé* permet de typer des termes typés à la Church, et en particulier des termes appartenant à une signature d'ordre supérieur. Nous y définirons les typages *potentiellement négativement non-dupliquants* (ou *typages PN*) comme étant une généralisation des typages négativement non-dupliquants, puis, étant donné un terme quasi-affine, nous lui associerons un ensemble de typages appelés typages potentiellement négativement non-dupliquants moins généraux de ce terme. Nous verrons comment les propriétés de ces derniers sont proches de celles du typage principal d'un terme quasi-linéaire dans le système de typage simple, permettant ainsi de construire des reconnaisseurs Datalog à la Kanazawa pour les ACGs non-contextuelles quasi-affines.

7.1.1 Définitions et propriétés

Nous introduisons à présent le système de typage listé :

Définition 7.1.1.1. *Étant donnés A et B deux ensembles énumérables de types atomiques et un type $\alpha \in \mathcal{T}(A)$, l'ensemble $\mathcal{U}_\alpha(B)$ des* types uniformes à α construits sur *B est défini par induction sur α :*

- *si $\alpha = a \in A$ alors $\mathcal{U}_\alpha(B) = B$.*
- *si $\alpha = \alpha_1 \to \alpha_2$ alors $\mathcal{U}_\alpha(B) = \{\beta_1 \to \beta_2 \mid \beta_1 \in \mathcal{U}_{\alpha_1}(B)$ et $\beta_2 \in \mathcal{U}_{\alpha_2}(B)\}$*

L'uniformité de types permet donc d'imposer des contraintes structurelles à ces derniers, puisqu'ils sont égaux modulo renommages :

Lemme 7.1.1.1. *Étant donné un type $\alpha \in \mathcal{T}(A)$, deux types γ_1 et γ_2 appartenant à $\mathcal{U}_\alpha(B)$ sont toujours unifiables et leur unificateur le plus général est un renommage.*

Démonstration. Par induction sur α : si $\alpha = a \in A$, alors $\gamma_1 = c_1$ et $\gamma_2 = c_2$ appartiennent à B par définition. L'unificateur le plus général de γ_1 et γ_2 est donc σ tel que $c_1 \cdot \sigma = c_2 \cdot \sigma$.

Sinon, $\alpha = \alpha_1 \to \alpha_1$, et par construction, $\gamma_1 = \beta_1 \to \beta_2$ et $\gamma_2 = \delta_1 \to \delta_2$, tels que $\beta_1, \delta_1 \in \mathcal{U}_{\alpha_1}(B)$ et $\beta_2, \delta_2 \in \mathcal{U}_{\alpha_2}(B)$. D'après l'hypothèse d'induction, pour $i \in \{1,2\}$, β_i et δ_i sont unifiables et leur unificateur le plus général est un renommage σ_i contenant des égalités de la forme $a \cdot \sigma_i = b \cdot \sigma_i$, où $a, b \in B$. À partir de σ_1 et σ_2, nous construisons donc σ de telle sorte que $a \cdot \sigma = b \cdot \sigma$ ssi il existe des types atomiques $a_1, \ldots, a_n$ et une fonction $f : \{1, \ldots, n\} \mapsto \{1,2\}$ tels que

$$a \cdot \sigma_{f(1)} = a_1 \cdot \sigma_{f(1)}, a_1 \cdot \sigma_{f(2)} = a_2 \cdot \sigma_{f(2)} \ldots, a_n \cdot \sigma_{f(n-1)} = b \cdot \sigma_{f(n)}$$

Alors σ est l'unificateur le plus général de γ_1 et γ_2, et est bien un renommage. □

Définition 7.1.1.2. *Étant donnés deux ensembles énumérables de types A et B, et un type $\alpha \in \mathcal{T}(A)$, nous définissons l'ensemble $\mathcal{L}_\alpha(B)$ des* types listés sur B et uniformes à α *comme le plus petit ensemble tel que :*

- *$\mathcal{U}_\alpha(B) \subseteq \mathcal{L}_\alpha(B)$.*
- *si l_1 et l_2 appartiennent à $\mathcal{L}_\alpha(B)$, alors $l_1 \cap l_2$ appartient à $\mathcal{L}_\alpha(B)$.*

L'ensemble $\mathcal{L}(B)$ des types listés sur B est défini comme étant l'ensemble $\{\mathcal{L}_\alpha(B) \mid \alpha \in \mathcal{T}(A)\}$

Notation 7.1.1. Les types listés seront notés par des lettres grecques minuscules surlignées $\overline{\alpha}, \overline{\alpha_1}, \ldots$

$$\frac{\gamma \in \mathbf{\Delta}(\mathbf{c}^{\alpha})}{\langle \Gamma; \mathbf{\Delta} \rangle \vdash \mathbf{c}^{\alpha} : \gamma}\ \text{AxC} \qquad \frac{\Gamma(x^{\alpha}) = \gamma}{\langle \Gamma; \mathbf{\Delta} \rangle \vdash x^{\alpha} : \gamma}\ \text{AxV}$$

$$\frac{\langle \Gamma, x^{\alpha} : \gamma; \mathbf{\Delta} \rangle \vdash M : \delta}{\langle \Gamma; \mathbf{\Delta} \rangle \vdash \lambda x^{\alpha}.M : \gamma \to \delta}\ \text{Abs}$$

$$\frac{\langle \Gamma; \mathbf{\Delta} \rangle \vdash M_1 : \gamma \to \delta \quad \langle \Gamma; \mathbf{\Delta} \rangle \vdash M_2 : \gamma}{\langle \Gamma; \mathbf{\Delta} \rangle \vdash M_1 M_2 : \delta}\ \text{App}$$

FIG. 7.1 – Système de dérivation pour les types listés

Nous allons, à présent, assigner ces types listés aux constantes d'un terme appartenant à une signature d'ordre supérieur. De manière intuitive, assigner un type listé $\alpha_1 \cap \ldots \cap \alpha_n$ à une constante, revient à lui assigner les types simples $\alpha_1, \ldots, \alpha_n$, à la manière des types intersection (pour plus de détails, se référer à [Dezani-Ciancaglini et al., 1998]). Il convient cependant de noter que nous imposons aux types simples composant un type listé d'être uniformes deux à deux, et que nous n'utilisons pas le type universel ω, à l'image du système de [Coppo and Dezani-Ciancaglini, 1980]. Qui plus est, nous contraignons le connecteur d'intersection $\cap$ à n'être utilisé qu'au niveau le plus externe, un type listé appartenant à $\mathcal{L}_{\alpha}(B)$ pouvant ainsi être vu comme un sous-ensemble non-vide de $\mathcal{U}_{\alpha}(B)$. En effet, à l'image de ses propriétés dans les types intersection, le connecteur $\cap$ est commutatif (*i.e.* $\overline{\alpha}_1 \cap \overline{\alpha}_2 = \overline{\alpha}_2 \cap \overline{\alpha}_1$), associatif (*i.e.* $(\overline{\alpha}_1 \cap \overline{\alpha}_2) \cap \overline{\alpha}_3 = \overline{\alpha}_1 \cap (\overline{\alpha}_2 \cap \overline{\alpha}_3) = \overline{\alpha}_1 \cap \overline{\alpha}_2 \cap \overline{\alpha}_3$) et idempotent (*i.e.* $\overline{\alpha} \cap \overline{\alpha} = \overline{\alpha}$).

Notation 7.1.2. En conséquence, étant donnés un type simple α et deux types listés $\overline{\gamma_1}$ et $\overline{\gamma_2}$, nous utiliserons les notations intuitives suivantes :

- $\alpha \in \overline{\gamma_1}$ si $\overline{\gamma_1} = \alpha_1 \cap \ldots \cap \alpha \cap \ldots \alpha_n$.
- $\overline{\alpha_1} \subseteq \overline{\alpha_2}$ si pour tout $\alpha \in \overline{\alpha_1}$, $\alpha \in \overline{\alpha_2}$.
- la taille $|\overline{\alpha_1}|$ de $\overline{\alpha_1}$ donnée par : $|\overline{\alpha_1}| = 1$ si $\overline{\alpha}_1 = \alpha_1$ est un type simple, $|\overline{\gamma_1} \cap \overline{\gamma_2}| = |\overline{\gamma_1}| + |\overline{\gamma_2}|$ sinon.

Par la suite, nous imposerons à un type listé d'être fini au sens ensembliste.

Définition 7.1.1.3. *Étant donnés un ensemble énumérable de types atomiques A, un* environnement de typage listé sur *A est un couple* $\langle \mathbf{\Delta}; \Gamma \rangle$ *tel que :*

- Γ *est une fonction de l'ensemble énumérable de variables* $\mathcal{V}$ *(typées à la Church) sur l'ensemble* $\mathcal{T}(A)$ *des types simples sur A, tel que* $\Gamma(x^{\alpha}) \in \mathcal{U}_{\alpha}(A)$.
- $\mathbf{\Delta}$ *est une fonction de l'ensemble énumérable de constantes C (typées à la Church) sur l'ensemble* $\mathcal{L}(A)$ *des types listés sur A, tel que* $\mathbf{\Delta}(\mathbf{c}^{\alpha}) \in \mathcal{L}_{\alpha}(A)$.

Les conventions d'écriture pour les environnements de typage listé sont les mêmes que pour les environnements de typage simple : nous écrirons $\Gamma = x_1^{\beta_1} : \alpha_1, \ldots, x_n^{\beta_n} : \alpha_n$ si $\mathrm{Dom}(\Gamma) = \{x_1^{\beta_1}, \ldots, x_n^{\beta_n}\}$ et si pour tout $i \in \{1, \ldots, n\}$, $\Gamma(x_i^{\beta_i}) = \alpha_i$. De même, nous écrirons $\mathbf{\Delta} = \mathbf{c_1}^{\delta_1} : \overline{\gamma_1}, \ldots, \mathbf{c_m}^{\delta_m} : \overline{\gamma_m}$ si $\mathrm{Dom}(\mathbf{\Delta}) = \{\mathbf{c_1}^{\delta_1}, \ldots, \mathbf{c_m}^{\delta_m}\}$ et si pour tout $i \in \{1, \ldots, m\}$, $\mathbf{\Delta}(\mathbf{c_i}^{\delta_i}) = \overline{\gamma_i}$. Si $\mathrm{Dom}(\Gamma) = \emptyset$ (*resp.* $\mathrm{Dom}(\mathbf{\Delta}) = \emptyset$), alors nous écrirons $\Gamma = \_$ (*resp.* $\mathbf{\Delta} = \_$).

Un typage listé $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ sur un ensemble énumérable de types A est donné par un environnement de typage listé $\langle \mathbf{\Delta}; \Gamma \rangle$ sur A et un type simple $\alpha \in \mathcal{T}(A)$. Nous dirons qu'*un terme M habite un typage* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ s'il existe une dérivation de $\langle \mathbf{\Delta}; \Gamma \rangle \vdash M : \alpha$ dans le système de dérivation de la Figure 7.1.

Nous remarquerons qu'un terme M peut être typé dans le système de typage listé si et seulement s'il est simplement typable.

Définition 7.1.1.4. *Étant donnés un terme M et deux typages listés $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1$ et $\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$ de M, nous dirons que $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1$* est plus général que *$\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$ s'il existe un renommage σ tel que :*

- $\alpha_2 = \alpha_1 \cdot \sigma$.
- *pour toute variable x^α de* $\mathrm{Dom}(\Gamma_1)$*, x^α appartient à* $\mathrm{Dom}(\Gamma_2)$ *et* $\Gamma_2(x^\alpha) = \Gamma_1(x^\alpha) \cdot \sigma$.
- *pour toute constante $\mathbf{c}^\alpha$ de* $\mathrm{Dom}(\mathbf{\Delta_1})$*, $\mathbf{c}$ appartient à* $\mathrm{Dom}(\mathbf{\Delta_2})$ *et* $\mathbf{\Delta_1}(\mathbf{c}^\alpha) \cdot \sigma \subseteq \mathbf{\Delta_2}(\mathbf{c}^\alpha)$.

La relation de généralité entre typages préserve certaines propriétés évidentes ; ainsi, si un terme M habite un typage $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1$ et que ce dernier est plus général que $\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$, alors M habite $\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$. De plus, pour un terme M, il existe un typage principal (ou le plus général). Mais ce typage n'est pas unique pour un terme donné, même à renommage près, car il est possible d'assigner un type listé arbitrairement grand aux constantes.

Exemple 7.1.1.1. Étant donné le terme $\mathbf{c}^o$, les deux typages $\langle \mathbf{c}^o : a; \_ \rangle \vdash a$ et $\langle \mathbf{c}^o : a \cap b; \_ \rangle \vdash a$ sont chacun plus général que l'autre ; en effet $a \subseteq a \cap b$ et pour $\sigma = [b \mapsto a]$, $\langle \mathbf{c}^o : a \cap b; \_ \rangle \cdot \sigma \vdash a \cdot \sigma = \langle \mathbf{c}^o : a; \_ \rangle \vdash a$

Néanmoins, si nous considérons un ordre sur les typages les plus généraux, donné par la taille des types assignés aux constantes, nous pouvons créer une partition de l'ensemble des typages les plus généraux, telles que deux typages d'une même partie sont égaux à renommage près. Les typages de la partie pour laquelle cette mesure est la plus petite seront appelés les *typages listés les plus généraux* ou *typages listés principaux* de M.

Ce typage listé principal d'un terme M peut être obtenu en remplaçant, premièrement, toute occurrence de constante dans M par une constante fraîche, obtenant ainsi un terme typé à la Church que nous noterons $c - lin(M)$; il nous faut ensuite calculer le typage principal $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ de $c - lin(M)$ dans le système de typage simple ; puis calculer $\langle \mathbf{\Delta'}; \Gamma \rangle \vdash \alpha$ de telle sorte que $\alpha \in \Delta'(\mathbf{c})$ si et seulement si $\alpha = \mathbf{\Delta}(\mathbf{c'})$ et que $\mathbf{c'}$ remplace une occurrence de $\mathbf{c}$ dans M.

Exemple 7.1.1.2. Soit le terme $M = \lambda P^{o \to o}.\mathbf{f}^{o \to o}((\lambda x^o.\mathbf{f}^{o \to o}\mathbf{c}^o)(P^{o \to o}\mathbf{c}^o))$. Nous calculons $c - lin(M) = \lambda P^{o \to o}.\mathbf{f_1}^{o \to o}((\lambda x^o.\mathbf{f_2}^{o \to o}\mathbf{c_1}^o)(P^{o \to o}\mathbf{c_2}^o))$ dont le typage principal est $\langle \mathbf{c_1}^o : a_1, \mathbf{c_2}^o : a_2, \mathbf{f_1}^{o \to o} : a_1 \to b_1, \mathbf{f_2}^{o \to o} : b_1 \to b_2; \_ \rangle \vdash (a_2 \to c) \to b_2$. Le typage principal listé de M est donc :

$$\langle \mathbf{\Delta_1}; \_ \rangle \vdash \alpha_1 = \langle \mathbf{c}^o : a_1 \cap a_2, \mathbf{f}^{o \to o} : a_1 \to b_1 \cap b_1 \to b_2; \_ \rangle \vdash (a_2 \to c) \to b_2$$

Les propriétés suivantes se déduisent naturellement des propriétés de dérivation des termes dans le système de typage simple :

Propriété 7.1.1.1. *Étant donné un terme M, tout typage listé de M est moins général que son typage principal listé.*

Propriété 7.1.1.2 (Réduction du sujet)**.** *Étant donnés deux termes M_1 et M_2 tels que $M_1 \to_\beta^* M_2$, tout typage listé de M_1 est un typage listé de M_2.*

Notation 7.1.3. Par la suite, nous écrirons $\mathbf{c}$ et x en lieu et place de $\mathbf{c}^\alpha$ et x^α lorsque le type associée aux constantes et variables n'est qu'informatif.

7.1.2 Arènes et jeux pour les typages listés

Afin d'étudier les propriétés de typages de termes quasi-affines dans le système de typage listé, nous étendons la notion d'arènes de typages simples, définie dans le Chapitre 3, au système de typage listé.

Définition 7.1.2.1. *Étant donnés un ensemble fini d'arène $\mathcal{A} = \{A_1, \ldots, A_n\}$ (où toute arène $A_i \in \mathcal{A}$ s'écrit $A_i = (M_i, \tau_i)$) et une bijection $f : \mathcal{A} \mapsto \{1, \ldots, n\}$, nous définissons l'opération $\times_f$ entre arènes de $\mathcal{A}$ par $A_{i_1} \times_f \ldots \times_f A_{i_k} = (M, \tau)$, où :*

- $M \subseteq \biguplus_{i \in \{i_1, \ldots, i_k\}} M_i \times \mathbb{N}$, *où + est l'opération d'union disjointe, et $(m, p) \in M$ si et seulement s'il existe $j \in \{i_1, \ldots, i_k\}$ tel que $m \in M_j$ et $f(A_j) = p$.*
- *pour tout coup $(m, f(A_i)) \in M$, $\tau(m) = \tau_i(m)$.*

Notation 7.1.4. Par la suite, nous omettrons la bijection f dans la notation du produit d'arène par souci de lisibilité. Ainsi, nous écrirons $A_1 \times \ldots \times A_n$ pour l'arène $A_1 \times_f \ldots \times_f A_n$ où $f(A_i) = i$ pour tout $i \in \{1, \ldots, n\}$.

Définition 7.1.2.2. *Soit un type listé $\overline{\alpha} = \alpha_1 \cap \ldots \cap \alpha_n$. Pour tout $i \in \{1, \ldots, n\}$, nous notons par $A_{\alpha_i} = (M_i, \tau_i)$ l'arène de type associée à α_i. L'arène $A_{\overline{\alpha}}$ associée à $\overline{\alpha}$ est définie par $A_{\overline{\alpha}} = A_1 \times \ldots \times A_n$.*

L'arène associée à un type listé est donc simplement définie comme étant l'ensemble des arènes associées à chacun des types simples composant ce type listé.

De manière identique à la définition d'une arène associée à une assignation de type simple, nous notons par $A_{\mathbf{c}:\overline{\alpha}}$ l'arène associée à l'assignation d'un type listé $\overline{\alpha}$ à une constante $\mathbf{c}$. Les coups d'une telle arène seront notés $m = (s_{\mathbf{c}}, i)$, où $s \in \mathbb{N}^*$ et $i \in |\overline{\alpha}|$. Nous préservons la notation $m \cdot j = (s_{\mathbf{c}} \cdot j, i)$, où $m = (s_{\mathbf{c}}, i)$ et $j \in \mathbb{N}$, pour le coup $(s'_{\mathbf{c}}, i)$, où $s' = s \cdot j$.

Définition 7.1.2.3. *Soit un typage listé $\langle \mathbf{c_1} : \overline{\gamma_1}, \ldots, \mathbf{c_m} : \overline{\gamma_m}; x_1 : \alpha_1, \ldots, x_n : \alpha_n \rangle \vdash \alpha$; l'arène $A = (M, \tau)$ associée à ce typage est définie comme :*

- $M = M_\alpha \cup \bigcup_{i \in \{1, \ldots, n\}} M_{x:\alpha_i} \cup \bigcup_{j \in \{1, \ldots, m\}} M_{\mathbf{c_j}:\overline{\gamma_j}}$
- $\tau(m) = \begin{cases} \tau_\alpha(m) & \text{pour } m \in M_\alpha \\ \tau_{x_i:\alpha_i}(m) & \text{pour } m \in M_{x_i:\alpha_i} \text{ et } i \in \{1, \ldots, n\} \\ \tau_{\mathbf{c_j}:\overline{\gamma_j}(m)} & \text{pour } m \in M_{\mathbf{c}_j:\overline{\gamma_j}} \text{ et } j \in \{1, \ldots, m\} \end{cases}$

De plus, la relation d'accessibilité est étendue de manière naturelle, de telle sorte que $m_1 \vdash m_2$ soit vérifiée ssi

- $m_2 = m_1 \cdot j$ pour un certain $j \in \mathbb{N}$, ou bien
- $m_1 = \epsilon \in M_\alpha$ et $m_2 = \epsilon_{x_i}$ pour un certain $i \in \{1, \ldots, n\}$, ou $m_2 = (\epsilon_{\mathbf{c_j}}, k)$ pour un certain $j \in \{1, \ldots, m\}$ et un certain $k \in \mathbb{N}$.

On remarquera que l'arène associée à un typage listé est en fait identique à une arène associée à un typage simple, si ce n'est que le nommage des coups est différent. Les notions de stratégie et de stratégie gagnante s'étendent donc de manière naturelle aux arènes de typages listés. De même, l'interprétation $[\![\Sigma]\!]_A$ d'une stratégie gagnante Σ sur une arène A est également étendue de manière intuitive.

Notation 7.1.5. Pour une arène de typage listé A, et une stratégie gagnante Σ sur A, nous noterons pas $[\![\Sigma]\!]_A$ le terme résultant de l'interprétation de Σ dans A.

Les résultat suivants se déduisent de manière évidente des mêmes théorèmes sur les arènes de typage simple :

Théorème 7.1.2.1. *Étant donnée une arène A associée à un typage listé $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$, un terme N est un habitant de ce typage ssi il existe une stratégie gagnante Σ sur A telle que $N =_{\beta\eta} [\![\Sigma]\!]_A$ et que $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ est uniforme au typage de N.*

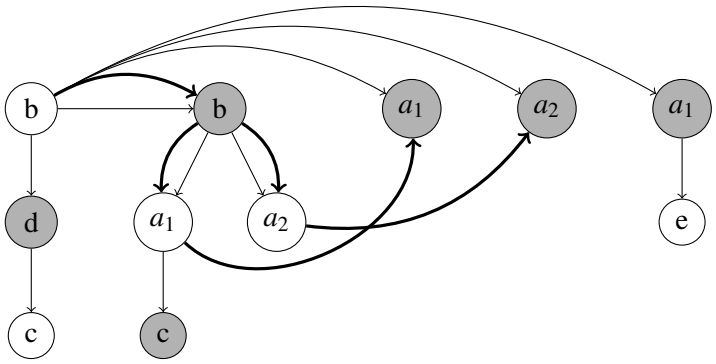


FIG. 7.2 – Exemple de stratégie de typage

Théorème 7.1.2.2. *Si un typage listé négativement non-dupliquant* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ *est habité par un terme N, alors N est quasi-affine et tout terme M habitant* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ *vérifie* $M =_{\beta\eta} N$.

En Figure 7.2, nous donnons un exemple de stratégie sur l'arène du typage $\langle \mathbf{c} : a_1 \cap a_2, \mathbf{c}' : e \to a_1; f : (c \to a_1) \to a_2 \to b \rangle \vdash (d \to c) \to b$.

Étant donné un terme N, il est également possible de caractériser ses typages principaux listés en modifiant quelque peu la notion de stratégie recouvrante définie sur les arènes associées aux typages simples. Pour ce faire, étant données une arène de typage listé $A = (M, \tau)$ et une stratégie Σ sur A, nous définissons la notion de Σ-*succession* $\rhd_\Sigma \subseteq M^P \times M^O$ sur une stratégie Σ, comme complémentaire de la notion de Σ-précédence : $n \rhd_\Sigma m$ est vérifiée si et seulement si $m \lhd_\Sigma n$. De manière identique à la Σ-précédence, et étant donné un O-coup n de M, nous définissons $\rhd_\Sigma^n = \{m \in M^P \mid n \rhd_\Sigma m\}$. Enfin, la relation $\between_\Sigma$ sur $M^O \times M^O$ est définie de sorte que $n_1 \between_\Sigma n_2$ si et seulement si $\rhd_\Sigma^{n_1} \cap \rhd_\Sigma^{n_2} = \emptyset$.

Définition 7.1.2.4. *Soit A une arène de typage listé et* Σ *une stratégie gagnante sur A.* Σ *est dite recouvrante si :*

- *Pour tout P-coup m de la forme* ϵ_x *ou* $(\epsilon_{\mathbf{c}}, i)$ $(i \in \mathbb{N})$*, il existe une séquence* $S.(m, j)$ *dans* Σ *(**ϵ-complétude**).*
- *Pour toute paire de P-coups* (m_1, m_2) *de* $M^P \times M^P$ *,* $\tau(m_1) = \tau(m_2)$ *ssi* $m_1 R_\Sigma^* m_2$*. (**plus faible contrainte de typage sur P-coups**)*
- *Pour toute paire de O-coups* (n_1, n_2) *de* $M^O \times M^O$ *,* $\tau(n_1) = \tau(n_2)$ *ssi* $n_1 \between_\Sigma^* n_2$*. (**plus faible contrainte de typage sur O-coups**)*

Théorème 7.1.2.3. *Soit une arène A. Le typage listé* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ *associé à A est le typage principal listé du terme β-réduit N ssi il existe une stratégie recouvrante* Σ *sur A telle que* $[\![\Sigma]\!]_A =_\eta N$ *et que* $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ *est uniforme au typage de N.*

Démonstration. La preuve est une simplification de la preuve du théorème 3.3.5.1, puisque nous ne considérons plus que des renommages dans le cas des typages listés, et non des substitutions. Les parties de la preuve concernant la condition de plus faible contrainte de typage sur les P-coups sont alors identiques à celles concernant la même condition sur les O-coups. □

Exemple 7.1.2.1. La stratégie Σ de la Figure 7.2 est recouvrante sur l'arène associée au typage $\langle \mathbf{c} : a_1 \cap a_2, \mathbf{c}' : e \to a_3; f : (c \to a_1) \to a_2 \to b \rangle \vdash (d \to i) \to b$

Il convient de remarquer que la condition de O-complétude n'est plus vérifiée dans les typages principaux listés dans le cas des types simples, en raison de l'uniformité des types associés aux variables et aux constantes. Cette condition est donc remplacée par une condition de plus faible contrainte de typage sur les O-coups.

Remarque 7.1.2.1. Considérons, à nouveau une arène A comme un arbre, et Σ un ensemble de chemins sur les noeuds de cet arbre. Déterminer le typage principal du terme $[\![\Sigma]\!]_A$ revient, à nouveau, à déterminer le nombre maximal d'étiquettes sur A tel que, pour tout sous-chemin nm de chemin de Σ, où $n \in M^O$ et $m \in M^P$, n et m ont la même étiquette. Le nombre de noeuds du graphe (*i.e.* le nombre de types atomiques constituant le typage) est cette fois fixe, puisque nous construisons des typages uniformes deux à deux.

Nous pouvons en déduire le résultat suivant :

Théorème 7.1.2.4. *Étant donné un terme quasi-linéaire M, son typage principal listé est négativement non-dupliquant.*

Nous avons vu que les types listés ont une structure rigide, puisqu'un tel type est uniforme à un autre. Nous définissons à présent la notion de *potentialité de coups* liée à cette propriété.

Définition 7.1.2.5. *Soit une arène $A = (M, \tau)$ associée à un typage listé $\langle \mathbf{\Lambda}; \Gamma \rangle \vdash \alpha$. Un coup m est dit* potentiel *dans A s'il existe une stratégie Σ sur A et une séquence $S_1.(m,i).S_2 \in \Sigma$.*
Sinon le coup m est dit injouable.

Il convient de remarquer que la définition de coups potentiels fait intervenir la notion de stratégie, et non celle de stratégie gagnante. De plus, cette notion est tout à fait indépendante du fait que l'arène A considérée soit une arène de typage simple ou de typage listé.

Notation 7.1.6. Étant donné une arène A associée à un typage listé $\langle \mathbf{\Lambda}; \Gamma \rangle \vdash \alpha$, nous noterons par $\mathrm{Pot}(A)$, l'ensemble des coups potentiels sur A. L'ensemble des coups injouables de A sera noté $\mathrm{Inj}(A)$.

Exemple 7.1.2.2. Sur l'arène de la Figure 7.2, l'ensemble des coups injouables est $\{1, 11, 11_f\}$.

Sachant que les coups d'une arène correspondent à des occurrences de types atomiques dans le typage associé, et qu'une stratégie correspond à une dérivation (non-valide dans le cas d'une stratégie non-gagnante) dans ce typage, un coup injouable correspond à une occurrence de type atomique qui ne sera en conclusion d'aucun séquent dans toutes les dérivations (partielles) de ce typage. Il existe une manière constructive de calculer les coups potentiels, décrite à travers la formule suivante, qui développe l'ensemble des stratégies sur une arène.

Notation 7.1.7. Étant donnés une arène $A = (M, \tau)$ et un coup $m \in M$, l'ensemble des coups en relation d'accessibilité directe avec m est noté $\vdash_m = \{n \in M \mid m \vdash n\}$. De plus, pour Q un sous-ensemble de M, nous écrivons $\mathrm{eq}(Q, m)$ l'ensemble $\{n \in Q \mid \tau(m) = \tau(n)\}$.

L'ensemble des coups potentiels de A, l'arène associée à un typage $\langle \mathbf{\Lambda}; \Gamma \rangle \vdash \alpha$, est défini par :

$$\mathrm{Pot}(A) = \mathtt{Pot}(\vdash_\epsilon, \epsilon)$$

où $\mathtt{Pot}(Q, m)$ est le plus petit ensemble de coups vérifiant :

$$\mathtt{Pot}(Q, m) = \{m\} \cup \bigcup_{n \in \mathrm{eq}(Q,m)} \left(\{n\} \cup \bigcup_{p \in \vdash_n} \mathtt{Pot}(Q \cup \vdash_p, p) \right)$$

Notation 7.1.8. Étant donnés une arène $A = (M, \tau)$ et un ensemble $E \subseteq M$, nous noterons par E^P (*resp.* E^O), l'ensemble des P-coups (*resp.* O-coups) de E. De plus, étant donné un type atomique a, nous noterons par E_a l'ensemble $\{m \in E \mid \tau(m) = a\}$.

Une remarque importante dans le cas d'un typage listé négativement non-dupliquant habité par un terme N sous forme normale, est que l'ensemble des coups potentiels de l'arène A associée à un tel typage correspond exactement à l'ensemble des coups apparaissant sur les séquences de Σ telle que $[\![\Sigma]\!]_A =_\eta N$. En effet, toute stratégie Σ' sur A vérifie $\Sigma' \subseteq \Sigma$. Par ailleurs, de par la caractérisation du typage principal par une stratégie recouvrante, un coup m injouable de l'arène $A = (M, \tau)$ associée au typage principal listé d'un terme quasi-affine vérifie que, pour tout coup $n \neq m \in M$, $\tau(m) \neq \tau(n)$.

7.2 Typages potentiellement négativement non-dupliquants

Ayant introduits le système de typage listé et une notion de jeux pour celui-ci, nous cherchons à présent à caractériser une famille de typages des termes quasi-affines pour lesquels ces derniers vérifient les propriétés de cohérence, de clôture par composition, et d'expansion du sujet (ou d'une théorème analogue) dans ce système. Ces typages sont en fait une généralisation des typages négativement non-dupliquants. Dans un premier temps, nous prouvons la clôture par composition des termes quasi-affines en nous servant donc uniquement du fait que leurs typages principaux listés sont négativement non-dupliquants.

7.2.1 Composition de termes quasi-affines et préservation de cohérence

Dans un premier temps, nous montrons que les termes quasi-affines sont clos par composition. Pour ce faire, nous nous intéressons, non pas aux λ-termes et à leurs syntaxes, mais à leurs typages principaux listés. Ayant caractérisé les termes quasi-affines comme les termes dont le typage principal listé est négativement non-dupliquant (Théorème 7.1.2.4), nous démontrons que, sous certaines conditions, la composition de deux typages négativement non-dupliquants génère un typage lui-même négativement non-dupliquant. Afin de réaliser cette démonstration, nous nous aidons d'un algorithme d'unification.

En entrée du problème d'unification, nous considérons un type simple γ, deux ensembles disjoints et finis de types atomiques $\mathcal{A}_1$ et $\mathcal{A}_2$, et deux types simples $\alpha_1 \in \mathcal{U}_\gamma(\mathcal{A}_1)$ et $\alpha_2 \in \mathcal{U}_\gamma(\mathcal{A}_2)$, et leurs arènes respectives $A_1 = (M_1, \tau_1)$ et $A_2 = (M_2, \tau_2)$. Puisque les types α_1 et α_2 sont uniformes, un coup $m_1 = s$ appartient à M_1 ssi il existe un coup $m_2 = s$ dans M_2, où $s \in \mathbb{N}^*$. Nous écrirons donc $M = M_1 = M_2$, $A_1 = (M, \tau_1)$ et $A_2 = (M, \tau_2)$. Nous construisons ensuite deux ensembles :

- un ensemble fini d'équations initiales

$$R_{\alpha_1}^{\alpha_2} = \{\tau_1(m) = \tau_2(m) \mid m \in M\}$$

- un ensemble fini d'ensembles de types atomique, initialement fait des singletons contenant les types atomiques apparaissant dans α_1 et α_2 :

$$U_{\alpha_1}^{\alpha_2} = \{\{\tau_1(m)\}, \{\tau_2(m)\} \mid m \in M\}$$

  Grâce à l'ensemble $U_{\alpha_1}^{\alpha_2}$, nous maintenons, au fil de l'exécution de l'algorithme d'unification, une partition des types atomiques, de telle sorte que deux types atomiques appartenant à la même partie doivent être unifiés.

Finalement, nous définissons une fonction récursive $\texttt{mgu}$ par $\texttt{mgu}(R \cup \{a = b\}, U \cup \{U_a\} \cup \{U_b\}) = \texttt{mgu}(R, U \cup \{U_a \cup U_b\})$, où R est un ensemble d'équations de la forme $a_1 = a_2$, a_1 et a_2 étant des types atomiques, U est un ensemble d'ensembles de types atomiques, et U_a et U_b sont les seuls ensembles contenant respectivement a et b dans $U \cup \{U_a\} \cup \{U_b\}$.

Puisque les deux ensembles de types atomiques $\mathcal{A}_1$ et $\mathcal{A}_2$ sont disjoints et que α_1 et α_2 sont uniformes, l'unification de ces types est toujours réalisable. De plus, le résultat de cette unification est donné par le résultat de $\texttt{mgu}(R^{\alpha_2}_{\alpha_1}, U^{\alpha_2}_{\alpha_1})$, et en particulier par la valeur finale du second paramètre, indiquant les types atomiques devant être unifiés, et donc le renommage σ tel que $\alpha_1 \cdot \sigma = \alpha_2 \cdot \sigma$.

Remarquons qu'une telle valeur finale est toujours atteinte ; en effet, $R^{\alpha_2}_{\alpha_1}$ contient un nombre fini d'équations, et la valeur du premier paramètre décroît à chaque étape itérative. Cette fonction s'arrête donc lorsque le premier paramètre est égal à l'ensemble vide.

Notation 7.2.1. Pour un ensemble d'équations R de la forme $\tau_1(m) = \tau_2(m)$ et pour $i \in \{1,2\}$ nous construisons l'ensemble $\mathrm{At}^P(\pi_i R)$ (*resp.* $\mathrm{At}^O(\pi_i R)$), tel que m est un P-coup (*resp.* un O-coup) dans l'arène A_i. L'ensemble $\mathrm{At}(\pi_i R)$ désignera l'ensemble $\mathrm{At}^P(\pi_i R) \cup \mathrm{At}^O(\pi_i R)$.

Nous considérons donc $\alpha_1 \in \mathcal{U}_\gamma(\mathcal{A}_1)$ et $\alpha_2 \in \mathcal{U}_\gamma(\mathcal{A}_2)$ négativement non-dupliquant et de polarité négative et positive respectivement, et tels que $\mathcal{A}_1$ et $\mathcal{A}_2$ sont deux ensembles disjoints de types atomiques :

Lemme 7.2.1.1. *Étant donnés σ l'unificateur le plus général de α_1 et α_2 et $i, \bar{i} \in \{1,2\}$ tels que $i \neq \bar{i}$, les propriétés suivantes sont vérifiées :*

1. *Si $a_1, a_2 \in \mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1}) - \mathrm{At}^P(\pi_i R^{\alpha_2}_{\alpha_1})$, alors $a_1 \cdot \sigma = a_2 \cdot \sigma$ ssi $a_1 = a_2$.*
2. *Si $a_1 \in \mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1}) - \mathrm{At}^P(\pi_i R^{\alpha_2}_{\alpha_1})$, alors tout type atomique $a_2 \in \mathrm{At}(\pi_{\bar{i}} R^{\alpha_2}_{\alpha_1})$ tel que $a_1 \cdot \sigma = a_2 \cdot \sigma$, appartient à $\mathrm{At}^P(\alpha_{\bar{i}})$.*

Démonstration. Prouvons ces deux propriétés :

1. Si $a_1 = a_2$, il est évident que $a_1 \cdot \sigma = a_2 \cdot \sigma$. Nous prouvons la seconde implication par induction sur le nombre d'équations dans le premier paramètre de $\texttt{mgu}(R^{\alpha_2}_{\alpha_1}, U^{\alpha_2}_{\alpha_1})$. Si cet ensemble ne contient qu'une équation, alors $R^{\alpha_2}_{\alpha_1} = \{a_1 = a_2\}$ et $\texttt{mgu}(R^{\alpha_2}_{\alpha_1}, \{\{a_1\}, \{a_2\}\}) = \texttt{mgu}(\emptyset, \{\{a_1, a_2\}\})$, d'où $a_1 \cdot \sigma = a_2 \cdot \sigma$.

   Sinon, étant donnés $a_1, a_2 \in \mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1}) - \mathrm{At}^P(\pi_i R^{\alpha_2}_{\alpha_1})$ et deux équations $E_1 : a_1 = b_1$ et $E_2 : a_2 = b_2$ dans $R^{\alpha_2}_{\alpha_1}$, nous savons par hypothèse que tout coup m tel que $\tau_i(m) = a_1$ appartient à $\mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1})$; puisque α_i et $\alpha_{\bar{i}}$ sont uniformes entre eux, et de polarités opposées, un tel coup m appartient à $\mathrm{At}^P(\pi_{\bar{i}} R^{\alpha_2}_{\alpha_1})$. La même propriété est vérifiée pour tout coup m tel que $\tau_i(m) = a_2$. Nous avons donc une configuration schématisée comme suit, ou les cellules brunes dénotent des O-coups, les cellules blanches des P-coups :

   $$\alpha_1 = \boxed{\ldots \quad a_1 \quad \ldots \quad a_2 \quad \ldots}$$

   $$\alpha_2 = \boxed{\ldots \quad b_1 \quad \ldots \quad b_2 \quad \ldots}$$

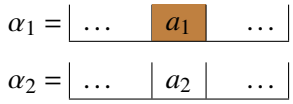
   Alors, puisque $\alpha_{\bar{i}}$ est négativement non-dupliquant, $R = R^{\alpha_2}_{\alpha_1} - \{(a_1 = b_1), (a_2 = b_2)\}$ vérifie $b_1, b_2 \notin \mathrm{At}^P(\pi_{\bar{i}} R)$. De plus, pour que $a_1 \cdot \sigma = a_2 \cdot \sigma$ soit vérifié, il faut que b_1 et b_2 appartiennent à $\mathrm{At}^O(\pi_{\bar{i}} R)$. Par hypothèse d'induction, $b_1 = b_2 = b$, et puisque $\alpha_{\bar{i}}$ est un type négativement non-dupliquant, il existe un seul coup tel que $\tau_{\bar{i}}(m) = b$. Nous pouvons conclure que deux coups m_1 et m_2 vérifient $\tau_i(m_1) = a_1$ et $\tau_i(m_2) = a_2$ ssi $m_1 = m_2 = m$, ce qui implique que $a_1 = a_2$.

2. Nous prouvons de manière identique la seconde propriété. Si il existe une seule équation dans $R^{\alpha_2}_{\alpha_1}$, celle-ci est $a_1 = a_2$, où a_1 et a_2 ont des polarités opposées, et en supposant que $a_1 \in \mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1}) - \mathrm{At}^P(\pi_i R^{\alpha_2}_{\alpha_1})$, nous obtenons que a_2 appartient à $\mathrm{At}^P(\alpha_{\bar{i}})$.

   Supposons à présent qu'il y ait plus d'une équation dans $R^{\alpha_2}_{\alpha_1}$ et posons $R^{\alpha_2}_{\alpha_1} = R \cup \{(a_1 = a_2)\}$ et $U^{\alpha_2}_{\alpha_1} = U \cup \{\{a_1\},\{a_2\}\}$. Alors le calcul de l'unificateur le plus général entre α_1 et α_2 est :

$$\mathtt{mgu}(R \cup \{a_1 = a_2\}, U \cup \{\{a_1\},\{a_2\}\}) = \mathtt{mgu}(R, U \cup \{\{a_1, a_2\}\})$$

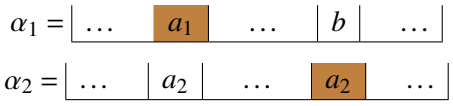
   Supposons, $a_1 \in \mathrm{At}^O(\pi_i R^{\alpha_2}_{\alpha_1}) - \mathrm{At}^P(\pi_i R^{\alpha_2}_{\alpha_1})$; alors tous les coups m tels que $\tau_i(m) = a_1$ et $\tau_{\bar{i}}(m) = a_2$ sont des O-coups dans A_i et des P-coups dans $A_{\bar{i}}$ par définition. De plus, puisque $\alpha_{\bar{i}}$ est négativement non-dupliquant, il existe une unique position m vérifiant les propriétés énoncés ci-dessus. Alors, si il n'existe pas d'équation de la forme $b = a_2$ dans R, la propriété est vérifiée (nous nous retrouvons dans la situation schématisée ci-dessous)



   Sinon, prenons une telle équation $b = a_2$ (remarquons que b ne peut être égal à a_1 par définition de $R^{\alpha_2}_{\alpha_1}$ pour les raisons énoncées ci-dessus) et écrivons $R = R' \cup \{(b = a_2)\}$ et $U = U' \cup \{b\}$. Alors, par définition de mgu,

$$\mathtt{mgu}(R, U) = \mathtt{mgu}(R', U' \cup \{\{a_1, a_2, b\}\})$$

   Le même raisonnement que précédemment montre que $b \in \mathrm{At}^O(\pi_i R') - \mathrm{At}^P(\pi_i R')$. D'après l'hypothèse d'induction, tout type atomique b' appartenant à $\pi_{\bar{i}} R'$ et qui est unifié avec b a une occurrence négative dans $\alpha_{\bar{i}}$. Par conséquent, nous pouvons en conclure qu'il en est de même pour les types atomiques apparaissant dans $\pi_{\bar{i}} R^{\alpha_2}_{\alpha_1}$ et qui sont unifiés avec a_1.



□

Théorème 7.2.1.1. *Soient deux typages listés* $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1 \to \beta_1$ *et* $\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$*, négativement non-dupliquants et tels que* $\alpha_1 \in \mathcal{U}_\gamma(\mathcal{A}_1)$ *et* $\alpha_2 \in \mathcal{U}_\gamma(\mathcal{A}_2)$ *où :*

- γ *est un type simple.*
- $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$

De plus, considérons σ *comme l'unificateur le plus général entre* α_1 *entre* α_2*.*

Alors le typage $\langle \mathbf{\Delta_1}, \mathbf{\Delta_2}; \Gamma_1, \Gamma_2 \rangle \cdot \sigma \vdash \beta_1 \cdot \sigma$ *est négativement non-dupliquant.*

Démonstration. Supposons que ce ne soit pas le cas et considérons A comme l'arène associée à $\langle \mathbf{\Delta_1 \Delta_2}; \Gamma_1, \Gamma_2 \rangle \cdot \sigma \vdash \beta_1 \cdot \sigma$. Alors, il existe $a_1, a_2 \in \mathrm{At}^P(A)$ tels que $a_1 \cdot \sigma = a_2 \cdot \sigma$. De plus, a_1 et a_2 apparaissent sur α_1 ou α_2. Posons A_1 et A_2 les arènes associées à $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1 \to \beta_1$ et $\langle \mathbf{\Delta_2}; \Gamma_2 \rangle \vdash \alpha_2$ respectivement

1. si $a_1, a_2 \in \mathrm{At}^P(A_1)$, alors a_1 et a_2 ont chacun une occurrence positive dans $\langle \mathbf{\Delta_1}; \Gamma_1 \rangle \vdash \alpha_1 \to \beta_1$ et apparaissant sur α_1. Le lemme 7.2.1.1.1 implique que $a_1 = a_2$.
2. si $a_1, a_2 \in \mathrm{At}^P(A_2;)$, alors a_1 et a_2 ont chacun une occurrence positive sur α_2, et nous obtenons le même résultat.

3. sinon, le lemme 7.2.1.1.2 invalide la cas où $a_1 \in \mathrm{At}^P(A_1)$ et $a_2 \in \mathrm{At}^P(A_2;)$.

□

Il est intéressant de remarquer que ce théorème ne nécessite pas d'hypothèse particulière sur la prouvabilité des séquents (*i.e.* sur le fait que les typages soient habités). Par ailleurs, ce théorème peut parfaitement être prouvé en considérant les typages principaux des termes quasi-affines dans le système de typage simple. Nous pouvons à présent en conclure que la famille des quasi-affines est close par composition :

Corollaire 7.2.1.1. *[Stabilité par composition] Étant donné deux termes quasi-affines M_1 et M_2, le terme $|M_1 M_2|_\beta$ est quasi-affine.*

Démonstration. D'après le Théorème 7.2.1.1, le théorème $M_1 M_2$ a un typage général listé négativement non-dupliquant. D'après le Théorème 7.1.2.4, ce terme a pour forme normale un terme quasi-affine. □

Nous nous attachons à présent à définir la famille une famille de typages pour les termes quasi-affines, permettant de vérifier la propriété de cohérence et un théorème analogue à la propriété d'expansion du sujet.

7.2.2 Théorème de cohérence des typages PN

Nous venons de voir que les termes quasi-affines ont un typage principal listé négativement non-dupliquant. Qui plus est, cette propriété s'étend aux termes appartenant à la clôture par composition de la famille des termes quasi-affines. Nous allons à présent relaxer la contrainte, pour un typage, d'être négativement non-dupliquant, afin de ne s'intéresser, d'une part, qu'aux coups potentiels de l'arène (les informations de typage sur les coups injouables n'ayant pas d'influence sur le théorème de cohérence) et, d'autre part, en quotientant les coups d'une arène par une classe d'équivalence. Dans cette étude, nous montrons en avant des typages dits *potentiellement négativement non-dupliquants*, qui vérifient également le théorème de cohérence dans le système de typage listé.

Équivalence de coups et propriétés

Nous avons vu que, étant donné un typage (simple ou listé) $\langle \mathbf{\Delta}; \Gamma \rangle \vdash$ et son arène associée $A = (M, \tau)$, nous pouvons séparer les coups de A en coups potentiels et en coups injouables. Ces derniers n'intervenant pas dans la recherche d'une stratégie gagnante sur A, il est possible de caractériser des typages qui sont uniquement habités en appliquant le théorème de [Aoto, 1999] uniquement sur les types atomiques apparaissant sur les coups potentiels de A, c'est-à-dire en vérifiant que tout type atomique apparaissant sur des coups potentiels de A apparaît sur au plus un P-coup potentiel de A.

Nous montrons que ce théorème de cohérence, dans les cas des typages listés, restent vrai à équivalence près, pour la relation d'équivalence suivante :

Définition 7.2.2.1. *Soit un typage listé $\langle \mathbf{\Delta}; \Gamma \rangle \vdash \alpha$ et son arène associée $A = (M, \tau)$. Deux coups m_1 et m_2 de M sont dits* équivalents dans *A (noté $m_1 \approx_A m_2$) si $m_1 = m_2$, ou s'il existe $s \in \mathbb{N}$, $\mathbf{c} \in \mathrm{Dom}(\mathbf{\Delta})$ et $i_1, i_2 \in \mathbb{N}$ tels que :*

1. $m_1 = (s_{\mathbf{c}}, i_1)$, $m_2 = (s_{\mathbf{c}}, i_2)$ *et*
2. $m_1 \in \mathrm{Pot}(A)$ *ssi* $m_2 \in \mathrm{Pot}(A)$, *et*
3. *si* $m_1, m_2 \in \mathrm{Pot}(A)$, *alors* $\tau(m_1) = \tau(m_2)$

4. *si il existe* $n_1, n_2 \in M$ *tel que* $n_1 \vdash m_1$ *et* $n_2 \vdash m_2$*, alors* $n_1 \approx_A n_2$
5. *pour tout* $l \in \mathbb{N}$ *tels que* $(s_{\mathbf{c}} \cdot l, i) \in M$*, où* $i \in \{i_1, i_2\}$*,* $(s_{\mathbf{c}} \cdot l, i_1) \approx_A (s_{\mathbf{c}} \cdot l, i_2)$

Il est direct de vérifier que la relation $\approx_A$ est une relation d'équivalence sur A. Nous parlerons donc des classes d'équivalence de $A/\approx_A$. De manière informelle, cette relation d'équivalence porte sur les types simples composant un type listé assigné à une même constante ; ainsi, deux types simples ainsi donnés sont équivalents si ils sont égaux modulo les types étiquetant leurs coups injouables associés.

Notation 7.2.2. Étant donnée une arène A, nous noterons par $\mathbf{m}, \mathbf{n}, \mathbf{m_1}, \ldots$ les classes d'équivalences de $A/\approx_A$.

Remarquons que, s'il existe $m \in \mathbf{m}$ tel que $m \in M^P$ (*resp.* $m \in M^O$), alors tout élément $n \in \mathbf{m}$ est un P-coup (*resp.* O-coup). Nous étendons donc la notion d'assignation de joueurs (et donc de polarité) aux classes d'équivalence d'une arène, par la notation $\mathbf{m} \in M^P/\approx_A$ et $\mathbf{m} \in M^O/\approx_A$. De même, remarquons que deux coups m_1, m_2 de $\mathbf{m} \in A/\approx_A$ vérifient $\tau(m_1) = \tau(m_2)$; nous noterons donc $\tau(\mathbf{m})$ le type associé à une classe d'équivalence de $A/\approx_A$.

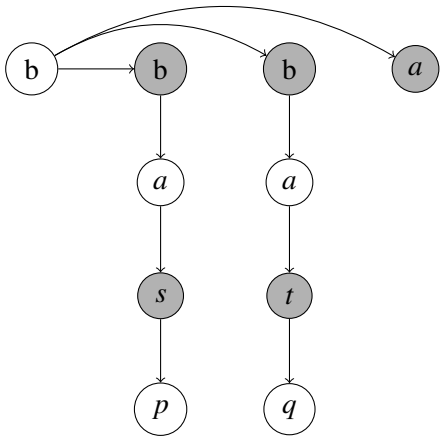


FIG. 7.3 – Illustration d'équivalences de coups

Exemple 7.2.2.1. Prenons pour exemple le terme $\mathbf{c}^{((o\to o)\to o)\to o}(\lambda y^{o\to o}.\mathbf{d}^o)$ et le typage suivant :

$$\langle \mathbf{d}^o : a, \mathbf{c}^{((o\to o)\to o)\to o} : ((q \to t) \to a) \to b \cap ((p \to s) \to a) \to b; \_\rangle \vdash b$$

Considérons $A = (M, \tau)$ l'arène de typage associée (représentée en Figure 7.3), telle que :

$$M = \{\epsilon, \epsilon_{\mathbf{d}}, (\epsilon_{\mathbf{c}}, 1), (\epsilon_{\mathbf{c}}, 2), (1_{\mathbf{c}}, 1), (1_{\mathbf{c}}, 2), (11_{\mathbf{c}}, 1), (11_{\mathbf{c}}, 2), (111_{\mathbf{c}}, 1), (111_{\mathbf{c}}, 2)\}$$

Nous pouvons calculer $\mathrm{Pot}(A) = \{\epsilon, \epsilon_{\mathbf{d}}, (\epsilon_{\mathbf{c}}, 1), (\epsilon_{\mathbf{c}}, 2)\}$, et remarquer que les deux coups $(\epsilon_{\mathbf{c}}, 1)$ et $(\epsilon_{\mathbf{c}}, 2)\}$ sont équivalents (par définition, il en est de même pour $(1_{\mathbf{c}}, 1)$ et $(1_{\mathbf{c}}, 2)$, pour $(11_{\mathbf{c}}, 1)$ et $(11_{\mathbf{c}}, 2)$, et pour $(111_{\mathbf{c}}, 1)$ et $(111_{\mathbf{c}}, 2)$). Il existe deux stratégies (gagnantes) Σ_1 et Σ_2 associées à cette arène et que nous donnons par les ensembles de séquences maximales suivants :

- $max(\Sigma_1) = \{\epsilon.(\epsilon_{\mathbf{c}}, 1).\epsilon_{\mathbf{d}}\}$
- $max(\Sigma_2) = \{\epsilon.(\epsilon_{\mathbf{c}}, 2).\epsilon_{\mathbf{d}}\}$

et ces deux stratégies vérifient $[\![\Sigma_1]\!]_A = [\![\Sigma_2]\!]_A = \mathbf{c}^{((o\to o)\to o)\to o}(\lambda y^{o\to o}.\mathbf{d}^o)$; il existe donc un unique habitant pour ce typage.

La notion d'équivalence associée aux coups d'une arène peut être étendue aux séquences de coups et aux stratégies sur cette arène :

Définition 7.2.2.2. *Soit une arène de typage A ; nous dirons que deux séquences justifiées de coups S_1 et S_2 sur A sont équivalentes sur A (noté $S_1 \approx_A S_2$) si :*

- $S_1 = (m_1, 0), S_2 = (m_2, 0)$ *et* $m_1 \approx_A m_2$.
- $S_1 = S_1'.(m_1, i), S_2 = S_2'.(m_2, i)$, $S_1' \approx_A S_2'$ *et* $m_1 \approx_A m_2$.

De même, deux ensembles de séquences justifiées Σ_1 et Σ_2 sont dits équivalents sur A (noté $\Sigma_1 \approx_A \Sigma_2$) si pour toute séquence de Σ_1, il existe une séquence équivalente sur A dans Σ_2, et inversement.

Remarquons que deux stratégies Σ_1 et Σ_2 gagnantes sur une arène A sont équivalentes si et seulement si $max(\Sigma_1) \approx_A max(\Sigma_2)$.

Étant données une arène de typage $A = (M, \tau)$, une séquence de coups S sur A et une classe $\mathbf{m} \in M/\approx_A$, nous parlerons des *occurrences de* $\mathbf{m}$ *dans* S pour désigner l'ensemble des occurrences des coups $m \in \mathbf{m}$ dans S.

Sur l'exemple précédent, les deux stratégies Σ_1 et Σ_2 sont équivalentes. Le théorème suivant est direct :

Théorème 7.2.2.1. *Soit une arène de typage $A = (M, \tau)$. Si il existe deux stratégies Σ_1 et Σ_2 gagnantes sur A et équivalentes, alors $[\![\Sigma_1]\!]_A = [\![\Sigma_2]\!]_A$.*

Typages PN : définition et propriétés

Nous étendons à présent la notion de typages négativement non-dupliquants, afin de ne prendre en compte que les coups potentiels d'une arène dans la caractérisation des typages uniquement habités. En effet, nous avons vu à travers plusieurs exemples, que la propriété d'expansion du sujet n'est pas vérifiée pour les termes quasi-affines ; néanmoins, nous allons monter que le type associé à un coup potentiel m de l'arène associée au typage principal d'un terme M quasi-affine est identique à celui assigné au même coup m sur l'arène du typage principal d'un terme M' tel que $M' \to_\beta^* M$. Il nous sera donc possible de ne s'intéresser qu'aux occurrences des types apparaissant sur les coups potentiels afin de caractériser la propriété de cohérence :

Définition 7.2.2.3. *Étant donnés un typage (simple ou listé) $\langle \Delta; \Gamma \rangle \vdash \alpha$ et son arène $A = (M, \tau)$, ce typage est dit* potentiellement négativement non-dupliquant *(nous parlerons de* typage PN *pour plus de concision) si :*

pour tout type atomique a, si il existe $m \in \mathrm{Pot}(A)$ tel que $\tau(m) = a$, alors il existe une et une seule classe $\mathbf{m} \in M^P/\approx_A$

Il est important de noter que cette condition ne porte que sur les P-coups potentiels d'une arène A associée à un typage PN. Il est donc possible que des O-coups (éventuellement injouables) de A partagent le même type atomique.

La formulation des typages PN étant proche de celle des typages négativement non-dupliquants, nous prouvons que le théorème de cohérence s'étend aux typages PN.

Lemme 7.2.2.1. *Soit $\langle \Delta; \Gamma \rangle \vdash \alpha$ un typage PN et Σ une stratégie victorieuse sur $A = (M, \tau)$, l'arène associée. Alors pour toute séquence $S \in \Sigma$ et pour tout $\mathbf{n} \in M^O/\approx_A$, il existe au plus une occurrence de $\mathbf{n}$ dans S.*

Démonstration. Supposons que l'énoncé soit faux ; il existe donc une séquence S de Σ, de la forme

$$S_1.(n_1,i_1).(m_1,j_1).S_2.(n_2,i_2).(m_2,j_2)$$

où $n_1 \approx_A n_2$. Alors, pour tout préfixe P de $S_2.(n_2,i_2).(m_2,j_2)$ nous montrons qu'il existe $P' \approx P$ tel que $S.P' \in \Sigma$, par induction sur $|P|/2$:

- si $|P| = 0$, le résultat est direct.
- si $|P| = 2p+2$, pour $p \in \mathbb{N}$; nous supposons que le résultat est vraie pour tout préfixe de $S_2.(n_2,i_2).(m_2,j_2)$, de longueur paire et inférieure à $2p$. Considérons P comme la séquence $S_p.(m_p,j_p).(n,i).(m,j)$. Par hypothèse d'induction, il existe une séquence $S'_p.(m'_p,j'_p)$ telle que $S'_p.(m'_p,j_p) \approx_A S_p.(m_p,j_p)$, et $S.S'_p.(m'_p,j_p)$ appartient à Σ. Par définition, il existe $n' \in M$ tel que $n \approx n'$. De plus, $n,n' \in \mathrm{Pot}(A)$, d'où $\tau(n) = \tau(n')$. Il reste à montrer qu'il existe $m' \in M^P$ tel que $m \approx_A m'$ et $S.(m'_p,j_p).S'_p.(n',i).(m',j) \in \Sigma$. Soit le O-coup o tel que (o,j) apparaisse dans $S.S_p.(m_p,j_p).(n_2,i_2)$. Par hypothèse, il existe un O-coup o' tel que (o,j) apparaît dans $S'.S'_p.(m'_p,j_p).(n'_2,i_2)$. De plus $o \approx_A o'$. Par définition, il existe alors m' tel que $o' \vdash m'$ et $m' \approx_A m$. De plus, m et m' sont potentiels ce qui implique $\tau(m') = \tau(m)$.

La stratégie Σ serait donc faite d'un nombre infini de séquences, ce qui est impossible. Nous en concluons l'existence d'au plus une occurrence de $\mathbf{n}$ dans une séquence S de Σ. □

Théorème 7.2.2.2. *Soit $\langle \Delta;\Gamma\rangle \vdash \alpha$ un typage PN ; alors il existe au plus un habitant N sous forme normale-longue pour $\langle \Delta;\Gamma\rangle \vdash \alpha$*

Démonstration. Soit $A = (M,\tau)$ l'arène associée à $\langle \Delta;\Gamma\rangle \vdash \alpha$. Supposons l'existence d'une stratégie gagnante Σ telle que $[\![\Sigma]\!]_A = N$. Supposons qu'il existe une stratégie gagnante $\Sigma' \not\approx_A \Sigma$ sur A. Étant donnée une séquence $S' \in \Sigma'$, nous montrons par induction sur $|S'|/2$ qu'il existe une séquence $S \in \Sigma$ équivalente à S' sur A :

- si $|S'| = 2$; alors $S' = (\epsilon,0).(m',0)$; donc m' est potentiel dans A. Soit $\mathbf{m} \in A/_{\approx_A}$ telle que $m' \in \mathbf{m}$. D'après la définition d'un typage PN, tout P-coup m de A tel que $\tau(m) = \tau(m')$, vérifie $m \in \mathbf{m}$. Or il existe une séquence $S = (\epsilon,0).(m,0)$ dans Σ, d'où $m \approx_A m'$. Nous en concluons que $S \approx_A S'$ est vraie.
- si $|S'| = 2n+2$ et si la propriété est vraie pour toute séquence S'' telle que $|S''| = 2n$, $n \in \mathbb{N} - \{0\}$. Alors, puisque $S' = S'_1.(n',i).(m',j')$, par hypothèse d'induction, nous obtenons l'existence d'une séquence $S_1 \in \Sigma$ telle que $S_1 \approx_A S'_1$. En particulier, pour $S'_1 = S'_2.(m'_1,k)$ et $S_1 = S_2.(m_1,k)$, nous obtenons $m_1 \approx_A m'_1$. De plus, si $m_1 \neq m'_1$, alors pour $m'_1 = (s_{\mathbf{c}},i_1)$, m_1 est de la forme $(s_{\mathbf{c}},i_2)$, où $i_1 \neq i_2$. Pour $n' = (s_{\mathbf{c}} \cdot l, i_1)$, $l \in \mathbb{N}$, il existe alors $n = (s_{\mathbf{c}} \cdot l, i_2)$ tel que n' et n sont des coups potentiels de A et $n \approx_A n'$, par définition de la relation d'équivalence entre m_1 et m'_1 ; on en déduit que $\tau(n) = \tau(n')$. Il existe donc $m \in M^P$ tel que $S = S_1.(n,i).(m,j)$ appartient à Σ. De plus, m est un P-coup potentiel de A, vérifiant $\tau(m) = \tau(m')$ et par définition d'un typage PN, $m \approx_A m'$.
  Il reste à montrer que $j = j'$. Or, étant donnés $S'_1.(o_1,j) \sqsubseteq S'$ et $\mathbf{o} \in A/\approx_A$ tel que $o \in \mathbf{p}$, d'après le lemme 7.2.2.1, il n'existe pas d'autre occurrence de O-coups de $\mathbf{o}$ dans S'. Puisque $S \approx_A S'$, il en est de même pour le O-coup o_2 tel que $S_1.(o_2,l) \sqsubseteq S$. Enfin, ces séquences sont forcément équivalentes dans A, ce qui implique que $j = j'$. On en conclut que $S \approx_A S'$.

□

Les typages potentiellement négativement non-dupliquants permettent donc d'obtenir la propriété de cohérence pour les termes quasi-affines dans le système de typage listé, de manière plus fine que les typages négativement non-dupliquants. Nous nous servons donc des typages PN afin d'obtenir une

propriété similaire à l'expansion du sujet, en cherchant certains typages PN qu'un terme quasi-affine M partage avec les termes quasi-affines obtenus par β-expansion de M.

Remarque 7.2.2.1. Nous remarquerons que la définition d'un typage PN n'est pas optimale pour préserver l'unicité de stratégies. En effet, pour certains typages, il est possible que, si un type atomique apparaît sur un coup potentiel de l'arène associée, ce type atomique apparaisse sur un P-coup injouable de l'arène. Par exemple, dans le terme $M = \lambda x^{o\to o}.\mathbf{c}^o$, un typage préservant l'unicité serait $\mathbf{c}^o : a \vdash M : (b \to a) \to a$, et ce dernier n'est pas un typage PN. Néanmoins, pour $M = \lambda x^o.\mathbf{c}^o$, il n'existe pas de typage qui ne soit pas un typage PN et pour lequel l'unicité de stratégie soit assurée. La définition de typage PN donne un critère suffisant afin d'assurer l'unicité de stratégies.

Par ailleurs, la notion d'équivalence de coups que nous avons introduite ne permet pas de capturer tous les typages pour lesquels il existe plusieurs dérivations et qui sont uniquement habités. Ainsi, étant donné le terme $M = \mathbf{c}^{(o\to o)\to o}(\lambda y^{o\to o}.\mathbf{d}^o)$ de l'exemple 7.2.2.1, considérons le typage suivant de M :

$$\langle \mathbf{d}^o : a_1 \cap a_2, \mathbf{c}^{(o\to o)\to o} : ((q \to t) \to a_1) \to b \cap ((p \to s) \to a_2) \to b; \_\rangle \vdash b$$

Les deux stratégies Σ_1 et Σ_2 évoquées dans l'exemple 7.2.2.1 sont deux stratégies gagnantes sur l'arène associée à ce typage, et vérifient $[\![\Sigma_1]\!]_A = [\![\Sigma_2]\!]_A$. Cependant, ces deux stratégies ne sont pas équivalentes sur l'arène associée à ce typage, du fait que $a_1 \neq a_2$.

7.2.3 Renommages et typages PN moins généraux

Dans ce qui suit, nous considérons des termes clos. Ainsi, les typages de tels termes seront notés $\Delta \vdash \alpha$ plutôt que $\langle \Delta; \_\rangle \vdash \alpha$. De plus, les termes que nous étudions appartiennent à une signature d'ordre supérieure Σ et sont donc vus comme des termes typés à la Church ; ainsi les typages listés que nous associons à un tel terme M seront uniformes avec le typage de M dans sa signature Σ. L'ensemble des constantes pouvant avoir une occurrence dans un terme obtenu par β-expansion de M est ainsi fini, et plus exactement inclus dans l'ensemble C des constantes de la signature $\Sigma = (A, C, \tau)$.

Nous présentons la construction des typages PN les moins généraux d'un terme exclusivement comme un problème de typage. Ainsi, étant donné $\Delta \vdash \alpha$ un typage PN associé à un terme quasi-affine M, nous montrons qu'un typage $\Delta' \vdash \alpha'$ PN moins général de M vérifie la propriété que pour tout renommage σ, $\Delta' \cdot \sigma \vdash \alpha' \cdot \sigma$ n'est pas un typage PN. De plus, $\Delta' \vdash \alpha'$ étant moins général que $\Delta \vdash \alpha$, il existe donc un renommage de types σ' entre ces deux typages ; nous montrons que $\sigma' = \sigma_1 \circ \sigma_2$, où σ_2 modifie les types atomiques de $\mathrm{Inj}(A)$, et σ_1, les types atomiques de $\mathrm{Pot}(A)$, où A est l'arène associée à $\Delta \vdash \alpha$.

Définition 7.2.3.1. *Soient un typage $\Delta \vdash \alpha$, et son arène $A = (M, \tau)$. L'ensemble des* types atomiques potentiels *de $\Delta \vdash \alpha$ est défini par* $\mathrm{AtPot}(\Delta \vdash \alpha) = \{\tau(m) \mid m \in \mathrm{Pot}(A)\}$. *L'ensemble des* types atomiques négligeables *de $\Delta \vdash \alpha$ est défini par* $\mathrm{AtNgg}(\Delta \vdash \alpha) = \{\tau(m) \mid m \in \mathrm{Inj}(A)\} - \mathrm{AtPot}(\Delta \vdash \alpha)$.

Rappelons que, dans le cas particulier où $\Delta \vdash \alpha$ est le typage principal (listé ou simple) d'un certain terme N sous forme β-réduite, les ensembles $\{\tau(m) \in \mathcal{A} \mid m \in \mathrm{Pot}(A)\}$ et $\{\tau(m) \in \mathcal{A} \mid m \in \mathrm{Inj}(A)\}$ sont disjoints. Il est aisé de le prouver en utilisant la définition d'une stratégie recouvrante sur une arène de typage.

Notation 7.2.3. Étant donnés un typage $\Delta \vdash \alpha$, l'arène associée $A = (M, \tau)$ et un ensemble de types atomiques E ayant au moins une occurrence dans $\Delta \vdash \alpha$, nous noterons par $E^+ = E \cap \{\tau(m) \mid m \in M^O\}$ et par $E^- = E \cap \{\tau(m) \mid m \in M^P\}$.

Lemme 7.2.3.1. *Soit $\Delta \vdash \alpha$ un typage PN pour lequel il existe un habitant N. Alors* $\mathrm{AtPot}^+(\Delta \vdash \alpha) = \mathrm{AtPot}^-(\Delta \vdash \alpha)$.

Démonstration. Puisqu'il existe un habitant N pour ce typage, $|N|_\beta$ habite également ce typage, d'après la propriété de réduction du sujet. De plus, puisque $\Delta \vdash \alpha$ est un typage PN, $|N|_\beta$ est l'unique habitant de ce typage sous forme normale. Notons par $\mathbf{\Sigma}$ l'ensemble des stratégies gagnantes Σ telles que $[\![\Sigma]\!]_A = |N|_\beta$, où $A = (M, \tau)$ est l'arène associée à $\Delta \vdash \alpha$. Alors $\mathrm{Pot}(A) = \{m \in M \mid m \text{ apparaît dans } \mathbf{\Sigma}\}$. Alors, puisque les stratégies de $\mathbf{\Sigma}$ sont équivalentes deux à deux, pour tout $\mathbf{n} \in \mathrm{Pot}^O(A)/_{\approx_A}$, il existe $\mathbf{m} \in \mathrm{Pot}^P(A)/_{\approx_A}$ tel que $\tau(\mathbf{n}) = \tau(\mathbf{m})$, et inversement. Nous en concluons que $\mathrm{AtPot}^+(\Delta \vdash \alpha) = \mathrm{AtPot}^-(\Delta \vdash \alpha)$. □

Définition 7.2.3.2. *Soit un typage $\Delta \vdash \alpha$; un renommage σ est dit* potentiel (*resp.* négligeable) sur $\Delta \vdash \alpha$ *si l'ensemble $\{a \mid a \cdot \sigma \neq a\}$ est inclus dans* $\mathrm{AtPot}(\Delta \vdash \alpha)$ *(resp. dans* $\mathrm{AtNgg}(\Delta \vdash \alpha)$*).*

Définition 7.2.3.3. *Étant donné $\Delta \vdash \alpha$ un typage PN, un renommage σ est dit* PN-préservant pour $\Delta \vdash \alpha$ *si $\Delta \cdot \sigma \vdash \alpha \cdot \sigma$ est un typage PN. De plus, si σ est potentiel (resp. négligeable), nous dirons que σ est* PN-potentiel *(resp.* PN-négligeable*).*

Nous montrons à présent comment construire les typages PN les moins généraux associés à un terme quasi-affine à partir de son typage principal listé.

Typages Σ-saturés, renommages PN-négligeables et gestion de l'effacement

Étant donnés une signature d'ordre supérieur Σ, un terme quasi-affine $M \in \Lambda(\Sigma)$ et son typage principal listé $\Delta \vdash \alpha$, nous cherchons, dans un premier temps, à trouver l'ensemble de ses typages PN qui sont moins généraux que $\Delta \vdash \alpha$ par application de renommages PN-négligeables. Par cette méthode, nous cherchons à retrouver les typages PN de termes quasi-affines $M' \in \Lambda(\Sigma)$ obtenus par β-expansion effaçante de M.

Définition 7.2.3.4. *Soient deux termes M et M' tels que $M \to_\beta M'$. Cette β-réduction (et l'opération inverse de β-expansion) est dite* effaçante *si $M = C[(\lambda x.N)P]$ et $M' = C[N]$.*

Afin d'illustrer les problématiques engendrées par de tels β-réduction, prenons les termes suivants :

Exemple 7.2.3.1. Soit un terme $N = \lambda f^{o\to o} x^o y^o.\mathbf{c_1} x^o y^o$ appartenant à la signature d'ordre supérieur $\Sigma = \{\{o\}, \{\mathbf{c_1}, \mathbf{c_2}\}, \{\mathbf{c_1} \mapsto o \to o \to o, \mathbf{c_2} \mapsto o \to o \to o\}\}$. Le typage principal listé de N est $\mathbf{c_1} : a \to b \to c \vdash (u \to v) \to a \to b \to c$. Nous considérons à présent des termes quasi-affines β-équivalents à N, et construits sur la signature Σ.

1. $N_1 = \lambda f^{o\to o\to o} x^o y^o.((\lambda g^o.\mathbf{c_1} x^o y^o)(\mathbf{c_2} x^o x^o))$, dont le typage principal listé est $\mathbf{c_1} : a \to b \to c, \mathbf{c_2} : a \to a \to d \vdash (u \to v \to t) \to a \to b \to c$
2. $N_2 = \lambda f^{o\to o\to o} x^o y^o.((\lambda g^o.\mathbf{c_1} x^o y^o)(\mathbf{c_1} x^o x^o))$, dont le typage principal listé est $\mathbf{c_1} : a \to b \to c \cap a \to a \to d \vdash (u \to v \to t) \to a \to b \to c$
3. $N_3 = \lambda f^{o\to o\to o} x^o y^o.((\lambda g^o.\mathbf{c_1} x^o y^o)(f^{o\to o\to o} x^o x^o))$, dont le typage principal listé est $\mathbf{c_1} : a \to b \to c \vdash (a \to a \to t) \to a \to b \to c$

Le premier exemple montre que certaines constantes de Σ n'ayant pas d'occurrences dans un terme quasi-affine N peuvent apparaître dans un terme N' obtenu par β-expansion effaçante de N ; tout typage de N' doit donc inclure des informations de typage sur ces constantes, contrainte qui n'existe pas dans les typages de N'. Comme illustré par le terme N_2, il est plus exactement possible

de faire apparaître de nouvelles occurrences de constantes dans le terme N. Par ailleurs, remarquons que, étant donné le typage principal listé $\mathbf{\Delta}' \vdash \alpha'$ d'un terme quasi-affine N' se réduisant en N par β-réduction effaçante, il est possible que les types atomiques de $\mathrm{AtNgg}^+(\mathbf{\Delta}' \vdash \alpha')$ appartiennent également à $\mathrm{AtPot}(\mathbf{\Delta}' \vdash \alpha')$. Ceci est illustré dans chacun des exemples.

Cet exemple illustre le fait qu'il est possible, dans certains cas de β-expansions effaçantes n'introduisant pas de nouvelles occurrences de constantes, de construire un typage PN $\mathbf{\Delta} \cdot \sigma \vdash \alpha \cdot \sigma$, par application d'un renommage σ sur un typage PN $\mathbf{\Delta} \vdash \alpha$, tel que σ ne modifie que les éléments de $\mathrm{AtNgg}^+(\mathbf{\Delta} \vdash \alpha)$.

Remarque 7.2.3.1. Dans les démonstrations qui suivent, étant donnés un typage $\mathbf{\Delta} \vdash \alpha$, l'arène associée $A = (M, \tau)$ et un renommage σ, nous considérons A', l'arène associée à $\mathbf{\Delta} \cdot \sigma \vdash \alpha \cdot \sigma$, comme définie par $A' = (M, \tau')$ *i.e.* l'ensemble des coups n'est pas modifié. Ceci revient à ne pas considérer les réductions par idempotence de $\cap$ dans les types listés appartenant à $\mathrm{Im}(\mathbf{\Delta} \cdot \sigma)$.

Propriété 7.2.3.1. *Étant donnés $\mathbf{\Delta} \vdash \alpha$ un typage PN et σ un renommage négligeable sur ce typage, si pour tout $a \in \mathrm{AtNgg}^-(\mathbf{\Delta} \vdash \alpha)$, $a \cdot \sigma$ n'appartient pas à $\mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha)$ alors σ est PN-négligeable pour $\mathbf{\Delta} \vdash \alpha$.*

Démonstration. Considérons $A = (M, \tau)$ et $A' = (M, \tau')$ les arènes associées à $\mathbf{\Delta} \vdash \alpha$ et $\mathbf{\Delta} \cdot \sigma \vdash \alpha \cdot \sigma$ respectivement. Soit $a \in \mathrm{AtNgg}^-(\mathbf{\Delta} \vdash \alpha)$; par définition de AtNgg, tout coup $m \in M$ tel que $\tau(m) = a$ vérifie $m \in \mathrm{Inj}(A)$. Par définition d'un typage PN, pour tout coup $m \in \mathrm{Inj}^P(A)$ et tout coup $m' \in \mathrm{Pot}^P(A)$, $\tau(m) \neq \tau(m')$. De plus, d'après le lemme 7.2.3.1, ce résultat s'étend au cas où $m' \in \mathrm{Pot}(A)$, et par définition de σ, nous obtenons $\tau(m) \cdot \sigma \neq \tau(m') \cdot \sigma$. Pour un O-coup $n \in \mathrm{Pot}(A)$, nous avons donc l'égalité suivante : $\{m \in M^P \mid \tau(m) = \tau(n)\} = \{m \in M^P \mid \tau'(m) = \tau'(n)\}$. Par conséquent, les ensembles $\mathrm{Pot}(A)$ et $\mathrm{Pot}(A')$ sont égaux, et il en est donc de même pour les ensembles $\mathrm{Pot}(A)/\approx_A$ et $\mathrm{Pot}(A')/\approx_{A'}$. Nous obtenons que pour toutes classes $\mathbf{m}, \mathbf{m}' \in \mathrm{Pot}^P(A')/\approx_{A'}$, si $\tau'(\mathbf{m}) = \tau'(\mathbf{m}')$, alors $\mathbf{m} = \mathbf{m}'$. Finalement, pour tout $\mathbf{m} \in M/\approx_{A'}$, l'hypothèse sur σ assure que pour toute classe $\mathbf{m} \in M^P/\approx_{A'}$ et tout $m \in M^P$, si $\tau(m) = \tau(\mathbf{m})$, nous avons $m \in \mathbf{m}$. Le typage $\mathbf{\Delta} \cdot \sigma \vdash \alpha \cdot \sigma$ est donc bien un typage PN. □

D'après ce lemme, étant donné un renommage négligeable pour un typage PN $\mathbf{\Delta} \vdash \alpha$, nous n'avons besoin que de considérer les valeurs que σ modifie sur les types atomiques de $\mathrm{AtNgg}^-(\mathbf{\Delta} \vdash \alpha)$ afin de nous assurer que σ est PN-préservant. Nous pouvons ainsi définir une notion de maximalité sur l'ensemble des renommages négligeables associés à un certain typage :

Définition 7.2.3.5. *Étant donnés un typage PN $\mathbf{\Delta} \vdash \alpha$, un renommage PN-négligeable σ pour ce typage est dit* maximal *si :*

- *pour tout type atomique de* $\mathrm{AtNgg}^-(\mathbf{\Delta} \vdash \alpha)$, $a \cdot \sigma = \omega \notin \mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha)$.
- *pour tout type atomique de* $\mathrm{AtNgg}^+(\mathbf{\Delta} \vdash \alpha)$, $a \cdot \sigma \in \mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha) \cup \{\omega\}$.

Étant donné un typage PN, $\mathbf{\Delta} \vdash \alpha$, nous notons par $\Omega(\mathbf{\Delta} \vdash \alpha)$ l'ensemble des renommages PN-négligeables maximaux de $\mathbf{\Delta} \vdash \alpha$. De plus, d'après la propriété 7.2.3.1, un tel renommage est PN-négligeable.

Cette notion de maximalité se retrouve à travers la notion de typages listés associés à un terme d'une signature d'ordre supérieur :

Définition 7.2.3.6. *Étant donnés une signature d'ordre supérieure $\Sigma = (\mathcal{A}, C, \mathit{type})$, un terme $M \in \Lambda(\Sigma)$ et $\mathbf{\Delta} \vdash \alpha$ son type typage principal listé, dont les types atomiques forment un ensemble $\mathcal{A}'$, le* typage Σ-principal listé *de M est défini par $\mathbf{\Delta}_\Sigma \vdash \alpha$ tel que pour toute constante $\mathbf{c} \in C$, $\mathbf{\Delta}_\Sigma(\mathbf{c}) = \mathbf{\Delta}(\mathbf{c}) \cap \gamma$, où $\gamma \in \mathcal{U}_{\mathit{type}(\mathbf{c})}(\mathcal{B})$, $\mathcal{B} \cap \mathcal{A}' = \emptyset$ et tout type atomique apparaissant dans γ a une unique occurrence dans $\mathbf{\Delta}_\Sigma \vdash \alpha$.*

Exemple 7.2.3.2. Soit le terme $N = \lambda f^{o\to o} x^o y^o.\mathbf{c_1} x^o y^o$ construit sur la signature d'ordre supérieure $\Sigma = (\{o\}, \{\mathbf{c_1}, \mathbf{c_2}\}, \{\mathbf{c_1} \mapsto o \to o \to o, \mathbf{c_2} \mapsto o \to o \to o\})$. Son typage principal listé est $\mathbf{c_1} : a \to b \to c \vdash (u \to v) \to a \to b \to c$. Le typage Σ-principal listé de N est donc :

$$\mathbf{c_1} : a \to b \to c \cap u_1 \to u_2 \to u_3, \mathbf{c_2} : u_4 \to u_5 \to u_6 \vdash (u \to v) \to a \to b \to c$$

L'idée d'un tel typage est donc de typer l'ensemble des constantes de la signature, et l'ensemble des occurrences de constantes, pouvant apparaître par β-expansion effaçante d'un terme quasi-affine. Cette idée apparaît plus clairement par la définition du typage suivant :

Définition 7.2.3.7. *Étant donnés une signature d'ordre supérieure $\Sigma = (\mathcal{A}, C, type)$, un terme quasi-affine $M \in \Lambda(\Sigma)$ et $\mathbf{\Delta}_\Sigma \vdash \alpha$ son typage Σ-principal listé, un typage $\mathbf{\Delta}_{sat} \vdash \gamma$ est appelé* typage saturé de *M si :*

- *pour toute constante $\mathbf{c} \in C$, et tout renommage $\sigma \in \Omega(\mathbf{\Delta}_\Sigma \vdash \alpha)$, $\alpha \in \mathbf{\Delta}_{sat}(\mathbf{c})$ ssi il existe $\alpha' \in \mathbf{\Delta}_\Sigma(\mathbf{c})$ tel que $\alpha' = \alpha \cdot \sigma$, et*
- *il existe $\sigma \in \Omega(\mathbf{\Delta}_\Sigma \vdash \alpha)$ tel que $\gamma = \alpha \cdot \sigma$.*

Remarquons qu'il existe donc plusieurs typages saturés pour un terme quasi-affine N. Par ailleurs, ces typages partagent le même environnement de typage. Qui plus est, étant donné le typage principal listé $\mathbf{\Delta} \vdash \alpha$ d'un terme quasi-affine N, les typages saturés de N sont construits sur l'ensemble fini $\mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha) \cup \{\omega\}$.

Notation 7.2.4. Étant donné un ensemble fini de types atomiques $\mathcal{A} = \{a_1, \ldots, a_n\}$ et un type α égal à $a \to b$ par exemple, nous notons par $\bigwedge_{x \in \mathcal{A}} x \to b$ le type listé $a_1 \to b \cap \ldots \cap a_n \to b$.

Exemple 7.2.3.3. Reprenons le terme N et la signature Σ de l'exemple 7.2.3.2. Pour $\mathbf{\Delta}_\Sigma \vdash \alpha$ le typage Σ-principal listé de N, nous savons que $\mathcal{A} = \mathrm{AtPot}(\mathbf{\Delta}_\Sigma \vdash \alpha) = \{a, b, c\}$; les typages saturés de N sont donc de la forme $\mathbf{c_1} : \overline{\alpha_1}, \mathbf{c_2} : \overline{\alpha_2} \vdash \gamma$ où :

- $\overline{\alpha_1} = a \to b \to c \cap \bigwedge_{x_1, x_2 \in \mathcal{A} \cup \{\omega\}} x_1 \to x_2 \to \omega$.
- $\overline{\alpha_2} = \bigwedge_{x_1, x_2 \in \mathcal{A} \cup \{\omega\}} x_1 \to x_2 \to \omega$.
- $\gamma = (x \to \omega) \to a \to b \to c$ où $x \in \mathcal{A} \cup \{\omega\}$

On remarquera qu'un typage saturé d'un terme quasi-affine N contient toutes les informations de typage associées à des occurrences de constantes susceptibles d'apparaître par β-expansion de N.

Nous donnons à présent un sens à la notion de maximalité associée aux typage PN-négligeables maximaux pour un tel typage, en prouvant qu'il n'existe pas de renommages PN-négligeables pour typage Σ-saturé d'un terme quasi-affine.

Lemme 7.2.3.2. *Étant donnés une signature $\Sigma = (\mathcal{A}, C, type)$ telle que $|C| > 1$, un terme quasi-affine $N \in \Lambda(\Sigma)$ et $\mathbf{\Delta}_{sat} \vdash \alpha$ un de ses typages saturés, le typage $\mathbf{\Delta}_{sat} \cdot \sigma \vdash \alpha \cdot \sigma$ est un typage PN ssi σ est un renommage PN-potentiel*

Démonstration. L'implication gauche est évidente, par définition d'un renommage PN-potentiel. Supposons alors que $\mathbf{\Delta} \cdot \sigma \vdash \alpha \cdot \sigma$ soit un typage PN, et que σ ne soit pas un renommage potentiel. Il existe donc $a \notin \mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha)$ tel que $a \cdot \sigma \neq a$. Donc a appartient à $\mathrm{AtNgg}(\mathbf{\Delta} \vdash \alpha)$; or cet ensemble est égal à $\{\omega\}$ par définition. Puisque σ est un renommage, $\omega \cdot \sigma \in \mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha)$. De plus, étant donnée l'arène $A = (M, \tau)$ associée à $\mathbf{\Delta} \vdash \alpha$, pour tout P-coup $m \in \mathrm{Inj}(A)$ nous avons $\tau(m) = \omega$. Puisque $|C| > 1$ et par construction de $\mathbf{\Delta}_{sat}$, il existe $\mathbf{c_1}, \mathbf{c_2} \in C$ et $i, j \in \mathbb{N}$ tels que $\tau((\epsilon_{\mathbf{c_1}}, i)) = \tau((\epsilon_{\mathbf{c_2}}, j)) = \omega$. En particulier, si nous prenons $A' = (M, \tau')$ comme arène associée à $\mathbf{\Delta} \cdot (\sigma \circ \sigma_N) \vdash \alpha \cdot (\sigma \circ \sigma_N)$, il n'existe pas de classe $\mathbf{m} \in M^P / \approx_{A'}$ contenant $(\epsilon_{\mathbf{c_1}}, i)$ et $(\epsilon_{\mathbf{c_2}}, j)$. Le typage $\mathbf{\Delta}_{sat} \cdot \sigma \vdash \alpha \cdot \sigma$ ne peut donc pas être un typage PN. □

Remarquons que, prenant les termes de l'exemple 7.2.3.1, les typages saturés des termes obtenus par β-expansion de N sont également des typages saturés de N. Nous allons à présent utiliser le lemme 7.2.3.2, afin d'obtenir les typages PN les moins généraux associés à un terme quasi-affine.

Renommages PN-potentiels et gestion des copies

Étant donnés une signature d'ordre supérieure $\Sigma = (\mathcal{A}, C, \mathit{type})$ et un terme $N \in \Lambda(\Sigma)$, nous utilisons la notion de renommages PN-potentiels sur les typages saturés de N afin d'identifier les occurrences de constantes ayant été copiées. De tels cas sont fréquents en sémantique de Montague, comme le montre le terme suivant :

$$N = \exists(\lambda x.\wedge\ (\mathbf{cake}\ x)\ (\wedge(\mathbf{buy}\ \mathbf{Mary}\ x)\ (\mathbf{eat}\ \mathbf{Mary}\ x)))$$

qui est la représentation sémantique de la structure syntaxique représentée par le terme

$$S = (\mathit{and}\ \mathit{buy}\ \mathit{eat})\ (\mathit{a}\ \mathit{cake})\ \mathit{Mary}$$

Sur cet exemple, les deux occurrences de la constante **Mary** dans la forme sémantique de cette phrase résulte de la duplication de cette constante par β-réduction de $\mathcal{H}(S)$, $\mathcal{H}$ étant le morphisme d'interprétation de S dans la sémantique de Montague. Par contre, les deux occurrences de $\wedge$ dans N ne sont pas produites par un mécanisme de copie au niveau de la β-expansion de $\mathcal{H}$: l'une d'elle provient de $\mathcal{H}(\mathit{and}) = \lambda PQxy.\wedge(Px)(Qx)$, la seconde provenant de $\mathcal{H}(a) = \lambda PQ.\exists(\lambda x.\wedge(Px)(Qx))$.

Définition 7.2.3.8. *Étant donnés deux termes N_1 et N_2 tels que $N_1 = C[(\lambda x^\alpha.P)Q] \rightarrow_\beta N_2 = C[P[x^\alpha := Q]]$, nous dirons que cette étape de β-réduction est* copiante *si il existe plusieurs occurrences libres de x^α dans P.*

Par définition, les β-réductions copiantes de termes quasi-affines respectent la propriété supplémentaire suivante : le type α est un type atomique. Nous parlons alors de *β-réduction copiante du premier ordre* ; l'opération inverse sera appelée β-expansion copiante du premier-ordre.

Dans [Kanazawa, 2007], de tels cas de β-expansion sont traités pour les termes quasi-linéaires, en utilisant des propriétés syntaxique sur les λ-termes, afin d'identifier les occurrences de constantes produites par copie (du premier ordre) d'une constante. Nous montrons qu'il est possible de s'abstraire de la syntaxe des termes pour traiter ce problème, et d'utiliser la notion de renommage PN-potentiel précédemment définie.

Exemple 7.2.3.4. Considérons la signature d'ordre supérieure $\Sigma = (\{o\}, \{\mathbf{c_1}, \mathbf{c_2}\}, \{\mathbf{c_1} \mapsto o \rightarrow o, \mathbf{c_2} \mapsto o \rightarrow o\})$ et le terme $N = \lambda f^{o\rightarrow o\rightarrow o} x^o y^o.f^{o\rightarrow o\rightarrow o}(\mathbf{c_1}x^o)(\mathbf{c_1}x^o)$. Étant donné l'ensemble de types atomiques $\mathcal{A} = \{a, b_1, b_2, c, \omega\}$, ce terme possède un unique typage saturé :

$$\mathbf{c_1} : a \rightarrow b_1 \cap a \rightarrow b_2 \cap \bigwedge_{x\in\mathcal{A}} x \rightarrow \omega, \mathbf{c_2} : \bigwedge_{x\in\mathcal{A}} x \rightarrow \omega \vdash (b_1 \rightarrow b_2 \rightarrow c) \rightarrow a \rightarrow \omega \rightarrow c$$

Mais, étant donné $\mathcal{A}' = \{a, b, c, \omega\}$, le typage $\mathbf{c_1} : a \rightarrow b \cap \bigwedge_{x\in\mathcal{A}'} x \rightarrow \omega, \mathbf{c_2} : \bigwedge_{x\in\mathcal{A}'} x \rightarrow \omega \vdash (b \rightarrow b \rightarrow c) \rightarrow a \rightarrow \omega \rightarrow c$ est également un typage PN de N, obtenu par application du renommage PN-potentiel $\sigma = \{b_1, b_2 \mapsto b\}$ sur le typage saturé de N.

Sur cet exemple, nous remarquerons que l'application de σ sur le typage saturé de N revient à assigner le même type aux deux occurrences de la constante $\mathbf{c_1}$, présentes dans le terme N. De cette manière, nous verrons que nous assurons que le même type est assigné à toutes les occurrences d'une constante issues de la copie d'une même constante par β-expansion.

Nous démontrons, à présent, que pour tout typage PN, il existe un renommage PN-potentiel maximal. À cette fin, nous introduisons les notions suivantes :

Définition 7.2.3.9. *Étant donnés un renommage σ et un type atomique $a \in \mathrm{Dom}(\sigma)$, l'ensemble* $\mathrm{Eq}_\sigma(a)$ *est défini par* $\{a' \in \mathrm{Dom}(\sigma) \mid a \cdot \sigma = a' \cdot \sigma\}$.

Théorème 7.2.3.1. *Étant donnés un typage PN $\Delta \vdash \alpha$ et deux renommages PN-potentiels σ_1 et σ_2 sur ce typage, il existe un renommage PN-potentiel tel que $\Delta \cdot \sigma \vdash \alpha \cdot \sigma$ est moins général que $\Delta \cdot \sigma_1 \vdash \alpha \cdot \sigma_1$ et $\Delta \cdot \sigma_2 \vdash \alpha \cdot \sigma_2$.*

Démonstration. Construisons le renommage σ de la manière suivante :

- $\mathrm{Dom}(\sigma) = \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma_2)$.
- pour tout $a \in \mathrm{Dom}(\sigma)$, $\mathrm{Eq}_\sigma(a) = (\mathrm{Eq}_{\sigma_1}(a) \cup \mathrm{Eq}_{\sigma_2}(a))^*$, la clôture transitive de $\mathrm{Eq}_{\sigma_1}(a) \cup \mathrm{Eq}_{\sigma_2}$.

Il existe donc σ'_1 et σ'_2 tels que $\sigma = \sigma'_1 \circ \sigma_1 = \sigma'_2 \circ \sigma_2$. Par conséquent, $\Delta \cdot \sigma \vdash \alpha \cdot \sigma$ est moins général que $\Delta \cdot \sigma_1 \vdash \alpha \cdot \sigma_1$ et $\Delta \cdot \sigma_2 \vdash \alpha \cdot \sigma_2$. De plus, σ est un renommage potentiel $\Delta \vdash \alpha$. Il suffit donc de montrer que σ est un renommage PN-préservant.

Considérons deux types atomiques a et a' tels que $a' \in \mathrm{Eq}_\sigma(a)$. Par définition, il existe $n \in \mathbb{N}$ tels que $a_0 = a$, $a_n = a'$ et pour tout $i \in \{0, \dots, n-1\}$, $a_i \cdot \sigma_{f(i)} = a_{i+1}$, où $f : \{0, \dots, n\} \mapsto \{1, 2\}$. Étant donnée $A = (M, \tau)$ l'arène de typage associée à $\Delta \vdash \alpha$, pour tout $i \in \{0, \dots, n\}$, nous notons par $\mathbf{m_i} \in \mathrm{Pot}(A)/\approx_A$ la classe telle que $\tau(\mathbf{m_i}) = a_i$. Puisque σ_1 et σ_2 sont PN-préservants, nous savons que pour tout $i \in \{0, \dots, n-1\}$, les P-coups $m_i \in \mathbf{m_i}$ et $m_{i+1} \in \mathbf{m_{i+1}}$ sont équivalents dans $A_{f(i)}$, l'arène associée à $\Delta \cdot \sigma_{f(i)} \vdash \alpha \cdot \sigma_{f(i)}$; ils le sont donc également dans $A' = (M, \tau')$, l'arène associée à $\Delta \cdot \sigma \vdash \alpha \cdot \sigma$, l'application de $\sigma'_{f(i)}$ ne modifiant pas cette propriété. Par transitivité, tous P-coups $m_1, m_2 \in \bigcup_{i \in \{0,\dots,n\}} \mathbf{m}_i$ sont équivalents dans A'. Enfin, puisque le renommage est potentiel, nous obtenons donc l'unicité de la classe de P-coups $\mathbf{m} \in M^P(A')/\approx_{A'}$ tels que $\tau(\mathbf{m}') = a \cdot \sigma$, et le typage $\Delta \cdot \sigma \vdash \alpha \cdot \sigma$ est bien un typage PN. □

Du Théorème 7.2.3.1 et du Lemme 7.2.3.2 nous déduisions l'existence d'un renommage PN-potentiel σ_P tel que, étant donné un typage saturé $\Delta_{sat} \vdash \alpha$ associé à un terme quasi-affine N, pour tout renommage σ, $\Delta_{sat} \cdot (\sigma \circ \sigma_P) \vdash \alpha \cdot (\sigma \circ \sigma_P)$ n'est pas un typage PN de N.

Par ailleurs, ce renommage PN-potentiel σ_P est commun à tous les typages saturés de N ; en effet, si nous considérons deux arènes $A_1 = (M_1, \tau_1)$ et $A_2 = (M_2, \tau_2)$ associées chacune à un typage saturé d'un tel terme N, ces arènes partagent les mêmes coups potentiels, et les fonctions τ_1 et τ_2 assignent les mêmes types atomiques à ces coups potentiels. Sachant que σ_P ne peut modifier le type atomique ω qui est le seul type atomique négligeable de tous les typages saturés de N, σ_P est commun à ces deux typages saturés.

Définition 7.2.3.10. *Étant donné un terme quasi-affine N, un typage $\Delta \vdash \alpha$ de N est dit* typage PN moins général *si il existe un typage saturé $\Delta_{sat} \vdash \alpha'$ de N tel que $\Delta \vdash \alpha = \Delta_{sat} \cdot \sigma_P \vdash \alpha' \cdot \sigma_P$, où σ_P est le renommage PN-potentiel maximal de $\Delta_{sat} \vdash \alpha'$.*

Étant donné un terme quasi-affine $N \in \Lambda(\Sigma)$, où Σ est une signature d'ordre supérieure, nous parlerons du *typage PN le moins général de N engendré par σ_N* pour le typage $\Delta_{sat} \cdot \sigma_P \vdash \alpha \cdot \sigma_P \circ \sigma_N$, où :

1. σ_N appartient à $\Omega(\Delta \vdash \alpha)$.
2. $\Delta_{sat} \vdash \alpha \cdot \sigma_N$ est un typage saturé de N.
3. σ_P est le renommage PN-potentiel maximal de $\Delta_{sat} \vdash \alpha \cdot \sigma_N$.

On remarquera donc que tous les typages PN moins généraux d'un terme quasi-affine N partagent le même environnement de typage $\Delta_{sat} \cdot \sigma_P$.

Il ne nous reste plus qu'à prouver le théorème suivant :

Théorème 7.2.3.2 (Pseudo expansion du sujet)**.** *Étant donnés une signature d'ordre supérieur* Σ *et un terme quasi-affine* $N \in \Lambda(\Sigma)$ *sous forme normale, pour tout terme quasi-affine* $N' \in \Lambda(\Sigma)$ *tel que* $N' \to_\beta^* N$*, il existe un typage PN moins général de* N*,* $\Delta_{mg} \vdash \alpha_{mg}$ *dont* N' *est un habitant.*

Démonstration. Considérons les typages principaux listés $\Delta \vdash \alpha$ et $\Delta' \vdash \alpha'$ de N et N' respectivement. Puisque N et N' sont des termes quasi-affines, ces typages sont négativement non-dupliquants, d'après le théorème 7.2.1.1. De plus, par la propriété de réduction du sujet, $\Delta' \vdash \alpha'$ est un typage de N ; il existe, par définition, un renommage σ tel que pour toute constante $\mathbf{c} \in \mathrm{Dom}(\Delta)$, $\Delta(\mathbf{c}) \cdot \sigma \subseteq \Delta'(\mathbf{c})$, et tel que $\alpha \cdot \sigma = \alpha'$. Nous considérons $\sigma_1 = \sigma|_{\mathrm{AtNgg}(\Delta \vdash \alpha)}$ et $\sigma_2 = \sigma|_{\mathrm{AtPot}(\Delta \vdash \alpha)}$ comme le renommage σ dont le domaine est restreint à $\mathrm{AtNgg}(\Delta \vdash \alpha)$ et $\mathrm{AtPot}(\Delta \vdash \alpha)$ respectivement. Puisque $\Delta \vdash \alpha$ est le typage principal listé de N et que ce dernier est sous-forme normale $\{m \in M \mid \tau(m) \in \mathrm{AtNgg}(\Delta \vdash \alpha)\} = \mathrm{Inj}(A)$, et inversement $\{m \in M \mid \tau(m) \in \mathrm{AtPot}(\Delta \vdash \alpha)\} = \mathrm{Pot}(A)$. Les domaines de σ_1 et σ_2 sont donc disjoints. Alors, étant donné l'ensemble $\mathcal{A}'$ des types atomiques du typage Σ-principal listé de N', il existe un renommage $\sigma_\omega : \mathrm{AtNgg}(\Delta' \vdash \alpha') \mapsto \{\omega\}$, où $\omega \notin \mathcal{A}$, tel que $\sigma_N = \sigma_\omega \circ \sigma_1$ est un renommage PN-négligeable maximal pour le typage Σ-principal listé de N. De même, il existe un renommage σ_2' tel que $\sigma_2' \circ \sigma_2 = \sigma_P$, le renommage PN-potentiel maximal du typage saturé de N engendré par σ_N. Par construction, étant donné $\Delta_\Sigma \vdash \alpha$ le typage Σ-principal de N, $\Delta_\Sigma \cdot (\sigma_P \circ \sigma_N) \vdash \alpha \cdot (\sigma_P \circ \sigma_N)$ est un typage de N' et un typage PN moins général de N. □

7.3 Les reconnaisseurs Datalog

Nous venons de donner les détails théoriques permettant de résoudre le problème de reconnaissance dans les ACGs quasi-affines du second-ordre. Nous montrons, à présent, comment construire des reconnaisseurs Datalog pour ces grammaires, et comment les résultats théoriques précédents permettent de prouver la correction et la complétude cette construction.

Dans ce qui suit, nous considérons donc $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ comme une ACG quasi-affine du second-ordre (où $\Sigma_1 = (A_1, C_1, \mathit{type}_1)$ et $\Sigma_2 = (A_2, C_2, \mathit{type}_2)$) et un terme quasi-affine $N \in \Lambda(\Sigma_2)$. Le problème de reconnaissance est donc de savoir si N appartient à $O(\mathcal{G})$.

7.3.1 Construction

Comme nous avons pu le voir dans la section précédente, la propriété d'expansion du sujet n'est pas vérifiée pour les termes quasi-affines. Malgré tout, nous nous appuyons sur le théorème 7.2.3.2 et sur l'unification de variables Datalog afin que le reconnaisseur puisse trouver les typages PN moins généraux du terme N qui sont habités par un terme $\mathcal{H}(N')$, N' appartenant à $\mathcal{A}(\mathcal{G})$.

Dans un premier temps, nous donnons une construction générale de reconnaisseurs Datalog, avant de regarder le cas plus spécifique du reconnaisseur construit à partir de l'ensemble des typages PN moins généraux du terme quasi-affine N en entrée du problème.

Notation 7.3.1. Étant donnés un type $\delta \in \mathcal{T}(A)$ et un type $\gamma \in \mathcal{U}_\delta(B)$, nous écrirons $\gamma = \delta[\overrightarrow{\gamma}]$, où $\overrightarrow{\gamma}$ est la séquence de types atomiques apparaissant dans γ de gauche à droite. Ainsi, $\overrightarrow{\gamma}$ est une séquence de types atomiques que nous utiliserons comme argument de prédicats Datalog.

De plus, pour un typage $\Delta \vdash \alpha$, et un type γ appartenant à $\{\alpha\} \cup \{\beta \mid \exists \mathbf{c} \in \mathrm{Dom}(\Delta), \beta \in \Delta(\mathbf{c})\}$, la notation $\gamma = \delta[\overrightarrow{\gamma}(x_1, \ldots, x_n)]$ sera utilisé afin de souligner que γ appartient à $\mathcal{U}_\delta(B)$, pour un certain ensemble B, et que γ contient n occurrences positives de types atomiques négligeables dans $\Delta \vdash \alpha$, chacune de ces occurrences étant remplacée par une variable parmi $x_1, \ldots, x_n$. Lorsque cette information n'est pas nécessaire, nous écrirons simplement $\gamma = \delta[\overrightarrow{\gamma}]$.

De la même manière que [Kanazawa, 2007], nous divisons la construction des reconnaisseurs Datalog en deux parties :

Les règles liées au terme N

Étant donné le terme N en entrée tel que $Cst(N) = \{\mathbf{c_1}, \ldots, \mathbf{c_n}\} \subseteq C_2$, nous considérons un typage $\Delta \vdash \alpha$ de N. Nous dirons que $\Delta \vdash \alpha$ est le *typage de référence* pour le reconnaisseur.

1. Pour toute constante $\mathbf{c} \in C_2$ et tout type $\delta[\overrightarrow{\gamma}(x_1, \ldots, x_n)] \in \Delta(\mathbf{c})$, nous associons la règle (éventuellement) intentionnelle :

$$c(\overrightarrow{\gamma}(x_1, \ldots, x_n)) \text{:- } \texttt{atome}(x_1), \ldots, \texttt{atome}(x_n)$$

2. Pour tout type atomique a apparaissant dans $\Delta \vdash \alpha$, nous créons le fait :

$$\texttt{atom(a)}$$

L'introduction des faits décrits en 1. est en fait similaire à la construction de [Kanazawa, 2007]. La deuxième règle nous permet uniquement d'avoir un contrôle sur les types atomiques pouvant apparaître, comme nous le verrons pour les règles associées à la grammaire.

Remarquons que les types atomiques de $\delta[\overrightarrow{\gamma}(x_1, \ldots, x_n)]$ apparaissant sur des O-coups négligeables de l'arène associée $\Delta \vdash \alpha$ sont identifiés par des variables Datalog, les autres occurrences de types atomiques étant identifiées par une constante associée au type atomique lui-même. Par ailleurs, cette règle n'est pas une règle intentionnelle si $n = 0$.

Remarque 7.3.1.1. Grâce à ces règles, étant donnés une constante $\mathbf{c} \in C_2$ et un type $\gamma \in \Delta(\mathbf{c})$, il est possible de dériver et de ne dériver que les types $\gamma \cdot \sigma$, où σ est un renommage des occurrences de types atomiques apparaissant sur les O-coups injouables de l'arène de $\Delta \vdash \alpha$ dans $\mathrm{AtPot}(\Delta \vdash \alpha)$. Un tel renommage correspond en fait à une instanciation de variables dans le programme Datalog.

On peut donc noter que, de cette manière, une règle Datalog dont la tête contient le prédicat c associé à une constante $\mathbf{c} \in C_1$ permet de dériver plusieurs types simples appartenant à $\Delta(\mathbf{c})$. En fait, étant donnée l'arène A associée à $\Delta \vdash \alpha$, si deux types γ_i et γ_j de $\Delta(\mathbf{c})$ vérifient $(\epsilon_{\mathbf{c}}, i) \approx_A (\epsilon_{\mathbf{c}}, j)$, nous n'associons qu'une règle Datalog commune à ces deux typages. En effet, dans ce cas, γ_i et γ_j ne diffère que sur les occurrences de types atomiques sur des O-coups injouables de l'arène (les P-coups injouables étant tous étiquetés ω). Notre construction permet donc de factoriser les règles relatives aux types simples contenus dans les types listés assignés aux constantes.

Par ailleurs, nous pouvons faire la même remarque pour deux types de la forme $\gamma_1 \to \ldots \to \gamma_n \to \omega$, assignés à une même constante $\mathbf{c}$, puisque ces deux types partageront également le même type atomique sur les P-coups injouables de l'arène A.

Les règles liées à la grammaire $\mathcal{G}$

Considérons une constante $\mathbf{c} \in C_1$ et le typage principal listé de $\mathcal{H}(\mathbf{c})$:

$$\mathbf{c_1} : \alpha_{11} \cap \ldots \cap \alpha_{1k_1}, \ldots, \mathbf{c_m} : \alpha_{m1} \cap \ldots \cap \alpha_{mk_m} \vdash \gamma$$

Nous associons à $\mathbf{c}$ la règle intentionnelle suivante :

$$p_0(\overrightarrow{x_0}) \text{:- } p_1(\overrightarrow{x_1}), \ldots, p_n(\overrightarrow{x_n}), c_1(\overrightarrow{y_{11}}), \ldots, c_1(\overrightarrow{y_{1k_1}}), \ldots, c_m(\overrightarrow{y_{m1}}), \ldots, c_m(\overrightarrow{y_{mk_m}}), \texttt{atome}(z_1), \ldots, \texttt{atome}(z_l)$$

de telle sorte que :

- $type_1(\mathbf{c}) = p_1 \to \ldots p_n \to p_0$
- $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}, \overrightarrow{y_{11}}, \ldots, \overrightarrow{y_{1k_1}}, \ldots, \overrightarrow{y_{m1}}, \ldots, \overrightarrow{y_{mk_m}}$ sont des séquences de variables Datalog.
- pour tout $i \in \{1, \ldots, m\}$ et tout $j \in \{1, \ldots, k_i\}$, $\overrightarrow{\alpha_{ij}} = \overrightarrow{y_{ij}}$.
- $\gamma = \mathcal{H}(p_1)[\overrightarrow{x_1}] \to \ldots \to \mathcal{H}(p_n)[\overrightarrow{x_n}] \to \mathcal{H}(p_0)[\overrightarrow{x_0}]$
- x_i appartient à $\{z_1, \ldots, z_l\}$ si et seulement si x_i apparaît dans $\overrightarrow{x_0}$ et a une unique occurrence dans la séquence $\overrightarrow{x_0}, \ldots, \overrightarrow{x_n}, \overrightarrow{y_1}, \ldots \overrightarrow{y_m}$.

On remarquera que les variables Datalog $z_1, \ldots, z_l$ sont associées aux positions injouables dans l'arène de $\mathbf{\Gamma} \vdash \gamma$. Nous contraignons donc, grâce aux règles dont la tête contient le prédicat `atome`, ces positions à être assignées des types atomiques appartenant au typage $\mathbf{\Delta} \vdash \alpha$ de référence dans la construction du reconnaisseur. Il est à noter que ces prédicats n'existent pas dans les reconnaisseurs Datalog proposés par [Kanazawa, 2007] ; en effet, l'absence d'effacement dans les λ-termes assure que tout type atomique a au moins une occurrence négative et une occurrence positive dans le typage principal d'un terme quasi-linéaire. Il n'y est donc pas nécessaire d'imposer des valeurs à certaines occurrences de types atomiques. Enfin, un des avantages en programmation logique de l'introduction des prédicats `atome` est qu'il permet d'assurer la condition de sûreté.

Grâce à ces règles, nous pouvons voir que, étant donnés une constante $\mathbf{c} \in C_1$, et le typage principal listé $\mathbf{\Gamma} \vdash \gamma$ de $\mathcal{H}(\mathbf{c})$, ces règles Datalog permettent de dériver tous les typages $\mathbf{\Gamma} \cdot \sigma \vdash \gamma \cdot \sigma$, où σ est un renommage de types qui substitue les types de $\mathrm{AtNgg}(\mathbf{\Gamma} \vdash \gamma)$ par des types atomiques ayant au moins une occurrence dans le typage de référence $\mathbf{\Delta} \vdash \alpha$.

Notation 7.3.2. Nous notons un programme construit selon la méthode précédemment décrite par $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$.

Nous allons à présent montrer qu'un tel programme Datalog permet de savoir si, étant donnés un terme $N \in \Lambda(\Sigma_2)$ et un typage $\mathbf{\Delta} \vdash \alpha$ de ce terme, il existe un terme $N' \in \Lambda(\Sigma_1)$ tel que $\mathcal{H}(N') \to_\beta^* N$. Il est important de remarquer que nous n'utilisons alors que le typage $\mathbf{\Delta} \vdash \alpha$ et les typages des termes $\mathcal{H}(\mathbf{c})$, où $\mathbf{c}$ appartient à C_1 afin de résoudre ce problème.

7.3.2 Correction et complétude de la construction

Étant donnés un reconnaisseur Datalog $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$ et une requête $?\text{-}\, p(\overrightarrow{\alpha})$ construits comme précédemment décrits, nous noterons l'existence d'une dérivation de $p(\overrightarrow{\alpha})$ par

$$\texttt{recog}(\mathcal{G}, \mathbf{\Delta}) \text{:-}\, p(\overrightarrow{\alpha})$$

Remarque 7.3.2.1. Il est possible d'encoder l'ensemble des typages PN moins généraux du terme N ; en effet, ces typages ne variant que sur les occurrences de types atomiques appartenant au type α, et qui sont négligeables et positives dans $\mathbf{\Delta} \vdash \alpha$, lancer la requête $p(\overrightarrow{\alpha}(x_1, \ldots, x_n))$ sur le reconnaisseur, revient à chercher un typage PN reconnu par $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$. Dans ce cas, nous écrirons

$$\texttt{recog}(\mathcal{G}, \mathbf{\Delta}) \text{:-}\, p(\overrightarrow{\alpha}(a_1, \ldots, a_n))$$

afin d'indiquer que le typage PN reconnu par le programme est le typage $\mathbf{\Delta} \vdash \alpha$ où, pour tout $i \in \{1, \ldots, n\}$, l'occurrence de type atomique associée à la variable x_i est substituée par le type atomique $a_i \in \mathrm{AtPot}(\mathbf{\Delta} \vdash \alpha) \cup \{\omega\}$.

Lemme 7.3.2.1. *Si il existe un terme $N \in \Lambda_p(\Sigma_1)$ et un typage $\mathbf{\Delta} \vdash \alpha$ du terme $\mathcal{H}(N)$, alors*

$$\texttt{recog}(\mathcal{G}, \mathbf{\Delta}) \text{:-}\, p(\alpha)$$

Démonstration. Considérons un terme clos $N \in \Lambda_p(\Sigma_1)$, sous forme normale, et pour lequel $p \in A_1$. Puisque N est sous forme normale :

- $N = \mathbf{c}N_1 \dots N_m$
- $type_1(\mathbf{c}) = p_1 \to \dots \to p_m \to p$, où $p_1, \dots, p_m$ appartiennent à A_1.
- pour tout $i \in \{1, \dots, m\}, N_i \in \Lambda_{p_i}(\Sigma_1)$.

Nous procédons par induction sur N. Nous savons que le terme $M = \mathcal{H}(N) = \mathcal{H}(\mathbf{c})\mathcal{H}(N_1)\dots\mathcal{H}(N_m)$ appartient à $\Lambda_{\mathcal{H}(p)}(\Sigma_2)$. Soit un typage $\mathbf{\Delta} \vdash \alpha$ du terme M. Nous obtenons alors $\mathbf{\Delta} \vdash \mathcal{H}(\mathbf{c}) : \alpha_1 \to \dots \to \alpha_m \to \alpha$ et pour tout $i \in \{1, \dots, m\}$, $\mathbf{\Delta} \vdash \mathcal{H}(N_i) : \alpha_i$. D'après l'hypothèse d'induction, nous savons que $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $p_i(\alpha_i)$.

Nous considérons, à présent, la règle $\rho_{\mathbf{c}}$ associée à la constante $\mathbf{c}$ dans le reconnaisseur. Cette règle est de la forme générale suivante :

$$p(\overrightarrow{x}) \text{ :- } p_1(\overrightarrow{x_1}), \dots, p_m(\overrightarrow{x_m}), e_1(\overrightarrow{y_{11}}), \dots, e_1(\overrightarrow{y_{1k_1}}), \dots, e_n(\overrightarrow{y_{n1}}), \dots, e_n(\overrightarrow{y_{nk_n}}), \texttt{atome}(z_1), \dots, \texttt{atome}(z_l)$$

Par construction, ρ_c est associée au typage Σ-principal listé $\mathbf{\Gamma} \vdash \gamma$ de $\mathcal{H}(c)$; il existe donc un renommage σ tel que, pour toute constante $\mathbf{c}$ de $\mathrm{Dom}(\mathbf{\Gamma})$, $\mathbf{\Gamma}(\mathbf{c}) \subseteq \mathbf{\Delta}(\mathbf{c})$, et $\gamma \cdot \sigma = \alpha$. Ce renommage se traduit par l'instanciation de variables dans ρ_c, de telle sorte que :

$$p(\overrightarrow{\gamma}) \text{ :- } p_1(\overrightarrow{\gamma_1}), \dots, p_m(\overrightarrow{\gamma_m}), e_1(\overrightarrow{\delta_{11}}), \dots, e_1(\overrightarrow{\delta_{1k_1}}), \dots, e_n(\overrightarrow{\delta_{n1}}), \dots, e_n(\overrightarrow{\delta_{nk_n}}), \texttt{atome}(a_1), \dots, \texttt{atome}(a_l)$$

soit dérivable. Cette dernière affirmation est vérifiée car, d'après le renommage σ

$$p(\overrightarrow{\gamma}) \text{ :- } p_1(\overrightarrow{\gamma_1}), \dots, p_m(\overrightarrow{\gamma_m}), e_1(\overrightarrow{\delta_{11}}), \dots, e_1(\overrightarrow{\delta_{1k_1}}), \dots, e_n(\overrightarrow{\delta_{n1}}), \dots, e_n(\overrightarrow{\delta_{nk_n}}),$$

est dérivable, et de plus, pour tout $i \in \{1, \dots, l\}$, la variable z_i est susceptible de s'unifier avec une valeur a_i apparaissant dans $\mathbf{\Delta} \vdash \alpha$; la contraindre à s'unifier avec une telle valeur suffit à assurer la correction de la dérivation. □

Lemme 7.3.2.2. *Si il existe un typage $\mathbf{\Delta} \vdash \alpha$ et un type $p \in A_1$ tels que $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $p(\overrightarrow{\alpha})$, alors il existe un terme clos $N \in \Lambda_p(\Sigma_1)$ tel que $\mathbf{\Delta} \vdash \mathcal{H}(N) : \alpha$.*

Démonstration. Supposons que $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $p(\alpha)$. Par définition, il existe une dérivation Datalog en réponse à la requête ? :- $p(\overrightarrow{\alpha})$. Cette dérivation se représente sous la forme d'un arbre, dont la racine est de la forme $p(\overrightarrow{\alpha})$, et dont les feuilles sont, soit de la forme $e(\overrightarrow{\gamma})$, soit de la forme $\texttt{atome}(a)$ *i.e.* les feuilles sont des faits du programme $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$. Nous procédons par induction sur la hauteur de l'arbre de dérivation. Soit $\rho_{\mathbf{c}}$ la règle dont la tête apparaît en racine de l'arbre de dérivation. Les fils de la racine nous donne donc l'instanciation suivante de ρ_c :

$$p(\overrightarrow{\gamma}) \text{ :- } p_1(\overrightarrow{\gamma_1}), \dots, p_m(\overrightarrow{\gamma_m}), e_1(\overrightarrow{\delta_{11}}), \dots, e_1(\overrightarrow{\delta_{1k_1}}), \dots, e_n(\overrightarrow{\delta_{n1}}), \dots, e_n(\overrightarrow{\delta_{nk_n}}), \texttt{atome}(a_1), \dots, \texttt{atome}(a_l)$$

Ce qui nous amène aux affirmations suivantes :

1. pour tout $i \in \{1, \dots, m\}$, $p_i(\overrightarrow{\gamma_i})$ peut être dérivé par le reconnaisseur. Ainsi, d'après l'hypothèse d'induction, il existe un terme $N_i \in \Lambda_{p_i}(\Sigma_1)$ tel que $\mathbf{\Delta} \vdash \mathcal{H}(N_i) : \gamma_i$.
2. les types atomiques $a_1, \dots, a_l$ ont des occurrences dans $\mathbf{\Delta} \vdash \alpha$.
3. pour tout $i \in \{1, \dots, n\}$, et tout $j \in \{1, \dots, k_n\}$, $e_i(\overrightarrow{\delta_{ij}})$ est dérivable ; par construction du reconnaisseur, δ_{ij} appartient à $\mathbf{\Delta}(\mathbf{e}_i)$.

Enfin, puisque la règle $\rho_{\mathbf{c}}$ est construite à partir de $\mathbf{\Gamma} \vdash \beta$, le typage Σ_2-principal listé de $\mathcal{H}(\mathbf{c})$, considérons le typage :

$$\mathbf{e}_1 : [\tau(\mathbf{e}_1)]\overrightarrow{\delta_{11}} \cap \ldots \cap [\tau(\mathbf{e}_1)]\overrightarrow{\delta_{1k_1}}, \ldots, \mathbf{e}_n : [\tau(\mathbf{e}_n)]\overrightarrow{\delta_{n1}} \cap \ldots \cap [\tau(\mathbf{e}_n)]\overrightarrow{\delta_{nk_n}} \vdash \mathcal{H}(\mathbf{c}) : \gamma_1 \to \ldots \to \gamma_n \to \gamma$$

Ce typage est donc moins général que $\mathbf{\Gamma} \vdash \beta$ (au sens de la substitution de types donnée par l'instanciation de la règle) et moins général que $\mathbf{\Delta} \vdash \gamma_1 \to \ldots \to \gamma_n \to \gamma$ (au sens de l'inclusion pour chaque type listé assigné aux constantes de l'environnement de typage). Finalement, nous pouvons en déduire que $N = \mathbf{c}N_1 \ldots N_n$ appartient à $\Lambda_p(\Sigma_1)$ et $\mathcal{H}(N)$ peut être typé par $\mathbf{\Delta} \vdash \alpha$. □

Nous nous appuyons à présent sur ces deux lemmes afin de prouver que la construction Datalog que nous donnons pour l'ensemble des typages PN moins généraux du terme N en entrée est correcte et complète :

Théorème 7.3.2.1. *Considérons une ACG quasi-affine du second-ordre $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$, un terme quasi-affine $N \in \Lambda(\Sigma_2)$ sous forme normale et $\mathbf{\Delta} \vdash \alpha$ un typage PN moins général de N. Alors N appartient à $O(\mathcal{G})$ si et seulement si* $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $s(\overrightarrow{\alpha}(x_1, \ldots, x_n))$.

Démonstration. Dans un premier temps, considérons que le terme N appartient au langage objet de $\mathcal{G}$. Par définition, il existe un terme $N' \in \mathcal{A}(\mathcal{G})$ tel que $|\mathcal{H}(N')|_\beta = N$. Alors, pour tout typage $\mathbf{\Gamma} \vdash \gamma$ de $\mathcal{H}(N')$, nous avons $\texttt{recog}(\mathcal{G}, \Gamma)$:- $s(\gamma)$, d'après le lemme 7.3.2.1. Cette propriété est en particulier vraie pour tout typage PN moins général de $|\mathcal{H}(N')|_\beta$ habité par $\mathcal{H}(N)$ et dont l'existence est assurée par le Théorème 7.2.3.2. Il existe donc $a_1, \ldots, a_n \in \text{AtPot}(\mathbf{\Delta} \vdash \alpha) \cup \{\omega\}$ tels que $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $s(\overrightarrow{\alpha}(a_1, \ldots, a_n))$, et donc une dérivation en réponse à la requête $s(\overrightarrow{\alpha}(x_1, \ldots, x_n))$

Supposons, à présent, que $\texttt{recog}(\mathcal{G}, \mathbf{\Delta})$:- $s(\overrightarrow{\alpha}(a_1, \ldots, a_n))$, et posons $\alpha' = \alpha[\overrightarrow{\alpha}(a_1, \ldots, a_n)]$. Alors, d'après le lemme 7.3.2.2, il existe un terme $N' \in \mathcal{A}(\mathcal{G})$ tel que $\mathcal{H}(N')$ habite le typage $\mathbf{\Delta} \vdash \alpha'$. Or, ce dernier est un typage PN moins général de N, et est donc un typage PN. D'après le Théorème 7.2.2.2, N et $\mathcal{H}(N')$ sont β-équivalents. □

7.3.3 Exemples et discussion

Afin d'illustrer la construction mise au point lors de la section précédente, nous proposons deux exemples de grammaires quasi-affines et les programmes Datalog correspondants.

Sélection d'aspects sémantiques

À travers le premier exemple, nous considérons la grammaire $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ donnée en Figure 7.4, où nous modélisons un traitement partiel de l'introduction d'information de la sémantique lexicale dans les ACGs. Dans cette grammaire, nous nous intéressons aux dérivations de phrases telles que "*Jean lit Hamlet*", "*Jean lit un livre*" ou encore "*Jean lit un livre intéressant*". Les termes représentant les arbres syntaxiques de ces phrases sont respectivement les suivants : *lire jean hamlet*, *lire Jean* (*un livre*) et *lire jean* (*un* (*intéressant livre*)).

Remarque 7.3.3.1. Cette grammaire est prise comme exemple afin d'illustrer la construction de reconnaisseurs Datalog pour les ACGs quasi-affines. Nous ne cherchons pas, par cet exemple, à résoudre entièrement les problèmes liées à la sémantique lexicale. Ainsi, nous déléguons aux verbes la sélection de l'aspect sémantique des phrases nominales, en étant conscient que cette solution est incomplète. Par exemple, pour la phrase "*Jean lit un livre intéressant*", l'adjectif "*intéressant*"

- $\Sigma_1 = \begin{cases} \{jean, hamlet, lire, intéressant, livre, un\}, \\ \{s, n, np\}, \\ \{jean, hamlet \mapsto np, lire : np \to np \to s, intéressant \mapsto n \to n, livre \mapsto n, un \mapsto n \to np\} \end{cases}$
- $\Sigma_2 = \begin{cases} \{\mathbf{J_{pers}}, \mathbf{undef}, \mathbf{H_{pers}}, \mathbf{H}_\phi, \mathbf{H_{info}}, \ell_\phi, \ell_{\mathbf{info}}, \checkmark, \wedge, \exists, \mathbf{undef_{pers}}, \mathbf{lire}\} \\ \{e, t\} \\ \{\mathbf{J_{pers}}, \mathbf{undef}, \mathbf{H_{pers}}, \mathbf{H}_\phi, \mathbf{H_{info}} \mapsto e, \\ \quad \ell_\phi, \ell_{\mathbf{info}}, \mathbf{undef_{pers}}, \checkmark \mapsto e \to t, \\ \quad \wedge \mapsto t \to t \to t, \exists \mapsto (e \to t) \to t, \mathbf{lire} \mapsto e \to e \to t\} \end{cases}$
- $\mathcal{H} = \begin{cases} jean \mapsto \lambda QP.P(Q\ \mathbf{J_{pers}}\ \mathbf{undef}\ \mathbf{undef}), \\ hamlet \mapsto \lambda QP.P(Q\ \mathbf{H_{pers}}\ \mathbf{H}_\phi\ \mathbf{H_{info}}), \\ lire \mapsto \lambda PQ.P\pi_3(\lambda x.Q\pi_1(\lambda y.\mathbf{lire}\ y\ x)), \\ a \mapsto \lambda PRQ.\exists(\lambda x.\wedge(P\ R\ x)(Q\ x)), \\ livre \mapsto \lambda Px.P(\mathbf{undef_{pers}}\ x)(\ell_\phi\ x)(\ell_{\mathbf{info}}\ \mathbf{x}), \\ intéressant \mapsto \lambda PQx.\wedge(\checkmark\ x)(P\ Q\ x), \\ n \mapsto (t^3 \to t) \to e \to t, \\ np \mapsto (e^3 \to e) \to (e \to t) \to t, \\ s \mapsto t \end{cases}$

où, pour $i \in \{1,2,3\}$, π_i est le λ-terme $\lambda x_1 x_2 x_3.x_i$.

FIG. 7.4 – Exemple d'ACG non-contextuelle quasi-affine

$NP(x_1,x_2,x_3,x_4,x_4,x_5,x_5) :\text{-}\ J_P(x_1), undef(x_2), undef(x_3), \texttt{atome}(x_4), \texttt{atome}(x_5).$
$NP(x_1,x_2,x_3,x_4,x_4,x_5,x_5) :\text{-}\ H_P(x_1), H_\phi(x_2), H_{info}(x_3), \texttt{atome}(x_4), \texttt{atome}(x_5).$
$S(x) :\text{-}\ NP(x_1,x_2,x_3,x_3,x_4,y_5,x), NP(y_1,y_2,y_3,y_1,y_4,x_5,y_5), lire(y_4,x_4,x_5).$
$N(z_1,z_2,z_3,z_4,x_1,y) :\text{-}\ N(z_1,z_2,z_3,z_4,x_1,y_2), \wedge(y_1,y_2,y), \checkmark(x_1,y_1).$
$N(x_1,x_2,x_3,x_4,x_5,x_4) :\text{-}\ undef_{pers}(x_5,x_1), livre_\phi(x_5,x_2), livre_{info}(x_5,x_3), \texttt{atome}(x_4).$
$NP(y_1,y_2,y_3,y,x_1,x_5,x_3) :\text{-}\ N(y_1,y_2,y_3,y,x_1,x_4), \exists(x_1,x_2,x_3), \wedge(x_4,x_5,x_2).$

FIG. 7.5 – Exemple : les règles Datalog construites pour $\mathcal{G}$

et le verbe "*lire*" devrait chacun sélectionner un aspect du nom "*livre*", puis il nous faudrait vérifier qu'il n'existe pas de conflits sur les deux aspects sélectionnés. Il serait ainsi possible de définir $\mathcal{H}(intéressant) = \lambda PQx.\wedge(\checkmark\ x)(P\ \pi_3\ x)$, mais la sélection d'aspects par le verbe serait alors rendue inutile par effacement.

Les règles Datalog construites à partir de cette grammaire sont représentées en Figure 7.5. Ces règles, que nous appellerons *règles Datalog sur* $\mathcal{G}$, décrivent les mécanismes de composition réalisables d'après le typage des constantes de la signature abstraite Σ_1.

À présent, nous construisons les règles Datalog relatives aux typages PN moins généraux du terme N à reconnaître. Nous prenons pour exemple, le terme

$$N = \exists(\lambda x.\wedge(\wedge(\checkmark\ x)(\ell_{\mathbf{info}}\ x))(\mathbf{lire}\ \mathbf{J_p}\ x))$$

$J_p(a).$	$\texttt{atome}(a).$	$J_p(\omega).$
$lire(a,b,c).$	$\texttt{atome}(b).$	$undef(\omega).$
$livre_{info}(b,d).$	$\texttt{atome}(c).$	$H_P(\omega).$
$\checkmark(b,e).$	$\texttt{atome}(d).$	$H_\phi(\omega).$
$\wedge(e,d,f).$	$\texttt{atome}(e).$	$H_{info}(\omega).$
$\wedge(f,c,g).$	$\texttt{atome}(f).$	$lire(x_1,x_2,\omega)\texttt{:-atome}(x_1),\texttt{atome}(x_2).$
$\exists(b,g,h).$	$\texttt{atome}(g).$	$undef_p(x,\omega)\texttt{:-atome}(x).$
	$\texttt{atome}(h).$	$livre_\phi(x,\omega)\texttt{:-atome}(x).$
		$livre_{info}(x,\omega)\texttt{:-atome}(x).$
		$\exists(\omega,x,\omega)\texttt{:-atome}(x).$
		$\wedge(x_1,x_2,\omega)\texttt{:-atome}(x_1),\texttt{atome}(x_2).$
		$\checkmark(x,\omega)\texttt{:-atome}(x).$

FIG. 7.6 – Exemple : les règles Datalog construites pour le terme N

Ce terme appartient au langage objet de $\mathcal{G}$; en effet, il s'agit de la forme normale du terme

$$\mathcal{H}(lire\ \ jean\ (un\ (intéressant\ livre)))$$

Les règles Datalog associées à N sont représentées en Figure 7.6. Nous les regroupons sous trois familles, données dans les trois colonnes de la figure :

- la première correspond au typage principal listé du terme N :

  $$\exists : (b \to g) \to h, \wedge : e \to d \to f \cap f \to c \to g, \checkmark : b \to e, \ell_{\textbf{info}} : b \to d, \textbf{lire} : a \to b \to c, \mathbf{J_p} : a \vdash N : h$$

  Puisque ce terme est un terme quasi-linéaire, nous remarquerons l'absence de type ω. Nous obtenons donc des règles similaires à celles obtenues dans [Kanazawa, 2007].
- Le deuxième bloc contient les règles permettant d'assurer une gestion correcte des phénomènes d'effacement par β-réduction, en indiquant les types pouvant être assignés sur des O-coups injouables de l'arène associée à un typage PN moins général de N. Ces règles limitent donc les types atomiques (ou constantes Datalog) pouvant être utilisés au cours de la dérivation. Par ailleurs, et comme nous l'avons évoqué, ces prédicats `atome` permettent d'assurer la condition de sûreté sur le programme construit à partir des règles de la Figure 7.5.
- La troisième famille de règles est une liste exhaustive de typages possibles pour les constantes de la signature objet, au cas où ces constantes seraient effacées ou non-utilisées dans le dérivation du terme à reconnaître. Comme décrit dans la construction d'un tel programme Datalog, nous encodons l'ensemble des types pouvant être assignés à une constante dans un typage PN minimal de N grâce à ces règles.

Une règle appartenant à une des deux première familles de règles sera appelée *règle relative au typage principal listé de N*. Une règle appartenant à la dernière famille sera appelée *règle de complétion pour N*. Finalement, la requête associée à la reconnaissance d'un tel mot est :

$$?\texttt{:-}S(h)$$

Nous pouvons remarquer que, pour une grammaire donnée, les règles pouvant être construites sont les règles de la Figure 7.5 et les règles de complétion pour N. Ainsi, un tel reconnaisseur peut construire l'ensemble de ces règles lorsque la grammaire lui est donnée, le nombre de ces règles étant

- $\Sigma'_1 = \begin{cases} \{\mathbf{e}, \mathbf{e_1}, \mathbf{e_{21}}, \mathbf{e_{22}}, \mathbf{e_{23}}\}, \\ \{a, b\}, \\ \{\mathbf{e} \mapsto b \to a \to a, \mathbf{e_1} \mapsto b, \mathbf{e_{21}} \mapsto a, \mathbf{e_{22}} \mapsto a, \mathbf{e_{23}} \mapsto a\} \end{cases}$
- $\Sigma'_2 = \begin{cases} \{\mathbf{c}, \mathbf{c_1}, \mathbf{c_2}\}, \\ \{o\}, \\ \{\mathbf{c} \mapsto o \to o, \mathbf{c_1} : o \to o \to o, \mathbf{c_2} \mapsto o \to o \to o\} \end{cases}$
- $\mathcal{H}' = \begin{cases} \mathbf{e} \mapsto \lambda GHfxy.\mathbf{c}(Gxy(Hfxy)), \\ \mathbf{e_1} \mapsto \lambda xyz.\mathbf{c_1}xy, \\ \mathbf{e_{21}} \mapsto \lambda fxy.\mathbf{c_2}xy, \\ \mathbf{e_{22}} \mapsto \lambda fxy.\mathbf{c_2}xx, \\ \mathbf{e_{23}} \mapsto \lambda fxy.\mathbf{c_1}(fxx)y, \\ a \mapsto (o \to o) \to o \to o \to o, \\ b \mapsto o \to o \to o \to o \end{cases}$

FIG. 7.7 – Exemple d'ACG non-contextuelle quasi-affine

$n_1 + n_2$ où n_1 est le nombre de constantes de la signature abstraite, et n_2 le nombre de constantes de la signature objet.

De plus, pour une grammaire $\mathcal{G}$ et un terme N donnés, le nombre de règles effectivement associée au terme N à reconnaître peut être considérablement réduit, et correspond aux règles relatives au typage principal listé de N. Ce nombre est de taille linéaire par rapport au terme à reconnaître, puisque :

1. le nombre de types atomiques intervenant dans le typage est borné par le nombre d'occurrences de variables et de constantes apparaissant dans N (pour générer les faits dont `atome` est la tête).
2. le nombre de typages de constantes dans l'environnement du typage principal de N est également borné par le nombre d'occurrences de constantes dans N.

Le nombre de règles ainsi trouvée est linéaire, en supposant que nous pouvant insérer et supprimer des règles dans la base intentionnelle. En effet, notre construction donne une représentation compacte de typages listés assignés à une constante. Remarquons que ceci n'est pas nécessaire si le terme est quasi-linéaire (auquel nous avons donc l'assurance d'avoir un nombre de faits linéaire dans la taille du terme). Si il n'est pas possible d'insérer et de supprimer des règles de la BD intentionnelle, le nombre de faits à inscrire dans la base extensionnelle devient polynomial, puisqu'il faut "déplier" chacune des règles intentionnelle liées au typage du terme à reconnaître.

Effacement, ambiguïté et dérivations

À présent, nous considérons un deuxième exemple de grammaires mettant en lumière quelques problèmes liés à la construction de notre reconnaisseur, et plus particulièrement, au traitement de l'effacement dans les ACGs quasi-affines du second-ordre. Pour ce faire, nous prenons un deuxième exemple de grammaire $\mathcal{G}' = (\Sigma'_1, \Sigma'_2, \mathcal{H}', a)$ décrite en Figure 7.7.

Nous pouvons voir que le langage objet de cette grammaire est fini et est égal à :

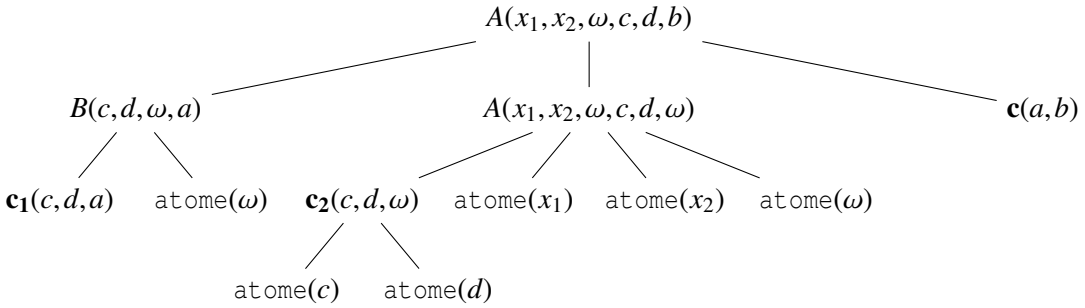
$$\{\lambda fxy.\mathbf{c_2}xy,\ \lambda fxy.\mathbf{c_2}xx,\ \lambda fxy.\mathbf{c_1}(fxx)y,\ \lambda fxy.\mathbf{c}(\mathbf{c_1}xy)\}$$

$A(y_1,y_2,y_3,x_1,x_2,x)$:- $B(x_1,x_2,y,z), A(y_1,y_2,y_3,x_1,x_2,y), \mathbf{c}(z,x)$.
$B(x_1,x_2,y,x)$:- $\mathbf{c_1}(x_1,x_2,x), \texttt{atome}(y)$.
$A(y_1,y_2,y_3,x_1,x_2,x)$:- $\mathbf{c_2}(x_1,x_2,x), \texttt{atome}(y_1), \texttt{atome}(y_1), \texttt{atome}(y_2), \texttt{atome}(y_3)$.
$A(y_1,y_2,y_3,x_1,x_2,x)$:- $\mathbf{c_2}(x_1,x_1,x), \texttt{atome}(y_1), \texttt{atome}(y_1), \texttt{atome}(y_2), \texttt{atome}(y_3), \texttt{atome}(x_2)$.
$A(x_1,x_1,y,x_1,x_2,x)$:- $\mathbf{c_1}(y,x_2,x)$.

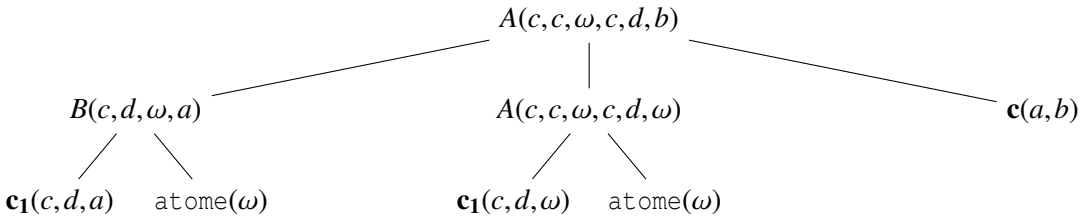
$\mathbf{c}(x,\omega)$	:- $\texttt{atome}(x)$	$\texttt{atome}(a)$.	$\mathbf{c}(a,b)$.
$\mathbf{c_1}(x_1,x_2,\omega)$	:- $\texttt{atome}(x_1), \texttt{atome}(x_2)$	$\texttt{atome}(b)$.	$\mathbf{c_1}(c,d,a)$.
$\mathbf{c_2}(x_1,x_2,\omega)$	:- $\texttt{atome}(x_1), \texttt{atome}(x_2)$	$\texttt{atome}(c)$.	
		$\texttt{atome}(d)$.	
		$\texttt{atome}(\omega)$.	

FIG. 7.8 – Exemple : programme Datalog pour $\mathcal{G}'$ et $\lambda fxy.\mathbf{c}(\mathbf{c_1}xy)$

Cependant, le langage abstrait de $\mathcal{G}'$ est infini, puisqu'il existe une infinité de dérivations possibles pour le terme $\lambda fxy.\mathbf{c}(\mathbf{c_1}xy)$. En effet, pour tout terme P de la forme $\mathbf{ee_1}(\mathbf{ee_1}(\ldots(\mathbf{ee_1e_{2i}})\ldots))$ (où $i \in \{1,\ldots,3\}$), nous obtenons $\mathcal{H}(P)\rightarrow^*_\beta \lambda fxy.\mathbf{c}(\mathbf{c_1}xy)$. Considérons cette grammaire et le terme $N = \lambda fxy.\mathbf{c}(\mathbf{c_1}xy)$ à reconnaître. Le programme Datalog correspondant est alors représenté en Figure 7.8. La requête Datalog associée à ce terme est donc $?\text{:-}A(x_1,x_2,\omega,c,d,b)$. Les résultats de la section précédente assurent que l'ensemble des dérivations Datalog répondant à cette requête sur le reconnaisseur que nous avons construit correspondent à une dérivation du terme N en entrée. De plus, nous obtenons toutes les dérivations possibles pour ce terme, dans le cas présent, une infinité de dérivations. Cependant, nous pouvons remarquer que, étant donné un terme P de $\mathcal{A}(\mathcal{G}')$ tel que $\mathcal{H}'(P)\rightarrow^*_\beta N$, il existe plusieurs dérivations Datalog correspondant au terme P ; en effet, pour le terme $P = (\mathbf{ee_1e_{21}})$, nous obtenons tous les arbres de dérivation Datalog de la forme suivante :



où x_1 et x_2 sont donc unifiés avec un des types atomiques de $\{a,b,c,d,\omega\}$ pour chacune des dérivations. Dans ce cas, nous remarquons que tous les typages PN moins généraux de N sont aussi des typages de $\mathcal{H}'(\mathbf{ee_1e_{21}})$. Tel n'est pas le cas pour le terme $\mathcal{H}'(\mathbf{ee_1e_{23}})$. En effet, la dérivation obtenue serait alors la suivante :



Le principe du reconnaisseur étant de s'assurer qu'il existe au moins une dérivation correspondant au terme N en entrée, il n'est pas essentiel de différencier les différentes dérivations ainsi obtenues.

Afin de conclure, il est intéressant de s'intéresser à la complexité d'un tel reconnaisseur. Nous considérons ici la complexité du problème de reconnaissance simple, c'est-à-dire en prenant uniquement le terme N comme paramètre du calcul de complexité. D'après la construction que nous avons donnée, connaître la complexité d'une requête $?\text{-}\ p(\overrightarrow{\alpha})$ sur un reconnaisseur $\texttt{recog}(\mathcal{G}, \mathbf{\Lambda})$ est équivalent à évaluer le complexité de cette requête sur un programme Datalog $(\mathbf{P}, D)$ où :

- $\mathbf{P}$ contient les règles relatives la grammaire $\mathcal{G}$ du problème, ainsi que les règles de complétion pour N.
- D contient les règles relatives au typage principal listé du terme N en entrée du problème.

Puisque nous avons construits un reconnaisseur Datalog correct et complet pour le problème de reconnaissance de N dans $\mathcal{G}$, un tel problème est résolu en temps polynomial par rapport à la base de données extensionnelle. Il est effectivement connu que l'évaluation ascendante d'une requête sur un programme Datalog s'effectue en temps polynomial relativement à la taille de la base de données. Malgré tout, nous aimerions obtenir un résultat plus fort, plaçant le problème de reconnaissance dans les ACGs quasi-affines dans la classe **LOGCFL** [Venkateswaran, 1987] des langages pouvant être réduits en espace polynomial à un langage hors-contexte. En effet, [Kanazawa, 2011] prouve ce résultat dans le cas des grammaires quasi-linéaires, et il serait donc intéressant de montrer que notre reconnaisseur à les mêmes propriétés de complexité que ce dernier.

Finalement, il est prouvé qu'une ACG $\mathcal{G}$ effaçante dont le langage objet est fait de termes quasi-linéaires, peut être transformée en une ACG quasi-linéaire $\mathcal{G}'$ dont la taille est exponentiellement plus grande que celle de $\mathcal{G}$ [Yoshinaka, 2006]. Il est donc possible de linéariser une grammaire quasi-affine puis d'exploiter les résultats de [Kanazawa, 2007]. De notre point de vue, préserver l'effacement dans une grammaire présente, tout d'abord, un avantage du point de vue de la modélisation de phénomènes linguistiques, alors que la linéarisation d'une telle grammaire rend plus opaque cette modélisation. De plus, notre construction construit un programme de taille linéaire comparativement à la taille de la grammaire en entrée, ce qui est un avantage sur une technique exploitant les résultats de Yoshinaka et Kanazawa. Il nous semble utile à renforcer la robustesse de notre technique, d'étudier plus en détails la complexité des deux méthodes évoquées.

7.4 Conclusion

Ce dernier chapitre nous a permis de présenter une extension des techniques de reconnaissance grammaticale des ACGs du second-ordre quasi-linéaires aux ACGs quasi-affines. Les techniques que nous employons sont effectivement similaires à celles de [Kanazawa, 2007], et étendent en fait ces dernières. Comme nous l'avons vu au cours des chapitres précédents, cette méthode repose essentiellement sur deux théorèmes de typages (le théorème de cohérence, et le théorème d'expansion du sujet) et sur la stabilité par composition de la famille de termes analyser. Or, les termes quasi-affines ne vérifient pas la propriété d'expansion du sujet. Nous avons néanmoins prouver un théorème moins fort, où sont caractérisés certains typages d'un terme normal quasi-affine M qui sont habités par tout terme quasi-affine M' tel que $M' \rightarrow_{\beta}^{*} M$ et M et M' sont construits sur la même signature d'ordre supérieur.

Il convient de remarquer que l'approche présentée dans ce document est uniquement basée sur des propriétés de typage. Pour ce faire, nous avons introduit le système de typage listé, qui est une restriction du système de typage intersection ; nous avons ensuite démontré que certains typages vérifient le théorème de cohérence dans ce système de typage. En particulier, nous avons introduit les typages PN, basés sur une notion d'équivalence entre occurrences de types atomiques dans un typage. Puis

nous avons établi un théorème similaire au théorème d'expansion du sujet dans le système de typage simple.

Finalement, nous avons montré comment exploiter ces résultats dans la construction de reconnaisseurs Datalog. La construction que nous donnons étend la construction de [Kanazawa, 2007]. Elle nous permet d'envisager certains résultat complémentaires, tels que l'utilisation de la méthode de réécriture de programmes Datalog par ensemble magiques afin d'obtenir des algorithmes de type Earley pour les ACGs non-contextuelles quasi-affines.

Chapitre 8

Conclusion

Au cours de ce travail de recherche, nous nous sommes donc intéressés à l'extension de techniques de reconnaissances pour les ACGs non-contextuelles de termes quasi-linéaires. Plus particulièrement, nous avons cherché à mettre en lumière des familles de termes pour lesquelles ce problème se résout en temps polynomial. Cette étude nécessite d'établir des résultats de λ-calcul simplement typé, et plus particulièrement le théorème de cohérence, qui stipule que certains typages sont uniquement habités. Nous avons donc cherché à caractériser de tels typages de manière syntaxique, *c.a.d.* à partir des propriétés des types polarisés les composant. À partir des résultats obtenus, nous avons utilisé la famille des termes quasi-affines, dont le typage principal est uniquement habité, afin d'étendre les techniques de reconnaissance données dans [Kanazawa, 2007] pour des grammaires catégorielles abstraites du second-ordre générant des termes quasi-affines. Nous prouvons la polynomialité en temps de l'algorithme de reconnaissance grâce à la construction d'un reconnaisseur Datalog pour de telles grammaires. Cette construction se base sur deux propriétés supplémentaires au théorème de cohérence : la propriété de stabilité par composition, stipulant que la composition de deux termes quasi-affines forme un terme quasi-affine, et la propriété d'expansion du sujet. Cette dernière n'étant pas vérifiée dans le cas des termes quasi-affines, nous avons démontré un théorème plus faible, mais permettant la construction du reconnaisseur Datalog.

La complexité exacte de cet algorithme reste à être déterminée, en particulier, le fait de savoir si il se situe dans la classe **LOGCFL**, à l'image de l'algorithme donné dans [Kanazawa, 2007], pour les λ-grammaires non-contextuelles de termes quasi-linéaires. Néanmoins, le fait d'être construit comme un programme Datalog assure que le problème de la reconnaissance de termes quasi-affine se résout en temps polynomial en fonction de son typage principal (en réalité, de la taille de la base de données extensionnelle du programme Datalog associé) et donne donc lieu à des algorithmes de reconnaissance efficaces. Sachant que les ACGs permettent de modéliser l'interface entre syntaxe et sémantique de manière tout à fait uniforme, les méthodes de reconnaissance de termes dans ces grammaires peuvent être utilisées aussi bien pour résoudre des problèmes d'analyse du langage que de génération de texte. Ainsi, par notre méthode, nous obtenons des générateurs de texte à partir de représentation sémantiques à la Montague plus puissants que ceux donnés par [Merenciano and Morrill, 1997, Pogodalla, 2000] dans le cadre du calcul de Lambek, et ceux de [Salvati, 2005, Kanazawa, 2007], dans le cadre des ACGs du second-ordre. En particulier, nous avons vu que l'ajout de l'effacement dans les grammaires permet de modéliser un certain polymorphisme lié aux entrées lexicales, soit dans la morpho-syntaxe de ces entrées, soit dans la gestion de la sémantique lexicale. Dans le cas de

la sémantique lexicale, de telles modélisations devraient être étudiées avec plus de détails, les modèles que nous avons présentés ne faisant office que de solutions partielles à ces problèmes. Un axe d'étude future consisterait également à rechercher d'autres phénomènes linguistiques éventuellement modélisés par l'effacement. Remarquons que des phénomènes tels que l'ellipse ne semblent pas pouvoir être considérés ici, puisque, dans le cadre des grammaires catégorielles abstraites, une modélisation par copie sur les λ-termes semblent plus appropriée pour traiter de telles structures.

D'un point de vue algorithme, la construction que nous donnons pour les reconnaisseurs d'ACGs non-contextuelles quasi-affines est donc une extension directe de la construction de reconnaisseurs donnée dans [Kanazawa, 2007]. Néanmoins, il convient de remarquer que, l'introduction d'un nouveau prédicat `atom` nécessaire à la gestion de l'effacement permet d'assurer la condition de sécurité sur le programme Datalog construit, contrairement à la construction de Kanazawa. Ceci nous assure la terminaison du programme pour une évaluation ascendante. Par ailleurs, notre construction ne donne, pour le moment que des algorithmes de style CYK. La réécriture de programmes Datalog par la méthode des ensembles magiques supplémentaires permettant ainsi d'écrire des analyseurs à la Earley devra donc être étudiée à l'avenir. Il convient de remarquer que cette procédure de réécriture ne permet cependant pas, dans le cas général, d'assurer la correction par préfixe de l'analyse. Bien que [Kanazawa, 2008] donne une méthode permettant d'obtenir cette propriété pour des reconnaisseurs de grammaires non-contextuelles multiples, nous envisageons d'étudier une méthode plus générale et applicable à tout reconnaisseur de grammaires catégorielles abstraites non-contextuelles.

Nous avons également relevé que, dans le cas des reconnaisseurs Datalog d'une λ-grammaire non-contextuelles quasi-affine $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{L}, s)$, la reconnaissance d'un terme M peut amener à la construction de plusieurs dérivations équivalentes, dans le sens où celles-ci dénotent toutes un même terme $N \in \mathcal{A}(\mathcal{G})$ tel que $\mathcal{H}(N) \rightarrow_\beta^* M$. Ce problème est dû à l'usage de l'effacement et à son impact sur les propriétés de typage que nous étudions, et ne semble pas pouvoir être évité. Néanmoins, afin de palier partiellement à ce problème, il convient d'envisager la construction d'un analyseur, et non plus d'un reconnaisseur, par l'intermédiaire d"une méthode similaire, utilisant les mêmes propriétés de typage et la programmation logique, mais où la structure dérivationnelle associée à un terme analysé serait construite pendant la reconnaissance du terme. Ceci nous permettrait d'identifier dérivations et termes. Une manière élégante de construire ces analyseurs serait d'inclure la construction de la dérivation dans les règles du reconnaisseur logique. Pour ce faire, il nous faudrait certainement utiliser le langage Prolog, afin de construire ainsi les arbres de dérivation comme des termes fonctionnels, ce que le langage Datalog ne permet pas. Une technique similaire est utilisée dans [Pereira and Shieber, 1987] dans le cas des grammaires de clauses définies. Par ailleurs, nous nous proposons de construire une méthode qui soit adaptable automatiquement par réécriture de programmes, et en particulier pour la méthode de réécriture par ensemble magiques supplémentaires.

Remarquons, également, que nous avons introduit l'utilisation d'une restriction des types intersection dans l'étude des propriétés de cohérence étudiées. De tels types pourraient être utilisés, non pas pour étudier les typages associés aux termes analysés, mais directement dans les grammaires catégorielles abstraites. En effet, sur les termes de la tectogrammaire, nous pourrions ainsi exprimer le polymorphisme grammatical de certaines entrées lexicales. Par exemple, il est possible d'assigner une grande variété de catégories grammaticales à la conjonction "et'. Cette conjonction est de type $NP \rightarrow NP \rightarrow NP$ dans la phrase "Jean et Marie mangent"; $VP \rightarrow VP \rightarrow VP$ dans la phrase "Jean mange du boeuf et boit du rouge"; $S \rightarrow ((NP \rightarrow NP \rightarrow S) \rightarrow S) \rightarrow S$ dans "Jean mange du boeuf et Marie, du saumon". Étendre la notion de signatures d'ordre supérieur des types simples aux types listés semble être une possibilité pour prendre en compte ces problèmes, tout en préservant un système de typage proche du système de typage simple.

Par ailleurs, une grande partie de cette thèse a été consacrée à l'étude de propriété de typage dans le λ-calcul simplement typé. Nous avons ainsi trouvé une nouvelle famille de termes qui vérifie le théorèmes de cohérence : la famille des termes quasi-affines. Cette famille de termes correspond exactement à la famille des typages négativement non-dupliquants, étendant les résultats précédents de [Belnap, 1976, Hirokawa, 1991] entre termes affines et typages balancés. Par ailleurs, nous avons montré qu'il existe une plus grande famille de typages uniquement habités et caractérisables syntaxiquement : la famille des typages négativement-séparants. Cependant, il nous reste à trouver la famille des λ-termes qui ont pour typages principaux de tels typages. Afin d'approcher ce résultat, nous avons défini les termes contextuellement déterministes. La famille de termes recherchée se trouve être une sous-famille de la famille des termes contextuellement déterministes. Par ailleurs, bien que tout typage uniquement habité le soit par un terme contextuellement déterministe, l'inverse se trouve être faux. Il nous reste donc à savoir si il est possible de caractériser de manière syntaxique les typages qui sont exactement les typages uniquement habités, et les λ-termes qui sont exactement ceux qui sont les uniques habitants de leurs typages principaux. Une caractérisation de ces termes semblent être accessible, et nous avons remarquer que le problème général de déterminer si un typage, principal pour un terme N, est uniquement habité par N, semble être un problème **PSPACE**-complet, résultat que nous souhaitons prouver.

Afin de pouvoir utiliser les techniques d'analyse définies par [Salvati, 2005, Kanazawa, 2007] et présentées dans ces travaux pour les grammaires de termes quasi-affines, aux termes contextuellement déterministes ou aux termes dont le typage principal est négativement séparant , il nous faudrait montrer le résultat supplémentaire indiquant que les termes appartenant à la clôture par composition de ces familles de termes vérifient la propriété de cohérence. Enfin, se pose la question de la pertinence de ces termes afin de modéliser de phénomènes linguistiques non traités par les ACGs non-contextuelles quasi-affines. En effet, bien qu'un tel résultat nous permettrait de prendre en compte certaines formes de copies d'ordre supérieur, les phénomènes faisant appel à de telles copies dans le langage naturel ne semblent pas pouvoir être modélisés par des ACGs non-contextuelles de termes contextuellement déterministes, par exemple. Ainsi, si nous considérons la conjonction dans la phrase "Pierre et Jean mangent", la forme sémantique de cette phrase serait représentée par le λ-terme $\wedge(\mathbf{mange}\ \mathbf{Pierre})(\mathbf{mange}\ \mathbf{Jean})$, dont le typage principal est $\mathbf{Pierre} : a, \mathbf{Jean} : a, \mathbf{mange} : a \to b, \wedge : b \to b \to c \vdash c$. Or, il apparaît clairement que ce typage n'est pas uniquement habité, et n'est pas un terme contextuellement déterministe. Un tel exemple met en lumière les limites de la technique d'analyse basée sur la propriété de cohérence, et nous pousse à explorer de nouvelles techniques d'analyse.

Finalement, les théorèmes de cohérence que nous avons donnés complètent les résultats prouvés par [Babaev and Soloviev, 1982, Aoto, 1999, Hirokawa, 1991]. Il convient de noter que notre méthode se distingue de celles utilisées dans les résultats précédemment cités par l'utilisation de la sémantique des jeux afin de prouver ces résultats. De fait, cette méthode semble permettre une meilleure compréhension du problème de cohérence, car elle est définie à partir des informations sur les polarités des occurrences de types atomiques d'un typage. Remarquons que la sémantique des jeux fut initialement formulée sur les catégories cartésiennes closes, et donc, sur un modèle du λ-calcul. Qui plus est, la famille des termes quasi-affines semble avoir été définie dans [Gaboardi and Piccolo, 2009], pour exprimer une forme de linéarité sur ces modèles, et il serait intéressant de faire un lien entre le résultat présenté dans ce document et ce dernier. Par ailleurs, nous pouvons contraster les résultats trouvés avec ceux qui ont été exposés dans [Broda and Damas, 2005], où une famille de termes plus grande que la famille des termes quasi-affines et ayant la propriété de cohérence a été définie. Ces termes habitent des typages appelés typages déterministes. Néanmoins, ils ne sont pas caractérisés de manière syntaxique, à travers la notion de polarité, mais grâce à un algorithme linéaire. Qui plus est, il semble que la famille des typages négativement séparants soit très proche de celle des typages

déterministes. En effet, nous pensons que les typages dont les occurrences potentielles négatives d'un même type atomique sont séparées par une occurrence potentielle négative de type atomique sont exactement les typages déterministes ; ce résultat reste cependant à prouver, mais montrerait que les termes contextuellement déterministes formeraient la plus grande famille de termes, caractérisée de manière syntaxique et vérifiant la propriété cohérence. Enfin, la caractérisation de la principalité par des stratégies sur des arènes de typage semble pouvoir être étendue à des compositions de stratégies, c'est-à-dire aux typages principaux de termes qui ne sont pas sous forme normale. Si un tel résultat pouvait être prouvé, un problème tel que le problème du typage principal inverse [Hindley, 1997] (*c.a.d* tout typage est le typage principal d'un certain terme) pourrait être prouvé d'une manière directe et nouvelle.

Bibliographie

[Abramsky et al., 2000] Abramsky, S., Jagadeesan, R., and Malacaria, P. (2000). Full abstraction for PCF. *Inf. Comput.*, 163(2) :409–470.

[Abramsky and McCusker, 1999] Abramsky, S. and McCusker, G. (1999). Game semantics. In Schwichtenberg, H. and Berger, U., editors, *Computational Logic : Proceedings of the 1997 Marktoberdorf Summer School*, pages 1–56. Springer-Verlag.

[Adjukiewicz, 1935] Adjukiewicz, K. (1935). Die syntaktische konnexikat. *Studia Philosophica*, 1 :1–27.

[Aoto, 1999] Aoto, T. (1999). Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information*, 8 :217–242.

[Babaev and Soloviev, 1982] Babaev, A. and Soloviev, S. (1982). A coherence theorem for canonical morphism in cartesian closed categories. *Journal of Soviet Mathematics*, 20 :2263 – 2279.

[Bar-Hillel, 1950] Bar-Hillel, Y. (1950). On syntactical categories. *Journal of Symbolic Logic*, 15 :1–16.

[Barendregt, 1984] Barendregt, H. (1984). *λ-calculus : its syntax and semantics*. Elsevier Science Publishers Ltd.

[Barendregt, 1993] Barendregt, H. (1993). Lambda calculi with types, handbook of logic in computer science (vol. 2) : background : computational structures.

[Bassac et al., 2010] Bassac, C., Mery, B., and Retoré, C. (2010). Towards a type-theoretical account of lexical semantics. *Journal of Logic, Language and Information*, 19(2) :229–245.

[Beeri and Ramakrishnan, 1991] Beeri, C. and Ramakrishnan, R. (1991). On the power of magic. *The journal of logic programming*, 10(3-4) :255–299.

[Belnap, 1976] Belnap, N. (1976). The two-property. *Relevance Logic Newsletter*, pages 173–180.

[Blass, 1992] Blass, A. (1992). A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56 :183–220.

[Bourreau and Salvati, 2011a] Bourreau, P. and Salvati, S. (2011a). A Datalog recognizer for almost affine λ-CFGs. In Kanazawa, M., Kornai, A., Kracht, M., and Seki, H., editors, *MOL*, volume 6878 of *Lecture Notes in Artificial Intelligence*, pages 21–38. Springer.

[Bourreau and Salvati, 2011b] Bourreau, P. and Salvati, S. (2011b). Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus. In Ong, L., editor, *TLCA*, volume 6690 of *Lecture Notes in Computer Science*, pages 61–75. Springer.

[Broda and Damas, 2005] Broda, S. and Damas, L. (2005). On long normal inhabitants of a type. *Journal of Logic and Computation*, 15(3) :353–390.

[Ceri et al., 1989] Ceri, S., Gottlob, G., and Tanca, L. (1989). What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. on Knowl. and Data Eng.*, 1 :146–166.

[Chandra and Stockmeyer, 1976] Chandra, A. K. and Stockmeyer, L. J. (1976). Alternation. *Foundations of Computer Science, Annual IEEE Symposium on*, 0 :98–108.

[Chomsky, 1956] Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2 :113–124.

[Church, 1936] Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2) :345–363.

[Church, 1941] Church, A. (1941). *The Calculi of Lambda Conversion.* Princeton University Press, Princeton, NJ, USA.

[Cocke and Schwartz, 1970] Cocke, J. and Schwartz, J. (1970). *Programming languages and their compilers : Preliminary notes*. Courant Institute of Mathematical Sciences, New York University, New York.

[Colmerauer, 1978] Colmerauer, A. (1978). Metamorphosis grammars. *Natural language communication with computers*, pages 133–188.

[Colmerauer and Roussel, 1996] Colmerauer, A. and Roussel, P. (1996). The birth of prolog. In *History of programming languages—II*, pages 331–367. ACM.

[Coppo and Dezani-Ciancaglini, 1980] Coppo, M. and Dezani-Ciancaglini, M. (1980). An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4) :685–693.

[Curry, 1961] Curry, H. B. (1961). Some Logical Aspects of Grammatical Structure. In Jakobson, R. O., editor, *Structure of Language and its Mathematical Aspects, volume 12 of Symposia on Applied Mathematics*, pages 56–68. American Mathematical Society, Providence.

[Damas and Milner, 1982] Damas, L. and Milner, R. (1982). Principal type-schemes for functional programs. In *POPL '82 : Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 207–212, New York, NY, USA. ACM.

[Danos and Regnier, 1989] Danos, V. and Regnier, L. (1989). The structure of multiplicatives. *Arch. Math. Logic*, 28 :181–203.

[de Groote, 2000] de Groote, P. (2000). Linear higher-order matching is np-complete. In Bachmair, L., editor, *RTA*, volume 1833 of *Lecture Notes in Computer Science*, pages 127–140. Springer.

[de Groote, 2001] de Groote, P. (2001). Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

[de Groote, 2002] de Groote, P. (2002). Tree-adjoining grammar as abstract categorial grammar. In *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia.

[de Groote, 2006] de Groote, P. (2006). Towards a montagovian account of dynamics. In *Proceedings of Semantics and Linguistic Theory XVI*.

[de Groote and Lebedeva, 2010] de Groote, P. and Lebedeva, E. (2010). Presupposition accommodation as exception handling. In Fernández, R., Katagiri, Y., Komatani, K., Lemon, O., and Nakano, M., editors, *SIGDIAL Conference*, pages 71–74. The Association for Computer Linguistics.

[de Groote et al., 2007] de Groote, P., Maarek, S., and Yoshinaka, R. (2007). On two extensions of abstract categorial grammars. In Dershowitz, N. and Voronkov, A., editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 273–287. Springer.

[de Groote and Pogodalla, 2004] de Groote, P. and Pogodalla, S. (2004). On the expressive power of abstract categorial grammars : Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4) :421–438.

[de Groote and Retoré, 1996] de Groote, P. and Retoré, C. (1996). On the semantic readings of proof nets. In *Proc. of Formal Grammar*, pages 57–70.

[de la Clergerie, 2002] de la Clergerie, É. V. (2002). Construire des analyseurs avec DyALog. In *Proceedings of TALN*, volume 2.

[de La Clergerie, 2002] de La Clergerie, É. V. (2002). Parsing mildly context-sensitive languages with thread automata. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

[de La Clergerie, 2005] de La Clergerie, E. V. (2005). DyALog : a tabular logic programming based environment for NLP. In *Proceedings of 2nd International Workshop on Constraint Solving and Language Processing (CSLP'05)*, Barcelona, Spain.

[Dezani-Ciancaglini et al., 1998] Dezani-Ciancaglini, M., Giovannetti, E., and de'Liguoro, U. (1998). Intersection types, lambda-models and Böhm trees. In *MSJ-Memoir Vol. 2 "Theories of Types and Proofs"*, volume 2, pages 45–97. Mathematical Society of Japan.

[Earley, 1970] Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2) :94–102.

[Engelfriet and Heyker, 1991] Engelfriet, J. and Heyker, L. (1991). The string generating power of context-free hypergraph grammars. *J. Comput. Syst. Sci.*, 43(2) :328–360.

[Felscher, 1986] Felscher, W. (1986). Dialogues as a foundation for intuitionistic logic. In M.Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic*, volume 3. Kluwer Academic Publishers.

[Gaboardi and Piccolo, 2009] Gaboardi, M. and Piccolo, M. (2009). Categorical models for a semantically linear lambda-calculus. In Florido, M. and Mackie, I., editors, *LINEARITY*, volume 22 of *EPTCS*, pages 1–13.

[Girard, 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50(1) :1–102.

[Girard, 1989] Girard, J.-Y. (1989). *Proofs and Types*. CUP, Cambridge.

[Girard, 1994] Girard, J.-Y. (1994). Geometry of interaction (abstract). In Jonsson, B. and Parrow, J., editors, *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, page 1. Springer.

[Girard, 1995] Girard, J.-Y. (1995). Linear logic : its syntax and semantics. In *Advances in Linear Logic*, pages 1–42. Cambridge University Press.

[Girard, 2001] Girard, J.-Y. (2001). Locus solum : From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3) :301–506.

[Groenendijk and Stokhof, 1991] Groenendijk, J. and Stokhof, M. (1991). Dynamic predicate logic. *Linguistics and Philososphy*, 14 :39–100.

[Harrison, 1978] Harrison, M. A. (1978). *Introduction to Formal Language Theory*. Addison Wesley, Reading, MA.

[Hindley, 1997] Hindley, R. J. (1997). *Basic Simple Type Theory*. Cambridge Press University.

[Hirokawa, 1991] Hirokawa, S. (1991). Balanced formulas, minimal formulas and their proofs. Technical report, Research Institute of Fundamental Information Science, Kyochu University.

[Huet, 1976] Huet, G. (1976). *Résolution d'équations dans les langages d'ordre 1,2,...,ω*. PhD thesis, Université Paris 7.

[Hughes, 2000] Hughes, D. (2000). *Hypergame Semantics : Full Completeness for System F*. PhD thesis, Oxford University.

[Huybregts, 1984] Huybregts, R. (1984). The weak inadequacy of context-free phrase structure grammars. *Van Preferie naar Kern*, pages 81–90.

[Hyland and Ong, 2000] Hyland, M. and Ong, L. (2000). On full abstraction for PCF. In *Information and Computation*, volume 163 of *2*, chapter 2, pages 285–408. Elsevier Science B.V.

[Joshi, 1985] Joshi, A. K. (1985). Tree-adjoining grammars : How much context-sensitivity is required to provide reasonable strucutral descriptions ? *Natural Language Parsing : Psychological, Computational and Theoretical Perspectives*, pages 206–250.

[Joshi et al., 1975] Joshi, A. K., Levy, L. S., and Takahashi, M. (1975). Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1) :136–163.

[Joshi and Schabes, 1997] Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. *Handbook of Formal Languages*, pages 69–123.

[Kamp, 1981] Kamp, H. (1981). A theory of truth and semantic representation. In *Formal Semantics - the Essential Readings*, pages 189–222. Blackwell.

[Kanazawa, 2007] Kanazawa, M. (2007). Parsing and generation as Datalog queries. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 176–183, Prague. Association for Computational Linguistics.

[Kanazawa, 2008] Kanazawa, M. (2008). A prefix-correct Earley recognizer for multiple context-free grammars. In *TAG+9, Proceedings of the ninth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Tubingen, Germany.

[Kanazawa, 2009] Kanazawa, M. (2009). The pumping lemma for well-nested multiple context-free languages. In Diekert, V. and Nowotka, D., editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 312–325. Springer.

[Kanazawa, 2010a] Kanazawa, M. (2010a). http://research.nii.ac.jp/~kanazawa/Courses/2010/MathLing/index.html. Lecture notes on Mathematical linguistics.

[Kanazawa, 2010b] Kanazawa, M. (2010b). Second-order abstract categorial grammars as hyperedge replacement grammars. *Journal of Logic, Language and Information*, 19(2) :137–161.

[Kanazawa, 2011] Kanazawa, M. (2011). Parsing and generation as Datalog query evaluation. To appear.

[Kanazawa and Salvati, 2010] Kanazawa, M. and Salvati, S. (2010). The copying power of well-nested multiple context-free grammars. In Dediu, A. H., Fernau, H., and Martín-Vide, C., editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 344–355. Springer.

[Kasami, 1965] Kasami, T. (1965). An efficient recognition and syntaxanalysis algorithm for context-free languages. Technical report, DTIC Document.

[Ker et al., 2002] Ker, A. D., Nickau, H., and Ong, L. (2002). Innocent game models of untyped λ-calculus. *Theoretical Computer Science*, 272(1-2) :247 – 292.

[Lamarche, 1994] Lamarche, F. (1994). Proof nets for intuitionistic linear logic I : Essential nets. Technical report, Imperial College, London.

[Lambek, 1958] Lambek, J. (1958). The mathematics of sentence structure. *Amer. Math. Mon.*, 65 :154–170.

[Lascarides and Asher, 2001] Lascarides, A. and Asher, N. (2001). Segmented discourse representation theory : Dynamic semantics with discourse structure. *COMPUTING MEANING*, 3.

[Laurent, 2004] Laurent, O. (2004). Sémantique des jeux. notes de cours.

[Lorenz, 1968] Lorenz, K. (1968). Dialogspiele als semantische grundlage von logikkalkiilen. *Arch. Math. Logik Grundlag*, 11 :32–55.

[Lorenzen, 1959] Lorenzen, P. (1959). Ein dialogisches konstruktivitatskriterium. *Infinitistic Methods*, pages 193–200.

[Merenciano and Morrill, 1997] Merenciano, J. M. and Morrill, G. (1997). Generation as deduction on labelled proof nets. In *LACL '96 : Selected papers from the First International Conference on Logical Aspects of Computational Linguistics*, pages 310–328, London, UK. Springer-Verlag.

[Michaelis, 1998] Michaelis, J. (1998). Derivational minimalism is mildly context-sensitive. In Moortgat, M., editor, *LACL*, volume 2014 of *Lecture Notes in Computer Science*, pages 179–198. Springer.

[Michaelis, 2001] Michaelis, J. (2001). Transforming linear context-free rewriting systems into minimalist grammars. In de Groote, P., Morrill, G., and Retoré, C., editors, *LACL*, volume 2099 of *Lecture Notes in Computer Science*, pages 228–244. Springer.

[Mints, 1979] Mints, G. E. (1979). A coherence theorem for cartesian closed categories (abstract). *Journal of Symbolic Logic*, 44(3) :453–479.

[Montague, 1970a] Montague, R. (1970a). English as a formal language. *Linguaggi nella società e nella tecnica*, pages 189–223.

[Montague, 1970b] Montague, R. (1970b). Universal grammar. *Theoria*, 36 :373–398.

[Montague, 1973] Montague, R. (1973). The proper treatment of quantification in ordinary English. *Approaches to Natural Language*, pages 221–242.

[Moortgat, 1996] Moortgat, M. (1996). Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4) :349–385.

[Moortgat, 1997] Moortgat, M. (1997). Categorial type logics. In Benthem, J. V. and Meulen, A. T., editors, *Handbook of Logic and Language*. Elsevier Science B.V.

[Moot, 1996] Moot, R. (1996). *Proof Nets and Labeling for Categorial Grammar Logics*. PhD thesis, Utrecht University.

[Morrill et al., 2007] Morrill, G., Fadda, M., and Valentin, O. (2007). Nondeterministic discontinuous lambek calculus. In *Proceedings of the Seventh International Workshop on Compu- tational Semantics, IWCS-7*, pages 129–141.

[Morrill et al., 2010] Morrill, G., Fadda, M., and Valentin, O. (2010). Generalized discontinuity. In *Proceedings of Formal Grammar 2010*.

[Morrill and Merenciano, 1996] Morrill, G. and Merenciano, J.-M. (1996). Generalising discontinuity. In *Traitement automatique des langues*, pages 119–143.

[Morrill and Solias, 1993] Morrill, G. and Solias, T. (1993). Tuples, discontinuity, and gapping in categorial grammar. In *EACL*, pages 287–296.

[Muskens, 2001] Muskens, R. (2001). Lambda Grammars and the Syntax-Semantics Interface. In van Rooy, R. and Stokhof, M., editors, *Proceedings of the Thirteenth Amsterdam Colloquium*, pages 150–155, Amsterdam.

[Muskens, 2006] Muskens, R. (2006). Tectogrammatics, phenogrammatics and the architecture of the grammar.

[Nederhof, 1999] Nederhof, M. J. (1999). The computational complexity of the correct-prefix property for tags. *Computational Linguistics*, 25(3) :345–360.

[Nederhof and Satta, 2011] Nederhof, M.-J. and Satta, G. (2011). Theory of parsing. To appear in : Handbook of Computational Linguistics and Natural Language Processing.

[Nickau, 1994] Nickau, H. (1994). Hereditarily sequential functionals. In Nerode, A. and Matiyasevich, Y., editors, *LFCS*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison Wesley.

[Parikh, 1966] Parikh, R. J. (1966). On context-free languages. *Journal of the Association for Computing Machinery*, 13(4) :570–581.

[Pereira and Shieber, 1987] Pereira, F. C. and Shieber, S. M. (1987). *Prolog and Natural-Language Analysis*, volume 10 of *CSLI Lecture Notes Series*. Center for the Study of Language and Information. Italian translation : *Prolog e Analisi del Linguaggio Naturale*, Tecniche Nuove, Milan, 1992.

[Pereira and Warren, 1983] Pereira, F. C. and Warren, D. H. (1983). Parsing as deduction. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 137–144, Morristown, NJ, USA. Association for Computational Linguistics.

[Pogodalla, 2000] Pogodalla, S. (2000). Generation, Lambek calculus, Montague's semantics and semantic proof nets. In *proceedings of the International Conference on Computational Linguistics*.

[Pogodalla, 2004] Pogodalla, S. (2004). 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.

[Pogodalla and Pompigne, 2011] Pogodalla, S. and Pompigne, F. (2011). Controlling extraction in abstract categorial grammars. In *Proceedings of the* 15^{th} *Conference on Formal Grammar*.

[Pollard, 1984] Pollard, C. (1984). *Generalized Phrase Structure Grammars, Head Grammars*. PhD thesis, Stanford University.

[Pustejovsky, 1991] Pustejovsky, J. (1991). The generative lexicon. *Comput. Linguist.*, 17 :409–441.

[Ranta, 2004] Ranta, A. (2004). Grammatical framework. *Journal of Functional Programming*, 14(2) :145–189.

[Salvati, 2005] Salvati, S. (2005). *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*. PhD thesis, Institut National Polytechnique de Lorraine.

[Salvati, 2006] Salvati, S. (2006). Encoding second-order ACGs with deterministic tree walking transducers. In *Proceedings of Formal Grammar*, Malaga, Spain.

[Salvati, 2007] Salvati, S. (2007). On the membership problem for non-linear abstract categorial grammars. In Muskens, R., editor, *Proceedings of the Workshop on New Directions in Type-theoretic Grammars*, pages 43–50, Dublin, Ireland. Foundation of Logic, Language and Information (FoLLI).

[Salvati, 2010] Salvati, S. (2010). On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2) :163–183.

[Salvati, 2011] Salvati, S. (2011). MIX is a 2-MCFL. To be published.

[Salvati and de Groote, 2003] Salvati, S. and de Groote, P. (2003). On the complexity of higher-order matching in the linear λ-calculus. In Nieuwenhuis, R., editor, *International Conference on Rewriting Techniques and Applications - RTA'2003, Valencia, Spain*, volume 2706 of *Lecture notes in Computer Science*, pages 234–245.

[Seki et al., 1991] Seki, H., Matsamura, T., Mamoru, F., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88(2) :191–229.

[Seldin, 1968] Seldin, J. (1968). *Studies in Illative Combinatory Logic*. PhD thesis, University of Amsterdam, Netherlands.

[Sipser, 1996] Sipser, M. (1996). *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition.

[Sørensen and Urzyczyn, 2006] Sørensen, M. H. and Urzyczyn, P. (2006). *Lectures on the Curry-Howard isomorphism*. Elsevier Science.

[Statman, 1979] Statman, R. (1979). Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9 :67–72.

[Sterling and Shapiro, 1986] Sterling, L. and Shapiro, E. (1986). *The art of Prolog*, volume 1991. Wiley Online Library.

[Turing, 1950] Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59 :443–460.

[Ullman, 1988a] Ullman, J. D. (1988a). *Principles of Database and Knowledge-Base Systems. Volume I*. Computer Science Press.

[Ullman, 1988b] Ullman, J. D. (1988b). *Principles of Database and Knowledge-Base Systems. Volume II*. Computer Science Press.

[van Eijck, 2005] van Eijck, J. (2005). Discourse representation theory. *Encyclopedia of Language and Linguistics*, 3 :660–669.

[Venkateswaran, 1987] Venkateswaran, H. (1987). Properties that characterize LOGCFL. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 141–150. ACM.

[Weir, 1988] Weir, D. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania.

[Weir, 1992] Weir, D. J. (1992). Linear context-free rewriting systems and deterministic tree-walking transducers. In *ACL*, pages 136–143.

[Wells, 2002] Wells, J. B. (2002). The essence of principal typings. In *ICALP '02 : Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 913–925, London, UK. Springer-Verlag.

[Yoshinaka, 2006] Yoshinaka, R. (2006). Linearization of affine abstract categorial grammars. In *Proceedings of the 11th Conference on Formal Grammar*, pages 185–199, Malaga, Spain.

[Younger, 1967] Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2) :189–208.